Introduction au langage Java et à l'écosystème Spring au travers d'une étude de cas

Serge Tahé, décembre 2015

Ce site a été créé avec le convertisseur [Word ou ODT - > HTML] créé par l'IA Gemini 3 en janvier 2026.

1 Introduction

1.1 Contexte

Le PDF du document peut être trouvé |ICI|.

Ce document est à la fois un cours et un TD (Travail Dirigé) d'université. Il est destiné à des débutants. Il utilise les sources suivantes :

Références :
1. Introduction au langage Java ;
2. Exploiter une base relationnelle avec l'écosystème Spring ;

Nous noterons ces références respectivement [ref1] et [ref2]. La source [ref1] est ancienne (2002) mais suffisante pour ce document où elle n'est utilisée que pour sa présentation de la syntaxe du langage Java et de ses actions élémentaires. Le reste nécessaire à la réalisation du TD est présenté dans ce document dans des chapitres intitulés [Cours]. Ces chapitres proviennent directement de [ref2] (2015) et ont été parfois simplifiés.

L'objectif du document est d'enseigner le langage Java dans une optique professionnelle. Pour cette raison, nous nous appuyons fortement sur le framework Spring [http://spring.io/] très utilisé dans le développement JEE (Java Enterprise Edition). Logiquement, ce cours devrait être suivi par un cours JEE. C'est le cas à l'Istia (université d'Angers). JEE est la principale source d'emplois actuellement (novembre 2015) pour les jeunes développeurs Bac+5. Il y a beaucoup d'autres technologies que Spring dans le monde JEE. Spring a l'avantage d'être compréhensible et surtout d'apporter de bonnes pratiques de codage réutilisables en dehors de l'écosystème Spring. C'est ce qui explique son choix ici.

Ce TD est utilisé depuis plus de 10 ans et a évolué avec les technologies. Il est suivi (à l'IstiA) d'un TD JEE [Introduction à Java EE]. Ce dernier TD date de 2012 (on est ici en 2015) et mériterait d'être rafraîchi. Il présente l'orthodoxie JEE au travers du framework web JSF2 (Java Server Faces) et les EJB3 (Enterprise Java Bean). Mis bout à bout, ces deux TD ont permis à de nombreux étudiants de décrocher des stages JEE en ESN (Entreprises de Services Numériques) et de s'y faire embaucher dans la foulée.

On ne trouvera pas dans ce document, une présentation formelle de toutes les facettes de Java. Au fil des ans, le comportement des développeurs juniors face à un problème a beaucoup évolué. Désormais, ils utilisent quasi systématiquement Internet pour trouver des bouts de code qui mis bout à bout font un programme. Si on leur fournit un cours, ils l'utilisent assez peu et préfèrent de nouveau aller sur Internet. Dubitatif au départ sur cette façon de travailler, j'ai quand même été étonné des résultats obtenus. Ainsi des étudiants faibles réussissaient à produire des programmes qui marchaient alors que sans l'aide d'Internet ils n'y seraient probablement pas arrivés. Je m'appuie désormais sur cette façon de travailler.

Des bouts de code ne donnent pas une vue d'ensemble de l'architecture d'une solution et c'est l'un des objectifs de ce document que de donner celle-ci. Les étudiants font ce TD comme un TP, en autonomie. Il n'y a pas de cours magistral. Il y a un planning qui leur donne l'état d'avancement attendu d'eux au fil des séances. Ils peuvent être en retard ou en avance sur ce planning. Leur avancement est vérifié par un certain nombre de validations qu'ils doivent présenter à l'enseignant. Celui-ci est présent à la fois pour leur fournir des explications lorsqu'ils en demandent et valider leur travail. Chacun va à son rythme. A la fin des 36 h alloués à ce TD, certains auront fait 50% de validations en plus que d'autres mais chacun, c'est en tout cas l'objectif, aura compris ce qu'il a fait en autonomie. Ce TD peut être fait sans l'accompagnement d'un enseignant. C'est pourquoi, il est disponible sur [https://tahe.developpez.com].

Ce document ne conviendra pas à ceux qui chercheraient un cours académique sur Java, quelque chose où on explique Java de façon progressive et structurée et où chaque détail de syntaxe est expliqué et justifié. C'est plutôt une démarche expérimentale qui est proposée ici. Il est probable que l'étudiant ne comprendra pas tout ce qui lui est proposé dans le document mais il saura probablement réutiliser son contenu à bon escient et la compréhension des détails viendra avec l'expérience.

Ce document n'est pas non plus un cours d'algorithmique. L'algorithme du TD est basique et peut être résolu par tout débutant suivant ses premiers cours d'algorithmique. Le document est centré sur l'environnement de développement professionnel en Java avec ses nombreuses bibliothèques ou frameworks et sur l'architecture du code. La plupart des étudiants que je vois passer présentent des faiblesses en algorithmique qui sont confirmées ensuite par les maîtres de stage. Donc oui, la maîtrise des algorithmes est importante mais ce n'est pas l'objet de ce cours-TD.

Enfin, ce document (décembre 2015) ne présente pas les dernières nouveautés de Java, notamment les streams et les fonctions lambda. Néanmoins, on y utilise quelques éléments du dernier JDK, le JDK 1.8 et les codes qui suivent doivent être compilés par ce JDK.

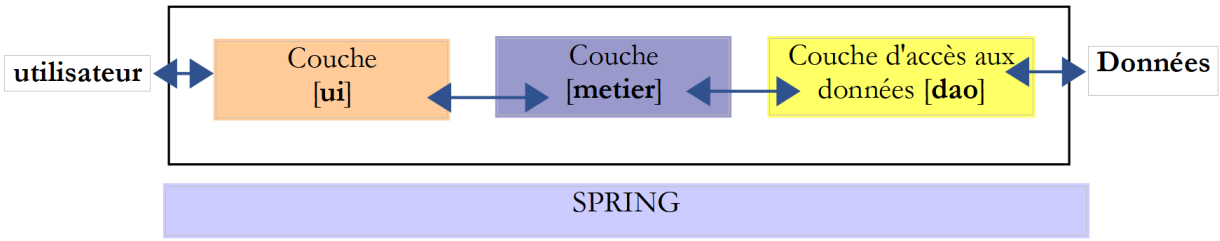
1.2 Contenu

Le chapitre 2 présente le sujet du TD, un calcul de résultats d'élections. Le problème est basique. Le chapitre 2 demande d'implémenter la solution avec deux langages C# et Java qui sont très proches. L'implémentation se fait sans classes. L'objectif est la présentation de la syntaxe de Java, de ses instructions élémentaires, de l'IDE (Integrated Development Environment) Eclipse qui sert à construire les projets Java.

Le chapitre 3 demande d'implémenter la solution du TD en Java avec des classes. L'objectif est de présenter les notions de classes, d'héritage, d'interfaces et de classes génériques. La notion de test unitaire JUnit est introduite.

Le chapitre 4 introduit les concepts qui sous-tendent les chapitres suivants :

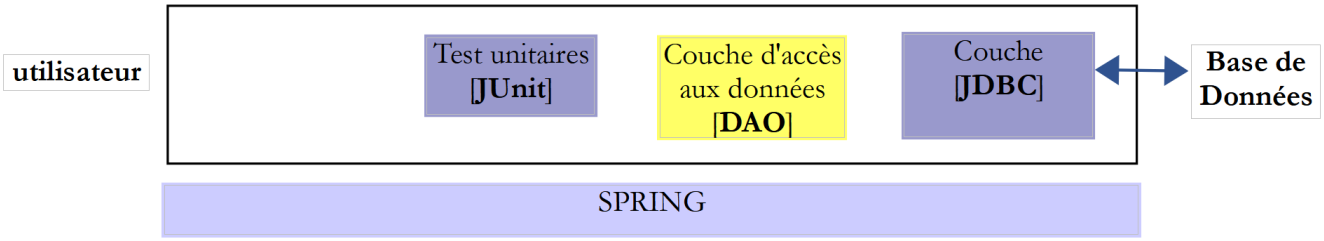
- les architectures en couches ;
- la programmation par interfaces ;
- l'utilisation de Spring pour implémenter les deux précédents concepts ;



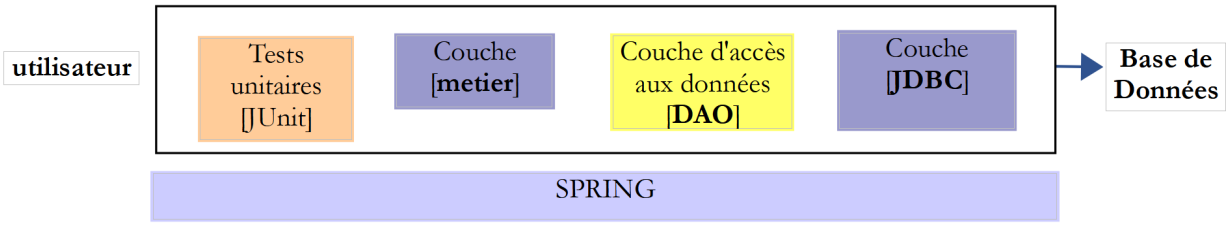
Le chapitre 5 présente le framework Spring avec quatre projets.

Le chapitre 6 présente l'API JDBC qui est une interface d'accès aux bases de données.

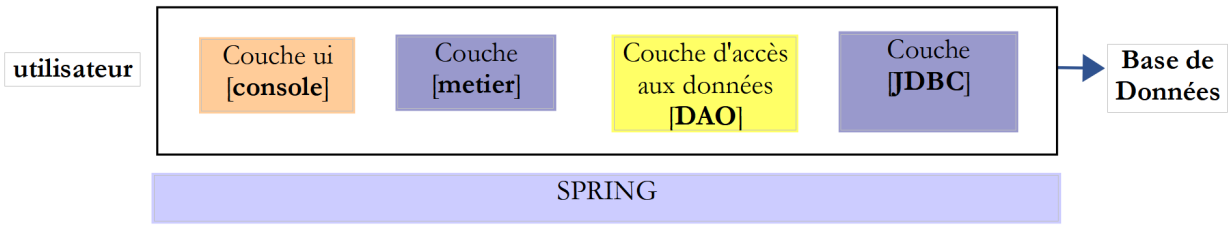
Le chapitre 7 implémente la couche [DAO] (Data Access Object) du TD avec l'API JDBC et Spring.



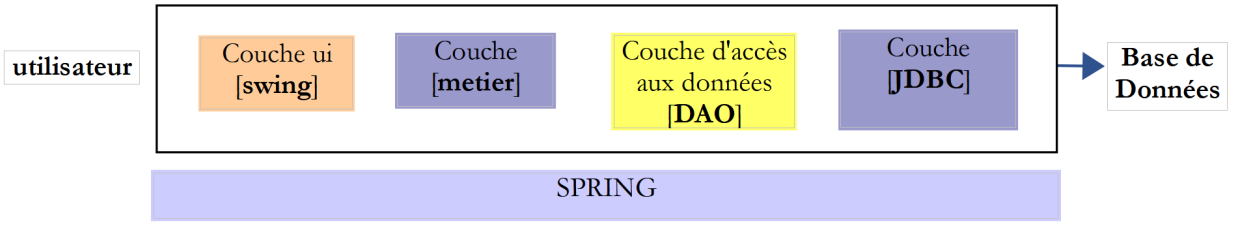
Le chapitre 8 implémente la couche [métier] du TD :

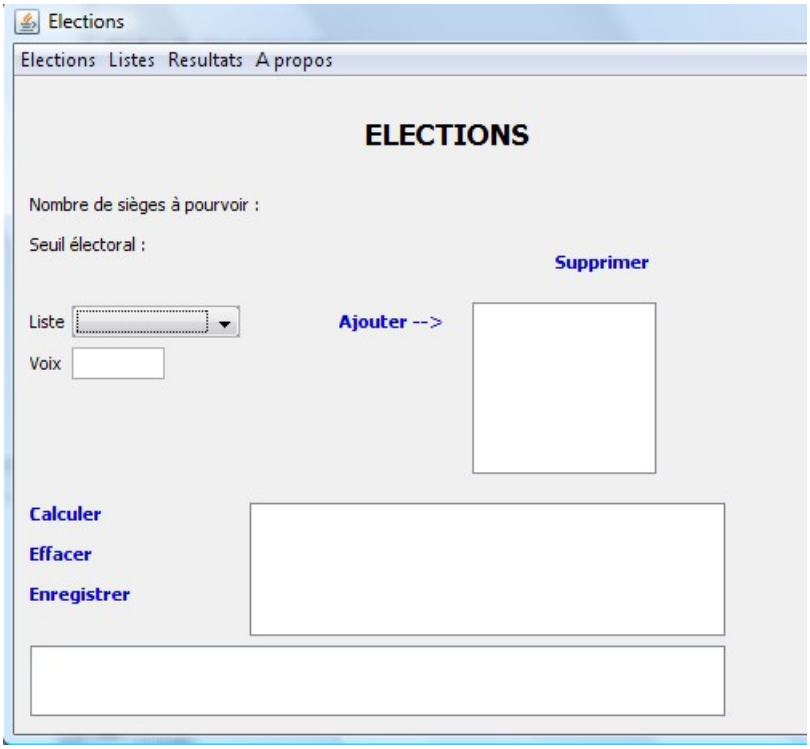


Le chapitre 9 implémente la couche [ui] du TD avec une application console :

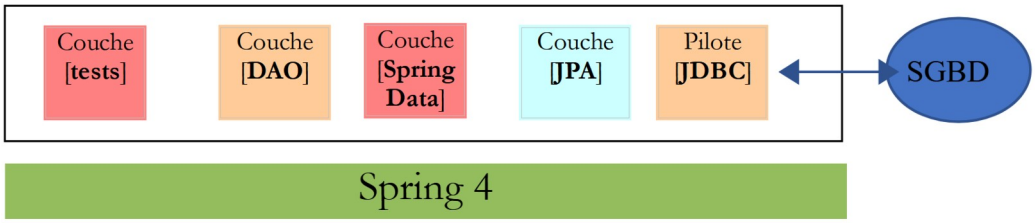


Le chapitre 10 implémente la couche [ui] du TD avec une application graphique utilisant la bibliothèque de composants Swing :



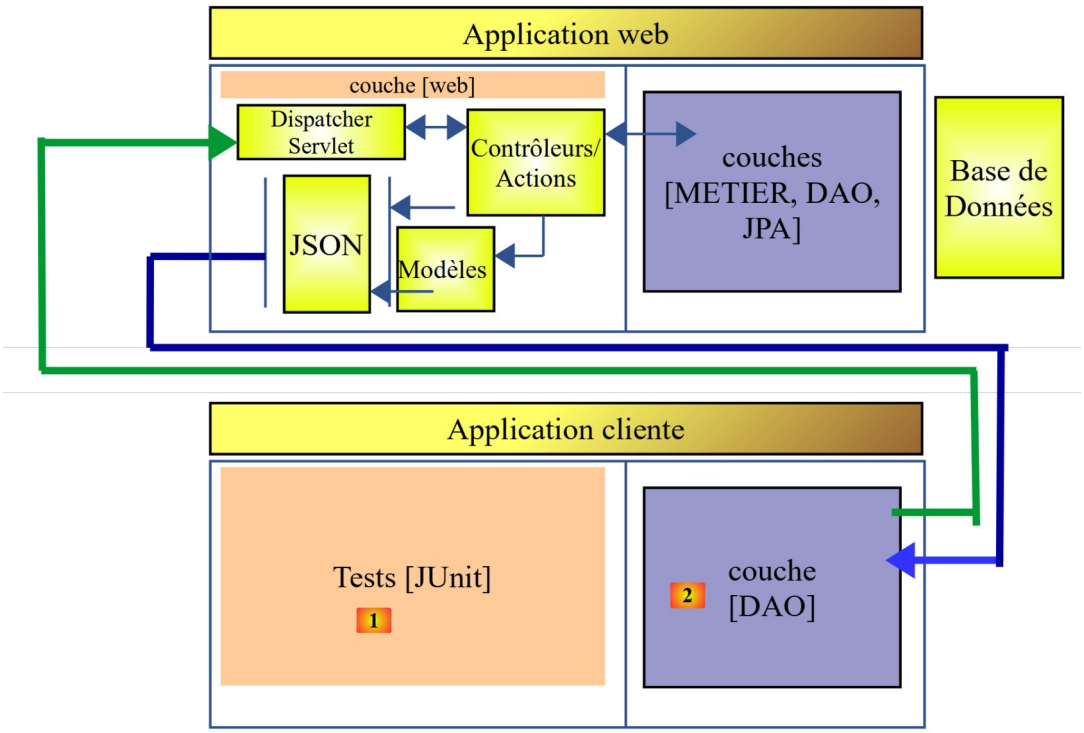


Le chapitre 11 présente la gestion des bases de données avec le framework [Spring Data], une branche de l'écosystème Spring. Elle introduit la spécification JPA (Java Persistence API) qui permet à la couche [DAO] de manipuler des objets au lieu de manipuler du SQL (Structured Query Language). L'architecture en couches évolue de la façon suivante :

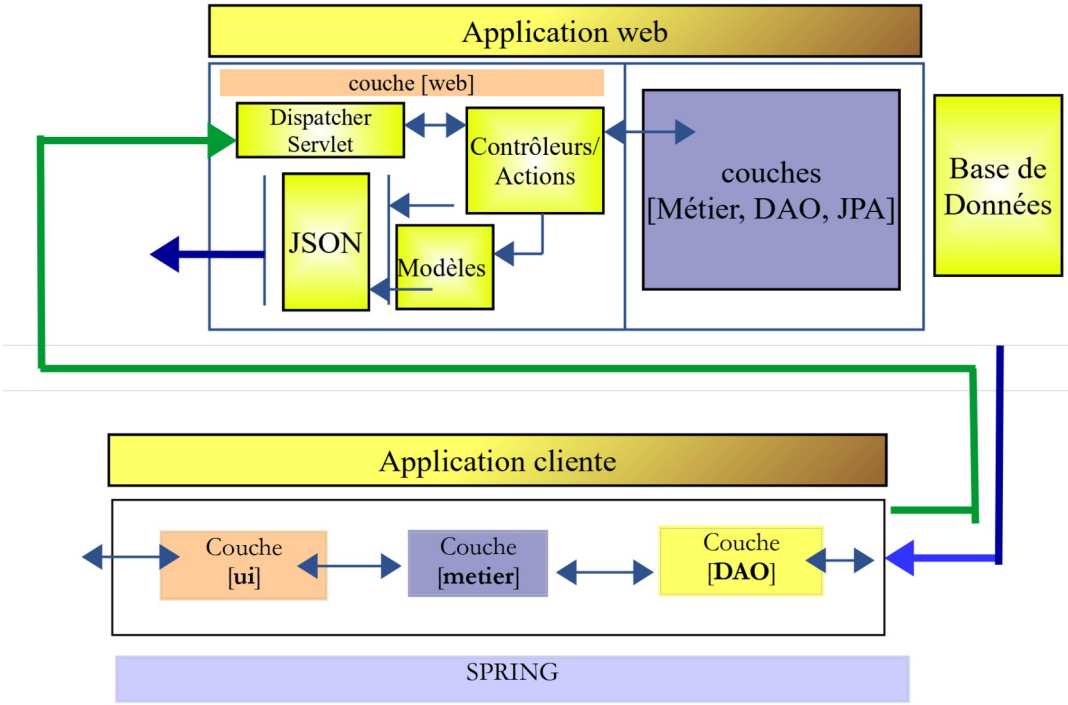


Le chapitre 12 applique le chapitre 11 en implémentant l'accès à la base de données du TD avec [Spring Data].

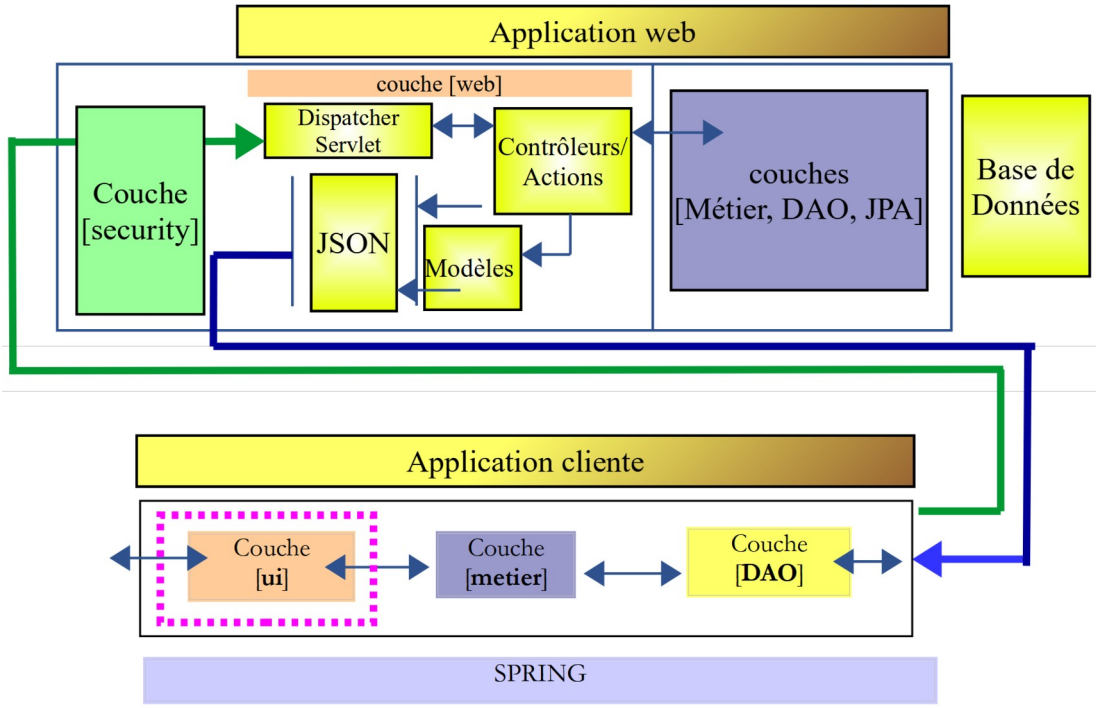
Le chapitre 13 montre comment exposer une base de données sur le web avec [Spring MVC] qui est une autre branche de l'écosystème Spring. L'architecture évolue en architecture client / serveur :



Les chapitres 14 et 15 transforment l'application du TD en application client / serveur :

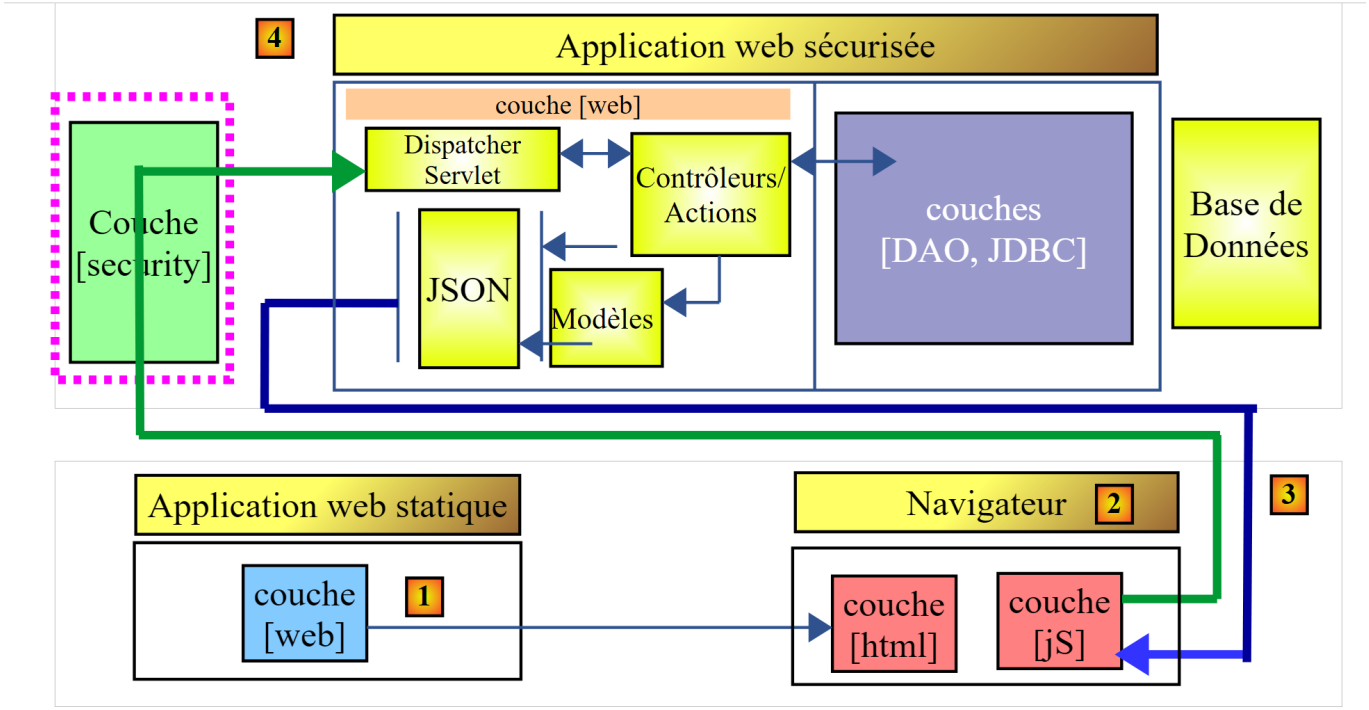


Le chapitre 16 montre comment sécuriser l'accès à une application web avec [Spring Security] une autre branche de l'écosystème Spring.



Le chapitre 17 reprend le TD et sécurise le service web des élections.

Le chapitre 18 aborde le problème des requêtes inter-domaines :



- en [1], une application web délivre des pages HTML / Javascript ;
- en [2], le navigateur exécute le Javascript embarqué dans les pages HTML pour interroger le service web sécurisé [3-4] ;

Soit D1=http://machine1:port1 le domaine du serveur [1] et D4=http://machine4:port4 le domaine du serveur [4]. Si les serveurs [1] et [4] ne sont pas dans le même domaine [D1!=D4], alors les requêtes de [3] vers [4] sont appelées des requêtes inter-domaines. A cause de restrictions de sécurité mises en oeuvre par les navigateurs, leur mise en place peut être problématique. Nous examinerons une solution.

Le chapitre 19 met en oeuvre les requêtes inter-domaines avec l'application des élections.

1.3 Les outils utilisés

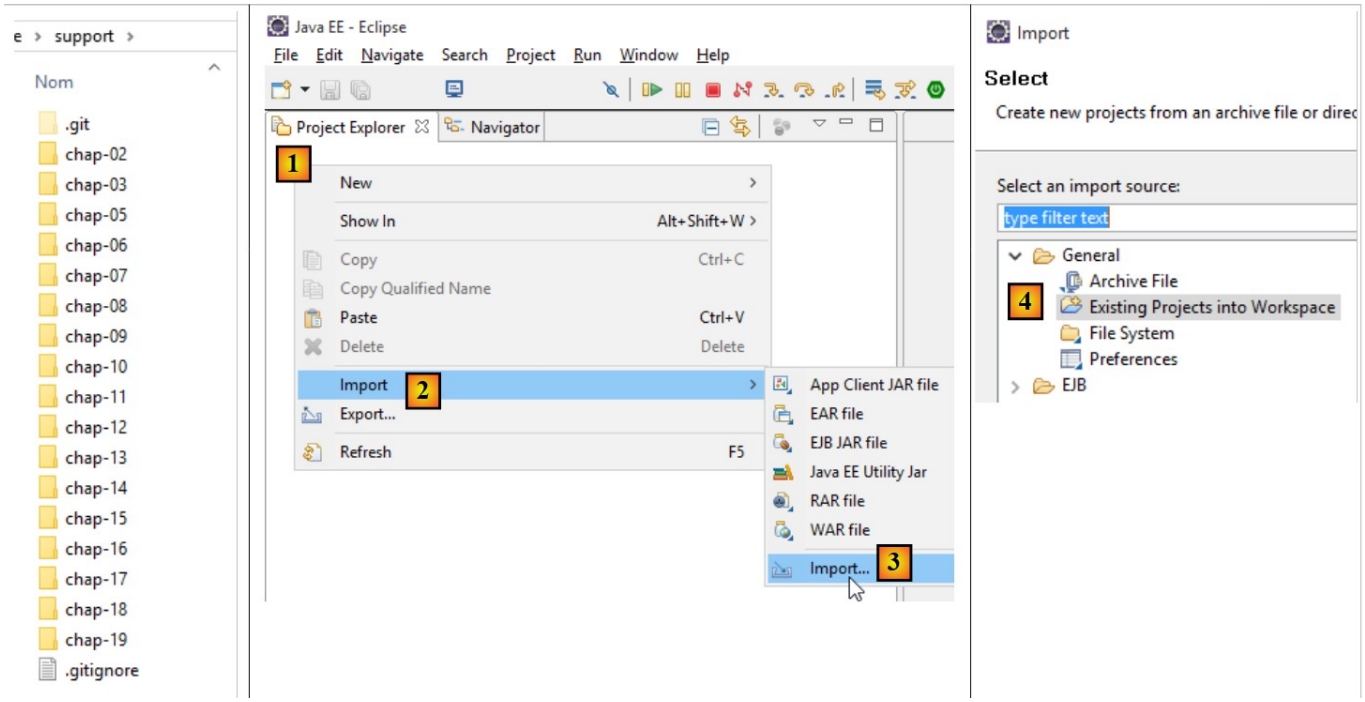
Les exemples qui suivent ont été testés dans l'environnement suivant :

- machine Windows 10 pro 64 bits ;
- JDK 1.8 (page 406) ;
- IDE Spring Tool Suite 3.6.3 (page 407) ;
- Netbeans 8.1 (page 418) ;
- navigateur Chrome (les autres navigateurs n'ont pas été utilisés) ;
- extension Chrome [Advanced Rest Client] (page 419) ;
- WampServer qui amène le SGBD MySQL et l'outil [PhpMyAdmin] pour le gérer (page 421) ;

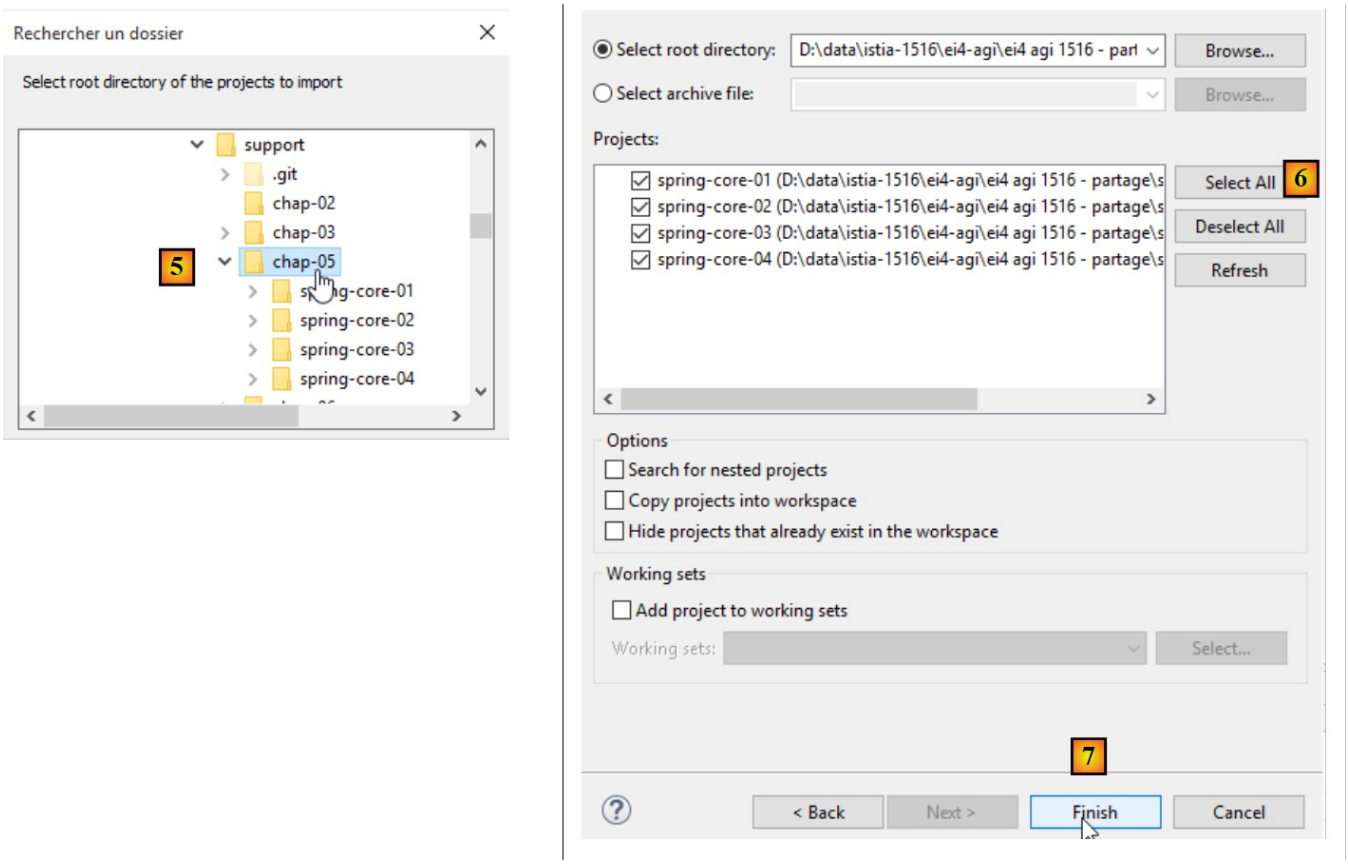
Il est important d'utiliser un JDK 1.8. Certains exemples utilisent des éléments de ce JDK. La plupart des exemples sont des projets Maven qui peuvent être ouverts indifféremment par les IDE Eclipse [https://www.eclipse.org/], IntellijIDEA Community Edition [https://www.jetbrains.com/idea/download/] et Netbeans [https://netbeans.org/]. Dans la suite, les copies d'écran proviennent de l'IDE Spring Tool Suite, une variante d'Eclipse.

1.4 Le support

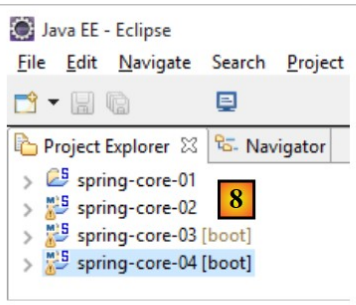
Les projets Eclipse de ce document sont disponibles sur le site [https://tahe.developpez.com/tutoriels-cours/intro-java-spring/serge-tahe-introduction-au-langage-java-et-a-l-ecosysteme-spring/].

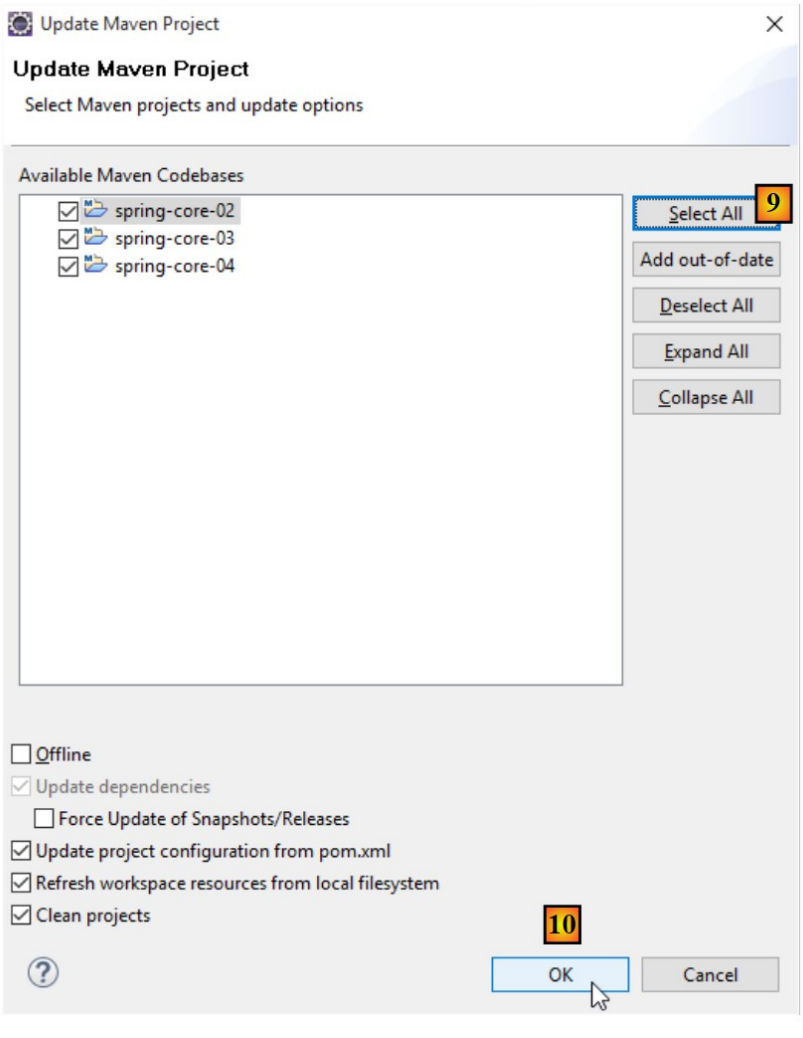


Pour importer les projets d'un chapitre, procédez avec Eclipse comme indiqué en [1-8] :



La plupart des projets sont des projets Maven. Si après chargement, ceux-ci présentent des erreurs, faites [Alt-F5] et suivez la procédure [9-10]. Les projets Maven sélectionnés vont être reconstruits.

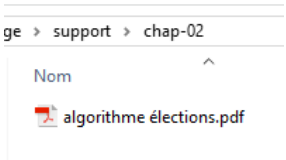

Java EE - Eclipse
File Edit Navigate Search Project
Project Explorer
Navigator
spring-core-01
spring-core-02
spring-core-03 [boot]
spring-core-04 [boot]
8


Update Maven Project
Update Maven Project
Select Maven projects and update options
Available Maven Codebases
spring-core-02
spring-core-03
spring-core-04
Select All
9
Add out-of-date
Deselect All
Expand All
Collapse All
Offline
Update dependencies
Force Update of Snapshots/Releases
Update project configuration from pom.xml
Refresh workspace resources from local filesystem
Clean projects
10
OK
Cancel

2 [TD] : Le problème

Mots clés : algorithmique, bases de Java : tableaux, E/S, boucles, tests, gestion d'exceptions
Lectures conseillées : chapitre 1 de [ref1] : Les bases du langage Java

2.1 Support



Le dossier [support / chap-02] contient l'algorithme à traduire en C# et Java.

2.2 Le problème à résoudre

On désire écrire un programme qui, au soir d'élections, puisse calculer le nombre de sièges obtenus par les différentes listes en présence. On trouvera un peu plus loin, le mode de calcul des sièges pour une élection proportionnelle à la plus forte moyenne, tel qu'expliqué dans un article du journal Ouest-France du 15 mars 1986.

On écrira une application Java "console", c.a.d. une application utilisant le **clavier** et **l'écran** pour communiquer avec l'utilisateur. Elle demandera les renseignements suivants à l'utilisateur (tapés au **clavier**) :

- nombre de sièges à pourvoir
- nombre de listes en compétition
- pour chaque liste : son nom, son nombre de voix

Avec ces renseignements, l'application calcule les sièges obtenus par chacune des listes et les affiche à **l'écran** sous la forme suivante :

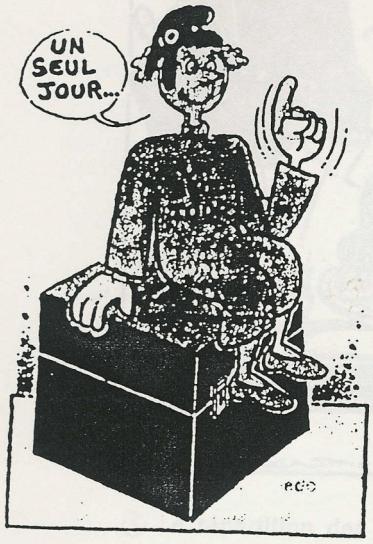
- La liste [X1] a obtenu [N1] sièges
- La liste [X2] a obtenu [N2] sièges
- ...

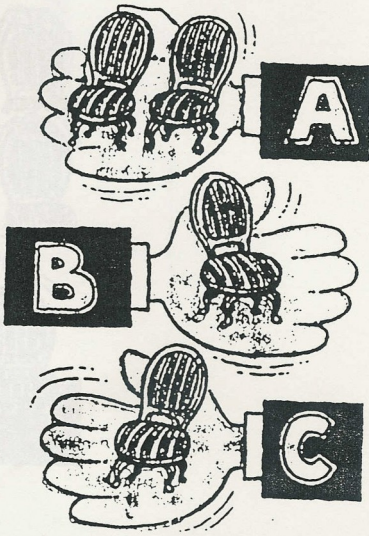
où [Xi] est le nom de la liste n° i et [Ni] le nombre de sièges qu'elle a obtenus.

L'article Ouest-France du 15 mars 1986 :

La proportionnelle, mode d'emploi

2





PRENONS UN DÉPARTEMENT où six sièges sont à pourvoir, sept listes en présence et où 100 000 suffrages ont été exprimés. Voici la règle du jeu, valable aussi bien pour les législatives que pour les régionales.

6F 15/3/84

① La barre des 5 %

Ont obtenu, liste A, 32 000 voix (32 %) ; liste B, 25 000 (25 %) ; liste C, 16 000 (16 %) ; liste D, 12 000 (12 %) ; liste E, 8 000 (8 %) ; liste F, 4 500 (4,5 %) ; liste G, 2 500 (2,5 %).

Les deux dernières, n'ayant pas atteint la barre des 5 %, sont éliminées.

Première opération : calculer les **suffrages exprimés utiles**, en additionnant les voix des listes ayant dépassé les 5 %, soit : A + B + C + D + E = 93 000

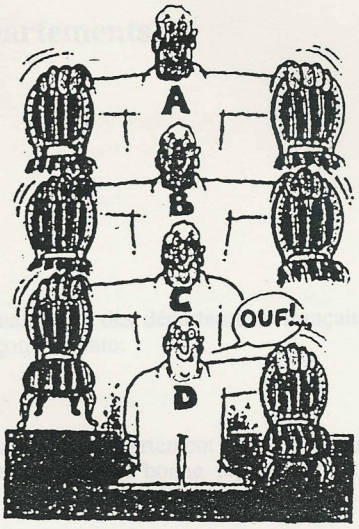
② Le calcul du quotient électoral

On calcule le quotient électoral en divisant le nombre de suffrages exprimés utiles par le nombre de sièges à pourvoir : 93 000 : 6 = 15 500. On attribue ensuite à chaque liste autant de sièges que son nombre de voix contient de fois le quotient. Ainsi, la liste A, ayant obtenu plus de deux fois le quotient, aura deux sièges.

Liste A : 2 sièges.
Liste B : 1 siège.
Liste C : 1 siège.

Au total, quatre sièges sont attribués « au quotient ».

https://tahe.developpez.com - Ce cours tutoriel écrit par Serge Tahé est mis à disposition du public selon les termes de la Licence Creative Commons Attribution – Pas d'Utilisation Commerciale – Partage dans les Mêmes Conditions 3.0 non transposé.



③ La répartition des restes

Restent, dans notre exemple, à répartir deux sièges à la plus forte moyenne. On attribue d'abord fictivement un siège supplémentaire à chacune des listes. On divise ensuite le nombre de suffrages recueillis par chaque liste par le nombre de sièges déjà attribués plus un. Celle qui a le plus fort résultat obtient un siège.

Liste A, 32 000 : 3 (2 + 1) = 10 666.
Liste B, 25 000 : 2 (1 + 1) = 12 500.
Liste C, 16 000 : 2 (1 + 1) = 8 000.
Liste D, 12 000 : 1 (0 + 1) = 12 000.
Liste E, 8 000 : 1 (0 + 1) = 8 000.

Le premier siège restant va à la liste B, qui a la plus forte moyenne. On recommence l'opération pour l'attribution du dernier siège, qui ira à la liste D.

④ Le classement final

Les deux sièges restants sont donc attribués aux listes B et D. A l'arrivée, les six sièges à pourvoir sont ainsi répartis :

Liste A : 2 sièges.
Liste B : 2 sièges.
Liste C : 1 siège.
Liste D : 1 siège.

OF 15/3/96

c.o.d liste A, 32000 : 3 (2+1) = 10666
" B, 25000 : 3 (2+1) = 8333 ← liste B a maintenant un siège de plus
" C, 16000 : 2 (1+1) = 8000
" D, 12000 : 1 (0+1) = 12000
" E, 8000 : 1 (0+1) = 8000

le siège va à la liste D

2.3 La solution algorithmique

Une solution algorithmique pourrait être la suivante :

```
1. début-programme
2.   // données
3.   saisieOK : booléen
4.   nbSiègesAPourvoir : entier
5.   nbListes : entier
6.   nomListe[] : chaînes de caractères
7.   voixListe[] : entier
8.   elimineListe[] : booléen
9.   siegesListe[] : entier
10.    moyenneListe[] : réel
11.    i : entier
12.    nbVoixUtiles : entier
13.    quotientElectoral : réel
14.    nbSiègesPourvus : entier
15.    moyenneMax : réel
16.    Max : entier iSiège : entier
```

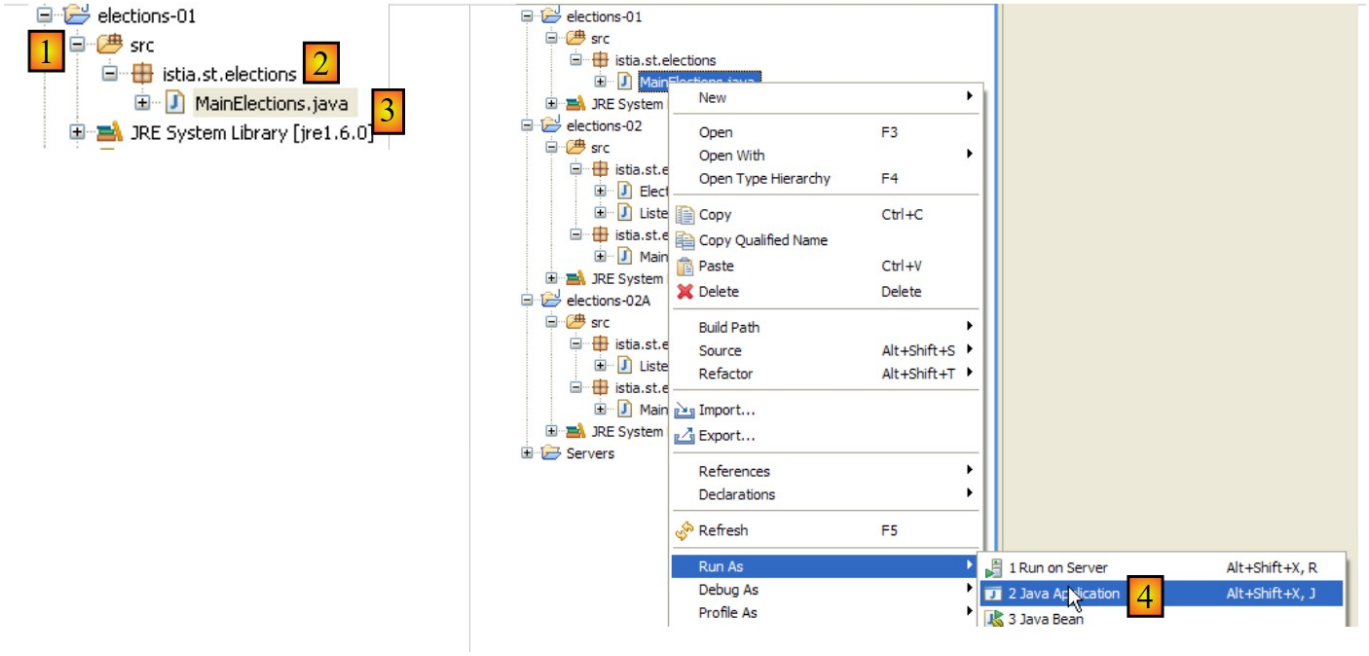
```
17.
18.   // code
19.   // nbre de sièges à pourvoir
20.   saisieOK<-faux
21.   tant que non saisieOK
22.       écrire "Nombre de sièges à pourvoir : "
23.       lire nbSiègesAPourvoir
24.       si nbSiègesAPourvoir n'est pas un entier >0 alors
25.           écrire "Erreur : tapez un nombre entier >0"
26.       sinon
27.           saisieOK<-vrai
28.       finsi
29.   fintantque
30.   // nbre de listes en compétition
31.   saisieOK<-faux
32.   tant que non saisieOK
33.       écrire "Nombre de listes en compétition : "
34.       lire nbListes
35.       si nbListes n'est pas un entier >0 alors
36.           écrire "Erreur : tapez un nombre entier >0"
37.       sinon
38.           saisieOK<-vrai
39.       finsi
40.   fintantque
41.
42.   // dimensionnement des tableaux
43.   dimensionner les tableau nomListe, voixListe, elimineListe, siegesListe, moyenneListe à nbListes éléments
44.
45.   // saisie des noms et voix des listes
46.   totalVoix<-0
47.   pour i variant de 0 à nbListes-1
48.       // saisie du nom de la liste i
49.       saisieOK<-faux
50.       tantque non saisieOK
51.       écrire "Nom de la liste n° ", i, " : "
52.       lire nomListe[i]
53.       si nomListe[i] est vide alors
54.           écrire "Erreur : Tapez un nom non vide"
55.       sinon
56.           saisieOK<-vrai
57.       finsi
58.   fintantque
59.   // saisie du nombre de voix de la liste i
60.   saisieOK<-faux
61.   tantque non saisieOK
62.       écrire "Nombre de voix de la liste ", nomListe[i] , " : "
63.       lire voixListe[i]
64.       si voixListe[i] n'est pas un nombre entier >=0 alors
65.           écrire "Erreur : tapez un nombre entier >=0"
66.       sinon
67.           saisieOK<-vrai
68.       finsi
69.   fintantque
70.
71.   // on incrémente le total des voix
72.   totalVoix<- totalVoix+voixListe[i]79.finpour 80.
73.   // calcul des voix utiles
74.   nbVoixUtiles<-0
75.   pour i variant de 0 à nbListes-1
76.       si (voixListe[i]/totalVoix)<0.05 alors
77.           elimineListe[i]<-vrai
78.       sinon
79.           elimineListe[i]<-faux
80.           nbVoixUtiles<-nbVoixUtiles+voixListe[i]
81.       finsi
82.   finpour
83.   // y-a-t-il des listes non éliminées ?
84.   si nbVoixUtiles=0 alors
85.       écrire "Erreur : toutes les listes ont été éliminées"
86.       arrêt du programme
87.   finsi
88.   // répartition des sièges au quotient
89.   quotientElectoral <- nbVoixUtiles / nbSiègesAPourvoir
90.   nbSiègesPourvus<- 0
91.   pour i variant de 0 à nbListes-1
92.       si non elimineListe[i] alors
93.           siegesListe[i]<- partie entière de (voixListe[i]/quotientElectoral)
94.           moyenneListe[i] <- voixListe[i] / (siegesListe[i]+1)
95.           nbSiègesPourvus<-nbSiègesPourvus+siegesListe[i]
96.       sinon
97.           siegesListe[i]<-0
98.       finsi
99.   finpour
100.  // répartition des sièges restants à la plus forte moyenne
101.  // 1 siège est attribué à chaque tour de boucle
102.  pour iSiège variant de 0 à nbSiègesAPourvoir - nbSiègesPourvus - 1
103.      // recherche de la liste ayant la + forte moyenne
```

```
104.      moyenneMax<- (-1)
105.      pour i variant de 0 à nbListes-1
106.          si non elimineListe[i] alors
107.              si moyenneListe[i] > moyenneMax alors
108.                  moyenneMax <- moyenneListe[i]
109.                  iMax <- i
110.              finsi
111.          finsi
112.      finpour
113.      // on attribue 1 siège à la liste de + forte moyenne
114.      siegeListe[iMax] <- siegeListe[iMax]+1
115.      // et on change sa moyenne
116.      moyenneListe[iMax] <- voixListe[iMax]/(siegeListe[iMax]+1)
117.  finpour
118.  // affichage résultats sans tri
119.  pour i variant de 0 à nbListes-1
120.      si elimineListe[i] alors
121.          écrire "La liste ", nomListe[i], " a été éliminée"
122.      sinon
123.          écrire "La liste ", nomListe[i], " a obtenu ",
124.          siegesListe[i], " siège(s)"
125.      finsi
126.  finpour
127. fin-programme
```

2.4 Travail à faire

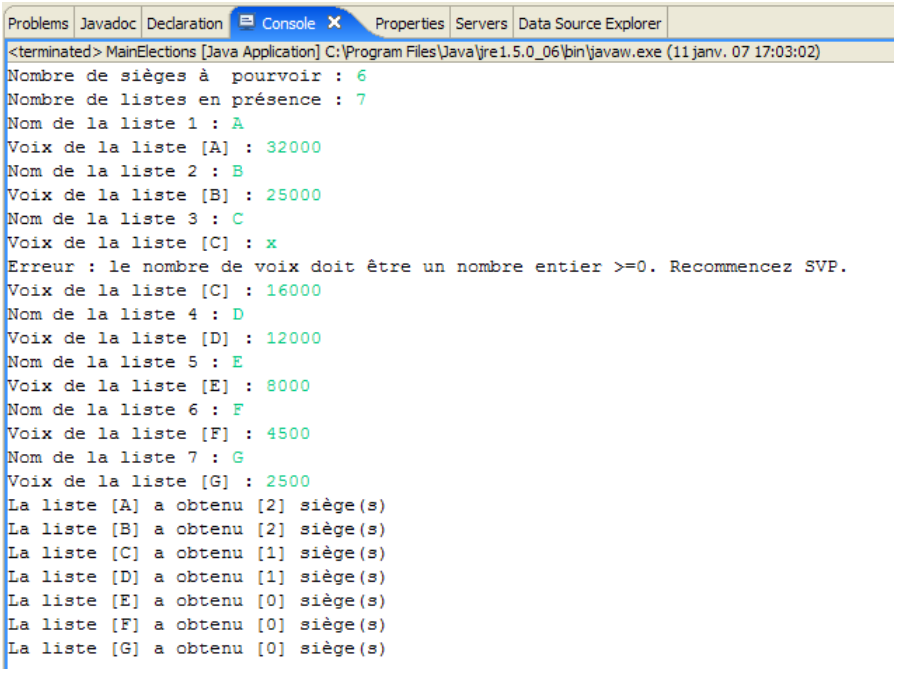
Q1 : traduire l'algorithme en C#. Le mettre en oeuvre avec Visual Studio.

Q2 : traduire l'algorithme en Java en vous inspirant du code C#. Mettre en oeuvre le programme Java dans un environnement Eclipse analogue au suivant :



- [1] : le projet s'appelle [elections-01]
- [2] : l'application sera placée dans un paquetage, ici [istia.st.elections]
- [3] : [MainElections.java] est le code source de l'application écrite dans la partie précédente
- [4] : la classe [MainElections] est exécutée

Un exemple d'exécution pourrait être le suivant :


Problems Javadoc Declaration Console Properties Servers Data Source Explorer
<terminated> MainElections [Java Application] C:\Program Files\Java\jre1.5.0_06\bin\javaw.exe (11 janv. 07 17:03:02)
Nombre de sièges à pourvoir : 6
Nombre de listes en présence : 7
Nom de la liste 1 : A
Voix de la liste [A] : 32000
Nom de la liste 2 : B
Voix de la liste [B] : 25000
Nom de la liste 3 : C
Voix de la liste [C] : x
Erreur : le nombre de voix doit être un nombre entier >=0. Recommencez SVP.
Voix de la liste [C] : 16000
Nom de la liste 4 : D
Voix de la liste [D] : 12000
Nom de la liste 5 : E
Voix de la liste [E] : 8000
Nom de la liste 6 : F
Voix de la liste [F] : 4500
Nom de la liste 7 : G
Voix de la liste [G] : 2500
La liste [A] a obtenu [2] siège(s)
La liste [B] a obtenu [2] siège(s)
La liste [C] a obtenu [1] siège(s)
La liste [D] a obtenu [1] siège(s)
La liste [E] a obtenu [0] siège(s)
La liste [F] a obtenu [0] siège(s)
La liste [G] a obtenu [0] siège(s)

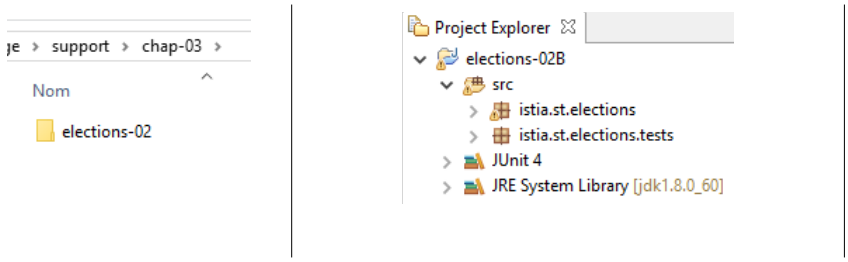
3 [TD] : Classes

Mots clés : classe, interface, héritage, exception, polymorphisme
Lectures conseillées :

- ✗ paragraphes 2.1, 2.2, 2.4 et 2.7 du chapitre 2 de [ref1] : Classes et interfaces
- ✗ paragraphes 3.3 (classe String), 3.5 (classe ArrayList), 3.6 (classe Arrays)

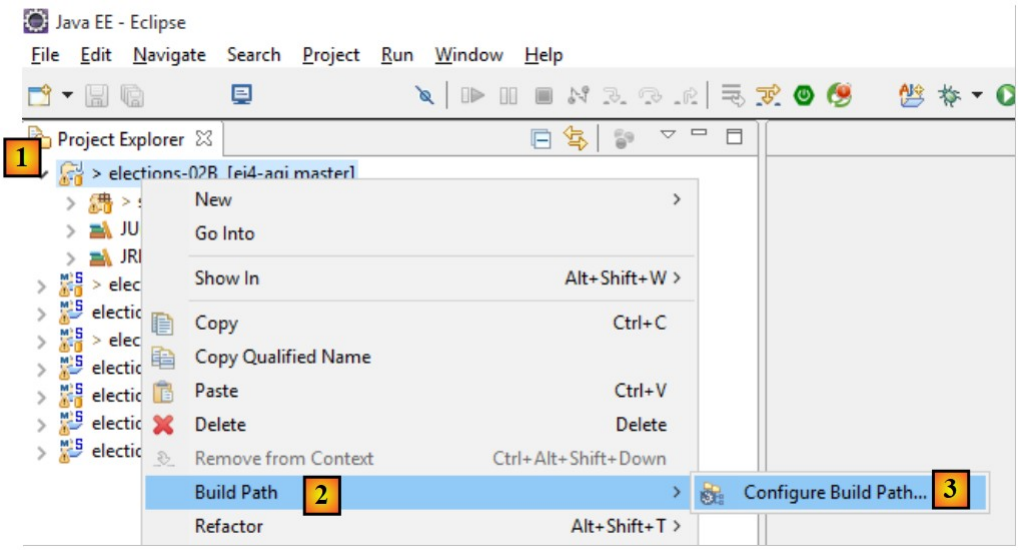
Dans la partie 1 de l'exercice ELECTIONS aucune classe n'a été utilisée. On a construit une solution comme on l'aurait construite en langage C. Nous introduisons maintenant la notion de **classe Java**.

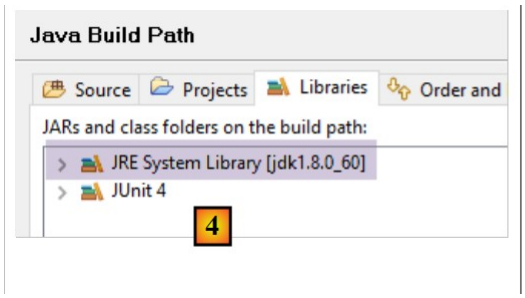
3.1 Support

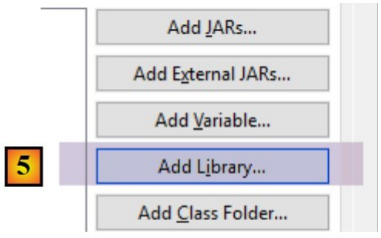


Le dossier [support / chap-03] contient le projet Eclipse de ce chapitre.

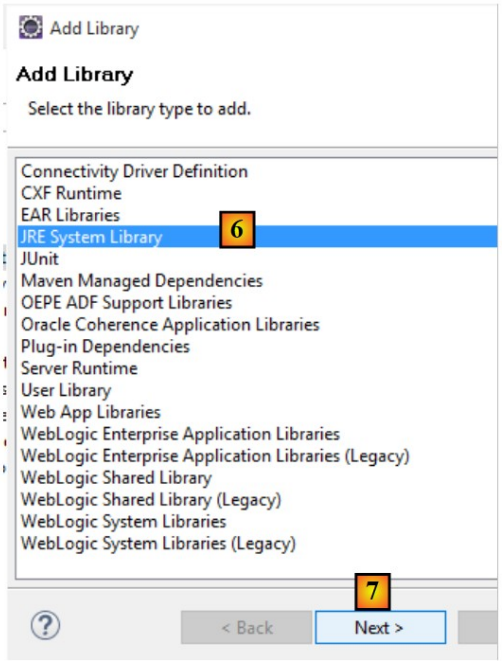
On travaillera désormais avec le JDK 1.8 car certains des projets à suivre nécessitent ce JDK. Pour connaître le JDK utilisé, procédez de la façon suivante :

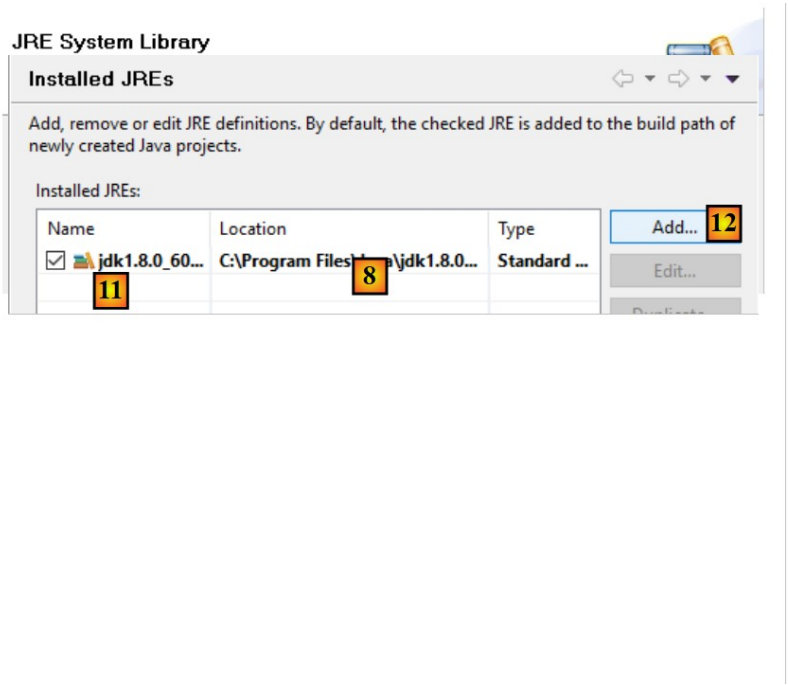




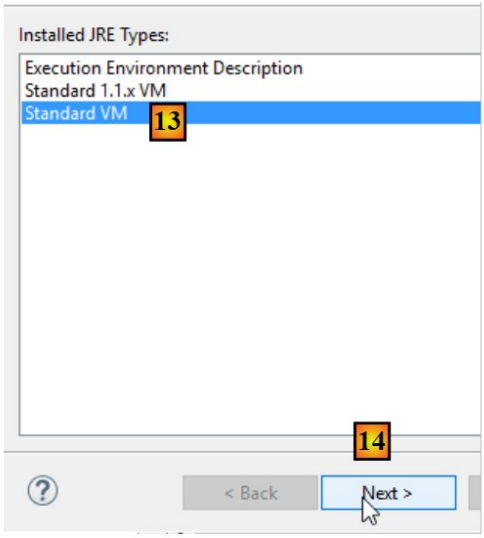


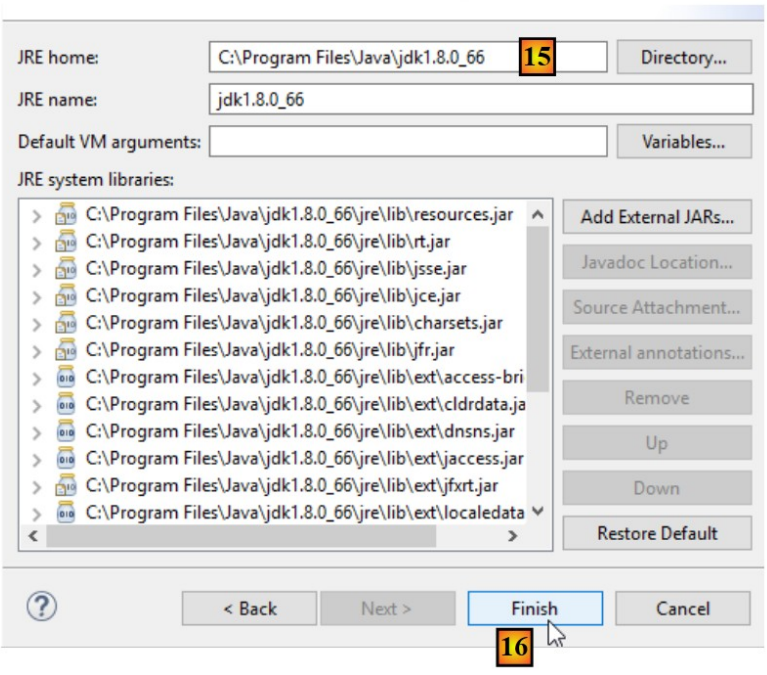
- en [4], le JRE (Java Runtime Environment) utilisé. Ce JRE est en fait ici un JDK (Java Development Kit), ici [jdk1.8.0_60]. Si ce n'est pas un JDK ou si vous avez une version inférieure à 1.8, procédez comme suit [5-21] ;



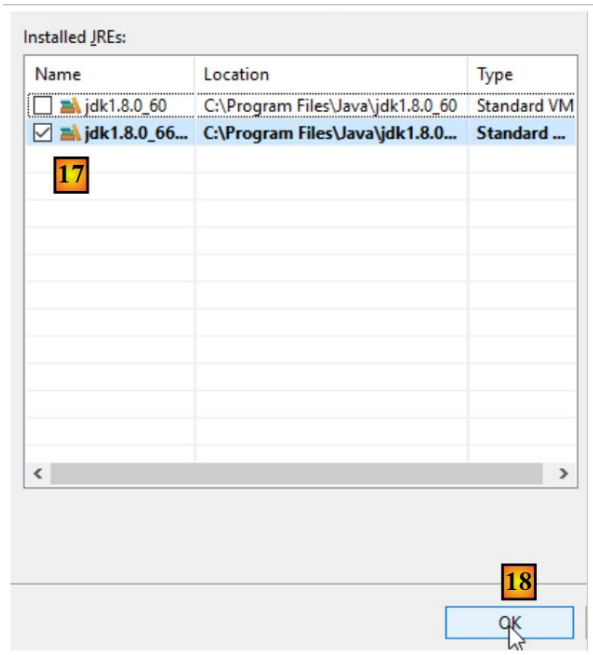


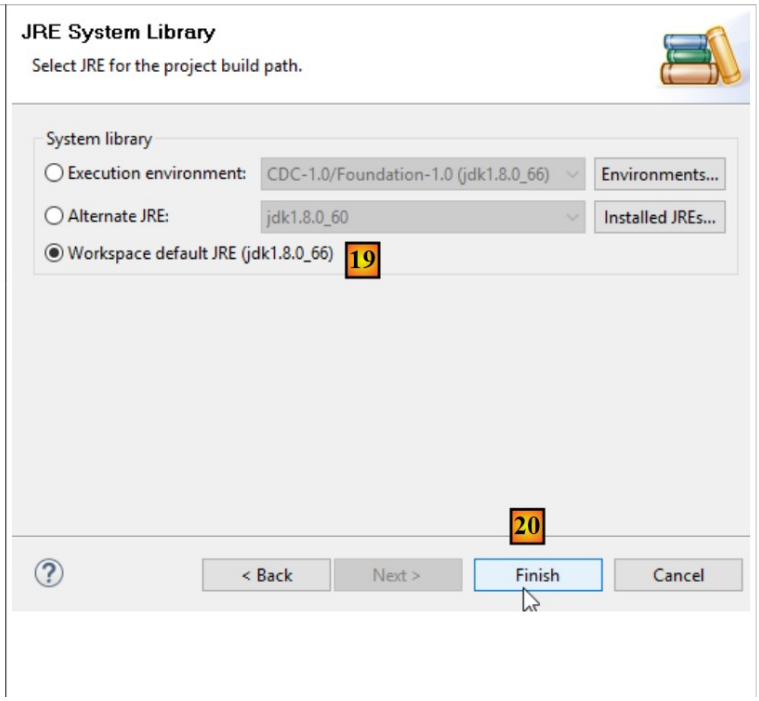
- en [8], le JRE utilisé actuellement par défaut par Eclipse ;
- en [11], les différents JDK et JRE actuellement connus par Eclipse ;

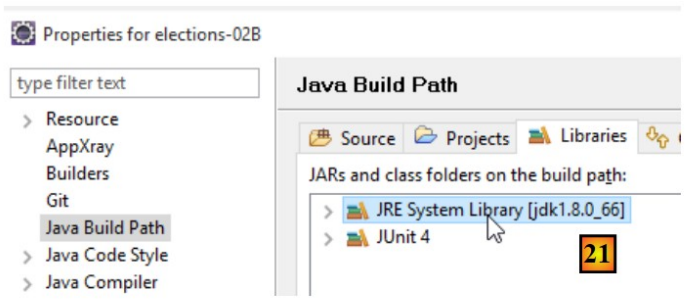


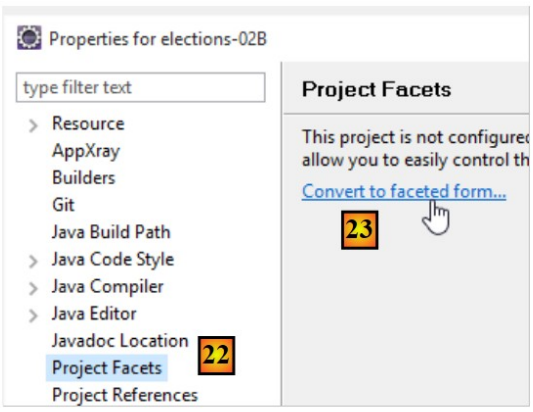


- en [15], **choisissez un JDK plutôt qu'un JRE**. Ce document utilise des projets Maven qui ont besoin d'un JDK ;

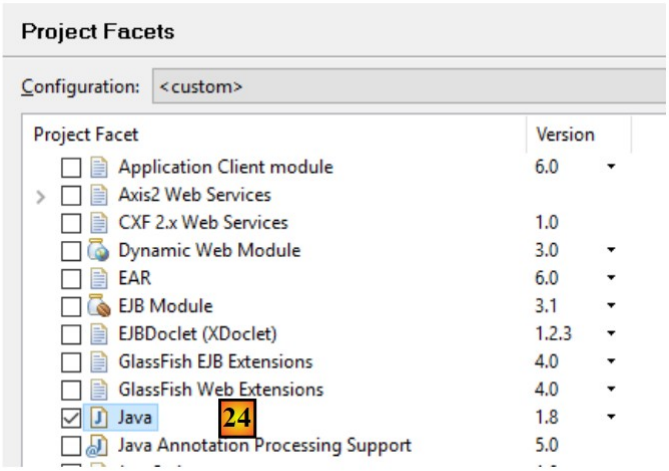








- en [21], on a un JDK de version >=1.8 ;
- en [22-23], accédez aux facets (différentes vues d'un même projet Eclipse) du projet ;



- en [24], vérifiez que vous utilisez une version Java >=1.8 ;

3.2 La classe [ListeElectorale]

En langage C, nous aurions probablement utilisé une **structure** pour représenter une liste participant à l'élection. Elle aurait pu être de la forme suivante :

```
struct t_liste
   {
     char nom[15];
     long voix;
     int  elimine;
     int sieges;
   };
```

La notion de structure n'existe pas dans le langage Java. Il faut la remplacer par celle de **classe**. On décide donc de créer une classe pour mémoriser les informations sur une liste candidate. Celle-ci aurait le squelette suivant :

```
1.  package istia.st.elections;
2.
3.  public class ListeElectorale {
4.
5.      /**
6.       * identité de la liste
7.       */
8.      private int id;
9.
10.     /**
11.      * nom de la liste
12.      */
13.     private String nom;
14.     /**
15.      * nombre de voix de la liste
16.      */
17.     private int voix;
18.     /**
19.      * nombre de sièges de la liste
20.      */
21.     private int sieges;
22.     /**
23.      * indique si la liste est éliminée ou non
24.      */
25.     private boolean elimine;
26.
27.     /**
28.      * constructeur par défaut
29.      */
30.     public ListeElectorale() {
31.     }
32.
33.     /**
34.      *
35.      * @param nom String : le nom de la liste
36.      * @param voix int : son nombre de voix
37.      * @param sieges int : son nombre de sieges
38.      * @param elimine boolean : son état éliminé ou non
39.      */
```

```
40.     public ListeElectorale(int id,String nom, int voix, int sieges, boolean elimine) {
41. ...
42.     }
43.
44.     /**
45.      *
46.      * @return int : l'identifiant de la liste
47.      */
48.     public int getId() {
49. ...
50.     }
51.
52.     /**
53.      * initialise l'identifiant de liste
54.      * @param id int : identifiant de la liste
55.      * @throws ElectionsException si id<1
56.      */
57.     public void setId(int id) {
58. ...
59.     }
60.
61.     /**
62.      *
63.      * @return String : le nom de la liste
64.      */
65.     public String getNom() {
66. ...
67.     }
68.
69.     /**
70.      * initialise le nom de la liste
71.      * @param nom String : nom de la liste
72.      *  @throws ElectionsException si le nom est vide ou blanc
73.      */
74.     public void setNom(String nom) {
75. ...
76.     }
77.
78.     /**
79.      *
80.      * @return int : le nombre de voix de la liste
81.      */
82.     public int getVoix() {
83.  ...
84.     }
85.
86.     /**
87.      * initialise le nombre de voix de la liste
88.      * @param voix int : le nombre de voix de la liste
89.      */
90.     public void setVoix(int voix) {
91.  ...
92.     }
93.
94.     /**
95.      *
96.      * @return int : le nombre de sièges de la liste
97.      */
98.     public int getSieges() {
99.  ...
100.    }
101.
102.    /**
103.     * fixe le nombre de sièges de la liste
104.     * @param sieges int : le nombre de sièges de la liste
105.     */
106.    public void setSieges(int sieges) {
107....
108.    }
109.
110.    /**
111.     *
112.     * @return boolean : valeur du champ elimine
113.     */
114.    public boolean isElimine() {
115.  ...
116.    }
117.
118.    /**
119.     *
120.     * @param sieges int
121.     */
122.    public void setElimine(boolean elimine) {
123. ...
124.    }
125.
126.    /**
127.     *
128.     * @return String : identité de la liste électorale
```

```
129.     */
130.    public String toString() {
131.   ...
132.    }
133. }
```

- ligne 8 : n° identifiant une liste de façon unique. N'est pas indispensable ici mais est prévu pour une utilisation future.
- ligne 13 : le nom de la liste.
- ligne 17 : le nombre de voix de la liste
- ligne 21 : le nombre de sièges de la liste
- ligne 25 : booléen indiquant si la liste est éliminée (pourcentage de voix obtenues au-dessous du seuil électoral) ou non.

Chaque champ privé nommé [xyz] peut être initialisé par une méthode nommée [set**X**yz]. La méthode [get**X**yz] permet elle d'obtenir la valeur du champ privé [xyz]. Dans le cas particulier où [xyz] est un champ de type booléen, la méthode [get**X**yz] peut être remplacée par la méthode [is**X**yz]. Le nommage particulier de ces méthodes obéit à une norme de codage appelée norme **JavaBean**. Ainsi nous définissons les méthodes publiques suivantes :

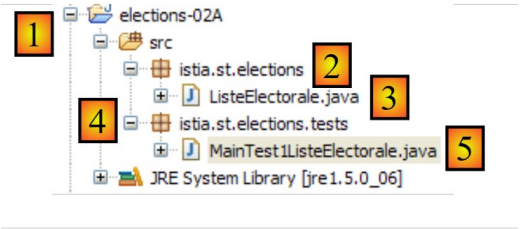
- getId (ligne 48), setId (ligne 57)
- getNom (ligne 65), setNom (ligne 74)
- getVoix (ligne 82), setVoix (ligne 90)
- getSieges (ligne 98), setSieges (ligne 106)
- isElimine (ligne 114), setElimine (ligne 122)

- lignes 30-31 : définissent un constructeur sans paramètres. Celui-ci permet de créer un objet [ListeElectorale] sans l'initialiser. Celui-ci peut ensuite être initialisé grâce aux méthodes **set**.
- lignes 40-42 : définissent un constructeur permettant de créer un objet [ListeElectorale] tout en initialisant ses cinq champs privés.
- ligne 130-132 : définissent la méthode [toString] qui rend une chaîne de caractères donnant les valeurs des cinq champs de l'objet.

Un programme de test de la classe **ListeElectorale** pourrait être le suivant :

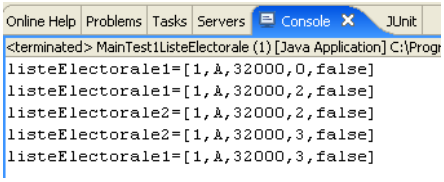
```
1.  package istia.st.elections.tests;
2.
3.  import istia.st.elections.ListeElectorale;
4.
5.  public class MainTest1ListeElectorale {
6.          public static void main(String[] args) {
7.                  // création d'une liste électorale
8.                  ListeElectorale listeElectorale1 = new ListeElectorale(1, "A", 32000,
9.                                  0, false);
10.                 // affichage identité liste
11.                 System.out.println("listeElectorale1=" + listeElectorale1);
12.                 // modification du nombre de sièges
13.                 listeElectorale1.setSieges(2);
14.                 // affichage identité liste 1
15.                 System.out.println("listeElectorale1=" + listeElectorale1);
16.                 // une nouvelle liste électorale
17.                 ListeElectorale listeElectorale2 = listeElectorale1;
18.                 // affichage identité liste 2
19.                 System.out.println("listeElectorale2=" + listeElectorale2);
20.                 // modification du nombre de sièges
21.                 listeElectorale2.setSieges(3);
22.                 // affichage identité des 2 listes
23.                 System.out.println("listeElectorale2=" + listeElectorale2);
24.                 System.out.println("listeElectorale1=" + listeElectorale1);
25.         }
26. }
```

L'environnement Eclipse de ce test pourrait être le suivant :



- [1] : le projet s'appelle [elections-02A]
- [2] : l'application sera placée dans un paquetage, ici [istia.st.elections]
- [3] : [ListeElectorale.java] est le code source de la classe [ListeElectorale]
- [4] : les classes de test seront placées dans un paquetage, ici [istia.st.elections.tests]
- [5] : la classe de test [MainTest1ListeElectorale]

L'affichage écran obtenu après exécution du programme ci-dessus est le suivant :



Travail à faire : en vous aidant de ce qui précède, complétez le code de la classe **ListeElectorale**.

3.3 Création d'une classe d'exception [ElectionsException]

Parmi les différentes classes d'exception du langage Java, il en est une appelée [RuntimeException]. Cette classe dérive de la classe [Exception], racine de toutes les classes d'exception. La particularité des instances de [RuntimeException] ou instances dérivées est que l'on n'est pas obligé de les déclarer ou de les gérer. On les appelle des **exceptions non contrôlées**.

Prenons un premier exemple. La classe [BufferedReader] est une classe dont les instances permettent de lire des lignes de texte dans un flux de données. Elle possède une méthode [readLine] dont la signature est la suivante :

```
public String readLine()throws IOException
```

On voit que la méthode peut lancer une exception de type [IOException]. L'arborescence de cette classe est la suivante :

```
1. java.lang.Object
2.   java.lang.Throwable
3.       java.lang.Exception
4.           java.io.IOException
```

La classe [IOException] dérive de la classe [Exception] (ligne 3). Le compilateur nous force à gérer et à déclarer les exceptions de type [java.lang.Exception] ou dérivé (sauf pour la branche [RuntimeException] que nous allons présenter plus loin). Ainsi, pour lire une ligne de texte tapée au clavier, on sera obligé d'écrire quelque chose comme :

```
1. BufferedReader clavier=....;
2. String ligne=null;
3. try{
4.       ligne=clavier.readLine();
5. }catch (IOException ex){
6.       // gérer l'exception
7.       ....
8. }
```

Prenons un autre exemple. Pour transformer une chaîne en entier on peut utiliser la méthode statique [Integer.parseInt] dont la signature est la suivante :

```
public static int parseInt(String s) throws NumberFormatException
```

L'argument [s] est la chaîne de caractères à transformer en entier. On voit que la méthode peut lancer une exception de type [NumberFormatException]. L'arborescence de cette classe est la suivante :

```
1. java.lang.Object
2.   java.lang.Throwable
3.       java.lang.Exception
4.           java.lang.RuntimeException
5.               java.lang.IllegalArgumentException
6.                   java.lang.NumberFormatException
```

La classe [NumberFormatException] dérive de la classe [RuntimeException] (ligne 4). Le compilateur ne nous force pas à gérer et à déclarer les exceptions de type [java.lang.RuntimeException] ou dérivé. Ainsi, on pourra écrire quelque chose comme :

```
1. BufferedReader clavier=....;
2. String ligne=null;
3. try{
4.       ligne=clavier.readLine();
5. }catch (IOException ex){
6.       // gérer l'exception
7.       ....
8. }
9. int age=Integer.parseInt(ligne);
```

Nous ne sommes pas obligés de mettre une clause [try - catch] pour gérer l'éventuelle exception générée par [Integer.parseInt] (ligne 9).

Il y a des avantages et inconvénients à créer et utiliser des classes d'exception dérivées de [RuntimeException] :

- au chapitre des avantages : le code est plus léger
- au chapitre des inconvénients : on peut être ramené aux méthodes du C où chaque fonction rend un code d'erreur que peu de gens utilisent, justement pour avoir un code plus léger. Lorsqu'une telle erreur non gérée se produit, le programme plante, généralement de façon peu élégante.

Nous décidons de créer une classe spéciale regroupant toutes les exceptions pouvant survenir dans notre application ELECTIONS. Elle s'appellera [ElectionsException] et dérivera de la classe [RuntimeException]. Son code est le suivant :

```
1. package istia.st.elections;
2.
3. public class ElectionsException extends RuntimeException {
4.   private static final long serialVersionUID = 1L;
5.
6.   public ElectionsException() {
7.     super();
8.   }
9.
10.    public ElectionsException(String message) {
11.        super(message);
12.    }
13.
14.    public ElectionsException(Throwable cause) {
15.        super(cause);
16.    }
17.
18.    public ElectionsException(String message, Throwable cause) {
19.        super(message, cause);
20.    }
21. }
```

- ligne 1 : nous plaçons la classe dans le paquetage [istia.st.elections] ;
- ligne 3 : la classe dérive de [RuntimeException]. Elle est donc **non contrôlée** ;
- ligne 4 : un identifiant de sérialisation qu'on peut ignorer pour le moment ;
- nous utiliserons dans notre application deux sortes de constructeur :
  - celui classique des lignes 15-17 comme ci-dessous :

```
throw new ElectionsException("Le nombre de sièges doit être >0")
```

Dans ce cas, la méthode qui appelle une méthode lançant une telle exception peut la gérer comme suit :

```
        // test exception
        try {
                listeElectorale2.setSieges(-3);
        } catch (ElectionsException ex) {
                System.err.println("L'exception suivante s'est produite : ["
                                + ex.toString() + "]");
        }
```

- ou celui des lignes 14-20 destiné à faire remonter une exception déjà survenue, en l'encapsulant dans une exception de type [ElectionsException] :

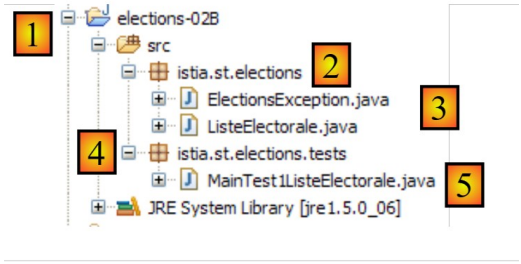
```
try {
	...;
	} catch (SQLException ex) {
		// on encapsule l'exception
		throw new ElectionsException("erreur de fermeture de la connexion à la BD",ex);
	}
```

Cette seconde méthode a l'avantage de conserver l'information que peut contenir la première exception. Dans ce cas, la méthode qui appelle une méthode lançant une telle exception peut la gérer comme suit :

```
	try {
		...;
	} catch (ElectionsException ex) {
		System.out.println(ex.getMessage() + ", Cause : "+ ex.getCause().getMessage());
		System.exit(1);
	}
```

Travail à faire : reprenez le code de la classe **ListeElectorale** de façon à ce que les méthodes **set** lancent une exception de type [ElectionsException] si l'initialisation demandée est incorrecte, comme par exemple initialiser le nom avec une chaîne vide.

Le projet Eclipse de test de cette nouvelle version pourrait être le suivant :



- [1] : le projet s'appelle [elections-02B]
- [2] : l'application est placée dans un paquetage, ici [istia.st.elections]
- [3] : les classes [ListeElectorale] et [ElectionsException]
- [4] : les classes de test sont placées dans un paquetage, ici [istia.st.elections.tests]
- [5] : la classe de test [MainTest1ListeElectorale]

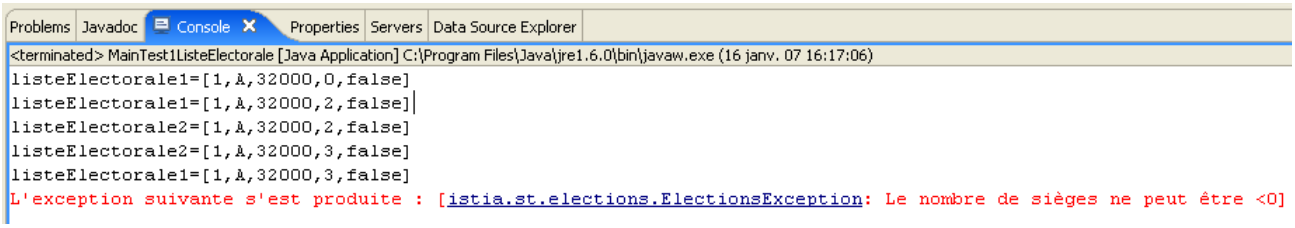
La classe de test [MainTest1ListeElectorale] déjà étudiée est légèrement modifiée pour tester les cas d'exception :

```
1.  package istia.st.elections.tests;
2.
3.  import istia.st.elections.ElectionsException;
4.  import istia.st.elections.ListeElectorale;
5.
6.  public class MainTest1ListeElectorale {
7.  	public static void main(String[] args) {
8.  		// création d'une liste électorale
9.  		ListeElectorale listeElectorale1 = new ListeElectorale(1, "A", 32000,
10. 				0, false);
11. 		// affichage identité liste
12. 		System.out.println("listeElectorale1=" + listeElectorale1);
13. 		// modification du nombre de sièges
14. 		listeElectorale1.setSieges(2);
15. 		// affichage identité liste 1
16. 		System.out.println("listeElectorale1=" + listeElectorale1);
17. 		// une nouvelle liste électorale
18. 		ListeElectorale listeElectorale2 = listeElectorale1;
19. 		// affichage identité liste 2
20. 		System.out.println("listeElectorale2=" + listeElectorale2);
21. 		// modification du nombre de sièges
22. 		listeElectorale2.setSieges(3);
23. 		// affichage identité des 2 listes
24. 		System.out.println("listeElectorale2=" + listeElectorale2);
25. 		System.out.println("listeElectorale1=" + listeElectorale1);
26. 		// test exception
27. 		try {
28. 			listeElectorale2.setSieges(-3);
29. 		} catch (ElectionsException ex) {
30. 			System.err.println("L'exception suivante s'est produite : ["
```

```
31.                                         + ex.toString() + "]");
32.                 }
33.
34.         }
35. }
```

- ligne 28 : on essaie d'initialiser le nombre de sièges avec une valeur interdite
- ligne 30 : s'il y a exception, elle est affichée

L'exécution du test donne les résultats suivants :



On remarque que la classe [ListeElectorale] a bien généré une exception lorsqu'on a voulu initialiser le nombre de sièges avec une valeur invalide (ligne 28 du code).

3.4 Une classe de test unitaire

Le type de test précédent repose sur une vérification visuelle. On vérifie qu'on obtient à l'écran ce qui est attendu. C'est une méthode à déconseiller en milieu professionnel. Les tests doivent toujours être automatisés au maximum et viser à ne nécessiter aucune intervention humaine. L'être humain est en effet sujet à la fatigue et sa capacité à vérifier des tests s'émousse au fil de la journée.

Une application évolue au fil du temps. A chaque évolution, on doit vérifier que l'application ne "régresse" pas, c.a.d. qu'elle continue à passer les tests de bon fonctionnement qui avaient été faits lors de son écriture initiale. On appelle ces tests, des tests de "non régression". Une application un peu importante peut nécessiter des centaines de tests. On teste en effet chaque méthode de chaque classe de l'application. On appelle cela des **tests unitaires**. Ceux-ci peuvent mobiliser beaucoup de développeurs s'ils n'ont pas été automatisés.

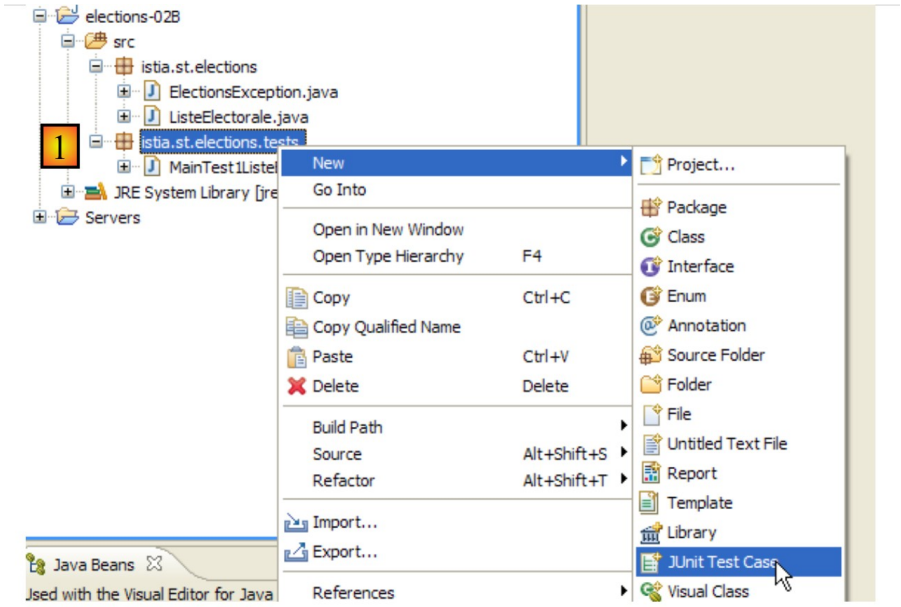
Des outils ont été développés pour automatiser les tests. L'un d'eux s'appelle [JUnit]. C'est une bibliothèque de classes destinées à gérer les tests. Nous allons utiliser cet outil pour tester la classe [ListeElectorale].

Un programme de test JUnit (versions 4.x) a la forme suivante :

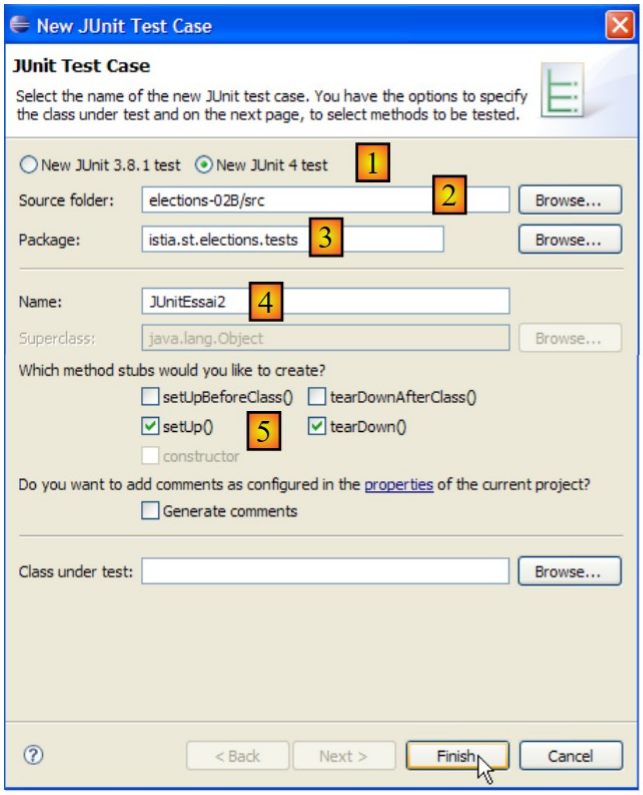
```
1.  package istia.st.elections.tests;
2.
3.  import org.junit.Assert;
4.
5.  import org.junit.After;
6.  import org.junit.Before;
7.  import org.junit.Test;
8.
9.  public class JUnitEssai {
10.
11.         @Before
12.         public void avant() throws Exception {
13.                 System.out.println("tearUp");
14.         }
15.
16.         @After
17.         public void après() throws Exception {
18.                 System.out.println("tearDown");
19.         }
20.
21.         @Test
22.         public void t1() {
23.                 System.out.println("test1");
24.                 Assert.assertEquals(1, 1);
25.         }
26.
27.         @Test
28.         public void t2() {
29.                 System.out.println("test2");
30.                 Assert.assertEquals(1, 2);
31.         }
32.
33. }
```

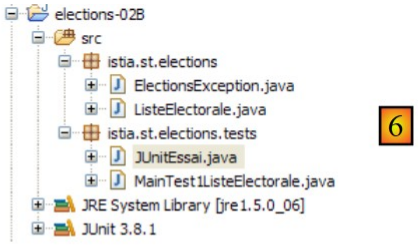
- ligne 1 : la classe a été placée dans le paquetage [istia.st.elections.tests] ;
- ligne 11 : la méthode annotée par l'annotation [**@Before**] est exécutée **avant** chaque test unitaire ;
- ligne 16 : la méthode annotée par l'annotation [**@After**] est exécutée **après** chaque test unitaire ;
- ligne 21 : une méthode annotée par l'annotation [**@Test**] est une méthode testée par le test unitaire. Le méthodes annotées par [@Test] seront exécutées les unes après les autres, sauf indication contraire du testeur qui peut sélectionner lui-même les méthodes à tester. Avant chaque exécution d'une méthode [@Test], la méthode [@Before] est exécutée. Après chaque exécution d'une méthode [@Test], la méthode [@After] est exécutée ;
- lignes 22-25 : définissent une méthode [t1] de test ;
- ligne 18 : l'une des méthodes [**Assert.assert***] qui permet de vérifier des assertions. On trouve les méthodes [assert] suivantes :
  - **assertEquals(expression1, expression2)** : vérifie que les valeurs des deux expressions sont égales. De nombreux types d'expression sont acceptés (int, String, float, double, boolean, char, short). Si les deux expressions ne sont pas égales, alors une exception de type [AssertionFailedError] est lancée,
  - **assertEquals(réel1, réel2, delta)** : vérifie que deux réels sont égaux à **delta** près, c.a.d abs(réel1-réel2)<=delta. On pourra écrire par exemple **assertEquals(réel1, réel2, 1E-6)** pour vérifier que deux valeurs sont égales à 10^{-6} près,
  - **assertEquals(message, expression1, expression2)** et **assertEquals(message, réel1, réel2, delta)** sont des variantes permettant de préciser le message d'erreur à associer à l'exception de type [AssertionFailedError] lancée lorsque la méthode [assertEquals] échoue,
  - **assertNotNull(Object)** et **assertNotNull(message, Object)** : vérifie que **Object** n'est pas égal à **null,**
  - **assertNull(Object)** et **assertNull(message, Object)** : vérifie que **Object** est égal à **null,**
  - **assertSame(Object1, Object2)** et **assertSame(message, Object1, Object2)** : vérifie que les références **Object1** et **Object2** pointent sur le même objet,
  - **assertNotSame(Object1, Object2)** et **assertNotSame(message, Object1, Object2)** : vérifie que les références **Object1** et **Object2** ne pointent pas sur le même objet ;
- ligne 24 : cette assertion doit réussir ;
- ligne 30 : cette assertion doit échouer ;

Dans l'environnement Eclipse, la création d'une classe de test JUnit peut se faire de la façon suivante :



- [1] : clic droit sur le paquetage dans lequel on veut ajouter la classe de test , puis option [JUnit / New / JUnit Test Case]





- [1] : choix d'une version JUnit ;
- [2] : choix du dossier dans lequel la classe de test doit être créée ;
- [3] : choix du paquetage dans lequel la classe de test doit être créée ;
- [4] : nom de la classe de test ;
- [5] : choix des méthodes à inclure dans la classe qui va être générée ;
- [6] : la classe *JUnitEssai* a été générée

L'assistant précédent génère une classe quasiment vide :

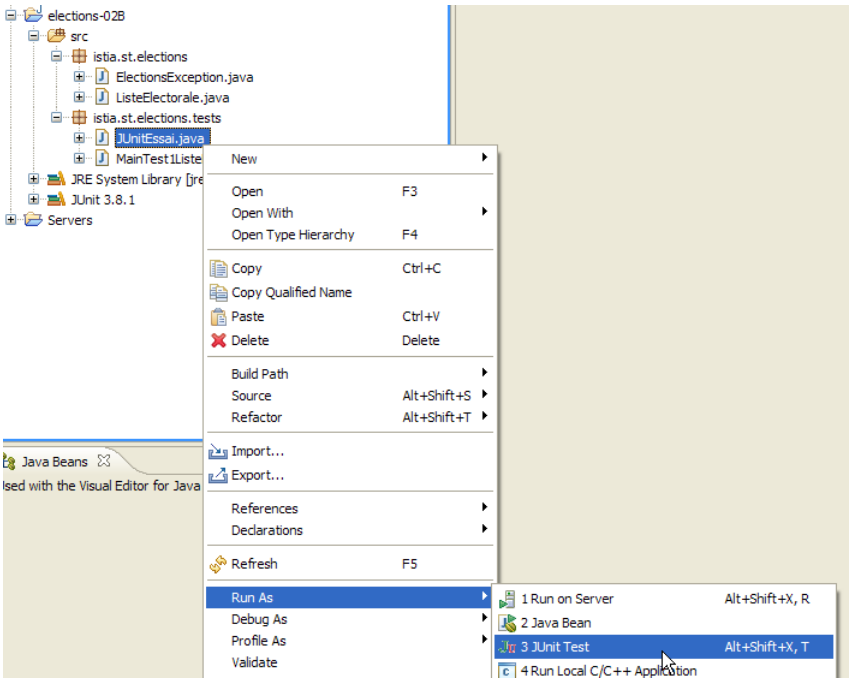
```
1.  package istia.st.elections.tests;
2.
3.  import org.junit.Assert;
4.  import org.junit.After;
5.  import org.junit.Before;
6.
7.  public class JUnitEssai {
8.
9.          @Before
10.         public void setUp() throws Exception {
11.         }
12.
13.         @After
14.         public void tearDown() throws Exception {
15.         }
16. }
```

Complétons et modifions le code précédent de la façon suivante :

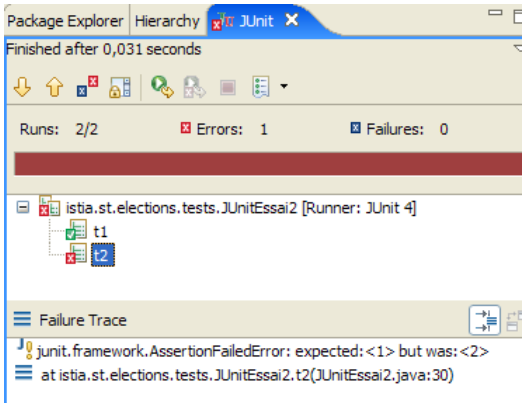
```
1.  package istia.st.elections.tests;
2.
3.  import org.junit.Assert;
4.
5.  import org.junit.After;
6.  import org.junit.Before;
7.  import org.junit.Test;
8.
9.  public class JUnitEssai2 {
10.
11.         @Before
12.         public void avant() throws Exception {
```

```
13.                     System.out.println("tearUp");
14.             }
15.
16.             @After
17.             public void après() throws Exception {
18.                     System.out.println("tearDown");
19.             }
20.
21.             @Test
22.             public void t1() {
23.                     System.out.println("test1");
24.                     Assert.assertEquals(1, 1);
25.             }
26.
27.             @Test
28.             public void t2() {
29.                     System.out.println("test2");
30.                     Assert.assertEquals(1, 2);
31.             }
32.
33. }
```

Sous Eclipse, un clic droit sur la classe de test, puis option [Run as / JUnit test], permet de l'exécuter :



Les résultats obtenus à l'exécution de ce test sont les suivants :

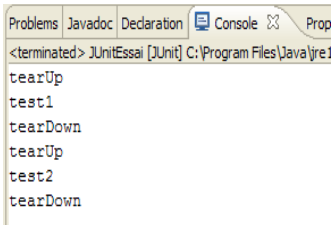


Ci-dessus la méthode [test2] a échoué. A chaque fois qu'un test échoue, un message d'erreur lui est associé. Pour [test2], c'est celui affiché ci-dessus. Le message indique le n° de la ligne où l'erreur s'est produite (ligne 30). Ligne 30, l'appel qui a échoué était :

```
    Assert.assertEquals(1, 2);
```

Le premier paramètre est appelé la valeur attendue, le second la valeur réelle. Le message d'erreur de [test2] ci-dessus indique que la valeur attendue était 2 mais que la valeur réelle a été 3.

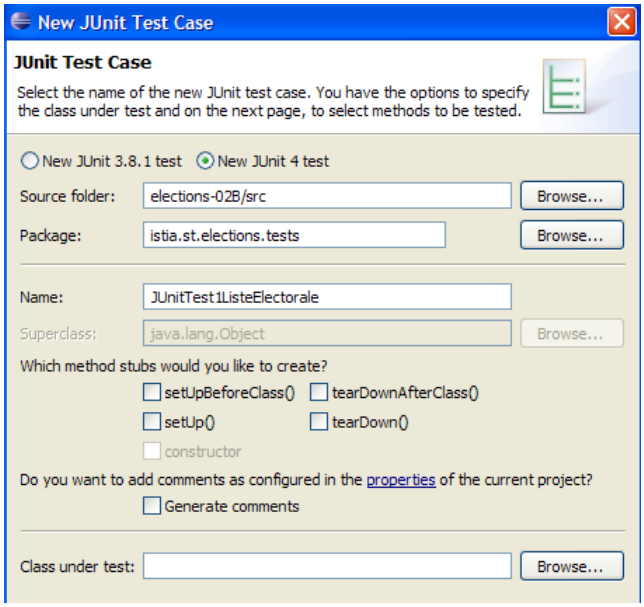
Enfin, les messages écrits sur la console par les différentes méthodes de test ont été les suivants :

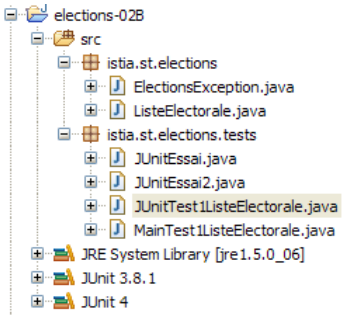


Ces messages montrent que les méthodes [@Before] et [@After] ont bien été appelées respectivement, avant et après chaque méthode de test.

Les classes de test ne sont pas nécessairement écrites par les développeurs eux-mêmes. Elles peuvent l'être par les personnes qui ont écrit les spécifications de l'application. Certaines méthodes de développement dites TDD (Test Driven Development) préconisent l'écriture des classes de tests avant même l'écriture des classes à tester. Cela permet parfois de clarifier des spécifications qui pourraient sinon être interprétées de plusieurs façons.

Créons un test JUnit 4, appelé [JUnitTest1ListeElectorale], pour la classe [ListeElectorale]. Sous Eclipse, on procédera comme décrit précédemment :



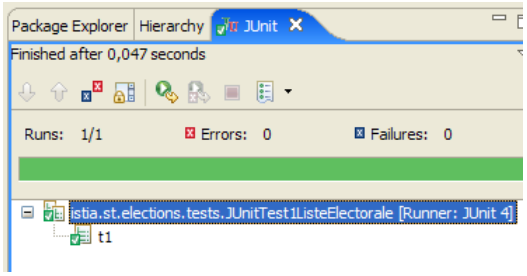


Nous complétons le code généré par l'assistant de la façon suivante :

```
1.  package istia.st.elections.tests;
2.
3.  import org.junit.Assert;
4.  import istia.st.elections.ElectionsException;
5.  import istia.st.elections.ListeElectorale;
6.
7.  import org.junit.Test;
8.
9.  public class JUnitTest1ListeElectorale {
10.
11.         @Test
12.         public void t1() {
13.                 // création liste électorale
14.                 ListeElectorale liste = new ListeElectorale(1, "a", 32000, 0, false);
15.                 // vérifications
16.                 Assert.assertEquals("a", liste.getNom());
17.                 Assert.assertEquals(32000, liste.getVoix());
18.                 Assert.assertEquals(false, liste.isElimine());
19.                 Assert.assertEquals(0, liste.getSieges());
20.                 // vérification validité id
```

```
21.                     boolean erreur = false;
22.                     try {
23.                             liste.setId(-4);
24.                     } catch (ElectionsException e) {
25.                             erreur = true;
26.                     }
27.                     Assert.assertEquals(true, erreur);
28.                     // vérification validité nom
29.                     erreur = false;
30.                     try {
31.                             liste.setNom("");
32.                     } catch (ElectionsException e) {
33.                             erreur = true;
34.                     }
35.                     Assert.assertEquals(true, erreur);
36.                     // vérification validité voix
37.                     erreur = false;
38.                     try {
39.                             liste.setVoix(-4);
40.                     } catch (ElectionsException e) {
41.                             erreur = true;
42.                     }
43.                     Assert.assertEquals(true, erreur);
44.                     // vérification validité sièges
45.                     erreur = false;
46.                     try {
47.                             liste.setSieges(-4);
48.                     } catch (ElectionsException e) {
49.                             erreur = true;
50.                     }
51.                     Assert.assertEquals(true, erreur);
52.             }
53.
54.     }
```

L'exécution du test donne le résultat suivant :



Les tests sont réussis. Nous considèrerons désormais que nous avons une classe [ListeElectorale] opérationnelle.

3.5 MainElections : version 2

Lectures conseillées :

- ✗ paragraphes 2.1, 2.2, 2.4 et 2.7 du chapitre 2 de [1] : Classes et interfaces
- ✗ paragraphes 3.3 (classe String), 3.5 (classe ArrayList), 3.6 (classe Arrays)

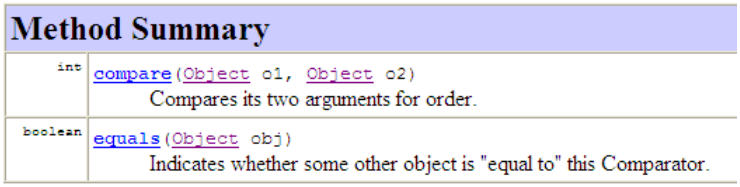
On désire réécrire l'application [Elections] en y ajoutant les nouvelles contraintes suivantes :

- on utilisera la classe [ListeElectorale] pour représenter une liste candidate
- l'application demandera au clavier les informations suivantes :
  - le nombre de sièges à pourvoir
  - les noms et voix des listes. On ne sait pas à priori combien il y a de listes. La dernière liste sera signalée par un nom égal à la chaîne "*".
- parce qu'on ne connaît pas à priori le nombre de listes, celles-ci seront tout d'abord mémorisées dans un objet de type [ArrayList]. Puis, lorsque toutes les listes auront été saisies, elles seront transférées dans un tableau de listes.
- les résultats seront affichés par ordre décroissant du nombre de sièges obtenus.

Pour trier un tableau T, on dispose de différentes méthodes statiques de la classe [Arrays] :

- **Arrays.sort(T)** : trie le tableau T selon un ordre naturel s'il en a un (croissant pour les nombres, les dates, alphabétique pour les chaînes, ...)
- **Arrays.sort(T,comparateur)** : pour trier des tableaux T n'ayant pas un ordre naturel. C'est le cas ici du tableau des listes qu'il faut trier selon un champ particulier de la liste : le nombre de sièges obtenus.

Dans la méthode **Arrays.sort(T,comparateur)**, le paramètre **comparateur** est un objet implémentant l'interface **Comparator** suivante :



- la méthode **compare** permet de comparer deux éléments du tableau T
- la méthode **equals** permet de dire si deux objets sont égaux

Les deux méthodes comparent des types **Object** *obj1* et *obj2*. Dire que *obj1<obj2* ou *obj1=obj2* ou *obj1>obj2* dépend de la relation d'ordre que l'on veut créer entre les deux objets. C'est au développeur qui implémente cette interface d'indiquer comment on sait que :

- *obj1* est plus petit que *obj2*
- *obj1* est plus grand que *obj2*
- *obj1* est égal à *obj2*

La classe **Object** dont dérive toute classe Java dispose déjà d'une méthode [equals]. Pour trier un tableau T d'objets de type O, la méthode [equals] de la classe O n'est pas utile. On peut donc laisser l'implémentation par défaut apportée par la classe **Object**. Seule la méthode [compare] doit être alors implémentée. Cette méthode est appelée de façon répétée par la méthode [**Arrays.sort**]. Celle-ci va à chaque fois passer comme paramètres **obj1** et **obj2** de la méthode **compare**, deux éléments du tableau **T** à trier. Dans notre cas, ces éléments seront de type [ListeElectorale]. On notera ici le polymorphisme à l'oeuvre. La méthode [compare] est définie comme recevant des paramètres de type [Object]. Cela veut dire qu'elle peut recevoir des paramètres de type [Object] ou dérivé (polymorphisme). Comme [Object] est la classe parent de toutes les classes Java, les paramètres effectifs peuvent avoir le type [ListeElectorale].

Pour un tri dans l'ordre croissant, la méthode [compare] doit rendre :

- -1 si *obj1* est plus petit que *obj2*
- +1 si *obj1* est plus grand que *obj2*
- 0 si *obj1* est égal à *obj2*

Pour un tri dans l'ordre décroissant, les valeurs +1 et -1 sont inversées. Les termes "est plus petit que", "est plus grand que", "est égal à" expriment une relation d'ordre. Pour des objets de type [ListeElectorale], on aura la relation **liste1** "est plus petit que" **liste2** si **liste1** a moins de voix que **liste2**.

Dans le même fichier source que la classe [MainElections], on pourra ajouter une deuxième classe :

```
1. // classe de comparaison de listes électorales
2. class CompareListesElectorales implements Comparator {
3.
4.     // comparaison de deux listes électorales selon le nombre de voix
5.     public int compare(Object obj1, Object obj2) {
6.         // on récupère les listes électorales
7.         ListeElectorale listeElectorale1 = (ListeElectorale) obj1;
8.         ListeElectorale listeElectorale2 = (ListeElectorale) obj2;
9.         // on compare les voix de ces deux listes
10. ....
11.     }
12. }
```

- ligne 2 : la classe n'est pas déclarée **publique**. Dans un fichier source Java, il peut y avoir plusieurs classes mais une seule peut avoir l'attribut **public**, celle qui porte le nom du fichier source.

Dans la méthode *compare* précédente, les paramètres sont de type *Object,* ce qui oblige lignes 7 et 8, à faire un transtypage des paramètres de la méthode, du type *Object* vers le type *ListeElectorale*. La signature de la méthode *compare* est imposée par l'interface *Comparator* qui a été écrite pour comparer des objets quelconques. Depuis le JDK 1.5, il existe une interface *Comparator* générique : *Comparator<T>* où *T* est un type Java quelconque. La méthode *compare* de l'interface *Comparator<T>* compare des objets de type *T* et non de type *Object* ce qui évite les transtypages précédents. La classe de comparaison des objets de type *ListeElectorale* pourrait ressembler à ceci :

```
1.  // classe de comparaison de listes électorales
2.  class CompareListesElectorales implements Comparator<ListeElectorale> {
```

```
3.
4.          // comparaison de deux listes électorales selon le nombre de sièges
5.          public int compare(ListeElectorale listeElectorale1,
6.                             ListeElectorale listeElectorale2) {
7.  ...
8.          }
9.  }
```

- ligne 2 : la classe implémente l'interface *Comparator<ListeElectorale>*
- lignes 5-6 : les paramètres de la méthode *compare* sont de type *ListeElectorale*. Le transtypage est désormais inutile.

Le JDK 1.5 a introduit le concept de classe / interface générique pour diverses classes / interfaces du JDK 1.4 qui manipulaient initialement uniquement des objets de type *Object*. C'est le cas des listes, dictionnaires, ...

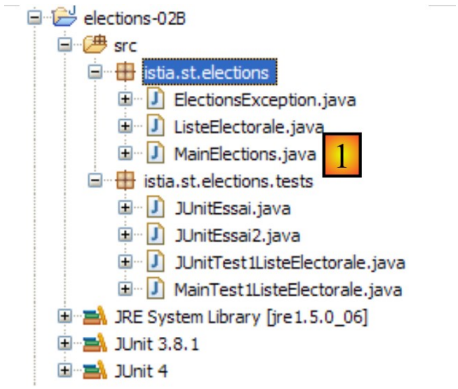
Nous avons dit un peu plus haut que, parce qu'on ne connaissait pas le nombre de listes, on ne pouvait mémoriser celles-ci dans un tableau. Elles peuvent être mémorisées dans un objet *ArrayList* qui implémente la notion de "liste d'objets". Cette classe mémorise des objets de type *Object*. Depuis le JDK 1.5, il existe des listes d'objets typés. Ainsi on utilisera un objet *ArrayList<ListeElectorale>* pour mémoriser les listes avant de les transférer dans un tableau. Si ce dernier s'appelle *tListes*, son tri sera obtenu par l'instruction :

```
1.  // tri des listes
2.  Arrays.sort(tListes, new CompareListesElectorales());
```

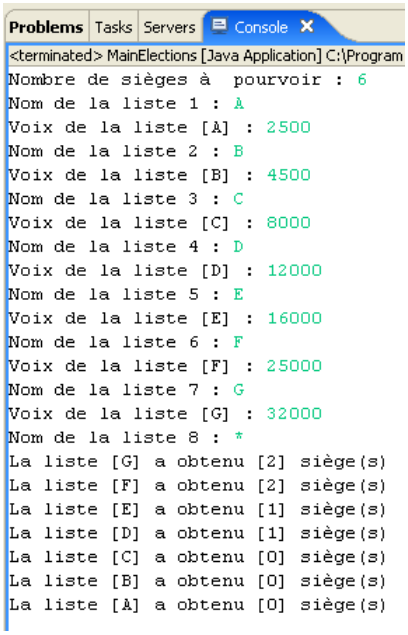
où *CompareListesElectorales* est la classe implémentant l'interface *Comparator<ListeElectorale>*.

Travail à faire : réécrivez l'application [Elections] en tenant compte de ces nouvelles spécifications.

Le projet Eclipse pourrait être le suivant :



Un exemple d'exécution de [1] est le suivant :



4 [TD] : Architectures en couches

Mots clés : architecture multicouche, Spring, injection de dépendances.

4.1 Introduction

Rappelons ce qui a été fait :

- dans la partie 1 de l'exercice ELECTIONS aucune classe n'a été utilisée. On a construit une solution comme on l'aurait construite en langage C.
- dans la partie 2 de l'exercice, deux classes ont été introduites :
  - [ListeElectorale] qui représente les attributs (id, nom, voix, sièges, élimine) d'une liste candidate
  - [ElectionsException] une classe d'exceptions non contrôlées. Ce type d'exception est utilisé à chaque fois que se produit une erreur fatale dans l'application des élections. Elle est non contrôlée, c.a.d. que le développeur n'est pas obligé de la gérer avec un try-catch.

Le calcul du résultat des élections a été confié jusqu'à maintenant à une méthode [main] d'une classe [MainElections]

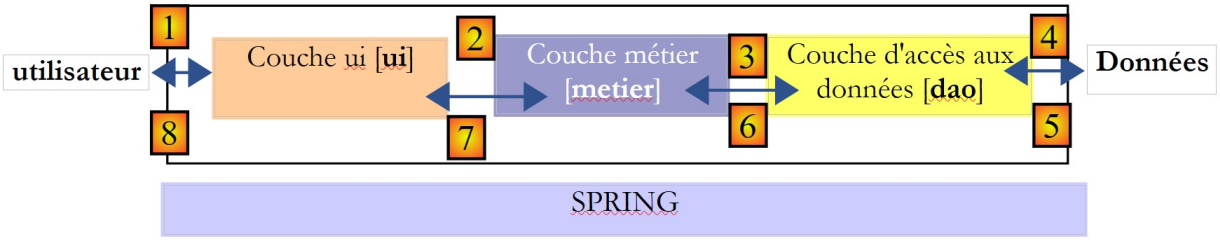
```
1. package istia.st.elections;
2.
3. import java.io.*;
4.
5. public class MainElections {
6.
7.    // qqs données
8.    private static final double barre = 0.05;
9.
10.   // ------------------------------------------------------------------
11.   // la procédure principale
12.   public static void main(String[] arguments) throws IOException {
13.
14.     // on prépare le flux d'entrée clavier
15.     BufferedReader clavier = new BufferedReader(new InputStreamReader(System.in));
16.
17.     // saisie des données nécessaires au calcul des sièges
18. ...
19.     // calcul des sièges obtenus par les différentes listes
20. ....
21.     // affichage des résultats
22. ...
23.   } // main
24. } // classe
```

La solution précédente inclut trois phases classiques :

- l'acquisition des données, lignes 17-18
- le calcul de la solution, lignes 19-20
- l'affichage et / ou la persistance des résultats, lignes 21-22

Seule la phase 2 est vraiment constante. La phase 1 peut varier : les données peuvent venir du clavier comme dans les exemples étudiés, d'un fichier texte, d'une interface graphique, d'une base de données, du réseau, ... De même il y a de multiples façons de restituer les résultats dans la phase 3 : les afficher à l'écran comme cela a été fait dans les exemples étudiés, les enregistrer dans un fichier, dans une base de données, les envoyer sur le réseau, ...

De façon plus générale, une application peut souvent être modélisée en trois couches ayant chacune un rôle bien défini :



On appelle également cette architecture, "**architecture trois tiers**", traduction de l'anglais "three tier architecture". Le terme "trois tiers" désigne normalement une architecture où chaque tier est sur une machine différente. Lorsque les tiers sont sur une même machine, l'architecture devient une architecture "**trois couches**".

- la couche [metier] est celle qui contient les règles métier de l'application. Pour notre application d'élections, ce sont les règles qui permettent de calculer les sièges obtenus par les différentes listes, une fois que l'on connaît les voix obtenues par chacune d'elles. Cette couche a besoin de données pour travailler. Par exemple dans l'application d'élections :
  - les listes avec pour chacune son nom et son nombre de voix
  - le nombre de sièges à pourvoir
  - le seuil électoral au-dessous duquel, une liste est éliminée

  Dans le schéma ci-dessus, les données peuvent provenir de deux endroits :

  - la couche d'accès aux données ou [dao] (DAO = Data Access Object) pour les données déjà enregistrées dans des fichiers ou bases de données. Ce pourrait être le cas ici du nom des listes, du nombre de sièges à pourvoir, du seuil électoral. On connaît en effet ces informations avant l'élection elle-même.
  - la couche d'interface avec l'utilisateur ou [ui] (UI = User Interface) pour les données saisies par l'utilisateur ou affichées à l'utilisateur. Ce pourrait être le cas ici des voix des listes qui ne sont connues qu'au dernier moment ainsi que de l'affichage des résultats de l'élection.

- de façon générale, la couche [dao] s'occupe de l'accès aux données persistantes (fichiers, bases de données) ou non persistantes (réseau, capteurs, ...).
- la couche [ui] elle, s'occupe des interactions avec l'utilisateur s'il y en a un.
- les trois couches sont rendues indépendantes grâce à l'utilisation d'interfaces Java.
- pour intégrer ces couches ensemble dans l'application, il existe différentes méthodes. Nous serons amenés à utiliser un outil appelé " Spring ". Sur le schéma, il est transversal aux autres couches.

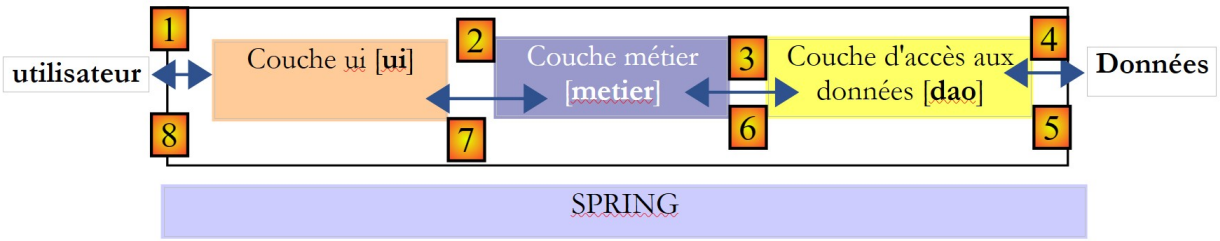
Nous allons reprendre l'application [Elections] développée précédemment pour lui donner une architecture 3 couches. Pour cela, nous allons étudier les couches [ui, metier, dao] les unes après les autres, en commençant par la couche [dao], couche qui s'occupe des données persistantes.

Auparavant, il nous faut définir les interfaces des différentes couches de l'application [Elections].

4.2 Les interfaces de l'application [Elections]

Rappelons qu'une interface définit un ensemble de signatures de méthodes. Les classes implémentant l'interface donnent un contenu à ces méthodes.

Revenons à l'architecture 3 couches de notre application :



Dans ce type d'architecture, c'est souvent l'utilisateur qui prend les initiatives. Il fait une demande en [1] et reçoit une réponse en [8]. On appelle cela le cycle demande - réponse. Prenons l'exemple du calcul des sièges obtenus au soir des élections. Celui-ci va nécessiter plusieurs étapes :

(a) la couche [ui] va devoir demander à l'utilisateur le nombre de voix obtenues par chacune des listes. Pour cela elle va devoir présenter à celui-ci le nom des listes en compétition. L'utilisateur n'aura alors qu'à mettre le nombre de voix en face de chaque liste puis à demander le calcul des sièges.
(b) la couche [ui] ne dispose pas du nom des listes. Celles-ci sont enregistrées dans la source de données à droite du schéma. Elle va utiliser le chemin [2, 3, 4, 5, 6, 7] pour les obtenir. L'opération [2] est la demande des listes, l'opération [7] la réponse à cette demande. Ceci fait, elle peut les présenter à l'utilisateur par [8].

(c) l'utilisateur va transmettre à la couche [ui] le nombre de voix obtenues par chacune des listes. C'est l'opération [1] ci-dessus. Au cours de cette étape, l'utilisateur n'interagit qu'avec la couche [ui]. C'est celle-ci qui va notamment vérifier la validité des données saisies. Ceci fait, l'utilisateur va demander la liste des sièges obtenus par chacune des listes.
(d) la couche [ui] va demander à la couche métier de faire le calcul des sièges. Pour cela elle va lui transmettre les données qu'elle a reçues de l'utilisateur. C'est l'opération [2].
(e) la couche [metier] a besoin de certaines informations pour mener à bien son travail. Elle a déjà les listes depuis l'opération (b). Il lui faut également le nombre de sièges à pourvoir ainsi que la valeur du seuil électoral. Elle va demander ces informations à la couche [dao] avec le chemin [3, 4, 5, 6]. [3] est la demande initiale et [6] la réponse à cette demande.
(f) ayant toutes les données dont elle avait besoin, la couche [metier] calcule les sièges obtenus par chacune des listes.
(g) la couche [metier] peut maintenant répondre à la demande de la couche [ui] faite en (d). C'est le chemin [7].
(h) la couche [ui] va mettre en forme ces résultats pour les présenter à l'utilisateur sous une forme appropriée puis les présenter. C'est le chemin [8].
(i) on peut imaginer que ces résultats doivent être mémorisés dans un fichier ou une base de données. Cela peut être fait de façon automatique. Dans ce cas, après l'opération (f), la couche [metier] va demander à la couche [dao] d'enregistrer les résultats. Ce sera le chemin [3, 4, 5, 6]. Cela peut être fait également seulement sur demande de l'utilisateur. Ce sera le chemin [1-8] qui sera utilisé par le cycle demande - réponse.

On voit dans cette description qu'une couche est amenée à utiliser les ressources de la couche qui est à sa droite, jamais de celle qui est à sa gauche. Considérons deux couches contigües :

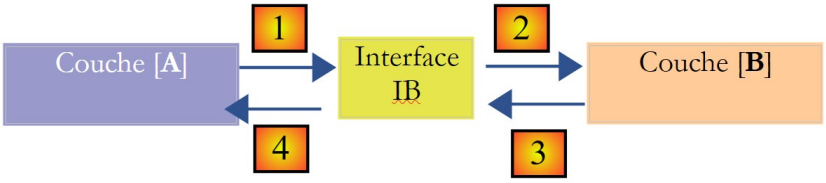
La couche [A] fait des demandes à la couche [B]. Dans les cas les plus simples, une couche est implémentée par une unique classe. Une application évolue au cours du temps. Ainsi la couche [B] peut avoir des classes d'implémentation différentes [B1, B2, ...]. Si la couche [B] est la couche [dao], celle-ci peut avoir une première implémentation [B1] qui va chercher des données dans un fichier. Quelques années plus tard, on peut vouloir mettre les données dans une base de données. On va alors construire une seconde classe d'implémentation [B2]. Si dans l'application initiale, la couche [A] travaillait directement avec la classe [B1] on est obligés de réécrire partiellement le code de la couche [A]. Supposons par exemple qu'on ait écrit dans la couche [A] quelque chose comme suit :

```
1. B1 b1=new B1(...);
2. ..
3. b1.getData(...);
```

- ligne 1 : une instance de la classe [B1] est créée
- ligne 3 : des données sont demandées à cette instance

Si on suppose, que la nouvelle classe d'implémentation [B2] utilise des méthodes de même signature que celle de la classe [B1], il faudra changer tous les [B1] en [B2]. Ca, c'est le cas très favorable et assez improbable si on n'a pas prêté attention à ces signatures de méthodes. Dans la pratique, il est fréquent que les classes [B1] et [B2] n'aient pas les mêmes signatures de méthodes et que donc une bonne partie de la couche [A] doive être totalement réécrite.

On peut améliorer les choses si on met une interface entre les couches [A] et [B]. Cela signifie qu'on fige dans une interface les signatures des méthodes présentées par la couche [B] à la couche [A]. Le schéma précédent devient alors le suivant :



La couche [A] ne s'adresse désormais plus directement à la couche [B] mais à son interface [IB]. Ainsi dans le code de la couche [A], la classe d'implémentation [Bi] de la couche [B] n'apparaît qu'une fois, au moment de l'implémentation de l'interface [IB]. Ceci fait, c'est l'interface [IB] et non sa classe d'implémentation qui est utilisée dans le code. Le code précédent devient celui-ci :

```
1. IB ib=new B1(...);
2. ..
```

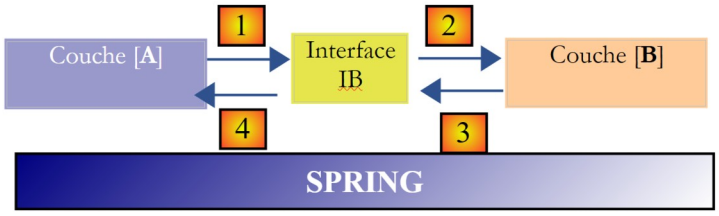
```
3. ib.getData(...);
```

- ligne 1 : une instance [ib] implémentant l'interface [IB] est créée par instanciation de la classe [B1]
- ligne 3 : des données sont demandées à l'instance [ib]

Désormais si on remplace l'implémentation [B1] de la couche [B] par une implémentation [B2], et que ces deux implémentations respectent la même interface [IB], alors seule la ligne 1 de la couche [A] doit être modifiée et aucune autre. C'est un grand avantage qui à lui seul justifie l'usage systématique des interfaces entre deux couches.

On peut aller encore plus loin et rendre la couche [A] totalement indépendante de la couche [B]. Dans le code ci-dessus, la ligne 1 pose problème parce qu'elle référence en dur la classe [B1]. L'idéal serait que la couche [A] puisse disposer d'une implémentation de l'interface [IB] sans avoir à nommer de classe. Ce serait cohérent avec notre schéma ci-dessus. On y voit que la couche [A] s'adresse à l'interface [IB] et on ne voit pas pourquoi elle aurait besoin de connaître le nom de la classe qui implémente cette interface. Ce détail n'est pas utile à la couche [A].

Le framework Spring (http://www.springframework.org) permet d'obtenir ce résultat. L'architecture précédente évolue de la façon suivante :



La couche transversale [Spring] va permettre à une couche d'obtenir par configuration une référence sur la couche située à sa droite sans avoir à connaître le nom de la classe d'implémentation de la couche. Ce nom sera dans les fichiers de configuration et non pas dans le code Java. Le code Java de la couche [A] prend alors la forme suivante :

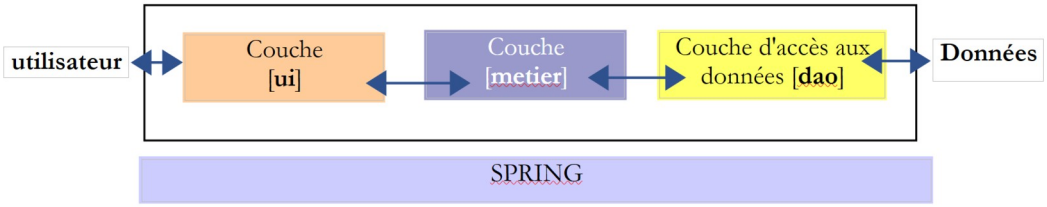
```
1.  IB ib; // initialisé par Spring
2.  ..
3.  ib.getData(...);
```

- ligne 1 : une instance [ib] implémentant l'interface [IB] de la couche [B]. Cette instance est créée par Spring sur la base d'informations trouvées dans un fichier de configuration. Spring va s'occuper de créer :
    - l'instance [b] implémentant la couche [B]
    - l'instance [a] implémentant la couche [A]. Cette instance sera initialisée. Le champ [ib] ci-dessus recevra pour valeur la référence [b] de l'objet implémentant la couche [B]
    - ligne 3 : des données sont demandées à l'instance [ib]

On voit maintenant que, la classe d'implémentation [B1] de la couche B n'apparaît nulle part dans le code de la couche [A]. Lorsque l'implémentation [B1] sera remplacée par une nouvelle implémentation [B2], rien ne changera dans le code de la classe [A]. On changera simplement les fichiers de configuration de Spring pour instancier [B2] au lieu de [B1].

Le couple **Spring** et **interfaces Java** apporte une amélioration décisive à la maintenance d'applications en rendant les couches de celles-ci étanches entre elles. C'est cette solution que nous utiliserons pour l'application [Elections].

Revenons à l'architecture trois couches de notre application :



Dans les cas simples, on peut partir de la couche [metier] pour découvrir les interfaces de l'application. Pour travailler, elle a besoin de données :

- déjà disponibles dans des fichiers, bases de données ou via le réseau. Elles sont fournies par la couche [dao].
- pas encore disponibles. Elles sont alors fournies par la couche [ui] qui les obtient auprès de l'utilisateur de l'application.

Quelle interface doit offrir la couche [dao] à la couche [metier] ? Quelles sont les interactions possibles entre ces deux couches ? La couche [dao] doit fournir les données suivantes à la couche [metier] :

- le nombre de sièges à pourvoir
- la valeur du seuil électoral au-dessous duquel une liste est éliminée
- les noms des listes

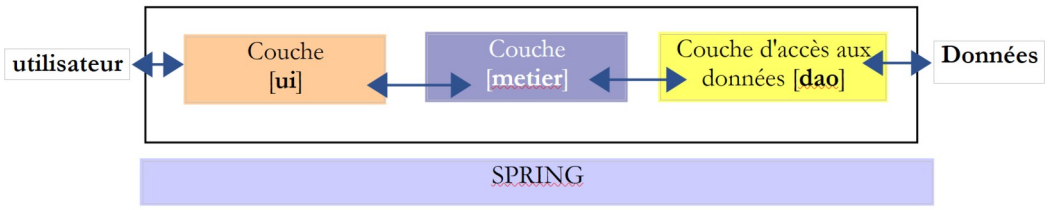
Ces informations sont en effet connues avant l'élection et peuvent donc être mémorisées. Dans le sens [metier] -> [dao], la couche [metier] peut demander à la couche [dao] d'enregistrer le résultat des élections, notamment les sièges obtenus par les différentes listes.

Avec ces informations, on pourrait tenter une première définition de l'interface de la couche [dao] :

```
1.  public interface IElectionsDao {
2.
3.    public double getSeuilElectoral();
4.
5.    public int getNbSiegesAPourvoir();
6.
7.    public ListeElectorale[] getListesElectorales();
8.
9.    public void setListesElectorales(ListeElectorale[] listesElectorales);
10. }
```

- ligne 1 : l'interface s'appelle [IElectionsDao]. Elle définit quatre méthodes :
  - trois méthodes pour lire des données venant de la source de données : [getSeuilElectoral, getNbSiegesAPourvoir, getListesElectorales]. Ces trois méthodes permettront à la couche [metier] d'obtenir les données qui caractérisent l'élection courante.
  - une méthode pour écrire des données dans la source de données : [setListesElectorales]. Cette méthode permettra à la couche [metier] de demander l'enregistrement des résultats qu'elle aura calculés.

Revenons à l'architecture trois couches de notre application :



Quelle interface la couche [metier] doit-elle présenter à la couche [ui] ? Examinons les interactions possibles entre ces deux couches.

1. la couche [ui] va avoir pour rôle de demander à l'utilisateur les voix des différentes listes en compétition. Pour cela, elle doit connaître le nombre de listes. Elle peut demander ce renseignement à la couche [metier] qui peut demander à son tour le tableau des listes en compétition à la couche [dao]. Si la couche [metier] a ce tableau, autant transférer celui-ci dans la couche [ui]. Celle-ci disposera ainsi des noms des listes et pourra affiner ses messages à l'utilisateur en demandant par exemple " Nombre de voix de la liste A ".
2. lorsque la couche [ui] aura obtenu les voix de toutes les listes, elle demandera le calcul des sièges à la couche [metier]. Celle-ci pourra faire ce calcul et rendre le résultat à la couche [ui].
3. la couche [ui] pourra alors présenter ces résultats à l'utilisateur. Celui-ci pourra également demander leur enregistrement.
4. la couche [ui] peut vouloir par ailleurs présenter des informations complémentaires à l'utilisateur, telles que le seuil électoral ou le nombre de sièges à pourvoir.

Avec ces informations, on pourrait tenter une première définition de l'interface de la couche [metier] :

```
1. public interface IElectionsMetier {
2.
3.   public ListeElectorale[] getListesElectorales();
```

```
4.
5.   public int getNbSiegesAPourvoir();
6.
7.   public double getSeuilElectoral();
8.
9.   public void recordResultats(ListeElectorale[] listesElectorales);
10.
11.    public ListeElectorale[] calculerSieges(ListeElectorale[] listesElectorales);
12.
13.  }
```

- ligne 51: l'interface s'appelle [IElectionsMetier]. Elle définit les méthodes suivantes :
  - ligne 3 : une méthode [getListesElectorales] qui permettra à la couche [ui] d'obtenir le tableau des listes en compétition ;
  - ligne 5 : la méthode [getNbSiegesAPourvoir] permet d'obtenir le nombre de sièges à pourvoir ;
  - ligne 7 : la méthode [getSeuilElectoral] permet d'obtenir le seuil électoral ;
  - ligne 11 : une méthode [calculerSieges] (ligne 36) qui permettra à la couche [ui] de demander le calcul des sièges une fois que les nombres de voix des différentes listes seront connus. Le paramètre est le tableau des listes en compétition, sans leurs sièges et sans le booléen éliminé. Le résultat rendu est ce même tableau avec cette fois les champs [sièges, elimine] initialisés ;
  - ligne 9 : une méthode [recordResultats] qui permettra à la couche [ui] de demander l'enregistrement des résultats.

Note : de par sa position, la couche [métier] reprend certaines des méthodes de la couche [DAO] pour les offrir à la couche [UI]. A cause de cette redondance, on peut être tenté de tout regrouper dans une unique couche qui regrouperait et le métier et l'accès aux données. Cette unique couche est parfois appelée le **modèle**, le M du sigle **MVC** (Modèle - Vue - Contrôleur). MVC est un modèle de conception (design pattern) répandu dans les applications web.

Examinons la signature de la méthode [calculerSieges] :

```
public ListeElectorale[] calculerSieges(ListeElectorale[] listesElectorales);
```

Il a été écrit plus haut : « Le paramètre est le tableau des listes en compétition, sans leurs sièges et sans le booléen éliminé. Le résultat est ce même tableau avec cette fois les champs [sièges, elimine] ». La signature de la méthode pourrait être également la suivante :

```
public void calculerSieges(ListeElectorale[] listesElectorales);
```

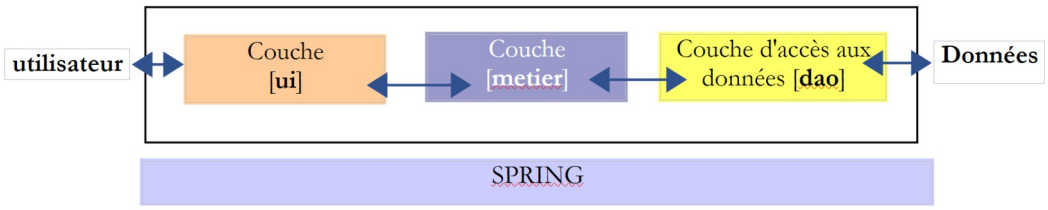
Le paramètre [listesElectorales] est une référence d'objet, ici un tableau. Chaque élément est à son tour une référence d'objet, ici un type [ListeElectorale]. La méthode [calculerSieges] va changer les champs [sieges, elimine] de chacun de ces objets. La méthode appelante détient un pointeur [listesElectorales] qui :

- avant l'appel, est la référence d'un tableau d'objets [ListeElectorale] ayant ses champs [sieges, elimine] non initialisés ;
- après l'appel, est la référence (la même) d'un tableau d'objets [ListeElectorale] ayant ses champs [sieges, elimine] initialisés ;

Alors pourquoi utiliser la signature :

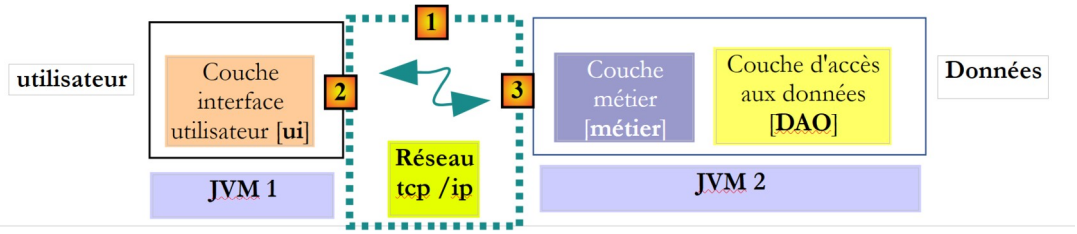
```
public ListeElectorale[] calculerSieges(ListeElectorale[] listesElectorales);
```

Lorsqu'on écrit une interface, il est bon de se rappeler qu'elle peut être utilisée dans deux contextes différents : *local* et *distant*. Dans le contexte *local*, la méthode appelante et la méthode appelée sont exécutées dans la même JVM (Java Virtual Machine) :



Si la couche [ui] fait appel à la méthode *calculerSieges* de la couche [DAO], elle a bien une référence sur le paramètre [ListeElectorale[] listesElectorales] qu'elle passe à la méthode.

Dans le contexte *distant*, la méthode appelante et la méthode appelée sont exécutées dans des JVM différentes :



Ci-dessus, la couche [ui] s'exécute dans la JVM 1 et la couche [métier] dans la JVM 2 sur deux machines différentes. Les deux couches ne communiquent pas directement. Entre-elles s'intercale une couche qu'on appellera couche de communication [1]. Celle-ci est composée d'une couche d'émission [2] et d'une couche de réception [3]. Le développeur n'a en général pas à écrire ces couches de communication. Elles sont générées automatiquement par des outils logiciels. La couche [metier] est écrite comme si elle s'exécutait dans la même JVM que la couche [DAO]. Il n'y a donc aucune modification de code.

Le mécanisme de communication entre la couche [ui] et la couche [métier] est le suivant :

- la couche [ui] fait appel à la méthode *calculerSieges* de la couche [métier] en lui passant le paramètre [ListeElectorale[] *listesElectorales1*] ;
- ce paramètre est en fait passé à la couche d'émission [2]. Celle-ci va transmettre sur le réseau, la **valeur** du paramètre *listesElectorales1* et non sa **référence**. La forme exacte de cette valeur dépend du protocole de communication utilisé ;
- la couche de réception [3] va récupérer cette valeur et reconstruire à partir d'elle un objet [ListeElectorale[] listesElectorales2] image du paramètre initial envoyé par la couche [metier]. On a maintenant deux objets identiques (au sens de contenu) dans deux JVM différentes : *listesElectorales1* et *listesElectorales2*.
- la couche de réception va passer l'objet *listesElectorales2* à la méthode *calculerSieges* de la couche [métier] ; qui va le persister en base de données. Après cette opération, la référence *listesElectorales2* pointe sur un tableau d'objets [ListeElectorale] ayant leurs champs [sieges, elimine] initialisés. . Ce n'est pas le cas de l'objet *listesElectorales1* sur lequel la couche [ui] a une référence. Si on veut que la couche [ui] ait une référence sur l'objet *listesElectorales2*, il faut lui envoyer celui-ci. Aussi est-on amenés à utiliser la signature suivante pour la méthode [calculerSieges] :

```
public ListeElectorale[] calculerSieges(ListeElectorale[] listesElectorales);
```

- avec cette signature, la méthode *calculerSieges* va rendre comme résultat la référence *listesElectorales2*. Ce résultat est rendu à la couche de réception [3] qui avait appelé la couche [métier]. Celle-ci va rendre la **valeur** (et non la **référence**) de *listesElectorales2* à la couche d'émission [2] ;
- la couche d'émission [2] va récupérer cette valeur et reconstruire à partir d'elle un objet [ListeElectorale[] *listesElectorales3*] image du résultat rendu par la méthode *calculerSieges* de la couche [métier].
- l'objet [ListeElectorale[] *listesElectorales3*] est rendu à la méthode de la couche [ui] dont l'appel à la méthode *calculerSieges* de la couche [DAO] avait initié tout ce mécanisme ;

Dans ce processus, des objets de type [ListeElectorale] vont transiter entre les couches [2] et [3] :

- lorsque la couche [2] transmet la valeur d'un objet [ListeElectorale] à la couche [3], on dit que l'objet est **sérialisé**. La forme exacte de cette sérialisation dépend du protocole de communication utilisé ;
- lorsque la couche [3] récupère la valeur d'un objet [ListeElectorale] afin de créer de nouveau un objet [ListeElectorale], on dit que l'objet est **désérialisé** ;

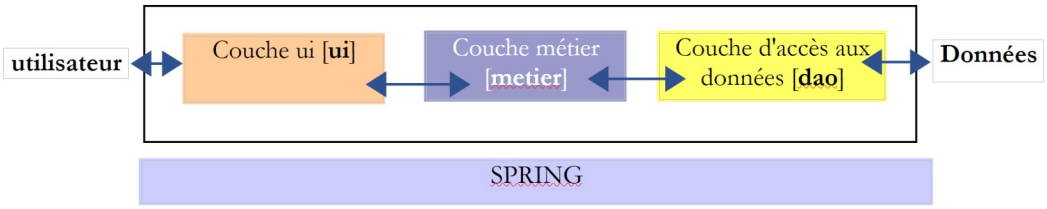
Pour qu'un objet puisse subir cette sérialisation / désérialisation, certains protocoles exigent que l'objet implémente l'interface [Serializable]. Cette interface est juste un marqueur. Il n'y a pas de méthodes à implémenter. Aussi la classe [ListeElectorale] sera-t-elle désormais déclarée de la façon suivante :

```
1. public abstract class ListeElectorale implements Serializable {
2.     private static final long serialVersionUID = 1L;
```

- le champ de la ligne 2 est imposé. On peut le conserver tel quel et l'utiliser pour toute classe de type [Serializable].

4.3 La classe d'exception

Revenons à l'interface de la couche [DAO] :




```
1.  public interface IElectionsDao {
2.
3.    public double getSeuilElectoral();
4.
5.    public int getNbSiegesAPourvoir();
6.
7.    public ListeElectorale[] getListesElectorales();
8.
9.    public void setListesElectorales(ListeElectorale[] listesElectorales);
10. }
```

Ces méthodes travaillent avec une base de données et peuvent rencontrer diverses erreurs, par exemple un SGBD non disponible. Lorsqu'on écrit une méthode, il faut toujours prévoir les cas d'erreur. Celles-ci sont signalées classiquement par une exception. Nous avons déjà rencontré la classe [ElectionsException] au paragraphe 3.3, page 22. Nous allons continuer à l'utiliser mais en l'enrichissant de la façon suivante :

```
1. package ...;
2.
3. import java.io.Serializable;
4. import java.util.ArrayList;
5. import java.util.List;
6.
7. // classe d'exception pour l'application Elections
8. // l'exception est non contrôlée
9.
10.  public class ElectionsException extends RuntimeException implements Serializable {
11.
12.    // serial ID
13.    private static final long serialVersionUID = 1L;
14.
15.    // champs locaux
16.    private int code;
17.    private List<String> erreurs;
18.
19.    // constructeurs
20.    public ElectionsException() {
21.        super();
22.    }
23.
24.    public ElectionsException(int code, Throwable e) {
25.        // parent
26.        super(e);
27.        // local
28.        this.code = code;
29.        this.erreurs = getErreursForException(e);
30.    }
31.
32.    public ElectionsException(int code, String message, Throwable e) {
33.        // parent
34.        super(message,e);
35.        // local
36.        this.code = code;
37.        this.erreurs = getErreursForException(e);
38.    }
39.
40.    public ElectionsException(int code, String message) {
41.        // parent
42.        super(message);
43.        // local
44.        this.code = code;
45.        List<String> erreurs = new ArrayList<>();
46.        erreurs.add(message);
47.        this.erreurs = erreurs;
48.    }
49.
50.    public ElectionsException(int code, List<String> erreurs) {
51.        // parent
52.        super();
53.        // local
54.        this.code = code;
```

```
55.         this.erreurs = erreurs;
56.     }
57.
58.     // liste des messages d'erreur d'une exception
59.     private List<String> getErreursForException(Throwable th) {
60.         // on récupère la liste des messages d'erreur de l'exception
61.         Throwable cause = th;
62.         List<String> erreurs = new ArrayList<>();
63.         while (cause != null) {
64.             // on récupère le message seulement s'il est !=null et non blanc
65.             String message = cause.getMessage();
66.             if (message != null) {
67.                 message = message.trim();
68.                 if (message.length() != 0) {
69.                     erreurs.add(message);
70.                 }
71.             }
72.             // cause suivante
73.             cause = cause.getCause();
74.         }
75.         return erreurs;
76.     }
77.
78.     // getters et setters
79. ...
80. }
```

- lignes 16-17 : le type [ElectionsException] encapsule :
  - un code d'erreur, ligne 16 ;
  - une liste de messages d'erreur, ligne 17 ;

La classe supporte cinq constructeurs :

- ligne 20 : **ElectionsException**()
- ligne 24 : **ElectionsException**(**int** code, Throwable e) : le second paramètre est un type [Throwable] qui est la classe parente de la classe [Exception]. Ce constructeur permet d'encapsuler l'exception **e** avec un code d'erreur. Le type [Throwable] (et donc le type Exception) permet d'encapsuler une ou plusieurs exceptions. L'idée est :
  - d'arrêter (catch) une exception qui se produit ;
  - de l'enrichir d'un message en l'encapsulant dans une nouvelle exception ;
  - de relancer la nouvelle exception ;

```
try{
...
}catch (Exception1 e1){
   throw new Exception2(«un message»,e1);
}
```

  L'encapsulation a lieu ligne 34 par l'instruction [**super**(message,e)]. Ce processus d'encapsulation peut être répété et l'exception initiale enrichie de différents messages. On dit alors qu'on a une pile d'exceptions. La méthode [**private** List<String> getErreursForException(Throwable th)] permet d'obtenir les différents messages associés aux exceptions encapsulées :
  - l'exception encapsulée est obtenue par la méthode Throwable [Throwable].getCause() ;
  - le message associé à une exception est la méthode String [Throwable].getMessage() ;
  - lignes 28-29 : on construit les champs [code, erreurs] ;
- ligne 32 : **public** ElectionsException(**int** code, String message, Throwable e) : ce constructeur est analogue au précédent, si ce n'est qu'il enrichit l'exception qu'il va encapsuler et d'un code et d'un message ;
- ligne 40 : **public** ElectionsException(**int** code, String message) : constructeur sans encapsulation d'exception ;
- ligne 50 : **public** ElectionsException(**int** code, List<String> erreurs) : constructeur sans encapsulation d'exception, ni mesage ;

La classe [ElectionsException] pourra être utilisée de la façon suivante :

```
try{
...
}catch (Exception1 e1){
   throw new ElectionsException(un_code,un_message,e1);
}
```

où le message sera ou non présent. Une fois créée, l'exception [ElectionsException] n'a pas vocation à encapsuler de nouvelles exceptions. Ci-dessus, elle encapsule l'exception *e1* et les exceptions que *e1* encapsule. Il n'y a ensuite, plus de nouvelles encapsulations.

La classe [ElectionsException] pourra être également utilisée de la façon suivante :

```
// code susceptible de rencontrer un cas d'erreur (mais pas sous la forme d'une exception)
...
if(erreur){
    throw new ElectionsException(un_code,un_message);
}
```

5 [Cours] : Introduction au framework Spring

Mots clés : architecture multicouche, Spring, injection de dépendances.

Spring est apparu en 2004 d'abord en tant que **conteneur d'objets**. Depuis, il a évolué en de multiple ramifications : Spring MVC, Spring Data, Spring Batch, ... [http://spring.io]. Nous ne présentons dans ce chapitre que le conteneur d'objets. Voici quelques points de repère :

- une application a de multiples classes et certaines d'entre-elles se partagent des objets qui doivent être uniques (singletons). Spring crée et gère ces singletons ;
- Spring place ces singletons dans une structure appelée **contexte** ;
- les classes ont accès aux singletons de l'application en les demandant à Spring via leur nom, leur type ou les deux ;
- Spring crée les singletons et gère leurs dépendances éventuelles : un singleton peut en effet avoir des références sur un ou plusieurs autres singletons. Lorsque Spring crée un singleton, il crée également leurs dépendances ;
- lorsqu'une application s'appuyant sur Spring démarre, elle peut demander à Spring de créer tous les singletons de l'application. Ceux-ci seront ensuite disponibles dans le contexte de Spring ;
- Spring facilite l'utilisation des **architectures en couches** et la **programmation par interfaces**. Dans les cas simples, chaque couche est implémentée par un singleton et implémente une interface. Si l'application travaille avec les interfaces des couches et non avec leurs classes d'implémentation, alors on obtient une architecture évolutive qui permet de changer l'implémentation d'une couche sans changer les autres grâce aux deux caractéristiques suivantes :
  - l'application obtient une référence sur la couche via son nom. Spring lui délivre une référence sur la classe implémentant la couche ;
  - l'application utilise cette référence comme celle de l'interface de la couche et non comme celle d'une classe ;

La déclaration des singletons peut être faite de trois façons qui peuvent être mixées :
- au sein d'un fichier XML,
- dans une classe spéciale de configuration ;
- avec toute classe à l'aide d'annotations ;

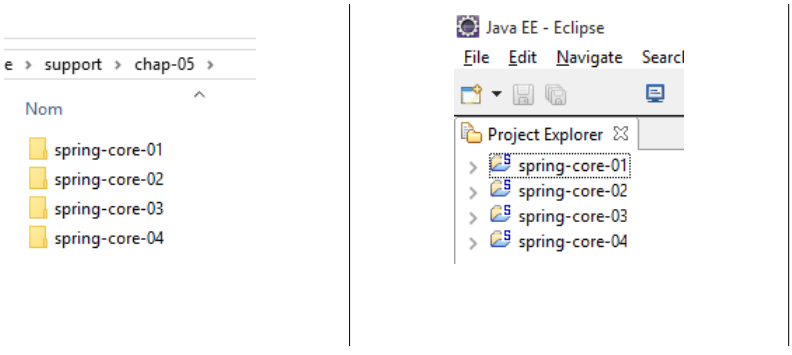
Nous présentons dans la suite trois exemples de configuration :

- [exemple-01] : configuration centralisée dans un unique fichier XML ;
- [exemple-02] : configuration centralisée dans une unique classe Java ;
- [exemple-03] : configuration distribuée sur plusieurs classes Java ;

Le dernier exemple [exemple-04] s'intéresse à la configuration Spring d'une architecture en couches. C'est l'exemple le plus important. C'est lui qui sera repris constamment pour configurer les architectures du document.

Ces quatre exemples posent les bases de ce qui suit :
- configuration Spring et injection de dépendances ;
- utilisation de Maven pour gérer les dépendances d'un projet ;
- utilisation de JUnit pour tester les projets ;

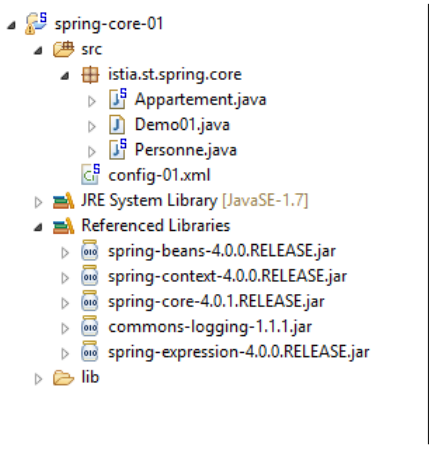
5.1 Support



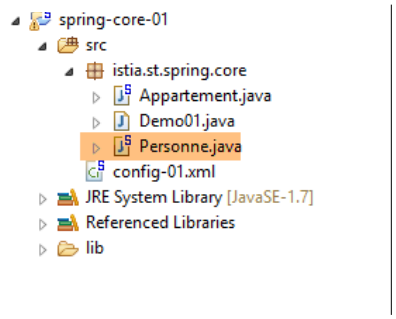
Le dossier [support / chap-05] contient les projets Eclipse de ce chapitre.

5.2 Exemple-01

5.2.1 Le projet Eclipse



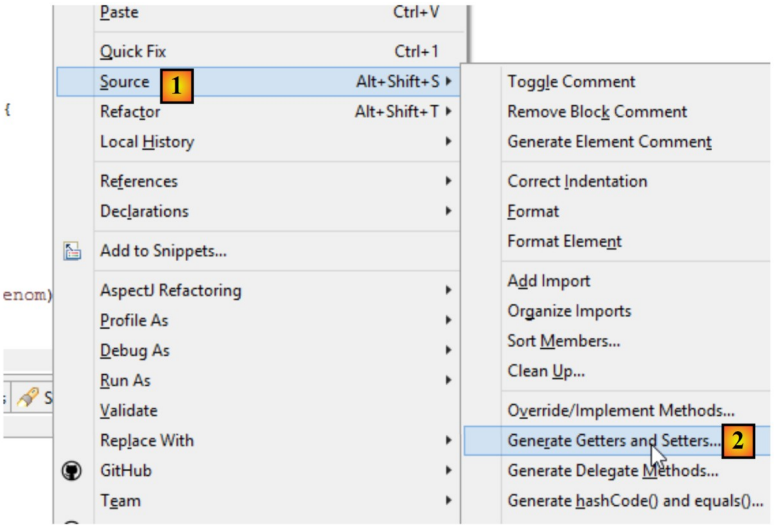
5.2.2 La classe [Personne]



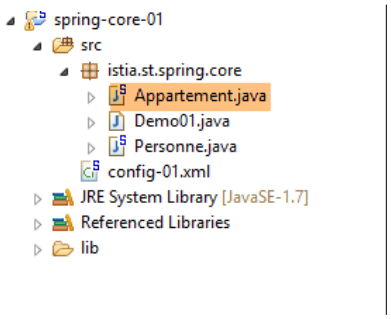

```
1. package istia.st.spring.core;
2.
3. public class Personne {
4.
5.   // champs
6.   private String nom;
7.   private String prenom;
8.   private int age;
9.
10.   // constructeurs
11.   public Personne() {
12.
13.   }
14.
15.   public Personne(String nom, String prénom, int âge) {
16.       this.nom = nom;
17.       this.prenom = prénom;
18.       this.age = âge;
19.   }
20.
21.   // toString
22.   public String toString() {
23.       return String.format("Personne[%s, %s,%d]", prenom, nom, age);
24.   }
25.
26.   // getters et setters
27.
28.   public String getNom() {
29.       return nom;
30.   }
31.
32.   public void setNom(String nom) {
33.       this.nom = nom;
34.   }
35.
36.   public String getPrenom() {
```

```
37.         return prenom;
38.     }
39.
40.     public void setPrenom(String prenom) {
41.         this.prenom = prenom;
42.     }
43.
44.     public int getAge() {
45.         return age;
46.     }
47.
48.     public void setAge(int age) {
49.         this.age = age;
50.     }
51.
52. }
```

Note : les getters et setters peuvent être générés automatiquement de la façon suivante [1-2] :



5.2.3 La classe [Appartement]



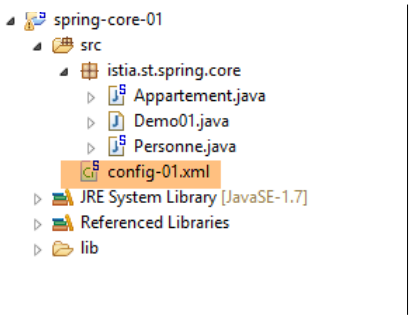

```
1. package istia.st.spring.core;
2.
3. public class Appartement {
4.
5.   // champs
6.   private Personne proprietaire;
7.   private int surface;
8.
9.   // getters et setters
10.
11.    public Personne getProprietaire() {
12.        return proprietaire;
13.    }
14.
15.    public void setProprietaire(Personne proprietaire) {
16.        this.proprietaire = proprietaire;
17.    }
18.
```

```
19.    public int getSurface() {
20.        return surface;
21.    }
22.
23.    public void setSurface(int surface) {
24.        this.surface = surface;
25.    }
26.
27.    // toString
28.    public String toString() {
29.        return String.format("Appartement[%s, %s]", proprietaire, surface);
30.    }
31.
32. }
```

Note : cette classe n'a pas de constructeur explicite. Dans ce cas, existe toujours par défaut, le constructeur sans paramètres qui ne fait rien. Lorsqu'on crée des constructeurs, ce constructeur par défaut n'existe plus implicitement. Il faut alors le définir **explicitement** :

```
public Appartement(){
}
```

5.2.4 Le fichier de configuration de Spring

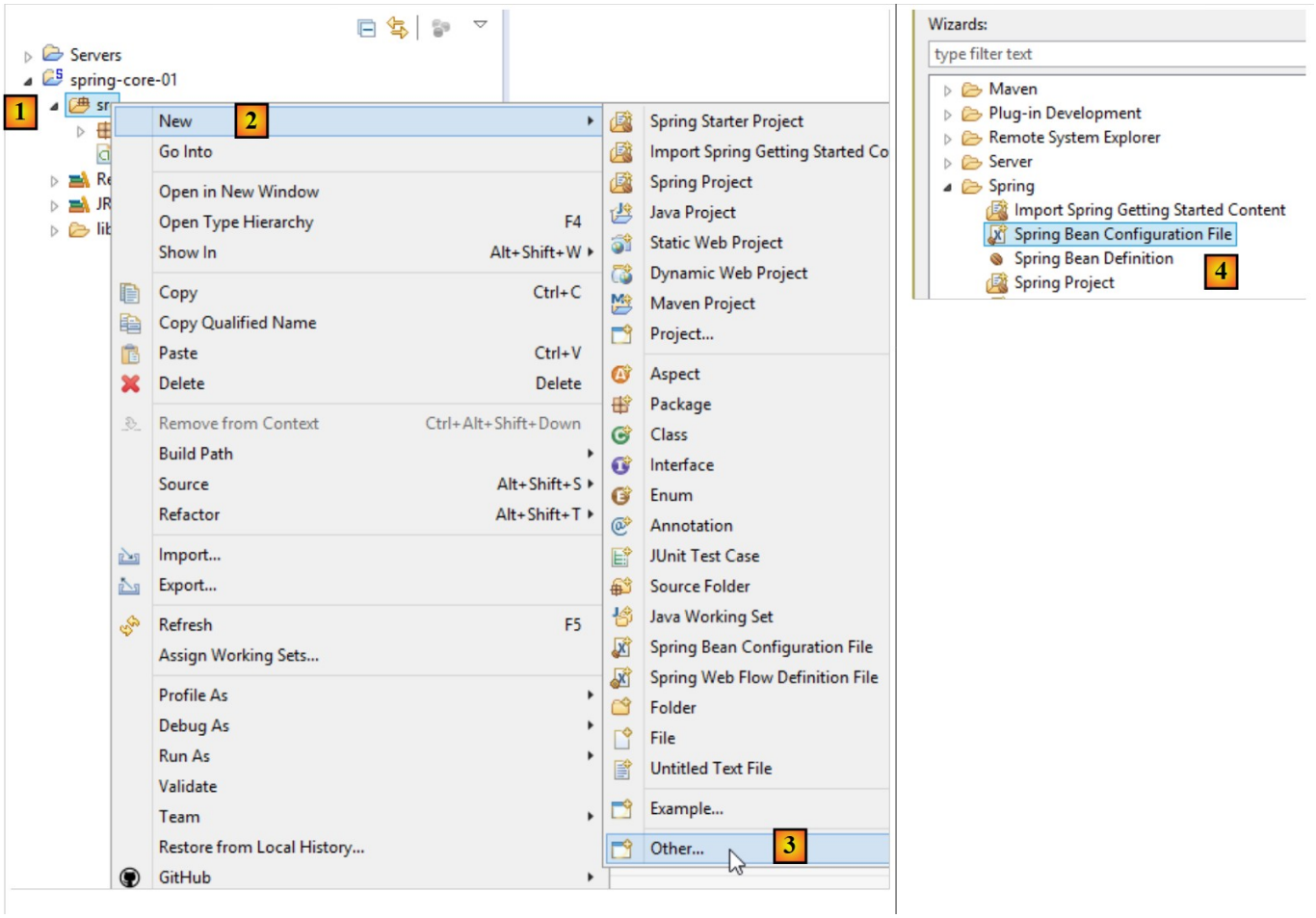



```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans xmlns="http://www.springframework.org/schema/beans" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
   instance" xmlns:util="http://www.springframework.org/schema/util"
3.  xsi:schemaLocation="http://www.springframework.org/schema/beans
   http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
4.    http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-
   util-4.0.xsd">
5.  <!-- Personne 01 -->
6.  <bean id="personne_01" class="istia.st.spring.core.Personne">
7.    <constructor-arg index="0" value="dubois" />
8.    <constructor-arg index="1" value="paul" />
9.    <constructor-arg index="2" value="34" />
10.   </bean>
11.   <!-- Personne 02 -->
12.   <bean id="personne_02" class="istia.st.spring.core.Personne">
13.      <property name="nom" value="martin" />
14.      <property name="prenom" value="micheline" />
15.      <property name="age" value="18" />
16.   </bean>
17.   <!-- une liste de personnes -->
18.   <util:list id="club">
19.      <ref bean="personne_01" />
20.      <ref bean="personne_02" />
21.   </util:list>
22.   <!-- un appartement -->
23.   <bean id="appartement" class="istia.st.spring.core.Appartement">
24.      <property name="surface" value="100" />
25.      <property name="proprietaire" ref="personne_01" />
26.   </bean>
27. </beans>
```

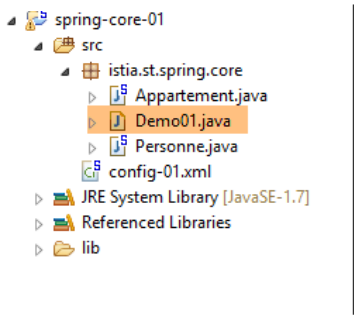
- lignes 2, 27 : les singletons sont définis au sein d'une balise <**beans**> ;
- lignes 6-10 : chaque singleton est défini par une balise <**bean**> ;
- ligne 6 : [id] est l'identifiant du singleton. [class] est le nom complet de la classe à instancier ;
- lignes 7-9 : les trois valeurs à passer au constructeur de la classe [Personne] ;

- lignes 12-16 : la classe [Personne] est d'abord créée avec son constructeur par défaut [new Personne()]. Puis pour chaque balise [property], un setter de la classe est utilisé. Par exemple pour la ligne 13, la méthode [**setNom**("martin")] va être exécutée. Il faut donc que la méthode [**setNom**] existe. C'est un point important à se rappeler ;
- lignes 18-21 : la balise <util:list> permet de définir un singleton qui est une liste ;
- ligne 19 : désigne le singleton [personne_01] défini ligne 6. On a là ce qu'on appelle une **injection de dépendances**. Deux attributs sont utilisables pour initialiser le champ d'un singleton :
  - [value] : pour affecter au champ une valeur primitive (chaîne, nombre, date, ...),
  - [ref] : pour affecter au champ la référence d'un objet Spring ;

Note : le fichier de configuration Spring peut être généré de la façon suivante [1-4] :



5.2.5 La classe exécutable

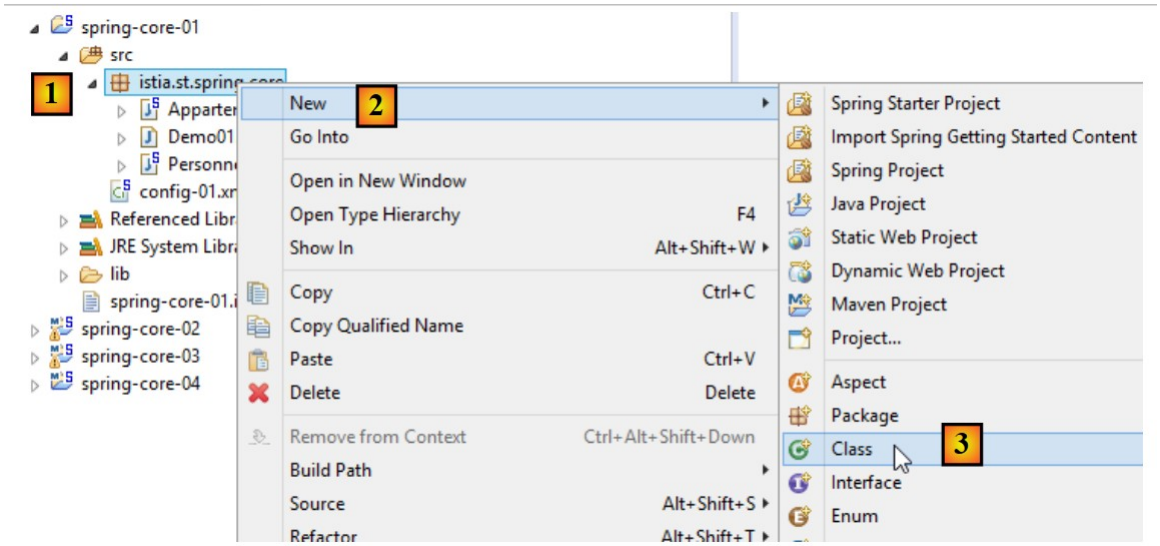


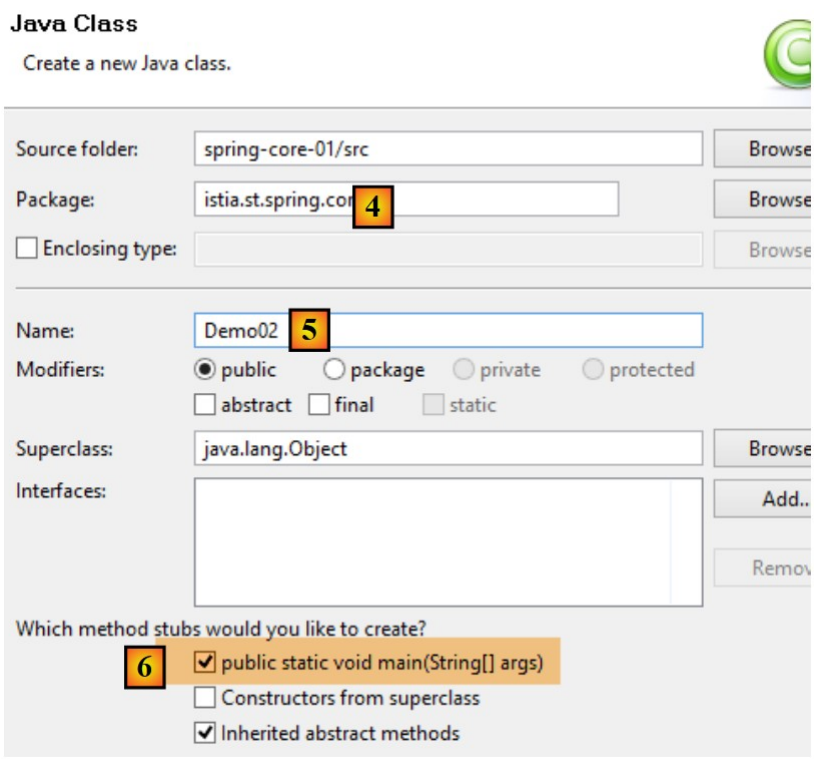

```
1. package istia.st.spring.core;
2.
3. import java.util.ArrayList;
```

```
4. import java.util.List;
5.
6. import org.springframework.context.ApplicationContext;
7. import org.springframework.context.support.ClassPathXmlApplicationContext;
8.
9. public class Demo01 {
10.
11.    @SuppressWarnings({ "unchecked", "resource" })
12.    public static void main(String[] args) {
13.        // récupération du contexte Spring
14.        ApplicationContext ctx = new ClassPathXmlApplicationContext("config-01.xml");
15.        // on récupère les beans
16.        Personne p01 = ctx.getBean("personne_01", Personne.class);
17.        Personne p02 = ctx.getBean("personne_02", Personne.class);
18.        List<Personne> club = ctx.getBean("club", new ArrayList<Personne>().getClass());
19.        Appartement appart01 = ctx.getBean(Appartement.class);
20.        // on les affiche
21.        System.out.println("personnes--------");
22.        System.out.println(p01);
23.        System.out.println(p02);
24.        System.out.println("club--------");
25.        for (Personne p : club) {
26.            System.out.println(p);
27.        }
28.        System.out.println("appartement--------");
29.        System.out.println(appart01);
30.        // les beans récupérés sont des singletons
31.        // on peut les demander plusieurs fois, on récupère toujours le même bean
32.        Personne p01b = ctx.getBean("personne_01", Personne.class);
33.        System.out.println(String.format("beans [p01,p01b] identiques ? %s", p01b == p01));
34.    }
35. }
```

- ligne 14 : crée le contexte Spring. Tous les singletons définis dans le fichier [config-01.xml] sont alors instanciés ;
- ligne 16 : demande une référence sur le singleton identifié par [personne_01] de type [Personne]. Ce second paramètre est facultatif mais alors on reçoit une référence sur un type [Object], référence qu'il faut alors transtyper vers le type [Personne] ;
- ligne 19 : on n'utilise pas le nom du bean mais uniquement son type car il n'y a qu'un singleton de type [Appartement] ;
- ligne 18 : on a utilisé à la fois l'identifiant et le type du singleton désiré. L'identifiant est superflu puisque qu'il n'y a qu'un singleton de type [**new** ArrayList<Personne>().getClass()] ;
- lignes 32-33 : montrent que si on demande plusieurs fois le même singleton, on obtient bien toujours la même référence, montrant par là qu'on a bien affaire à un singleton. Ce point est important à comprendre ;

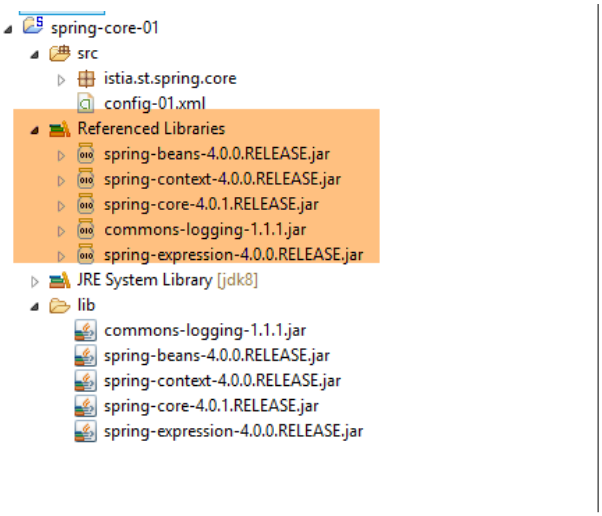
Note : une classe exécutable peut être générée de la façon suivante [1-6] :





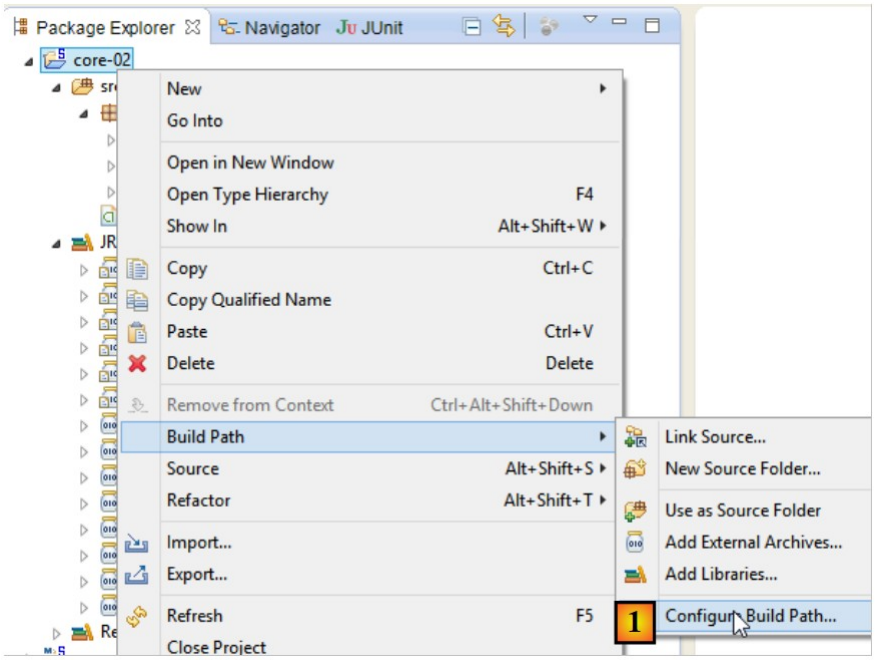
- c'est le fait de cocher [6] qui va faire que la classe générée va contenir une méthode statique [main] qui va la rendre exécutable ;

5.2.6 Les dépendances du projet

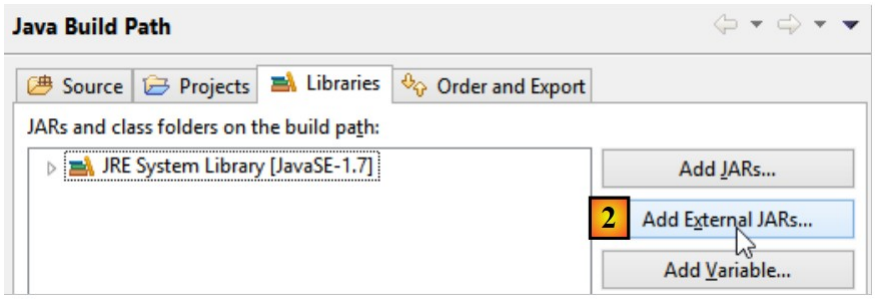


- dépendances Spring : [spring-core, spring-beans, spring-context, spring-expression, commons-logging] ;

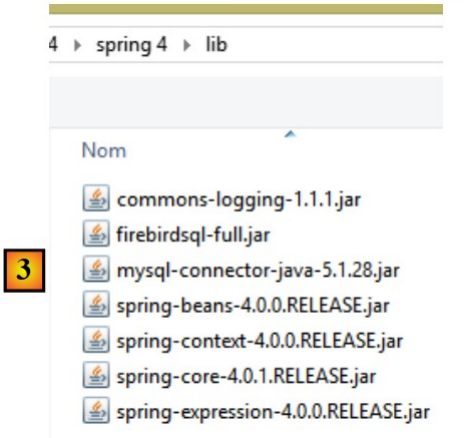
Les dépendances sont ajoutées au projet de la façon suivante :



- en [1] : clic droit sur le projet / [Build Path] / [Configure Build Path] ;



- en [2] : [Add JARs] si les JARs à ajouter sont dans un dossier du projet. Sinon [Add External JARs] ;



- en [3], sélectionnez les JARs à ajouter au ClassPath du projet (ils sont ici dans le dossier [lib] à l'intérieur du projet) ;

Définition : le [ClassPath] d'un projet est l'ensemble des dossiers explorés par la JVM (Java Virtual Machine) qui exécute le projet, pour trouver une classe référencée par celui-ci. Pour un projet Eclipse, le [ClassPath] est formé des éléments suivants :

- le dossier [bin] du projet ;
- les éléments du [Build Path] du projet ;

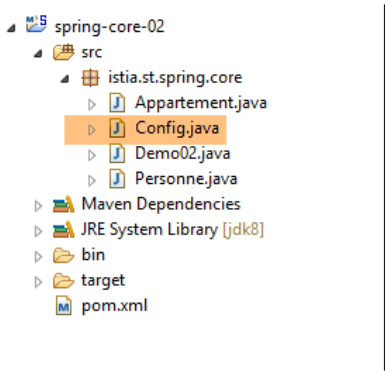
Le dossier [bin] est le dossier produit par la compilation du dossier [src]. Donc tout ce qui est placé dans le dossier [src] fait automatiquement partie du [ClassPath] (même si ce n'est pas un fichier .java). Donc dans le projet précédent, le fichier de configuration de spring [config-01.xml] qui est dans le dossier [src] fera partie du [Classpath] du projet à l'exécution.

5.2.7 Les résultats

```
1. févr. 21, 2014 1:16:23 PM org.springframework.context.support.AbstractApplicationContext prepareRefresh
2. Infos: Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@3ac67f69: startup date [Fri Feb 21
    13:16:23 CET 2014]; root of context hierarchy
3. févr. 21, 2014 1:16:23 PM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions
4. Infos: Loading XML bean definitions from class path resource [config-01.xml]
5. personnes--------
6. Personne[paul, dubois,34]
7. Personne[micheline, martin,18]
8. club--------
9. Personne[paul, dubois,34]
10.  Personne[micheline, martin,18]
11.  appartement--------
12.  Appartement[Personne[paul, dubois,34], 100]
13.  beans [p01,p01b] identiques ? true
```

5.3 Exemple-02

5.3.1 Le projet Eclipse



5.3.2 La classe de configuration de Spring

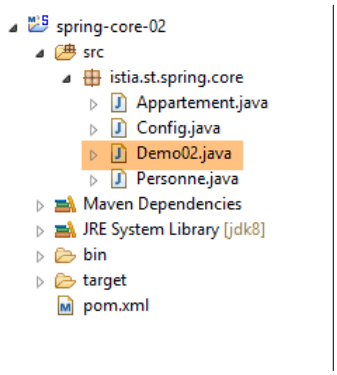
```
1. package istia.st.spring.core;
2.
3. import java.util.ArrayList;
4. import java.util.List;
5.
6. import org.springframework.context.annotation.Bean;
7. import org.springframework.context.annotation.Configuration;
8.
9. @Configuration
10.  public class Config {
11.
12.    @Bean
13.    public Personne personne_01() {
14.        return new Personne("Paul", "Dubois", 34);
15.    }
16.
17.    @Bean
18.    public Personne personne_02() {
19.        return new Personne("Martin", "Micheline", 18);
20.    }
21.
22.    @Bean
```

```
23.    public List<Personne> club(Personne personne_01, Personne personne_02) {
24.        List<Personne> personnes = new ArrayList<Personne>();
25.        personnes.add(personne_01);
26.        personnes.add(personne_02);
27.        return personnes;
28.    }
29.
30.    @Bean
31.    public Appartement appartement(Personne personne_01) {
32.        Appartement appartement = new Appartement();
33.        appartement.setSurface(200);
34.        appartement.setPropriétaire(personne_01);
35.        return appartement;
36.    }
37. }
```

- ligne 9 : l'annotation [@Configuration] est une annotation Spring. Elle indique que la classe annotée définit des singletons. Ceux-ci sont définis à l'aide de l'annotation [@Bean]. Spring va exécuter toutes les méthodes annotées par [@Bean]. Celles-ci créent les singletons de l'application ;
- lignes 12-15 : définit un singleton identifié par [personne_01], ç-à-d le nom de la méthode.
- ligne 23 : les paramètres [personne_01, personne_02] portent les noms de singletons. Spring va automatiquement les initialiser avec les références de ces singletons. On parle d'injection de paramètres ;

Cette façon de configurer les singletons est plus explicite que celle qui utilise le fichier XML. En fait, on reproduit nous-mêmes ce que faisait Spring implicitement à partir du fichier XML.

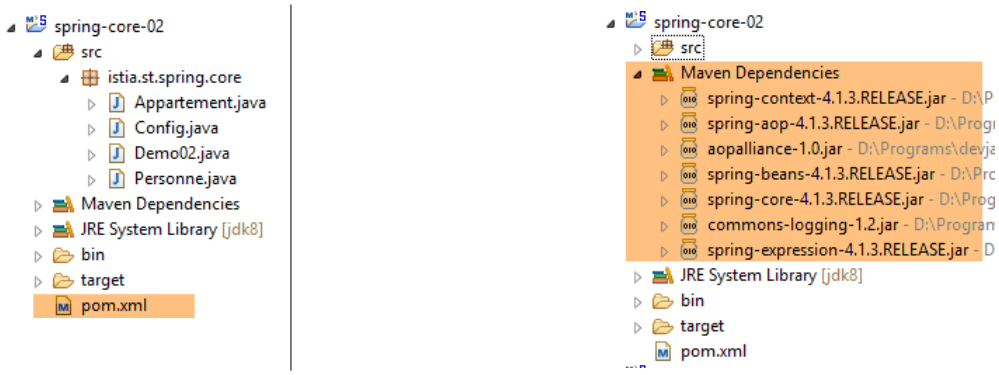
5.3.3 La classe exécutable




```
1. package istia.st.spring.core;
2.
3. import java.util.ArrayList;
4. import java.util.List;
5.
6. import org.springframework.context.annotation.AnnotationConfigApplicationContext;
7.
8. public class Demo02 {
9.
10.    @SuppressWarnings({ "unchecked", "resource" })
11.    public static void main(String[] args) {
12.        // récupération du contexte Spring
13.        AnnotationConfigApplicationContext ctx = new AnnotationConfigApplicationContext(Config.class);
14.        // on récupère les beans
15.        Personne p01 = ctx.getBean("personne_01", Personne.class);
16.        Personne p02 = ctx.getBean("personne_02", Personne.class);
17.        List<Personne> club = ctx.getBean("club", new ArrayList<Personne>().getClass());
18.        Appartement appart01 = ctx.getBean(Appartement.class);
19.        // on les affiche
20.        System.out.println("personnes--------");
21.        System.out.println(p01);
22.        System.out.println(p02);
23.        System.out.println("club--------");
24.        for (Personne p : club) {
25.            System.out.println(p);
26.        }
27.        System.out.println("appartement--------");
28.        System.out.println(appart01);
29.        // les beans récupérés sont des singletons
30.        // on peut les demander plusieurs fois, on récupère toujours le même bean
31.        Personne p01b = ctx.getBean("personne_01", Personne.class);
32.        System.out.println(String.format("beans [p01,p01b] identiques ? %s", p01b == p01));
33.    }
34. }
```

- la ligne 13 provoque l'instanciation de tous les beans définis dans la classe [Config] ;
- le reste du code ne change pas ;

5.3.4 Les dépendances du projet



Les dépendances sont fixées par le fichier [pom.xml] suivant :

```
1. <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2.   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
3.   <modelVersion>4.0.0</modelVersion>
4.   <groupId>istia.st.spring.core</groupId>
5.   <artifactId>spring-core-02</artifactId>
6.   <version>0.0.1-SNAPSHOT</version>
7.   <name>spring-core-02</name>
8.   <description>Introduction à Spring</description>
9.
10.    <properties>
11.       <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
12.       <java.version>1.7</java.version>
13.    </properties>
14.
15.    <dependencies>
16.       <!-- Spring -->
17.       <dependency>
18.          <groupId>org.springframework</groupId>
19.          <artifactId>spring-context</artifactId>
20.          <version>4.1.3.RELEASE</version>
21.       </dependency>
22.    </dependencies>
23.    <!-- plugins -->
24.    <build>
25.       <plugins>
26.          <plugin>
27.             <artifactId>maven-assembly-plugin</artifactId>
28.             <configuration>
29.                <descriptorRefs>
30.                   <descriptorRef>jar-with-dependencies</descriptorRef>
31.                </descriptorRefs>
32.             </configuration>
33.          </plugin>
34.          <plugin>
35.             <groupId>org.apache.maven.plugins</groupId>
36.             <artifactId>maven-surefire-plugin</artifactId>
37.             <version>2.18.1</version>
38.          </plugin>
39.       </plugins>
40.    </build>
41.
42. </project>
```

La gestion à la main des dépendances d'un projet devient un casse-tête lorsqu'on utilise des bibliothèques Java dont on ne connaît pas les dépendances. Ainsi le framework [Hibernate] qui gère l'accès aux bases de données a des dizaines de dépendances. Le projet [Maven] résoud ce problème. On nomme la dépendance dont on a besoin. Celle-ci est cherchée automatiquement dans des dépôts Maven répartis sur le net. Si la dépendance demandée a elle-même des dépendances, alors celles-ci sont automatiquement téléchargées également. Ces dépendances téléchargées sont stockées dans un dépôt local au poste. Si une autre application a besoin

plus tard de la même dépendance, celle-ci ne sera pas téléchargée mais cherchée dans le dépôt local. Une dépendance est caractérisée par les éléments suivants :

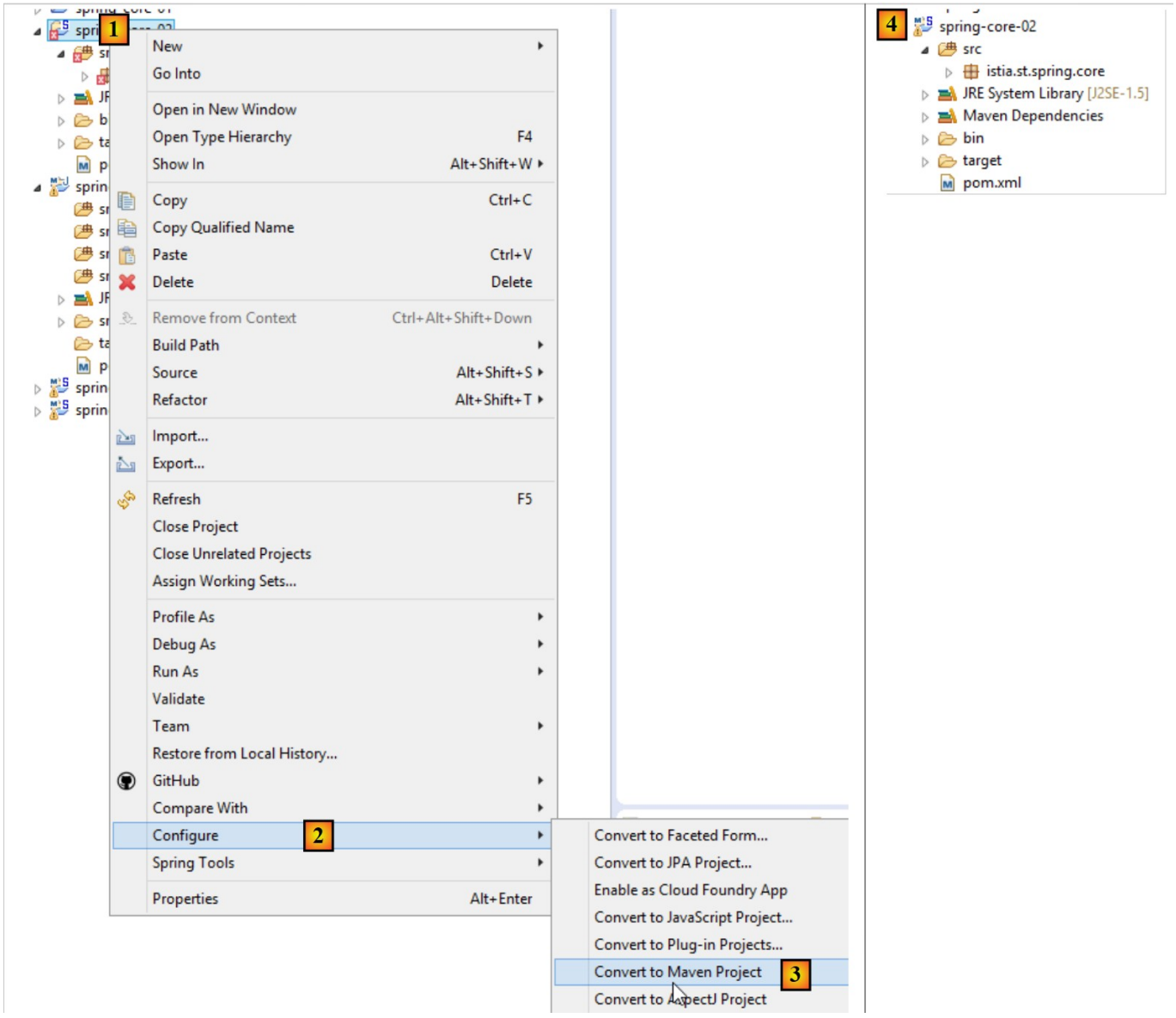
- ligne 17 : une balise <dependency> ;
- ligne 18 : un attribut [groupId] qui identifie en général l'entreprise qui a créé la dépendance ;
- ligne 19 : un attribut [artifactId] qui identifie la dépendance ;
- ligne 20 : un attribut [version] qui identifie la version désirée ;

La génération du projet va elle-même produire un composant Maven défini par les lignes 4-8 :

- lignes 4-6 : les attributs [groupId, artifactId,version] que nous venons de décrire ;
- lignes 7-8 : sont des attributs facultatifs ;

Nous allons revenir un peu plus loin sur le rôle des lignes 24-40. Pour transformer un projet Eclipse ordinaire en un projet Maven, il faut faire deux choses :

- créer le fichier [pom.xml] précédent ;
- déclarer que le projet est désormais un projet Maven [1-4] :



L'icône d'un projet **M**aven a un **M** [4]. Le **S** indique que le projet a des éléments **S**pring. Il est déconseillé de transformer (comme nous venons de le faire) un projet Eclipse en projet Maven car alors le projet n'a pas la structure attendue pour un projet Maven ce qui peut amener parfois des problèmes inattendus.

5.3.5 Génération de l'artifact Maven du projet

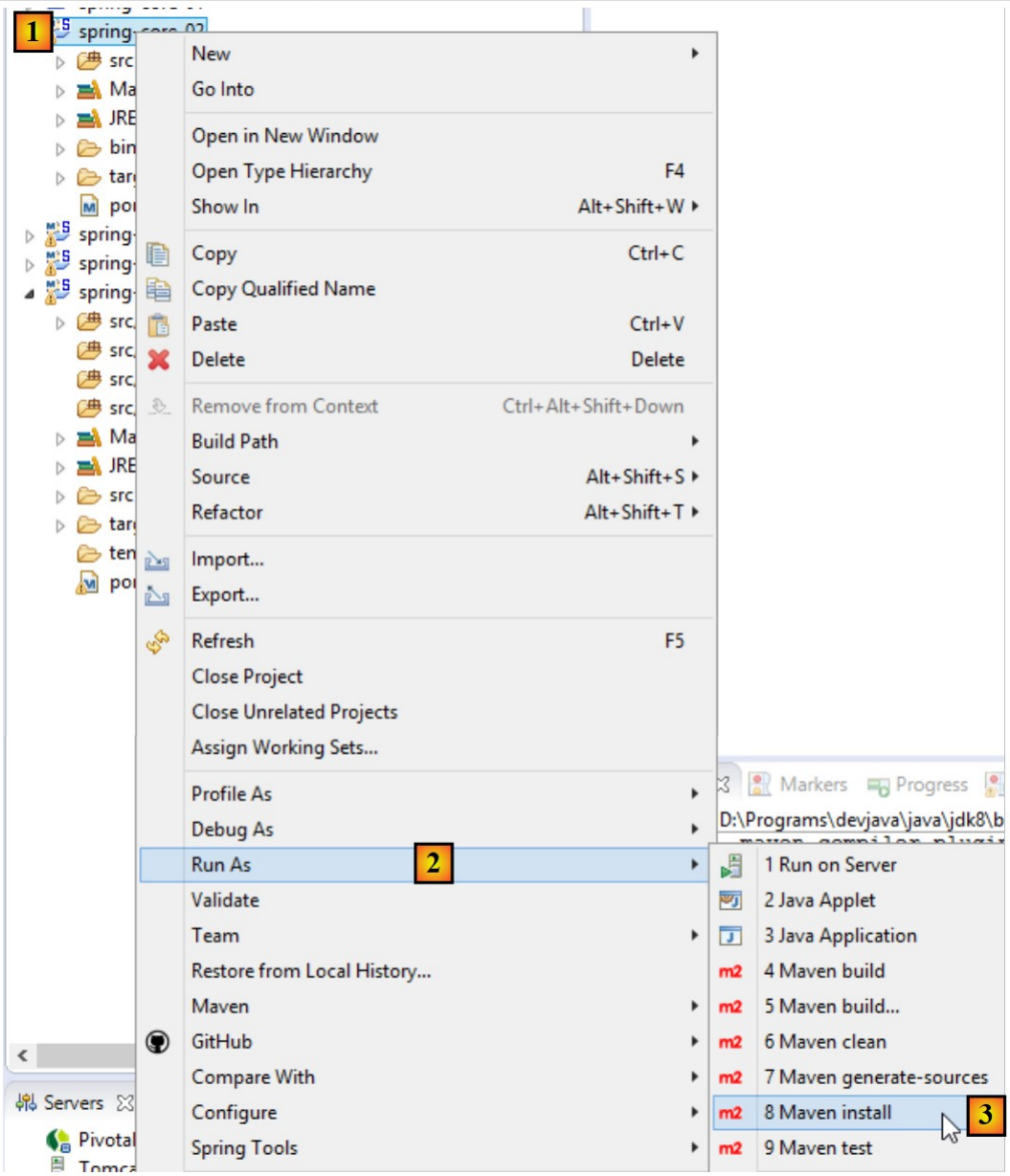
Nous appelons artifact Maven du projet, l'élément défini par les lignes 4-6 du fichier [pom.xml] :

```
1.   <groupId>istia.st.spring.core</groupId>
2.   <artifactId>spring-core-02</artifactId>
3. <version>0.0.1-SNAPSHOT</version>
```

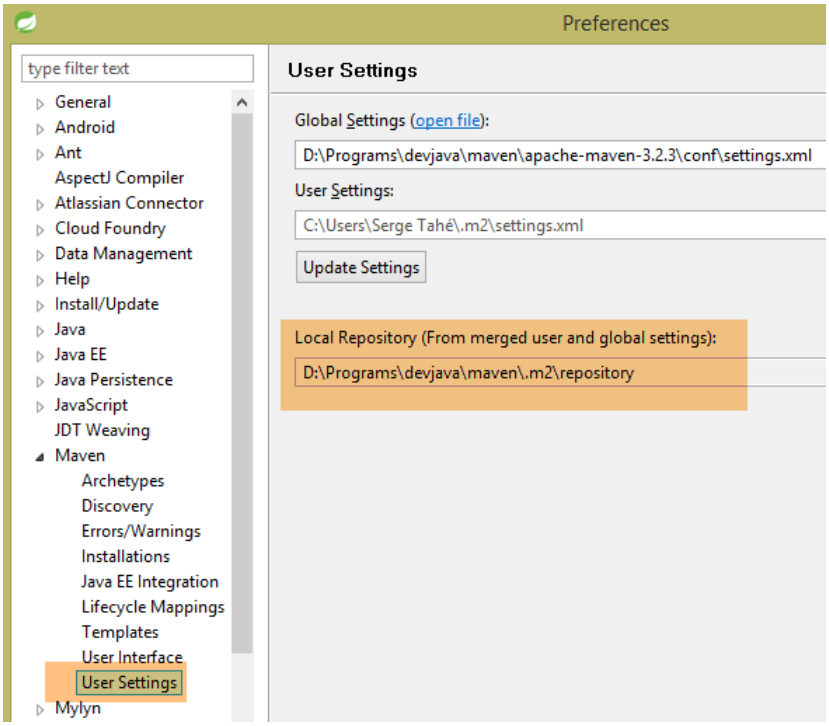
Pour générer cet artifact, il faut que les lignes 3-7 suivantes soient présentes dans le fichier [pom.xml] :

```
1.   <build>
2.     <plugins>
3.         <plugin>
4.             <groupId>org.apache.maven.plugins</groupId>
5.             <artifactId>maven-surefire-plugin</artifactId>
6.             <version>2.18.1</version>
7.         </plugin>
8.     </plugins>
9. </build>
```

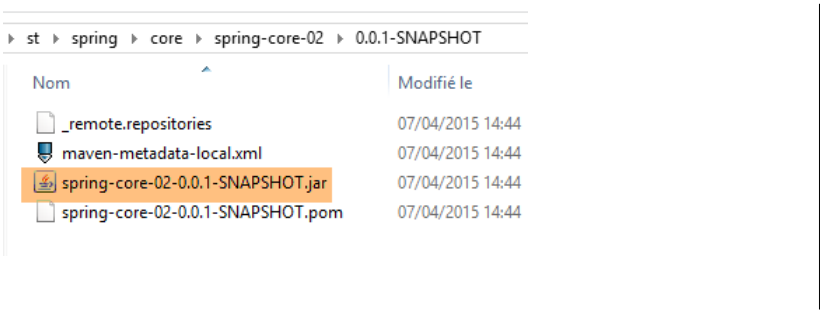
Elles définissent le plugin de Maven capable de générer l'artifact du projet. On procéde ensuite de la façon suivante :



L'artifact ainsi généré va dans le dépôt Maven local. La localisation de celui-ci peut être trouvée dans la configuration d'Eclipse :



Il est alors possible de vérifier la bonne installation de l'artifact Maven :

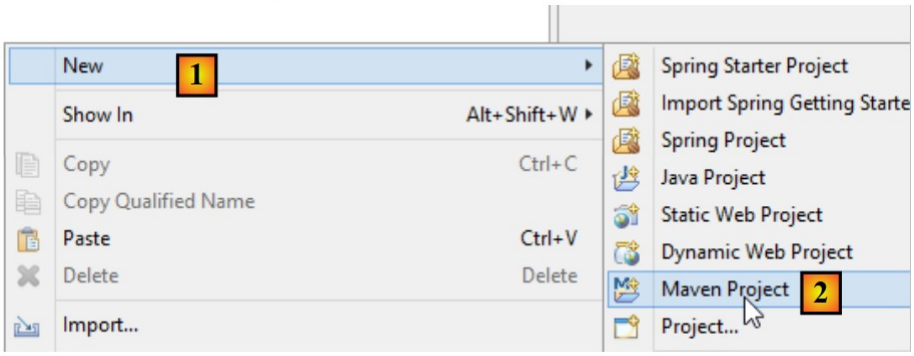


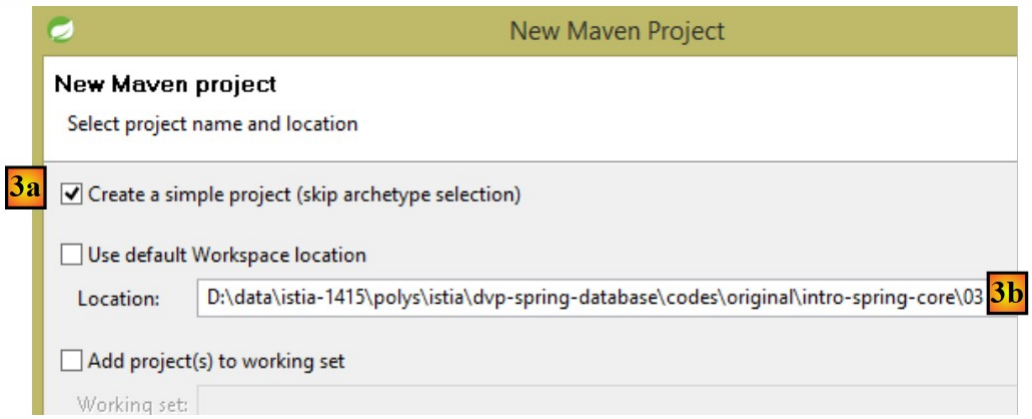
Désormais, un autre projet Maven local pourra utiliser cette archive.

5.4 Exemple-03

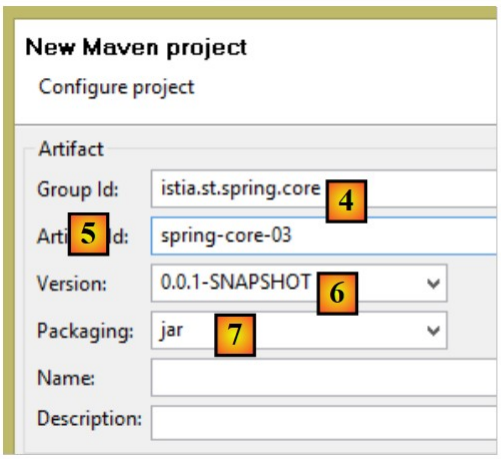
5.4.1 Le projet Eclipse

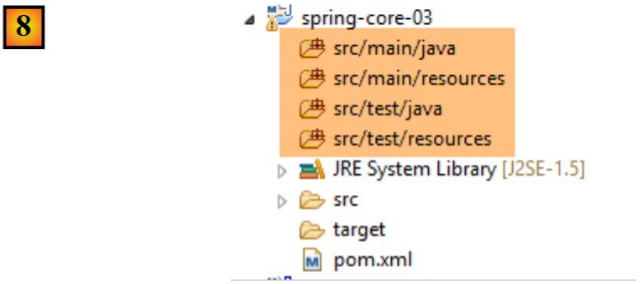
Nous créons cette fois un projet Maven [1-8] :





- en [3b] : désignez un dossier vide où sera généré le projet ;





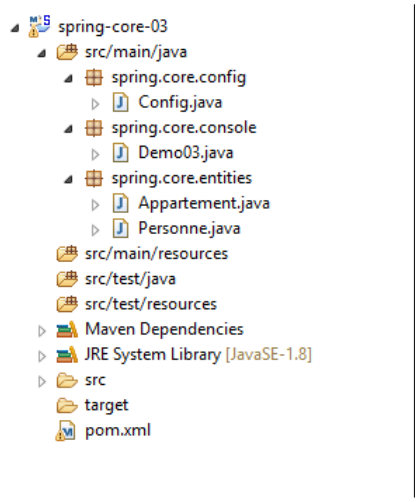
- en [4] : l'identifiant du groupe Maven auquel appartiendra le projet ;
- en [5] : le nom de l'artifact Maven produit :
- en [6] : sa version ;
- en [7]: son mode de packaging (il existe également war, ear, apk, ...) ;
- en [8] : le projet ainsi créé ;

Un projet Maven a par défaut, une arborescence précise :

- [src / main / java] : les codes source du projet. Les produits compilés de ces sources iront dans le dossier [target/classes] du projet ;

- [src / main / resources] : les ressources qui doivent être dans le Classpath du projet sans pour autant être des sources Java. Elles seront recopiées telles-quelles dans le dossier [target/classes] du projet ;
- [src / test / java] : les codes source des tests du projet. Les produits compilés de ces sources iront dans le dossier [target/test-classes] du projet. Ces éléments ne sont pas intégrés dans l'archive Maven du projet ;
- [src / test / resources] : les ressources qui doivent être dans le Classpath du projet pour les tests sans pour autant être des sources Java. Elles seront recopiées telles-quelles dans le dossier [target/test-classes] du projet ;

Nous complétons le projet de la façon suivante :



5.4.2 La configuration Maven

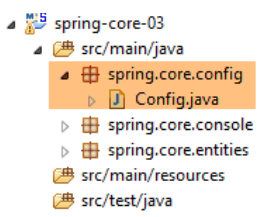
Un fichier [pom.xml] est généré par défaut. Nous le transformons de la façon suivante :

```
1. <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2.   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
3.   <modelVersion>4.0.0</modelVersion>
4.   <groupId>istia.st.spring.core</groupId>
5.   <artifactId>spring-core-03</artifactId>
6.   <version>0.0.1-SNAPSHOT</version>
7.   <name>spring-core-03</name>
8.   <description>Introduction à Spring</description>
9.
10.    <properties>
11.        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
12.        <java.version>1.8</java.version>
13.    </properties>
14.
15.    <!-- projet Maven parent -->
16.    <parent>
17.        <groupId>org.springframework.boot</groupId>
18.        <artifactId>spring-boot-starter-parent</artifactId>
19.        <version>1.2.3.RELEASE</version>
20.    </parent>
21.
22.    <dependencies>
23.        <!-- Spring Context -->
24.        <dependency>
25.            <groupId>org.springframework</groupId>
26.            <artifactId>spring-context</artifactId>
27.        </dependency>
28.        <!-- logs -->
29.        <dependency>
30.            <groupId>org.springframework.boot</groupId>
31.            <artifactId>spring-boot-starter-logging</artifactId>
32.        </dependency>
33.
34.    </dependencies>
35.
36.    <!-- plugins -->
37.    <build>
38.        <plugins>
39.            <!-- pour la génération de l'archive du projet avec ses dépendances -->
40.            <plugin>
41.                <artifactId>maven-assembly-plugin</artifactId>
42.                <configuration>
```

```
43.                 <descriptorRefs>
44.                     <descriptorRef>jar-with-dependencies</descriptorRef>
45.                 </descriptorRefs>
46.             </configuration>
47.         </plugin>
48.         <!-- pour l'installation de l'artifact du projet dans le dépôt local Maven -->
49.         <plugin>
50.             <groupId>org.apache.maven.plugins</groupId>
51.             <artifactId>maven-surefire-plugin</artifactId>
52.             <version>2.18.1</version>
53.         </plugin>
54.     </plugins>
55.   </build>
56.
57. </project>
```

- ligne 11 : le projet est codé en UTF-8 ;
- ligne 12 : on utilise un JDK 1.8 pour compiler le projet ;
- lignes 16-20 : pour les projets utilisant les bibliothèques Spring, il est pratique d'utiliser un projet Maven parent appelé [spring-boot-starter-parent]. Celui-ci définit les versions de différentes bibliothèques Spring ainsi que celles de leurs dépendances. Ceci permet de ne plus les définir dans la définition des dépendances. Ainsi lignes 24-27 on ne précise pas la version de [spring-context] désirée. Ce sera celle définie par le projet parent [spring-boot-starter-parent]. Cette technique permet de ne pas se soucier d'éventuelles incompatibilités de versions entre dépendances. Celles définies par le projet parent sont compatibles entre-elles ;
- lignes 29-32 : Spring écrit un nombre important d'informations sur la console au travers d'une bibliothèque de logs. Celle-ci est importée ici ;
- lignes 40-47 : un plugin Maven sur lequel nous allons revenir ;
- lignes 50-52 : le plugin de génération de l'artifact Maven du projet ;

5.4.3 La classe de configuration de Spring



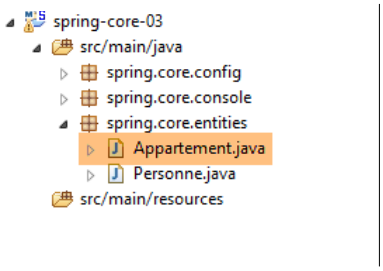
La classe [Config] est la suivante :

```
1. package istia.st.spring.core.config;
2.
3. import istia.st.spring.core.entities.Personne;
4.
5. import java.util.ArrayList;
6. import java.util.List;
7.
8. import org.springframework.context.annotation.Bean;
9. import org.springframework.context.annotation.ComponentScan;
10.  import org.springframework.context.annotation.Configuration;
11.
12.  @Configuration
13.  @ComponentScan({ "spring.core.entities" })
14.  public class Config {
15.
16.    @Bean
17.    public Personne personne_01() {
18.        return new Personne("Paul", "Dubois", 34);
19.    }
20.
21.    @Bean
22.    public Personne personne_02() {
23.        return new Personne("Martin", "Micheline", 18);
24.    }
25.
26.    @Bean
27.    public List<Personne> club(Personne personne_01, Personne personne_02) {
28.        List<Personne> personnes = new ArrayList<Personne>();
29.        personnes.add(personne_01);
30.        personnes.add(personne_02);
31.        return personnes;
32.    }
```

```
33.
34.    @Bean
35.    public int mySurface() {
36.        return 200;
37.    }
38. }
```

- on retrouve là un code déjà commenté avec deux nouveautés :
  - ligne 13 : indique qu'il y a d'autres beans à instancier dans le package [spring.core.entities],
  - lignes 34-37 : un bean [mySurface] ;

5.4.4 La classe [Appartement]




```
1. package spring.core.entities;
2.
3. import org.springframework.beans.factory.annotation.Autowired;
4. import org.springframework.beans.factory.annotation.Qualifier;
5. import org.springframework.stereotype.Component;
6.
7. @Component
8. public class Appartement {
9.
10.    // champs injectés par Spring
11.    @Autowired
12.    @Qualifier("personne_01")
13.    private Personne propriétaire;
14.
15.    @Autowired
16.    @Qualifier("mySurface")
17.    private int surface;
18.
19.    // getters et setters
20.    public Personne getPropriétaire() {
21.        return propriétaire;
22.    }
23.
24.    public void setPropriétaire(Personne propriétaire) {
25.        this.propriétaire = propriétaire;
26.    }
27.
28.    public int getSurface() {
29.        return surface;
30.    }
31.
32.    public void setSurface(int surface) {
33.        this.surface = surface;
34.    }
35.
36.    // toString
37.    public String toString() {
38.        return String.format("Appartement[%s, %s]", propriétaire, surface);
39.    }
40.
41. }
```

- ligne 7 : l'annotation [@Component] indique à Spring que la classe est un singleton que le framework doit instancier et gérer. C'est parce que dans la classe [Config], nous avons écrit [@ComponentScan({ "istia.st.spring.core.entities" })] que ce singleton va être trouvé ;
- ligne 11 : demande à Spring d'injecter dans le champ la référence d'un des singletons. Celui-ci peut être défini de deux façons :
  - par son identifiant (lignes 12, 16),
  - par son type s'il n'y a qu'un singleton ayant ce type ;

5.4.5 Exécution du projet

L'exécution du projet donne le résultat suivant dans la console :

```
1. 17:32:39.797 [main] DEBUG o.s.core.env.StandardEnvironment - Adding [systemProperties] PropertySource with lowest search
    precedence
2. ....
3. 17:32:40.134 [main] DEBUG o.s.b.f.s.DefaultListableBeanFactory - Returning cached instance of singleton bean 'appartement'
4. personnes--------
5. Personne[Dubois, Paul,34]
6. Personne[Micheline, Martin,18]
7. club--------
8. Personne[Dubois, Paul,34]
9. Personne[Micheline, Martin,18]
10.  appartement--------
11.  Appartement[Personne[Dubois, Paul,34], 200]
12.  17:32:40.135 [main] DEBUG o.s.b.f.s.DefaultListableBeanFactory - Returning cached instance of singleton bean 'personne_01'
13.  beans [p01,p01b] identiques ? true
```

- lignes 1-3 : Spring génère un très grand nombre de logs, plusieurs dizaines de lignes. Ces logs peuvent être très utiles pour déboguer un projet qui ne marche pas. Lorsqu'il fonctionne, on peut réduire les logs de la façon suivante :

Dans le dossier [src / main / resources] on crée le fichier [logback.xml] suivant :

```
1. <configuration>
2.
3.   <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
4.     <!-- encoders are  by default assigned the type
5.          ch.qos.logback.classic.encoder.PatternLayoutEncoder -->
6.     <encoder>
7.       <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>
8.     </encoder>
9.   </appender>
10.
11.    <!-- contrôle niveau des logs -->
12.    <root level="info"> <!-- info, debug, warn -->
13.      <appender-ref ref="STDOUT" />
14.    </root>
15. </configuration>
```

- ligne 12, on fixe le niveau des logs. [debug] est un niveau très détaillé, [info] beaucoup moins ;

Voici les résultats avec [level=info] :

```
1. 17:39:58.580 [main] INFO  o.s.c.a.AnnotationConfigApplicationContext - Refreshing
    org.springframework.context.annotation.AnnotationConfigApplicationContext@7cf10a6f: startup date [Tue Apr 07 17:39:58 CEST
    2015]; root of context hierarchy
2. personnes--------
3. Personne[Dubois, Paul,34]
4. Personne[Micheline, Martin,18]
5. club--------
6. Personne[Dubois, Paul,34]
7. Personne[Micheline, Martin,18]
8. appartement--------
9. Appartement[Personne[Dubois, Paul,34], 200]
10.  beans [p01,p01b] identiques ? true
```

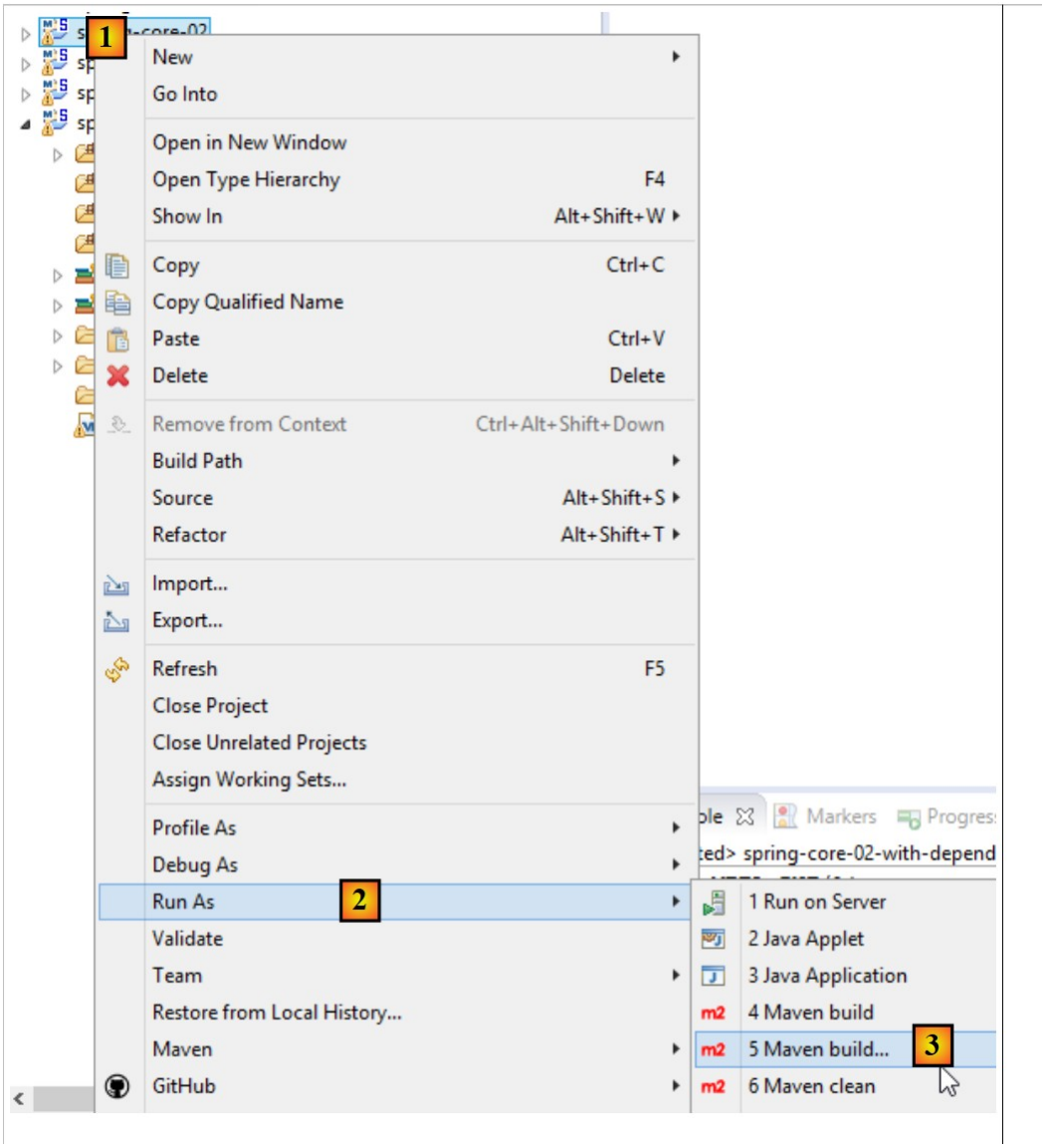
Il n'y a plus qu'une ligne de logs.

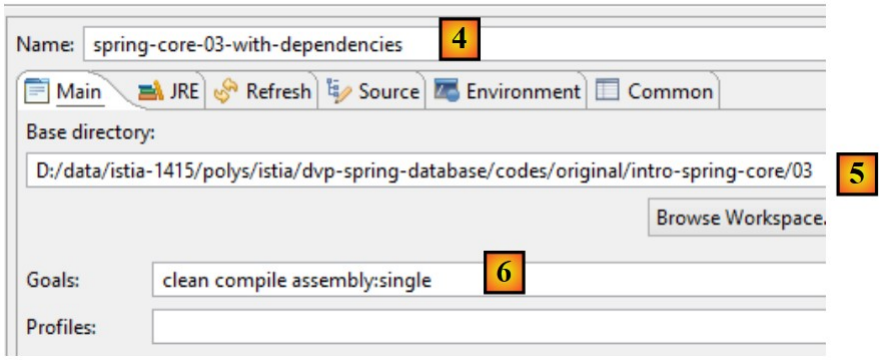
5.4.6 Génération de l'archive du projet avec ses dépendances

L'archive créée dans le projet précédent peut être également utilisée par un projet Eclipse non Maven. Certains projets utilisent de nombreuses bibliothèques et il peut être délicat de ne pas en oublier. C'est là que Maven fait merveille car il suffit de nommer la dépendance de niveau le plus haut pour que les autres de niveau plus bas soient automatiquement ajoutées au Classpath du projet. Lorsque qu'un projet Eclipse non Maven doit utiliser les archives d'un projet Maven, il est possible de générer l'artifact de ce dernier avec toutes ses dépendances (ce qui n'était pas le cas dans le projet précédent). Pour cette génération, il faut que les lignes 3-10 suivantes soient présentes dans le fichier [pom.xml] :

```
1.  <build>
2.    <plugins>
3.      <plugin>
4.        <artifactId>maven-assembly-plugin</artifactId>
5.        <configuration>
6.          <descriptorRefs>
7.            <descriptorRef>jar-with-dependencies</descriptorRef>
8.          </descriptorRefs>
9.        </configuration>
10.     </plugin>
11.     <plugin>
12.       <groupId>org.apache.maven.plugins</groupId>
13.       <artifactId>maven-surefire-plugin</artifactId>
14.       <version>2.18.1</version>
15.     </plugin>
16.   </plugins>
17. </build>
```

Elles définissent le plugin de Maven capable de générer l'artifact du projet avec ses dépendances. Ensuite on procède de la façon suivante [1-6] :

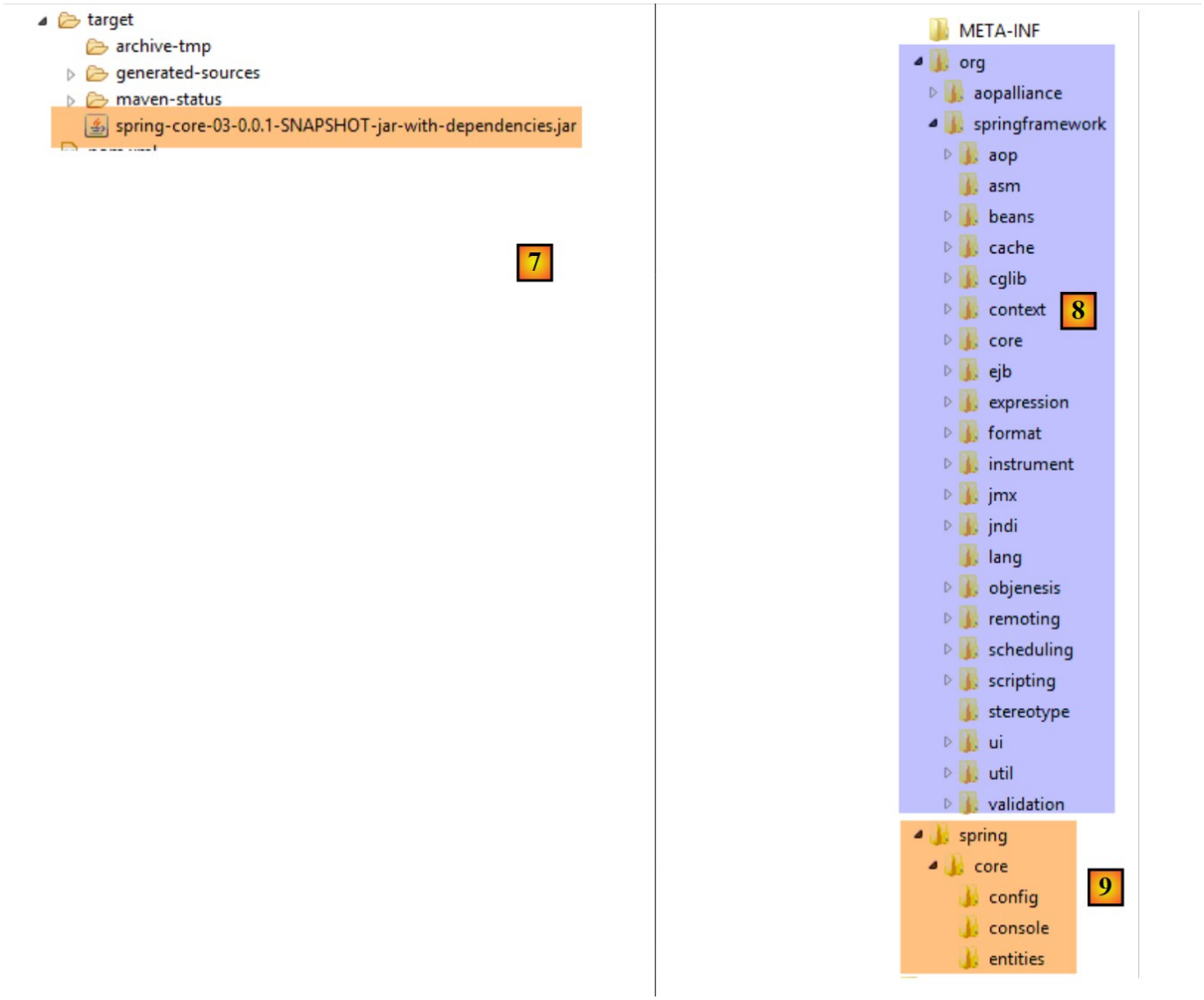




- [4-6] représentent une configuration d'exécution Maven ;
- en [4], mettre un nom quelconque ;
- en [5], désigner le dossier du projet ;
- en [6], mettre les cibles Maven (goals) :
  - [clean] : le dossier [target] du projet est supprimé ;
  - [compile] : le projet est compilé. Les produits de la compilation sont placés dans un dossier [target] régénéré ;

- [assembly:single] : les classes du projet et de ses dépendances sont placées dans une unique archive jar dans le dossier [target] ;

Après exécution, on obtient le résultat suivant :



Une archive jar est un fichier zippé qu'on peut donc ouvrir avec un dézippeur. Une fois l'archive précédente dézippée, on obtient l'arborescence suivante :

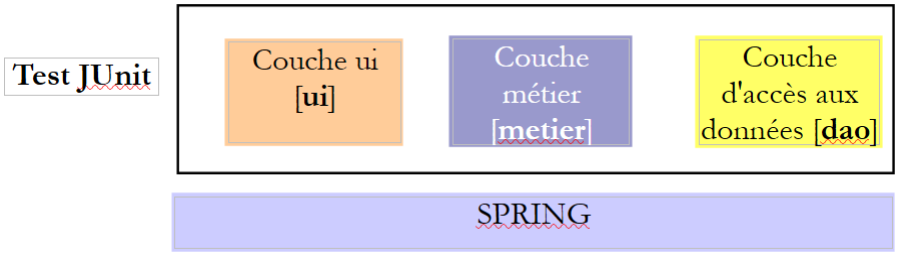
- en [8], les classes des dépendances du projet ;
- en [9], les classes du projet lui-même ;

5.5 Exemple-04

5.5.1 Objectif

Cet exemple reprend l'un de ceux présentés dans le document [Introduction à Spring IoC] dans lequel on montre l'apport de Spring pour la configuration d'architectures multi-couche. Dans le document original, l'exemple est traité avec une configuration Spring faite avec un fichier XML. Ici nous traitons l'exemple avec une configuration par classes Java et annotations.

On veut ici configurer un projet Spring pour l'architecture suivante:

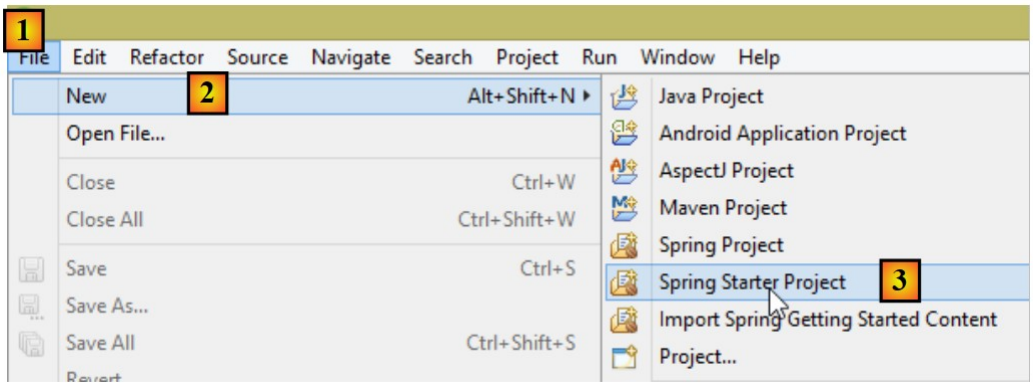


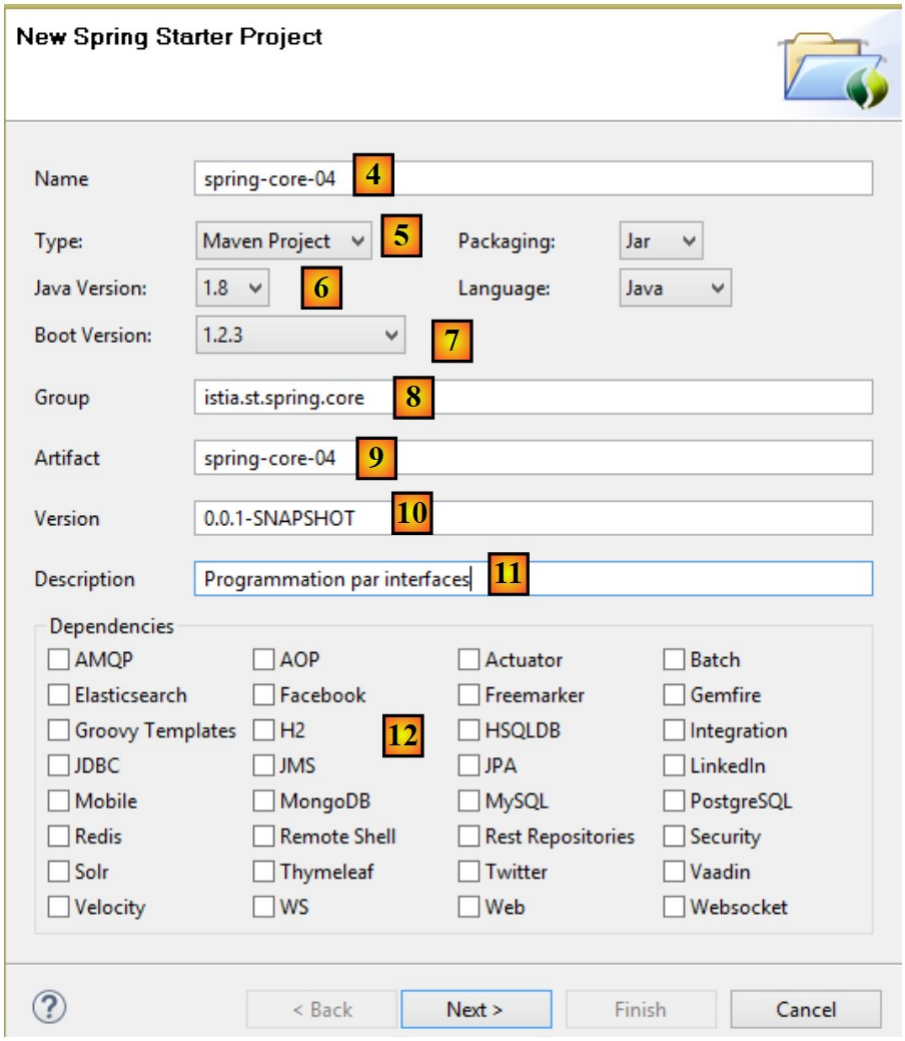
Chaque couche présente une interface implémentée avec deux classes. On veut montrer que grâce à Spring, on peut changer l'implémentation d'une couche avec un impact zéro sur le code des autres couches.

5.5.2 Le projet Eclipse

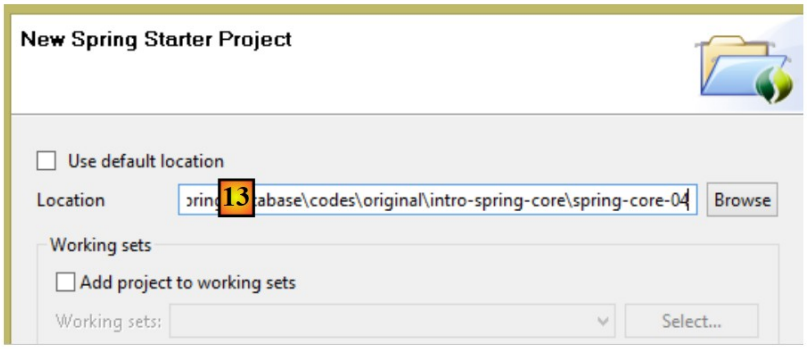
5.5.2.1 Génération

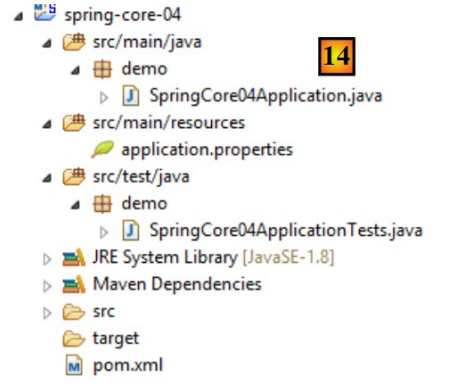
Nous créons un nouveau type de projet :





- en [4], mettre le nom du projet Eclipse ;
- en [5], choisir un projet Maven ;
- en [6], choisir une version de Java >=1.7 ;
- en [7], choisir la version de Spring Boot proposée ;
- les informations [8-11] sont des informations Maven ;
- en [12], on peut choisir une ou plusieurs des dépendances proposées. Cela va avoir pour effet d'intégrer dans le fichier [pom.xml] de Maven, un certain nombre de dépendances ;





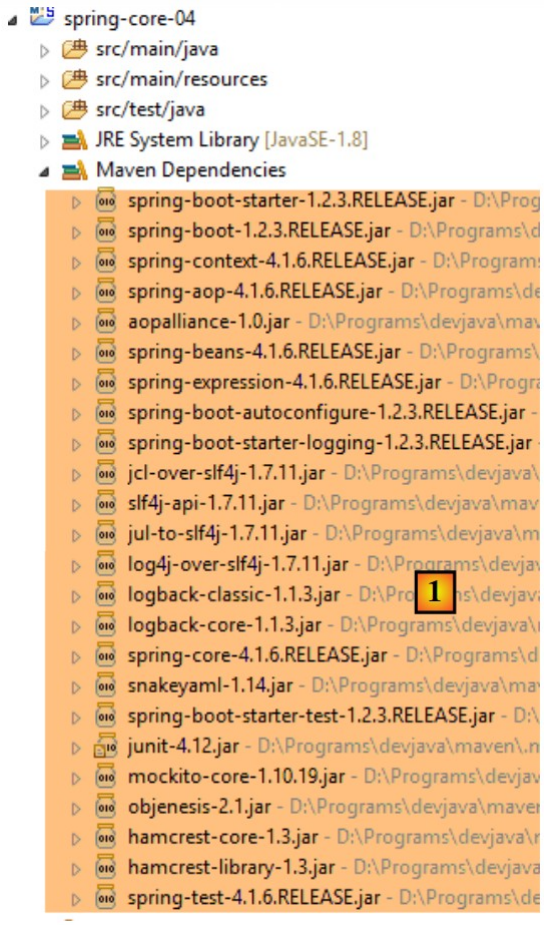
- en [13], désigner un dossier existant et vide pour accueillir le projet ;
- en [14], le projet généré. Nous allons en disséquer les éléments ;

Le projet est un projet Maven configuré par le fichier [pom.xml] suivant :

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3.   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
4.   <modelVersion>4.0.0</modelVersion>
5.
6.   <groupId>istia.st.spring.core</groupId>
7.   <artifactId>spring-core-04</artifactId>
8.   <version>0.0.1-SNAPSHOT</version>
9.   <packaging>jar</packaging>
10.
11.    <name>spring-core-04</name>
12.    <description>Programmation par interfaces</description>
13.
14.    <parent>
15.        <groupId>org.springframework.boot</groupId>
16.        <artifactId>spring-boot-starter-parent</artifactId>
17.        <version>1.2.3.RELEASE</version>
18.        <relativePath/> <!-- lookup parent from repository -->
19.    </parent>
20.
21.    <properties>
22.        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
23.        <start-class>demo.SpringCore04Application</start-class>
24.        <java.version>1.8</java.version>
25.    </properties>
26.
27.    <dependencies>
28.        <dependency>
29.            <groupId>org.springframework.boot</groupId>
30.            <artifactId>spring-boot-starter</artifactId>
31.        </dependency>
32.
33.        <dependency>
34.            <groupId>org.springframework.boot</groupId>
35.            <artifactId>spring-boot-starter-test</artifactId>
36.            <scope>test</scope>
37.        </dependency>
38.    </dependencies>
39.
40.    <build>
41.        <plugins>
42.            <plugin>
43.                <groupId>org.springframework.boot</groupId>
44.                <artifactId>spring-boot-maven-plugin</artifactId>
45.            </plugin>
46.        </plugins>
47.    </build>
48.
49. </project>
```

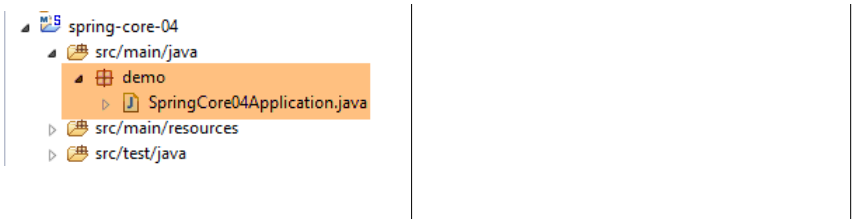
- lignes 6-12 : reprennent les informations saisies dans l'assistant de réation du projet ;
- lignes 14-19 : le projet Maven parent qui définit un certain nombre de bibliothèques avec leurs versions. Si l'une d'elles est une dépendance du projet, elle est mentionnée dans le fichier [pom.xml] sans sa version ;
- ligne 23 : cette ligne ne sert que si on a l'intention de générer une archive exécutable du projet. Elle est inutilisée sinon ;
- lignes 28-31 : la dépendance minimale d'un projet Spring Boot. On rappelle que nous n'avons sélectionné aucune dépendance dans la liste à cocher ;
- lignes 33-37 : la dépendance nécessaire pour gérer des tests unitaires JUnit [http://junit.org/] intégrés avec Spring. La ligne 36 indique que la dépendance n'est nécessaire que pour les tests. En conséquence, elle ne sera pas intégrée à l'archive du projet ;
- lignes 42-45 : le plugin qui permet de générer l'artifact Maven du projet ;

La liste des dépendances amenée par ce fichier est la suivante [1] :



Nous allons voir qu'elles sont suffisantes pour ce qu'on veut faire ici.

5.5.2.2 La classe exécutable



La classe exécutable [SpringCore04Application] [[2] est la suivante :

```
1. package demo;
2.
3. import org.springframework.boot.SpringApplication;
4. import org.springframework.boot.autoconfigure.SpringBootApplication;
5.
6. @SpringBootApplication
7. public class SpringCore04Application {
8.
9.     public static void main(String[] args) {
10.         SpringApplication.run(SpringCore04Application.class, args);
11.     }
12. }
```

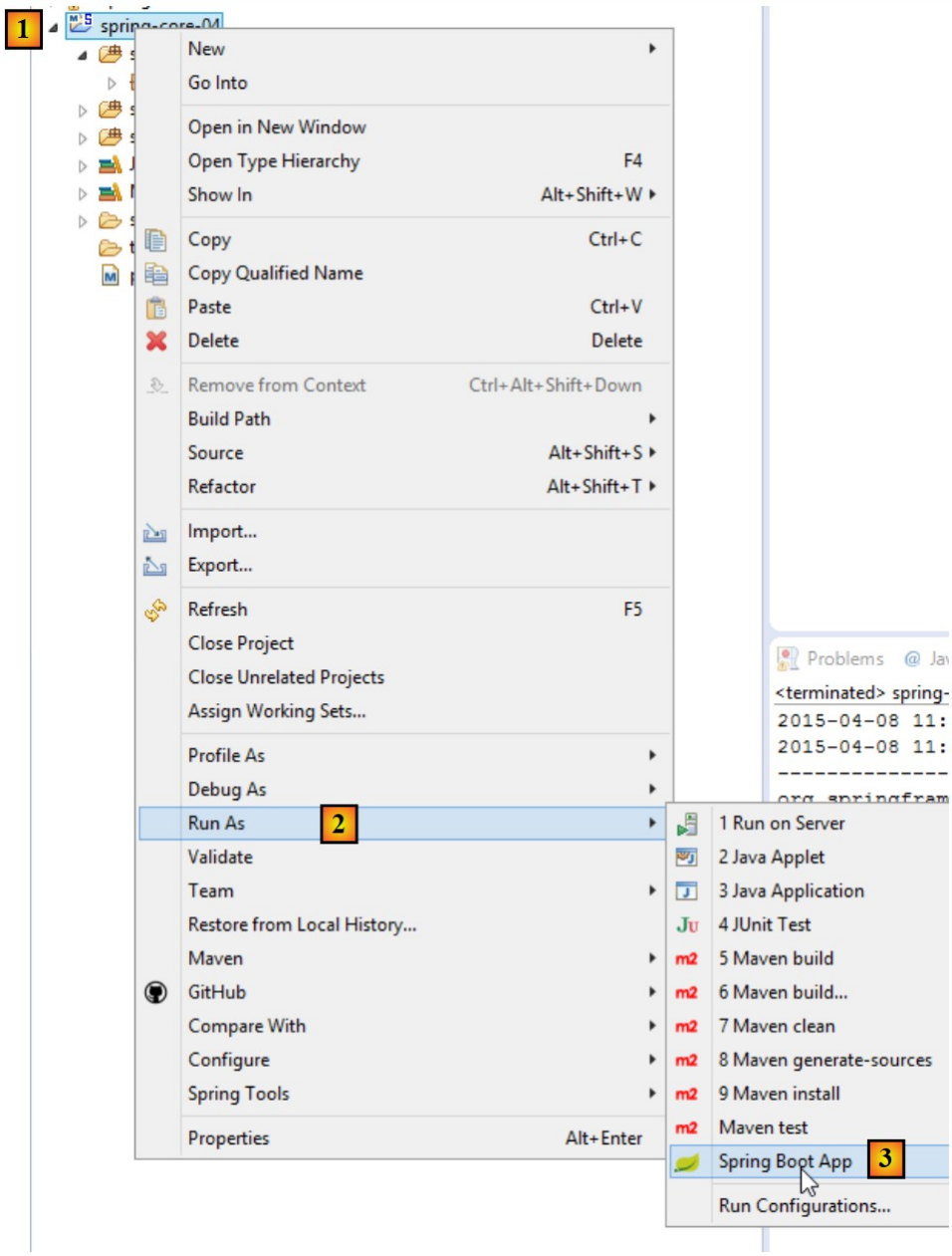
- ligne 6, l'annotation [@SpringBootApplication] est un raccourci pour les trois annotations [@Configuration, @EnableAutoConfiguration, @ComponentScan] ce qui signifie :
  - que la classe [SpringCore04Application] est une classe de configuration Spring ;

    - qu'il est demandé à Spring Boot d'opérer des configurations à partir des classes qu'il va trouver dans le Classpath du projet, donc ici dans les dépendances Maven ;
    - d'examiner le dossier courant (celui de la classe [SpringCore04Application]) pour y trouver d'éventuels autres composants Spring ;
- ligne 10 : la méthode statique [SpringApplication.run] est exécutée. Son premier paramètre est une classe de configuration Spring, ici la classe [SpringCore04Application]. Son deuxième paramètre est ici la liste des arguments passés à la méthode [main] (ligne 9). La méthode statique [SpringApplication.run] a pour rôle de créer le contexte Spring, ç-à-d créer les différents beans trouvés soit dans les classes de configuration soit dans les dossiers explorés par l'annotation [@ComponentScan]. La méthode [main] ici ne fait rien d'autre. Pour lui donner un peu plus de substance, nous allons la transformer de la façon suivante :

```
1. package demo;
2.
3. import org.springframework.boot.SpringApplication;
4. import org.springframework.boot.autoconfigure.SpringBootApplication;
5. import org.springframework.context.ConfigurableApplicationContext;
6.
7. @SpringBootApplication
8. public class SpringCore04Application {
9.
10.    public static void main(String[] args) {
11.        // instanciation du contexte Spring
12.        ConfigurableApplicationContext context = SpringApplication.run(SpringCore04Application.class, args);
13.        // affichage du contexte
14.        System.out.println("---------------- Liste des beans Spring");
15.        for (String beanName : context.getBeanDefinitionNames()) {
16.            System.out.println(beanName);
17.        }
18.        // fermeture contexte
19.        context.close();
20.    }
21. }
```

- ligne 12 : la méthode statique [SpringApplication.run] rend le contexte Spring qu'elle a construit ;
- lignes 15-17 : on affiche le nom de tous les beans de ce contexte ;

On peut exécuter l'application de la façon suivante [1-3]. La méthode habituelle [Run As Java Application] est également valide.



On obtient le résultat suivant :

```
1.   .   ____          _            __ _ _
2.  /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
3. ( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
4.  \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
5.   '  |____| .__|_| |_|_| |_\__, | / / / /
6.  =========|_|==============|___/=/_/_/_/
7.  :: Spring Boot ::        (v1.2.3.RELEASE)
8.
9. 2015-04-08 11:18:38.254  INFO 4796 --- [           main] demo.SpringCore04Application             : Starting
    SpringCore04Application on Gportpers3 with PID 4796 (D:\data\istia-1415\polys\istia\dvp-spring-
    database\codes\original\intro-spring-core\spring-core-04\target\classes started by ST in D:
    \data\istia-1415\polys\istia\dvp-spring-database\codes\original\intro-spring-core\spring-core-04)
10.  2015-04-08 11:18:38.295  INFO 4796 --- [           main] s.c.a.AnnotationConfigApplicationContext : Refreshing
    org.springframework.context.annotation.AnnotationConfigApplicationContext@64485a47: startup date [Wed Apr 08 11:18:38 CEST
    2015]; root of context hierarchy
11.  2015-04-08 11:18:38.776  INFO 4796 --- [           main] o.s.j.e.a.AnnotationMBeanExporter        : Registering beans for
    JMX exposure on startup
12.  2015-04-08 11:18:38.788  INFO 4796 --- [           main] demo.SpringCore04Application             : Started
    SpringCore04Application in 0.773 seconds (JVM running for 1.335)
13.  ---------------- Liste des beans Spring
14.  org.springframework.context.annotation.internalConfigurationAnnotationProcessor
15.  org.springframework.context.annotation.internalAutowiredAnnotationProcessor
16.  org.springframework.context.annotation.internalRequiredAnnotationProcessor
```

```
17. org.springframework.context.annotation.internalCommonAnnotationProcessor
18. springCore04Application
19. org.springframework.context.annotation.ConfigurationClassPostProcessor.importAwareProcessor
20. org.springframework.context.annotation.ConfigurationClassPostProcessor.enhancedConfigurationProcessor
21. org.springframework.boot.autoconfigure.AutoConfigurationPackages
22. org.springframework.boot.autoconfigure.PropertyPlaceholderAutoConfiguration
23. org.springframework.boot.autoconfigure.condition.BeanTypeRegistry
24. propertySourcesPlaceholderConfigurer
25. org.springframework.boot.autoconfigure.jmx.JmxAutoConfiguration
26. mbeanExporter
27. objectNamingStrategy
28. mbeanServer
29. 2015-04-08 11:18:38.789  INFO 4796 --- [           main] s.c.a.AnnotationConfigApplicationContext : Closing
    org.springframework.context.annotation.AnnotationConfigApplicationContext@64485a47: startup date [Wed Apr 08 11:18:38 CEST
    2015]; root of context hierarchy
30. 2015-04-08 11:18:38.790  INFO 4796 --- [           main] o.s.j.e.a.AnnotationMBeanExporter        : Unregistering JMX-
    exposed beans on shutdown
```

- lignes 14-28 : les beans du contexte Spring. Nous ne connaissons pas leur rôle. Nous retrouvons le bean [springCore04Application] ligne 18 qui de par son annotation [@SpringBootApplication] devient automatiquement un bean Spring ;
- les autres lignes sont des logs de Spring de niveau [INFO]. Comme nous l'avons déjà vu, ces logs peuvent être contrôlés par le fichier [logback.xml] placé dans le Classpath du projet :

```
1. <configuration>
2.
3.   <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
4.     <!-- encoders are  by default assigned the type
5.          ch.qos.logback.classic.encoder.PatternLayoutEncoder -->
6.     <encoder>
7.       <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>
8.     </encoder>
9.   </appender>
10.
11.   <!-- contrôle niveau des logs -->
12.   <root level="warn"> <!-- info, debug, warn -->
13.     <appender-ref ref="STDOUT" />
14.   </root>
15. </configuration>
```

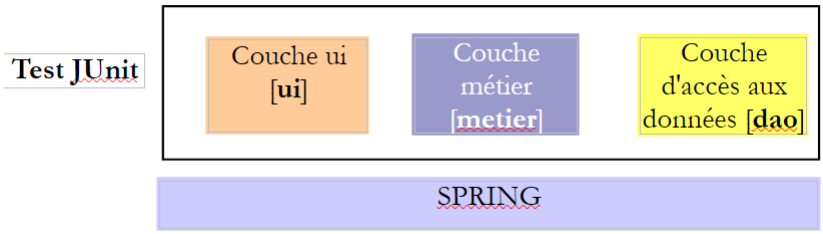
Si ligne 12 ci-dessus, on met le niveau [warn] à la place de [info], on obtient le résultat suivant :

```
1.   .   ____          _            __ _ _
2.  /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
3. ( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
4.  \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
5.   '  |____| .__|_| |_|_| |_\__, | / / / /
6.  =========|_|==============|___/=/_/_/_/
7.  :: Spring Boot ::        (v1.2.3.RELEASE)
8.
9. ---------------- Liste des beans Spring
10.  org.springframework.context.annotation.internalConfigurationAnnotationProcessor
11.  org.springframework.context.annotation.internalAutowiredAnnotationProcessor
12.  org.springframework.context.annotation.internalRequiredAnnotationProcessor
13.  org.springframework.context.annotation.internalCommonAnnotationProcessor
14.  springCore04Application
15.  org.springframework.context.annotation.ConfigurationClassPostProcessor.importAwareProcessor
16.  org.springframework.context.annotation.ConfigurationClassPostProcessor.enhancedConfigurationProcessor
17.  org.springframework.boot.autoconfigure.AutoConfigurationPackages
18.  org.springframework.boot.autoconfigure.PropertyPlaceholderAutoConfiguration
19.  org.springframework.boot.autoconfigure.condition.BeanTypeRegistry
20.  propertySourcesPlaceholderConfigurer
```

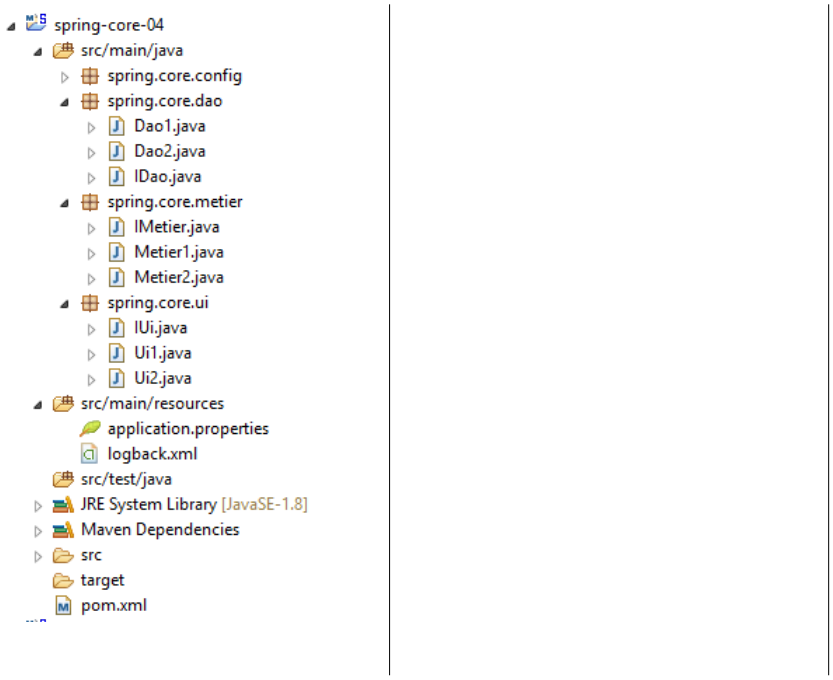
```
21.  org.springframework.boot.autoconfigure.jmx.JmxAutoConfiguration
22.  mbeanExporter
23.  objectNamingStrategy
24.  mbeanServer
```

Les logs ont disparu. N'apparaissent que les messages de niveau [warn] et ici il n'y en pas eu.

5.5.3 Implémentation des différentes couches de l'architecture



Nous allons maintenant implémenter les trois couches de l'architecture ci-dessus :



La couche [DAO] est implémentée par le package [spring.core.dao]. Elle présente l'interface [IDao] suivante :

```
1. package spring.core.dao;
2.
3. public interface IDao {
4.
5.   public int doSomethingInDaoLayer(int a, int b);
6. }
```

Cette interface a deux implémentations : [Dao1] et [Dao2] :

```
1. package spring.core.dao;
2.
3. public class Dao1 implements IDao {
4.
5.   public int doSomethingInDaoLayer(int a, int b) {
6.     return a+b;
7.   }
8.
9. }
```

```
1. package spring.core.dao;
```

```
2.
3. public class Dao2 implements IDao {
4.
5.   public int doSomethingInDaoLayer(int a, int b) {
6.     return a-b;
7.   }
8.
9. }
```

La couche [métier] est implémentée par le package [spring.core.metier]. Elle présente l'interface [IMetier] suivante :

```
1. package spring.core.metier;
2.
3. public interface IMetier {
4.
5.   public int doSomethingInMetierLayer(int a, int b);
6. }
```

Cette interface a deux implémentations : [Metier1] et [Metier2] :

```
1. package spring.core.metier;
2.
3. import spring.core.dao.IDao;
4.
5. public class Metier1 implements IMetier {
6.
7.   private IDao dao;
8.
9.   public int doSomethingInMetierLayer(int a, int b) {
10.       a++;
11.       b++;
12.       return dao.doSomethingInDaoLayer(a, b);
13.    }
14.
15.    public void setDao(IDao dao) {
16.       this.dao = dao;
17.    }
18.  }
```

```
1. package spring.core.metier;
2.
3. import spring.core.dao.IDao;
4.
5. public class Metier2 implements IMetier {
6.
7.   private IDao dao;
8.
9.   public int doSomethingInMetierLayer(int a, int b) {
10.       a--;
11.       b++;
12.       return dao.doSomethingInDaoLayer(a, b);
13.    }
14.
15.    public void setDao(IDao dao) {
16.       this.dao = dao;
17.    }
18.
19.
20.  }
```

La couche [UI] est implémentée par le package [spring.core.ui]. Elle présente l'interface [IUi] suivante :

```
1. package spring.core.ui;
2.
3. public interface IUi {
4.
5.   public int doSomethingInUiLayer(int a, int b);
6. }
```

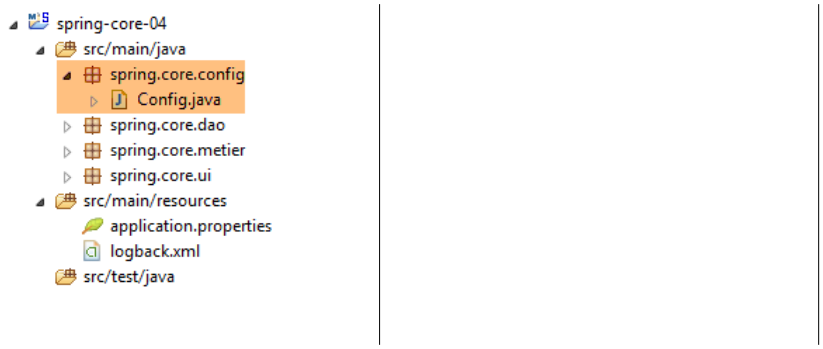
Cette interface a deux implémentations : [Ui1] et [Ui2] :

```
1. package spring.core.ui;
2.
3. import spring.core.metier.IMetier;
4.
5. public class Ui1 implements IUi {
6.
7.   private IMetier metier;
8.
9.   public int doSomethingInUiLayer(int a, int b) {
10.       a++;
11.       b++;
```

```
12.         return metier.doSomethingInMetierLayer(a, b);
13.     }
14.
15.     public void setMetier(IMetier metier) {
16.         this.metier = metier;
17.     }
18.
19. }
```

```
1. package spring.core.ui;
2.
3. import spring.core.metier.IMetier;
4.
5. public class Ui2 implements IUi {
6.
7.   private IMetier metier;
8.
9.   public int doSomethingInUiLayer(int a, int b) {
10.         a--;
11.         b++;
12.         return metier.doSomethingInMetierLayer(a, b);
13.     }
14.
15.     public void setMetier(IMetier metier) {
16.         this.metier = metier;
17.     }
18.
19. }
```

5.5.4 Configuration du projet Spring



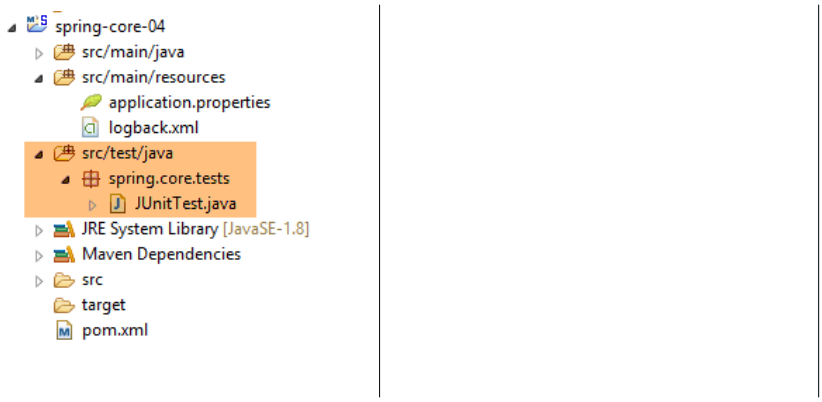
La classe de configuration [Config] est la suivante :

```
1. package spring.core.config;
2.
3. import org.springframework.context.annotation.Bean;
4. import org.springframework.context.annotation.Configuration;
5.
6. import spring.core.dao.Dao1;
7. import spring.core.dao.Dao2;
8. import spring.core.dao.IDao;
9. import spring.core.metier.IMetier;
10.  import spring.core.metier.Metier1;
11.  import spring.core.metier.Metier2;
12.  import spring.core.ui.IUi;
13.  import spring.core.ui.Ui1;
14.  import spring.core.ui.Ui2;
15.
16.  @Configuration
17.  public class Config {
18.
19.    // -------------- implémentation [Ui1, Metier1, Dao1]
20.    @Bean
21.    public IDao dao1() {
22.        return new Dao1();
23.    }
24.
25.    @Bean
26.    public IMetier metier1(IDao dao1) {
27.        Metier1 metier = new Metier1();
28.        metier.setDao(dao1);
29.        return metier;
30.    }
31.
32.    @Bean
33.    public IUi ui1(IMetier metier1) {
```

```
34.         Ui1 ui = new Ui1();
35.         ui.setMetier(metier1);
36.         return ui;
37.     }
38.
39.     // -------------- implémentation [Ui2, Metier2, Dao2]
40.     @Bean
41.     public IDao dao2() {
42.         return new Dao2();
43.     }
44.
45.     @Bean
46.     public IMetier metier2(IDao dao2) {
47.         Metier2 metier = new Metier2();
48.         metier.setDao(dao2);
49.         return metier;
50.     }
51.
52.     @Bean
53.     public IUi ui2(IMetier metier2) {
54.         Ui2 ui = new Ui2();
55.         ui.setMetier(metier2);
56.         return ui;
57.     }
58. }
```

- lignes 20-23 : le bean nommé [dao1] (nom de la méthode) est une instance de la classe [Dao1] (ligne 22) considérée comme une implémentation de l'interface [IDao] (ligne 21). Le bean [dao1] est donc vu comme une instance d'interface (la terminologie est incorrecte mais elle peut se comprendre) et non une instance de classe. C'est un point important à comprendre. Tous les autres beans vont être eux aussi des instances d'interfaces ;
- lignes 25-30 : une instance de l'interface [IMetier] implémentée par la classe [Metier1] ;
- lignes 32-37 : une instance de l'interface [IUi] implémentée par la classe [Ui1] ;
- lignes 20-37 : implémentent les couches [UI, Metier, DAO] avec des instances [Ui1, Metier1, Dao1] ;
- lignes 40-57 : implémentent les couches [UI, Metier, DAO] avec des instances [Ui2, Metier2, Dao2] ;

5.5.5 Test unitaire [JUnitTest]



La classe [JUnitTest] est mise dans la branche [src / test / java] du projet Maven. Les éléments de cette branche ne sont pas placés dans l'archive finale du projet. Son code est le suivant :

```
1. package spring.core.tests;
2.
3. import org.junit.Assert;
4. import org.junit.Test;
5. import org.junit.runner.RunWith;
6. import org.springframework.beans.factory.annotation.Autowired;
7. import org.springframework.beans.factory.annotation.Qualifier;
8. import org.springframework.boot.test.SpringApplicationConfiguration;
9. import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
10.
11.  import spring.core.config.Config;
12.  import spring.core.dao.IDao;
13.  import spring.core.metier.IMetier;
14.  import spring.core.ui.IUi;
15.
16.  @SpringApplicationConfiguration(classes = { Config.class })
17.  @RunWith(SpringJUnit4ClassRunner.class)
```

```
18. public class JUnitTest {
19. ...
20. }
```

- ligne 16 : l'annotation [@SpringApplicationConfiguration] est une annotation du projet Spring Boot Test (ligne 8). Elle est amenée par la dépendance suivante du fichier [pom.xml] :

```
1.     <dependency>
2.         <groupId>org.springframework.boot</groupId>
3.         <artifactId>spring-boot-starter</artifactId>
4. </dependency>
```

  Cette annotation a pour paramètre la liste des classes de configuration à utiliser pour construire le contexte Spring nécessaire au test. Ici nous utilisons la classe de configuration [Config] déjà présentée ;
- ligne 17 : l'annotation [@RunWith] est une annotation JUnit (ligne 5). Son paramètre est la classe chargée de conduire les tests à la place de la classe par défaut du framework Junit. Cette classe est une classe Spring (ligne 9). Elle va utiliser les annotations Spring présentes dans la classe de test ;

La classe complète est la suivante

```
1. ...
2.
3. @SpringApplicationConfiguration(classes = { Config.class })
4. @RunWith(SpringJUnit4ClassRunner.class)
5. public class JUnitTest {
6.
7.   // couche [UI]
8.   @Autowired
9.   @Qualifier("ui1")
10.   private IUi ui1;
11.
12.   @Autowired
13.   @Qualifier("ui2")
14.   private IUi ui2;
15.
16.   // couche [métier]
17.   @Autowired
18.   @Qualifier("metier1")
19.   private IMetier metier1;
20.
21.   @Autowired
22.   @Qualifier("metier2")
23.   private IMetier metier2;
24.
25.   // couche [dao]
26.   @Autowired
27.   @Qualifier("dao1")
28.   private IDao dao1;
29.
30.   @Autowired
31.   @Qualifier("dao2")
32.   private IDao dao2;
33.
34.   @Test
35.   public void testDao() {
36.       Assert.assertEquals(30, dao1.doSomethingInDaoLayer(10, 20));
37.       Assert.assertEquals(-10, dao2.doSomethingInDaoLayer(10, 20));
38.   }
39.
40.   @Test
41.   public void testMetier() {
42.       Assert.assertEquals(32, metier1.doSomethingInMetierLayer(10, 20));
43.       Assert.assertEquals(-12, metier2.doSomethingInMetierLayer(10, 20));
44.   }
45.
46.   @Test
47.   public void testUI() {
48.       Assert.assertEquals(34, ui1.doSomethingInUiLayer(10, 20));
49.       Assert.assertEquals(-14, ui2.doSomethingInUiLayer(10, 20));
50.   }
51.
52. }
```

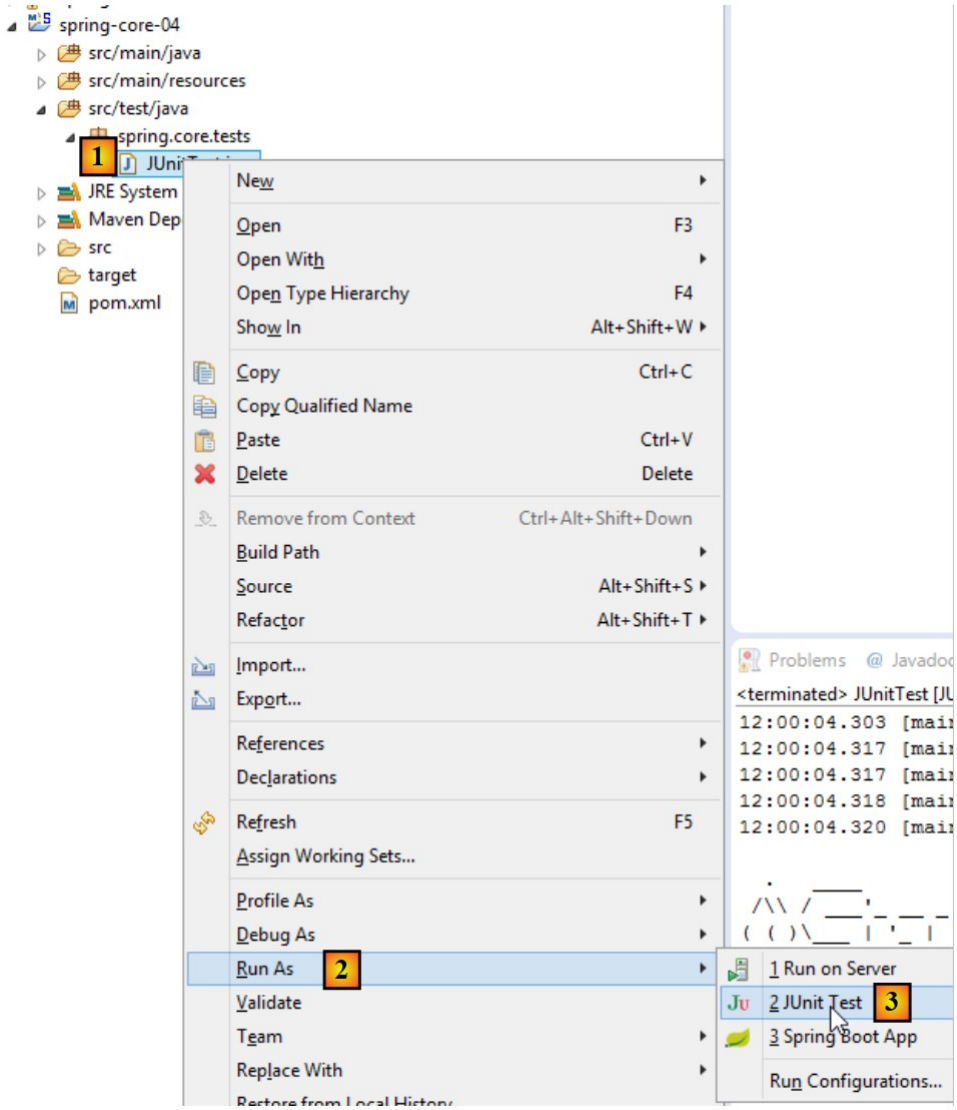
- lignes 8-10 : on injecte (ligne 8) le bean nommé (ligne 9) [ui1]. On remarquera ligne 10 qu'on injecte une instance d'interface et non une instance de classe ;
- lignes 21-32 : on injecte de la même façon les autres beans définis dans la classe [Config] ;
- ligne 34 : l'annotation [@Test] désigne une méthode à exécuter au cours des tests. Les autres annotations possibles sont les suivantes :
  - [@BeforeClass] : méthode à exécuter avant de commencer les tests ;

    - [@AfterClass] : méthode à exécuter une fois tous les tests faits ;
    - [@Before] : méthode à exécuter **avant chaque** test ;
    - [@After] : méthode à exécuter **après chaque** test ;
- line 36 : on vérifie que l'appel [dao1.doSomethingInDaoLayer(10, 20)] rend bien 30. Le premier paramètre est par convention, la valeur attendue et le second la valeur réelle ;
- ligne 36 : teste l'instance [dao1] de l'interface [IDao] ;
- ligne 37 : teste l'instance [dao2] de l'interface [IDao] ;
- ligne 42 : teste l'instance [metier1] de l'interface [IMetier] ;
- ligne 43 : teste l'instance [metier2] de l'interface [IMetier] ;
- ligne 48 : teste l'instance [ui1] de l'interface [IUi] ;
- ligne 36 : teste l'instance [ui2] de l'interface [IUi] ;

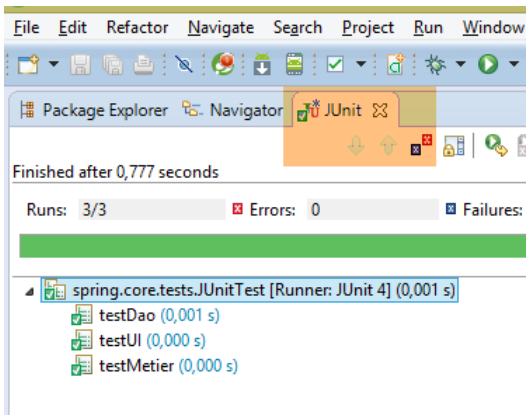
Les assertions utilisables dans une méthode de test sont les suivantes :

- **assertEquals(expression1, expression2)** : vérifie que les valeurs des deux expressions sont égales. De nombreux types d'expression sont acceptés (int, String, float, double, boolean, char, short). Si les deux expressions ne sont pas égales, alors une exception de type [AssertionFailedError] est lancée,
- **assertEquals(réel1, réel2, delta)** : vérifie que deux réels sont égaux à **delta** près, c.a.d abs(réel1-réel2)<=delta. On pourra écrire par exemple **assertEquals(réel1, réel2, 1E-6)** pour vérifier que deux valeurs sont égales à 10^{-6} près,
- **assertEquals(message, expression1, expression2)** et **assertEquals(message, réel1, réel2, delta)** sont des variantes permettant de préciser le message d'erreur à associer à l'exception de type [AssertionFailedError] lancée lorsque la méthode [assertEquals] échoue,
- **assertNotNull(Object)** et **assertNotNull(message, Object)** : vérifie que la référence **Object** n'est pas égale à **null,**
- **assertNull(Object)** et **assertNull(message, Object)** : vérifie que la référence **Object** est égale à **null,**
- **assertSame(Object1, Object2)** et **assertSame(message, Object1, Object2)** : vérifie que les références **Object1** et **Object2** pointent sur le même objet,
- **assertNotSame(Object1, Object2)** et **assertNotSame(message, Object1, Object2)** : vérifie que les références **Object1** et **Object2** ne pointent pas sur le même objet ;

Pour exécuter le test, on peut procéder de la façon suivante :



On obtient le résultat suivant :



Ici tous les test ont réussi. Au fait que montre cet exemple ? Il montre la souplesse apportée par le framework Spring dans la configuration d'une architecture en couches. On peut décider d'utiliser l'implémentation [Ui1, Metier1, Dao1] ou [Ui2, Metier2,

Dao2] simplement par configuration. Ainsi dans le test JUnit précédent, si on ne garde que l'injection des beans [ui1, metier1, dao1] on travaille avec la 1ère architecture. Pour changer d'architecture, il suffit de changer les beans injectés. Cela se fait sans changement de code des couches implémentant les interfaces. On appelle ce type de programmation, programmation par interfaces car on n'utilise pas les instances des classes implémentant les couches mais les instances de leurs interfaces.

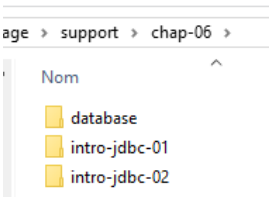
5.6 Conclusion

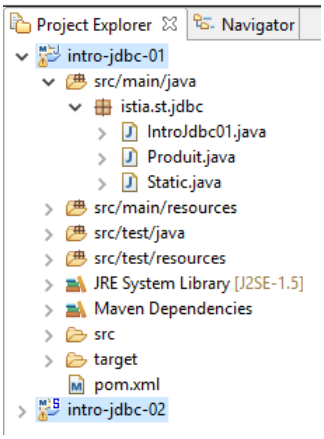
- Spring gère des objets qui sont des singletons (un seul exemplaire). Spring gère également des objets qui sont instanciés à chaque fois qu'on en demande une instance à Spring. Ce cas sera également présenté dans ce document ;
- ces objets peuvent être déclarés de diverses façons qui peuvent être mixées :
  - dans un fichier XML,
  - dans une classe Java annotée par [@Configuration],
  - avec n'importe quelle classe Java annotée par [@Component, @Service, ...] ;
- un objet Spring peut être injecté dans un autre objet Spring avec l'annotation [@Autowired]. On parle alors d'injection de dépendances (DI : Dependency Injection) ;
- Spring se montre très pratique pour configurer des architectures en couches avec l'utilisation conjointe du paradigme de de la programmation par interfaces ;

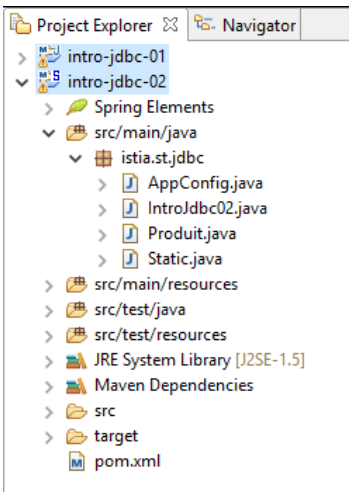
6 [Cours] : Introduction à l'API JDBC

Mots clés : bases de données relationnelles, API JDBC, SQLException.

6.1 Support

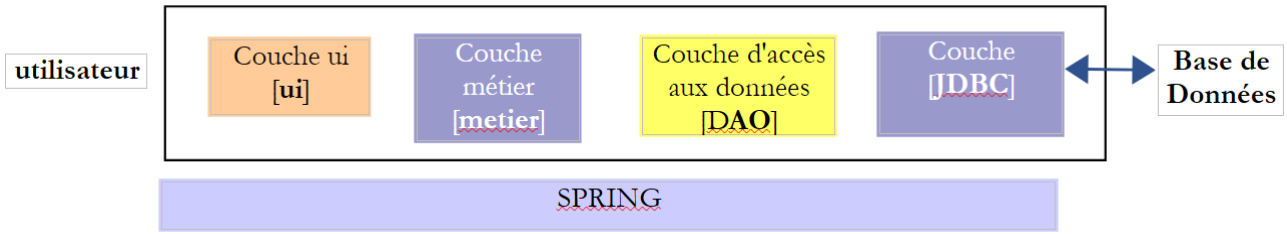






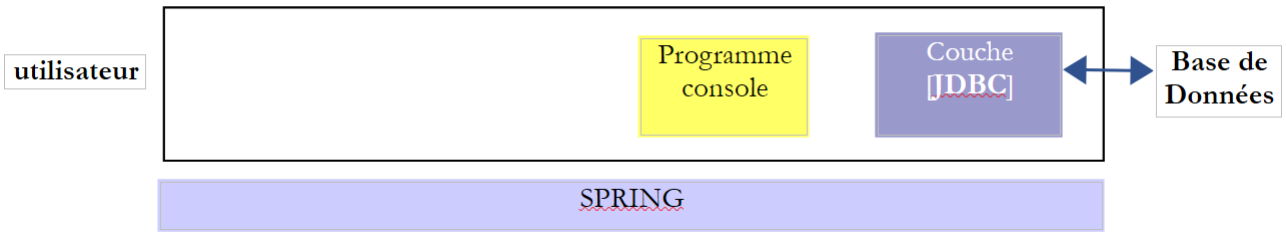
Le dossier [support / chap-06] contient les projets Eclipse de ce chapitre.

6.2 Architecture



La couche JDBC (Java DataBase Connectivity) est une interface d'accès universelle aux bases de données. Elle présente toujours la même interface à la couche [DAO]. Si on change de SGBD, il suffit de changer le pilote JDBC. La couche [DAO] ne change pas.

6.3 Les étapes d'exploitation d'une base de données



Dans l'architecture ci-dessus, l'exploitation d'une base de données par le programme console comporte les étapes suivantes :

1. chargement du pilote JDBC de la base de données ;
2. ouverture d'une connexion avec la base ;
3. émission d'un ordre SQL sur la base et traitement des résultats de l'ordre SQL ;
4. fermeture de la connexion ;

L'étape 1 ne se fait qu'une fois. Les étapes 2-4 se font de façon répétée. On notera qu'on ne laisse pas une connexion ouverte. On la ferme dès qu'on en n'a plus besoin.

6.3.1 étape 1 - chargement en mémoire du pilote JDBC

Le code

```
1.    // chargement du pilote JDBC
2.    try {
3.        Class.forName(nom de la classe du pilote JDBC);
4.    } catch (ClassNotFoundException e1) {
5.        // traiter l'exception
6. }
```

L'opération de la ligne 3 a pour but de charger en mémoire le pilote JDBC de la base de données. Cette opération n'a besoin d'être faite qu'une fois. La répéter ne cause cependant pas d'erreur. La classe du pilote JDBC est cherchée dans le Classpath du projet. Il faut donc que dans le projet Eclipse, le [jar] contenant la classe du pilote JDBC ait été inclus dans le Classpath du projet.

6.3.2 étape 2 - ouverture d'une connexion

Une fois le pilote JDBC en place, on lui demande d'ouvrir une connexion avec la BD :

Le code

```
1. package spring.jdbc;
2.
3. import java.sql.Connection;
4. import java.sql.DriverManager;
5. import java.sql.PreparedStatement;
6. import java.sql.ResultSet;
7. import java.sql.SQLException;
8.
9. public class IntroJdbc01 {
10.
11. ...
12.      Connection connexion = null;
13.      PreparedStatement ps = null;
14.      ResultSet rs = null;
15.      try {
16.         // ouverture connexion
17.         connexion = DriverManager.getConnection(url, user, passwd);
18. ...
19.      } catch (SQLException e1) {
20.         // on traite l'exception
21.         ...
22.      } finally {
23.       // fermer la connexion
24.       if (connexion != null) {
25.         try {
26.            connexion.close();
27.         } catch (SQLException e2) {
28.            // traiter l'exception
29.            ...
30.         }
31.       }
32. }
```

- lignes 3-7 : les classes d'implémentation de l'interface JDBC sont toutes dans le package [java.sql]. Par ailleurs, en cas d'erreur elles lancent toutes une exception de type [SQLException] (ligne 19, 27). Cette exception dérive de la classe [Exception] et est une exception dite contrôlée : on est obligé de mettre un try / catch pour la gérer ou de façon alternative de ne pas la gérer et d'indiquer que la méthode laisse sortir l'exception en complétant la signature de la méthode par [throws SQLException] ;
- ligne 17, [DriverManager.getConnection] est une méthode statique qui attend trois paramètres :

- [url] : l'URL de la base de données. C'est une chaîne de caractères dépendante de la BD utilisée. Pour MySQL, elle est de la forme [*jdbc:mysql://localhost:3306/nom_de_la_bd*];
- [user] : le propriétaire de la connexion ;
- [passwd] : son mot de passe ;

- lignes 24-30 : la connexion doit être fermée dans la clause [finally] afin qu'elle soit fermée qu'il y ait exception ou non.

6.3.3 étape 3 - émission d'ordres SQL [SELECT]

Une fois obtenue une connexion, on peut émettre des ordres SQL. La façon de gérer des ordres de lecture [SELECT] diffère de celle utilisée pour les opérations de mise à jour [UPDATE, INSERT, DELETE]. Nous commençons par les ordres SQL [SELECT] :

Le code

```
1. Connection connexion = null;
2.    PreparedStatement ps = null;
3.    ResultSet rs = null;
4.    try {
5.       // ouverture connexion
6.       connexion = DriverManager.getConnection(url, user, passwd);
7.       // début transaction
8.       connexion.setAutoCommit(false);
9.       // en mode lecture seule
10.         connexion.setReadOnly(true);
11.         // on lit la table [PRODUITS]
12.         ps = connexion.prepareStatement("SELECT ID, NOM, CATEGORIE, PRIX, DESCRIPTION FROM PRODUITS");
13.         rs = ps.executeQuery();
14.         System.out.println("Liste des produits : ");
15.         while (rs.next()) {
16.            System.out.println(new Produit(rs.getInt(1), rs.getString(2), rs.getInt(3), rs.getDouble(4),
    rs.getString(5)));
17.         }
18.         // commit transaction
19.         connexion.commit();
20.      } catch (SQLException e1) {
21.         // on traite l'exception
22.          doCatchException(connexion,e1);
23.      } finally {
24.         // on traite le finally
25.         doFinally(rs, ps, connexion);
26.      }
27.
28.   private void doFinally(ResultSet rs, PreparedStatement ps, Connection connexion) {
29. ....
30. }
```

- lignes 8, 10 : ouverture d'une transaction (ligne 8) en mode lecture seule (ligne 10). Une transaction est une séquence d'ordes SQL qui soit tous réussissent soit tous échouent. Ainsi dans une transaction comportant N ordres SQL, si l'ordre I+1 échoue, alors les I précédents seront annulés. Pour une opération de lecture, une transaction n'est pas nécessaire. Néanmoins créer une transaction en lecture seule peut permettre à certains SGBD de faire certaines optimisations ;
- ligne 12 : utilisation d'un [PreparedStatement]. Un [PreparedStatement] a normalement des paramètres notés par le caractère ?. Ici il n'en a pas. Un [PreparedStatement] est un ordre préparé par le SGBD. Cette préparation a un coût et elle n'est faite qu'une fois. Ensuite cet ordre préparé est exécuté par le SGBD avec différents paramètres effectifs qui vont venir remplacer les paramètres formels ?. A noter qu'il est préférable de nommer les colonnes désirées plutôt que d'utiliser la notation * pour obtenir toutes les colonnes. En précisant le nom des colonnes on peut ensuite obtenir leurs valeurs à partir de leur position dans la requête SELECT ;
- ligne 13 : exécution du [PreparedStatement]. On récupère un objet de type [ResultSet] ;

  Un objet de type [ResultSet] représente une table, c'est à dire un ensemble de lignes et de colonnes. A un moment donné, on n'a accès qu'à une ligne de la table appelée ligne courante. Lors de la création initiale du [ResultSet], il n'y a pas de ligne courante. Il faut faire une opération [ResultSet.next()] pour l'obtenir. La signature de la méthode *next* est la suivante :

  ```
  boolean next()
  ```

  Cette méthode tente de passer à la ligne suivante du [ResultSet] et rend *true* si elle réussit, *false* sinon. En cas de réussite, la ligne suivante devient la nouvelle ligne courante. La ligne précédente est perdue et on ne pourra revenir en arrière pour la récupérer.

La table du [ResultSet] a des colonnes nommées labelCol1, labelCol2,... précisées dans la requête [SELECT] exécutée. Avec la requête :

```
SELECT ID as myId, NOM as myNom, CATEGORIE as myCategorie, PRIX as myPrix, DESCRIPTION as myDescription FROM PRODUITS
```

- la colonne [ID] ira dans une colonne du [ResultSet] nommée [myId] ;
- la colonne [NOM] ira dans une colonne du [ResultSet] nommée [myNom] ;
- ...

Ci-dessus, les identifiants [myCol] sont appelés des **labels de colonne**. En l'absence de ces labels, les noms des colonnes du [ResultSet] sont dépendants du SGBD. Lorsque le [SELECT] opère sur une unique table, les labels des colonnes seront par défaut les noms des colonnes demandées par le SELECT. Le problème surgit lorsque le [SELECT] opère sur plusieurs tables et que dans celles-ci on trouve des noms de colonnes identiques comme dans l'exemple suivant :

```
SELECT PRODUITS.NOM, CATEGORIES.NOM FROM PRODUITS, CATEGORIES WHERE PRODUITS.CATEGORIE_ID=CATEGORIES.ID
```

en imaginant que la table [PRODUITS] ait une clé étrangère vers la table [CATEGORIES] symbolisée par la relation [Produits].CATEGORIE_ID --> [CATEGORIES].ID, et que les tables [PRODUITS] et [CATEGORIES] aient toutes les deux un champ [NOM]. Dans ce cas, les noms donnés dans le [ResultSet] aux colonnes [`PRODUITS.NOM`] et [`CATEGORIES.NOM`] sont dépendants du SGBD. Pour la portabilité entre SGBD, il faut donc utiliser des labels de colonnes ici et on écrira :

```
SELECT PRODUITS.NOM as p_NOM, CATEGORIES.NOM as c_NOM FROM PRODUITS, CATEGORIES WHERE PRODUITS.CATEGORIE_ID=CATEGORIES.ID
```

Pour exploiter les différents champs de la ligne courante du [ResultSet], on dispose des méthodes suivantes :

```
Type getType("labelColi")
```

pour obtenir la colonne nommée «labelColi» de la ligne courante et donc la colonne du [SELECT] ayant ce label. *Type* désigne le type du champ *coli*. On peut utiliser les méthodes [getType] suivantes : getInt, getLong, getString, getDouble, getFloat, getDate, ... Au lieu d'utiliser le nom de la colonne, on peut utiliser sa position dans la requête [SELECT] exécutée :

```
Type getType(i)
```

où i est l'indice de la colonne désirée (i>=1).

- lignes 15-17 : récupération des valeurs lues dans la BD ;
- ligne 19 : la transaction est validée (on dit également committée). Cela la termine et libère les ressources que le SGBD avait mobilisées pour elle ;
- ligne 25 : les ressources sont libérées dans le [finally]. Celui-ci appelle la méthode [doFinally] suivante :

```
1.  private void doFinally(ResultSet rs, PreparedStatement ps, Connection connexion) {
2.     // fermeture ResultSet
3.     if (rs != null) {
4.        try {
5.           rs.close();
6.        } catch (SQLException e1) {
7.
8.        }
9.     }
10.    // fermeture [PreparedStatement]
11.    if (ps != null) {
12.       try {
13.          ps.close();
14.       } catch (SQLException e2) {
15.
16.       }
17.    }
18.    if (connexion != null) {
19.       try {
20.          // fermer la connexion
21.          connexion.close();
22.       } catch (SQLException e3) {
```

```
23.             // traiter l'exception
24.         }
25.     }
26. }
```

- lignes 3-9 : fermeture du [ResultSet] ;
- lignes 11-17 : fermeture du [PreparedStatement] ;
- lignes 18-27 : fermeture de la connexion ;

Les fermetures des lignes 3-17 semblent redondantes dans la mesure on ferme la connexion lignes 18-25. En fait, dans certains cas elles ne le sont pas et il est conseillé de les laisser [http://stackoverflow.com/questions/4507440/must-jdbc-resultsets-and-statements-be-closed-separately-although-the-connection].

- ligne 22 : l'exception est traitée par la méthode [doCatchException] suivante :

```
1.    private static void doCatchException(Connection connexion, Throwable th) {
2.        // annulation transaction
3.        try {
4.            if (connexion != null) {
5.                connexion.rollback();
6.            }
7.        } catch (SQLException e2) {
8.            // traiter l'exception
9.        }
10.   }
```

  - lignes 4-6 : la transaction est annulée. Cela la termine et le SGBD va pouvoir relâcher les ressources mobilisées pour elle ;

6.3.4 étape 3 - émission d'ordres SQL [INSERT, UPDATE, DELETE]

Les ordres SQL [INSERT, UPDATE, DELETE] sont des opérations de mise à jour : elles modifient la base de données mais ne ramènent aucune ligne. La seule information rendue est le nombre de lignes affectées par l'opération de mise à jour.

Le code

```
1. Connection connexion = null;
2.     PreparedStatement ps = null;
3.     try {
4.         // ouverture connexion
5.         connexion = DriverManager.getConnection(url, user, passwd);
6.         // début transaction
7.         connexion.setAutoCommit(false);
8.         // en mode lecture / écriture
9.         connexion.setReadOnly(false);
10.            // on met à jour la table
11.            ps = connexion.prepareStatement("UPDATE PRODUITS SET PRIX=PRIX*1.1 WHERE CATEGORIE=?");
12.            // catégorie 1
13.            ps.setInt(1, 10);
14.            // exécution
15.            int nbLignes=ps.executeUpdate();
16.            // commit transaction
17.            connexion.commit();
18.        } catch (SQLException e1) {
19.            // on traite l'exception
20.            doCatchException(connexion, e1);
21.        } finally {
22.            // on traite le finally
23.            doFinally(null, ps, connexion);
24.        }
25.    }
```

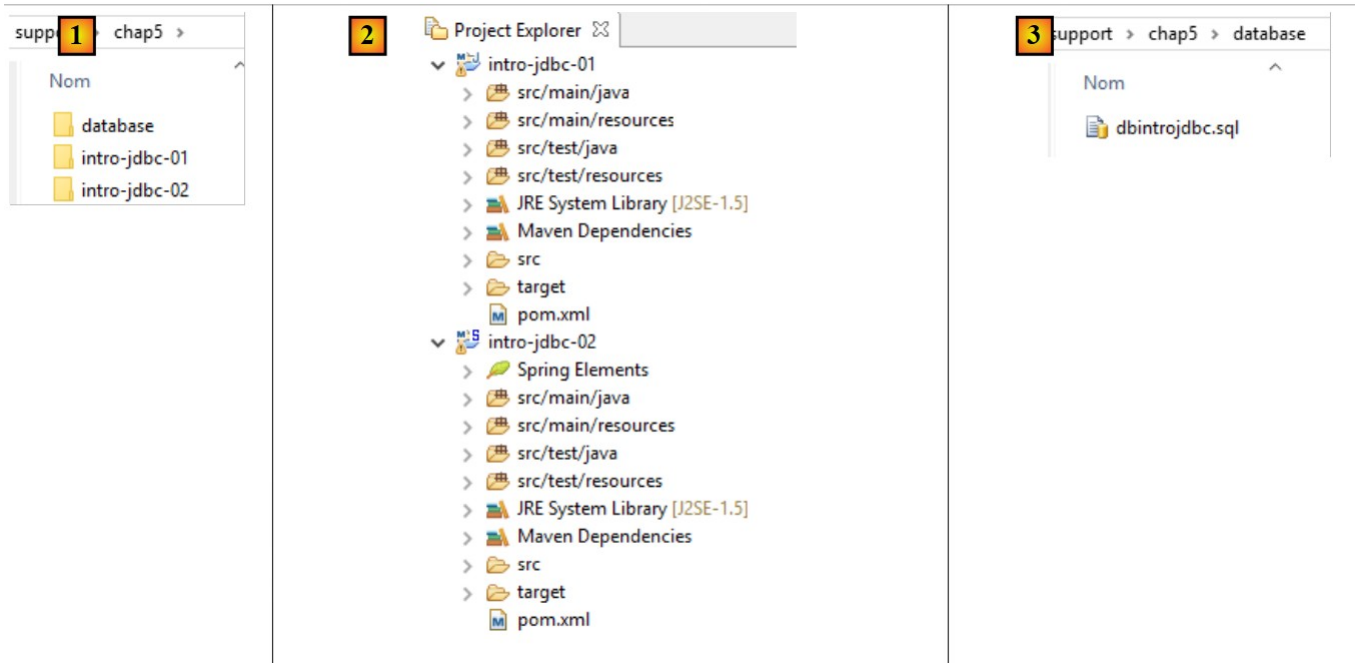
- ligne 9 : la connexion est utilisée en lecture et écriture ;
- ligne 11 : un [PreparedStatement] avec 1 paramètre (symbolisé par ?). On peut avoir plusieurs paramètres. Ils sont numérotés à partir de 1 ;
- ligne 13 : on affecte sa valeur à l'unique paramètre. Le 1er paramètre de [setType] est la position du paramètre dans le [PreparedStatement] (1, 2, ...) et le second la valeur qui lui est attribuée. On peut utiliser les méthodes [setInt, setLong, setFloat, setDouble, setString, setDate, ...] ;
- ligne 15 : on utilise la méthode [executeUpdate] et non [executeQuery] réservée aux ordres SELECT. La méthode rend le nombre de lignes affectées par l'opération. Peut être 0.
- ligne 17 : la transaction est validée ;

6.3.5 étape 4 - fermeture de la connexion

Une connexion doit être fermée le plus vite possible dans un contexte multi-utilisateurs car un SGBD accepte un nombre limité de connexions ouvertes. Dans les exemples précédents, elle était fermée dans la clause [finally] des opérations SQL afin qu'elle soit fermée qu'il y ait eu exception ou pas.

6.4 Un projet exemple

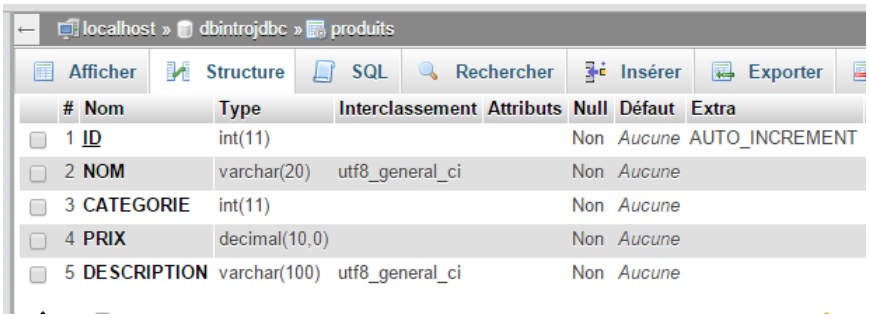
6.4.1 Support



Le dossier [support / chap5] contient les projets Eclipse de ce chapitre [1, 2]. Le dossier [database] contient le script SQL permettant de créer la base MySQL exemple de ce chapitre [1, 3].

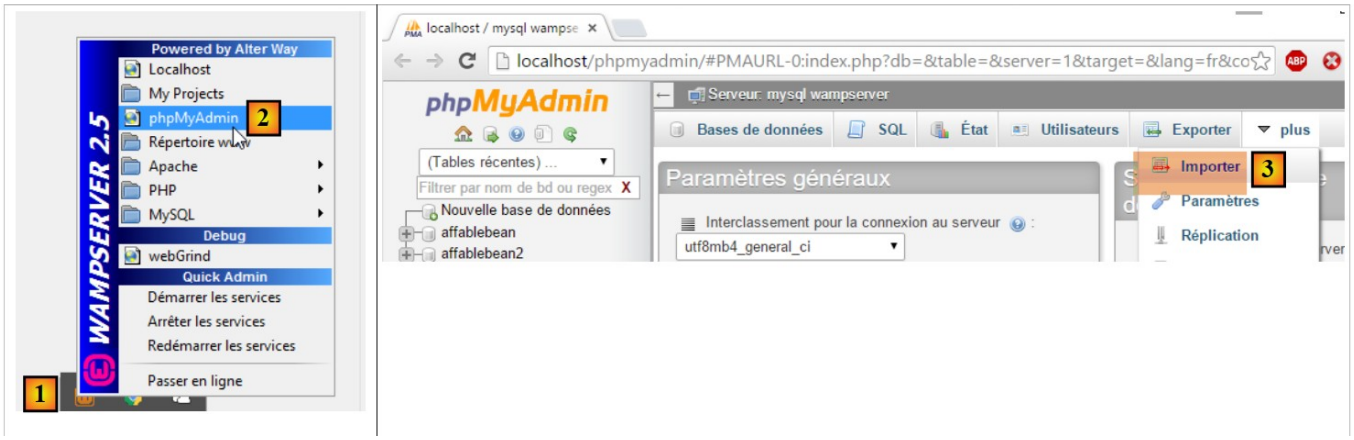
6.4.2 La base de données exploitée

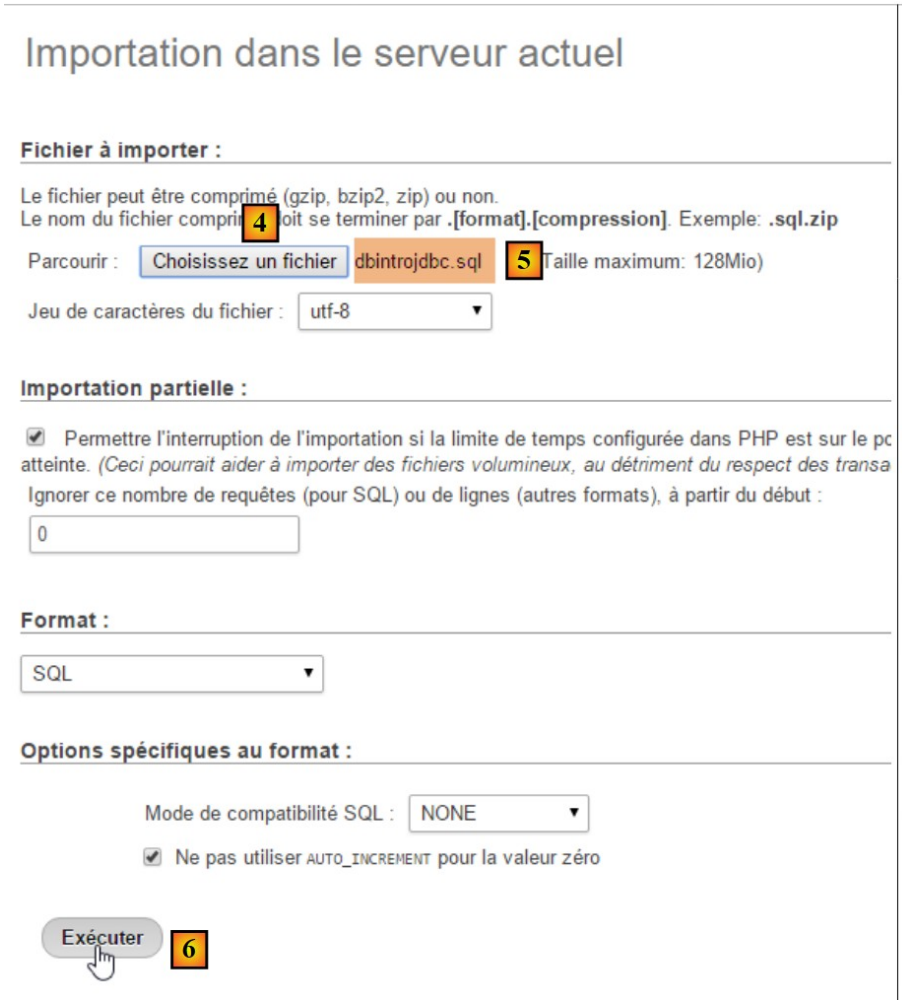
Les exemples qui suivent exploitent la base de données MySQL suivante :

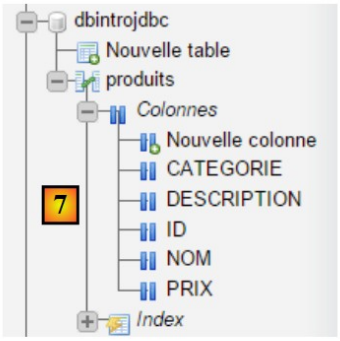


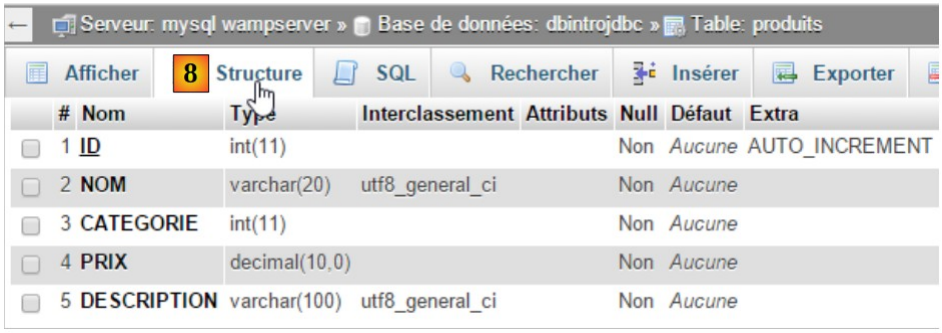
- [ID] : clé primaire en mode AUTO_INCREMENT (si on ne donne pas de clé primaire, le SGBD la génère) ;
- [NOM] : nom d'un produit - unique ;
- [CATEGORIE] : n° de sa catégorie ;
- [PRIX] : son prix ;
- [DESCRIPTION] : une description du produit ;

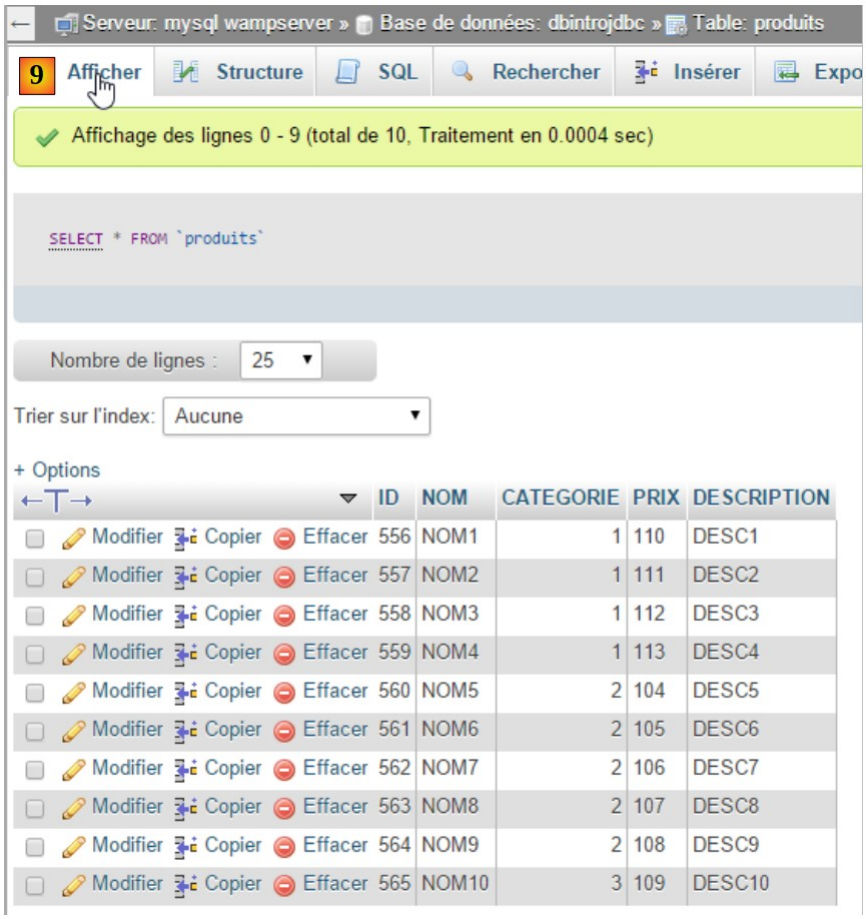
On la créera avec l'outil [WampServer] de la façon suivante [1-9] :



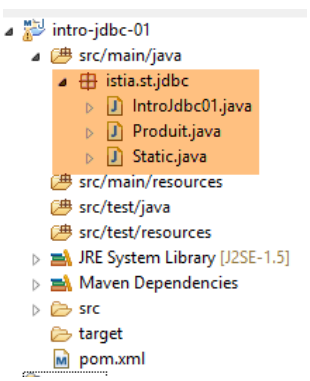








6.4.3 Le projet Eclipse



Le projet est un projet Maven défini par le fichier [pom.xml] suivant :

```
1. <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2.   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
3.   <modelVersion>4.0.0</modelVersion>
4.   <groupId>istia.st.jdbc</groupId>
5.   <artifactId>intro-jdbc-01</artifactId>
6.   <version>0.0.1-SNAPSHOT</version>
7.   <dependencies>
8.     <dependency>
9.       <groupId>mysql</groupId>
10.         <artifactId>mysql-connector-java</artifactId>
11.         <version>5.1.34</version>
12.      </dependency>
13.      <dependency>
14.         <groupId>com.fasterxml.jackson.core</groupId>
15.         <artifactId>jackson-databind</artifactId>
16.         <version>2.5.1</version>
17.      </dependency>
18.   </dependencies>
19. </project>
```

- lignes 8-12 : le pilote JDBC du SGBD MySQL5 ;
- lignes 13-17 : une bibliothèque capable de gérer du jSON (Javascript Object Notation) (cf paragraphe 22.6, page 420). Nous allons l'utiliser pour afficher sous forme jSON les produits de la base de données ;

6.4.4 La classe des produits

La classe [Produit] est la suivante :

```
1. package istia.st.jdbc;
2.
3. import com.fasterxml.jackson.core.JsonProcessingException;
4. import com.fasterxml.jackson.databind.ObjectMapper;
5.
6. public class Produit {
7.
8.   // champs
9.   private int id;
10.   private String nom;
11.   private int categorie;
12.   private double prix;
13.   private String description;
14.
15.   // constructeurs
16.   public Produit() {
17.
18.   }
19.
20.   public Produit(int id, String nom, int categorie, double prix, String description) {
21.       this.id = id;
22.       this.nom = nom;
23.       this.categorie = categorie;
24.       this.prix = prix;
25.       this.description = description;
26.   }
27.
28.   // getters et setters
```

```
29.    ...
30.
31.    // to String
32.    public String toString() {
33.        try {
34.            return new ObjectMapper().writeValueAsString(this);
35.        } catch (JsonProcessingException e) {
36.            e.printStackTrace();
37.            return null;
38.        }
39.    }
40. }
```

- ligne 34 : on utilise la bibliothèque jSON pour afficher la chaîne jSON du produit. Cela donne un affichage analogue au suivant :

```
Liste des produits :
{"id":1,"nom":"NOM1","categorie":1,"prix":100.0,"description":"DESC1"}
{"id":2,"nom":"NOM2","categorie":1,"prix":101.0,"description":"DESC2"}
```

L'intérêt de la méthode [toString] ci-dessus est que si on ajoute / retranche des champs à la classe, sa méthode [toString] est toujours valide. Par ailleurs, si les champs sont eux-mêmes des objets (listes, tableaux, dictionnaires, objets utilisateur), les bibliothèques jSON savent les transformer à leur tour en chaîne jSON ;

6.4.5 La classe [Static]

La classe [Static] regroupe dans des méthodes du code utilisé fréquemment dans la classe principale :

```
1. package istia.st.jdbc;
2.
3. import java.util.ArrayList;
4. import java.util.List;
5.
6. public class Static {
7.
8.   public static List<String> getErreursFromThrowable(Throwable th) {
9.     // on récupère la liste des msg d'erreur de l'exception
10.       List<String> erreurs = new ArrayList<String>();
11.       while (th != null) {
12.           // message d'erreur du throwable
13.           erreurs.add(th.getMessage());
14.           // on passe à la cause du throwable
15.           th = th.getCause();
16.       }
17.       // résultat
18.       return erreurs;
19.   }
20.
21.   public static void show(String title, List<String> messages){
22.       // titre
23.       System.out.println(String.format("%s : ",title));
24.       // messages
25.       for(String message : messages){
26.           System.out.println(String.format("- %s",message));
27.       }
28.   }
29. }
```

- lignes 8-19 : permet d'avoir la liste des erreurs encapsulées dans un objet de type [Throwable] qui est la classe mère de la classe [Exception] ;
- lignes 21-28 : affiche à l'écran une liste de messages ;

Ce code pourrait être dans la classe principale parce qu'ici c'est la seule à l'utiliser. On se place ici dans un cas plus large ou d'autres classes auraient besoin de ce code.

6.4.6 Le squelette de la classe principale

```
1. package istia.st.jdbc;
2.
3. import java.sql.Connection;
4. import java.sql.DriverManager;
5. import java.sql.PreparedStatement;
6. import java.sql.ResultSet;
7. import java.sql.SQLException;
8.
9. public class IntroJdbc01 {
10.
11.   // constantes
```

```
12.    final static String url = "jdbc:mysql://localhost:3306/dbIntroJdbc";
13.    final static String user = "root";
14.    final static String passwd = "";
15.    final static String insert = "INSERT INTO PRODUITS(ID, NOM, CATEGORIE, PRIX, DESCRIPTION) VALUES (?, ?, ?, ?, ?)";
16.    final static String delete = "DELETE FROM PRODUITS";
17.    final static String select = "SELECT ID, NOM, CATEGORIE, PRIX, DESCRIPTION FROM PRODUITS";
18.    final static String update = "UPDATE PRODUITS SET PRIX=PRIX*1.1 WHERE CATEGORIE=?";
19.    final static String insert2 = "INSERT INTO PRODUITS(ID, NOM, CATEGORIE, PRIX, DESCRIPTION) VALUES (100,'X',1,1,'x')";
20.
21.    public static void main(String[] args) {
22.        // chargement du pilote JDBC de MySQL
23.        try {
24.            Class.forName("com.mysql.jdbc.Driver");
25.        } catch (ClassNotFoundException e1) {
26.            doCatchException("Pilote JDBC introuvable", null, e1);
27.            return;
28.        }
29.        // on vide la table [PRODUITS]
30.        delete();
31.        // on la remplit
32.        insert();
33.        // on la lit
34.        select();
35.        // mise à jour
36.        update();
37.        // affichage
38.        select();
39.        // insertion de deux éléments identiques
40.        // l'insertion doit échouer et aucun des deux éléments n'est inséré à cause de la transaction
41.        insert2();
42.        // on vérifie
43.        select();
44.        // fini
45.        System.out.println("Travail terminé");
46.    }
47.
48.    // liste des produits
49.    private static void select() {
50.  ...
51.    }
52.
53.    // suppression produits
54.    public static void delete() {
55.  ...
56.    }
57.
58.    // ajout produits
59.    public static void insert() {
60.  ...
61.    }
62.
63.    // ajout 2 produits
64.    public static void insert2() {
65.  ...
66.    }
67.
68.    // mise à jour de certains produits
69.    public static void update() {
70.  ..
71.    }
72.
73.    private static void doFinally(ResultSet rs, PreparedStatement ps, Connection connexion) {
74.        // fermeture ResultSet
75.        if (rs != null) {
76.            try {
77.                rs.close();
78.            } catch (SQLException e1) {
79.
80.            }
81.        }
82.        // fermeture [PreparedStatement]
83.        if (ps != null) {
84.            try {
85.                ps.close();
86.            } catch (SQLException e2) {
87.
88.            }
89.        }
90.        // fermer la connexion
91.        if (connexion != null) {
92.            try {
93.                connexion.close();
94.            } catch (SQLException e3) {
95.                // on affiche les msg d'erreur
96.                Static.show("Les erreurs suivantes se sont produites lors de la fermeture de la connexion",
97.                        Static.getErreursFromThrowable(e3));
98.            }
```

```
99.       }
100.  }
101.
102.  private static void doCatchException(String title, Connection connexion, Throwable th) {
103.      // on affiche les msg d'erreur
104.      Static.show(title, Static.getErreursFromThrowable(th));
105.      // annulation transaction
106.      try {
107.          if (connexion != null) {
108.              connexion.rollback();
109.          }
110.      } catch (SQLException e2) {
111.          // on affiche les msg d'erreur
112.          Static.show(title, Static.getErreursFromThrowable(e2));
113.      }
114.  }
115. }
```

6.4.7 Suppression du contenu de la table des produits

La méthode [delete] supprime le contenu de la table :

```
1. // suppression produits
2.   public static void delete() {
3.     Connection connexion = null;
4.     PreparedStatement ps = null;
5.     try {
6.         // ouverture connexion
7.         connexion = DriverManager.getConnection(url, user, passwd);
8.         // début transaction
9.         connexion.setAutoCommit(false);
10.            // on vide la table [PRODUITS]
11.            ps = connexion.prepareStatement(delete);
12.            ps.executeUpdate();
13.            // commit transaction
14.            connexion.commit();
15.            // retour au mode par défaut
16.            connexion.setAutoCommit(true);
17.        } catch (SQLException e1) {
18.            // on traite l'exception
19.            doCatchException("Les erreurs suivantes se sont produites à la suppression du contenu de la table", connexion,
    e1);
20.        } finally {
21.            // on traite le finally
22.            doFinally(null, null, connexion);
23.        }
24.    }
```

Cet exemple utilise des transactions. Une transaction permet de regrouper des ordres SQL qui doivent être tous réussis ou tous annulés. Il y a quatre opérations à connaître :

- début d'une transaction : [**connexion.setAutoCommit(false)**] ;
- fin d'une transaction avec succès : [**connexion.commit()**]. Dans ce cas, toutes les opérations faites sur la BD lors de la transaction sont validées ;
- fin d'une transaction avec échec : [**connexion.rollback()**]. Dans ce cas, toutes les opérations faites sur la BD lors de la transaction sont annulées ;
- retour au mode [auto-commit] qui est le mode par défaut de l'API JDBC : [**connexion.setAutoCommit(true)**]. Dans ce mode, chaque ordre SQL fait l'objet d'une transaction. Ainsi si on fait deux insertions dont la deuxième échoue :
  - en mode [AutoCommit=true], la première insertion reste (elle a été validée par le 1er AutoCommit) ;
  - en mode [AutoCommit=false], la première insertion est annulée ;

Dans nos exemples, à chaque fois que se produit une exception, nous annulons la transaction dans la méthode [*doCatchException*] :

```
1.   private static void doCatchException(String title, Connection connexion, Throwable th) {
2.     // on affiche les msg d'erreur
3.     Static.show(title, Static.getErreursFromThrowable(th));
4.     // annulation transaction
5.     try {
6.         if (connexion != null) {
7.             connexion.rollback();
8.         }
9.     } catch (SQLException e2) {
10.            // on affiche les msg d'erreur
11.            Static.show("Erreur lors de l'annulation de la transaction", Static.getErreursFromThrowable(e2));
12.        }
13.  }
```

6.4.8 Création du contenu de la table des produits

La méthode [insert] crée le contenu de la table :

```
1. // ajout produits
2.   public static void insert() {
3.     Connection connexion = null;
4.     PreparedStatement ps = null;
5.     try {
6.         // ouverture connexion
7.         connexion = DriverManager.getConnection(url, user, passwd);
8.         // début transaction
9.         connexion.setAutoCommit(false);
10.            // on remplit la table
11.            ps = connexion.prepareStatement(insert);
12.            for (int i = 0; i < 10; i++) {
13.                // préparation
14.                int n = i + 1;
15.                ps.setInt(1, n);
16.                ps.setString(2, String.format("NOM%s", n));
17.                ps.setInt(3, n / 5 + 1);
18.                ps.setDouble(4, 100 * (1 + (double) i / 100));
19.                ps.setString(5, String.format("DESC%s", n));
20.                // exécution
21.                ps.executeUpdate();
22.            }
23.            // commit transaction
24.            connexion.commit();
25.            // retour au mode par défaut
26.            connexion.setAutoCommit(true);
27.        } catch (SQLException e1) {
28.            // on traite l'exception
29.            doCatchException("Les erreurs suivantes se sont produites à la création du contenu de la table", connexion, e1);
30.        } finally {
31.            // on traite le finally
32.            doFinally(null, null, connexion);
33.        }
34.    }
```

6.4.9 Affichage du contenu de la table des produits

La méthode [select] affiche le contenu de la table :

```
1.   // liste des produits
2.   private static void select() {
3.     Connection connexion = null;
4.     PreparedStatement ps = null;
5.     ResultSet rs = null;
6.     try {
7.         // ouverture connexion
8.         connexion = DriverManager.getConnection(url, user, passwd);
9.         // début transaction
10.            connexion.setAutoCommit(false);
11.            // on lit la table [PRODUITS]
12.            ps = connexion.prepareStatement(select);
13.            rs = ps.executeQuery();
14.            System.out.println("Liste des produits : ");
15.            while (rs.next()) {
16.                System.out.println(new Produit(rs.getInt(1), rs.getString(2), rs.getInt(3), rs.getDouble(4),
    rs.getString(5)));
17.            }
18.            // commit transaction
19.            connexion.commit();
20.            // retour au mode par défaut
21.            connexion.setAutoCommit(true);
22.        } catch (SQLException e1) {
23.            // on traite l'exception
24.            doCatchException("Les erreurs suivantes se sont produites à la lecture de la table", connexion, e1);
25.        } finally {
26.            // on traite le finally
27.            doFinally(null, null, connexion);
28.        }
29.  }
```

6.4.10 Mise à jour du contenu de la table

La méthode [update] met à jour certains produits :

```
1. // mise à jour de certains produits
2.   public static void update() {
3.     Connection connexion = null;
4.     PreparedStatement ps = null;
```

```
    try {
        // ouverture connexion
        connexion = DriverManager.getConnection(url, user, passwd);
        // début transaction
        connexion.setAutoCommit(false);
            // on met à jour la table
            ps = connexion.prepareStatement(update);
            // catégorie 1
            ps.setInt(1, 1);
            // exécution
            ps.executeUpdate();
            // commit transaction
            connexion.commit();
            // retour au mode par défaut
            connexion.setAutoCommit(true);
        } catch (SQLException e1) {
            // on traite l'exception
            doCatchException("Les erreurs suivantes se sont produites à la mise à jour du contenu de la table", connexion, e1);
        } finally {
            // on traite le finally
            doFinally(null, null, connexion);
        }
    }
```

6.4.11 Rôle de la transaction

La méthode [insert2] insère deux produits de même clé primaire dans la table, ce qui n'est pas possible. Comme on est dans une transaction, le première insertion va être annulée.

```
// ajout de  2 produits de mêmes clés primaires
  public static void insert2() {
    Connection connexion = null;
    PreparedStatement ps = null;
    try {
        // ouverture connexion
        connexion = DriverManager.getConnection(url, user, passwd);
        // début transaction
        connexion.setAutoCommit(false);
            // on ajoute 1 ligne
            ps = connexion.prepareStatement(insert2);
            // exécution
            ps.executeUpdate();
            // on ajoute la même ligne une 2ème fois donc avec la même clé primaire
            // l'insertion doit échouer et aucun des deux éléments ne doit être inséré à cause de la transaction
            ps.executeUpdate();
            // commit transaction
            connexion.commit();
            // retour au mode par défaut
            connexion.setAutoCommit(true);
        } catch (SQLException e1) {
            // on traite l'exception
            doCatchException("Les erreurs suivantes se sont produites lors de l'ajout", connexion, e1);
        } finally {
            // on traite le finally
            doFinally(null, null, connexion);
        }
    }
```

6.4.12 Résultats

Les résultats de l'exécution de la méthode [main] donne les résultats suivants :

```
Liste des produits :
{"id":1,"nom":"NOM1","categorie":1,"prix":100.0,"description":"DESC1"}
{"id":2,"nom":"NOM2","categorie":1,"prix":101.0,"description":"DESC2"}
{"id":3,"nom":"NOM3","categorie":1,"prix":102.0,"description":"DESC3"}
{"id":4,"nom":"NOM4","categorie":1,"prix":103.0,"description":"DESC4"}
{"id":5,"nom":"NOM5","categorie":2,"prix":104.0,"description":"DESC5"}
{"id":6,"nom":"NOM6","categorie":2,"prix":105.0,"description":"DESC6"}
{"id":7,"nom":"NOM7","categorie":2,"prix":106.0,"description":"DESC7"}
{"id":8,"nom":"NOM8","categorie":2,"prix":107.0,"description":"DESC8"}
 {"id":9,"nom":"NOM9","categorie":2,"prix":108.0,"description":"DESC9"}
 {"id":10,"nom":"NOM10","categorie":3,"prix":109.0,"description":"DESC10"}
 Liste des produits :
 {"id":1,"nom":"NOM1","categorie":1,"prix":110.0,"description":"DESC1"}
 {"id":2,"nom":"NOM2","categorie":1,"prix":111.0,"description":"DESC2"}
 {"id":3,"nom":"NOM3","categorie":1,"prix":112.0,"description":"DESC3"}
 {"id":4,"nom":"NOM4","categorie":1,"prix":113.0,"description":"DESC4"}
 {"id":5,"nom":"NOM5","categorie":2,"prix":104.0,"description":"DESC5"}
 {"id":6,"nom":"NOM6","categorie":2,"prix":105.0,"description":"DESC6"}
 {"id":7,"nom":"NOM7","categorie":2,"prix":106.0,"description":"DESC7"}
```

```
{"id":8,"nom":"NOM8","categorie":2,"prix":107.0,"description":"DESC8"}
{"id":9,"nom":"NOM9","categorie":2,"prix":108.0,"description":"DESC9"}
{"id":10,"nom":"NOM10","categorie":3,"prix":109.0,"description":"DESC10"}
Les erreurs suivantes se sont produites lors de l'ajout :
- Duplicate entry '100' for key 'PRIMARY'
Liste des produits :
{"id":1,"nom":"NOM1","categorie":1,"prix":110.0,"description":"DESC1"}
{"id":2,"nom":"NOM2","categorie":1,"prix":111.0,"description":"DESC2"}
{"id":3,"nom":"NOM3","categorie":1,"prix":112.0,"description":"DESC3"}
{"id":4,"nom":"NOM4","categorie":1,"prix":113.0,"description":"DESC4"}
{"id":5,"nom":"NOM5","categorie":2,"prix":104.0,"description":"DESC5"}
{"id":6,"nom":"NOM6","categorie":2,"prix":105.0,"description":"DESC6"}
{"id":7,"nom":"NOM7","categorie":2,"prix":106.0,"description":"DESC7"}
{"id":8,"nom":"NOM8","categorie":2,"prix":107.0,"description":"DESC8"}
{"id":9,"nom":"NOM9","categorie":2,"prix":108.0,"description":"DESC9"}
{"id":10,"nom":"NOM10","categorie":3,"prix":109.0,"description":"DESC10"}
Travail terminé
```

6.5 Utilisation d'une source de données de type [DataSource]

Nous allons reprendre l'application précédente en utilisant une source de données de type [javax.sql.DataSource] :

javax.sql

Interface DataSource

All Superinterfaces:
CommonDataSource, Wrapper

```
public interface DataSource
extends CommonDataSource, Wrapper
```

A factory for connections to the physical data source that this `DataSource` object represents. An alternative to the `DriverManager` facility, a `DataSource` object is the preferred means of getting a connection. An object that implements the `DataSource` interface will typically be registered with a naming service based on the Java™ Naming and Directory (JNDI) API.

The `DataSource` interface is implemented by a driver vendor. There are three types of implementations:

1. Basic implementation -- produces a standard `Connection` object
2. Connection pooling implementation -- produces a `Connection` object that will automatically participate in connection pooling. This implementation works with a middle-tier connection pooling manager.
3. Distributed transaction implementation -- produces a `Connection` object that may be used for distributed transactions and almost always participates in connection pooling. This implementation works with a middle-tier transaction manager and almost always with a connection pooling manager.

A `DataSource` object has properties that can be modified when necessary. For example, if the data source is moved to a different server, the property for the server can be changed. The benefit is that because the data source's properties can be changed, any code accessing that data source does not need to be changed.

A driver that is accessed via a `DataSource` object does not register itself with the `DriverManager`. Rather, a `DataSource` object is retrieved though a lookup operation and then used to create a `Connection` object. With a basic implementation, the connection obtained through a `DataSource` object is identical to a connection obtained through the `DriverManager` facility.

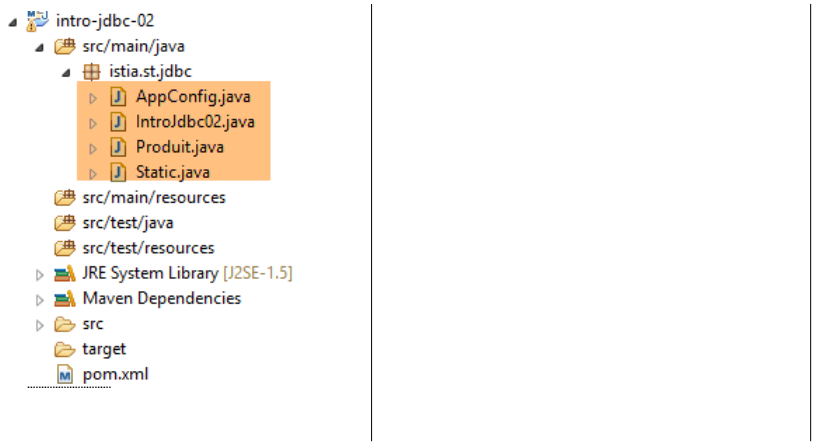
Since:
1.4

Nous allons utiliser une source de données implémentée par la classe [org.apache.tomcat.jdbc.pool.DataSource]. Cette classe utilise un pool de connexions ç-à-d un ensemble de connexions ouvertes :

- lorsque le pool est instancié, un certain nombre de connexions est ouvert avec la base de données. Ce nombre est configurable ;
- lorsque le code Java ouvre une connexion, celle-ci est fournie par le pool ;
- lorsque le code Java ferme une connexion, celle-ci est rendue au pool ;

Au final, les connexions ne sont ouvertes qu'une fois, ce qui améliore la performance d'accès à la base de données. La source de données sera définie dans une classe de configuration Spring

6.5.1 Le projet Eclipse



Le projet est un projet Maven défini par le fichier [pom.xml] suivant :

```
1. <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2.   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
3.   <modelVersion>4.0.0</modelVersion>
4.   <groupId>istia.st.jdbc</groupId>
5.   <artifactId>intro-jdbc-02</artifactId>
6.   <version>0.0.1-SNAPSHOT</version>
7.   <dependencies>
8.     <!-- MySQL -->
9.     <dependency>
10.          <groupId>mysql</groupId>
11.          <artifactId>mysql-connector-java</artifactId>
12.          <version>5.1.34</version>
13.      </dependency>
14.      <!-- bibliothèque jSON -->
15.      <dependency>
16.          <groupId>com.fasterxml.jackson.core</groupId>
17.          <artifactId>jackson-databind</artifactId>
18.          <version>2.5.1</version>
19.      </dependency>
20.      <!-- Spring -->
21.      <dependency>
22.          <groupId>org.springframework</groupId>
23.          <artifactId>spring-context</artifactId>
24.          <version>4.1.3.RELEASE</version>
25.      </dependency>
26.      <!-- Tomcat Jdbc -->
27.      <dependency>
28.          <groupId>org.apache.tomcat</groupId>
29.          <artifactId>tomcat-jdbc</artifactId>
30.          <version>8.0.20</version>
31.      </dependency>
32.  </dependencies>
33. </project>
```

- lignes 21-25 : dépendance sur Spring ;
- lignes 27-31 : dépendance sur la bibliothèque qui fournit la source de données ;

La classe de configuration de Spring [AppConfig] est la suivante :

```
1. package istia.st.jdbc;
2.
3. import org.apache.tomcat.jdbc.pool.DataSource;
4. import org.springframework.context.annotation.Bean;
5. import org.springframework.context.annotation.Configuration;
6.
7. @Configuration
8. public class AppConfig {
9.
10.    // constantes
11.    final static String URL = "jdbc:mysql://localhost:3306/dbIntroJdbc";
12.    final static String USER = "root";
13.    final static String PASSWD = "";
14.    final static String INSERT = "INSERT INTO PRODUITS(ID, NOM, CATEGORIE, PRIX, DESCRIPTION) VALUES (?, ?, ?, ?, ?)";
15.    final static String DELETE = "DELETE FROM PRODUITS";
16.    final static String SELECT = "SELECT ID, NOM, CATEGORIE, PRIX, DESCRIPTION FROM PRODUITS";
17.    final static String UPDATE = "UPDATE PRODUITS SET PRIX=PRIX*1.1 WHERE CATEGORIE=?";
```

```
18.    final static String INSERT2 = "INSERT INTO PRODUITS(ID, NOM, CATEGORIE, PRIX, DESCRIPTION) VALUES (100,'X',1,1,'x')";
19.    final static String DRIVER_CLASSNAME = "com.mysql.jdbc.Driver";
20.
21.    @Bean
22.    public DataSource dataSource() {
23.        // source de données TomcatJdbc
24.        DataSource dataSource = new DataSource();
25.        // configuration accès JDBC
26.        dataSource.setDriverClassName(DRIVER_CLASSNAME);
27.        dataSource.setUsername(USER);
28.        dataSource.setPassword(PASSWD);
29.        dataSource.setUrl(URL);
30.        // une connexion ouverte initialement
31.        dataSource.setInitialSize(1);
32.        // résultat
33.        return dataSource;
34.    }
35. }
```

- lignes 11-19 : les constantes précédemment définies dans [IntroJdbc01] ont migré dans [AppConfig] ;
- lignes 31-34 : le bean Spring définissant la source de données ;
- ligne 24 : création de la source de données encore non configurée ;
- lignes 26-29 : les informations qui permettent à la source de données de se connecter à la base de données ;
- ligne 31 ; crée un pool de 1 connexion. On n'en a pas besoin de plus ici. Il n'y a jamais plusieurs connexions simultanées ;

6.5.2 La classe principale

La classe principale [IntroJdbc02] est la suivante :

```
1. package istia.st.jdbc;
2.
3. import java.sql.Connection;
4. import java.sql.PreparedStatement;
5. import java.sql.ResultSet;
6. import java.sql.SQLException;
7.
8. import javax.sql.DataSource;
9.
10.  import org.springframework.context.annotation.AnnotationConfigApplicationContext;
11.
12.  public class IntroJdbc02 {
13.    // source de données
14.    private static DataSource dataSource;
15.
16.    public static void main(String[] args) {
17.        // récupération du contexte Spring
18.        AnnotationConfigApplicationContext ctx = new AnnotationConfigApplicationContext(AppConfig.class);
19.        // récupération de la source de données
20.        dataSource = ctx.getBean(DataSource.class);
21.        // on vide la table [PRODUITS]
22.        delete();
23.  ...
24.        // fini
25.        ctx.close();
26.        System.out.println("Travail terminé");
27.    }
28.
29.    // liste des produits
30.    private static void select() {
31.        Connection connexion = null;
32.        PreparedStatement ps = null;
33.        ResultSet rs = null;
34.        try {
35.            // ouverture connexion
36.            connexion = dataSource.getConnection();
37.            // début transaction
38.            connexion.setAutoCommit(false);
39.  ...
40.        } catch (SQLException e1) {
41.            // on traite l'exception
42.            doCatchException("Les erreurs suivantes se sont produites à la lecture de la table", connexion, e1);
43.        } finally {
44.            // on traite le finally
45.            doFinally(null, null, connexion);
46.        }
47.    }
48.  ...
49. }
```

- ligne 14 : la source de données. On notera qu'elle est de type [javax.sql.DataSource] qui est une interface ;
- ligne 18 : instanciation des objets Spring ;

- ligne 20 : obtention d'une référence sur la source de données. On notera qu'à aucun moment on ne cite la classe réellement utilisée. Ainsi ici, rien ne laisse supposer qu'on utilise une implémentation [TomcatJdbc] ;
- ligne 36 : obtention d'une connexion ouverte ;
- le reste du code est identique à celui de la classe [IntroJdbc01] ;

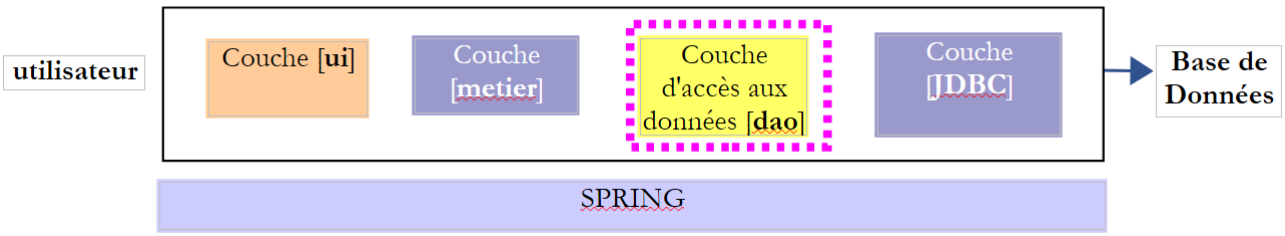
6.6 Conclusion

On trouvera davantage d'informations sur la gestion des bases de données dans le document [Exploiter une base relationnelle avec l'écosystème Spring].

7 [TD] : Implémentation de la couche [DAO] du TD avec l'API JDBC

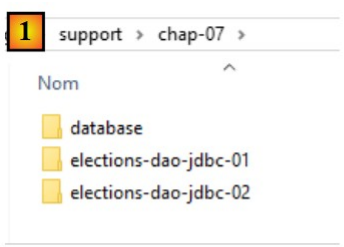
Mots clés : bases de données relationnelles, API JDBC, SQLException.

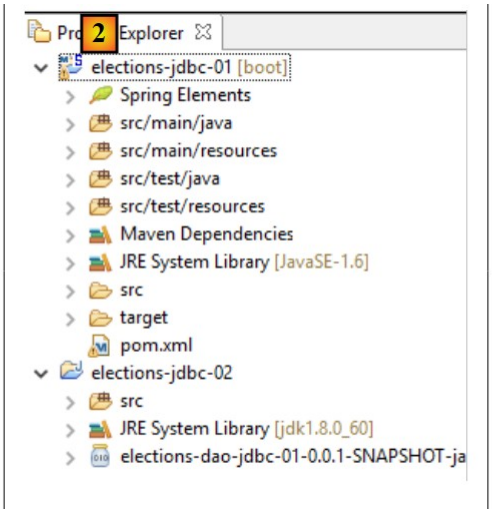
Revenons sur l'architecture en couches de notre application :

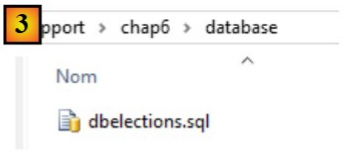


Les données nécessaires à l'élection sont enregistrées dans une base de données MySQL [dbelections]

7.1 Support







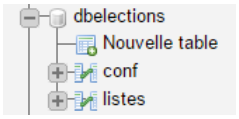
Le dossier [support / chap-07] [1] contient :

- les projets Eclipse de ce chapitre [2] ;
- le script SQL de création de la base MySQL [dbelections] [3] ;

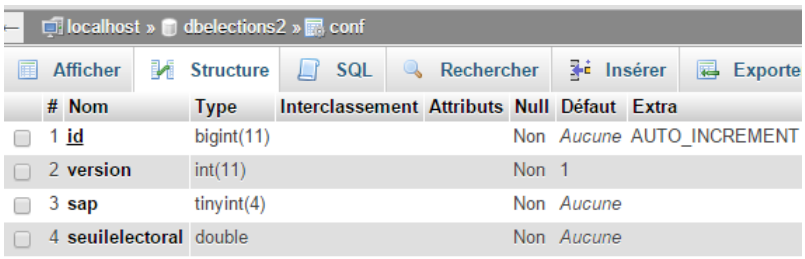
7.2 La base de données [dbelections]

Travail à faire : créez la base MySQL [dblelections] en suivant la procédure suivie au paragraphe 6.4.2, page 85.

La base [dbelections] est une base MySQL avec deux tables :

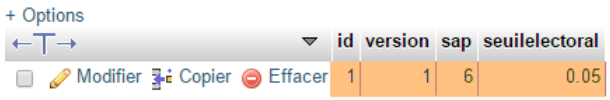


La table [conf] contient les informations de configuration de l'élection :

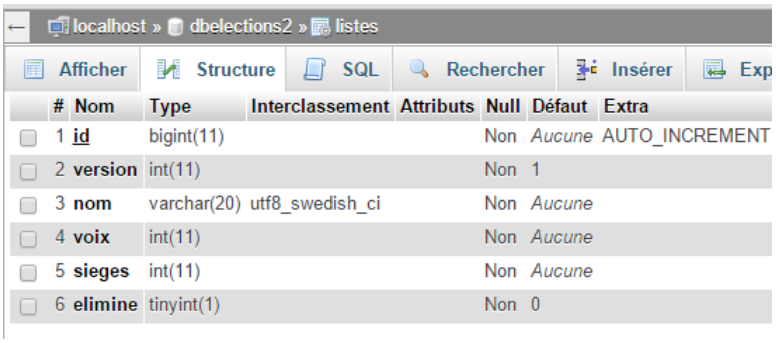


- [id] : clé primaire auto-incrémentée ;
- [version] : n° de version de l'enregistrement - peut être ignoré ici ;
- [sap] : nombre de sièges à pourvoir ;
- [seuilelectoral] : la barre au-dessous de laquelle une liste est éliminée ;

Son contenu est le suivant :



La table [listes] contient les listes candidates de l'élection :



- [id] : clé primaire auto-incrémentée ;
- [version] : n° de version de l'enregistrement - peut être ignoré ici ;
- [nom] : nom de la liste ;
- [voix] : voix de la liste - n'est connu qu'après saisie de l'utilisateur dans la couche [présentation] ;
- [sieges] : nombre de sièges obtenus - n'est connu qu'après calcul de la couche [métier] ;
- [elimine] : à 1 si la liste est éliminée, à 0 sinon - n'est connu qu'après calcul de la couche [métier] ;

Le contenu de la table [listes] est le suivant :

id	version	nom	voix	sieges	elimine
1	21	A	10	1	0
2	22	B	20	2	0
3	21	C	30	3	0
4	13	D	40	3	0
5	17	E	50	6	0
6	18	F	60	1	0
7	19	G	70	2	0

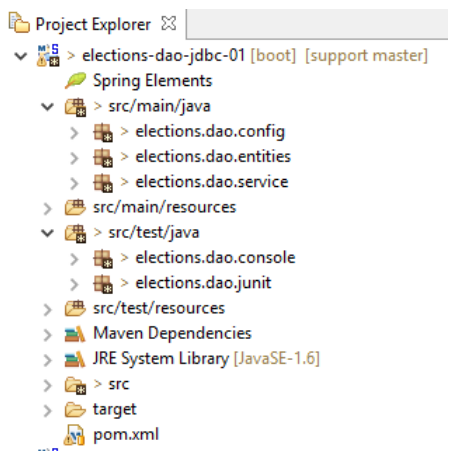
Le script SQL pour générer la base [dbelections] s'appelle [dbelections.sql] et est sur le serveur. Son code est le suivant :

```
-- phpMyAdmin SQL Dump
-- version 4.0.4
-- http://www.phpmyadmin.net
--
-- Client: localhost
-- Généré le: Mer 11 Mars 2015 à 12:20
-- Version du serveur: 5.6.12-log
-- Version de PHP: 5.4.12

SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
SET time_zone = "+00:00";


/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8 */;

--
-- Base de données: `dbelections`
--
CREATE DATABASE IF NOT EXISTS `dbelections` DEFAULT CHARACTER SET utf8 COLLATE utf8_swedish_ci;
USE `dbelections`;

-- --------------------------------------------------------

--
-- Structure de la table `conf`
--

CREATE TABLE IF NOT EXISTS `conf` (
  `id` bigint(11) NOT NULL AUTO_INCREMENT,
  `version` int(11) NOT NULL DEFAULT '1',
  `sap` tinyint(4) NOT NULL,
  `seuilelectoral` double NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB  DEFAULT CHARSET=utf8 COLLATE=utf8_swedish_ci AUTO_INCREMENT=2 ;

--
-- Contenu de la table `conf`
--

INSERT INTO `conf` (`id`, `version`, `sap`, `seuilelectoral`) VALUES
(1, 1, 6, 0.05);

-- --------------------------------------------------------

--
-- Structure de la table `listes`
--

CREATE TABLE IF NOT EXISTS `listes` (
  `id` bigint(11) NOT NULL AUTO_INCREMENT,
  `version` int(11) NOT NULL DEFAULT '1',
  `nom` varchar(20) COLLATE utf8_swedish_ci NOT NULL,
  `voix` int(11) NOT NULL,
  `sieges` int(11) NOT NULL,
```

```
  `elimine` tinyint(1) NOT NULL DEFAULT '0',
  PRIMARY KEY (`id`),
  UNIQUE KEY `nom` (`nom`)
) ENGINE=InnoDB  DEFAULT CHARSET=utf8 COLLATE=utf8_swedish_ci AUTO_INCREMENT=8 ;

--
-- Contenu de la table `listes`
--

INSERT INTO `listes` (`id`, `version`, `nom`, `voix`, `sieges`, `elimine`) VALUES
(1, 21, 'A', 10, 1, 0),
(2, 22, 'B', 20, 2, 0),
(3, 21, 'C', 30, 3, 0),
(4, 13, 'D', 40, 3, 0),
(5, 17, 'E', 50, 6, 0),
(6, 18, 'F', 60, 1, 0),
(7, 19, 'G', 70, 2, 0);

/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
```

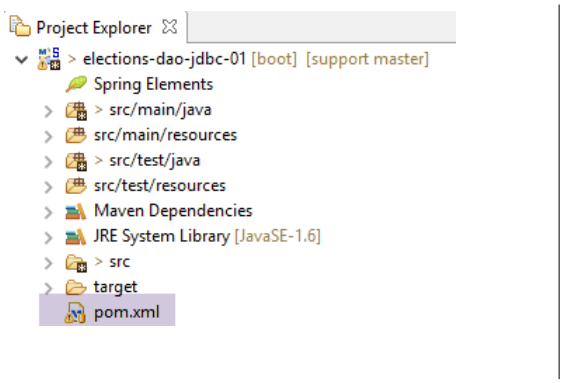
7.3 Le projet Eclipse

Le projet Eclipse de la couche [DAO] sera le suivant :



- le package [elections.dao.entities] contient les objets manipulés par la couche [DAO] ;
- le package [elections.dao.service] contient l'implémentation de la couche [DAO] ;
- le package [elections.dao.config] contient la configuration de la couche [DAO]
- le package [elections.dao.junit] contient une classe de test JUnit du projet ;
- le package [elections.dao.console] contient une classe exécutable de test ;

7.4 Configuration du projet Maven



Le fichier [pom.xml] qui configure le projet Maven est le suivant :

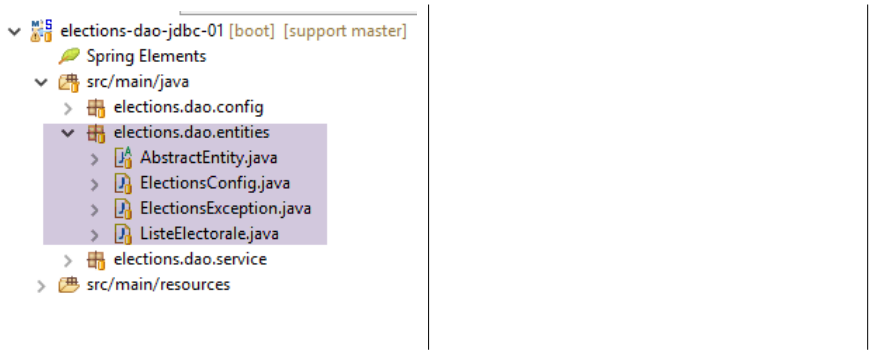
```
1. <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2.   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
3.   <modelVersion>4.0.0</modelVersion>
4.   <groupId>istia.st.elections</groupId>
5.   <artifactId>elections-dao-jdbc-01</artifactId>
6.   <version>0.0.1-SNAPSHOT</version>
7.
8.   <parent>
9.     <groupId>org.springframework.boot</groupId>
10.       <artifactId>spring-boot-starter-parent</artifactId>
11.       <version>1.2.7.RELEASE</version>
12.   </parent>
13.
14.   <properties>
15.       <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
16.       <java.version>1.8</java.version>
17.   </properties>
18.
19.   <dependencies>
20.       <!-- MySQL -->
21.       <dependency>
22.           <groupId>mysql</groupId>
23.           <artifactId>mysql-connector-java</artifactId>
24.       </dependency>
25.       <!-- Spring -->
26.       <dependency>
27.           <groupId>org.springframework</groupId>
28.           <artifactId>spring-context</artifactId>
29.       </dependency>
30.       <!-- Tomcat Jdbc -->
31.       <dependency>
32.           <groupId>org.apache.tomcat</groupId>
33.           <artifactId>tomcat-jdbc</artifactId>
34.       </dependency>
35.       <!-- bibliothèque jSON -->
36.       <dependency>
37.           <groupId>com.fasterxml.jackson.core</groupId>
38.           <artifactId>jackson-databind</artifactId>
39.       </dependency>
40.       <!-- Spring Boot -->
41.       <dependency>
42.           <groupId>org.springframework.boot</groupId>
43.           <artifactId>spring-boot</artifactId>
44.           <scope>test</scope>
45.       </dependency>
46.       <!-- Spring Boot Test -->
47.       <dependency>
48.           <groupId>org.springframework.boot</groupId>
49.           <artifactId>spring-boot-starter-test</artifactId>
50.           <scope>test</scope>
51.       </dependency>
52.       <!-- Spring Boot Logging -->
53.       <dependency>
54.           <groupId>org.springframework.boot</groupId>
55.           <artifactId>spring-boot-starter-logging</artifactId>
56.       </dependency>
57.   </dependencies>
58.
59.   <!-- plugins -->
60.   <build>
61.       <plugins>
```

```
62.          <plugin>
63.              <artifactId>maven-assembly-plugin</artifactId>
64.              <configuration>
65.                  <archive>
66.                      <manifest>
67.                          <mainClass>config.AppConfig</mainClass>
68.                      </manifest>
69.                  </archive>
70.                  <descriptorRefs>
71.                      <descriptorRef>jar-with-dependencies</descriptorRef>
72.                  </descriptorRefs>
73.              </configuration>
74.          </plugin>
75.          <!-- pour l'installation de l'artifact du projet dans le dépôt local Maven -->
76.          <plugin>
77.              <groupId>org.apache.maven.plugins</groupId>
78.              <artifactId>maven-surefire-plugin</artifactId>
79.              <version>2.18.1</version>
80.          </plugin>
81.      </plugins>
82.  </build>
83. </project>
```

Ce fichier est analogue à celui décrit au paragraphe 6.5.1, page 95. On y a apporté les modifications suivantes :

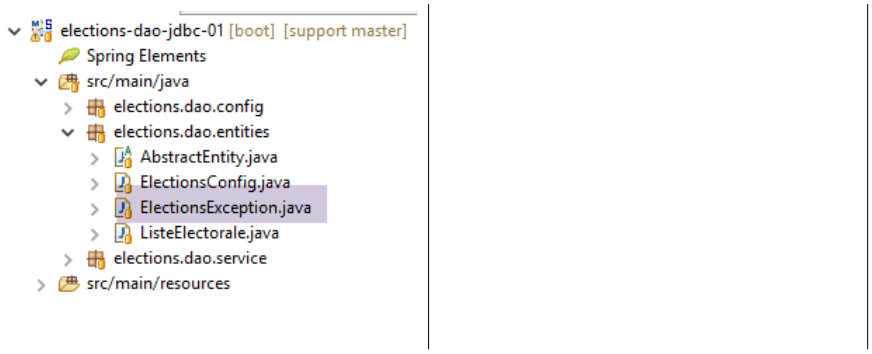
- lignes 8-12 : on a défini un projet Maven parent. Le projet [`spring-boot-starter-parent`] (ligne 10) définit un très grand nombre de dépendances avec leurs versions. Lorsqu'on utilise l'une d'elles (lignes 19-57), il est alors inutile d'en préciser la version, celle-ci étant définie dans le projet Maven parent ;
- lignes 40-51 : dépendances nécessaires à la classe de test du projet. Ces dépendances ont l'attribut [`<scope>test</scope>`] qui signifie qu'elles ne sont nécessaires que pour les classes du dossier [src / test / java]. Ces dépendances ne seront pas incluses dans l'archive finale du projet ;
- lignes 53-56 : la bibliothèque [spring-boot-starter-logging] sera utilisée par Spring pour faire des logs à l'écran ;
- lignes 14-17 : des propriétés de configuration Maven :
  - ligne 15 : indique que les fichiers source sont codés en UTF-8 ;
  - ligne 16 : indique que le compilateur à utiliser doit être de niveau 1.8 ;

7.5 Les entités de la couche [DAO]



- [ElectionsConfig] est le modèle objet associé à une ligne de la table [CONF] ;
- [ListeElectorale] est le modèle objet associé à une ligne de la table [LISTES] ;
- [AbstactEntity] est la classe parent des deux classes précédentes. Elle factorise les champs [id, version] communs aux deux classes ;
- [ElectionsException] est une classe d'exception ;

7.5.1 La classe [ElectionsException]



La classe [ElectionsException] a été décrite au paragraphe 4.3, page 39. Nous redonnons son code :

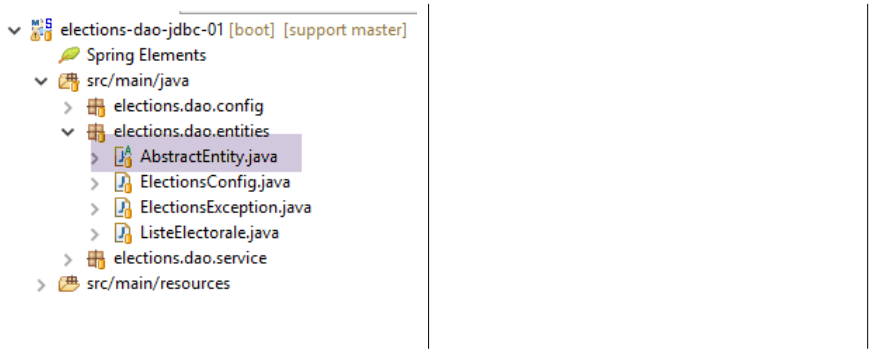
```
1. package ...;
2.
3. import java.io.Serializable;
4. import java.util.ArrayList;
5. import java.util.List;
6.
7. // classe d'exception pour l'application Elections
8. // l'exception est non contrôlée
9.
10.  public class ElectionsException extends RuntimeException implements Serializable {
11.
12.    // serial ID
13.    private static final long serialVersionUID = 1L;
14.
15.    // champs locaux
16.    private int code;
17.    private List<String> erreurs;
18.
19.    // constructeurs
20.    public ElectionsException() {
21.        super();
22.    }
23.
24.    public ElectionsException(int code, Throwable e) {
25.        // parent
26.        super(e);
27.        // local
28.        this.code = code;
29.        this.erreurs = getErreursForException(e);
30.    }
31.
32.    public ElectionsException(int code, String message, Throwable e) {
33.        // parent
34.        super(message,e);
35.        // local
36.        this.code = code;
37.        this.erreurs = getErreursForException(e);
38.    }
39.
40.    public ElectionsException(int code, String message) {
41.        // parent
42.        super(message);
43.        // local
44.        this.code = code;
45.        List<String> erreurs = new ArrayList<>();
46.        erreurs.add(message);
47.        this.erreurs = erreurs;
48.    }
49.
50.    public ElectionsException(int code, List<String> erreurs) {
51.        // parent
52.        super();
53.        // local
54.        this.code = code;
55.        this.erreurs = erreurs;
56.    }
57.
58.    // liste des messages d'erreur d'une exception
59.    private List<String> getErreursForException(Throwable th) {
60.        // on récupère la liste des messages d'erreur de l'exception
61.        Throwable cause = th;
62.        List<String> erreurs = new ArrayList<>();
63.        while (cause != null) {
64.            // on récupère le message seulement s'il est !=null et non blanc
65.            String message = cause.getMessage();
```

```
66.           if (message != null) {
67.               message = message.trim();
68.               if (message.length() != 0) {
69.                   erreurs.add(message);
70.               }
71.           }
72.           // cause suivante
73.           cause = cause.getCause();
74.       }
75.       return erreurs;
76.   }
77.
78.   // getters et setters
79. ...
80. }
```

Un objet [ElectionsException] est caractérisé par deux informations :

- ligne 16 : un code d'erreur ;
- ligne 17 : une liste de messages d'erreur associés à la pile des exceptions qui se sont produites ;
- la classe a 5 constructeurs (lignes 20, 24, 32, 40, 50) ;
- lignes 59-76 : la méthode [getErreursForException] permet de récupérer les messages d'erreur de la pile d'exceptions ;

7.5.2 La classe [AbstractEntity]



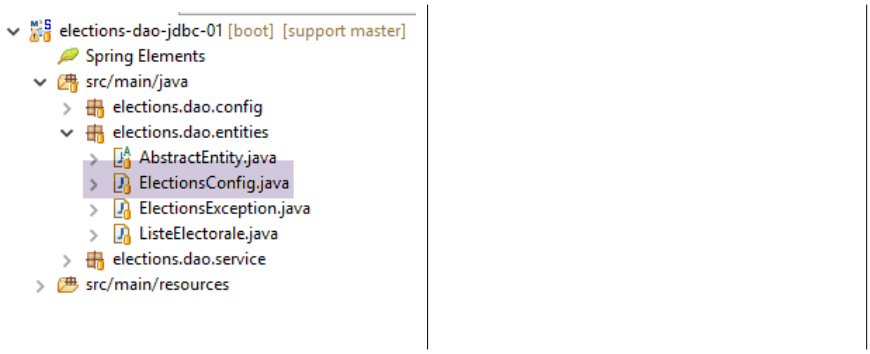
La classe [AbstractEntity] est la suivante :

```
1. package elections.dao.entities;
2.
3. import java.io.Serializable;
4.
5. import com.fasterxml.jackson.core.JsonProcessingException;
6. import com.fasterxml.jackson.databind.ObjectMapper;
7.
8. public abstract class AbstractEntity implements Serializable {
9.   private static final long serialVersionUID = 1L;
10.
11.   // champs
12.   protected Long id;
13.   protected Long version;
14.
15.   // constructeurs
16.   public AbstractEntity() {
17.
18.   }
19.
20.   public AbstractEntity(Long id, Long version) {
21.       this.id = id;
22.       this.version = version;
23.   }
24.
25.   // signature
26.   public String toString() {
27.       try {
28.           return new ObjectMapper().writeValueAsString(this);
29.       } catch (JsonProcessingException e) {
30.           e.printStackTrace();
31.           return null;
32.       }
33.   }
34.
35.   // getters et setters
36. ...
37. }
```

Elle mémorise les champs [ID, NOM] des lignes des tables [CONF] et [LISTES] (lignes 8-9).

- ligne 8 : la classe a l'attribut [Abstract] indiquant qu'elle ne peut être instanciée. Elle ne peut être que dérivée ;
- lignes 26-33 : signature jSON de l'objet ;
- ligne 28 : on retourne la chaîne jSON de [this]. Si au moment de l'exécution, [this] représente un objet dérivé de [AbstactEntity], c'est la chaîne jSON de l'objet dérivé qui est obtenue. Les classes dérivées n'auront ainsi pas besoin de définir une méthode [toString]. Celle de la classe parent est suffisante ;

7.5.3 La classe [ElectionsConfig]



La classe [ElectionsConfig] est la suivante :

```
1. package elections.dao.entities;
2.
3.
4. public class ElectionsConfig extends AbstractEntity {
5.
6.   private static final long serialVersionUID = 1L;
7.   // champs
8.   private int nbSiegesAPourvoir;
9.   private double seuilElectoral;
10.
11.    // constructeurs
12.    public ElectionsConfig() {
13.    }
14.
15.    public ElectionsConfig(Long id, Long version, int nbSiegesAPourvoir, double seuilElectoral) {
16.        // parent
17.        super(id, version);
18.        // champs locaux
19.        this.nbSiegesAPourvoir = nbSiegesAPourvoir;
20.        this.seuilElectoral = seuilElectoral;
21.    }
22.
23.    // getters et setters
24. ...
25. }
```

- ligne 4 : la classe étend la classe [AbstractEntity] ;
- lignes 8-9 : mémorisent les informations des colonnes [SAP, SEUILELECTORAL] de la table [CONF] ;

7.5.4 La classe [ListeElectorale]

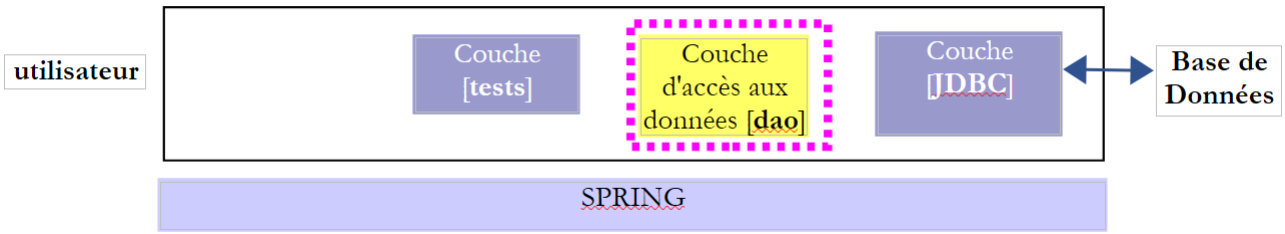
La classe [ListeElectorale] est la suivante :

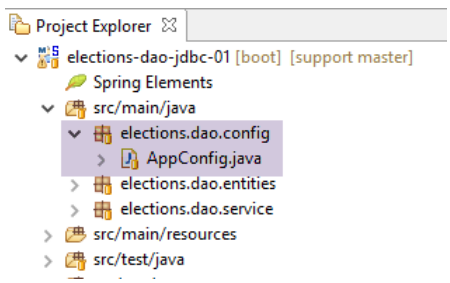
```
1. package elections.dao.entities;
2.
3. public class ListeElectorale extends AbstractEntity {
4.
5.   // champs
6.   private String nom;
7.   private int voix;
8.   private int sieges;
9.   private boolean elimine;
10.
11.    // constructeurs
12.    public ListeElectorale() {
13.    }
14.
15.    public ListeElectorale(String nom, int voix, int sieges, boolean elimine) {
16.        // parent
17.        super();
18.        // champs locaux
19.        initNom(nom);
20.        initVoix(voix);
21.        initSieges(sieges);
22.        this.elimine=elimine;
23.    }
24.
25.    public ListeElectorale(Long id, Long version, String nom, int voix, int sieges, boolean elimine) {
26.        // parent
27.        super(id, version);
28.        // champs locaux
29.        initNom(nom);
30.        initVoix(voix);
31.        initSieges(sieges);
32.        this.elimine=elimine;
33.    }
34.
35.    // méthodes privées
36.    private void initNom(String nom) {
37.        this.nom = nom.trim();
38.        if ("".equals(nom)) {
39.            throw new ElectionsException(10, "Le nom ne peut être vide");
40.        }
41.    }
42.
43.    private void initVoix(int voix) {
44.        this.voix = voix;
45.        if (voix < 0) {
46.            throw new ElectionsException(11, "Le nombre de voix ne peut être <0");
47.        }
48.    }
49.
50.    private void initSieges(int sieges) {
51.        this.sieges = sieges;
52.        if (sieges < 0) {
53.            throw new ElectionsException(12, "Le nombre de sièges ne peut être <0");
54.        }
55.    }
56.
57.    // getters et setters
58.
59.    public String getNom() {
60.        return nom;
61.    }
62.
63.    public void setNom(String nom) {
64.        initNom(nom);
65.    }
66.
67.    public int getVoix() {
68.        return voix;
69.    }
70.
71.    public void setVoix(int voix) {
72.        initVoix(voix);
73.    }
74.
75.    public int getSieges() {
76.        return sieges;
77.    }
78.
79.    public void setSieges(int sieges) {
80.        initSieges(sieges);
81.    }
82.
83.    public boolean isElimine() {
84.        return elimine;
```

```
85.    }
86.
87.    public void setElimine(boolean elimine) {
88.        this.elimine = elimine;
89.    }
90.
91. }
```

- ligne 3 : la classe étend la classe [AbstractEntity] ;
- lignes 6-9 : la classe mémorise les colonnes [NOM, VOIX, SIEGES, ELIMINE] de la table [LISTES] ;

7.6 Configuration Spring de la couche [DAO]





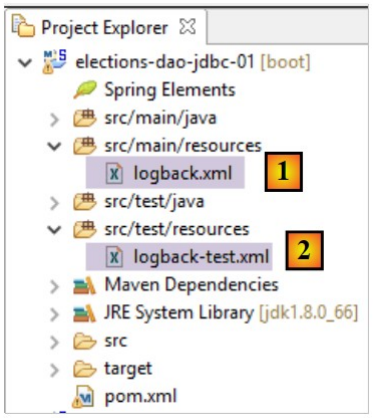
La classe [AppConfig] est une classe de configuration Spring qui configure l'accès à la base de données de la façon suivante :

```
1. package elections.dao.config;
2.
3. import org.apache.tomcat.jdbc.pool.DataSource;
4. import org.springframework.cache.CacheManager;
5. import org.springframework.cache.annotation.EnableCaching;
6. import org.springframework.cache.concurrent.ConcurrentMapCacheManager;
7. import org.springframework.context.annotation.Bean;
8. import org.springframework.context.annotation.ComponentScan;
9. import org.springframework.context.annotation.Configuration;
10.
11. @ComponentScan(basePackages = { "elections.dao.service" })
12. @EnableCaching
13. public class AppConfig {
14.
15.   // constantes
16.   public final static String URL = "jdbc:mysql://localhost:3306/dbelections";
17.   public final static String USER = "root";
18.   public final static String PASSWD = "";
19.   public final static String DRIVER_CLASSNAME = "com.mysql.jdbc.Driver";
20.   public final static String SELECT_LISTES = "SELECT ID, VERSION, NOM, VOIX, SIEGES, ELIMINE FROM LISTES";
21.   public final static String SELECT_CONF = "SELECT ID, VERSION, SAP, SEUILELECTORAL, SAP FROM CONF";
22.   public final static String UPDATE_LISTES = "UPDATE LISTES SET VOIX=?, SIEGES=?, ELIMINE=? WHERE ID=?";
23.
24.   @Bean
25.   public DataSource dataSource() {
26.       // source de données TomcatJdbc
27.       DataSource dataSource = new DataSource();
28.       // configuration accès JDBC
29.       dataSource.setDriverClassName(DRIVER_CLASSNAME);
30.       dataSource.setUsername(USER);
31.       dataSource.setPassword(PASSWD);
32.       dataSource.setUrl(URL);
```

```
33.         // une connexion ouverte initialement
34.         dataSource.setInitialSize(1);
35.         // résultat
36.         return dataSource;
37.     }
38.
39.     @Bean
40.     public CacheManager cacheManager() {
41.         return new ConcurrentMapCacheManager("electionsConfig");
42.     }
43. }
```

- lignes 25-38 : l'accès à la BD se fera via une source de données [tomcat-jdbc]. Ce type de source a été utilisée et expliquée au paragraphe 6.5, page 94 ;
- lignes 17-23 : un ensemble de constantes statiques accessibles par toutes les classes du projet ;
- ligne 13 : l'annotation [@Configuration] fait de la classe [AppConfig] une classe de configuration Spring ;
- ligne 11 : l'annotation [@ComponentScan] indique les packages où on peut trouver des objets Spring. Ici, nous allons définir un objet Spring dans le package [dao]. L'annotation [@ComponentScan] fait de la classe une classe de configuration qui nous épargne de mettre l'annotation [@Configuration] ;
- la ligne 12 active la gestion d'un cache. Le principe est la suivant :
  - on met l'annotation [@Cacheable('nom_du_cache')] à une méthode M. Le 'nom_du-cache' est le nom utilisé ligne 41 ;
  - lorsque la méthode M est appelée la 1ère fois, ses résultats sont rendus et également mis dans le cache nommé par l'annotation ;
  - lorsque la méthode M est appelée les fois suivantes avec les mêmes paramètres que la 1ère fois, elle n'est pas exécutée et Spring se contente de rendre les valeurs mises en cache ;

7.7 Configuration des logs



Les bibliothèques de logs sont définies par la dépendance suivante dans [pom.xml] :

```
1.     <!-- Spring Boot Logging -->
2.     <dependency>
3.         <groupId>org.springframework.boot</groupId>
4.         <artifactId>spring-boot-starter-logging</artifactId>
5. </dependency>
```

Cette dépendance amène les bibliothèques suivantes :

C'est la bibliothèque [logback] qui va assurer les logs. Elle est configurée par deux fichiers :

- [logback.xml] pour la branche principale du code ;
- [logback-test.xml] pour la branche de test du code. En l'absence de ce fichier, le fichier précédent est alors utilisé ;

Ces deux fichiers doivent se trouver dans le [Classpath] du projet. Pour cette raison, ils sont placés dans le dossier :
- [src / main / ressources] pour la branche principale du code ;
- [src / test / ressources] pour la branche de test du code ;

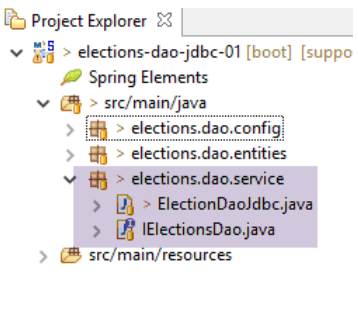
Le contenu des fichiers est ici le même :

```
1. <configuration>
2.
3.   <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
4.     <!-- encoders are  by default assigned the type
5.          ch.qos.logback.classic.encoder.PatternLayoutEncoder -->
6.     <encoder>
7.       <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>
8.     </encoder>
9.   </appender>
10.
11.    <!-- contrôle niveau des logs -->
12.    <root level="info"> <!-- info, debug, warn -->
13.      <appender-ref ref="STDOUT" />
14.    </root>
15.  </configuration>
```

Tout se passe ligne 12 où on fixe le niveau d'information désiré :
- [debug] : le niveau le plus détaillé ;
- [off] : pas de logs ;
- [info] : le niveau normal des logs ;
- [warn] : idem niveau [info] plus les messages d'avertissement (warning). Ces messages indiquent une possibilité d'erreur ;

Passez en mode [debug] dès que des erreurs apparaissent à l'exécution.

7.8 Implémentation de la couche [DAO]



L'interface [IDao] de la couche [DAO] est la suivante :

```
1. package istia.st.elections.webapi.client.dao;
2.
3. import istia.st.elections.webapi.client.entities.ElectionsConfig;
4. import istia.st.elections.webapi.client.entities.ListeElectorale;
5.
```

```
6. public interface IDao {
7.   // configuration des élections
8.   public ElectionsConfig getElectionsConfig();
9.
10.    // les listes en compétition
11.    public ListeElectorale[] getListesElectorales();
12.
13.    // sauvegarde des résultats de l'élection
14.    public void setListesElectorales(ListeElectorale[] listesElectorales);
15.
16.  }
```

Le squelette de la classe [ElectionsDaoJdbc] implémentant la couche [dao] avec une base de données sera le suivant :

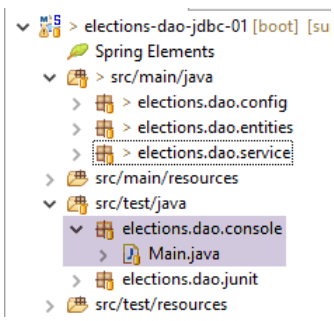
```
1. package elections.dao.service;
2.
3. import java.sql.Connection;
4. import java.sql.PreparedStatement;
5. import java.sql.ResultSet;
6. import java.sql.SQLException;
7.
8. import org.apache.tomcat.jdbc.pool.DataSource;
9. import org.springframework.beans.factory.annotation.Autowired;
10.  import org.springframework.cache.annotation.Cacheable;
11.  import org.springframework.stereotype.Component;
12.
13.  import elections.dao.entities.ElectionsConfig;
14.  import elections.dao.entities.ElectionsException;
15.  import elections.dao.entities.ListeElectorale;
16.
17.  @Component
18.  @SuppressWarnings("unused")
19.  public class ElectionDaoJdbc implements IElectionsDao {
20.
21.    @Autowired
22.    private DataSource dataSource;
23.
24.    @Cacheable("electionsConfig")
25.    // obtention conf de l'élection
26.    public ElectionsConfig getElectionsConfig() {
27.        throw new RuntimeException("[getElectionsConfig] not yet implemented");
28.    }
29.
30.    // obtention des listes
31.    public ListeElectorale[] getListesElectorales() {
32.        throw new RuntimeException("[getListesElectorales] not yet implemented");
33.    }
34.
35.    // modification des listes [voix, sieges, elimine]
36.    public void setListesElectorales(ListeElectorale[] listesElectorales) {
37.        throw new RuntimeException("[setListesElectorales] not yet implemented");
38.    }
39.
40.    // -------------------- méthodes privées
41.
42.    // gestion du finally
43.    private ElectionsException doFinally(int code, ResultSet rs, PreparedStatement ps, Connection connexion) {
44.        // fermeture ResultSet
45.        if (rs != null) {
46.            try {
47.                rs.close();
48.            } catch (SQLException e1) {
49.
50.            }
51.        }
52.        // fermeture [PreparedStatement]
53.        if (ps != null) {
54.            try {
55.                ps.close();
56.            } catch (SQLException e2) {
57.
58.            }
59.        }
60.        // fermer la connexion
61.        if (connexion != null) {
62.            try {
63.                connexion.close();
64.            } catch (SQLException e3) {
65.                // on retourne une [ElectionsException]
66.                return new ElectionsException(code, e3);
67.            }
68.        }
69.        // pas d'exception
70.        return null;
71.    }
```

```
72.
73.    // gestion du catch
74.    private ElectionsException doCatchException(int code1, int code2, Connection connexion, Throwable th) {
75.        // on génère une [ElectionsException]
76.        ElectionsException ex1 = new ElectionsException(code1, th);
77.        // annulation transaction
78.        try {
79.            if (connexion != null) {
80.                connexion.rollback();
81.            }
82.        } catch (SQLException e2) {
83.        }
84.        // on retourne l'exception
85.        return ex1;
86.    }
87. }
```

- ligne 24 : l'annotation [@Cacheable] est une annotation Spring qui demande de mettre en mémoire ('cacher') les résultats de la méthode [getElectionsConfig]. Il est possible de faire cela ici parce que le contenu de la table [CONF] ne change jamais. Il ne serait pas possible de mettre cette annotation sur la méthode [getListesElectorales] car le contenu de la table [LISTES] change au cours du temps ;
- les méthodes [doCatchException] et [doFinally] retournent un type [ElectionsException]. La méthode [doFinally] rend un pointeur *null* si la libération des ressources s'est faite sans erreurs ;

Travail à faire : écrivez la classe [ElectionDaoJdbc] en vous inspirant de la classe [IntroJdbc02] étudiée au paragraphe 6.5.2, page 96.

7.9 La classe de test [Main]



La classe [Main] est la suivante :

```
1. package elections.dao.console;
2.
3. import java.util.ArrayList;
4. import java.util.List;
5.
6. import org.springframework.context.annotation.AnnotationConfigApplicationContext;
7.
8. import elections.dao.config.AppConfig;
9. import elections.dao.entities.ElectionsConfig;
10.  import elections.dao.entities.ElectionsException;
11.  import elections.dao.entities.ListeElectorale;
```

```
12. import elections.dao.service.IElectionsDao;
13.
14. public class Main {
15.
16.   // source de données
17.   private static IElectionsDao dao;
18.
19.   public static void main(String[] args) {
20.       // on récupère la référence de la couche [DAO] après avoir instancié le contexte Spring
21.       try (AnnotationConfigApplicationContext ctx = new AnnotationConfigApplicationContext(AppConfig.class)) {
22.           // récupération de la source de données
23.           dao = ctx.getBean(IElectionsDao.class);
24.       } catch (Exception e) {
25.           System.out.println("Les erreurs suivantes se sont produites lors de l'initialisation du contexte Spring -------");
26.           for (String erreur : getErreursForThrowable(e)) {
27.               System.out.println(erreur);
28.           }
29.           // fin
30.           return;
31.       }
32.       // lecture de la BD
33.       ElectionsConfig electionsConfig = null;
34.       ListeElectorale[] listes;
35.       try {
36.           // contenu des deux tables
37.           electionsConfig = dao.getElectionsConfig();
38.           listes = dao.getListesElectorales();
39.       } catch (ElectionsException e) {
40.           System.out.println("Les erreurs suivantes se sont produites lors de la lecture des tables ----------");
41.           for (String erreur : e.getErreurs()) {
42.               System.out.println(erreur);
43.           }
44.           // fin
45.           return;
46.       }
47.       // tout s'est bien passé - affichage
48.       System.out.println(String.format("Nombre de sièges à pourvoir : %d", electionsConfig.getNbSiegesAPourvoir()));
49.       System.out.println(String.format("Seuil électoral : %5.2f", electionsConfig.getSeuilElectoral()));
50.       System.out.println("Listes candidates----------------");
51.       for (ListeElectorale liste : listes) {
52.           System.out.println(liste);
53.       }
54.   }
55.
56.   // méthodes privées ------------------
57.   private static List<String> getErreursForThrowable(Throwable th) {
58.       // on récupère la liste des messages d'erreur de l'exception
59.       Throwable cause = th;
60.       List<String> erreurs = new ArrayList<>();
61.       while (cause != null) {
62.           // on récupère le message seulement s'il est !=null et non blanc
63.           String message = cause.getMessage();
64.           if (message != null) {
65.               message = message.trim();
66.               if (message.length() != 0) {
67.                   erreurs.add(message);
68.               }
69.           }
70.           // cause suivante
71.           cause = cause.getCause();
72.       }
73.       return erreurs;
74.   }
75.
76. }
```

- lignes 21-31 : récupération d'une référence sur la couche [DAO] ;
- ligne 21 : on utilise une syntaxe appelée *try_with_resources*. Sa syntaxe est la suivante :

```
1. try (T ressource=expression) {
2.       // exploitation de ressource
3. ...
4. }
```

    - le type T de la ligne 1 doit implémenter l'interface [java.lang.AutoCloseable] ;
    - à la sortie du bloc des lignes 1-4, la ressource de type [java.lang.AutoCloseable] est automatiquement libérée qu'il y ait eu exception ou pas. Ceux qui connaissent le langage C# reconnaîtront là un frère syntaxique et fonctionnel de la clause **using** (T ressource=expression) ;
- lignes 40-49 : utilisation de la couche [DAO] pour obtenir le contenu des tables [CONF] et [LISTES] ;
- lignes 41-47 : on teste le cache [electionsconfig]. Celui-ci a été défini à deux endroits :
    - dans la classe [ElectionsDaoJdbc] :

```
  @Cacheable("electionsConfig")
  public ElectionsConfig getElectionsConfig() {
```

- dans la classe de configuration [AppConfig] :

```
@EnableCaching
public class AppConfig {
...
  @Bean
  public CacheManager cacheManager() {
    return new ConcurrentMapCacheManager("electionsConfig");
  }
}
```

- ligne 59 : fermeture du contexte Spring ;
- lignes 62-67 : affichage des informations obtenues ;

Les résultats obtenus avec une couche [DAO] implémentée sont les suivants :

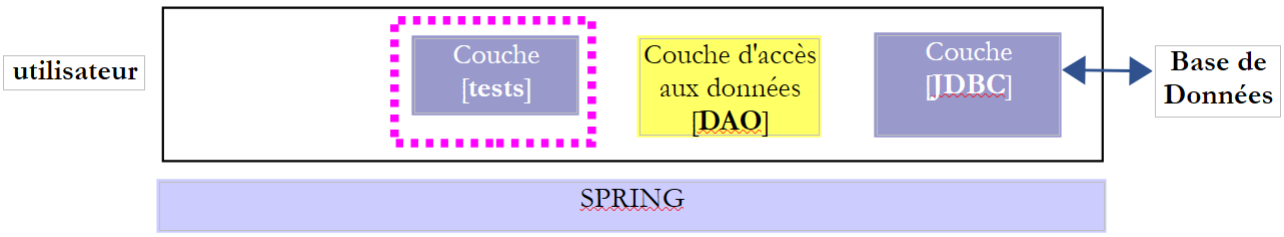
```
1. ...
2. début requête 1 : 11:09:29:752
3. fin requête 1 et début requête 2: 11:09:30:132
4. fin requête 2 : 11:09:30:133
5. ...
6. Nombre de sièges à pourvoir : 6
7. Seuil électoral :  0,05
8. Listes candidates----------------
9. {"id":1,"version":9,"nom":"A","voix":32000,"sieges":2,"elimine":false}
10.  {"id":2,"version":13,"nom":"B","voix":25000,"sieges":2,"elimine":false}
11.  {"id":3,"version":14,"nom":"C","voix":16000,"sieges":1,"elimine":false}
12.  {"id":4,"version":13,"nom":"D","voix":12000,"sieges":1,"elimine":false}
13.  {"id":5,"version":14,"nom":"E","voix":8000,"sieges":0,"elimine":false}
14.  {"id":6,"version":13,"nom":"F","voix":4500,"sieges":0,"elimine":true}
15.  {"id":7,"version":13,"nom":"G","voix":2500,"sieges":0,"elimine":true}
```

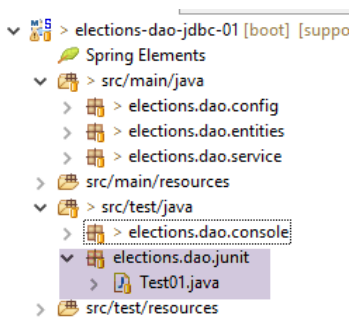
- lignes 2-4 : montrent l'influence du cache :
  - la requête 1 dure 80 millisecondes ;
  - la requête 2 dure 1 milliseconde ;

Lorsque la base de données est coupée on obtient les résultats suivants :

```
1. Les erreurs suivantes se sont produites lors de la lecture des tables ----------
2. Communications link failure
3.
4. The last packet sent successfully to the server was 0 milliseconds ago. The driver has not received any packets from the
    server.
5. Connection refused: connect
```

7.10 Tests JUnit de la classe [ElectionsDaoJdbc]





La classe [Test01] est la classe de test [JUnit] suivante :

```
1. package elections.dao.junit;
2.
3. import org.junit.Assert;
4. import org.junit.Before;
5. import org.junit.Test;
6. import org.junit.runner.RunWith;
7. import org.springframework.beans.factory.annotation.Autowired;
8. import org.springframework.boot.test.SpringApplicationConfiguration;
9. import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
10.
11.  import elections.dao.config.AppConfig;
12.  import elections.dao.entities.ElectionsConfig;
13.  import elections.dao.entities.ListeElectorale;
14.  import elections.dao.service.IElectionsDao;
15.
16.
17.  @SpringApplicationConfiguration(classes = AppConfig.class)
18.  @RunWith(SpringJUnit4ClassRunner.class)
19.  public class Test01 {
20.
21.    // couche [DAO]
22.    @Autowired
23.    private IElectionsDao electionsDao;
24.
25.    @Before
26.    public void init() {
27.        // on nettoie la table [LISTES]
28.        // listes en compétition
29.        ListeElectorale[] listes = electionsDao.getListesElectorales();
30.        // on met à 0 les voix et sièges et elimine à false
31.        int voix = 0;
32.        int sièges = 0;
33.        boolean elimine = false;
34.        for (ListeElectorale liste : listes) {
35.            liste.setVoix(voix);
36.            liste.setSieges(sièges);
37.            liste.setElimine(elimine);
38.        }
39.        // on rend ces données persistantes grâce à la couche [dao]
40.        electionsDao.setListesElectorales(listes);
41.    }
42.
43.    @Test
44.    public void testElections01() {
45.        ...
46.    }
47.  }
```

- ligne 17 : l'annotation [RunWith] qui est une annotation [JUnit] (ligne 6) assure l'intégration avec Spring via la classe [SpringJUnit4ClassRunner] ;
- ligne 16 : l'annotation [SpringApplicationConfiguration] est une annotation [Spring] (ligne 8) qui permet de désigner les classes de configuration du test JUnit. Ici on désigne la classe [AppConfig] utilisée pour configurer le projet. On dispose alors de tous les objets Spring définis par cette classe de configuration. C'est ainsi qu'on peut injecter lignes 21-22 une référence sur la couche [DAO] qui va être testée ;
- ligne 25 : l'annotation [Before] désigne une méthode qui doit être exécutée avant chaque test ;
- lignes 26-41 : la méthode [init] met les voix et sièges des listes de la table [LISTES] à zéro et le booléen [elimine] à [false] ;

L'unique méthode de test est la suivante :

```
1. @Test
```

```
2.   public void testElections01() {
3.     System.out.println("testElections01-------------------------------------");
4.     // récupération de la configuration des élections
5.     ElectionsConfig electionsConfig = electionsDao.getElectionsConfig();
6.     int nbSiegesAPourvoir = electionsConfig.getNbSiegesAPourvoir();
7.     double seuilElectoral = electionsConfig.getSeuilElectoral();
8.     Assert.assertEquals(6, nbSiegesAPourvoir);
9.     Assert.assertEquals(0.05, seuilElectoral, 1E-6);
10.
11.        // listes en compétition
12.        ListeElectorale[] listes = electionsDao.getListesElectorales();
13.        // affichage valeurs lues
14.        System.out.println("Nombre de sièges à pourvoir : " + nbSiegesAPourvoir);
15.        System.out.println("Seuil électoral : " + seuilElectoral);
16.        System.out.println("Listes en compétition ----------------------");
17.        for (int i = 0; i < listes.length; i++) {
18.            System.out.println(listes[i]);
19.        }
20.
21.        // on affecte des voix et des sièges aux listes
22.        int voix = 0;
23.        int sièges = 0;
24.        boolean elimine = false;
25.        for (ListeElectorale liste : listes) {
26.            liste.setVoix(voix);
27.            liste.setSieges(sièges);
28.            liste.setElimine(elimine);
29.            voix += 10;
30.            sièges += 1;
31.            elimine = !elimine;
32.        }
33.
34.        // on rend ces données persistantes grâce à la couche [dao]
35.        electionsDao.setListesElectorales(listes);
36.
37.        // on relit les données
38.        ListeElectorale[] listesElectorales2 = electionsDao.getListesElectorales();
39.        // on vérifie les données lues
40.        Assert.assertEquals(7, listesElectorales2.length);
41.        voix = 0;
42.        sièges = 0;
43.        elimine = false;
44.        for (ListeElectorale liste : listesElectorales2) {
45.            Assert.assertEquals(voix, liste.getVoix());
46.            Assert.assertEquals(sièges, liste.getSieges());
47.            Assert.assertEquals(elimine, liste.isElimine());
48.            voix += 10;
49.            sièges += 1;
50.            elimine = !elimine;
51.        }
52.        System.out.println("Listes en compétition ----------------------");
53.        for (int i = 0; i < listes.length; i++) {
54.            System.out.println(listes[i]);
55.        }
56.    }
```

- lignes 5-9 : on s'assure qu'on est capable de récupérer le contenu de la table [CONF] ;
- lignes 11-19 : on affiche le contenu de la table [LISTES]. Il n'y a pas de tests ici, seulement un contrôle visuel ;
- lignes 21-35 : on modifie en base, la table [LISTES] en donnant aux listes des valeurs pour leurs champs [voix, sieges, elimine] ;
- lignes 37-51 : on relit la table [LISTES] et on vérifie que ce qui est obtenu est bien égal à ce qui a été mis ;
- lignes 52-55 : vérification visuelle ;

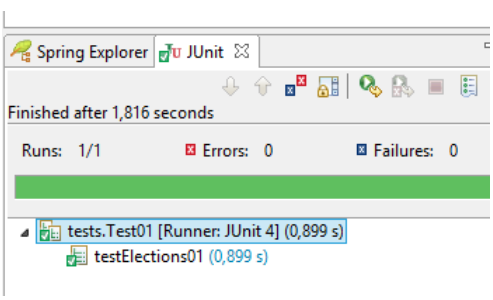
Les résultats console obtenus avec une couche [DAO] implémentée sont les suivants :

```
1. mars 11, 2015 4:50:00 PM org.springframework.test.context.support.DefaultTestContextBootstrapper
    getDefaultTestExecutionListenerClassNames
2. INFOS: Loaded default TestExecutionListener class names from location [META-INF/spring.factories]:
    [org.springframework.test.context.web.ServletTestExecutionListener,
    org.springframework.test.context.support.DependencyInjectionTestExecutionListener,
    org.springframework.test.context.support.DirtiesContextTestExecutionListener,
    org.springframework.test.context.transaction.TransactionalTestExecutionListener,
    org.springframework.test.context.jdbc.SqlScriptsTestExecutionListener]
3. mars 11, 2015 4:50:00 PM org.springframework.test.context.support.DefaultTestContextBootstrapper instantiateListeners
4. INFOS: Could not instantiate TestExecutionListener
    [org.springframework.test.context.transaction.TransactionalTestExecutionListener]. Specify custom listener classes or make
    the default listener classes (and their required dependencies) available. Offending class:
    [org/springframework/transaction/interceptor/TransactionAttributeSource]
5. mars 11, 2015 4:50:00 PM org.springframework.test.context.support.DefaultTestContextBootstrapper instantiateListeners
6. INFOS: Could not instantiate TestExecutionListener [org.springframework.test.context.web.ServletTestExecutionListener].
    Specify custom listener classes or make the default listener classes (and their required dependencies) available. Offending
    class: [javax/servlet/ServletContext]
```

```
7. mars 11, 2015 4:50:00 PM org.springframework.test.context.support.DefaultTestContextBootstrapper instantiateListeners
8. INFOS: Could not instantiate TestExecutionListener [org.springframework.test.context.jdbc.SqlScriptsTestExecutionListener].
    Specify custom listener classes or make the default listener classes (and their required dependencies) available. Offending
    class: [org/springframework/transaction/interceptor/TransactionAttribute]
9. mars 11, 2015 4:50:00 PM org.springframework.test.context.support.DefaultTestContextBootstrapper getTestExecutionListeners
10.  INFOS: Using TestExecutionListeners:
    [org.springframework.test.context.support.DependencyInjectionTestExecutionListener@483bf400,
    org.springframework.test.context.support.DirtiesContextTestExecutionListener@21a06946]
11.
12.    .   ____          _            __ _ _
13.   /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
14.  ( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
15.   \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
16.    '  |____| .__|_| |_|_| |_\__, | / / / /
17.   =========|_|==============|___/=/_/_/_/
18.   :: Spring Boot ::        (v1.2.2.RELEASE)
19.
20.  [2015-03-11 16:50:01.272] - 11696 INFOS [main] --- org.eclipse.jdt.internal.junit.runner.RemoteTestRunner: Starting
    RemoteTestRunner on Gportpers3 with PID 11696 (started by ST in D:\data\istia-1415\eclipse\intro-jdbc\elections-jdbc-01)
21.  [2015-03-11 16:50:01.317] - 11696 INFOS [main] ---
    org.springframework.context.annotation.AnnotationConfigApplicationContext: Refreshing
    org.springframework.context.annotation.AnnotationConfigApplicationContext@74ad1f1f: startup date [Wed Mar 11 16:50:01 CET
    2015]; root of context hierarchy
22.  mars 11, 2015 4:50:01 PM org.eclipse.jdt.internal.junit.runner.RemoteTestRunner logStarted
23.  INFOS: Started RemoteTestRunner in 0.775 seconds (JVM running for 1.433)
24.  testElections01---------------------------------------
25.  Nombre de sièges à pourvoir : 6
26.  Seuil électoral : 0.05
27.  Listes en compétition ---------------------
28.  {"id":1,"version":21,"nom":"A","voix":0,"sieges":0,"elimine":false}
29.  {"id":2,"version":22,"nom":"B","voix":0,"sieges":0,"elimine":false}
30.  {"id":3,"version":21,"nom":"C","voix":0,"sieges":0,"elimine":false}
31.  {"id":4,"version":13,"nom":"D","voix":0,"sieges":0,"elimine":false}
32.  {"id":5,"version":17,"nom":"E","voix":0,"sieges":0,"elimine":false}
33.  {"id":6,"version":18,"nom":"F","voix":0,"sieges":0,"elimine":false}
34.  {"id":7,"version":19,"nom":"G","voix":0,"sieges":0,"elimine":false}
35.  Listes en compétition ---------------------
36.  {"id":1,"version":21,"nom":"A","voix":0,"sieges":0,"elimine":false}
37.  {"id":2,"version":22,"nom":"B","voix":10,"sieges":1,"elimine":true}
38.  {"id":3,"version":21,"nom":"C","voix":20,"sieges":2,"elimine":false}
39.  {"id":4,"version":13,"nom":"D","voix":30,"sieges":3,"elimine":true}
40.  {"id":5,"version":17,"nom":"E","voix":40,"sieges":4,"elimine":false}
41.  {"id":6,"version":18,"nom":"F","voix":50,"sieges":5,"elimine":true}
42.  {"id":7,"version":19,"nom":"G","voix":60,"sieges":6,"elimine":false}
```

- lignes 1-23 : logs de Spring Test ;
- lignes 25-26 : le contenu de la table [CONF] ;
- lignes 27-34 : le contenu initial de la table [LISTES] ;
- lignes 35-42 : le contenu de la table [LISTES] après affectation de valeurs aux champs [voix, sieges, elimine] ;

Par ailleurs le test JUnit réussit :



7.11 Création de l'archive [with-dependencies] de la couche [dao]

Le projet final a l'architecture suivante :

Rappelons la configuration Maven du projet :

```
1. <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2.   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
3.   <modelVersion>4.0.0</modelVersion>
4.   <groupId>istia.st.elections</groupId>
5.   <artifactId>elections-dao-jdbc-01</artifactId>
6.   <version>0.0.1-SNAPSHOT</version>
7.
8.   <parent>
9.     <groupId>org.springframework.boot</groupId>
10.       <artifactId>spring-boot-starter-parent</artifactId>
11.       <version>1.2.7.RELEASE</version>
12.   </parent>
13.
14.   <dependencies>
15.       <!-- MySQL -->
16.       <dependency>
17.           <groupId>mysql</groupId>
18.           <artifactId>mysql-connector-java</artifactId>
19.       </dependency>
20.       <!-- Spring -->
21.       <dependency>
22.           <groupId>org.springframework</groupId>
23.           <artifactId>spring-context</artifactId>
24.       </dependency>
25.       <!-- Tomcat Jdbc -->
26.       <dependency>
27.           <groupId>org.apache.tomcat</groupId>
28.           <artifactId>tomcat-jdbc</artifactId>
29.       </dependency>
30.       <!-- bibliothèque jSON -->
31.       <dependency>
32.           <groupId>com.fasterxml.jackson.core</groupId>
33.           <artifactId>jackson-databind</artifactId>
34.       </dependency>
35.       <!-- Spring Boot -->
36.       <dependency>
37.           <groupId>org.springframework.boot</groupId>
38.           <artifactId>spring-boot</artifactId>
39.           <scope>test</scope>
40.       </dependency>
41.       <!-- Spring Boot Test -->
42.       <dependency>
43.           <groupId>org.springframework.boot</groupId>
44.           <artifactId>spring-boot-starter-test</artifactId>
45.           <scope>test</scope>
46.       </dependency>
47.       <!-- Spring Boot Logging -->
48.       <dependency>
49.           <groupId>org.springframework.boot</groupId>
50.           <artifactId>spring-boot-starter-logging</artifactId>
51.       </dependency>
52.   </dependencies>
53.
54.   <!-- plugins -->
55.   <build>
56.       <plugins>
57.           <plugin>
58.               <artifactId>maven-assembly-plugin</artifactId>
59.               <configuration>
60.                   <archive>
61.                       <manifest>
62.                           <mainClass>config.AppConfig</mainClass>
63.                       </manifest>
64.                   </archive>
65.                   <descriptorRefs>
66.                       <descriptorRef>jar-with-dependencies</descriptorRef>
67.                   </descriptorRefs>
68.               </configuration>
```

```
69.           </plugin>
70.           <!-- pour l'installation de l'artifact du projet dans le dépôt local Maven -->
71.           <plugin>
72.               <groupId>org.apache.maven.plugins</groupId>
73.               <artifactId>maven-surefire-plugin</artifactId>
74.               <version>2.18.1</version>
75.           </plugin>
76.       </plugins>
77.   </build>
78. </project>
```

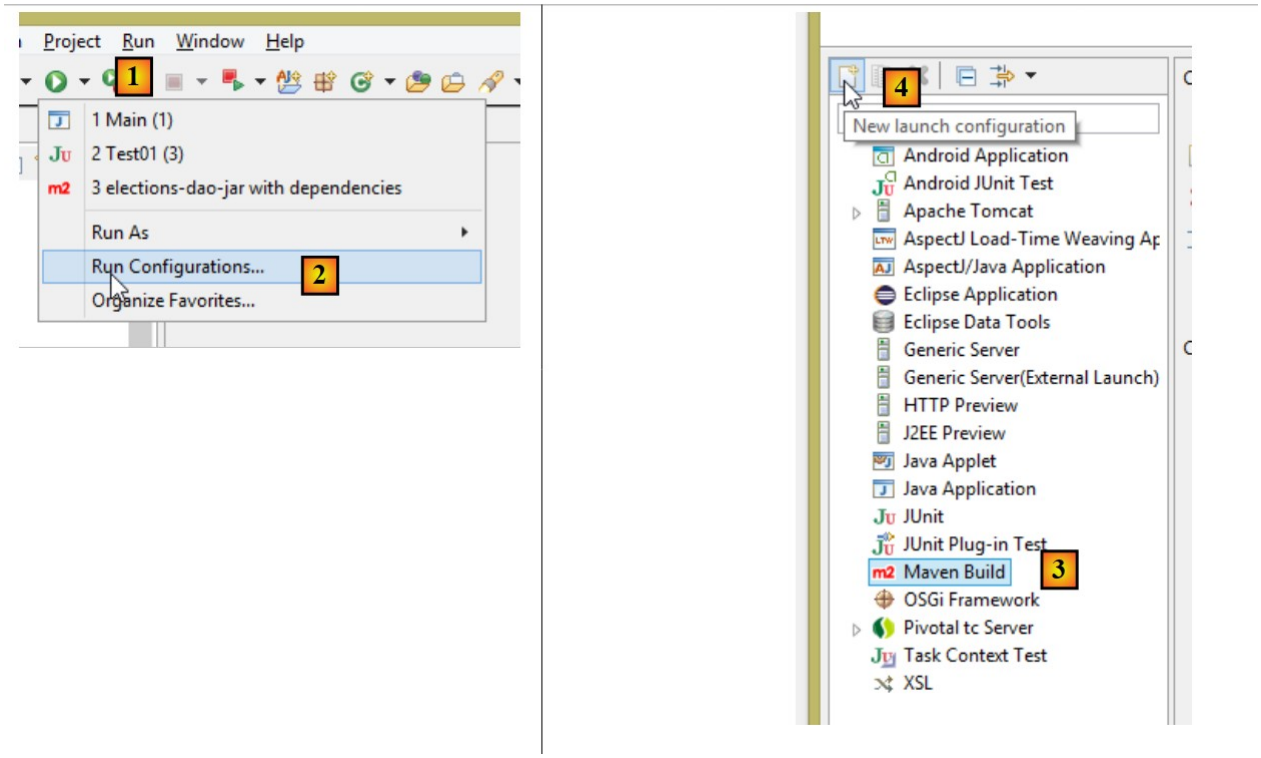
Nous allons générer un unique jar qui contiendra les classes de tous les jars ci-dessus plus celles du projet de la couche [DAO]. Cela se fait à l'aide d'une modification dans le fichier [pom.xml] :

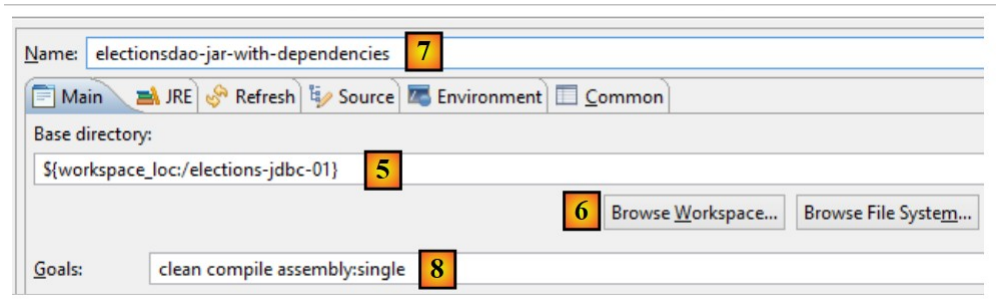
```
1. <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2.   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
3.   <modelVersion>4.0.0</modelVersion>
4.   <groupId>istia.st.elections</groupId>
5.   <artifactId>elections-jdbc-01</artifactId>
6.   <version>0.0.1-SNAPSHOT</version>
7.
8.   <parent>
9.     <groupId>org.springframework.boot</groupId>
10.       <artifactId>spring-boot-starter-parent</artifactId>
11.       <version>1.2.2.RELEASE</version>
12.   </parent>
13.   <properties>
14.       <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
15.       <java.version>1.8</java.version>
16.   </properties>
17.
18.   <dependencies>
19.       ...
20.   </dependencies>
21.
22.   <!-- plugins -->
23.   <build>
24.       <plugins>
25.           <plugin>
26.               <artifactId>maven-assembly-plugin</artifactId>
27.               <configuration>
28.                   <archive>
29.                       <manifest>
30.                           <mainClass>console.Main</mainClass>
31.                       </manifest>
32.                   </archive>
33.                   <descriptorRefs>
34.                       <descriptorRef>jar-with-dependencies</descriptorRef>
35.                   </descriptorRefs>
36.               </configuration>
37.           </plugin>
```

```
38.         </plugins>
39.     </build>
40. </project>
```

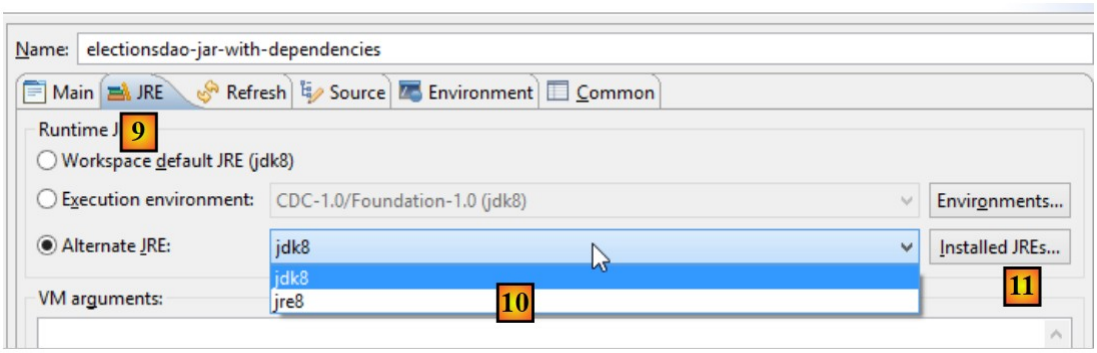
- lignes 25-37 : configurent un plugin Maven pour générer le jar ;
- ligne 15 : indique la version de Java à utiliser pour la compilation ;

Une fois cette modification faite, on peut générer le jar de la façon suivante [1-10] :

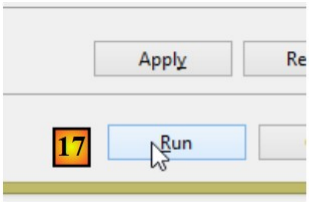


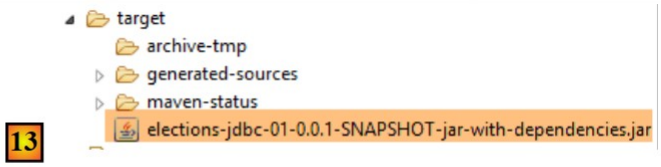


- en [5], mettre le dossier du projet en utilisant le bouton [6] ;
- en [7], donner un nom à la configuration d'exécution ;
- en [8], mettre la liste des buts Maven à exécuter :
  - [clean] : vide le dossier [target] du projet dans lequel va être placé le jar généré ;
  - [compile] : compile le projet ;
  - [assembly:single] : génère un unique jar comprenant toutes les classes du projet et de ses dépendances ;



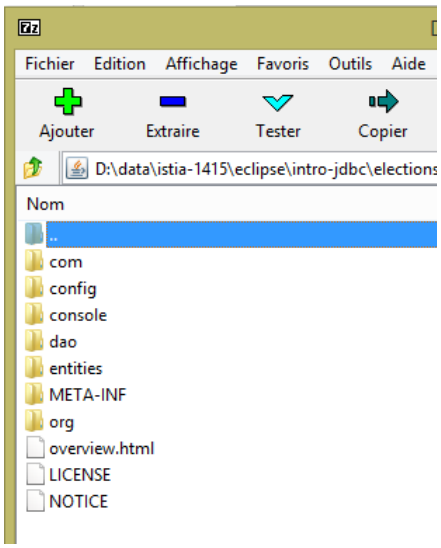
- en [9-10], vérifiez que vous avez un **JDK** (Java Development Kit) et non un **JRE** (Java Runtime Environment). La différence est que le JDK a un compilateur et pas le JRE. Si vous n'avez pas un JDK, il vous faut en ajouter un dans [10] avec [11]. Pour cela, suivre la procédure décrite au paragraphe 3.1, page 16 ;





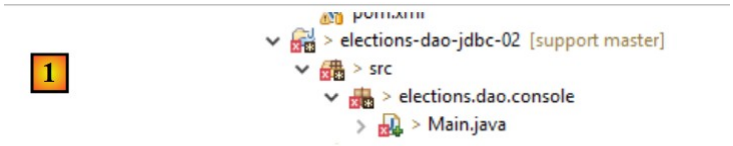
- en [17], exécutez la configuration d'exécution ;
- en [13], l'archive générée ;

On peut ouvrir cette archive avec un dézippeur :



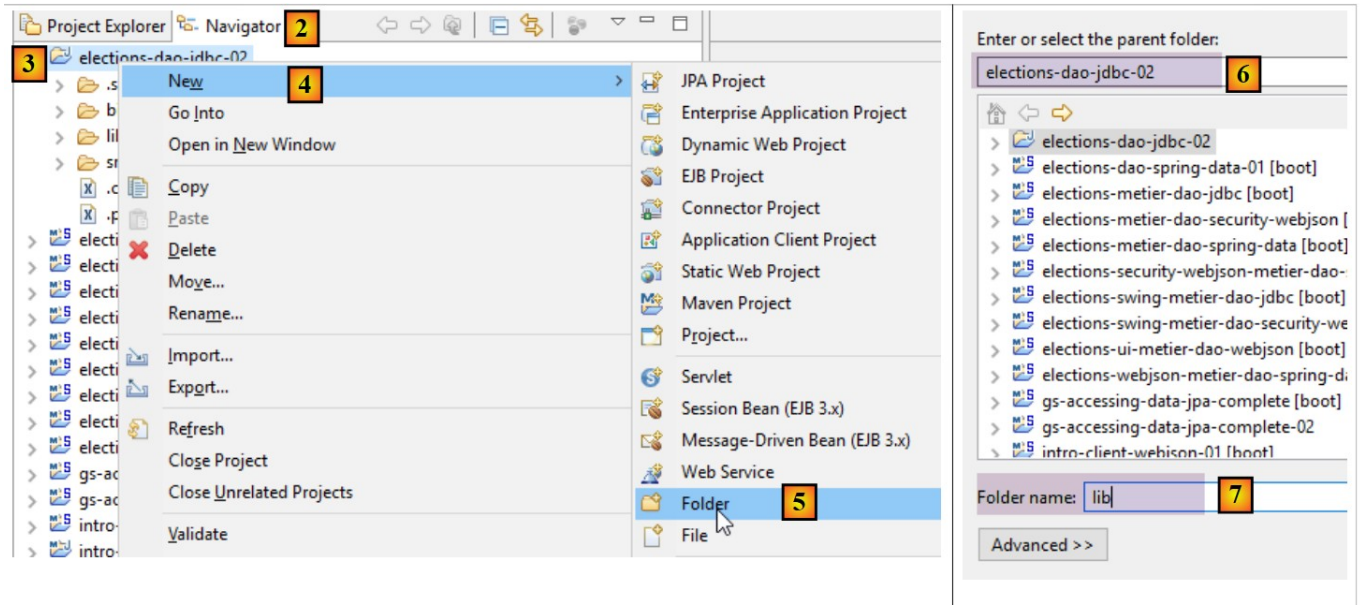
7.12 Test de l'archive de la couche [DAO]

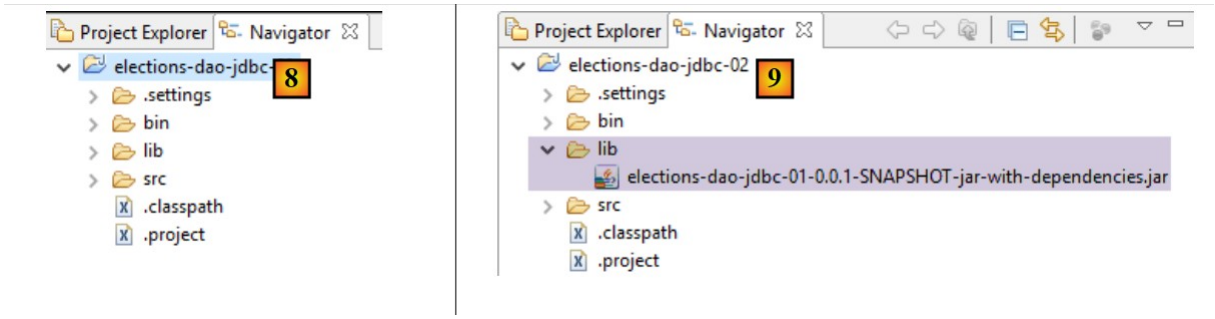
Créons un projet Eclipse standard (pas Maven) [1] :



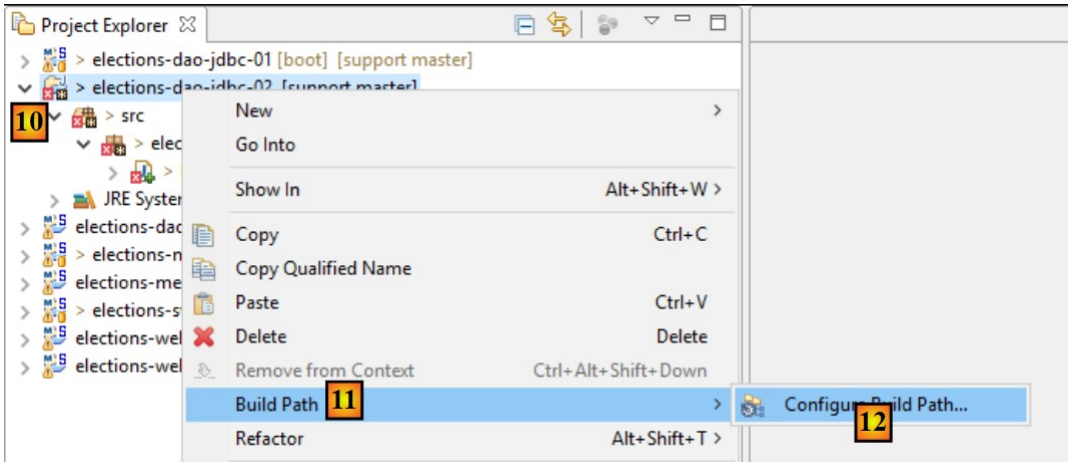
Faisons un copier / coller du package [dao.console] du projet [elections-dao-jdbc-01] dans le projet [elections-dao-jdbc-02] [2]. De nombreuses erreurs apparaissent car la classe [Main] référence des classes qui ne sont pas dans son [Classpath]. Nous allons modifier celui-ci.

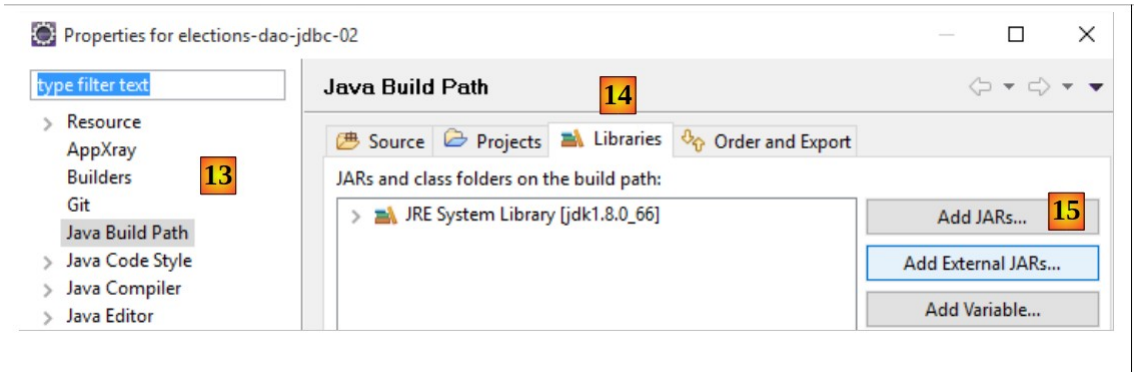
Nous créons tout d'abord [2-8] un dossier [lib] dans le nouveau projet :

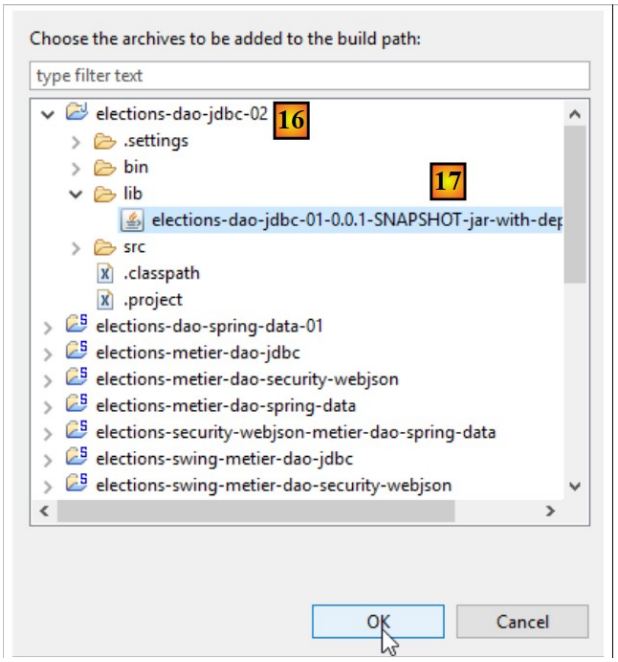


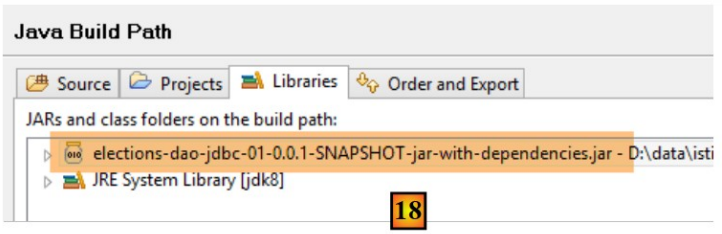


En [9], nous mettons l'archive créée à l'étape précédente dans le dossier [lib] puis nous modifions le Build Path du projet :









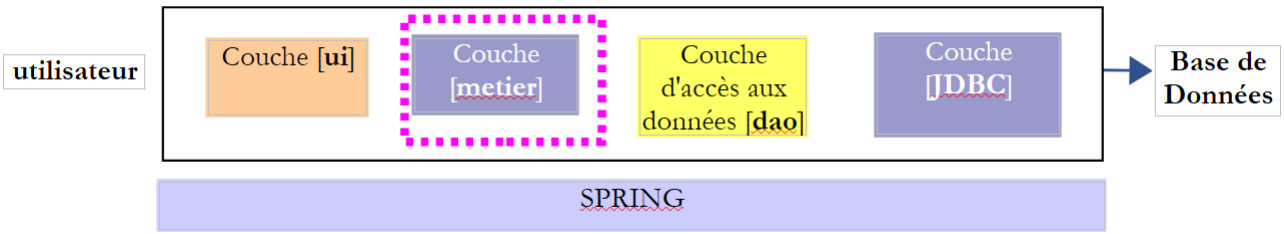
- en [18], on a importé l'archive de la couche [DAO] créée précédemment ;

- en [19], le projet ne présente plus d'erreurs ;

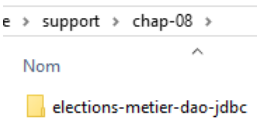
On peut alors exécuter la classe [Main]. On obtient les mêmes résultats que précédemment.

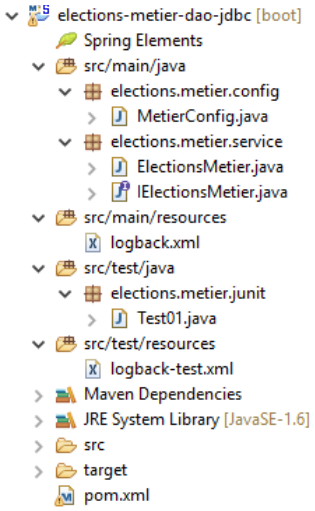
8 [TD] : La couche [metier]

Mots clés : architecture multicouche, Spring, injection de dépendances.



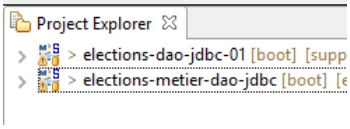
8.1 Support



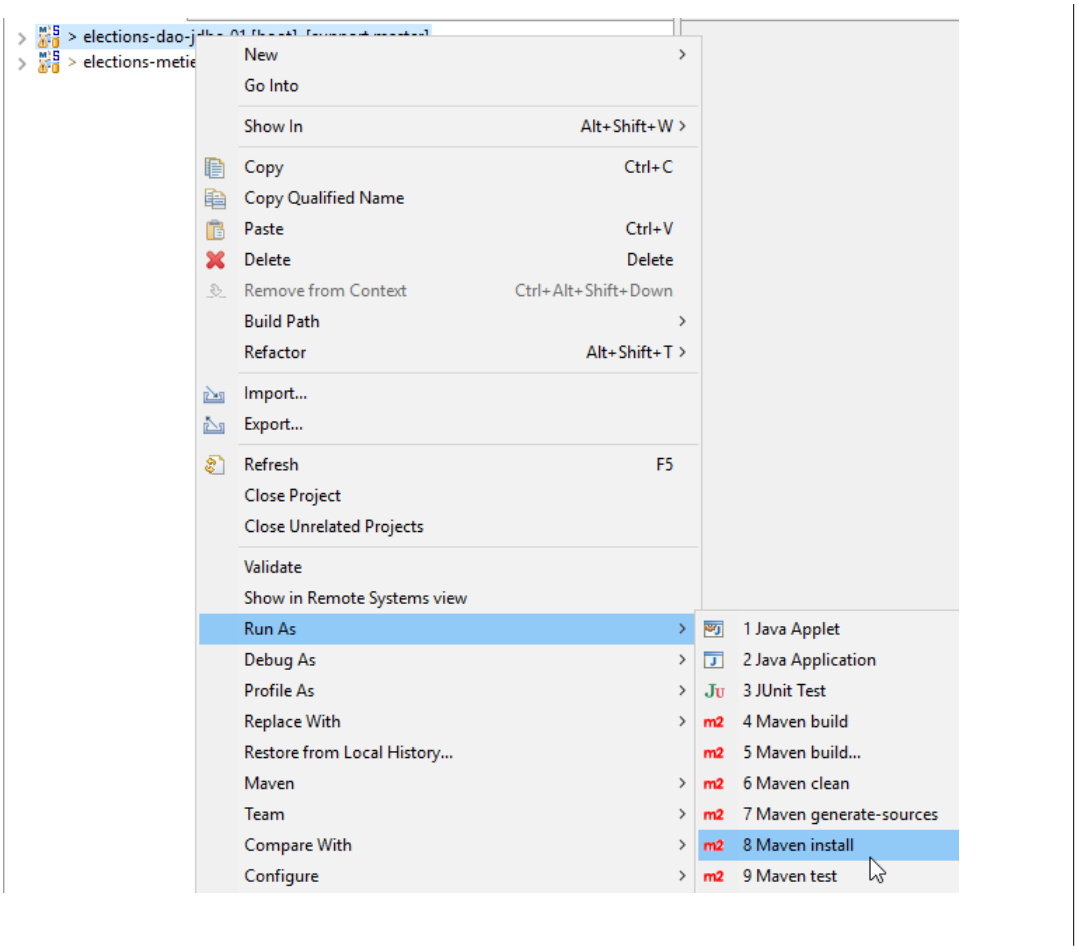


Le dossier [support / chap-08] contient le projet Eclipse de ce chapitre.

8.2 Configuration Maven



Le projet [elections-metier-dao-jdbc] va s'appuyer sur le projet [elections-dao-jdbc-01]. Nous allons installer le jar de ce dernier projet dans le dépôt Maven local :



Ceci fait, la configuration Maven du projet Eclipse [elections-metier-dao-jdbc] est la suivante :

```
1. <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2.   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
3.   <modelVersion>4.0.0</modelVersion>
4.   <groupId>istia.st.elections</groupId>
5.   <artifactId>elections-metier-dao-jdbc</artifactId>
6.   <version>0.0.1-SNAPSHOT</version>
7.
8.   <parent>
9.     <groupId>org.springframework.boot</groupId>
10.       <artifactId>spring-boot-starter-parent</artifactId>
11.       <version>1.2.7.RELEASE</version>
12.   </parent>
13.
14.   <properties>
15.       <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
16.       <java.version>1.8</java.version>
17.   </properties>
18.
```

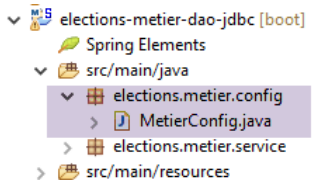
```
19.    <dependencies>
20.        <!-- DAO -->
21.        <dependency>
22.            <groupId>istia.st.elections</groupId>
23.            <artifactId>elections-dao-jdbc-01</artifactId>
24.            <version>0.0.1-SNAPSHOT</version>
25.        </dependency>
26.        <!-- Spring Boot -->
27.        <dependency>
28.            <groupId>org.springframework.boot</groupId>
29.            <artifactId>spring-boot</artifactId>
30.            <scope>test</scope>
31.        </dependency>
32.        <!-- Spring Boot Test -->
33.        <dependency>
34.            <groupId>org.springframework.boot</groupId>
35.            <artifactId>spring-boot-starter-test</artifactId>
36.            <scope>test</scope>
37.        </dependency>
38.    </dependencies>
39.
40.    <!-- plugins -->
41.    <build>
42.        <plugins>
43.            <plugin>
44.                <artifactId>maven-assembly-plugin</artifactId>
45.                <configuration>
46.                    <archive>
47.                        <manifest>
48.                            <mainClass>config.AppConfig</mainClass>
49.                        </manifest>
50.                    </archive>
51.                    <descriptorRefs>
52.                        <descriptorRef>jar-with-dependencies</descriptorRef>
53.                    </descriptorRefs>
54.                </configuration>
55.            </plugin>
56.            <!-- pour l'installation de l'artifact du projet dans le dépôt local Maven -->
57.            <plugin>
58.                <groupId>org.apache.maven.plugins</groupId>
59.                <artifactId>maven-surefire-plugin</artifactId>
60.                <version>2.18.1</version>
61.            </plugin>
62.        </plugins>
63.    </build>
64. </project>
```

- lignes 21-25 : la dépendance sur le jar du projet [elections-jdbc-01]. On prend ces lignes directement dans le fichier [pom.xml] du projet que l'on veut importer :

```
1.  <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
2.      <modelVersion>4.0.0</modelVersion>
3.      <groupId>istia.st.elections</groupId>
4.      <artifactId>elections-dao-jdbc-01</artifactId>
5.      <version>0.0.1-SNAPSHOT</version>
6.  ...
```

- lignes 26-37 : les dépendances nécessaires aux tests. Bien qu'elles soient présentes dans le fichier [pom.xml] du projet [elections-jdbc-01], nous sommes obligés de les remettre car leur attribut [`<scope>test</scope>`] fait qu'elles n'ont pas été incluses dans le jar mis dans le dépôt Maven local ;

8.3 Configuration Spring

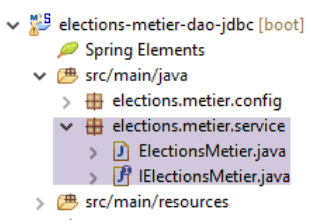



```
1. package elections.metier.config;
2.
3. import org.springframework.cache.annotation.EnableCaching;
4. import org.springframework.context.annotation.ComponentScan;
5. import org.springframework.context.annotation.Import;
```

```
6.
7. @Import({ elections.dao.config.AppConfig.class })
8. @EnableCaching
9. @ComponentScan(basePackages = { "elections.metier.service" })
10.  public class MetierConfig {
11.  }
```

- ligne 7 : on importe tous les beans définis dans la couche [DAO] ;
- ligne 8 : on active le cache. On ne redéfinit pas le bean [CacheManager], car il est déjà défini dans la couche [DAO] ;
- ligne 9 : la couche [métier] définit de nouveaux beans dans le package [elections.metier.service] ;

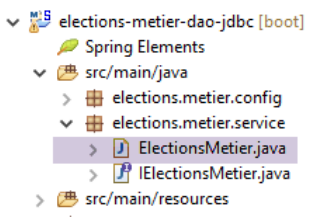
8.4 L'interface [IElectionsMetier]



La couche [metier] aura l'interface présentée page 37. Nous la rappelons :

```
1. public interface IElectionsMetier {
2.
3.   public ListeElectorale[] getListesElectorales();
4.
5.   public int getNbSiegesAPourvoir();
6.
7.   public double getSeuilElectoral();
8.
9.   public void recordResultats(ListeElectorale[] listesElectorales);
10.
11.    public ListeElectorale[] calculerSieges(ListeElectorale[] listesElectorales);
12.
13.  }
```

8.5 La classe d'implémentation [ElectionsMetier]



Cette classe implémente l'interface [IElectionsMetier]. L'implémentation proposée aura la structure suivante :

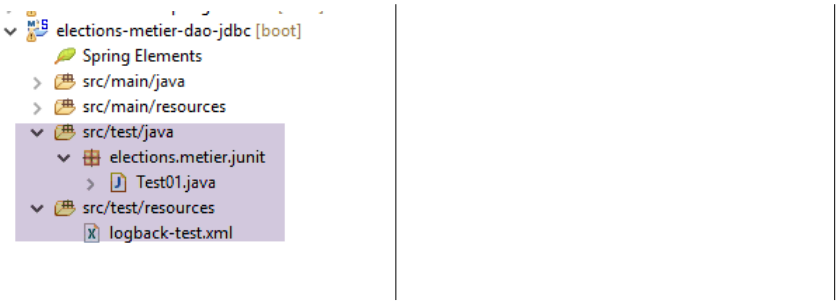
```
1. package elections.metier.service;
2.
3. import org.springframework.beans.factory.annotation.Autowired;
4. import org.springframework.cache.annotation.Cacheable;
5. import org.springframework.stereotype.Component;
6.
7. import elections.dao.entities.ListeElectorale;
8. import elections.dao.service.IElectionsDao;
9.
10.  @Component
11.  public class ElectionsMetier implements IElectionsMetier {
12.
13.    // le point d'accès à la couche [dao] instanciée par [Spring]
14.    @SuppressWarnings("unused")
15.    @Autowired
16.    private IElectionsDao electionsDao;
```

```
17.
18.    // calcul des sièges obtenus
19.    @Override
20.    public ListeElectorale[] calculerSieges(ListeElectorale[] listesElectorales) {
21.        throw new RuntimeException("[calculerSieges] not yet implemented");
22.    }
23.
24.    // listes en compétition
25.    @Override
26.    public ListeElectorale[] getListesElectorales() {
27.        throw new RuntimeException("[getListesElectorales] not yet implemented");
28.    }
29.
30.    // sauvegarde des résultats
31.    @Override
32.    public void recordResultats(ListeElectorale[] listesElectorales) {
33.        throw new RuntimeException("[recordResultats] not yet implemented");
34.    }
35.
36.    // nombre de sièges à pourvoir
37.    @Override
38.    public int getNbSiegesAPourvoir() {
39.        throw new RuntimeException("[getNbSiegesAPourvoir] not yet implemented");
40.    }
41.
42.    // seuil électoral
43.    @Override
44.    public double getSeuilElectoral() {
45.        throw new RuntimeException("[getSeuilElectoral] not yet implemented");
46.    }
47.
48. }
```

- ligne 10 : la classe [ElectionsMetier] est un composant Spring ;
- ligne 11 : la classe [ElectionsMetier] implémente l'interface [IElectionsMetier] ;
- lignes 15-16 : injection Spring d'une référence sur la couche [DAO] ;
- lignes 37 et 44 : le nombre de sièges à pourvoir et le seuil électoral sont mis en cache ;

Travail à faire : écrivez la classe [ElectionsMetier]. On s'aidera des commentaires lorsqu'il y en a. On n'essaiera pas d'arrêter (try / catch) les exceptions de type [ElectionsException] qui remontent de la couche [DAO]. On les laissera remonter à la couche [UI]. Parce que la classe [ElectionsException] est un type d'exception non contrôlée, il n'y a pas obligation à la gérer par un (try / catch).

8.6 La classe de test



La classe de test JUnit aura la forme suivante :

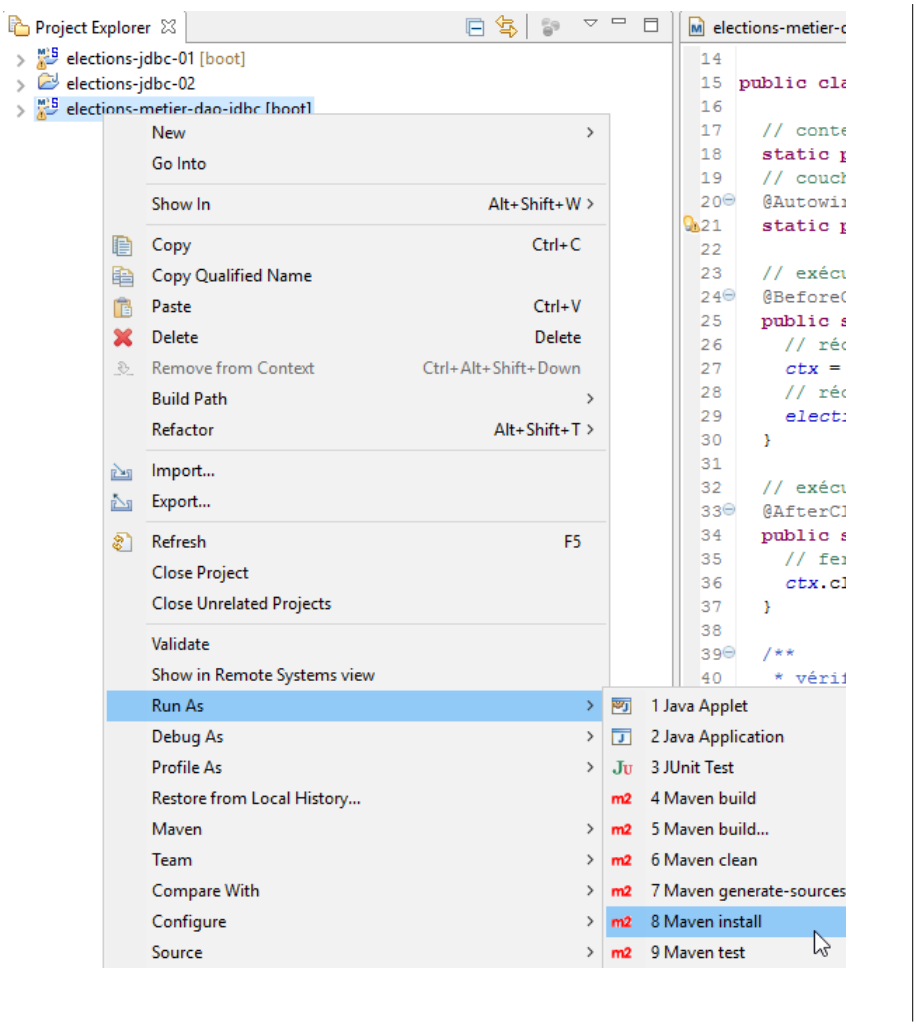
```
1. package tests;
2.
3. import org.junit.Test;
4. import org.junit.runner.RunWith;
5. import org.springframework.beans.factory.annotation.Autowired;
6. import org.springframework.boot.test.SpringApplicationConfiguration;
7. import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
8.
9. import dao.config.AppConfig;
10.  import dao.entities.ElectionsException;
11.  import metier.service.IElectionsMetier;
12.
13.  @SpringApplicationConfiguration(classes = MetierConfig.class)
14.  @RunWith(SpringJUnit4ClassRunner.class)
15.  public class Test01 {
16.
17.    // couche [métier]
```

```
18.   @Autowired
19.   static private IElectionsMetier electionsMetier;
20.
21.   /**
22.    * vérification 1 : méthode de calcul des sièges on fixe en dur les listes
23.    */
24.   @Test
25.   public void calculSieges1() {
26.       // on crée le tableau des 7 listes candidates (nom, voix)
27.
28.       // on calcule les sièges de chacune des listes
29.
30.       // on vérifie les résultats (sièges, voix, elimine)
31.
32.       // temporaire
33.       throw new RuntimeException("[calculSieges1] not yet implemented");
34.   }
35.
36.   /**
37.    * vérification 2 : méthode de calcul des sièges on demande les listes à la
38.    * couche [metier] puis on fixe en dur les voix
39.    */
40.   @Test
41.   public void calculSieges2() {
42.       // on récupère en base le tableau des 7 listes candidates
43.
44.       // on fixe en dur les voix
45.
46.       // on calcule les sièges obtenus par chacune des listes
47.
48.       // on vérifie les résultats (sièges, voix, elimine)
49.
50.       // temporaire
51.       throw new RuntimeException("[calculSieges2] not yet implemented");
52.   }
53.
54.   /**
55.    * vérification 3 méthode de calcul des sièges on provoque une exception
56.    */
57.   @Test(expected = ElectionsException.class)
58.   // écrire un test qui provoque une exception de type [ElectionsException]
59.   public void calculSieges3() {
60.       // on crée un tableau de 25 listes candidates avec chacune 1 voix
61.       // les 25 listes auront le même nombre de voix (4%)
62.
63.       // calcul des sièges - normalement on doit avoir une ElectionsException
64.       // avec un seuil électoral de 5%
65.
66.       // temporaire
67.       throw new RuntimeException("[calculSieges3] not yet implemented");
68.   }
69.
70.   /**
71.    * enregistrement des résultats de l'élection
72.    */
73.   @Test
74.   public void ecritureResultatsElections() {
75.       // on crée le tableau des 7 listes candidates
76.
77.       // on fixe en dur les voix
78.
79.       // on calcule les sièges obtenus par chacune des listes
80.
81.       // on enregistre les résultats dans la base de données
82.
83.       // on relit les listes en base
84.
85.       // on vérifie les résultats (sièges, voix, elimine)
86.
87.       // temporaire
88.       throw new RuntimeException("[ecritureResultatsElections] not yet implemented");
89.   }
90. }
```

Travail à faire : écrivez les quatre méthodes de tests en vous aidant des commentaires. On se rappellera que lorsque ces méthodes s'exécutent, le champ [electionsMetier] de la ligne 19 a déjà été initialisé. Vérifiez que le test JUnit passe.

8.7 Création de l'archive de la couche [metier]

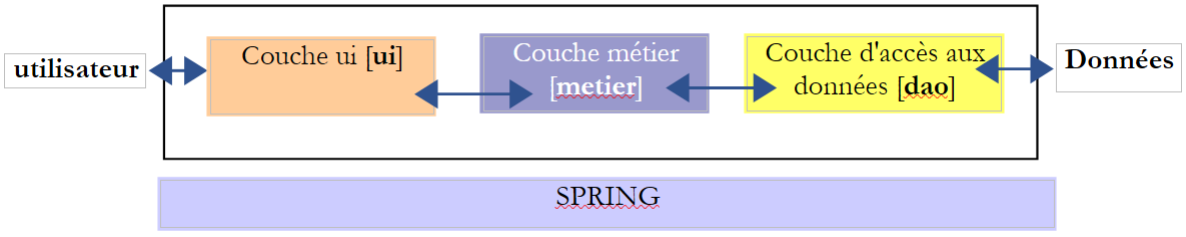
Comme il a été fait pour la couche [DAO], nous mettons l'archive du projet [elections-metier-dao-jdbc] dans le dépôt Maven local :



Note : cette opération peut échouer si votre projet Eclipse est associé à un JRE (Java Runtime Environment) au lieu d'un JDK (Java Development Kit). Pour le savoir, procédez comme indiqué au paragraphe 3.1, page 16. Si vous découvrez que vous avez un JRE et non un JDK, associez votre projet à un JDK comme indiqué dans ce paragraphe.

8.8 Conclusion

Rappelons l'architecture générale de l'application [Elections] que nous sommes en train de construire :

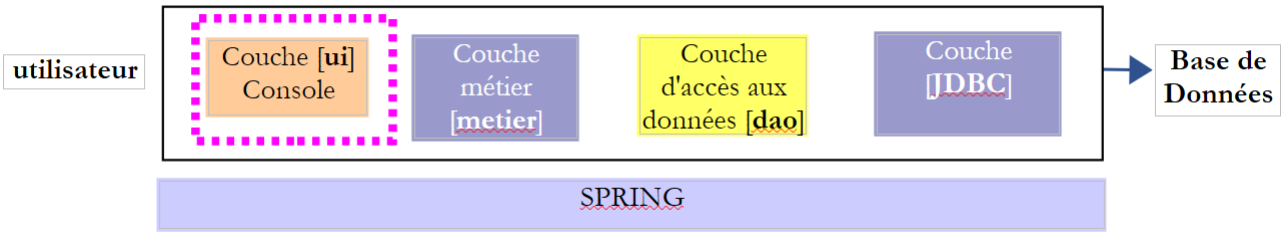


Nous avons construit les couches [metier] et [dao]. Nous allons maintenant construire la couche [ui]. Nous proposerons deux implémentations pour cette couche :

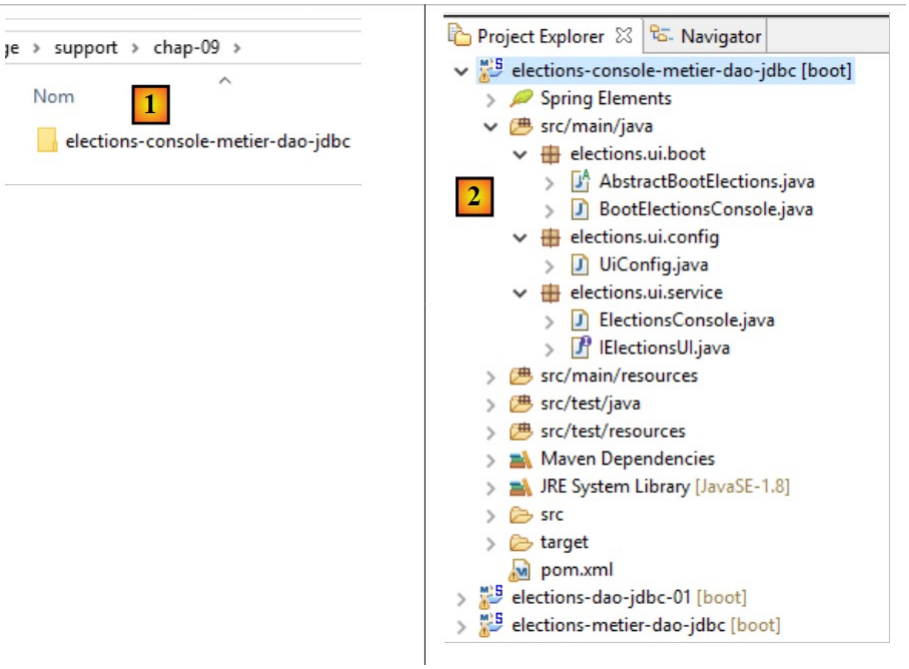
- une implémentation " console "
- une implémentation avec interface graphique

9 [TD] : Implémentation de la couche [ui] avec un programme console

Mots clés : architecture multicouche, Spring, injection de dépendances.



9.1 Support



En [1], le dossier [support / chap-09] contient le projet Eclipse de la couche [UI] de l'application console.

9.2 Configuration Maven

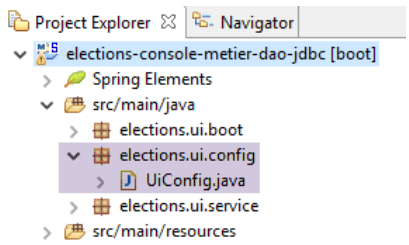
Le projet Eclipse [elections-ui-metier-dao-jdbc] est configuré par le fichier Maven [pom.xml] suivant :

```
1. <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2.   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
3.   <modelVersion>4.0.0</modelVersion>
4.   <groupId>istia.st.elections</groupId>
5.   <artifactId>elections-ui-metier-dao-jdbc</artifactId>
6.   <version>0.0.1-SNAPSHOT</version>
7.   <name>elections-ui-metier-dao-jdbc</name>
8.
9.   <parent>
10.       <groupId>org.springframework.boot</groupId>
11.       <artifactId>spring-boot-starter-parent</artifactId>
12.       <version>1.2.7.RELEASE</version>
13.    </parent>
14.
15.    <properties>
```

```
16.         <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
17.         <java.version>1.8</java.version>
18.     </properties>
19.
20.     <dependencies>
21.         <!-- métier -->
22.         <dependency>
23.             <groupId>istia.st.elections</groupId>
24.             <artifactId>elections-metier-dao-jdbc</artifactId>
25.             <version>0.0.1-SNAPSHOT</version>
26.         </dependency>
27.         <!-- Spring Boot -->
28.         <dependency>
29.             <groupId>org.springframework.boot</groupId>
30.             <artifactId>spring-boot</artifactId>
31.             <scope>test</scope>
32.         </dependency>
33.         <!-- Spring Boot Test -->
34.         <dependency>
35.             <groupId>org.springframework.boot</groupId>
36.             <artifactId>spring-boot-starter-test</artifactId>
37.             <scope>test</scope>
38.         </dependency>
39.     </dependencies>
40.
41.     <!-- plugins -->
42.     <build>
43.         <plugins>
44.             <plugin>
45.                 <artifactId>maven-assembly-plugin</artifactId>
46.                 <configuration>
47.                     <archive>
48.                         <manifest>
49.                             <mainClass>config.AppConfig</mainClass>
50.                         </manifest>
51.                     </archive>
52.                     <descriptorRefs>
53.                         <descriptorRef>jar-with-dependencies</descriptorRef>
54.                     </descriptorRefs>
55.                 </configuration>
56.             </plugin>
57.             <!-- pour l'installation de l'artifact du projet dans le dépôt local Maven -->
58.             <plugin>
59.                 <groupId>org.apache.maven.plugins</groupId>
60.                 <artifactId>maven-surefire-plugin</artifactId>
61.                 <version>2.18.1</version>
62.             </plugin>
63.         </plugins>
64.     </build>
65. </project>
```

- lignes 22-26 : on importe l'archive de la couche [métier] et par cascade celle de la couche [DAO] ;

9.3 Configuration Spring



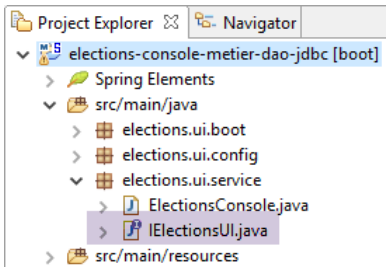
La classe [UiConfig] configure l'application Spring :

```
1. package elections.ui.config;
2.
3. import org.springframework.context.annotation.ComponentScan;
4. import org.springframework.context.annotation.Import;
5.
6. import elections.metier.config.MetierConfig;
7.
8. @Import(MetierConfig.class)
9. @ComponentScan(basePackages = { "elections.ui.service" })
10.  public class UiConfig {
```

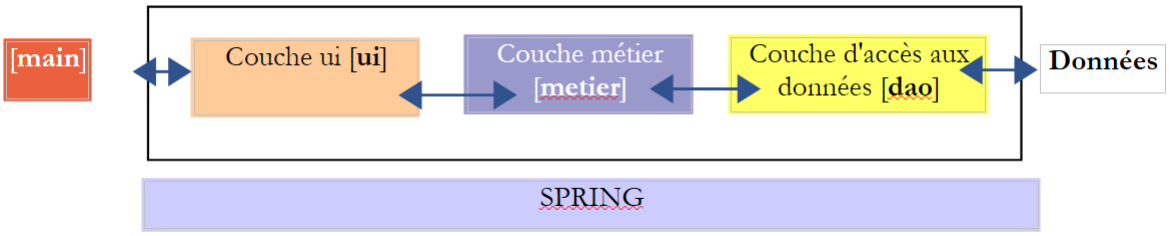
```
11. }
```

- ligne 8 : on importe les beans définis dans la couche [métier]. Celle-ci importait déjà les beans définis dans la couche [DAO]. On a donc ici accès aux beans des trois couches ;
- ligne 9 : le package [elections.ui.service] contient d'autres beans ;

9.4 L'interface de la couche [UI]



Pour comprendre ce que peut être l'interface Java de la couche [UI], il nous faut savoir qui va utiliser cette interface. Il ne s'agit pas de l'utilisateur du schéma ci-dessus mais du programme qui va lancer l'application dans son ensemble.

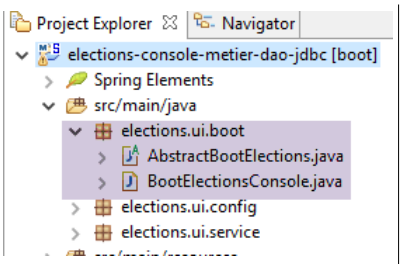


L'interface [IElectionsUI] est présentée au programme principal [main] qui va lancer l'application. Que peut demander [main] à la couche [ui] ? Elle peut lui demander de commencer les échanges avec l'utilisateur qui va saisir les données manquantes de l'élection. On adoptera l'interface minimale suivante :

```
1.  package istia.st.elections.ui;
2.
3.  public interface IElectionsUI {
4.      /**
5.       * lance le dialogue avec l'utilisateur
6.       */
7.      public void run();
8.  }
```

- ligne 7 : l'interface n'a qu'une unique méthode : **run**. En appelant cette méthode, on demande à la couche [ui] de commencer les échanges avec l'utilisateur.

9.5 La classe de démarrage de l'application



Le classe de démarrage de l'application est une classe Java ayant une méthode statique [main]. Cette méthode doit créer des instances des couches [ui, metier, demo] et demander à la couche [ui] de commencer le dialogue avec l'utilisateur. Cette classe pourrait être la classe [AbstractBootElections] suivante :

```
1. package elections.ui.boot;
2.
3. import org.springframework.context.annotation.AnnotationConfigApplicationContext;
4.
5. import elections.dao.entities.ElectionsException;
6. import elections.ui.config.UiConfig;
7. import elections.ui.service.IElectionsUI;
8.
9. public abstract class AbstractBootElections {
10.
11.    // récupération du contexte Spring
12.    protected AnnotationConfigApplicationContext ctx;
13.
14.    public void run() {
15.        // instanciation couche [ui]
16.        IElectionsUI electionsUI = null;
17.        try {
18.            // récupération du contexte Spring
19.            ctx = new AnnotationConfigApplicationContext(UiConfig.class);
20.            // récupération de la couche [ui]
21.            electionsUI = getUI();
22.        } catch (RuntimeException ex) {
23.            // on signale l'erreur
24.            afficheExceptions("Les erreurs suivantes se sont produites :", ex);
25.            // on arrête l'application
26.            System.exit(1);
27.        }
28.        // exécution couche [ui]
29.        try {
30.            electionsUI.run();
31.        } catch (ElectionsException ex2) {
32.            // on signale l'erreur
33.            afficheExceptions("Les erreurs suivantes se sont produites :", ex2);
34.            // on arrête l'application
35.            System.exit(3);
36.        } catch (RuntimeException ex1) {
37.            // on signale l'erreur
38.            afficheExceptions("Les erreurs suivantes se sont produites :", ex1);
39.            // on arrête l'application
40.            System.exit(2);
41.        }
42.    }
43.
44.    protected abstract IElectionsUI getUI();
45.
46.    private void afficheExceptions(String message, ElectionsException ex) {
47.        // on affiche le message
48.        System.out.println(String.format("%s -------------", message));
49.        System.out.println(String.format("Code erreur : %d", ex.getCode()));
50.        // on affiche les erreurs
51.        for (String erreur : ex.getErreurs()) {
52.            System.out.println(String.format("-- %s", erreur));
53.        }
54.
55.    }
56.
57.    public void afficheExceptions(String message, Exception ex) {
58.        // on affiche le message
59.        System.out.println(String.format("%s -------------", message));
60.        // on affiche la pile des exceptions
61.        Throwable cause = ex;
62.        while (cause != null) {
63.            System.out.println(String.format("-- %s", cause.getMessage()));
64.            cause = cause.getCause();
65.        }
66.    }
67. }
```

- ligne 19 : le contexte Spring est instancié : tous les beans définis dans les différents fichiers de configuration vont être créés ;
- ligne 21 : on récupère une référence sur le bean qui implémente l'interface [IElectionsUI]. Nous allons implémenter l'interface [IElectionsUI] par deux beans :
  - [ElectionsConsole] pour une application console ;
  - [ElectionsSwing] pour une application Swing ;

  La méthode [getUI] est abstraite (ligne 44). En effet, la classe [AbstractBootElections] va être parente de deux classes :
  - [BootElectionsConsole] qui fournira le bean [ElectionsConsole] à sa classe parent ;
  - [BootElectionsSwing] qui fournira le bean [ElectionsSwing] à sa classe parent ;

- ligne 30 : la méthode [run] de l'interface est exécutée ;

Les exceptions sont gérées à divers endroits :
- lignes 22-27 : une exception peut se produire lors de l'instanciation du contexte Spring ;
- lignes 31-41 : une exception peut se produire lors de l'exécution de la couche [UI]. On distingue deux types d'exception :
  - la classe [ElectionsException] que la couche [DAO] lance ;
  - la classe [RuntimeException] pour les autres exceptions qui pourraient survenir pendant l'exécution ;

La classe [BootElectionsConsole] est la classe qui lance l'implémentation console. Son code est le suivant :

```
1. package elections.ui.boot;
2.
3. import elections.ui.service.IElectionsUI;
4.
5. public class BootElectionsConsole extends AbstractBootElections{
6.   public static void main(String[] arguments) {
7.     new BootElectionsConsole().run();
8.   }
9.
10.    @Override
11.    protected IElectionsUI getUI() {
12.        return ctx.getBean("electionsConsole",IElectionsUI.class);
13.    }
14.  }
```

- ligne 5, la classe [BootElectionsConsole] étend la classe [AbstractBootElections]. Elle doit donc implémenter la méthode [getUI] que sa classe parente avait déclarée abstraite ;
- lignes 10-13 : implémentation de la méthode [getUI] ;
- ligne 12 : on rend le bean nommé [electionsConsole] implémentant l'interface [IElectionsUI]. [ctx] est le contexte Spring défini dans la classe parente par la déclaration :

```
// récupération du contexte Spring
protected AnnotationConfigApplicationContext ctx;
```

Parce que le champ a l'attribut [protected], il est visible dans les classes filles.

Le bean de la ligne 12 sera déclaré de la façon suivante :

```
1. @Component
2. public class ElectionsConsole implements IElectionsUI {
```

Le nom par défaut de ce bean est le nom de la classe avec sa première lettre en minuscule. On peut s'affranchir de ce nom par défaut en écrivant explicitement :

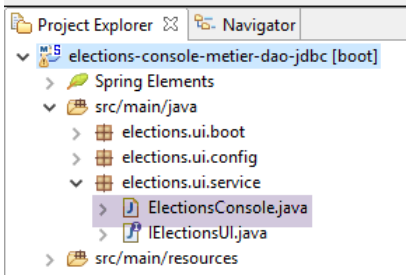
```
@Component(nom_du_bean_entre_guillemets)
```

La classe [BootElectionsSwing] qui lancerait l'implémentation Swing pourrait être la suivante :

```
1. package elections.ui.boot;
2.
3. import elections.ui.service.IElectionsUI;
4.
5. public class BootElectionsSwing extends AbstractBootElections {
6.   public static void main(String[] arguments) {
7.     new BootElectionsSwing().run();
8.   }
9.
10.    @Override
11.    protected IElectionsUI getUI() {
12.        return ctx.getBean("electionsSwing", IElectionsUI.class);
13.    }
14.  }
```

Ce modèle de conception où on factorise le comportement commun à des classes dans une classe parent, et où on laisse les classes filles implémenter les détails qui leur sont spécifiques s'appelle le modèle de conception *Strategy*. Ce modèle de conception impose un comportement commun à toutes les classes filles.

9.6 La classe d'implémentation [ElectionsConsole]



Notre première classe d'implémentation de la couche [ui] sera une classe utilisant la console pour communiquer avec l'utilisateur. Voici un exemple de dialogue obtenu sur la console Eclipse :

```
1.  Il y a 7 listes en compétition. Veuillez indiquer le nombre de voix de chacune d'elles :
2.  Nombre de voix de la liste [A] : 2500
3.  Nombre de voix de la liste [B] : 4500
4.  Nombre de voix de la liste [C] : x
5.  Nombre de voix incorrect. Veuillez recommencer
6.  Nombre de voix de la liste [C] : 8000
7.  Nombre de voix de la liste [D] : 12000
8.  Nombre de voix de la liste [E] : 16000
9.  Nombre de voix de la liste [F] : 25000
10. Nombre de voix de la liste [G] : 32000
11.
12. Résultats de l'élection
13.
14. [G,32000,2,false]
15. [F,25000,2,false]
16. [E,16000,1,false]
17. [D,12000,1,false]
18. [C,8000,0,false]
19. [B,4500,0,true]
20. [A,2500,0,true]
```

La classe [ElectionsConsole] pourrait avoir le squelette suivant :

```
1. package elections.ui.service;
2.
3. import java.util.Comparator;
4. import java.util.Scanner;
5.
6. import org.springframework.beans.factory.annotation.Autowired;
7. import org.springframework.stereotype.Component;
8.
9. import elections.dao.entities.ListeElectorale;
10.  import elections.metier.service.IElectionsMetier;
11.
12.  @Component
13.  public class ElectionsConsole implements IElectionsUI {
14.
15.    @Autowired
16.    private IElectionsMetier electionsMetier;
17.
18.    @Override
19.    public void run() {
20.        // saisie des données
21.        try (Scanner clavier = new Scanner(System.in)) {
22.            // on demande les listes en compétition à la couche [metier]
23.
24.            // on fait la saisie des voix
25.
26.        }
27.        // on fait le calcul des sièges
28.
29.        // on enregistre les résultats
30.
31.        // tri des listes dans l'ordre décroissant des voix
32.
33.        // on les affiche
34.    }
35.
36.    // classe de comparaison de listes électorales
37.    class CompareListesElectorales implements Comparator<ListeElectorale> {
38.
39.        // comparaison de deux listes candidates selon le nombre de voix
40.        @Override
41.        public int compare(ListeElectorale listeElectorale1, ListeElectorale listeElectorale2) {
```

```
42.           // on compare les voix de ces deux listes
43.           int nbVoix1 = listeElectorale1.getVoix();
44.           int nbVoix2 = listeElectorale2.getVoix();
45.           if (nbVoix1 < nbVoix2) {
46.               return +1;
47.           } else {
48.               if (nbVoix1 > nbVoix2)
49.                   return -1;
50.               else
51.                   return 0;
52.           }
53.       }
54.   }
55.
56. }
```

- ligne 12 : la classe [ElectionsConsole] est un composant Spring ;
- ligne 13 : la classe implémente l'interface [IElectionsUI]
- lignes 15-16 : injection par Spring d'une référence sur la couche [metier] ;
- ligne 18-34 : la méthode [run] de l'interface [IelectionsUI] ;

Le *try* des lignes 21-28 s'appelle *try-with-resources* et sa syntaxe est la suivante :

```
1. try(ressource){
2. …
3. }
```

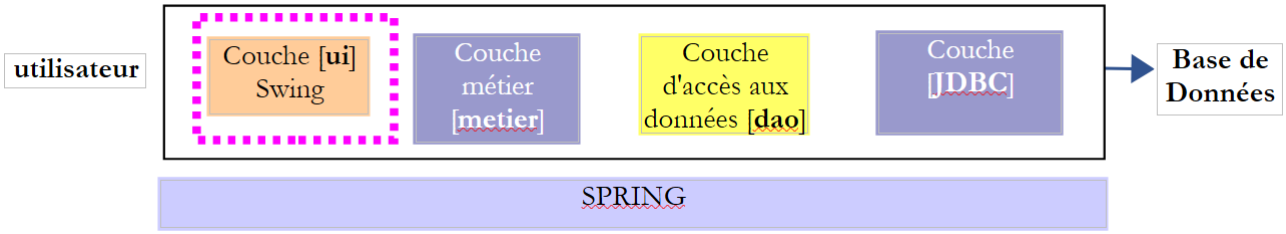
La ressource de la ligne 1 doit implémenter l'interface [`java.lang.AutoCloseable`]. La ressource est ouverte ligne1 et automatiquement fermée après la ligne 3, qu'il y ait exception ou non dans le code exécuté entre les deux lignes. Cette syntaxe permet de s'assurer qu'une ressource ouverte sera bien fermée, quoiqu'il arrive.

Travail à faire : écrivez le code de la méthode [run]. On s'aidera des commentaires.

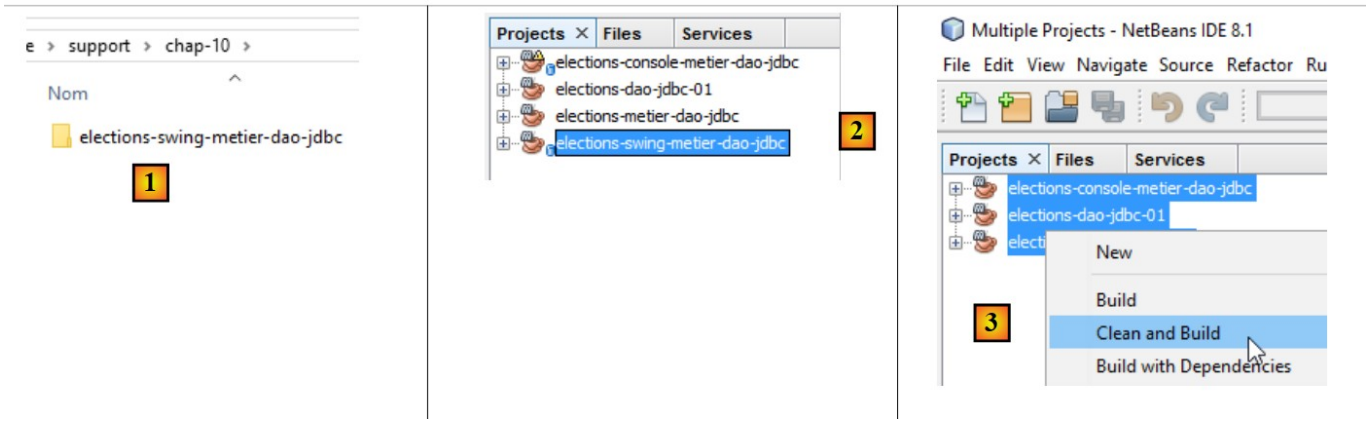
---

10 [TD] : Implémentation de la couche [ui] avec une interface Swing

Mots clés : architecture multicouche, Spring, injection de dépendances, bibliothèque de composants Swing.

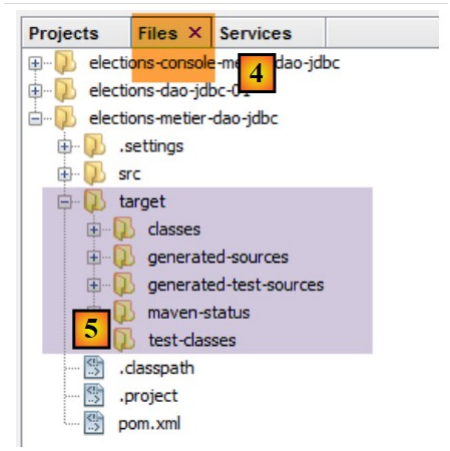


10.1 Support



Dans la couche [UI], on veut construire une interface graphique Swing. Netbeans a un outil [Matisse] pour construire ces interfaces Swing qui est supérieur à ce que peut proposer Eclipse. Les interfaces Swing tendent à être remplacées par des interfaces JavaFx. Netbeans et Eclipse utilisent le même outil pour construire ces dernières. Si donc, on construit des interfaces JavaFx, on peut garder Eclipse de bout en bout de l'architecture en couches.

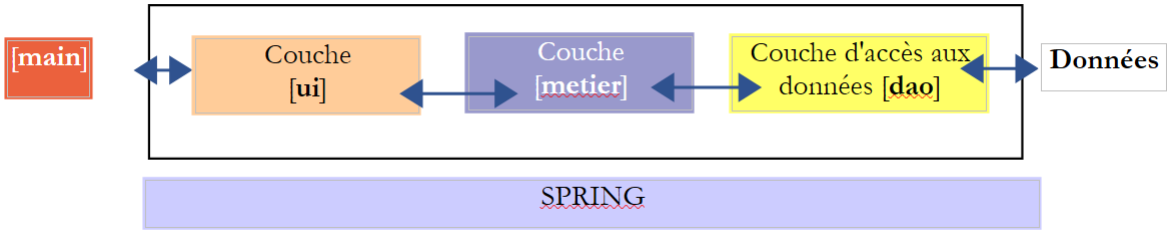
Netbeans peut ouvrir n'importe quel projet Maven. On va donc utiliser le projet Maven précédent et lui ajouter une interface Swing. En [2], on charge (File / Open project) les projets Maven des trois couches que nous avons construites avec Eclipse. Puis, on construit leurs binaires [3]. Les options [Build] et [Clean and Build] construisent le binaire du prjet sur lequel elles sont appliquées. Ces binaires sont placés dans le dossier [target] [4-5] du projet :



L'option [Clean] supprime ce dossier [target]. L'option [Build] le reconstruit. L'expérience montre que lorsqu'on rencontre des problèmes inattendus, la première chose à faire est un [Clean and Build] sur le projet pour être sûr qu'on travaille bien avec la dernière version de celui-ci. C'est notamment nécessaire lorsqu'on a des fichiers de configuration qui s'ils sont modifiés ne provoquent pas une recompilation automatique lorsqu'on exécute le projet. Il faut alors forcer cette recompilation par un [Clean and Build] pour que leurs nouvelles versions soient installées dans le dossier [target].

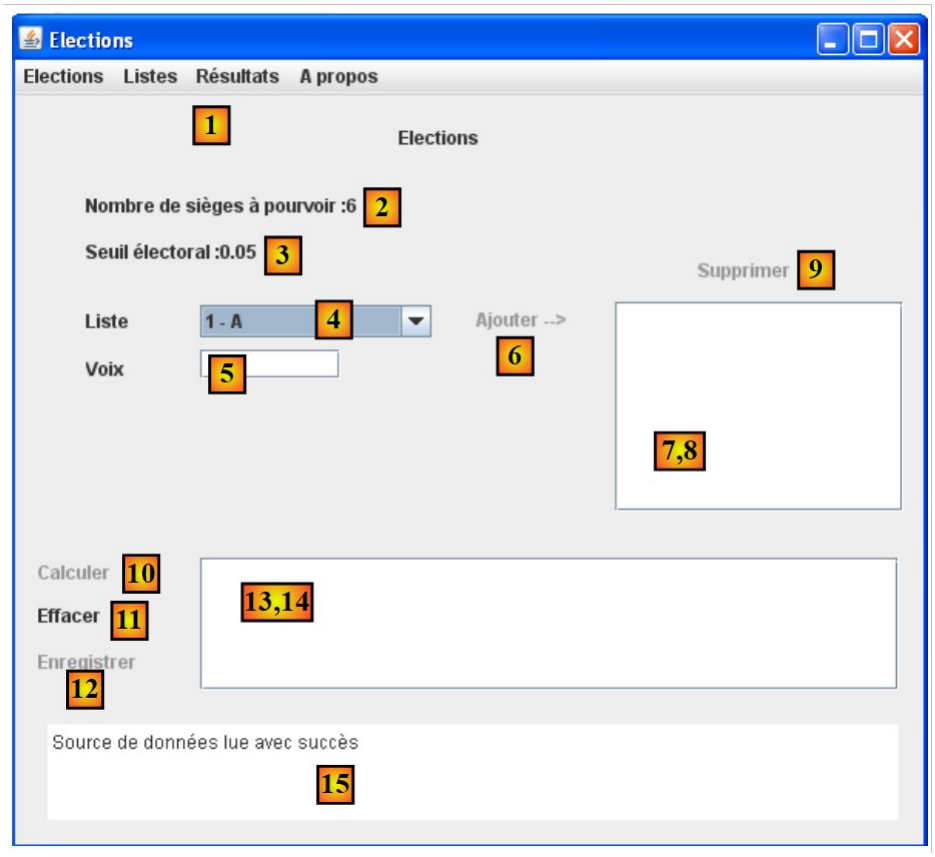
10.2 Fonctionnement de l'application

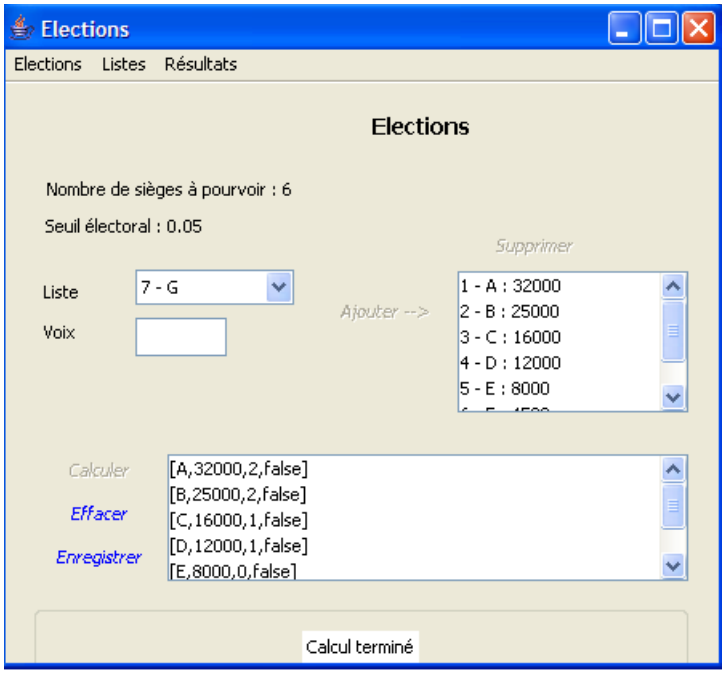
Revenons à l'architecture globale de l'application [Elections] :



Nous nous intéressons maintenant à une nouvelle implémentation de la couche [ui]. L'unique implémentation réalisée pour le moment est une interface console. Nous créons maintenant une interface graphique.

L'utilisateur disposera de l'interface suivante pour interagir avec l'application [Elections] :



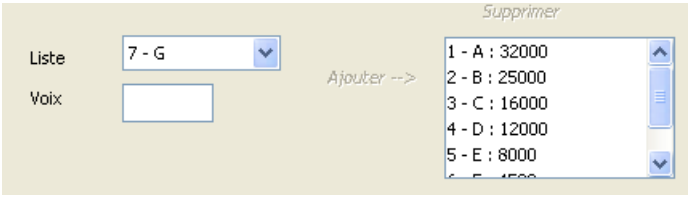


L'interface graphique se trouve dans la couche [ui]. C'est elle qui interagit avec l'utilisateur.

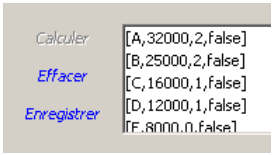
- au démarrage, l'application console [**main**] instancie les trois couches de l'application grâce à Spring. Ceci est fait avant même que l'interface graphique ne soit visible. Toujours dans cette phase d'initialisation, les renseignements caractérisant l'élection (nombre de sièges à pourvoir, seuil électoral, listes en compétition) sont demandés à la couche [dao]. Si cette phase d'initialisation échoue (impossibilité d'accéder aux données par exemple), un message d'erreur est affiché sur la console et l'interface graphique n'est pas affichée.

- si la lecture des données s'est bien passée, l'interface graphique est affichée avec les renseignements suivants (cf copie d'écran plus haut) :
  - le nombre de sièges à pourvoir dans (2)
  - le seuil électoral dans (3)
  - les identifiants et noms des listes candidates dans (4)

- l'utilisateur affecte alors à chaque liste candidate son nombre de voix à l'aide des champs 4 (id - nom), 5 (voix), 6 (pour ajouter).



- on peut alors utiliser le lien (10) pour calculer les sièges :



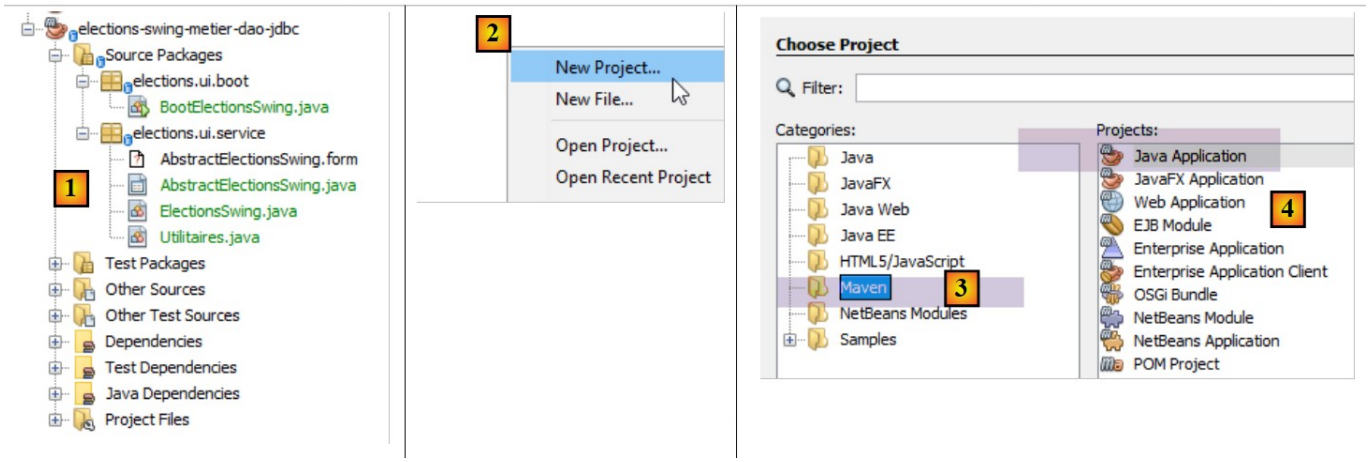
- le lien [Enregistrer] (12) permet d'enregistrer les résultats dans la source de données.

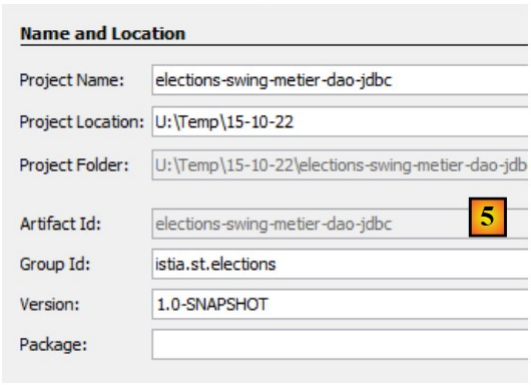
10.3 La classe [ElectionsSwing] d'implémentation de la couche [ui]

10.3.1 Le projet Netbeans

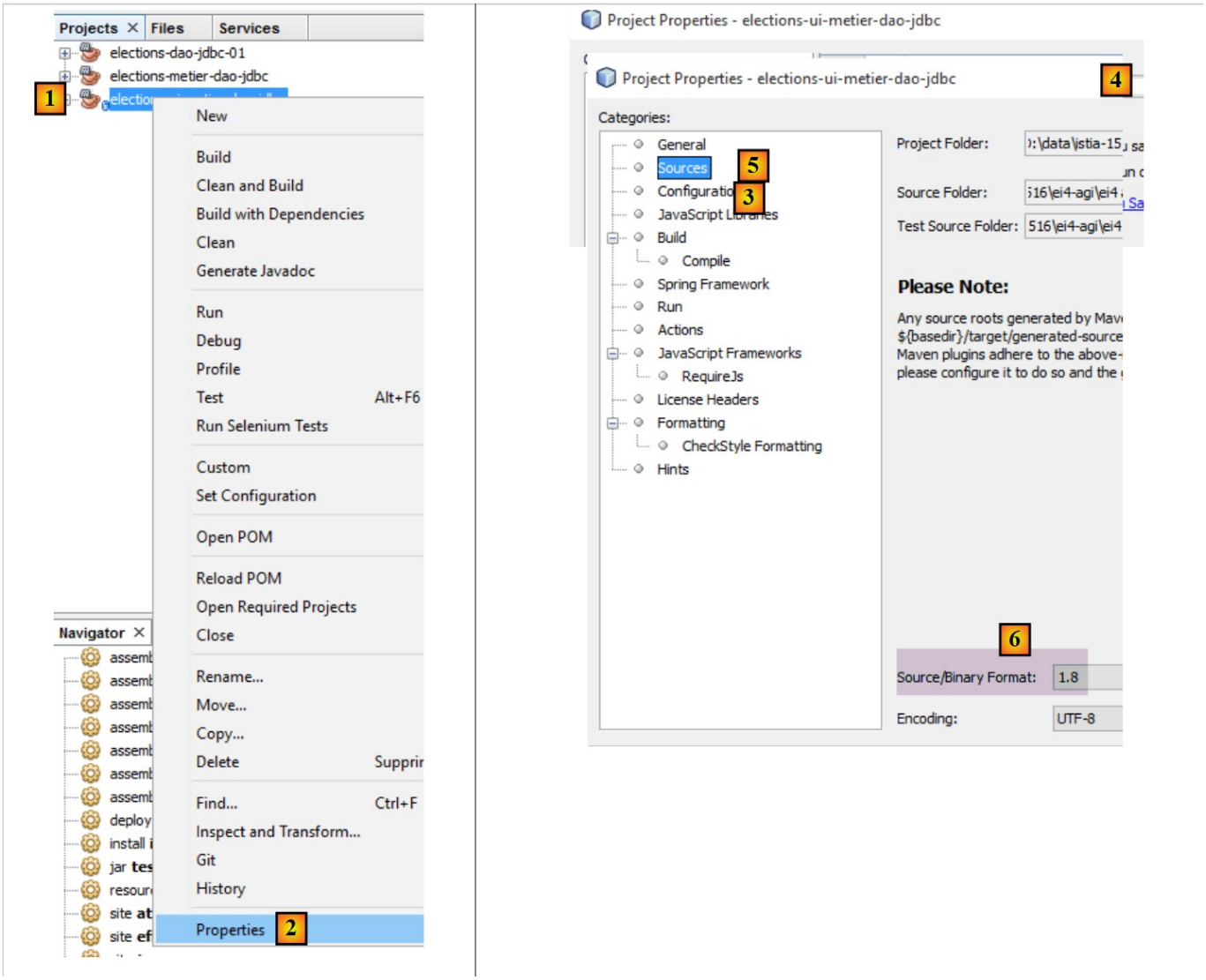
Note : Le paragraphe 22.4, page 418, indique comment se procurer Netbeans.

Le projet Netbeans final de l'application aura la forme suivante [1]. Construisez-le en suivant les étape [2-5] :



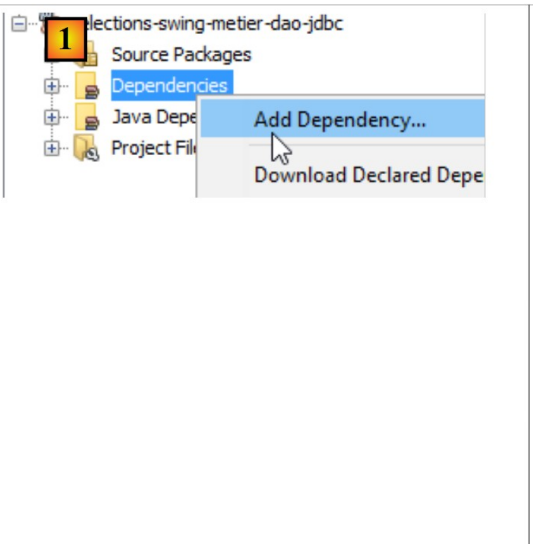


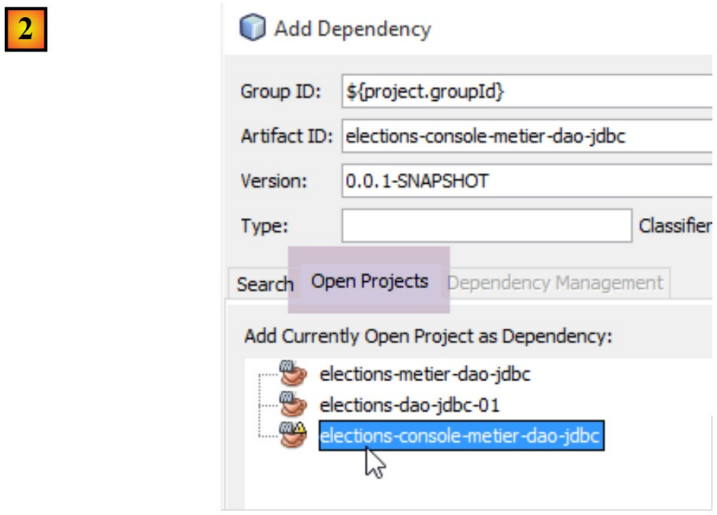
On s'assurera que le projet est configuré pour être compilé par un JDK 1.8 [1-6] :

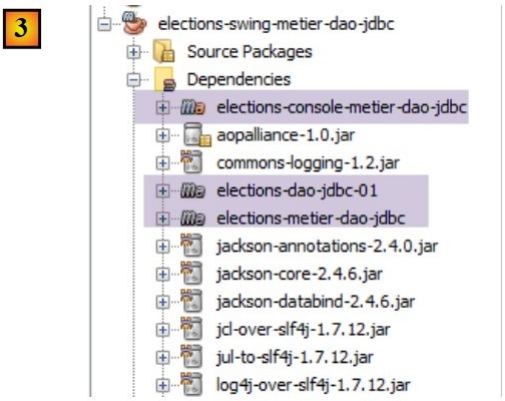


10.3.2 Configuration Maven

Le nouveau projet [elections-swing-metier-dao-jdbc] va s'appuyer sur le précédent projet [elections-console-metier-dao-jdbc]. Pour cela, on ajoute une dépendance Maven de la façon suivante [1-3] :

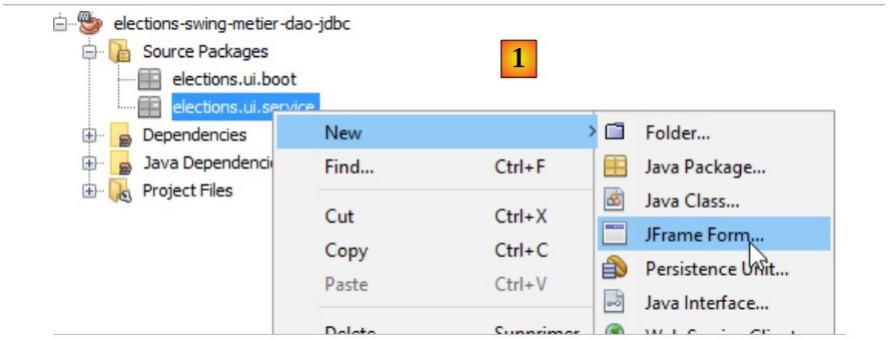


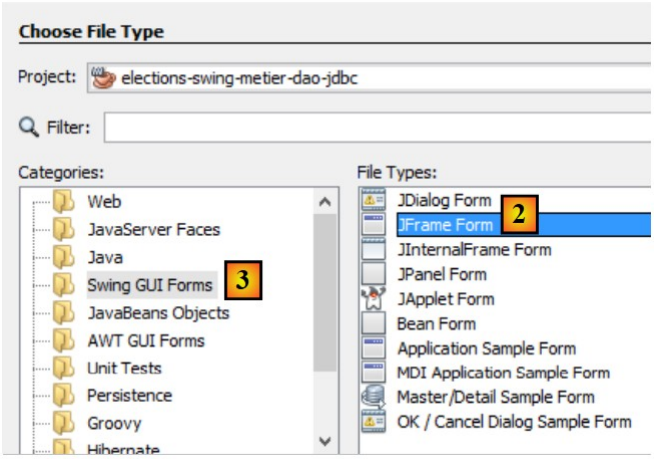




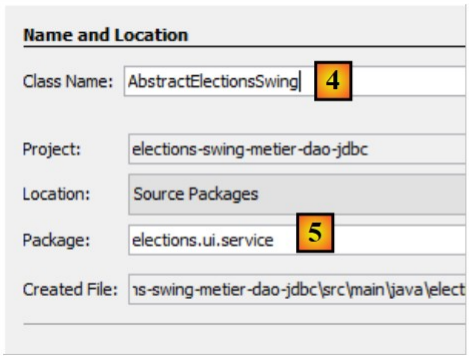
10.3.3 Construction de l'interface graphique

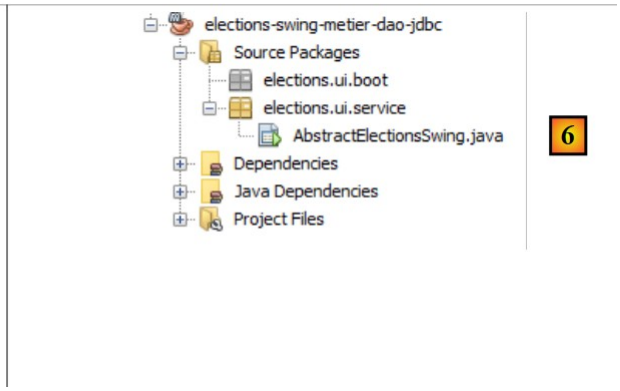
Pour créer l'interface graphique, nous pouvons procéder comme suit :



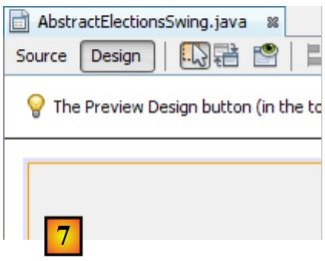


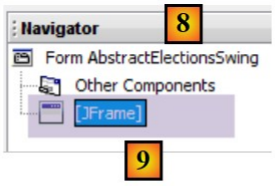
- [1] : ajout d'un objet au paquetage [elections.ui.service]
- [2] : choix de l'option [JFrame Form] dans la catégorie [Swing GUI Forms]

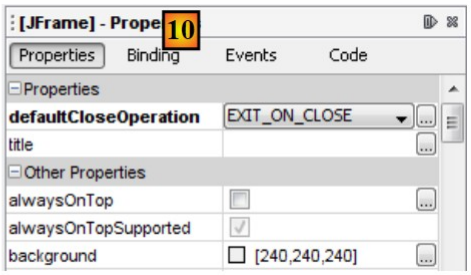




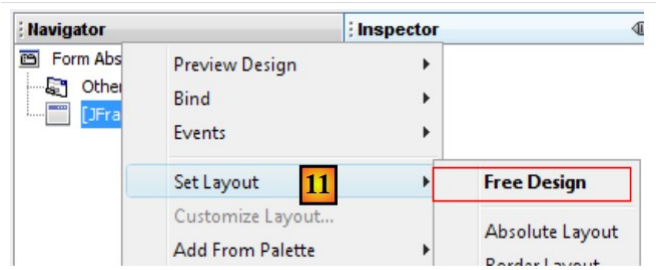
- [4] : donner un nom à la classe
- [5] : le package de la classe.
- Terminer l'assistant.
- [6] : la classe générée





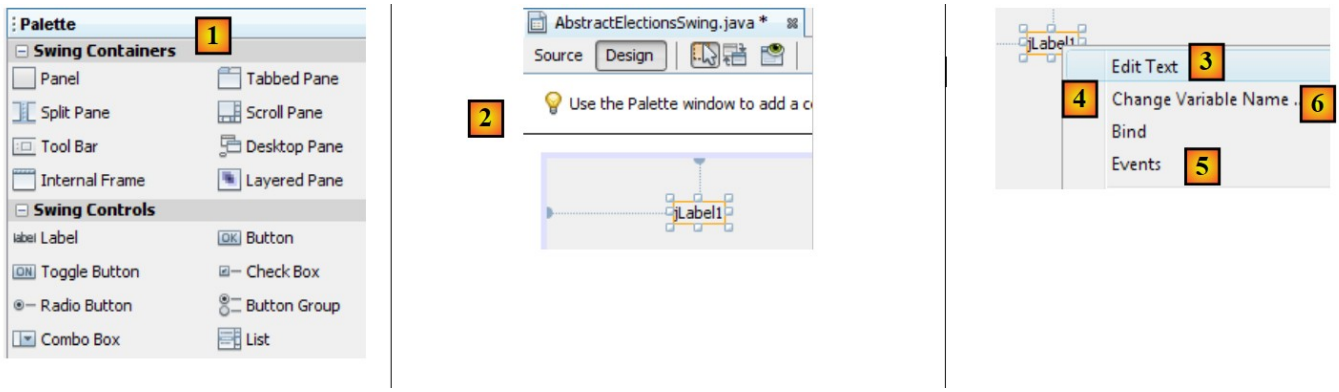


- [7] : la classe [AbstractElectionsSwing] en mode [Design]
- [8] : l'onglet [Navigator] qui affiche l'arbre [9] des composants de la fenêtre
- [10] : l'onglet [Properties] qui affiche les propriétés du composant [jFrame] sélectionné en [9]

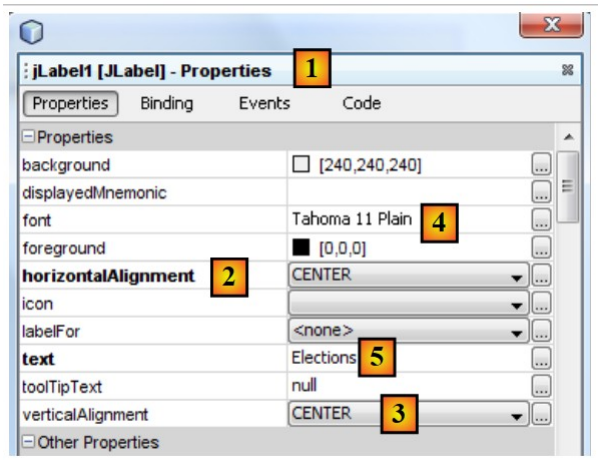


- [11] : [JFrame] est un conteneur de composants. Ceux-ci peuvent être disposés dans le conteneur selon diverses règles de positionnement appelées *layouts*. Ici, nous choisissons le *layout* [Free Design] [14] qui permet de positionner des composants de façon libre dans le conteneur.

Nous trouvons les composants dans la barre d'outils appelée *Palette* :



- [1] : la palette
- [2] : un composant JLabel est déposé dans le conteneur de composants
- avec un clic droit dessus, on a accès à diverses propriétés : son nom [4], son texte [3] ou bien ses gestionnaires d'événements [5]. Nous utilisons [3] pour fixer le texte [6].



- [1] : l'onglet [Properties] du composant [JLabel] donne accès aux propriétés de celui-ci : son positionnement horizontal [2], vertical [3], la police de caractères du texte [4], le texte [5].

Lorsqu'un composant est déposé et configuré sur l'interface graphique et qu'on sauvegarde (Ctrl-S) le travail fait, du code est généré dans la classe [AbstractElectionsSwing] :

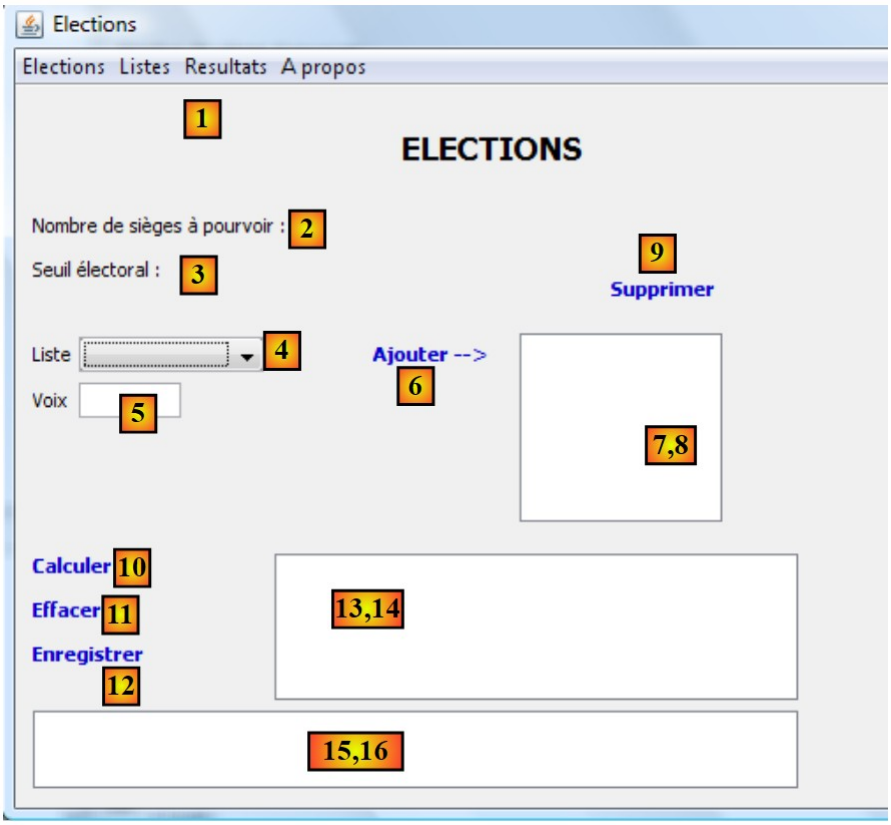
```
370
371     // Variables declaration - do not modify
372     protected javax.swing.JComboBox<String> jComboBoxNomsListes;
373     protected javax.swing.JLabel jLabel1;
374     protected javax.swing.JLabel jLabel4;
375     protected javax.swing.JLabel jLabel5;
376     protected javax.swing.JLabel jLabelAjouter;
377     protected javax.swing.JLabel jLabelCalculer;
378     protected javax.swing.JLabel jLabelEffacer;
379     protected javax.swing.JLabel jLabelEnregistrer;
380     protected javax.swing.JLabel jLabelSAP;
```

```
17      */
18     @SuppressWarnings("unchecked")
19     // <editor-fold defaultstate="collapsed" desc="Generated Code">
20     private void initComponents() {
21
22       jLabel1 = new javax.swing.JLabel();
23       jLabelSAP = new javax.swing.JLabel();
24       jLabelSE = new javax.swing.JLabel();
25       jLabel4 = new javax.swing.JLabel();
26       jLabel5 = new javax.swing.JLabel();
27       jTextFieldVoixListe = new javax.swing.JTextField();
28       jComboBoxNomsListes = new javax.swing.JComboBox<>();
29       jScrollPane1 = new javax.swing.JScrollPane();
30       jListNomsVoix = new javax.swing.JList<>();
31       jLabelAjouter = new javax.swing.JLabel();
```

Il ne faut pas modifier ce code grisé car il est supprimé et régénéré lors de la sauvegarde suivante. Les modifications faites seraient alors perdues.

On trouvera un tutoriel de création d'une interface graphique avec Netbeans à l'URL [https://netbeans.org/kb/docs/java/quickstart-gui.html?print=yes#design] (novembre 2015).

Nous construisons maintenant l'interface suivante :

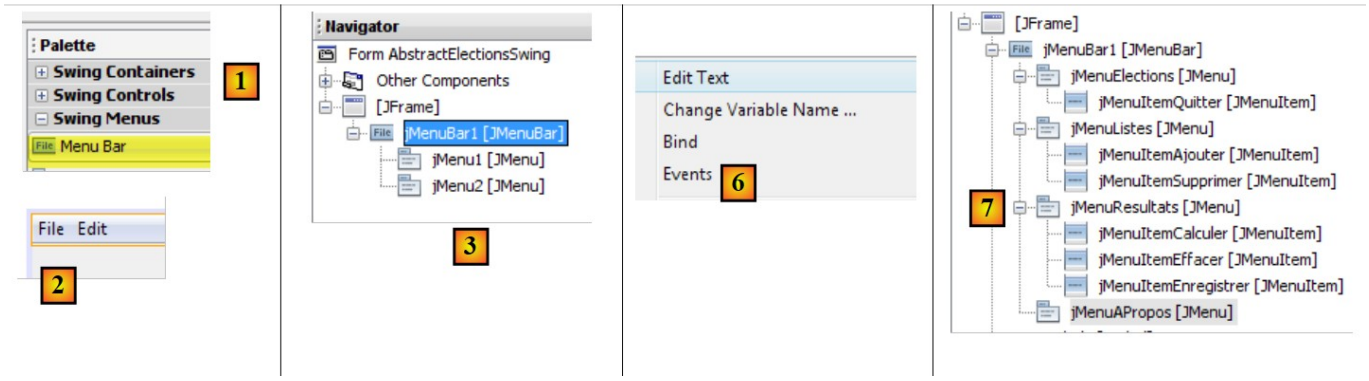


Les composants de l'interface sont les suivants :

n°	type	nom	rôle
1	JMenuBar	jMenuBar1	un menu
2	JLabel	jLabelSAP	le nombre de sièges à pourvoir
3	JLabel	jLabelSE	le seuil électoral
4	JComboBox	jComboBoxNomsListes	liste des noms des listes en compétition
5	JTextField	jTextFieldVoixListe	le nombre de voix d'une liste
6	JLabel	jLabelAjouter	pour ajouter une liste à (8)
7,8	(JScrollPane, JList)	jListNomsVoix	les noms et voix des listes
9	JLabel	jLabelSupprimer	pour supprimer de (8) la liste sélectionnée dans (8)
10	JLabel	jLabelCalculer	pour calculer les résultats de l'élection
11	JLabel	jLabelEffacer	pour effacer les résultats de l'élection
12	JLabel	jLabelEnregistrer	pour enregistrer les résultats de l'élection
13,14	(JScrollPane, JList)	jListResultats	pour afficher les résultats de l'élection
15,16	(JScrollPane, JTextPane)	jTextPaneMessages	pour afficher des messages de suivi

L'annotation (JScrollPane, JList) [13-14] est là pour indiquer que lorsqu'on dépose un composant [JList] dans la fenêtre, il est automatiquement inséré dans un composant [JScrollPane] qui permet le défilement de la liste. C'est le composant [JScrollPane] qui permet de voir tous les éléments de la liste alors que seul un nombre restreint d'entre-eux n'est visible à un moment donné. Il en est de même pour le composant [JTextPane] [15-16].

Le menu pourra être créé de la façon suivante :

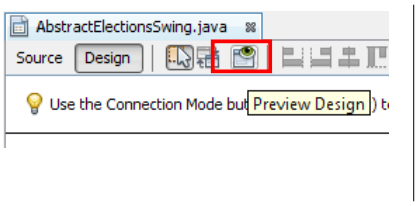


- [1, 2] : un composant [Menu Bar] est déposé sur la fenêtre
- [3] : le menu généré par défaut tel que présenté dans l'onglet [Navigator]
- [4,5,6] : par un clic droit sur une option de menu, on peut :
    - changer son texte [4], son nom [5]
    - gérer ses événements [6]
- [7] : le menu souhaité

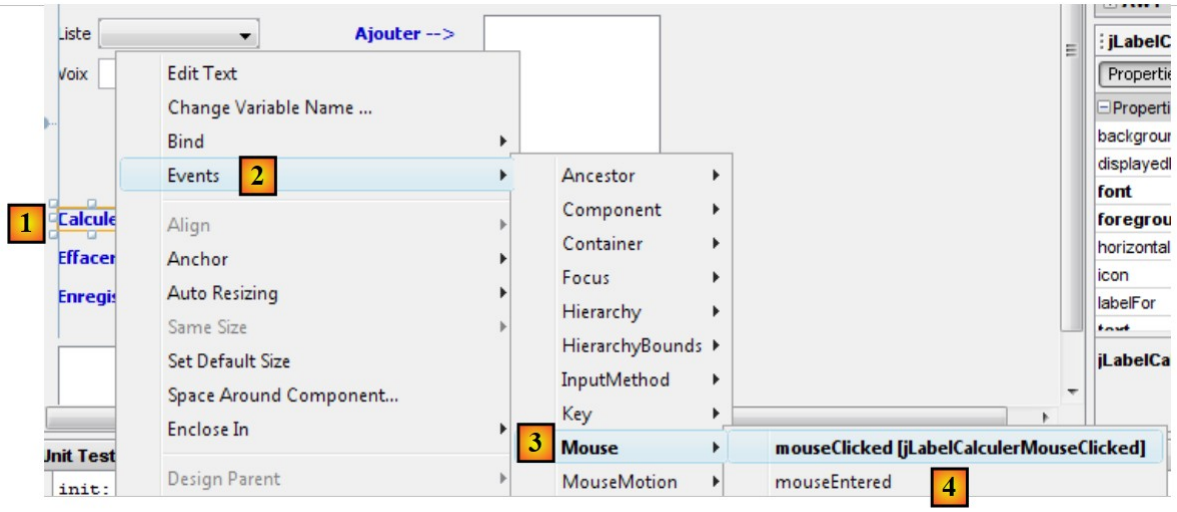
Le menu désiré est le suivant :

niveau 1	*niveau 2*
Elections	
	Quitter
Listes	
	Ajouter
	Supprimer
Résultats	
	Calculer
	Effacer
	Enregistrer
A propos	

On peut tester l'interface graphique à tout moment :



Lors de la construction de l'interface, il faut associer un gestionnaire d'événement sur certains labels et menus [Ajouter, Effacer, ...]. Voici comment faire :



- [1] : cliquer droit sur le composant dont on veut gérer un événement
- [2] : choisir l'option [Events]
- [3] : choisir une catégorie d'événements
- [4] : choisir l'événement qu'on veut gérer

Le code Java généré par cette opération est le suivant :

```
1.      jLabelCalculer.addMouseListener(new java.awt.event.MouseAdapter() {
2.        public void mouseClicked(java.awt.event.MouseEvent evt) {
3.          jLabelCalculerMouseClicked(evt);
4.        }
5.      });
6.  ...
7.
8.    private void jLabelCalculerMouseClicked(java.awt.event.MouseEvent evt) {
9.      // TODO add your handling code here:
10. }
```

- lignes 1-5 : un gestionnaire d'événement est ajouté au composant *jLabelCalculer*. La méthode **addMouseListener** attend comme paramètre une classe implémentant l'interface **MouseListener** suivante :

Method Summary

void	mouseClicked(MouseEvent e) Invoked when the mouse has been clicked on a component.
void	mouseEntered(MouseEvent e) Invoked when the mouse enters a component.
void	mouseExited(MouseEvent e) Invoked when the mouse exits a component.
void	mousePressed(MouseEvent e) Invoked when a mouse button has been pressed on a component.
void	mouseReleased(MouseEvent e) Invoked when a mouse button has been released on a component.

L'interface *MouseListener* est implémentée par différentes classes dont la classe *MouseAdapter*. Celle-ci implémente les cinq méthodes de l'interface *MouseListener* mais ces méthodes ne font rien. Aussi faut-il dériver cette classe pour implémenter la ou les méthodes de son choix. Ce qui est fait dans le code ci-dessus et repris ci-dessous :

```
1.      jLabelCalculer.addMouseListener(new java.awt.event.MouseAdapter() {
2.        public void mouseClicked(java.awt.event.MouseEvent evt) {
3.          jLabelCalculerMouseClicked(evt);
4.        }
5.    });
```

Le code ci-dessus utilise la technique de la classe anonyme exposée au paragraphe 2.5 du cours [ref1].

Ligne 1, le paramètre de la méthode *addMouseListener* est une classe anonyme, définie à la volée. C'est une instance d'une classe dérivée de la classe *MouseAdapter* (ligne 1) dont on redéfinit la méthode *mouseClicked* (lignes 2-4) afin qu'elle fasse quelque chose.

La méthode *jLabelCalculerMouseClicked* appelée ligne 3 est définie comme suit :

```
1.   private void jLabelCalculerMouseClicked(java.awt.event.MouseEvent evt) {
2.     // TODO add your handling code here:
3. }
```

Le développeur gère l'événement "*MouseClicked*" en mettant du code dans cette méthode.

Tous les gestionnaires d'événements sont générés par Netbeans de cette façon. Le développeur peut ignorer les lignes de code générées par Netbeans pour associer une méthode à un événement d'un composant. Il peut se contenter de mettre son code en ligne 2 ci-dessus. Voici un exemple :

```
1.    private void jLabelCalculerMouseClicked(java.awt.event.MouseEvent evt) {
2.      System.out.println("Mouse Clicked");
3.  }
```

Si on exécute l'interface graphique et qu'on clique sur le lien [Calculer] on obtient un message sur la console :

- [1] : on clique deux fois sur le label [Calculer]
- [2] : le gestionnaire de cet événement a été exécuté et a produit les messages *mouseClicked* dans la console de Netbeans.

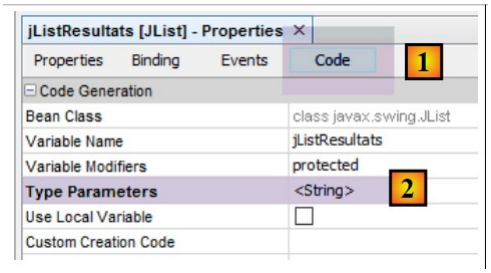
Les composants [jComboBoxNomsListes, jListNomsVoix, jListResultats] sont déclarés de la façon suivante :

```
1.  protected javax.swing.JComboBox jComboBoxNomsListes;
2.  protected javax.swing.JList jListNomsVoix;
3.  protected javax.swing.JList jListResultats;
```

Ces composants sont des listes qui normalement sont paramétrées par un type T : le type des éléments du modèle affiché par les composants. Ce type T, peut être quelconque. La valeur affichée dans le composant liste est de type [String]. Par défaut, c'est alors la méthode [T.toString()] qui est utilisée pour l'affichage. Afin de mieux maîtriser ce qui sera affiché, le type T sera ici le type String. Aussi, la déclaration correcte de nos listes est la suivante :

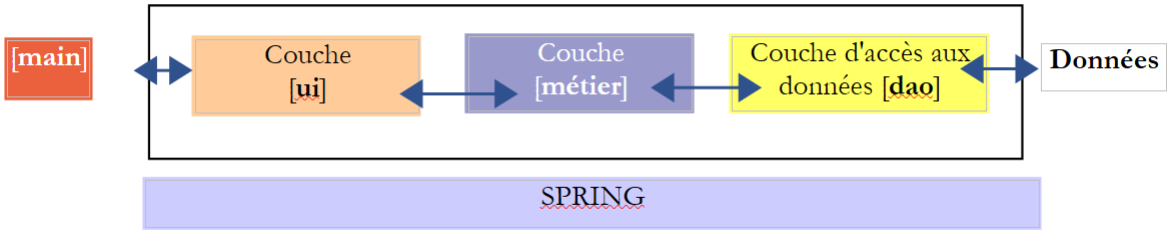
```
1. protected javax.swing.JComboBox<String> jComboBoxNomsListes;
2. protected javax.swing.JList<String> jListNomsVoix;
3. protected javax.swing.JList<String> jListResultats;
```

On obtient ce résultat en modifiant l'une des propriétés du composant :



10.3.4 Séparation du code

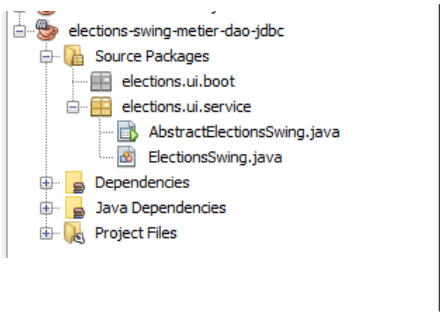
Revenons à la structure de notre application :



La classe [AbstractElectionsSwing] doit implémenter la couche [ui] ci-dessus. Son code généré par Netbeans ne contient pour l'instant que du code de gestion de la fenêtre et des gestionnaires d'événement qui pour l'instant ne font rien. Ci-dessus, on voit que la classe [AbstractElectionsSwing] devra gérer des échanges avec la couche [métier]. Cette gestion se fera dans les gestionnaires d'événement. Pour clarifier la structure du code, on décide de le placer dans deux classes :

- [AbstractElectionsSwing] qui restera comme elle a été générée par Netbeans à quelques détails près. Cette classe ne gèrera aucun événement elle-même. Les gestionnaires d'événements seront vides et déclarés abstraits. Ils seront implémentés par une classe dérivée de [AbstractElectionsSwing].
- [ElectionsSwing], classe dérivée de [AbstractElectionsSwing] qui implémentera tous les gestionnaires d'événement.

Ce type de séparation n'est pas inhabituel. On le trouve par exemple dans les pages web ASP.NET (version non MVC). Le projet Netbeans évolue comme suit :



Le code de la classe [AbstractElectionsSwing] lui, évolue de la façon suivante :

```
1. public abstract class AbstractElectionsSwing {
2. ....
3.   private void jMenuItemCalculerActionPerformed(java.awt.event.ActionEvent evt) {
4.     doCalculer();
5.   }
6.
7. ...
8.
9.   private void jLabelCalculerMouseClicked(java.awt.event.MouseEvent evt) {
10.       if (jLabelCalculer.isEnabled()) {
11.           doCalculer();
12.       }
13.   }
14.
15.  ....
16.   // gestionnaires d'événements
17.   abstract protected void doSupprimer();
18.
19.   abstract protected void doCalculer();
20.
21.   abstract protected void doQuitter();
22.
23.   abstract protected void doEffacer();
24.
25.   abstract protected void doEnregistrer();
26.
27.   abstract protected void doAjouter();
28.
29.   abstract protected void doInformer();
```

```
30.
31.    abstract protected void doMajLabelAjouter();
32.
33.    abstract protected void doMajLabelSupprimer();
34.  ...
35.  }
```

- ligne 1 : la classe est déclarée **abstraite**
- lignes 3-5 : gestion du clic sur sur l'option de menu [jMenuItemCalculer]. On voit que le traitement de l'événement est déporté sur la méthode *doCalculer* de la ligne 19. Cette méthode n'est pas implémentée et est déclarée abstraite. C'est la classe dérivée [ElectionsSwing] qui l'implémentera ;
- lignes 9-13 : le gestionnaire de l'événement *clic* sur le label [jLabelCalculer]. Le clic provoque toujours un événement, que le composant [jLabel] soit actif (enabled=true) ou inactif (enabled=false). On s'assure ici qu'il est bien actif pour traiter l'événement ;
- lignes 15 et au-delà : cette technique de délégation du traitement des événements vers une méthode abstraite est apliquée à tous les gestionnaires d'événements.

La classe [ElectionsSwing] dérivée de [AbstractElectionsSwing] implémente tous les gestionnaires d'événements non implémentés par [AbstractElectionsSwing] :

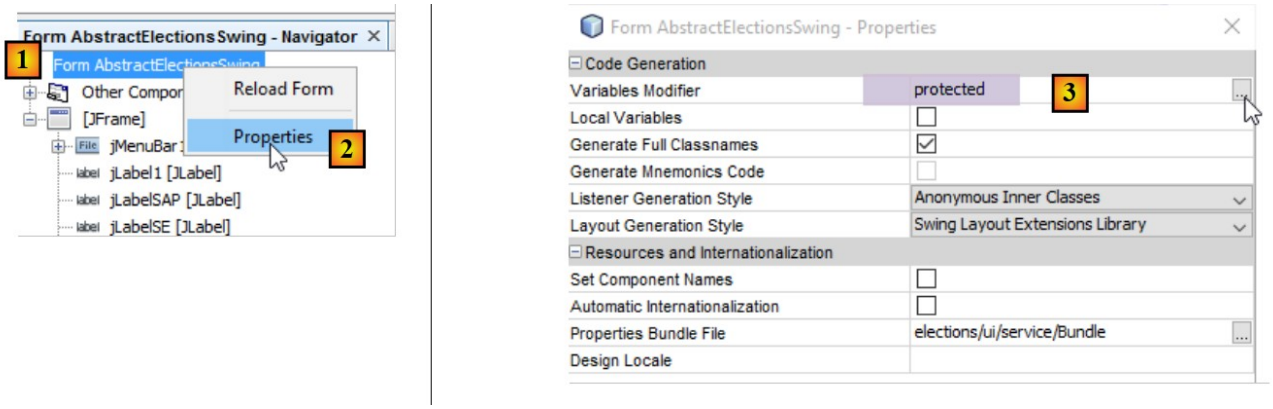
```
1. package elections.ui.service;;
2. ...
3. public class ElectionsSwing extends AbstractElectionsSwing {
4.
5.    // gestionnaires d'évts
6.
7.    @Override
8.    protected void doInformer() {
9. ...
10.     }
11.
12.     @Override
13.     protected void doAjouter() {
14.     ...
15.     }
16.
17.     @Override
18.     protected void doCalculer() {
19.     ...
20.     }
21.
22.     @Override
23.     protected void doEffacer() {
24.     ...
25.     }
26.
27.     @Override
28.     protected void doEnregistrer() {
29.  ...
30.     }
31.
32.     @Override
33.     protected void doQuitter() {
34.         System.exit(0);
35.     }
36.
37.     @Override
38.     protected void doSupprimer() {
39.  ...
40.     }
41.
42.     @Override
43.     protected void doMajLabelAjouter() {
44.  ...
45.     }
46.
47.     @Override
48.     protected void doMajLabelSupprimer() {
49.  ...
50.     }
51.
52.  }
```

- ligne 3 : [ElectionsSwing] dérive de [AbstractElectionsSwing]
- lignes 7-50 : les gestionnaires des événements de la fenêtre graphique

Les méthodes de la classe dérivée [ElectionsSwing] vont manipuler les composants de la classe parent [AbstractElectionsSwing]. Actuellement ces composants ont une portée *private*, interdisant à la classe fille [ElectionsSwing] d'y avoir accès :

```
1. private JMenuItem jMenuItemAPropos = null;
2.
3. private JLabel jLabelAjouter = null;
```

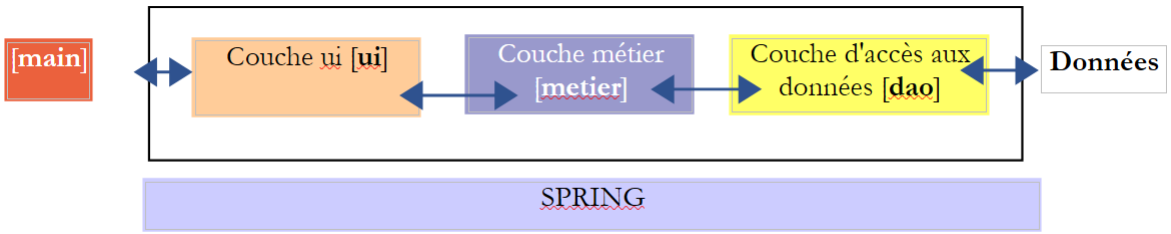
Pour résoudre ce problème, on fera en sorte que la portée des composants de l'interface graphique soit [**protected**] :



- mettre en [3], l'attribut [protected] ;

10.3.5 Implémentation de l'interface [IElectionsUI]

Revenons à la structure de notre application :



Ci-dessus la couche [ui] doit présenter l'interface [IElectionsUI] à l'objet [main] :

```
1.  package elections.ui.service;
2.
3.  public interface IElectionsUI {
4.      /**
5.       * lance le dialogue avec l'utilisateur
6.       */
7.      public void run();
8.  }
```

Cette interface a été définie dans le projet [elections-console-metier-dao-jdbc] et décrite au paragraphe 9.4, page 134. Comme ce projet est une dépendance du projet [swing], cette interface est connue.

Parce que la classe [AbstractElectionsSwing] est devenue abstraite, elle ne peut plus être instanciée par Spring. C'est la classe [ElectionsSwing] qui doit désormais l'être. La classe [ElectionsSwing] doit implémenter l'interface [IElectionsUI]. Son code évolue donc comme suit :

```
1.  public class ElectionsSwing extends AbstractElectionsSwing implements IElectionsUI {
2.
3.    // méthode run de l'interface [ElectionsUI]
4.    public void run() {
5.  ...
6.    }
7.
```

- ligne 1 : la classe [ElectionsSwing] implémente l'interface [IElectionsUI]
- lignes 4-6 : la méthode [run] de cette interface

Que doit faire la méthode *run* ? Afficher la fenêtre graphique. Comment fait-on cela ? On peut se laisser guider par la méthode [main] qu'à générée Netbeans dans la classe [AbstractElectionsSwing] et qui fait ce qui est souhaité :

```
1.  public static void main(String args[]) {
2.      /* Set the Nimbus look and feel */
3.      //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
4.      /* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
5.           * For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
6.       */
7.      try {
8.        for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {
9.          if ("Nimbus".equals(info.getName())) {
10.           javax.swing.UIManager.setLookAndFeel(info.getClassName());
11.           break;
12.         }
13.       }
14.     } catch (ClassNotFoundException ex) {
15.       java.util.logging.Logger.getLogger(NewJFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
16.     } catch (InstantiationException ex) {
17.       java.util.logging.Logger.getLogger(NewJFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
18.     } catch (IllegalAccessException ex) {
19.       java.util.logging.Logger.getLogger(NewJFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
20.     } catch (javax.swing.UnsupportedLookAndFeelException ex) {
21.       java.util.logging.Logger.getLogger(NewJFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
22.     }
23.     //</editor-fold>
24.
25.     /* Create and display the form */
26.     java.awt.EventQueue.invokeLater(new Runnable() {
27.       public void run() {
28.         new NewJFrame().setVisible(true);
29.       }
30.     });
31.   }
```

Le constructeur [`AbstractElectionsSwing`] utilisé ligne 28 est le suivant :

```
1.    public AbstractElectionsSwing() {
2.      initComponents();
3.  }
```

- ligne 2 : la méthode [initComponents] est une méthode privée générée par le générateur de l'interface graphique. On ne peut changer son code.

La méthode [run] de la classe [ElectionsSwing] pourrait alors être la suivante :

```
1.    @Override
2.    public void run() {
3.      // on affiche l'interface graphique
4.      SwingUtilities.invokeLater(new Runnable() {
5.        public void run() {
6.          init();
7.          setVisible(true);
8.        }
9.      });
10. }
```

- ligne 6 : l'interface graphique est initialisée au moyen d'une méthode [init]. Ici, on voudrait appeler la méthode [initComponents] de la classe parent, mais celle-ci est privée. On ajoute alors dans la classe parent [AbstractElectionsSwing], la méthode [init] suivante :

```
1.    protected void init(){
2.      initComponents();
3.  }
```

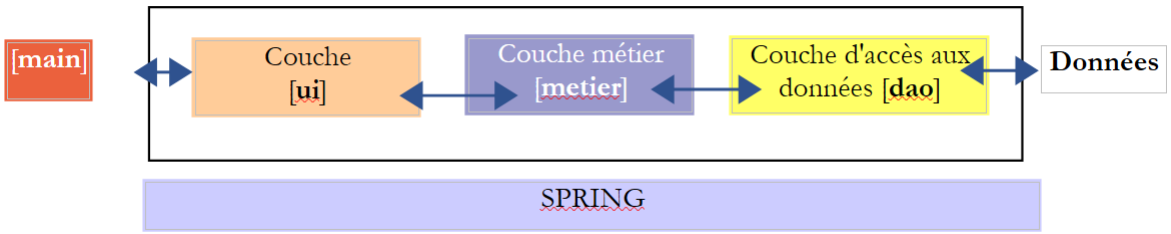
  - parce qu'elle est dans la classe [AbstractElectionsSwing], la méthode [init] a accès à la méthode privée [initComponents] de la même classe ;
  - parce qu'elle a l'attribut [protected], elle est visible dans la classe fille [ElectionsSwing] ;

- ligne 7 : l'interface graphique est rendue visible ;

Note : lorsque la méthode [run] a été écrite dans la classe [ElectionsSwing], la méthode [main] de la classe abstraite [AbstractElectionsSwing] peut être supprimée.

10.3.6 La classe exécutable

Revenons à la structure de notre application :

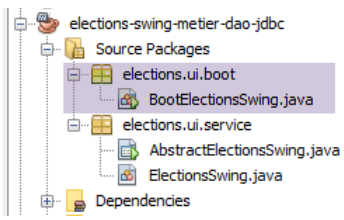


Nous voudrions que Spring instancie la couche [ui] comme il a été fait lorsque celle-ci était implémentée par une application *console*. Pour cela, il faut que la classe d'implémentation [ElectionsSwing] ait une référence sur la couche [métier] :

```
1. @Component
2. public class ElectionsSwing extends AbstractElectionsSwing implements IElectionsUI{
3.
4.   // référence sur la couche [métier]
5.   @Autowired
6.   private IElectionsMetier metier;
7. ...
```

- ligne 1 : la classe [ElectionsSwing] est un composant Spring ;
- lignes 5-6 : injection par Spring d'une référence sur la couche [métier] ;

L'interface graphique est lancée par l'exécution de la classe [BootElectionsSwing] suivante :

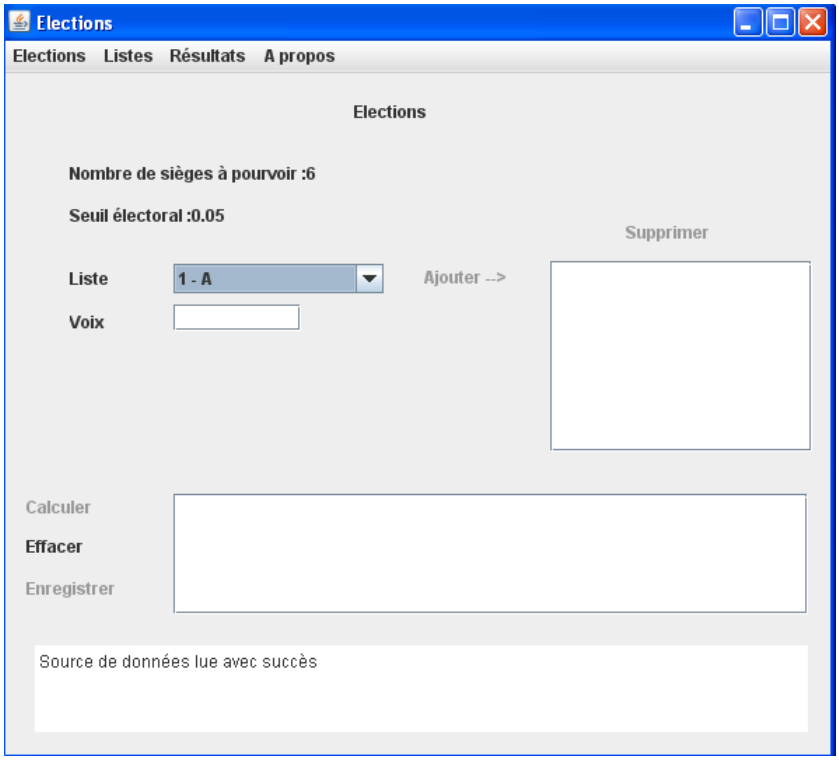



```
1. package elections.ui.boot;
2.
3. import elections.ui.service.IElectionsUI;
4.
5. public class BootElectionsSwing extends AbstractBootElections {
6.   public static void main(String[] arguments) {
7.     new BootElectionsSwing().run();
8.   }
9.
10.    @Override
11.    protected IElectionsUI getUI() {
12.        return ctx.getBean("electionsSwing", IElectionsUI.class);
13.    }
14. }
```

Nous avons expliqué un code analogue lorsqu'au paragraphe 9.5, page 134, nous avons expliqué le code des classes [AbstractBootElections] et [BootElectionsConsole]. Ligne 12, on récupère le bean nommé [electionsSwing] qui correspond au nom Spring standard pour la classe [ElectionsSwing].

10.3.7 Initialisation de l'interface graphique

Lorsque l'interface graphique est affichée, certains de ses composants ont été initialisés :



On voit ci-dessus :

- que le combo a été rempli avec les noms des listes ;
- que le nombre de sièges à pourvoir et le seuil électoral sont indiqués ;
- que certains liens ont été désactivés ;
- qu'un message de succès est affiché en bas de la fenêtre ;

A quel moment vont se faire ces initialisations ? Elles ne peuvent se faire qu'après initialisation du champ [electionsMetier] de la classe [ElectionsSwing]. En effet, les noms des listes vont être demandés à la couche [metier]. L'initialisation de ce champ va être faite par Spring dans l'ordre suivant :

- utilisation du constructeur sans paramètres de la classe [ElectionsSwing] ;
- injection des dépendances, ici la référence de la couche [métier] ;
- exécution de la méthode [run] de la classe [ElectionsSwing] :

```
1.  @Override
2.  public void run() {
3.    // on affiche l'interface graphique
4.    SwingUtilities.invokeLater(new Runnable() {
5.        public void run() {
6.            init();
7.            setVisible(true);
8.        }
9.    });
10. }
```

- ligne 12, nous avons dit qu'on appelait la méthode [init] de la classe parent qui va dessiner les composants de l'interface graphique. Nous allons redéfinir cette méthode, localement, dans la classe [ElectionsSwing]. C'est dans cette méthode que nous initialiserons, avec des données cette fois, les composants de la fenêtre (combo, labels) :

La méthode locale [init] pourrait avoir le squelette suivant :

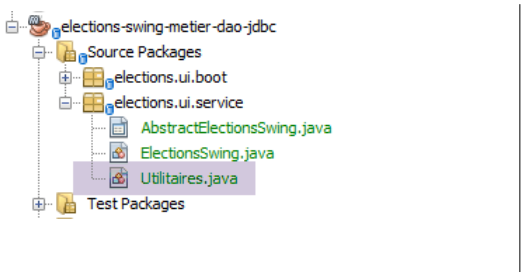
```
1. public class ElectionsSwing extends AbstractElectionsSwing implements IElectionsUI {
2.
3.   ...
4.   // référence sur la couche [métier]
5.   @Autowired
6.   private IElectionsMetier metier;
7.
8.   // initialisations
9.   public void init() {
10.       // génération des composants par la classe parent
11.      super.init();
12.
13.    // on demande les listes à la couche [metier]
```

```
14.       ...
15.       // on associe les noms des listes au combo jComboBoxNomsListes
16.       ...
17.       // ainsi que les paramètres de l'élection
18.       ...
19.       // on initialise les labels liés à ces deux informations
20.       ...
21.       // message de succès
22.       ...
23.       // initialisation état de certains composants formulaire
24.   ...
25.   }
```

On notera bien, ligne 11, l'appel de la méthode [init] de la classe parent.

10.3.8 La classe [Utilitaires]

Un certain nombre de méthodes utilitaires statiques ont été rassemblées dans la classe [Utilitaires] :



La classe [Utilitaires] est la suivante :

```
1.  package istia.st.elections.ui;
2.
3.  import javax.swing.JLabel;
4.  import javax.swing.JMenuItem;
5.
6.  //classe utilitaire
7.  class Utilitaires {
8.    // gérer l'état d'un tableau de labels
9.    public static void setEnabled(JLabel[] labels, boolean value) {
10.       for (int i = 0; i < labels.length; i++) {
11.           labels[i].setEnabled(value);
12.       }
13.   }
14.
15.   // gérer l'état d'un tableau d'options de menu
16.   public static void setEnabled(JMenuItem[] menuItems, boolean value) {
17.       ...
18.   }
19.
20. }
```

- ligne 9 : la méthode *setEnabled* fixe l'état de composants *JLabel* définis dans un tableau. La méthode *setEnabled* d'un composant *JLabel* permet d'activer ou de désactiver le *JLabel*.

Travail à faire : en suivant l'exemple de la méthode *setEnabled* de la ligne 9, écrivez la méthode *setEnabled* de la ligne 16 qui fait la même chose avec des composants *JMenuItem*.

10.3.9 Le code de la classe [ElectionsSwing]

Rappelons la structure générale de la classe [ElectionsSwing] :

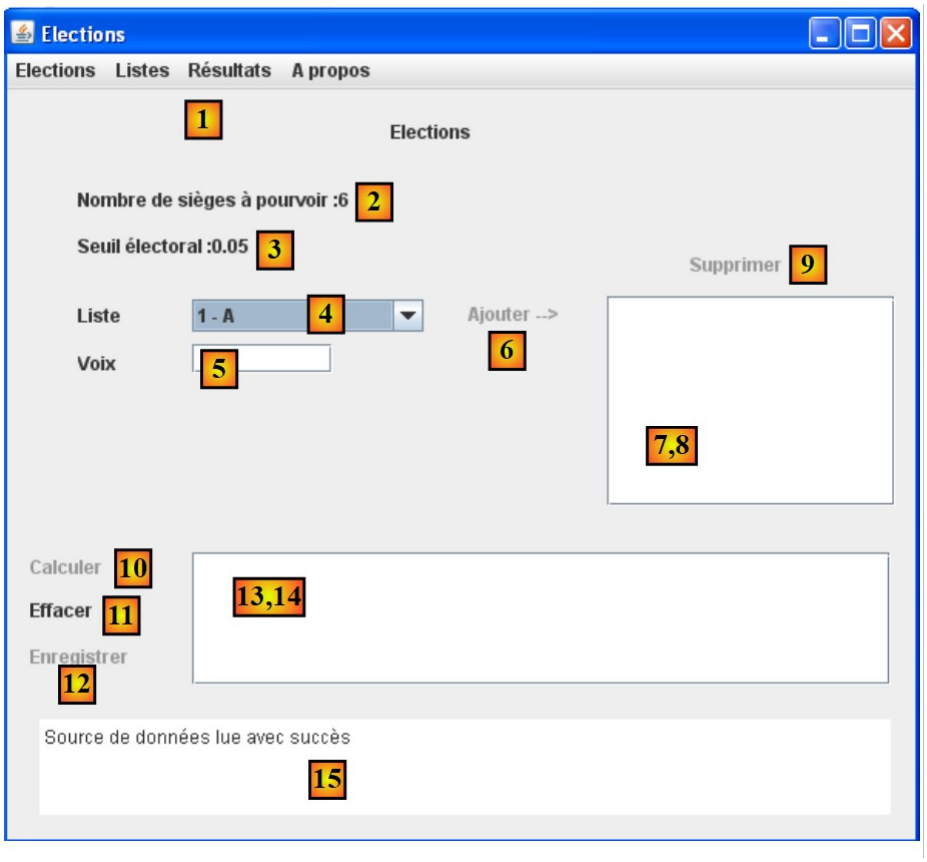
```
1. package istia.st.elections.ui;
2.
3. ...
4. public class ElectionsSwing extends AbstractElectionsSwing implements IElectionsUI {
5.
6.   // référence sur la couche [métier]
7.   @Autowired
8.   private IElectionsMetier metier;
9.
10.
11.    // initialisations
12.    public void init() {
```

```
13.    ...
14.    }
15.
16.    // gestionnaires d'évts
17.
18.    @Override
19.    protected void doInformer() {
20.  ...
21.    }
22.
23.    @Override
24.    protected void doAjouter() {
25.  ...
26.    }
27.
28.    @Override
29.    protected void doCalculer() {
30.    ...
31.    }
32.
33.    @Override
34.    protected void doEffacer() {
35.  ...
36.    }
37.
38.    @Override
39.    protected void doEnregistrer() {
40.  ...
41.    }
42.
43.    @Override
44.    protected void doQuitter() {
45.        System.exit(0);
46.    }
47.
48.    @Override
49.    protected void doSupprimer() {
50.  ...
51.    }
52.
53.    @Override
54.    protected void doMajLabelAjouter() {
55.    ...
56.    }
57.
58.    @Override
59.    protected void doMajLabelSupprimer() {
60.  ...
61.    }
62.
63.  }
```

Nous allons étudier les méthodes de la classe les unes après les autres.

10.3.9.1 La méthode [init]

Revenons sur l'interface graphique :



La méthode [init] a pour objectifs :

- de remplir le combo [4] avec les identifiants et noms des listes sous la forme [id - nom]
- d'afficher un message de succès dans [15]
- d'initialiser les labels [2] et [3]
- de désactiver certains liens

Le squelette de la méthode [init] pourrait être le suivant :

```
1. @Override
2.   protected void init() {
3.     // génération des composants par la classe parent
4.     super.init();
5.     // initialisations locales
6.     modèleNomsVoix = new DefaultListModel<>();
7.     jListNomsVoix.setModel(modèleNomsVoix);
8.     modèleRésultats = new DefaultListModel<>();
9.     jListResultats.setModel(modèleRésultats);
10.        String info;
11.        try {
12.            // on demande les listes à la couche [métier]
13.            listes = ...
14.            // on associe les noms des listes au combo jComboBoxNomsListes
15.            ...
16.            // ainsi que les paramètres de l'élection
17.            int nbSiegesAPourvoir = ...
18.            double seuilElectoral = ...
19.            // on initialise les labels liés à ces deux informations
20.            ...
21.            // message de succès
22.            info = "Source de données lue avec succès";
23.        } catch (ElectionsException ex1) {
24.            // on note l'erreur
25.            info = getInfoForException("Les erreurs suivantes se sont produites :", ex1);
26.        } catch (RuntimeException ex2) {
27.            // on note l'erreur
28.            info = getInfoForException("Les erreurs suivantes se sont produites :", ex2);
29.        }
30.        // on affiche l'info
31.        jTextPaneMessages.setText(info);
32.        jTextPaneMessages.setCaretPosition(0);
33.        // état formulaire
```

```
34.       Utilitaires.setEnabled(new JLabel[] { jLabelAjouter, jLabelCalculer, jLabelEnregistrer, jLabelSupprimer }, false);
35.       Utilitaires.setEnabled(
36.               new JMenuItem[] { jMenuItemAjouter, jMenuItemCalculer, jMenuItemEnregistrer, jMenuItemSupprimer }, false);
37.       // centrer la fenêtre
38.       Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
39.       Dimension frameSize = getSize();
40.       if (frameSize.height > screenSize.height) {
41.           frameSize.height = screenSize.height;
42.       }
43.       if (frameSize.width > screenSize.width) {
44.           frameSize.width = screenSize.width;
45.       }
46.       setLocation((screenSize.width - frameSize.width) / 2, (screenSize.height - frameSize.height) / 2);
47.   }
```

```
48.   private String getInfoForException(String message, ElectionsException ex) {
49.       // on affiche le message
50.       StringBuffer info = new StringBuffer(String.format("%s ------------\n", message));
51.       info.append(String.format("Code erreur : %d\n", ex.getCode()));
52.       // on affiche les erreurs
53.       for (String erreur : ex.getErreurs()) {
54.           info.append(String.format("-- %s\n", erreur));
55.       }
56.       return info.toString();
57.   }
58.
59.   private String getInfoForException(String message, RuntimeException ex) {
60.       // on affiche le message
61.       StringBuffer info = new StringBuffer(String.format("%s ------------\n", message));
62.       // on affiche la pile des exceptions
63.       Throwable cause = ex;
64.       while (cause != null) {
65.           info.append(String.format("-- %s\n", cause.getMessage()));
66.           cause = cause.getCause();
67.       }
68.       return info.toString();
69.   }
```

Travail à faire : complétez le code de la méthode [init].

A lire dans le cours : composants *JTextField*, *JLabel*

A savoir :

Un composant *JList* affiche les données présentes dans un modèle. Par défaut, ce modèle est de type *DefaultListModel* (lignes 2 et 3). Un objet de *DefaultListModel* se comporte un peu comme un type *ArrayList* :

- pour ajouter un objet *o* dans le modèle :

```
[DefaultListModel].addElement(Object o) ;
```

  Dans cette application, l'objet *o* sera toujours de type *String*.

- pour enlever l'élément n° i du modèle :

```
[DefaultListModel].remove(int i) ;
```

- pour obtenir l'élément n° i du modèle :

```
[DefaultListModel].elementAt(int i) ;
```

Pour ajouter un élément au combo [jComboBoxNomsListes], on utilisera la méthode [addItem] :

```
jComboBoxNomsListes.addItem(chaîne de caractères)
```

Un composant *JTextPane* a les méthodes *getText*() et *setText*() pour lire / écrire le texte affiché.

10.3.9.2 Gérer l'état du lien [Ajouter]

Le lien [Ajouter] [6] n'est actif que lorsque le champ [5] des voix est non vide. Dans la classe [AbstractElectionsSwing], le gestionnaire qui suit les mouvements du curseur dans le champ [5] est le suivant :

```
1.    private void jTextFieldVoixListeCaretUpdate(javax.swing.event.CaretEvent evt) {
2.      doMajLabelAjouter()
3.  }
```

La ligne 2 appelle la méthode [doMajLabelAjouter] de la classe [ElectionsSwing].

```
1.   protected void doMajLabelAjouter() {
2.     // on fixe l'état du label [jLabelAjouter]
3.     ...
4.     // on fixe l'état du menu [jMenuItemAjouter]
5.     ...
6. }
```

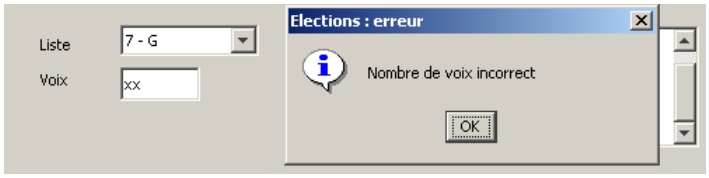
Travail à faire : complétez le code de la méthode [doMajLabelAjouter].

10.3.9.3 Affecter les voix à chaque liste

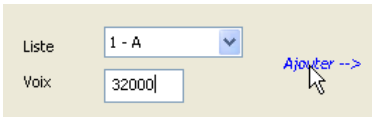
Pour chaque liste candidate de (4), on procède ainsi :

- choix d'une liste dans (4)
- saisie du nombre de voix dans (5)
- validation par un clic sur le lien [Ajouter]

Les erreurs de saisie sont signalées comme le montre l'exemple suivant :



Si le nombre de voix est correct, la liste est ajoutée dans le composant (8), le nombre de voix effacé et le lien [Ajouter] éteint :



Dans la classe [AbstractElectionsSwing], le gestionnaire qui gère le clic sur le lien [Ajouter] est le suivant :

```
1.   private void jLabelAjouterMouseClicked(java.awt.event.MouseEvent evt) {
2.     if (jLabelAjouter.isEnabled()) {
3.         doAjouter();
4.     }
5. }
```

La ligne 3 appelle la méthode [doAjouter] de la classe [ElectionsSwing] :

```
1.   // modèles des listes JList
2.   private DefaultListModel<String> modèleNomsVoix = null;
3.   private DefaultListModel<String> modèleRésultats = null;
4.
5.   // les listes en compétition
6.   private ListeElectorale[] listes;
7.
8.   // listes saisies par l'utilisateur
9.   private final List<ListeElectorale> listesSaisies = new ArrayList<>();
10.    private ListeElectorale[] tListesSaisies;
11.  ...
12.    @Override
13.    protected void doAjouter() {
14.      // le nombre de voix est-il correct ?
15.      ...
16.      // si erreur, alors on la signale
17.      if (erreur) {
18.        JOptionPane.showMessageDialog(null, "Nombre de voix incorrect", "Elections : erreur",
19.                JOptionPane.INFORMATION_MESSAGE);
20.        jTextFieldVoixListe.requestFocus();
21.        // retour à l'interface graphique
22.        return;
23.      }
```

```
24.      // pas d'erreur - on enregistre la liste
25.      listesSaisies.add(...);
26.      modèleNomsVoix.addElement(...);
27.      // on nettoie le nombre de voix
28.      jTextFieldVoixListe.setText("");
29.      // état formulaire (menus, labels)
30.  ...
31.  }
```

- ligne 25 : à chaque fois que l'utilisateur ajoute des voix à une liste et valide son choix, cette liste est mise dans le champ [listesSaisies] de la ligne 9. On y enregistrera la liste avec les informations [id, version, nom, voix]. Les trois premières informations viennent des listes enregistrées initialement dans le tableau de la ligne 6. La méthode [getSelectedIndex] du combo permet de connaître l'indice de la liste sélectionnée ;

Travail à faire : complétez le code de la méthode [doAjouter].

10.3.9.4 Gérer l'état du lien [Supprimer]

Le lien [Supprimer] [9] n'est actif que lorsqu'un élément est sélectionné dans [8].

Dans la classe [AbstractElectionsSwing], le gestionnaire qui réagit au clic sur un élément de la liste [8] est le suivant :

```
1.    private void jListNomsVoixValueChanged(javax.swing.event.ListSelectionEvent evt) {
2.      doMajLabelSupprimer();
3.  }
```

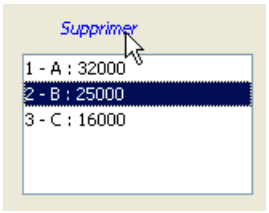
La ligne 2 appelle la méthode [doMajLabelSupprimer] de la classe [ElectionsSwing].

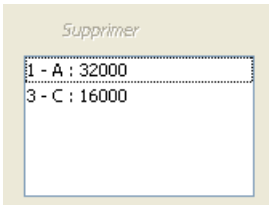
```
1.  @Override
2.  protected void doMajLabelSupprimer() {
3.        // on allume le label [jLabelSupprimer] et l'option de menu correspondante
4.        ...
5. }
```

Travail à faire : complétez le code de la méthode [doMajLabelSupprimer].

10.3.9.5 Supprimer une liste candidate

Le lien [Supprimer] [9] permet de supprimer le couple (nom,voix) sélectionné dans (8). Une fois la suppression opérée, le lien [Supprimer] est éteint. Il ne sera rallumé que sur sélection d'une nouvelle liste dans (8).





Dans la classe [AbstractElectionsSwing], le gestionnaire qui réagit au clic sur le lien [Supprimer] est le suivant :

```
1.    private void jLabelSupprimerMouseClicked(java.awt.event.MouseEvent evt) {
2.      if(jLabelSupprimer.isEnabled()){
3.        doSupprimer();
4.      }
5.  }
```

La ligne 3 appelle la méthode [doSupprimer] de la classe [ElectionsSwing].

```
1. @Override
2.  protected void doSupprimer() {
3.    // suppression de la liste sélectionnée, du modèle modèleNomsVoix et des listes saisies
4.    ...
5.    // maj de l'état des labels et options de menu du formulaire
6.    Utilitaires.setEnabled(...);
7.  ...
8. }
```

Travail à faire : complétez le code de la méthode [doSupprimer].

10.3.9.6 Gérer l'état du lien [Calculer]

Le lien [Calculer] [10] n'est actif que lorsqu'il existe au moins un élément dans [8].

Travail à faire : ajoutez le code nécessaire à la gestion de ce lien dans les méthodes [doAjouter] et [doSupprimer] écrites précédemment. L'option de menu correspondante sera également gérée.

A savoir : le nombre d'éléments d'un élément de type *DefaultListModel* est obtenu avec la méthode *size*().

10.3.9.7 Calculer les sièges

Le lien [Calculer] [10] permet de lancer le calcul des sièges et d'afficher les résultats dans (14). En cas d'échec (toutes les listes ont été éliminées), un message d'erreur est affiché dans [15]. Dans tous les cas, après calcul, le lien [Calculer] [10] est désactivé.

Dans la classe [AbstractElectionsSwing], le gestionnaire qui réagit au clic sur le lien [Calculer] est le suivant :

```
1.   private void jLabelCalculerMouseClicked(java.awt.event.MouseEvent evt) {
2.     if(jLabelCalculer.isEnabled()){
3.       doCalculer();
4.     }
5. }
```

La ligne 3 appelle la méthode [doCalculer] de la classe [ElectionsSwing].

```
1.    // listes saisies par l'utilisateur
2.    private final List<ListeElectorale> listesSaisies = new ArrayList<>();
3.    private ListeElectorale[] tListesSaisies;
4.
5.  ...
6.    @Override
7.    protected void doCalculer() {
8.      tListesSaisies = listesSaisies.toArray(new ListeElectorale[0]);
9.      // calcul des sièges
10.     try {
11.       ...
12.     } catch (ElectionsException ex) {
13.       // on affiche l'exception
14.       ...
15.       return;
16.     }
17.     // affichage des résultats
18.     ...
19.     // maj état formulaire
20.     Utilitaires.setEnabled(...);
21. }
```

Travail à faire : complétez le code de la méthode [doCalculer].

10.3.9.8 Enregistrer les résultats dans la source de données

Le lien [Enregistrer] (12) permet d'enregistrer les résultats du calcul des sièges dans la source de données. Une fois l'enregistrement opéré avec succès, le lien [Enregistrer] est désactivé. En cas d'échec, un message d'erreur est affiché dans [15]. Dans tous les cas, le lien [Enregistrer] est ensuite désactivé.

Dans la classe [AbstractElectionsSwing], le gestionnaire qui gère le clic sur le label [Enregistrer] est le suivant :

```
1.   private void jLabelEnregistrerMouseClicked(java.awt.event.MouseEvent evt) {
2.     if(jLabelEnregistrer.isEnabled()){
3.       doEnregistrer();
4.     }
5. }
```

La ligne 3 appelle la méthode [doEnregistrer] de la classe [ElectionsSwing] :

```
1.  @Override
2.   protected void doEnregistrer() {
3.     // on demande l'enregistrement à la couche métier
4.     try {
5.         ...
```

```
6.      } catch (ElectionsException ex) {
7.          // on affiche l'exception
8.          ...
9.          // retour à l'interface graphique
10.         return;
11.     }
12.     // maj du formulaire
13.     Utilitaires.setEnabled(...);
14. ...
15. }
```

Travail à faire : complétez le code de la méthode [doEnregistrer].

10.3.9.9 Effacer les résultats

Le lien [Effacer] (11) permet d'effacer les résultats affichés dans (14).

Dans la classe [AbstractElectionsSwing], le gestionnaire qui gère le clic sur le label [Effacer] est le suivant :

```
1.   private void jLabelEffacerMouseClicked(java.awt.event.MouseEvent evt) {
2.     if(jLabelEffacer.isEnabled()){
3.       doEffacer();
4.     }
5. }
```

La ligne 3 appelle la méthode [doEffacer] de la classe [ElectionsSwing] :

```
1.   @Override
2.   protected void doEffacer() {
3.     // on vide la liste des résultats
4.     ....
5.     // maj du formulaire
6.     Utilitaires.setEnabled(...);
7. }
```

Travail à faire : complétez le code de la méthode [doEffacer].

A savoir : la classe *DefaultListModel* a une méthode *clear*() qui supprime tous ses éléments.

10.3.10 Améliorations

L'interface graphique précédente peut être améliorée de diverses façons : l'utilisateur peut, par oubli, ne pas saisir les voix de toutes les listes présentes dans le combo et par ailleurs, il peut, par erreur, saisir plusieurs fois les voix d'une même liste.

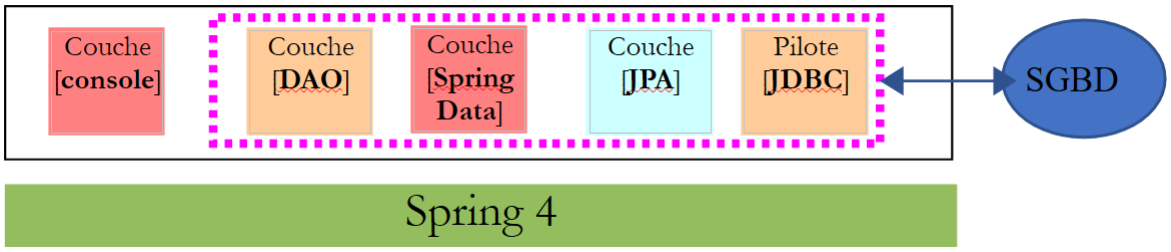
Travail à faire : améliorez l'algorithme afin que ces deux cas ne puissent pas se produire. Une solution simple est de gérer un dictionnaire des listes saisies dont les clés seraient les éléments du combo. On fera également en sorte que le lien [Calculer] ne soit allumé que lorsque la totalité des listes a été saisie.

A lire dans le cours [ref1] : la classe *HashTable* au paragraphe 3.8.

11 [Cours] : Gestion des bases de données relationnelles avec Spring Data

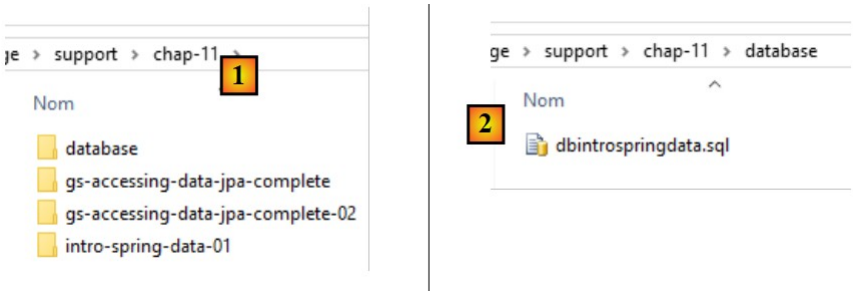
Mots clés : architecture multicouche, Spring, injection de dépendances, API JPA (Java Persistence API), Spring Data.

Nous allons implémenter la couche [DAO] du TD avec [Spring Data], une branche de l'écosystème Spring. [Spring Data] s'appuie sur une couche JPA (Java Persistence API) qui permet à la couche [DAO] de manipuler des objets plutôt que des ordres SQL. Au final, la couche [DAO] ignore qu'elle dialogue avec une base de données. Elle ne connaît que l'interface de la couche [Spring Data].



Nous allons d'abord découvrir [Spring Data] sur deux exemples.

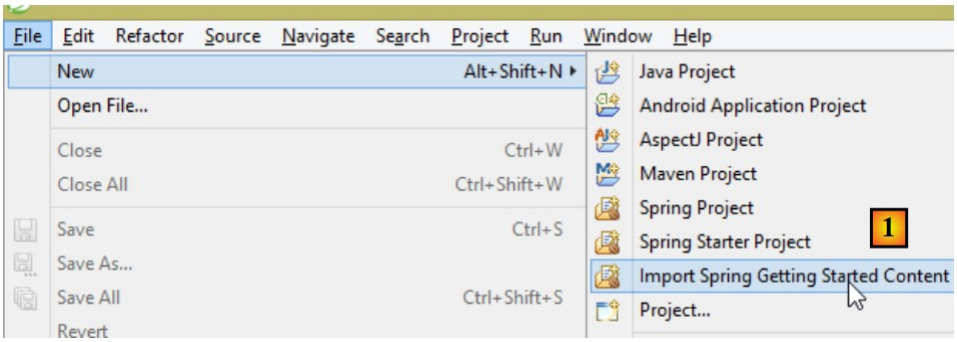
11.1 Support



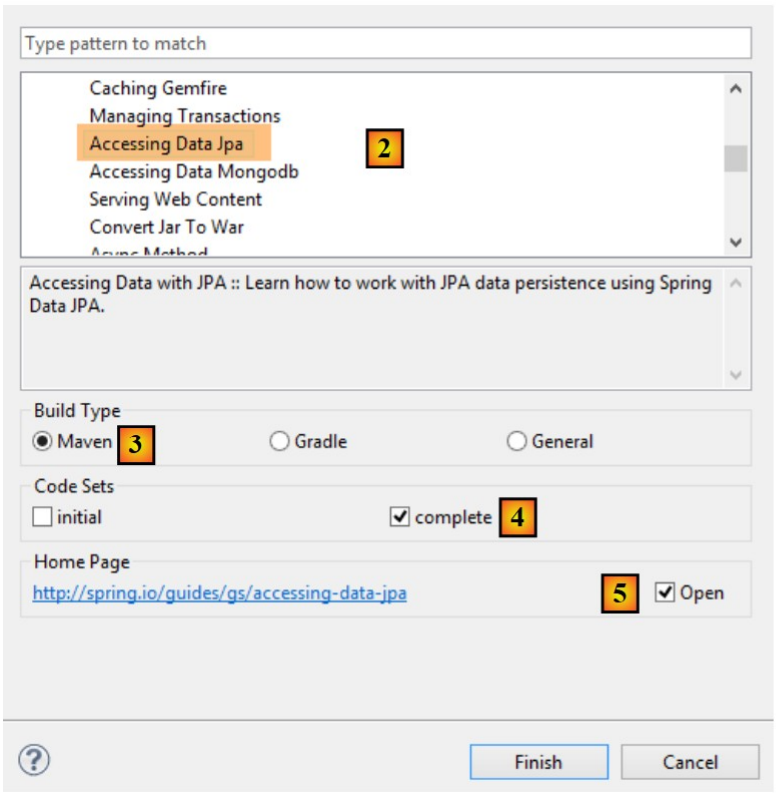
- en [1], le dossier [support / chap-11] contient trois projets Eclipse ;
- en [2], le script SQL permettant de créer la base de données exemple de ce chapitre ;

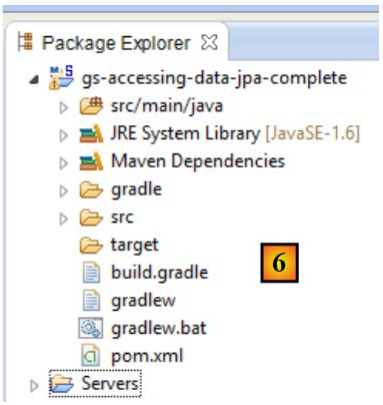
11.2 Exemple 1

Sur le site de Spring existent de nombreux tutoriels pour démarrer avec Spring [http://spring.io/guides]. Nous allons utiliser l'un d'eux pour introduire Spring Data. Nous utilisons pour cela Spring Tool Suite (STS).



- en [1], nous importons l'un des tutoriels de [spring.io/guides] ;





- en [2], on choisit le tutoriel [Accessing Data Jpa] qui montre comment accéder à une base de données avec Spring Data ;
- en [3], on choisit un projet configuré par Maven ;
- en [4], le tutoriel peut être délivré sous deux formes : [initial] qui est une version vide qu'on remplit en suivant le tutoriel ou [complete] qui est la version finale du tutoriel. Nous choisissons cette dernière ;
- en [5], on peut choisir de visualiser le tutoriel dans un navigateur ;
- en [6], le projet final.

11.2.1 La configuration Maven du projet

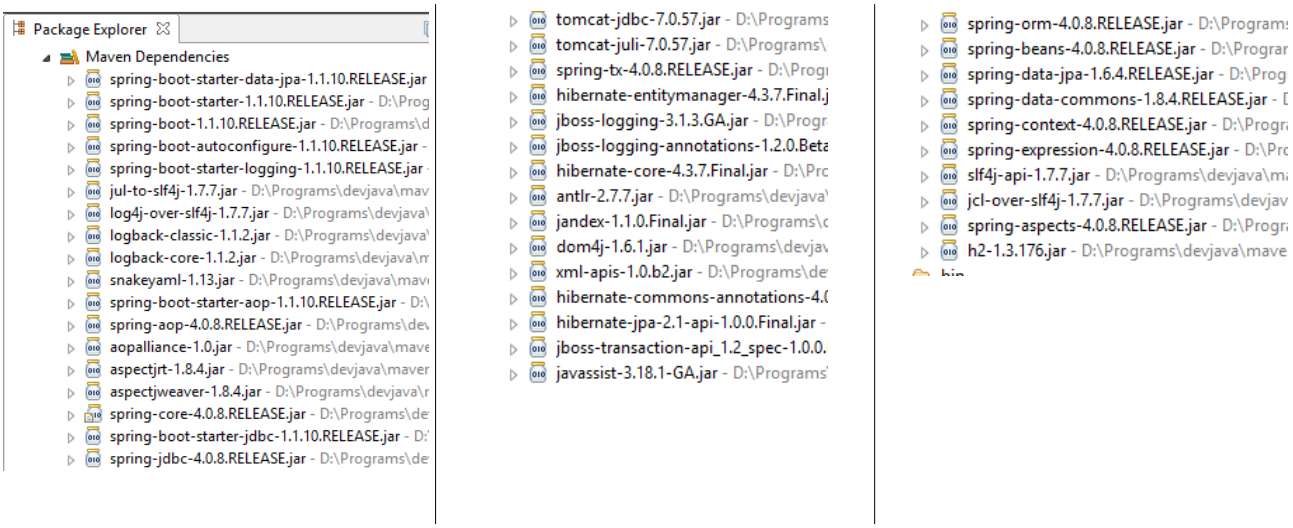
Les dépendances Maven du projet sont configurées dans le fichier [pom.xml] :

```
1.     <groupId>org.springframework</groupId>
2.     <artifactId>gs-accessing-data-jpa</artifactId>
3.     <version>0.1.0</version>
4.
5.     <parent>
6.         <groupId>org.springframework.boot</groupId>
7.         <artifactId>spring-boot-starter-parent</artifactId>
8.         <version>1.1.10.RELEASE</version>
9.     </parent>
10.
11.      <dependencies>
12.          <dependency>
13.              <groupId>org.springframework.boot</groupId>
14.              <artifactId>spring-boot-starter-data-jpa</artifactId>
15.          </dependency>
16.          <dependency>
17.              <groupId>com.h2database</groupId>
18.              <artifactId>h2</artifactId>
19.          </dependency>
20.      </dependencies>
21.
22.      <properties>
23.          <!-- use UTF-8 for everything -->
24.          <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
25.          <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
26.          <start-class>hello.Application</start-class>
27.  </properties>
```

- lignes 5-9 : définissent un projet Maven parent. C'est lui qui définit l'essentiel des dépendances du projet. Elles peuvent être suffisantes, auquel cas on n'en rajoute pas, ou pas, auquel cas on rajoute les dépendances manquantes ;

- lignes 12-15 : définissent une dépendance sur [spring-boot-starter-data-jpa]. Cet artifact contient les classes de Spring Data ;
- lignes 16-19 : définissent une dépendance sur le SGBD H2 qui permet de créer et gérer des bases de données en mémoire.

Regardons les classes amenées par ces dépendances :



Elles sont très nombreuses :

- certaines appartiennent à l'écosystème Spring (celles commençant par spring) ;
- d'autres appartiennent à l'écosystème Hibernate (hibernate, jboss) dont on utilise ici l'implémentation JPA ;
- d'autres sont des bibliothèques de tests (junit, hamcrest) ;
- d'autres des bibliothèques de logs (log4j, logback, slf4j) ;

Nous allons les garder toutes. Pour une application en production, il faudrait ne garder que celles qui sont nécessaires.

Ligne 26 du fichier [pom.xml] on trouve la ligne :

```
<start-class>hello.Application</start-class>
```

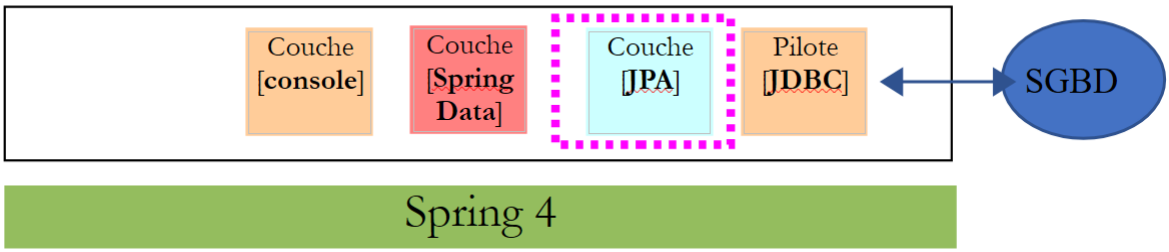
Cette ligne est liée aux lignes suivantes :

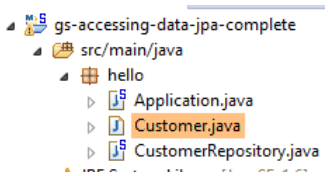
```
1. <build>
2.         <plugins>
3.             <plugin>
4.                 <artifactId>maven-compiler-plugin</artifactId>
5.             </plugin>
6.             <plugin>
7.                 <groupId>org.springframework.boot</groupId>
8.                 <artifactId>spring-boot-maven-plugin</artifactId>
9.             </plugin>
10.          </plugins>
11.      </build>
```

Lignes 6-9, le plugin [spring-boot-maven-plugin] permet de générer le jar exécutable de l'application. La ligne 26 du fichier [pom.xml] désigne alors la classe exécutable de ce jar.

11.2.2 La couche [JPA]

L'accès à la base de données se fait au travers d'une couche [JPA], Java Persistence API :





L'application est basique et gère des clients [Customer]. La classe [Customer] fait partie de la couche [JPA] et est la suivante :

```
1. package hello;
2.
3. import javax.persistence.Entity;
4. import javax.persistence.GeneratedValue;
5. import javax.persistence.GenerationType;
6. import javax.persistence.Id;
7.
8. @Entity
9. public class Customer {
10.
11.    @Id
12.    @GeneratedValue(strategy = GenerationType.AUTO)
13.    private long id;
14.    private String firstName;
15.    private String lastName;
16.
17.    protected Customer() {
18.    }
19.
20.    public Customer(String firstName, String lastName) {
21.        this.firstName = firstName;
22.        this.lastName = lastName;
23.    }
24.
25.    @Override
26.    public String toString() {
27.        return String.format("Customer[id=%d, firstName='%s', lastName='%s']", id, firstName, lastName);
28.    }
29.
30. }
```

Un client a un identifiant [id], un prénom [firstName] et un nom [lastName]. Chaque instance [Customer] représente une ligne d'une table de la base de données.

- ligne 8 : annotation JPA qui fait que la persistence des instances [Customer] (Create, Read, Update, Delete) va être gérée par une implémentation JPA. D'après les dépendances Maven, on voit que c'est l'implémentation JPA / Hibernate qui est utilisée ;
- lignes 11-12 : annotations JPA qui associent le champ [id] à la clé primaire de la table des [Customer]. La ligne 12, indique que l'implémentation JPA utilisera la méthode de génération de clé primaire propre au SGBD utilisé, ici H2 ;

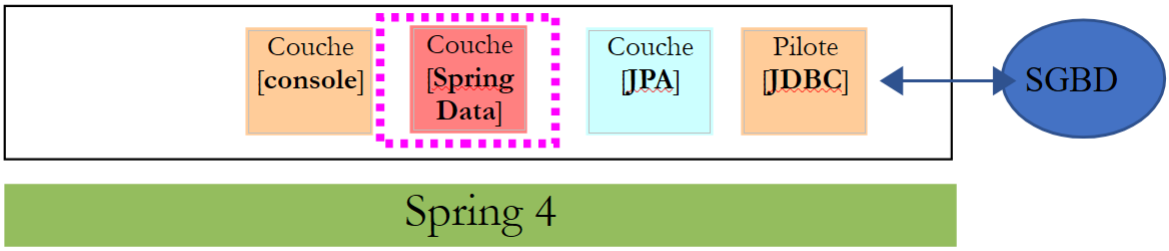
Il n'y a pas d'autres annotations JPA. Des valeurs par défaut seront alors utilisées :

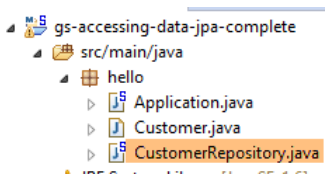
- la table des [Customer] portera le nom de la classe, ç-à-d [Customer] ;
- les colonnes de cette table porteront le nom des champs de la classe : [id, firstName, lastName] sachant que la casse n'est pas prise en compte dans le nom d'une colonne de table ;

On notera qu'à aucun moment, l'implémentation JPA utilisée n'est nommée.

11.2.3 La couche [Spring Data]

La classe [CustomerRepository] implémente la couche d'accès à la table [Customer]. Son code est le suivant :






```
1. package hello;
2.
3. import java.util.List;
4.
5. import org.springframework.data.repository.CrudRepository;
6.
7. public interface CustomerRepository extends CrudRepository<Customer, Long> {
8.
9.   List<Customer> findByLastName(String lastName);
10.  }
```

C'est donc une interface et non une classe (ligne 7). Elle étend l'interface [CrudRepository], une interface de Spring Data (ligne 5). Cette interface est paramétrée par deux types : le premier est le type des éléments gérés, ici le type [Customer], le second le type de la clé primaire des éléments gérés, ici un type [Long]. L'interface [CrudRepository] est la suivante :

```
1. package org.springframework.data.repository;
2.
3. import java.io.Serializable;
4.
5. @NoRepositoryBean
6. public interface CrudRepository<T, ID extends Serializable> extends Repository<T, ID> {
7.
8.   <S extends T> S save(S entity);
9.
10.   <S extends T> Iterable<S> save(Iterable<S> entities);
11.
12.   T findOne(ID id);
13.
14.   boolean exists(ID id);
15.
16.   Iterable<T> findAll();
17.
18.   Iterable<T> findAll(Iterable<ID> ids);
19.
20.   long count();
21.
22.   void delete(ID id);
23.
24.   void delete(T entity);
25.
26.   void delete(Iterable<? extends T> entities);
27.
28.   void deleteAll();
29. }
```

Cette interface définit les opérations CRUD (Create – Read – Update – Delete) qu'on peut faire sur un type JPA T :

- ligne 8 : la méthode **save** permet de **persister** une entité T en base. Elle rend l'entité persistée avec la clé primaire que lui a donnée le SGBD. Elle permet également de **mettre à jour** une entité T identifiée par sa clé primaire *id*. Le choix de l'une ou l'autre action se fait selon la valeur de la clé primaire *id* : si celle-ci vaut **null** c'est l'opération de persistence qui a lieu, sinon c'est l'opération de mise à jour ;
- ligne 10 : idem mais pour une liste d'entités ;
- ligne 12 : la méthode **findOne** permet de retrouver une entité T identifiée par sa clé primaire *id* ;
- ligne 22 : la méthode **delete** permet de supprimer une entité T identifiée par sa clé primaire *id* ;
- lignes 24-28 : des variantes de la méthode [delete] ;
- ligne 16 : la méthode [findAll] permet de retrouver toutes les entités persistées T ;
- ligne 18 : idem mais limitées aux entités dont on a passé la liste des identifiants ;

Revenons à l'interface [CustomerRepository] :

```
1. package hello;
2.
3. import java.util.List;
4.
5. import org.springframework.data.repository.CrudRepository;
6.
7. public interface CustomerRepository extends CrudRepository<Customer, Long> {
8.
9.   List<Customer> findByLastName(String lastName);
10.  }
```

- la ligne 9 permet de retrouver un [Customer] par son nom [lastName] ;

Et c'est tout pour la couche [DAO]. Il n'y a pas de classe d'implémentation de l'interface précédente. Celle-ci est générée à l'exécution par [Spring Data]. Les méthodes de l'interface [CrudRepository] sont automatiquement implémentées. Pour les méthodes rajoutées dans l'interface [CustomerRepository], ça dépend. Revenons à la définition de [Customer] :

```
1. private long id;
2. private String firstName;
3. private String lastName;
```

La méthode de la ligne 9 est implémentée automatiquement par [Spring Data] parce qu'elle référence le champ [lastName] (ligne 3) de [Customer]. Lorsqu'il rencontre une méthode [findBySomething] dans l'interface à implémenter, Spring Data l'implémente par la requête JPQL (Java Persistence Query Language) suivante :

```
select t from T t where t.something=:value
```

Il faut donc que le type T ait un champ nommé [something]. Ainsi la méthode

```
List<Customer> findByLastName(String lastName);
```

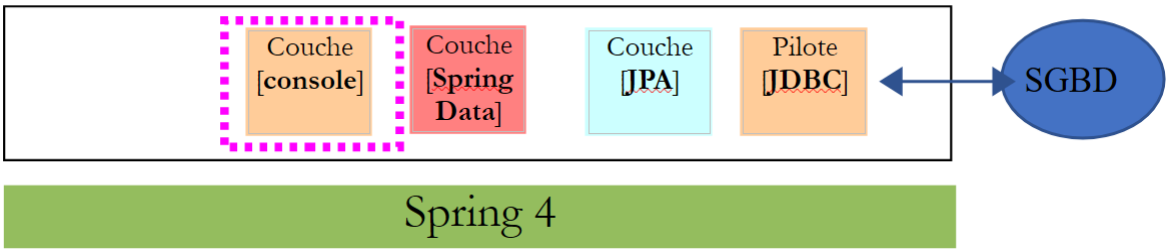
va être implémentée par un code ressemblant au suivant :

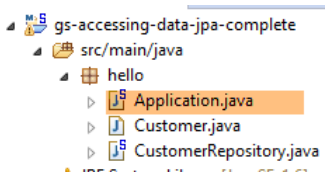
```
return [em].createQuery("select c from Customer c where  c.lastName=:value").setParameter("value",lastName).getResultList()
```

où [em] désigne le contexte de persistance JPA. Cela n'est possible que si la classe [Customer] a un champ nommé [lastName], ce qui est le cas.

En conclusion, dans les cas simples, Spring Data nous permet d'implémenter la couche [DAO] avec une simple interface.

11.2.4 La couche [console]





La classe [Application] est la suivante :

```
1. package hello;
2.
3. import org.springframework.beans.factory.annotation.Autowired;
4. import org.springframework.boot.CommandLineRunner;
5. import org.springframework.boot.SpringApplication;
6. import org.springframework.boot.autoconfigure.SpringBootApplication;
7.
8. @SpringBootApplication
9. public class Application implements CommandLineRunner {
10.
11.     @Autowired
12.     CustomerRepository repository;
13.
14.     public static void main(String[] args) {
15.         SpringApplication.run(Application.class);
16.     }
17.
18.     @Override
19.     public void run(String... strings) throws Exception {
20.         // save a couple of customers
21.         repository.save(new Customer("Jack", "Bauer"));
22.         repository.save(new Customer("Chloe", "O'Brian"));
23.         repository.save(new Customer("Kim", "Bauer"));
24.         repository.save(new Customer("David", "Palmer"));
25.         repository.save(new Customer("Michelle", "Dessler"));
26.
27.         // fetch all customers
28.         System.out.println("Customers found with findAll():");
29.         System.out.println("-------------------------------");
30.         for (Customer customer : repository.findAll()) {
31.             System.out.println(customer);
32.         }
33.         System.out.println();
34.
35.         // fetch an individual customer by ID
36.         Customer customer = repository.findOne(1L);
37.         System.out.println("Customer found with findOne(1L):");
38.         System.out.println("--------------------------------");
39.         System.out.println(customer);
40.         System.out.println();
41.
42.         // fetch customers by last name
43.         System.out.println("Customer found with findByLastName('Bauer'):");
44.         System.out.println("--------------------------------------------");
45.         for (Customer bauer : repository.findByLastName("Bauer")) {
46.             System.out.println(bauer);
47.         }
48.     }
49.
50. }
```

- ligne 9 : la classe implémente l'interface [CommandLineRunner] qui est une interface [Spring Boot] (ligne 4). Cette interfae n'a qu'une méthode, celle de la ligne 19 ;
- ligne 8 : **@SpringBootApplication** est une annotation regroupant plusieurs annotations [Spring Boot] :
  - **@Configuration** : indique que la classe est une classe de configuration ;
  - **@EnableAutoConfiguration** : demande à [Spring Boot] de créer lui-même un certain nombre de beans en fonction de diverses propriétés, en particulier le contenu du Classpath du projet. Parce que les bibliothèques Hibernate sont dans le Classpath, le bean [entityManagerFactory] sera implémenté avec Hibernate. Parce que la bibliothèque du SGBD H2 est dans le Classpath, le bean [dataSource] sera implémenté avec H2. Dans le bean [dataSource], on doit définir également l'utilisateur et son mot de passe. Ici Spring Boot utilisera l'administrateur par défaut de H2, **sa** sans mot de passe. Parce que la bibliothèque [spring-tx] est dans le Classpath, c'est le gestionnaire de transactions de Spring qui sera utilisé ;
  - **@EnableWebMvc** : si dans le Classpath se trouve la bibliothèque [spring-mvc]. Dans ce cas, une auto-configuration est faite pour l'application web ;

    - **@ComponentScan** : qui dit à Spring où chercher les autres beans, configurations et services. Ici ils sont cherchés par défaut dans le package contenant la classe taguée, ç-à-d le package [hello]. Ainsi les classes [Customer] et [CustomerRepository] vont-elles être trouvées. Parce que la première a l'annotation [@Entity] elle sera cataloguée comme entité à gérer par Hibernate. Parce que la seconde étend l'interface [CrudRepository] elle sera enregistrée comme bean Spring ;
- lignes 11-12 : le bean [CustomerRepository] est injecté dans le code de la classe principale ;
- ligne 15 : la méthode statique [run] de la classe [SpringApplication] du projet Spring Boot est exécutée. Son paramètre est la classe qui a une annotation [Configuration] ou [EnableAutoConfiguration]. Tout ce qui a été expliqué précédemment va alors se dérouler. Le résultat est un contexte d'application Spring, ç-à-d un ensemble de beans gérés par Spring ;

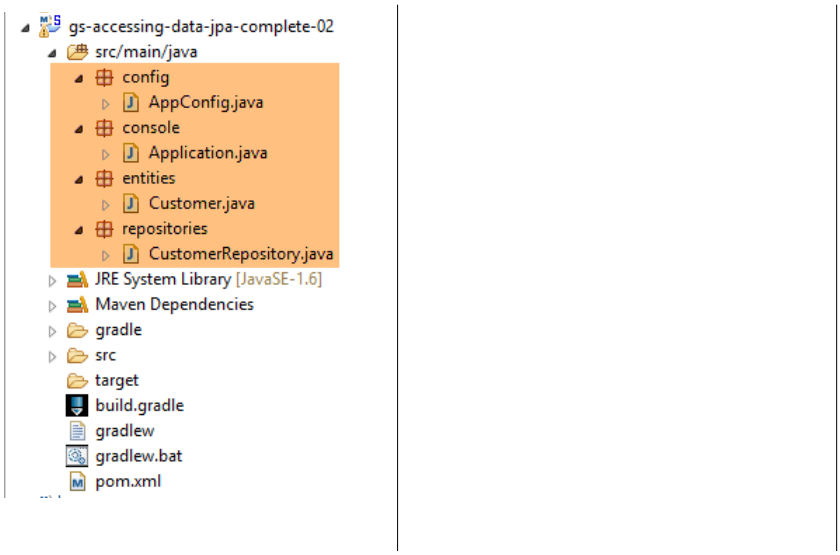
Les opérations qui suivent ne font qu'utiliser les méthodes du bean implémentant l'interface [CustomerRepository]. Les résultats console sont les suivants :

```
1.  .   ____          _            __ _ _
2.   /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
3.  ( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
4.   \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
5.    '  |____| .__|_| |_|_| |_\__, | / / / /
6.   =========|_|==============|___/=/_/_/_/
7.   :: Spring Boot ::        (v1.2.2.RELEASE)
8.
9.  2015-03-10 15:35:43.661  INFO 5784 --- [           main] hello.Application                        : Starting Application
    on Gportpers3 with PID 5784 (started by ST in C:\Users\Serge Tahé\Documents\workspace-sts-3.6.3.RELEASE\gs-accessing-
    data-jpa-complete)
10. 2015-03-10 15:35:43.708  INFO 5784 --- [           main] s.c.a.AnnotationConfigApplicationContext : Refreshing
    org.springframework.context.annotation.AnnotationConfigApplicationContext@5d11346a: startup date [Tue Mar 10 15:35:43
    CET 2015]; root of context hierarchy
11. 2015-03-10 15:35:45.230  INFO 5784 --- [           main] j.LocalContainerEntityManagerFactoryBean : Building JPA
    container EntityManagerFactory for persistence unit 'default'
12. 2015-03-10 15:35:45.254  INFO 5784 --- [           main] o.hibernate.jpa.internal.util.LogHelper  : HHH000204: Processing
    PersistenceUnitInfo [
13.  name: default
14.  ...]
15. 2015-03-10 15:35:45.331  INFO 5784 --- [           main] org.hibernate.Version                    : HHH000412: Hibernate
    Core {4.3.8.Final}
16. 2015-03-10 15:35:45.332  INFO 5784 --- [           main] org.hibernate.cfg.Environment            : HHH000206:
    hibernate.properties not found
17. 2015-03-10 15:35:45.334  INFO 5784 --- [           main] org.hibernate.cfg.Environment            : HHH000021: Bytecode
    provider name : javassist
18. 2015-03-10 15:35:45.651  INFO 5784 --- [           main] o.hibernate.annotations.common.Version   : HCANN000001:
    Hibernate Commons Annotations {4.0.5.Final}
19. 2015-03-10 15:35:45.754  INFO 5784 --- [           main] org.hibernate.dialect.Dialect            : HHH000400: Using
    dialect: org.hibernate.dialect.H2Dialect
20. 2015-03-10 15:35:45.877  INFO 5784 --- [           main] o.h.h.i.ast.ASTQueryTranslatorFactory    : HHH000397: Using
    ASTQueryTranslatorFactory
21. 2015-03-10 15:35:46.154  INFO 5784 --- [           main] org.hibernate.tool.hbm2ddl.SchemaExport  : HHH000227: Running
    hbm2ddl schema export
22. 2015-03-10 15:35:46.169  INFO 5784 --- [           main] org.hibernate.tool.hbm2ddl.SchemaExport  : HHH000230: Schema
    export complete
23. 2015-03-10 15:35:46.779  INFO 5784 --- [           main] o.s.j.e.a.AnnotationMBeanExporter        : Registering beans for
    JMX exposure on startup
24. Customers found with findAll():
25. -------------------------------
26. Customer[id=1, firstName='Jack', lastName='Bauer']
27. Customer[id=2, firstName='Chloe', lastName='O'Brian']
28. Customer[id=3, firstName='Kim', lastName='Bauer']
29. Customer[id=4, firstName='David', lastName='Palmer']
30. Customer[id=5, firstName='Michelle', lastName='Dessler']
31.
32. Customer found with findOne(1L):
33. --------------------------------
34. Customer[id=1, firstName='Jack', lastName='Bauer']
35.
36. Customer found with findByLastName('Bauer'):
37. --------------------------------------------
38. Customer[id=1, firstName='Jack', lastName='Bauer']
39. Customer[id=3, firstName='Kim', lastName='Bauer']
40. 2015-03-10 15:35:47.040  INFO 5784 --- [           main] hello.Application                        : Started Application
    in 3.623 seconds (JVM running for 4.324)
41. 2015-03-10 15:35:47.042  INFO 5784 --- [       Thread-1] s.c.a.AnnotationConfigApplicationContext : Closing
    org.springframework.context.annotation.AnnotationConfigApplicationContext@5d11346a: startup date [Tue Mar 10 15:35:43
    CET 2015]; root of context hierarchy
42. 2015-03-10 15:35:47.044  INFO 5784 --- [       Thread-1] o.s.j.e.a.AnnotationMBeanExporter        : Unregistering JMX-
    exposed beans on shutdown
43. 2015-03-10 15:35:47.046  INFO 5784 --- [       Thread-1] j.LocalContainerEntityManagerFactoryBean : Closing JPA
    EntityManagerFactory for persistence unit 'default'
44. 2015-03-10 15:35:47.047  INFO 5784 --- [       Thread-1] org.hibernate.tool.hbm2ddl.SchemaExport  : HHH000227: Running
    hbm2ddl schema export
45. 2015-03-10 15:35:47.051  INFO 5784 --- [       Thread-1] org.hibernate.tool.hbm2ddl.SchemaExport  : HHH000230: Schema
    export complete
```

- lignes 1-8 : le logo du projet Spring Boot ;
- ligne 9 : la classe [hello.Application] est exécutée ;
- ligne 10 : [AnnotationConfigApplicationContext] est une classe implémentant l'interface [ApplicationContext] de Spring. C'est un conteneur de beans ;
- ligne 11 : le bean [entityManagerFactory] est implémenté avec la classe [LocalContainerEntityManagerFactory], une classe de Spring ;
- ligne 12 : on voit apparaître [Hibernate]. C'est cette implémentation JPA qui a été choisie ;
- ligne 19 : un dialecte Hibernate est la variante SQL à utiliser avec le SGBD. Ici le dialecte [H2Dialect] montre qu'Hibernate va travailler avec le SGBD H2 ;
- lignes 21-22 : la base de données est créée. La table [CUSTOMER] est créée. Cela signifie qu'Hibernate a été configuré pour générer les tables à partir des définitions JPA, ici la définition JPA de la classe [Customer] ;
- lignes 26-30 : résultat de la méthode [findAll] de l'interface ;
- ligne 34 : résultat de la méthode [findOne] de l'interface ;
- lignes 38-39 : résultats de la méthode [findByLastName] ;
- lignes 41 et suivantes : logs de la fermeture du contexte Spring.

11.2.5 Configuration manuelle du projet Spring Data

Nous dupliquons le projet précédent dans le projet [gs-accessing-data-jpa-02] :



Dans ce nouveau projet, nous n'allons pas nous reposer sur la configuration automatique faite par Spring Boot. Nous allons la faire manuellement. Cela peut être utile si les configurations par défaut ne nous conviennent pas.

Tout d'abord, nous allons expliciter les dépendances nécessaires dans le fichier [pom.xml] :

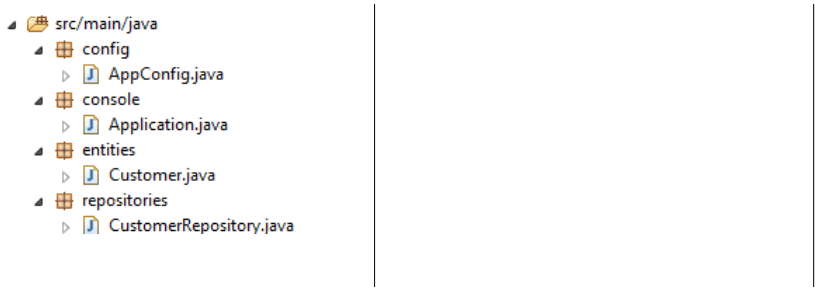
```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <project xmlns="http://maven.apache.org/POM/4.0.0" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/maven-v4_0_0.xsd"
3.   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
4.   <modelVersion>4.0.0</modelVersion>
5.
6.   <groupId>org.springframework</groupId>
7.   <artifactId>gs-accessing-data-jpa-02</artifactId>
8.   <version>0.1.0</version>
9.
10.    <parent>
11.        <groupId>org.springframework.boot</groupId>
12.        <artifactId>spring-boot-starter-parent</artifactId>
13.        <version>1.2.7.RELEASE</version>
14.    </parent>
15.
16.    <dependencies>
17.        <!-- Spring Data -->
18.        <dependency>
19.            <groupId>org.springframework.data</groupId>
20.            <artifactId>spring-data-jpa</artifactId>
21.        </dependency>
22.        <!-- Hibernate -->
23.        <dependency>
24.            <groupId>org.hibernate</groupId>
```

```
25.             <artifactId>hibernate-entitymanager</artifactId>
26.         </dependency>
27.         <!-- H2 Database -->
28.         <dependency>
29.             <groupId>com.h2database</groupId>
30.             <artifactId>h2</artifactId>
31.         </dependency>
32.         <!-- Tomcat JDBC -->
33.         <dependency>
34.             <groupId>org.apache.tomcat</groupId>
35.             <artifactId>tomcat-jdbc</artifactId>
36.         </dependency>
37.     </dependencies>
38.
39.     <properties>
40.         <!-- use UTF-8 for everything -->
41.         <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
42.         <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
43.     </properties>
44.
45.     <build>
46.         <plugins>
47.             <plugin>
48.                 <groupId>org.springframework.boot</groupId>
49.                 <artifactId>spring-boot-maven-plugin</artifactId>
50.             </plugin>
51.         </plugins>
52.     </build>
53.
54.     <repositories>
55.         <repository>
56.             <id>spring-releases</id>
57.             <name>Spring Releases</name>
58.             <url>https://repo.spring.io/libs-release</url>
59.         </repository>
60.         <repository>
61.             <id>org.jboss.repository.releases</id>
62.             <name>JBoss Maven Release Repository</name>
63.             <url>https://repository.jboss.org/nexus/content/repositories/releases</url>
64.         </repository>
65.     </repositories>
66.
67.     <pluginRepositories>
68.         <pluginRepository>
69.             <id>spring-releases</id>
70.             <name>Spring Releases</name>
71.             <url>https://repo.spring.io/libs-release</url>
72.         </pluginRepository>
73.     </pluginRepositories>
74.
75. </project>
```

- lignes 10-14 : le projet Maven parent dont nous allons utiliser les bibliothèques qu'il définit ;
- lignes 18-21 : Spring Data utilisé pour accéder à la base de données ;
- lignes 23-26 : l'implémentation Hibernate de la spécification JPA ;
- lignes 28-31 : le SGBD H2 ;
- lignes 33-36 : les bases de données sont souvent utilisées avec des pools de connexions ouvertes qui évitent les ouvertures / fermetures de connexion à répétition. Ici, l'implémentation utilisée est celle de [tomcat-jdbc] ;

Dans le nouveau projet, l'entité [Customer] et l'interface [CustomerRepository] ne changent pas. On va changer la classe [Application] qui va être scindée en deux classes :

- [Config] qui sera la classe de configuration :
- [Main] qui sera la classe exécutable ;



La classe exécutable [Application] est désormais la suivante :

```
1. package console;
2.
3. import org.springframework.context.annotation.AnnotationConfigApplicationContext;
4.
5. import repositories.CustomerRepository;
6. import config.AppConfig;
7. import entities.Customer;
8.
9. public class Application {
10.    public static void main(String[] args) {
11.        // instanciation contexte Spring
12.        AnnotationConfigApplicationContext context = new AnnotationConfigApplicationContext(AppConfig.class);
13.        CustomerRepository repository = context.getBean(CustomerRepository.class);
14.
15.        // save a couple of customers
16.        repository.save(new Customer("Jack", "Bauer"));
17.        repository.save(new Customer("Chloe", "O'Brian"));
18.        repository.save(new Customer("Kim", "Bauer"));
19.        repository.save(new Customer("David", "Palmer"));
20.        repository.save(new Customer("Michelle", "Dessler"));
21.
22.        ...
23.
24.        // fermeture contexte
25.        context.close();
26.    }
27. }
```

- ligne 9 : la classe [Application] n'a plus d'annotations de configuration ;
- lignes 3-7 : on notera qu'il n'y a plus d'imports de packages [Spring Boot] ;
- ligne 12 : on instancie les beans Spring. On obtient le contexte de Spring qui contient la référence des beans ainsi créés ;
- ligne 13 : on demande une référence sur le bean de type [CustomerRepository] ;

La classe [Config] qui configure le projet est la suivante :

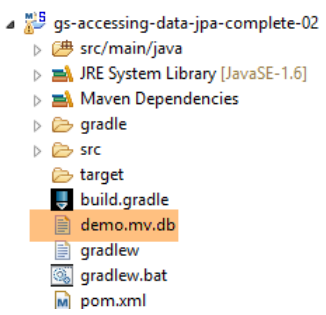
```
1. package config;
2.
3. import javax.persistence.EntityManagerFactory;
4.
5. import org.apache.tomcat.jdbc.pool.DataSource;
6. import org.springframework.context.annotation.Bean;
7. import org.springframework.context.annotation.Configuration;
8. import org.springframework.data.jpa.repository.config.EnableJpaRepositories;
9. import org.springframework.orm.jpa.JpaTransactionManager;
10.  import org.springframework.orm.jpa.JpaVendorAdapter;
11.  import org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean;
12.  import org.springframework.orm.jpa.vendor.Database;
13.  import org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter;
14.  import org.springframework.transaction.PlatformTransactionManager;
15.  import org.springframework.transaction.annotation.EnableTransactionManagement;
16.
17.  //@EnableTransactionManagement
18.  @EnableJpaRepositories(basePackages = { "repositories" })
19.  @Configuration
20.  // @ComponentScan(basePackages={"package1","package2"})
21.  public class AppConfig {
22.
23.    // la base de données H2
24.    @Bean
25.    public DataSource dataSource() {
26.        // source de données TomcatJdbc
27.        DataSource dataSource = new DataSource();
28.        // configuration accès JDBC
29.        dataSource.setDriverClassName("org.h2.Driver");
30.        dataSource.setUrl("jdbc:h2:./demo");
31.        dataSource.setUsername("sa");
32.        dataSource.setPassword("");
33.        // une connexion ouverte initialement
34.        dataSource.setInitialSize(1);
35.        // résultat
36.        return dataSource;
37.    }
38.
39.    // le provider JPA
40.    @Bean
41.    public JpaVendorAdapter jpaVendorAdapter() {
42.        HibernateJpaVendorAdapter hibernateJpaVendorAdapter = new HibernateJpaVendorAdapter();
43.        hibernateJpaVendorAdapter.setShowSql(false);
44.        hibernateJpaVendorAdapter.setGenerateDdl(true);
45.        hibernateJpaVendorAdapter.setDatabase(Database.H2);
46.        return hibernateJpaVendorAdapter;
47.    }
```

```
48.
49.    // EntityManagerFactory
50.    @Bean
51.    public EntityManagerFactory entityManagerFactory(JpaVendorAdapter jpaVendorAdapter, DataSource dataSource) {
52.        LocalContainerEntityManagerFactoryBean factory = new LocalContainerEntityManagerFactoryBean();
53.        factory.setJpaVendorAdapter(jpaVendorAdapter);
54.        factory.setPackagesToScan("entities");
55.        factory.setDataSource(dataSource);
56.        factory.afterPropertiesSet();
57.        return factory.getObject();
58.    }
59.
60.    // Transaction manager
61.    @Bean
62.    public PlatformTransactionManager transactionManager(EntityManagerFactory entityManagerFactory) {
63.        JpaTransactionManager txManager = new JpaTransactionManager();
64.        txManager.setEntityManagerFactory(entityManagerFactory);
65.        return txManager;
66.    }
67.
68. }
```

- ligne 17 : l'annotation [@EnableTransactionManagement] indique que les méthodes des interfaces [CrudRepository] doivent se dérouler à l'intérieur d'une transaction. Elle a été mise en commentaires car c'est le cas par défaut ;
- ligne 18 : l'annotation [@EnableJpaRepositories] permet de désigner les dossiers où se trouvent les interfaces Spring Data [CrudRepository]. Ces interfaces vont devenir des composants Spring et être disponibles dans son contexte ;
- ligne 19 : l'annotation [@Configuration] fait de la classe [Config] une classe de configuration Spring ;
- ligne 20 : l'annotation [@ComponentScan] permet de lister les dossiers où les composants Spring doivent être recherchés. Les composants Spring sont des classes taguées avec des annotations Spring telles que **@Service**, **@Component**, **@Controller**, ... Ici, il n'y en a pas d'autres que ceux qui sont définis au sein de la classe [AppConfig], aussi l'annotation a-t-elle été mise en commentaires ;
- lignes 24-37 : définissent la source de données, la base de données H2. C'est l'annotation @Bean de la ligne 25 qui fait de l'objet créé par cette méthode un composant géré par Spring. Le nom de la méthode peut être ici quelconque. Cependant elle doit être appelée [**dataSource**] si l'EntityManagerFactory de la ligne 51 est absent et défini par autoconfiguration ;
- ligne 30 : la base de données s'appellera [demo] et sera générée dans le dossier du projet ;
- lignes 40-47 : définissent l'implémentation JPA utilisée, ici une implémentation Hibernate. Le nom de la méthode peut être ici quelconque ;
- ligne 43 : pas de logs SQL ;
- ligne 44 : la base de données sera créée si elle n'existe pas ;
- lignes 50-58 : définissent l'EntityManagerFactory qui va gérer la persistance JPA. La méthode doit s'appeler obligatoirement [**entityManagerFactory**] ;
- ligne 51 : la méthode reçoit deux paramètres ayant le type des deux beans définis précédemment. Ceux-ci seront alors construits puis injectés par Spring comme paramètres de la méthode ;
- ligne 53 : fixe l'implémentation JPA utilisée ;
- ligne 54 : fixe les dossiers où trouver les entités JPA ;
- ligne 55 : fixe la source de données à gérer ;
- lignes 61-66 : le gestionnaire de transactions. La méthode doit s'appeler obligatoirement [**transactionManager**]. Elle reçoit pour paramètre le bean des lignes 51-58 ;
- ligne 64 : le gestionnaire de transactions est associé à l'EntityManagerFactory ;

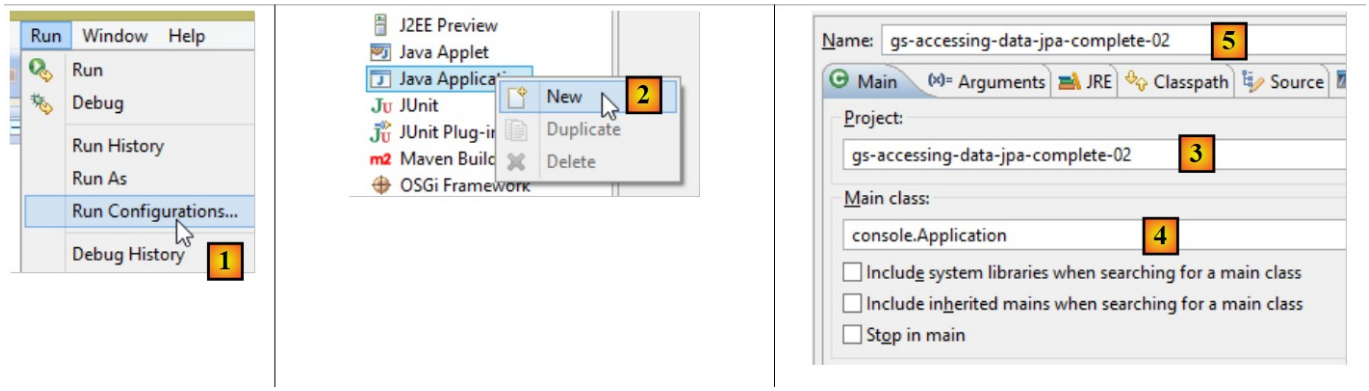
Les méthodes précédentes peuvent être définies dans un ordre quelconque.

L'exécution du projet donne les mêmes résultats. Un nouveau fichier apparaît dans le dossier du projet, celui de la base de données H2 :

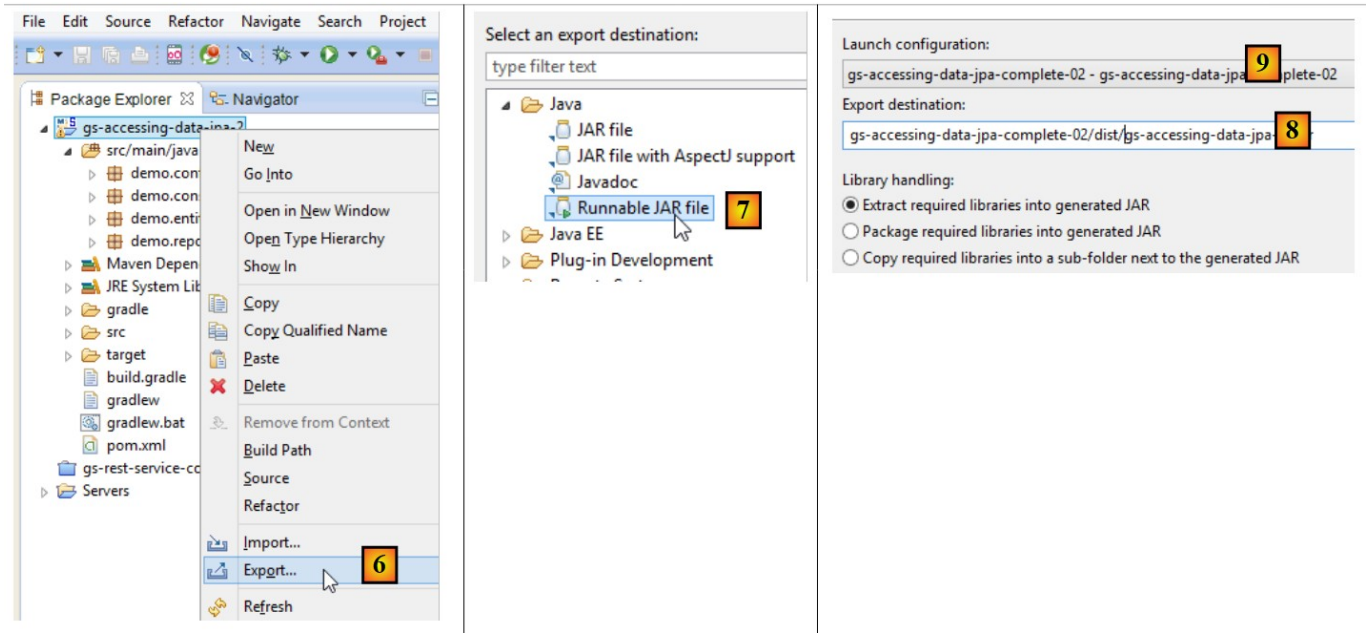


11.2.6 Création d'une archive exécutable

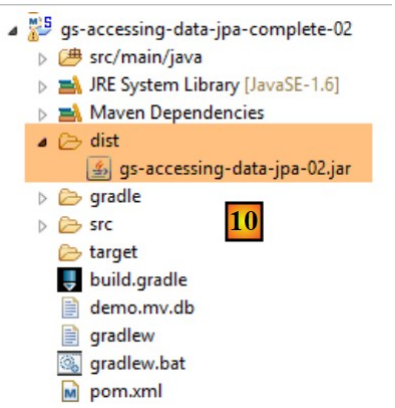
Pour créer une archive exécutable du projet, on peut procéder ainsi :



- en [1] : on crée une configuration d'exécution ;
- en [2] : de type [Java Application]
- en [3] : désigne le projet à exécuter (utiliser le bouton *Browse*) ;
- en [4] : désigne la classe à exécuter ;
- en [5] : le nom de la configuration d'exécution – peut être quelconque ;



- en [6] : on exporte le projet ;
- en [7] : sous la forme d'une archive JAR exécutable ;
- en [8] : indique le chemin et le nom du fichier exécutable à créer ;
- en [9] : le nom de la configuration d'exécution créée en [5] ;



- en [10], l'archive créée ;

Ceci fait, on ouvre une console dans le dossier contenant l'archive exécutable :

```
.....\dist>dir
12/06/2014  09:11        15 104 869 gs-accessing-data-jpa-02.jar
```

L'archive est exécutée de la façon suivante :

```
1. .....\dist>java -jar gs-accessing-data-jpa-02.jar
```

Les résultats obtenus dans la console sont les suivants :

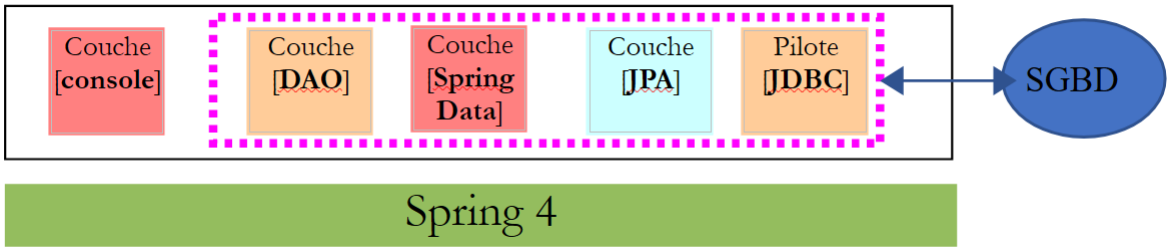
```
1. SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
2. SLF4J: Defaulting to no-operation (NOP) logger implementation
3. SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
4. mars 10, 2015 5:27:20 PM org.hibernate.jpa.internal.util.LogHelper logPersistenceUnitInformation
5. INFO: HHH000204: Processing PersistenceUnitInfo [
6.   name: default
7.   ...]
8. mars 10, 2015 5:27:20 PM org.hibernate.Version logVersion
9. INFO: HHH000412: Hibernate Core {4.3.8.Final}
10.  mars 10, 2015 5:27:20 PM org.hibernate.cfg.Environment <clinit>
11.  INFO: HHH000206: hibernate.properties not found
12.  mars 10, 2015 5:27:20 PM org.hibernate.cfg.Environment buildBytecodeProvider
13.  INFO: HHH000021: Bytecode provider name : javassist
14.  mars 10, 2015 5:27:22 PM org.hibernate.annotations.common.reflection.java.JavaReflectionManager <clinit>
15.  INFO: HCANN000001: Hibernate Commons Annotations {4.0.5.Final}
16.  mars 10, 2015 5:27:22 PM org.hibernate.dialect.Dialect <init>
17.  INFO: HHH000400: Using dialect: org.hibernate.dialect.H2Dialect
18.  mars 10, 2015 5:27:22 PM org.hibernate.hql.internal.ast.ASTQueryTranslatorFactory <init>
19.  INFO: HHH000397: Using ASTQueryTranslatorFactory
20.  mars 10, 2015 5:27:22 PM org.hibernate.tool.hbm2ddl.SchemaUpdate execute
21.  INFO: HHH000228: Running hbm2ddl schema update
22.  mars 10, 2015 5:27:22 PM org.hibernate.tool.hbm2ddl.SchemaUpdate execute
23.  INFO: HHH000102: Fetching database metadata
24.  mars 10, 2015 5:27:22 PM org.hibernate.tool.hbm2ddl.SchemaUpdate execute
25.  INFO: HHH000396: Updating schema
26.  mars 10, 2015 5:27:22 PM org.hibernate.tool.hbm2ddl.DatabaseMetadata getTableMetadata
27.  INFO: HHH000262: Table not found: Customer
28.  mars 10, 2015 5:27:22 PM org.hibernate.tool.hbm2ddl.DatabaseMetadata getTableMetadata
29.  INFO: HHH000262: Table not found: Customer
30.  mars 10, 2015 5:27:22 PM org.hibernate.tool.hbm2ddl.DatabaseMetadata getTableMetadata
31.  INFO: HHH000262: Table not found: Customer
32.  mars 10, 2015 5:27:22 PM org.hibernate.tool.hbm2ddl.SchemaUpdate execute
33.  INFO: HHH000232: Schema update complete
34.  Customers found with findAll():
35.  -------------------------------
36.  Customer[id=1, firstName='Jack', lastName='Bauer']
37.  Customer[id=2, firstName='Chloe', lastName='O'Brian']
38.  Customer[id=3, firstName='Kim', lastName='Bauer']
39.  Customer[id=4, firstName='David', lastName='Palmer']
40.  Customer[id=5, firstName='Michelle', lastName='Dessler']
41.
42.  Customer found with findOne(1L):
43.  --------------------------------
44.  Customer[id=1, firstName='Jack', lastName='Bauer']
45.
46.  Customer found with findByLastName('Bauer'):
47.  --------------------------------------------
48.  Customer[id=1, firstName='Jack', lastName='Bauer']
```

```
49.  Customer[id=3, firstName='Kim', lastName='Bauer']
```

11.3 Exemple 2

11.3.1 Introduction

Nous allons reprendre l'exemple de la table de produits que nous avons utilisée pour introduire l'API JDBC et créer l'architecture suivante :



La base de données [dbintrospringjpa] a deux tables [PRODUITS] et [CATEGORIES]. La table [CATEGORIES] est la suivante :

#	Nom	Type	Interclassement	Attributs	Null	Défaut	Extra
1	ID	bigint(20)			Non	*Aucune*	AUTO_INCREMENT
2	VERSION	int(11)			Non	0	
3	NOM	varchar(20)	utf8_general_ci		Non	*Aucune*	

- [ID] : clé primaire en mode AUTO_INCREMENT ;
- [VERSION] : n° de version de l'enregistrement ;
- [NOM] : nom de la catégorie - unique ;

La table [PRODUITS] est la suivante :

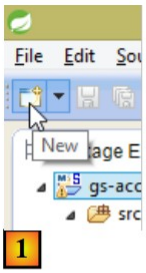
#	Nom	Type	Interclassement	Attributs	Null	Défaut	Extra
1	ID	int(11)			Non	*Aucune*	AUTO_INCREMENT
2	VERSION	int(11)			Non	0	
3	NOM	varchar(20)	utf8_general_ci		Non	*Aucune*	
4	CATEGORIE_ID	bigint(11)			Oui	*NULL*	
5	PRIX	decimal(10,3)			Non	*Aucune*	
6	DESCRIPTION	varchar(100)	utf8_general_ci		Non	*Aucune*	

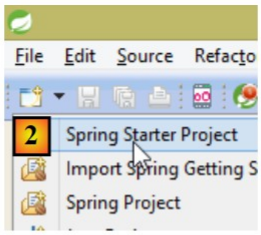
- [ID] : clé primaire en mode AUTO_INCREMENT ;
- [VERSION] : n° de version de l'enregistrement ;
- [NOM] : nom d'un produit - unique ;
- [ID_CATEGORIE] : n° de sa catégorie - clé étrangère sur le champ [CATEGORIES.ID] ;
- [PRIX] : son prix ;
- [DESCRIPTION] : une description du produit ;

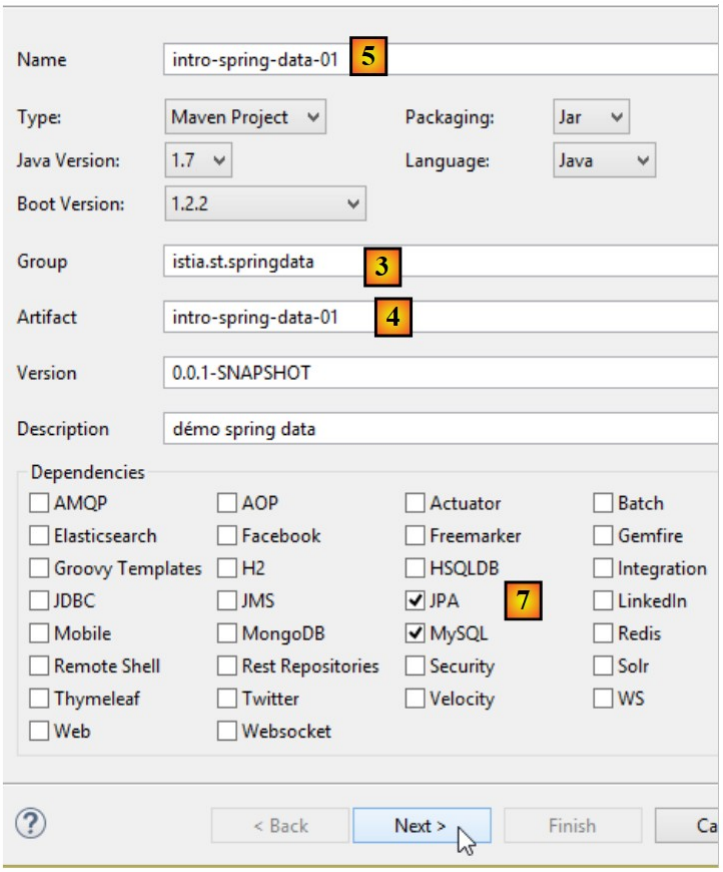
Travail à faire : créez la base de données [dbintrospringdata] avec le script SQL [dbintrospringdata.sql] du support :

11.3.2 Création du projet Maven

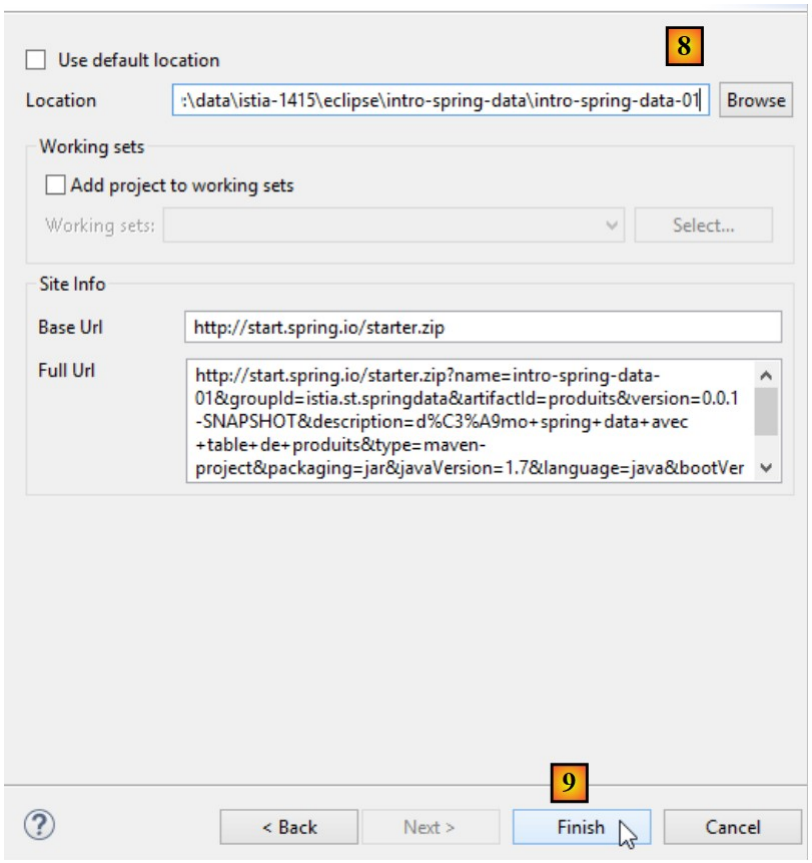
Pour créer un squelette de projet Spring Data, on peut procéder de la façon suivante :



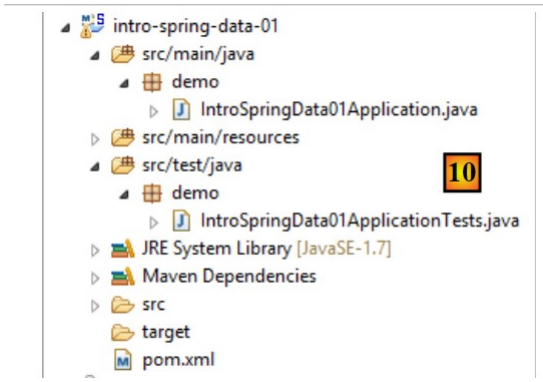




- en [1], on crée un nouveau projet ;
- en [2] : de type [Spring Starter Project] ;
- le projet généré sera un projet Maven. En [3], on indique le nom du groupe du projet ;
- en [4] : on indique le nom de l'artifact (un jar ici) qui sera créé par construction du projet ;
- en [5] : le nom Eclipse du projet – peut être quelconque (n'a pas à être identique à [4]) ;
- en [7] : on indique qu'on va créer un projet ayant une couche [JPA] avec le SGBD MySQL. Les dépendances nécessaires à un tel projet vont alors être incluses dans le fichier [pom.xml] ;



- en [8], donner le nom du dossier du projet ;
- en [9], terminer l'assistant ;



- en [10] : le projet créé ;

Le fichier [pom.xml] intègre les dépendances nécessaires à un projet JPA :

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3.   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
4.   <modelVersion>4.0.0</modelVersion>
5.
6.   <groupId>istia.st.springdata</groupId>
7.   <artifactId>intro-spring-data-01</artifactId>
8.   <version>0.0.1-SNAPSHOT</version>
9.   <packaging>jar</packaging>
10.
11.    <name>intro-spring-data-01</name>
12.    <description>démo spring data avec table de produits</description>
13.
```

```
14.    <parent>
15.        <groupId>org.springframework.boot</groupId>
16.        <artifactId>spring-boot-starter-parent</artifactId>
17.        <version>1.2.2.RELEASE</version>
18.        <relativePath/> <!-- lookup parent from repository -->
19.    </parent>
20.
21.    <properties>
22.        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
23.        <start-class>demo.IntroSpringData01Application</start-class>
24.        <java.version>1.7</java.version>
25.    </properties>
26.
27.    <dependencies>
28.        <dependency>
29.            <groupId>org.springframework.boot</groupId>
30.            <artifactId>spring-boot-starter-data-jpa</artifactId>
31.        </dependency>
32.        <dependency>
33.            <groupId>mysql</groupId>
34.            <artifactId>mysql-connector-java</artifactId>
35.            <scope>runtime</scope>
36.        </dependency>
37.        <dependency>
38.            <groupId>org.springframework.boot</groupId>
39.            <artifactId>spring-boot-starter-test</artifactId>
40.            <scope>test</scope>
41.        </dependency>
42.    </dependencies>
43.
44.    <build>
45.        <plugins>
46.            <plugin>
47.                <groupId>org.springframework.boot</groupId>
48.                <artifactId>spring-boot-maven-plugin</artifactId>
49.            </plugin>
50.        </plugins>
51.    </build>
52.
53.  </project>
```

- lignes 14-19 : le projet Maven parent - définit un grand nombre de bibliothèques avec leurs versions - on utilise ces bibliothèques comme dépendances Maven sans préciser leur version ;
- lignes 28-31 : la dépendance nécessaire à JPA – va inclure [Spring Data] ;
- lignes 32-36 : la dépendance sur le pilote JDBC de MySQL ;
- lignes 37-41 : les dépendances nécessaires aux tests JUnit intégrés avec Spring ;

La classe exécutable [Application] ne fait rien mais est préconfigurée :

```
1. package demo;
2.
3. import org.springframework.boot.SpringApplication;
4. import org.springframework.boot.autoconfigure.SpringBootApplication;
5.
6. @SpringBootApplication
7. public class IntroSpringData01Application {
8.
9.     public static void main(String[] args) {
10.         SpringApplication.run(IntroSpringData01Application.class, args);
11.     }
12. }
```

- l'annotation [@SpringBootApplication] fait de la classe une classe d'auto-configuration du projet ;

La classe de tests [ApplicationTests] ne fait rien mais est préconfigurée :

```
1. package demo;
2.
3. import org.junit.Test;
4. import org.junit.runner.RunWith;
5. import org.springframework.boot.test.SpringApplicationConfiguration;
6. import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
7.
8. @RunWith(SpringJUnit4ClassRunner.class)
9. @SpringApplicationConfiguration(classes = IntroSpringData01Application.class)
10.  public class IntroSpringData01ApplicationTests {
11.
12.    @Test
13.    public void contextLoads() {
14.    }
15.
```

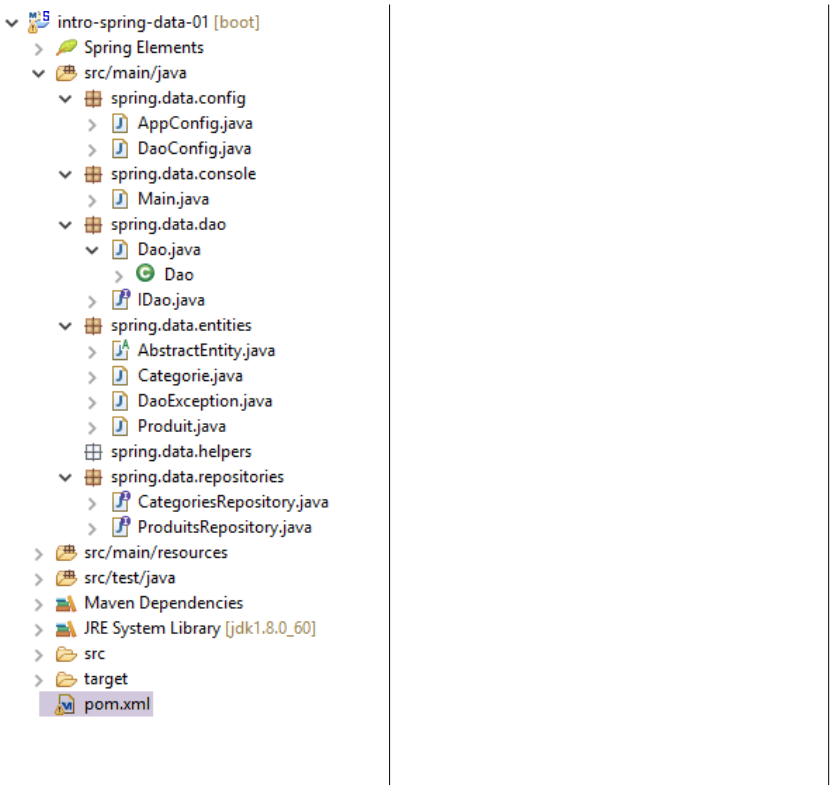
```
16. }
```

- ligne 9 : l'annotation [@SpringApplicationConfiguration] permet d'exploiter le fichier de configuration [Application]. La classe de test bénéficiera ainsi de tous les beans qui seront définis par ce fichier ;
- ligne 8 : l'annotation [@RunWith] permet l'intégration de Spring avec JUnit : la classe va pouvoir être exécutée comme un test JUnit. [@RunWith] est une annotation JUnit (ligne 4) alors que la classe [SpringJUnit4ClassRunner] est une classe Spring (ligne 6) ;

Maintenant que nous avons un squelette d'application JPA, nous pouvons le compléter pour écrire le projet de la couche de persistance associée à la base de données des produits.

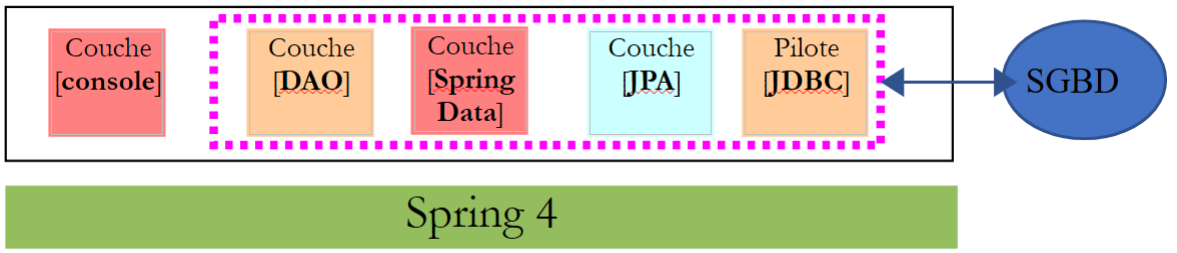
11.3.3 Le projet Eclipse

Nous faisons évoluer le projet précédent de la façon suivante :



- [AppConfig.java] : la classe de configuration du projet Spring ;
- [Main.java] : la classe exécutable du projet ;
- [IDao.java] : l'interface de la couche [DAO] ;
- [Dao.java] : la classe d'implémentation de la couche [DAO] ;
- [AbstractEntity.java] : la classe parent des classes [Produit] et [Categorie] ;
- [Produit.java] : classe associée à une ligne de la table [PRODUITS] de la base de données ;
- [Categorie.java] : classe associée à une ligne de la table [CATEGORIES] de la base de données ;
- [ProduitsRepository] : l'interface Spring Data d'accès à la table [PRODUITS] ;
- [CategoriesRepository] : l'interface Spring Data d'accès à la table [CATEGORIES] ;
- [pom.xml] : le fichier de configuration du projet Maven ;

Ce projet implémente l'architecture suivante :



La couche [DAO] ne voit que la couche implémentée par [Spring Data].

11.3.4 Configuration Maven

Le fichier [pom.xml] du projet Maven est le suivant :

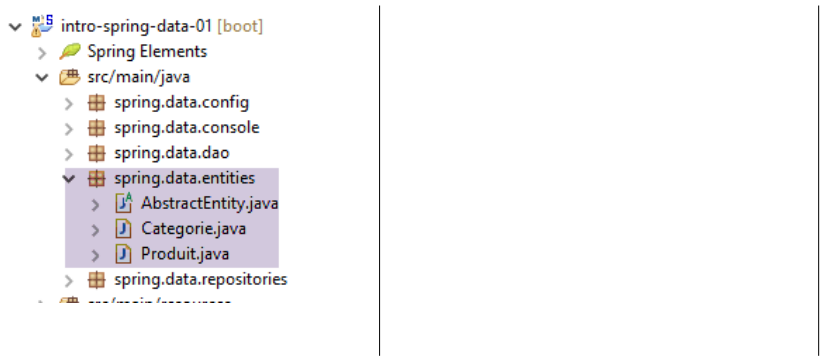
```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>istia.st.springdata</groupId>
  <artifactId>intro-spring-data-01</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

   <name>intro-spring-data-01</name>
   <description>démo spring data avec table de produits</description>

   <parent>
       <groupId>org.springframework.boot</groupId>
       <artifactId>spring-boot-starter-parent</artifactId>
       <version>1.2.7.RELEASE</version>
   </parent>

   <dependencies>
       <!-- Spring Data -->
       <dependency>
           <groupId>org.springframework.data</groupId>
           <artifactId>spring-data-jpa</artifactId>
       </dependency>
       <!-- Hibernate -->
       <dependency>
           <groupId>org.hibernate</groupId>
           <artifactId>hibernate-entitymanager</artifactId>
       </dependency>
       <!-- MySQL Database -->
       <dependency>
           <groupId>mysql</groupId>
           <artifactId>mysql-connector-java</artifactId>
       </dependency>
       <!-- Tomcat JDBC -->
       <dependency>
           <groupId>org.apache.tomcat</groupId>
           <artifactId>tomcat-jdbc</artifactId>
       </dependency>
       <!-- bibliothèque jSON -->
       <dependency>
           <groupId>com.fasterxml.jackson.core</groupId>
           <artifactId>jackson-core</artifactId>
       </dependency>
       <dependency>
           <groupId>com.fasterxml.jackson.core</groupId>
           <artifactId>jackson-databind</artifactId>
       </dependency>
       <!-- Google Guava -->
       <dependency>
           <groupId>com.google.guava</groupId>
           <artifactId>guava</artifactId>
           <version>16.0.1</version>
       </dependency>
       <!-- Spring Boot Test -->
       <dependency>
           <groupId>org.springframework.boot</groupId>
           <artifactId>spring-boot-starter-test</artifactId>
           <scope>test</scope>
       </dependency>
       <!-- Spring Boot -->
       <dependency>
           <groupId>org.springframework.boot</groupId>
```

```
65.           <artifactId>spring-boot</artifactId>
66.           <scope>test</scope>
67.       </dependency>
68.       <!-- bibliothèque de logs -->
69.       <dependency>
70.           <groupId>org.springframework.boot</groupId>
71.           <artifactId>spring-boot-starter-logging</artifactId>
72.       </dependency>
73.   </dependencies>
74.
75.   <properties>
76.       <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
77.       <java.version>1.8</java.version>
78.   </properties>
79.
80.   <build>
81.       <plugins>
82.           <plugin>
83.               <groupId>org.apache.maven.plugins</groupId>
84.               <artifactId>maven-surefire-plugin</artifactId>
85.               <version>2.18.1</version>
86.           </plugin>
87.       </plugins>
88.   </build>
89.
90. </project>
```

Cette configuration est celle utilisée et expliquée au paragraphe 11.2.5, page 174. Nous y ajoutons les bibliothèques suivantes :

- lignes 42-49 : une bibliothèque jSON utilisée par la méthode [toString] de la classe [Produit] ;
- lignes 51-55 : la bibliothèque [Google Guava] qui amène des méthodes utilitaires pour gérer des collections d'éléments. Elle sera utilisée par la classe [Dao] qui implémente la couche [DAO] ;
- lignes 56-67 : les bibliothèques nécessaires aux tests JUnit ;
- lignes 69-72 : une bibliothèque de logs ;
- lignes 81-86 : les plugins Maven nécessaires au projet ;

11.3.5 Les entités de la couche [JPA]



11.3.5.1 La classe [AbstractEntity]

La classe [AbstractEntity] est la suivante :

```
1. package spring.data.entities;
2.
3. import javax.persistence.Column;
4. import javax.persistence.GeneratedValue;
5. import javax.persistence.GenerationType;
6. import javax.persistence.Id;
7. import javax.persistence.MappedSuperclass;
8. import javax.persistence.Version;
9.
10.  import com.fasterxml.jackson.core.JsonProcessingException;
11.  import com.fasterxml.jackson.databind.ObjectMapper;
12.
13.  @MappedSuperclass
14.  public abstract class AbstractEntity {
15.    // propriétés
16.    @Id
17.    @GeneratedValue(strategy = GenerationType.IDENTITY)
```

```
18.    @Column(name = "ID")
19.    protected Long id;
20.    @Version
21.    @Column(name = "VERSION")
22.    protected Long version;
23.
24.    // constructeurs
25.    public AbstractEntity() {
26.
27.    }
28.
29.    public AbstractEntity(Long id, Long version) {
30.        this.id = id;
31.        this.version = version;
32.    }
33.
34.    // redéfinition [equals] et [hashcode]
35.    @Override
36.    public int hashCode() {
37.        return (id != null ? id.hashCode() : 0);
38.    }
39.
40.    @Override
41.    public boolean equals(Object entity) {
42.        if (!(entity instanceof AbstractEntity)) {
43.            return false;
44.        }
45.        String class1 = this.getClass().getName();
46.        String class2 = entity.getClass().getName();
47.        if (!class2.equals(class1)) {
48.            return false;
49.        }
50.        AbstractEntity other = (AbstractEntity) entity;
51.        return id != null && this.id.longValue() == other.id.longValue();
52.    }
53.
54.    // signature jSON
55.    public String toString() {
56.        ObjectMapper mapper = new ObjectMapper();
57.        try {
58.            return mapper.writeValueAsString(this);
59.        } catch (JsonProcessingException e) {
60.            e.printStackTrace();
61.            return null;
62.        }
63.    }
64.
65.    // getters et setters
66.  ....
67.  }
68.
```

Cette classe a pour objectif de fournir une classe mère aux entités JPA en encapsulant à un unique endroit les propriétés [id, version] (lignes 19, 22) communes aux deux entités [Produit] et [Categorie] liées à la base de données. Ces propriétés sont liées aux colonnes [ID, VERSION] des tables (lignes 18, 21).

- ligne 13 : l'annotation [@MappedSuperclass] indique que la classe est une classe parent d'entités JPA ;
- ligne 16 : l'annotation [@Id] indique que le champ [id] (il pourrait avoir un autre nom) est associé à la clé primaire d'une table ;
- ligne 17 : l'annotation [@GeneratedValue(strategy=GenerationType.***IDENTITY***)] fixe le mode de génération des clés primaires. Le mode [GenerationType.***IDENTITY***] va utiliser avec MySQL le mode [AUTO_INCREMENT]. Avec un autre SGBD, ce mode utiliserait une autre méthode. L'intérêt est que le développeur n'a pas à s'en préoccuper et que son code reste valide quelque soit le SGBD utilisé ;
- ligne 18 : l'annotation [@Column] indique la colonne associée au champ. Lorsque cette annotation n'est pas présente, JPA assume que la colonne porte le même nom que le champ. C'est le cas ici. On n'aurait donc pu ne pas mettre cette annotation ;
- ligne 20 : l'annotation [@Version] indique que le champ [version] est associé à une colonne de versioning. L'implémentation JPA va incrémenter ce n° de version à chaque fois que l'entité sera modifiée. Ce n° sert à empêcher la mise à jour simultanée de l'entité par deux utilisateur différents : deux utilisateurs U1 et U2 lisent l'entité E avec un n° de version égal à V1. U1 modifie E et persiste cette modification en base : le n° de version passe alors à V1+1. U2 modifie E à son tour et persiste cette modification en base : il recevra une exception car il possède une version (V1) différente de celle en base (V1+1) ;
- lignes 35-52 : redéfinition des méthodes [hashCode] et [equals]. Par défaut, [obj1.equals(obj2)] vaut true si [obj1==obj2], ç-à-d si ob1 et obj2 sont deux pointeurs égaux. Si on veut comparer les objets pointés plutôt que les pointeurs eux-mêmes, il faut redéfinir la méthode [equals] et la méthode [hashCode]. Celle-ci doit rendre la même valeur pour deux objets que la méthode [equals] dit égaux ;
- lignes 42-51 : deux objets de type [AbstractEntity] ou dérivés seront dits égaux si leurs clés primaires [id] sont égales ;

- lignes 35-38 : la méthode [hashCode] rend bien la même valeur pour deux objets [AbstractEntity] identiques et donc ayant la même clé primaire [id] ;
- lignes 55-63 : la méthode [toString] rend la chaîne jSon de l'objet [this]. Si cet objet désigne une classe fille, cette méthode rendra alors la chaîne jSON de la classe fille. Cela nous dispense de créer une méthode [toString] dans les classes filles ;

11.3.5.2 L'entité JPA [Produit]

La classe [Produit] est une entité JPA associée à une ligne de la table [PRODUITS] :

#	Nom	Type	Interclassement	Attributs	Null	Défaut	Extra
1	ID	int(11)			Non	*Aucune*	AUTO_INCREMENT
2	VERSION	int(11)			Non	0	
3	NOM	varchar(20)	utf8_general_ci		Non	*Aucune*	
4	CATEGORIE_ID	bigint(11)			Oui	*NULL*	
5	PRIX	decimal(10,3)			Non	*Aucune*	
6	DESCRIPTION	varchar(100)	utf8_general_ci		Non	*Aucune*	

```
1. package spring.data.entities;
2.
3. import javax.persistence.Column;
4. import javax.persistence.Entity;
5. import javax.persistence.FetchType;
6. import javax.persistence.JoinColumn;
7. import javax.persistence.ManyToOne;
8. import javax.persistence.Table;
9.
10.  import com.fasterxml.jackson.annotation.JsonFilter;
11.
12.  @Entity
13.  @Table(name = "PRODUITS")
14.  @JsonFilter("jsonFilterProduit")
15.  public class Produit extends AbstractEntity {
16.
17.    // propriétés
18.    @Column(name = "NOM")
19.    private String nom;
20.
21.    @Column(name = "CATEGORIE_ID", insertable = false, updatable = false)
22.    private Long idCategorie;
23.
24.    @Column(name = "PRIX")
25.    private double prix;
26.
27.    @Column(name = "DESCRIPTION")
28.    private String description;
29.
30.    // la catégorie
31.    @ManyToOne(fetch = FetchType.LAZY)
32.    @JoinColumn(name = "CATEGORIE_ID")
33.    private Categorie categorie;
34.
35.    // constructeurs
36.    public Produit() {
37.
38.    }
39.
40.    public Produit(String nom, double prix, String description) {
41.        this.nom = nom;
42.        this.prix = prix;
43.        this.description = description;
44.    }
45.
46.    // getters et setters
47.  ...
48.  }
```

- ligne 12 : l'annotation [@Entity] fait de la classe [Produit] une entité gérée par la couche [JPA] ;
- ligne 13 : l'annotation [@Table(name = "PRODUITS")] indique que la classe [Produit] est l'image objet d'une ligne de la table [PRODUITS] de la base de données ;
- ligne 14 : le nom du filtre jSON à appliquer à l'entité. Nous allons voir que la propriété [categorie] de la ligne 13 n'est pas toujours disponible. Il faut alors l'exclure de la représentation jSON de l'objet. Pour cela nous avons besoin d'un filtre. C'est donc dans un filtre nommé [jsonFilterCategorie] que nous indiquerons si on veut ou non la propriété [categorie] ;

- ligne 18 : l'annotation [@Column] associe le champ [nom] à la colonne [NOM] de la table [PRODUITS]. Lorsque le champ porte le même nom que la colonne associée, l'annotation [@Column] peut être omise. Ce serait le cas ici ;
- lignes 31-33 : la catégorie du produit ;
- ligne 31 : l'annotation [@ManyToOne] indique que la colonne de l'annotation de la ligne 32 [@JoinColumn(name = "CATEGORIE_ID")] est clé étrangère de la table [PRODUITS] de l'entité [Produit] sur la table [CATEGORIES] associée à l'entité de la ligne 33. Cette annotation doit annoter une entité JPA. Ainsi la classe de la ligne 33 doit être une entité JPA ;
- ligne 31 : l'annotation [fetch = FetchType.***LAZY***] demande à ce que lorsqu'on ramène un produit de la table [PRODUITS], sa catégorie (ligne 33) ne soit pas ramenée immédiatement (lazy loading). Elle est alors obtenue lors du premier appel à la méthode [getCategorie]. Cet attribut **n'est pas contraignant**. L'implémentation JPA utilisée a le droit de l'ignorer. C'est parce que la propriété [categorie] peut être présente ou non que nous avons introduit le filtre jSON de la ligne 14. Les implémentations JPA existantes (Hibernate, Eclipselink, OpenJPA) ne gèrent pas cette annotation de la même façon. Hibernate enrichit la méthode [getCategorie] initiale (qui se contente de rendre le champ *categorie*) par un appel au SGBD pour aller chercher la catégorie. Pour que cela soit possible, il faut que la connexion au SGBD utilisée initialement pour obtenir le produit soit encore ouverte sinon on a une exception.

11.3.5.3 L'entité JPA [Categorie]

La classe [Categorie] est une entité JPA associée à une ligne de la table [CATEGORIES] :

#	Nom	Type	Interclassement	Attributs	Null	Défaut	Extra
1	ID	bigint(20)			Non	*Aucune*	AUTO_INCREMENT
2	VERSION	int(11)			Non	0	
3	NOM	varchar(20)	utf8_general_ci		Non	*Aucune*	

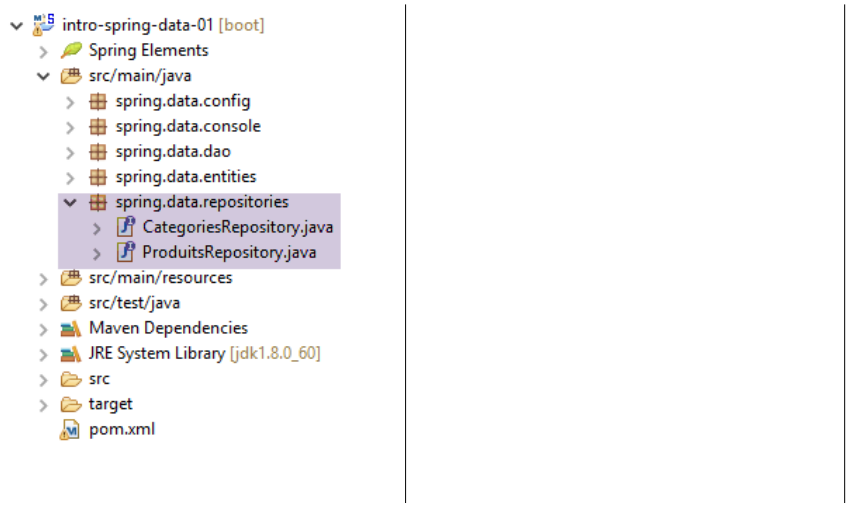
Son code est le suivant :

```
1. package spring.data.entities;
2.
3. import java.util.HashSet;
4. import java.util.Set;
5.
6. import javax.persistence.CascadeType;
7. import javax.persistence.Column;
8. import javax.persistence.Entity;
9. import javax.persistence.FetchType;
10.  import javax.persistence.OneToMany;
11.  import javax.persistence.Table;
12.
13.  import com.fasterxml.jackson.annotation.JsonFilter;
14.
15.  @Entity
16.  @Table(name = "CATEGORIES")
17.  @JsonFilter("jsonFilterCategorie")
18.  public class Categorie extends AbstractEntity {
19.
20.    // propriétés
21.    @Column(name = "NOM")
22.    private String nom;
23.
24.    // les produits associés
25.    @OneToMany(fetch = FetchType.LAZY, mappedBy = "categorie", cascade = { CascadeType.ALL })
26.    public Set<Produit> produits = new HashSet<Produit>();
27.
28.    // constructeurs
29.    public Categorie() {
30.
31.    }
32.
33.    public Categorie(String nom) {
34.        this.nom = nom;
35.    }
36.
37.    // méthodes
38.    public void addProduit(Produit produit) {
39.        // on ajoute le produit
40.        produits.add(produit);
41.        // on fixe sa catégorie
42.        produit.setCategorie(this);
43.    }
44.
```

```
45.    // getters et setters
46.  ...
47.  }
```

- lignes 21-22 : le nom de la catégorie ;
- lignes 25-26 : les produits de cette catégorie ;
- ligne 25 : l'annotation [@OneToMany] est la **relation inverse** de la relation [@ManyToOne] que nous avons rencontrée dans l'entité [Produit]. L'attribut [mappedBy = "categorie"] indique le champ de l'entité [Produit] annoté par la relation inverse [@ManyToOne]. L'attribut [cascade = { CascadeType.***ALL*** }] demande à ce que les opérations (persist, merge, remove) faites sur une @Entity [Categorie] soient cascadées sur les [produits] de la ligne 26. On peut indiquer des cascades partielles avec les constantes [CascadeType.***PERSIST***, CascadeType.***MERGE***, CascadeType.***REMOVE***] ;
- ligne 25 : l'attribut [fetch = FetchType.***LAZY***] demande à ce que, lorsqu'on ramène une catégorie de la table [CATEGORIES], ses produits ne soient pas immédiatement ramenés. Ils le seront lors du premier appel à la méthode [getProduits]. Les implémentations JPA existantes (Hibernate, Eclipselink, OpenJPA) ne gèrent pas cette annotation de la même façon. Hibernate enrichit la méthode [getProduits] initiale (qui se contente de rendre le champ *produits*) par un appel au SGBD pour aller chercher les produits de la catégorie. Pour que cela soit possible, il faut que la connexion au SGBD utilisée initialement pour obtenir la catégorie soit encore ouverte. Cet attribut **est contraignant**. L'implémentation JPA ne peut l'ignorer. Parce que la propriété [produits] peut être ou non initialisée, nous avons introduit le filtre jSON de la ligne 17 qui nous permettra d'indiquer si on veut ou non cette propriété ;
- ligne 26 : le type [Set] est une interface. Le type [HashSet] est une classe implémentant cette interface. Elle implémente une collection d'éléments appelée *ensemble*. Un ensemble ne peut contenir deux objets identiques. Ici les objets sont de type [Produit]. Ainsi dans l'ensemble, on ne pourra avoir deux objets identiques. Comme la méthode [equals] de la classe parent [AbstractEntity] a été redéfinie pour dire que deux produits sont identiques s'ils ont la même clé primaire, alors le champ [produits] ne pourra contenir deux produits de même clé primaire ;
- lignes 38-43 : la méthode [addProduit] permet d'ajouter un produit à la catégorie ;

11.3.6 La couche [Spring Data]



L'interface [CategoriesRepository] gère les accès à la table [CATEGORIES] :

```
1. package spring.data.repositories;
2.
3. import org.springframework.data.jpa.repository.Query;
4. import org.springframework.data.repository.CrudRepository;
5.
6. import spring.data.entities.Categorie;
7.
8. public interface CategoriesRepository extends CrudRepository<Categorie, Long> {
9.
10.    // categorie avec ses produits
11.    @Query("select c from Categorie c left join fetch c.produits p where c.id=?1")
12.    public Categorie getCategorieByIdWithProduits(Long id);
13.
14.    @Query("select c from Categorie c left join fetch c.produits p where c.nom=?1")
15.    public Categorie getCategorieByNameWithProduits(String nom);
16.
17.    // une catégorie sans ses produits désignée par son nom
18.    public Categorie findByNom(String nom);
19. }
```

- ligne 8 : l'interface [CrudRepository] a été utilisée et expliquée page 170 . On rappelle que :
  - le 1er type de l'interface est l'entité JPA gérée pour des accès CRUD (findOne, findAll, save, delete, deleteAll),
  - le second type est celui de la clé primaire de l'entité JPA, ici un entier [Long] ;
- ligne 12 : la méthode de la ligne 12 est implémentée par la requête JPQL (Java Persistence Query Language) de la ligne 11. Celle-ci requête des entités JPA. Dans une telle requête :
  - les tables sont remplacées par leurs entités JPA associées ;
  - les colonnes sont remplacées par des champs des entités JPA utilisées dans la requête ;
- ligne 11 : la requête JPQL ramène une catégorie avec ses produits. On se rappelle que dans l'entité [Categorie], le champ [produits] avait l'attribut [fetch = FetchType.LAZY] (lazy loading). Dans la requête JPQL, on force le chargement des produits avec le mot clé [fetch]. Le paramètre **?1** de la requête sera remplacé à l'exécution par la valeur du 1er paramètre de la méthode de la ligne 12, donc par le paramètre [Long id] ;
- lignes 14-15 : une méthode analogue pour une catégorie identifiée par son nom ;
- ligne 18 : la méthode [findByNom] sera automatiquement implémentée par [Spring Data] car le type [Category] a un champ [nom] ;

L'interface [ProduitsRepository] gère les accès à la table [PRODUITS] :

```
1. package spring.data.repositories;
2.
3. import org.springframework.data.jpa.repository.Query;
4. import org.springframework.data.repository.CrudRepository;
5.
6. import spring.data.entities.Produit;
7.
8. public interface ProduitsRepository extends CrudRepository<Produit, Long> {
9.
10.    // un produit avec sa catégorie
11.    @Query("select p from Produit p left join fetch p.categorie c where p.id=?1")
12.    public Produit getProduitByIdWithCategorie(Long id);
13.
14.    @Query("select p from Produit p left join fetch p.categorie c where p.nom=?1")
15.    public Produit getProduitByNameWithCategorie(String nom);
16.
17.    // un produit sans sa catégorie désigné par son nom
18.    public Produit findByNom(String nom);
19. }
```

Les explications sont les mêmes que pour l'interface [CategoriesRepository].

Ces interfaces vont être implémentées par des classes générées par [Spring Data] au moment de l'exécution du projet. On appelle de telles classes des [proxy]. Par défaut, les méthodes de la classe d'implémentation **s'exécutent dans une transaction**. Le fait que ces interfaces étendent la classe [CrudRepository] font d'elles des composants Spring.

11.3.7 La couche [DAO]

L'interface [IDao] de la couche [DAO] est la suivante :

```
1. package spring.data.dao;
2.
3. import java.util.List;
4.
5. import spring.data.entities.Categorie;
6. import spring.data.entities.Produit;
7.
8. public interface IDao {
```

```
9.
10.    // insertion d'une liste de produits
11.    public List<Produit> addProduits(List<Produit> produits);
12.
13.    // suppression de tous les produits
14.    public void deleteAllProduits();
15.
16.    // mise à jour d'une liste de produits
17.    public List<Produit> updateProduits(List<Produit> produits);
18.
19.    // obtention de tous les produits
20.    public List<Produit> getAllProduits();
21.
22.    // insertion d'une liste de categories
23.    public List<Categorie> addCategories(List<Categorie> categories);
24.
25.    // suppression de tous les categories
26.    public void deleteAllCategories();
27.
28.    // mise à jour d'une liste de categories
29.    public List<Categorie> updateCategories(List<Categorie> categories);
30.
31.    // obtention de tous les categories
32.    public List<Categorie> getAllCategories();
33.
34.    // un produit particulier avec ou non sa catégorie
35.    public Produit getProduitByIdWithoutCategorie(Long idProduit);
36.
37.    public Produit getProduitByIdWithCategorie(Long idProduit);
38.
39.    public Produit getProduitByNameWithCategorie(String nom);
40.
41.    public Produit getProduitByNameWithoutCategorie(String nom);
42.
43.    // une catégorie particulière avec ou pas ses produits
44.    public Categorie getCategorieByIdWithoutProduits(Long idCategorie);
45.
46.    public Categorie getCategorieByIdWithProduits(Long idCategorie);
47.
48.    public Categorie getCategorieByNameWithProduits(String nom);
49.
50.    public Categorie getCategorieByNameWithoutProduits(String nom);
51.  }
```

On a adopté ici la règle que toute méthode qui modifie les objets qu'elle a en paramètres d'entrée doit alors les rendre dans son résultat. La raison de cette règle a été expliquée au paragraphe 4.2, page 34 : elle permet à une couche et à son client d'être dans deux JVM séparées et donc de travailler en client / serveur.

L'implémentation [Dao] de cette interface est la suivante :

```
1. package spring.data.dao;
2.
3. import java.util.ArrayList;
4. import java.util.List;
5.
6. import org.springframework.beans.factory.annotation.Autowired;
7. import org.springframework.stereotype.Component;
8.
9. import com.google.common.collect.Lists;
10.
11.  import spring.data.entities.Categorie;
12.  import spring.data.entities.Produit;
13.  import spring.data.repositories.CategoriesRepository;
14.  import spring.data.repositories.ProduitsRepository;
15.
16.  @Component
17.  public class Dao implements IDao {
18.
19.    @Autowired
20.    private ProduitsRepository produitsRepository;
21.
22.    @Autowired
23.    private CategoriesRepository categoriesRepository;
24.
25.    @Override
26.    public List<Produit> addProduits(List<Produit> produits) {
27.        try {
28.            return Lists.newArrayList(produitsRepository.save(produits));
29.        } catch (Exception e) {
30.            throw new DaoException(101, getMessagesForException(e));
31.        }
32.    }
33.
34.    @Override
```

```
35.    public void deleteAllProduits() {
36.        try {
37.            produitsRepository.deleteAll();
38.        } catch (Exception e) {
39.            throw new DaoException(102, getMessagesForException(e));
40.        }
41.    }
42.
43.    @Override
44.    public List<Produit> updateProduits(List<Produit> produits) {
45.        try {
46.            return Lists.newArrayList(produitsRepository.save(produits));
47.        } catch (Exception e) {
48.            throw new DaoException(103, getMessagesForException(e));
49.        }
50.    }
51.
52.    @Override
53.    public List<Categorie> addCategories(List<Categorie> categories) {
54.        try {
55.            return Lists.newArrayList(categoriesRepository.save(categories));
56.        } catch (Exception e) {
57.            throw new DaoException(104, getMessagesForException(e));
58.        }
59.    }
60.
61.    @Override
62.    public void deleteAllCategories() {
63.        try {
64.            categoriesRepository.deleteAll();
65.        } catch (Exception e) {
66.            throw new DaoException(105, getMessagesForException(e));
67.        }
68.    }
69.
70.    @Override
71.    public List<Categorie> updateCategories(List<Categorie> categories) {
72.        try {
73.            return Lists.newArrayList(categoriesRepository.save(categories));
74.        } catch (Exception e) {
75.            throw new DaoException(106, getMessagesForException(e));
76.        }
77.    }
78.
79.    @Override
80.    public List<Categorie> getAllCategories() {
81.        try {
82.            return Lists.newArrayList(categoriesRepository.findAll());
83.        } catch (Exception e) {
84.            throw new DaoException(107, getMessagesForException(e));
85.        }
86.    }
87.
88.    @Override
89.    public List<Produit> getAllProduits() {
90.        try {
91.            return Lists.newArrayList(produitsRepository.findAll());
92.        } catch (Exception e) {
93.            throw new DaoException(108, getMessagesForException(e));
94.        }
95.    }
96.
97.    @Override
98.    public Produit getProduitByIdWithCategorie(Long idProduit) {
99.        try {
100.           return produitsRepository.getProduitByIdWithCategorie(idProduit);
101.       } catch (Exception e) {
102.           throw new DaoException(109, getMessagesForException(e));
103.       }
104.   }
105.
106.   @Override
107.   public Categorie getCategorieByIdWithProduits(Long idCategorie) {
108.       try {
109.           return categoriesRepository.getCategorieByIdWithProduits(idCategorie);
110.       } catch (Exception e) {
111.           throw new DaoException(110, getMessagesForException(e));
112.       }
113.   }
114.
115.   @Override
116.   public Categorie getCategorieByNameWithProduits(String nom) {
117.       try {
118.           return categoriesRepository.getCategorieByNameWithProduits(nom);
119.       } catch (Exception e) {
120.           throw new DaoException(111, getMessagesForException(e));
121.       }
```

```
122.  }
123.
124.  @Override
125.  public Produit getProduitByNameWithCategorie(String nom) {
126.      try {
127.          return produitsRepository.getProduitByNameWithCategorie(nom);
128.      } catch (Exception e) {
129.          throw new DaoException(112, getMessagesForException(e));
130.      }
131.  }
132.
133.  @Override
134.  public Produit getProduitByIdWithoutCategorie(Long idProduit) {
135.      try {
136.          return produitsRepository.findOne(idProduit);
137.      } catch (Exception e) {
138.          throw new DaoException(113, getMessagesForException(e));
139.      }
140.  }
141.
142.  @Override
143.  public Categorie getCategorieByIdWithoutProduits(Long idCategorie) {
144.      try {
145.          return categoriesRepository.findOne(idCategorie);
146.      } catch (Exception e) {
147.          throw new DaoException(114, getMessagesForException(e));
148.      }
149.  }
150.
151.  @Override
152.  public Produit getProduitByNameWithoutCategorie(String nom) {
153.      try {
154.          return produitsRepository.findByNom(nom);
155.      } catch (Exception e) {
156.          throw new DaoException(115, getMessagesForException(e));
157.      }
158.  }
159.
160.  @Override
161.  public Categorie getCategorieByNameWithoutProduits(String nom) {
162.      try {
163.          return categoriesRepository.findByNom(nom);
164.      } catch (Exception e) {
165.          throw new DaoException(116, getMessagesForException(e));
166.      }
167.  }
168.
169. }
```

- ligne 16 : l'annotation [@Component] fait de la classe [Dao] un composant Spring ;
- lignes 19-23 : injection des références sur les deux interfaces [CrudRepository] de [Spring Data]. Cette injection aura lieu lors de l'instantiation des objets Spring, en général au début de l'exécution du projet Spring ;
- on notera lignes 28 et 46 que la méthode [save] de l'interface [produitsRepository] est utilisée aussi bien pour l'insertion que pour la mise à jour de produits. [Spring Data] utilise la clé primaire du produit pour savoir s'il doit faire une insertion ou une mise à jour. Si la clé primaire vaut [null], ce sera une insertion sinon ce sera une mise à jour ;
- ligne 82 : on utilise la méthode [Lists.*newArrayList*] de la bibliothèque Guava pour obtenir une liste de produits. La méthode [produitsRepository.findAll()] rend un type [Iterable<Produit>] ;
- ligne 28 : la méthode [produitsRepository.save(produits)] rend un type [Iterable<Produit>]. Il en est de même pour les autres opérations [save] de la classe ;

Dans la classe [Dao] ci-dessus, les exceptions qui peuvent se produire sont encapsulées dans le type [DaoException] suivant :

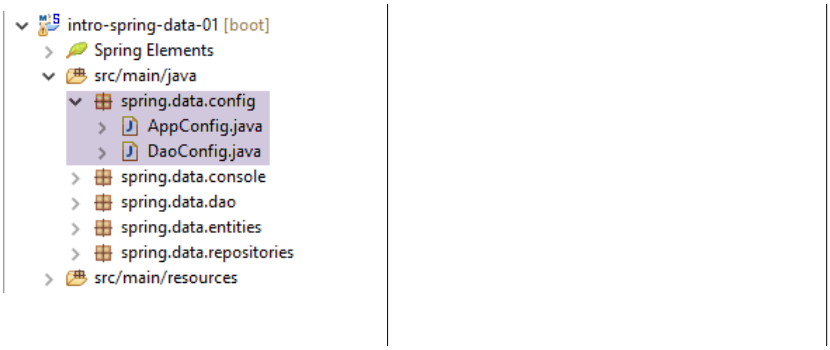
```
1. package spring.data.dao;
2.
3. import java.io.Serializable;
4. import java.util.ArrayList;
5. import java.util.List;
6.
7. // classe d'exception pour l'application Elections
8. // l'exception est non contrôlée
9.
10.  public class DaoException extends RuntimeException implements Serializable {
11.
12.    // serial ID
13.    private static final long serialVersionUID = 1L;
14.
15.    // champs locaux
16.    private int code;
17.    private List<String> erreurs;
18.
19.    // constructeurs
```

```
20.    public DaoException() {
21.        super();
22.    }
23.
24.    public DaoException(int code, Throwable e) {
25.        // parent
26.        super(e);
27.        // local
28.        this.code = code;
29.        this.erreurs = getErreursForException(e);
30.    }
31.
32.    public DaoException(int code, String message, Throwable e) {
33.        // parent
34.        super(message, e);
35.        // local
36.        this.code = code;
37.        this.erreurs = getErreursForException(e);
38.    }
39.
40.    public DaoException(int code, String message) {
41.        // parent
42.        super(message);
43.        // local
44.        this.code = code;
45.        List<String> erreurs = new ArrayList<>();
46.        erreurs.add(message);
47.        this.erreurs = erreurs;
48.    }
49.
50.    public DaoException(int code, List<String> erreurs) {
51.        // parent
52.        super();
53.        // local
54.        this.code = code;
55.        this.erreurs = erreurs;
56.    }
57.
58.    // liste des messages d'erreur d'une exception
59.    private List<String> getErreursForException(Throwable th) {
60.        // on récupère la liste des messages d'erreur de l'exception
61.        Throwable cause = th;
62.        List<String> erreurs = new ArrayList<>();
63.        while (cause != null) {
64.            // on récupère le message seulement s'il est !=null et non blanc
65.            String message = cause.getMessage();
66.            if (message != null) {
67.                message = message.trim();
68.                if (message.length() != 0) {
69.                    erreurs.add(message);
70.                }
71.            }
72.            // cause suivante
73.            cause = cause.getCause();
74.        }
75.        return erreurs;
76.    }
77.
78.    // getters et setters
79. ...
80.  }
```

- ligne 10 : la classe dérive de la classe [RuntimeException] et est donc une exception non contrôlée ;
- ligne 16 : un code d'erreur ;
- ligne 17 : une liste de messages d'erreur, ceux associés à la pile des exceptions qui ont provoqué la [DaoException] ;
- lignes 59-76 : la méthode privée [getMessagesForException] permet d'obtenir la liste des messages d'erreurs associées aux exception de la pile d'exceptions. Il est en effet possible d'empiler les exceptions avec les constructeurs suivants de la classe **Exception** :
  - **Exception(String message, Throwable cause)** : crée une exception avec un message et l'exception qu'on veut encapsuler ;
  - **Exception(Throwable cause)** : crée une exception avec l'exception qu'on veut encapsuler ;

  Le type [Throwable] est la classe parent de la classe [Exception]. Si les constructeurs précédents sont exécutés de façon répétée, l'exception finale contient alors plusieurs exceptions. On dit qu'on a une pile d'exceptions.
  - la dernière cause d'une exception e1 est obtenue par l'expression [e1.getCause()] ;
  - l'avant-dernière cause d'une exception e1 est obtenue par l'expression [e1.getCause().getCause()] ;
  - on poursuit ainsi jusqu'à obtenir [getCause()==null] ;

11.3.8 Configuration du projet Spring



La classe [DaoConfig] configure la couche [DAO] :

```
1. package spring.data.config;
2.
3. import javax.persistence.EntityManagerFactory;
4.
5. import org.apache.tomcat.jdbc.pool.DataSource;
6. import org.springframework.context.annotation.Bean;
7. import org.springframework.context.annotation.ComponentScan;
8. import org.springframework.context.annotation.Configuration;
9. import org.springframework.data.jpa.repository.config.EnableJpaRepositories;
10.  import org.springframework.orm.jpa.JpaTransactionManager;
11.  import org.springframework.orm.jpa.JpaVendorAdapter;
12.  import org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean;
13.  import org.springframework.orm.jpa.vendor.Database;
14.  import org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter;
15.  import org.springframework.transaction.PlatformTransactionManager;
16.
17.  @EnableJpaRepositories(basePackages = { "spring.data.repositories" })
18.  @Configuration
19.  @ComponentScan(basePackages = { "spring.data.dao" })
20.  public class DaoConfig {
21.
22.    // constantes
23.    final static String URL = "jdbc:mysql://localhost:3306/dbIntroSpringData";
24.    final static String USER = "root";
25.    final static String PASSWD = "";
26.    final static String DRIVER_CLASSNAME = "com.mysql.jdbc.Driver";
27.    final static String[] ENTITIES_PACKAGES = { "spring.data.entities" };
28.
29.    // la source de données [tomcat-jdbc]
30.    @Bean
31.    public DataSource dataSource() {
32.        // source de données TomcatJdbc
33.        DataSource dataSource = new DataSource();
34.        // configuration accès JDBC
35.        dataSource.setDriverClassName(DRIVER_CLASSNAME);
36.        dataSource.setUsername(USER);
37.        dataSource.setPassword(PASSWD);
38.        dataSource.setUrl(URL);
39.        // une connexion ouverte initialement
40.        dataSource.setInitialSize(1);
41.        // résultat
42.        return dataSource;
43.    }
44.
45.    // le provider JPA
46.    @Bean
47.    public JpaVendorAdapter jpaVendorAdapter() {
48.        HibernateJpaVendorAdapter hibernateJpaVendorAdapter = new HibernateJpaVendorAdapter();
49.        hibernateJpaVendorAdapter.setShowSql(false);
50.        hibernateJpaVendorAdapter.setDatabase(Database.MYSQL);
51.        return hibernateJpaVendorAdapter;
52.    }
53.
54.    // EntityManagerFactory
55.    @Bean
56.    public EntityManagerFactory entityManagerFactory(JpaVendorAdapter jpaVendorAdapter, DataSource dataSource) {
57.        LocalContainerEntityManagerFactoryBean factory = new LocalContainerEntityManagerFactoryBean();
58.        factory.setJpaVendorAdapter(jpaVendorAdapter);
59.        factory.setPackagesToScan(packagesToScan());
60.        factory.setDataSource(dataSource);
61.        factory.afterPropertiesSet();
62.        return factory.getObject();
63.    }
```

```
64.
65.    // Transaction manager
66.    @Bean
67.    public PlatformTransactionManager transactionManager(EntityManagerFactory entityManagerFactory) {
68.        JpaTransactionManager txManager = new JpaTransactionManager();
69.        txManager.setEntityManagerFactory(entityManagerFactory);
70.        return txManager;
71.    }
72.
73.    @Bean
74.    public String[] packagesToScan() {
75.        return ENTITIES_PACKAGES;
76.    }
77.
78. }
```

Une configuration analogue a été rencontrée et expliquée page 176. Nous y avons ajouté les annotations Spring suivantes :

- ligne 17 : l'annotation [@EnableJpaRepositories] sert à indiquer les packages où se trouvent les interfaces [CrudRepository] de [Spring Data] ;
- ligne 18 : la classe est une classe de configuration Spring. Cette information est importante. Si on l'enlève le projet fonctionne. Mais plus loin dans le document, lorsqu'on construira des projets qui s'appuient sur celui-ci, alors certains d'entre-eux ne fonctionnent plus si l'annotation de la ligne 18 est enlevée ;
- ligne 19 : l'annotation [@ComponentScan] indique les packages où se trouvent les objets Spring. Ce sont les classes annotées par [@Component, @Service, @Controller, ...]. Ici le composant Spring [Dao] va être trouvé et instancié ;
- lignes 73-76 : nous avons défini un bean qui représente le tableau des packages à scanner pour trouver des entités JPA. Cela permettra à un projet qui importe la classe [DaoConfig] de redéfinir ce bean et de changer ainsi les packages scannés ligne 59. Plus loin dans le document, nous allons rencontrer cette problématique ;

La classe [AppConfig] configure l'ensemble du projet :

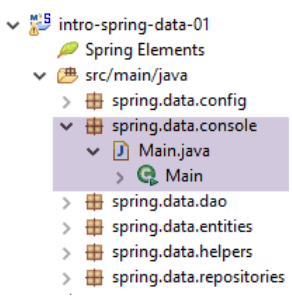
```
1. package spring.data.config;
2.
3. import org.springframework.context.annotation.Bean;
4. import org.springframework.context.annotation.Configuration;
5. import org.springframework.context.annotation.Import;
6.
7. import com.fasterxml.jackson.databind.ObjectMapper;
8. import com.fasterxml.jackson.databind.ser.impl.SimpleBeanPropertyFilter;
9. import com.fasterxml.jackson.databind.ser.impl.SimpleFilterProvider;
10.
11.  @Configuration
12.  @Import({DaoConfig.class})
13.  public class AppConfig {
14.    // filtres jSON
15.    @Bean(name = "jsonMapper")
16.    public ObjectMapper jsonMapper() {
17.        return new ObjectMapper();
18.    }
19.
20.    @Bean(name = "jsonMapperCategorieWithProduits")
21.    public ObjectMapper jsonMapperCategorieWithProduits() {
22.        // mappeur jSON
23.        ObjectMapper mapper = new ObjectMapper();
24.        // filtres
25.        mapper.setFilters(
26.                new SimpleFilterProvider().addFilter("jsonFilterCategorie", SimpleBeanPropertyFilter.serializeAllExcept())
27.                        .addFilter("jsonFilterProduit", SimpleBeanPropertyFilter.serializeAllExcept("categorie")));
28.        // résultat
29.        return mapper;
30.    }
31.
32.    @Bean(name = "jsonMapperProduitWithCategorie")
33.    public ObjectMapper jsonMapperProduitWithCategorie() {
34.        // mappeur jSON
35.        ObjectMapper mapper = new ObjectMapper();
36.        // filtres
37.        mapper.setFilters(
38.                new SimpleFilterProvider().addFilter("jsonFilterProduit", SimpleBeanPropertyFilter.serializeAllExcept())
39.                        .addFilter("jsonFilterCategorie", SimpleBeanPropertyFilter.serializeAllExcept("produits")));
40.        // résultat
41.        return mapper;
42.    }
43.
44.    @Bean(name = "jsonMapperCategorieWithoutProduits")
45.    public ObjectMapper jsonMapperCategorieWithoutProduits() {
46.        // mappeur jSON
47.        ObjectMapper mapper = new ObjectMapper();
48.        // filtres
49.        mapper.setFilters(new SimpleFilterProvider().addFilter("jsonFilterCategorie",
50.                SimpleBeanPropertyFilter.serializeAllExcept("produits")));
```

```
51.       // résultat
52.       return mapper;
53.   }
54.
55.   @Bean(name = "jsonMapperProduitWithoutCategorie")
56.   public ObjectMapper jsonMapperProduitWithoutCategorie() {
57.       // mappeur jSON
58.       ObjectMapper mapper = new ObjectMapper();
59.       // filtres
60.       mapper.setFilters(new SimpleFilterProvider().addFilter("jsonFilterProduit",
61.               SimpleBeanPropertyFilter.serializeAllExcept("categorie")));
62.       // résultat
63.       return mapper;
64.   }
65. }
```

- ligne 11 : la classe est une classe de configuration Spring ;
- ligne 12 : qui importe les beans définis par la classe [DaoConfig] que nous venons de voir ;
- la couche [console] utilise des mappeurs jSON qui sont définis ici ;
- lignes 14-64 : définissent cinq mappeurs jSON ;
- lignes 15-18 : le filtre jSON [jsonMapper] n'a pas de filtres ;
- lignes 20-30 : le filtre jSON [jsonMapperCategorieWithProduits] permet de sérialiser / désérialiser un objet [Categorie] **avec** ses produits ;
- lignes 32-42 : le filtre jSON [jsonMapperProduitWithCategorie] permet de sérialiser / désérialiser un objet [Produit] **avec** sa catégorie ;
- lignes 43-53 : le filtre jSON [jsonMapperCategorieWithoutProduits] permet de sérialiser / désérialiser un objet [Categorie] **sans** ses produits ;
- lignes 55-64 : le filtre jSON [jsonMapperProduitWithoutCategorie] permet de sérialiser / désérialiser un objet [Produit] **sans** sa catégorie ;

On notera que l'on construit un filtre jSON pour une entité T, on doit configurer non seulement le filtre de l'entité T mais également ceux des entités Ti qu'elle-même peut contenir.

11.3.9 La couche [console]



La classe [Main] est la suivante :

```
1. package spring.data.console;
2.
3. import java.util.ArrayList;
4. import java.util.List;
5. import java.util.Set;
6.
7. import org.springframework.context.annotation.AnnotationConfigApplicationContext;
8.
9. import com.fasterxml.jackson.core.JsonProcessingException;
10.  import com.fasterxml.jackson.databind.ObjectMapper;
11.  import com.google.common.collect.Lists;
12.
13.  import spring.data.config.AppConfig;
14.  import spring.data.dao.DaoException;
15.  import spring.data.dao.IDao;
16.  import spring.data.entities.Categorie;
17.  import spring.data.entities.Produit;
18.
19.  public class Main {
20.
21.    public static void main(String[] args) throws JsonProcessingException {
22.       AnnotationConfigApplicationContext context = null;
```

```
23.       try {
24.           // instanciation contexte Spring
25.           context = new AnnotationConfigApplicationContext(AppConfig.class);
26.           ObjectMapper jsonMapperCategorieWithProduits = context.getBean("jsonMapperCategorieWithProduits",
27.                   ObjectMapper.class);
28.           ObjectMapper jsonMapperProduitWithCategorie = context.getBean("jsonMapperProduitWithCategorie",
29.                   ObjectMapper.class);
30.           ObjectMapper jsonMapperCategorieWithoutProduits = context.getBean("jsonMapperCategorieWithoutProduits",
31.                   ObjectMapper.class);
32.           ObjectMapper jsonMapperProduitWithoutCategorie = context.getBean("jsonMapperProduitWithoutCategorie",
33.                   ObjectMapper.class);
34.           IDao dao = context.getBean(IDao.class);
35.           // ------------------------------------------------------------------------------------
36.           // on vide la base de données
37.           log("Vidage de la base de données", 1);
38.           // on vide la table [CATEGORIES] - par cascade la table [PRODUITS] va être vidée
39.           dao.deleteAllCategories();
40.           // ------------------------------------------------------------------------------------
41.           log("Remplissage de la base", 1);
42.           // on remplit les tables
43.           List<Categorie> categories = new ArrayList<Categorie>();
44.           for (int i = 0; i < 2; i++) {
45.               Categorie categorie = new Categorie(String.format("categorie%d", i));
46.               for (int j = 0; j < 5; j++) {
47.                   categorie.addProduit(new Produit(String.format("produit%d%d", i, j), 100 * (1 + (double) (i * 10 + j) /
    100),
48.                           String.format("desc%d%d", i, j)));
49.               }
50.               categories.add(categorie);
51.           }
52.           // ajout de la catégorie - par cascade les produits vont eux aussi être insérés
53.           dao.addCategories(categories);
54.           // ------------------------------------------------------------------------------------
55.           log("Affichage de la base", 1);
56.           // liste des catégories
57.           log("Liste des catégories", 2);
58.           affiche(dao.getAllCategories(), jsonMapperCategorieWithoutProduits);
59.           // liste des produits
60.           log("Liste des produits", 2);
61.           affiche(dao.getAllProduits(), jsonMapperProduitWithoutCategorie);
62.           // catégorie 1 avec ses produits
63.           Categorie categorie = dao.getCategorieByNameWithProduits("categorie1");
64.           log("Catégorie 1 avec ses produits", 2);
65.           affiche(categorie, jsonMapperCategorieWithProduits);
66.           // le produit [produit14] avec sa catégorie
67.           Produit p = dao.getProduitByNameWithCategorie("produit14");
68.           log("Produit [produit14] avec sa catégorie", 2);
69.           affiche(p, jsonMapperProduitWithCategorie);
70.           // ------------------------------------------------------------------------------------
71.           log("Mise à jour du prix des produits de [categorie1]", 1);
72.           log("Produits de la catégorie [categorie1] avant la mise à jour", 2);
73.           Categorie categorie1 = dao.getCategorieByNameWithProduits("categorie1");
74.           Set<Produit> produits = categorie1.getProduits();
75.           affiche(categorie1, jsonMapperCategorieWithProduits);
76.           for (Produit produit : produits) {
77.               produit.setPrix(1.1 * produit.getPrix());
78.           }
79.           dao.updateProduits(Lists.newArrayList(produits));
80.           log("Produits de la catégorie [categorie1] après la mise à jour", 2);
81.           affiche(dao.getCategorieByNameWithProduits("categorie1"), jsonMapperCategorieWithProduits);
82.           // ------------------------------------------------------------------------------------
83.           log("Vidage de la base de données", 1);
84.           // on vide la table [CATEGORIES] - par cascade la table [PRODUITS] va être vidée
85.           dao.deleteAllCategories();
86.           // affichage de la base
87.           log("Liste des categories avant l'ajout", 2);
88.           affiche(dao.getAllCategories(), jsonMapperCategorieWithoutProduits);
89.           log("Liste des produits avant l'ajout", 2);
90.           affiche(dao.getAllProduits(), jsonMapperProduitWithoutCategorie);
91.           log("Ajout d'une catégorie [cat1] avec deux produits de même nom", 1);
92.           // on fait l'insertion
93.           categorie = new Categorie("cat1");
94.           categorie.addProduit(new Produit("x", 1.0, ""));
95.           categorie.addProduit(new Produit("x", 1.0, ""));
96.           // ajout de la catégorie - par cascade les produits vont eux aussi être insérés
97.           try {
98.               dao.addCategories(Lists.newArrayList(categorie));
99.           } catch (DaoException e) {
100.              System.out.println(e);
101.          }
102.          // vérification
103.          log("Liste des categories après l'ajout", 2);
104.          affiche(dao.getAllCategories(), jsonMapperCategorieWithoutProduits);
105.          log("Liste des produits après l'ajout", 2);
106.          affiche(dao.getAllProduits(), jsonMapperProduitWithoutCategorie);
107.      } catch (DaoException e) {
108.          System.out.println(e);
```

```
109.      } finally {
110.          if (context != null) {
111.              // fini
112.              context.close();
113.          }
114.      }
115.      System.out.println("Travail terminé");
116.  }
117.
118.  // affichage d'un élément de type T
119.  static private <T> void affiche(T element, ObjectMapper jsonMapper) throws JsonProcessingException {
120.      System.out.println(jsonMapper.writeValueAsString(element));
121.  }
122.
123.  // affichage d'une liste d'éléments de type T
124.  static private <T> void affiche(List<T> elements, ObjectMapper jsonMapper) throws JsonProcessingException {
125.      for (T element : elements) {
126.          affiche(element, jsonMapper);
127.      }
128.  }
129.
130.  private static void log(String message, int mode) {
131.      // affiche message
132.      String toPrint = null;
133.      switch (mode) {
134.      case 1:
135.          toPrint = String.format("%s --------------------------------", message);
136.          break;
137.      case 2:
138.          toPrint = String.format("-- %s", message);
139.          break;
140.      }
141.      System.out.println(toPrint);
142.  }
143. }
```

- ligne 25 : instanciation des beans Spring à partir de la classe de configuration [AppConfig] ;
- lignes 26-33 : récupération des références sur les mappeurs jSON. On utilise la signature suivante de la méthode [ApplicationContext].getBean :
  - **[ApplicationContext].getBean(String id, Class classe)** : qu'on utilise lorsqu'il y a plusieurs beans ayant le type [classe]. Dans ce cas, on précise l'identifiant du bean demandé. Si celui-ci a été défini avec l'annotation [**@Bean**], son identifiant est le nom de la méthode annotée. S'il a été défini avec l'annotation [**@Bean(« identifiant »**], son identifiant est celui indiqué dans l'annotation ;
- ligne 34 : récupération d'une référence sur la couche [DAO] ;
- lignes 37-39 : vidage de la base de données. On vide la table des catégories (ligne 39). Parce qu'on a écrit :

```
@OneToMany(fetch = FetchType.LAZY, mappedBy = "categorie", cascade = { CascadeType.ALL })
public Set<Produit> produits = new HashSet<Produit>();
```

lorsqu'une catégorie est supprimée, tous les produits qui lui sont liés le sont aussi ;

- lignes 43-53 : remplissage de la table avec 2 catégories de 5 produits chacune. Ligne 50, l'insertion des deux catégories va provoquer en même temps l'insertion de leurs produits, toujours parce qu'on a écrit [cascade = { CascadeType.***ALL*** }] ;
- ligne 58 : on affiche les catégories. On utilise le mappeur jSON [`jsonMapperCategorieWithoutProduits`] pour afficher les catégories sans leurs produits. En effet la méthode [`dao.getAllCategories()`] rend les catégories sans leurs produits (lazy loading) ;
- ligne 61 : on affiche les produits sans leur catégorie. En effet la méthode [`dao.getAllProduits()`] rend les produits sans leur catégorie (lazy loading) ;
- lignes 63-65 : affichent la catégorie de nom [categorie1] avec ses produits (eager loading) ;
- lignes 67-69 : affiche un produit avec sa catégorie ;
- lignes 71-81 : on augmente de 10% tous les prix des produits de la catégorie [categorie1] ;
- lignes 91-101 : on ajoute une catégorie avec deux produits de même nom. Or dans la table [PRODUITS] on a une contrainte d'unicité sur la colonne [NOM]. L'insertion du 2ième produit va donc être rejetée et une exception lancée. Or la méthode [`dao.addProduits`] s'exécute dans une transaction. Le fait que la seconde insertion échoue doit donc également annuler l'insertion du premier produit ainsi que celle de leur catégorie [cat1]. C'est ce qu'on veut vérifier ;
- lignes 119-121 : une méthode générique capable d'afficher la chaîne jSON de tout élément de type T. La sérialisation jSON est contrôlée par le mappeur passé en paramètre ;
- lignes 124-128 : une méthode analogue, cette-fois pour une liste d'éléments de type T ;

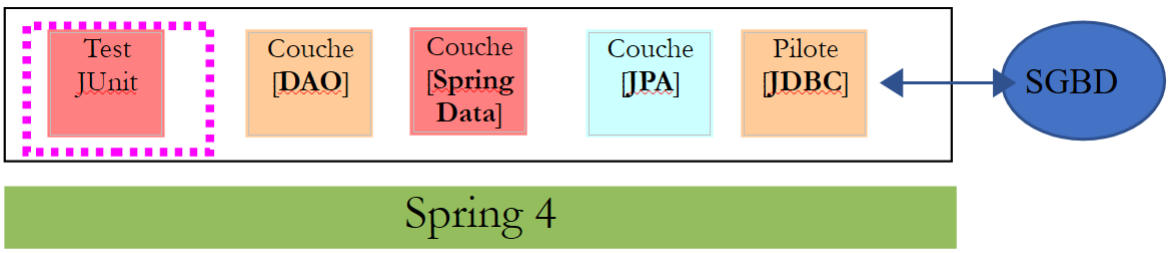
L'exécution de la classe [Main] donne les résultats suivants (hors logs de Spring) :

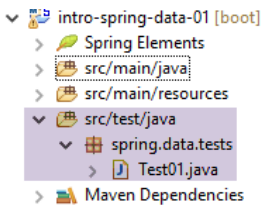
```
1. Vidage de la base de données --------------------------------
2. Remplissage de la base --------------------------------
3. Affichage de la base --------------------------------
```

```
4. -- Liste des catégories
5. {"id":4,"version":0,"nom":"categorie0"}
6. {"id":5,"version":0,"nom":"categorie1"}
7. -- Liste des produits
8. {"id":13,"version":0,"nom":"produit00","idCategorie":4,"prix":100.0,"description":"desc00"}
9. {"id":14,"version":0,"nom":"produit01","idCategorie":4,"prix":101.0,"description":"desc01"}
10.  {"id":15,"version":0,"nom":"produit02","idCategorie":4,"prix":102.0,"description":"desc02"}
11.  {"id":16,"version":0,"nom":"produit03","idCategorie":4,"prix":103.0,"description":"desc03"}
12.  {"id":17,"version":0,"nom":"produit04","idCategorie":4,"prix":104.0,"description":"desc04"}
13.  {"id":18,"version":0,"nom":"produit10","idCategorie":5,"prix":110.0,"description":"desc10"}
14.  {"id":19,"version":0,"nom":"produit11","idCategorie":5,"prix":111.0,"description":"desc11"}
15.  {"id":20,"version":0,"nom":"produit12","idCategorie":5,"prix":112.0,"description":"desc12"}
16.  {"id":21,"version":0,"nom":"produit13","idCategorie":5,"prix":113.0,"description":"desc13"}
17.  {"id":22,"version":0,"nom":"produit14","idCategorie":5,"prix":114.0,"description":"desc14"}
18.  -- Catégorie 1 avec ses produits
19.  {"id":5,"version":0,"nom":"categorie1","produits":
     [{"id":18,"version":0,"nom":"produit10","idCategorie":5,"prix":110.0,"description":"desc10"},
     {"id":19,"version":0,"nom":"produit11","idCategorie":5,"prix":111.0,"description":"desc11"},
     {"id":20,"version":0,"nom":"produit12","idCategorie":5,"prix":112.0,"description":"desc12"},
     {"id":21,"version":0,"nom":"produit13","idCategorie":5,"prix":113.0,"description":"desc13"},
     {"id":22,"version":0,"nom":"produit14","idCategorie":5,"prix":114.0,"description":"desc14"}]}
20.  -- Produit [produit14] avec sa catégorie
21.  {"id":22,"version":0,"nom":"produit14","idCategorie":5,"prix":114.0,"description":"desc14","categorie":
     {"id":5,"version":0,"nom":"categorie1"}}
22.  Mise à jour du prix des produits de [categorie1] --------------------------------
23.  -- Produits de la catégorie [categorie1] avant la mise à jour
24.  {"id":5,"version":0,"nom":"categorie1","produits":
     [{"id":18,"version":0,"nom":"produit10","idCategorie":5,"prix":110.0,"description":"desc10"},
     {"id":19,"version":0,"nom":"produit11","idCategorie":5,"prix":111.0,"description":"desc11"},
     {"id":20,"version":0,"nom":"produit12","idCategorie":5,"prix":112.0,"description":"desc12"},
     {"id":21,"version":0,"nom":"produit13","idCategorie":5,"prix":113.0,"description":"desc13"},
     {"id":22,"version":0,"nom":"produit14","idCategorie":5,"prix":114.0,"description":"desc14"}]}
25.  -- Produits de la catégorie [categorie1] après la mise à jour
26.  {"id":5,"version":0,"nom":"categorie1","produits":
     [{"id":18,"version":1,"nom":"produit10","idCategorie":5,"prix":121.0,"description":"desc10"},
     {"id":19,"version":1,"nom":"produit11","idCategorie":5,"prix":122.1,"description":"desc11"},
     {"id":20,"version":1,"nom":"produit12","idCategorie":5,"prix":123.2,"description":"desc12"},
     {"id":21,"version":1,"nom":"produit13","idCategorie":5,"prix":124.3,"description":"desc13"},
     {"id":22,"version":1,"nom":"produit14","idCategorie":5,"prix":125.4,"description":"desc14"}]}
27.  Vidage de la base de données ------------------------------
28.  -- Liste des categories avant l'ajout
29.  -- Liste des produits avant l'ajout
30.  Ajout d'une catégorie [cat1] avec deux produits de même nom -------------------------------
31.  Les erreurs suivantes se sont produites :
32.  - org.hibernate.exception.ConstraintViolationException: could not execute statement
33.  - could not execute statement
34.  - Duplicate entry 'x' for key 'NOM'
35.  -- Liste des categories après l'ajout
36.  -- Liste des produits après l'ajout
37.  Travail terminé
```

- lignes 4-17 : les catégories et produits insérés dans la table ;
- lignes 18-19 : une catégorie avec ses produits ;
- lignes 20-21 : un produit avec sa catégorie ;
- lignes 22-26 : mise à jour du prix de certains produits. Ligne 24, on voit que les prix ont bien augmenté de 10% ;
- lignes 27-36 : ajout de la catégorie [cat1] avec deux produits de même nom. On voit que la table est la même avant (lignes 28-29) et après ajout (lignes 35-36) montrant par là que toutes les insertions de la transaction ont bien été annulées ;
- lignes 31-34 : l'exception qui s'est produite lors de l'insertion du second produit et qui a fait échouer toute la transaction ;

11.3.10 Le test unitaire JUnit





La classe [Test01] est la suivante :

```
1. package spring.data.tests;
2.
3. import java.util.ArrayList;
4. import java.util.List;
5. import java.util.Set;
6.
7. import org.junit.Assert;
8. import org.junit.Before;
9. import org.junit.Test;
10.  import org.junit.runner.RunWith;
11.  import org.springframework.beans.BeansException;
12.  import org.springframework.beans.factory.annotation.Autowired;
13.  import org.springframework.beans.factory.annotation.Qualifier;
14.  import org.springframework.boot.test.SpringApplicationConfiguration;
15.  import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
16.
17.  import com.fasterxml.jackson.core.JsonProcessingException;
18.  import com.fasterxml.jackson.databind.ObjectMapper;
19.  import com.google.common.collect.Lists;
20.
21.  import spring.data.config.AppConfig;
22.  import spring.data.dao.DaoException;
23.  import spring.data.dao.IDao;
24.  import spring.data.entities.Categorie;
25.  import spring.data.entities.Produit;
26.
27.  @SpringApplicationConfiguration(classes = AppConfig.class)
28.  @RunWith(SpringJUnit4ClassRunner.class)
29.  public class Test01 {
30.
31.    // couche [DAO]
32.    @Autowired
33.    private IDao dao;
34.
35.    // filtres jSON
36.    @Autowired
37.    @Qualifier("jsonMapper")
38.    private ObjectMapper jsonMapper;
39.    @Autowired
40.    @Qualifier("jsonMapperCategorieWithProduits")
41.    private ObjectMapper jsonMapperCategorieWithProduits;
42.    @Autowired
43.    @Qualifier("jsonMapperProduitWithCategorie")
44.    private ObjectMapper jsonMapperProduitWithCategorie;
45.    @Autowired
46.    @Qualifier("jsonMapperCategorieWithoutProduits")
47.    private ObjectMapper jsonMapperCategorieWithoutProduits;
48.    @Autowired
49.    @Qualifier("jsonMapperProduitWithoutCategorie")
50.    private ObjectMapper jsonMapperProduitWithoutCategorie;
51.
52.    @Before
53.    public void cleanAndFill() {
54.        // on nettoie la base avant chaque test
55.        log("Vidage de la base de données", 1);
56.        // on vide la table [CATEGORIES] - par cascade la table [PRODUITS] va être vidée
57.        dao.deleteAllCategories();
58.        // ----------------------------------------------------------------------------------------
59.        log("Remplissage de la base", 1);
60.        // on remplit les tables
61.        List<Categorie> categories = new ArrayList<Categorie>();
62.        for (int i = 0; i < 2; i++) {
63.            Categorie categorie = new Categorie(String.format("categorie%d", i));
64.            for (int j = 0; j < 5; j++) {
65.                categorie.addProduit(new Produit(String.format("produit%d%d", i, j), 100 * (1 + (double) (i * 10 + j) / 100),
66.                        String.format("desc%d%d", i, j)));
67.            }
68.            categories.add(categorie);
69.        }
```

```
70.         // ajout de la catégorie - par cascade les produits vont eux aussi être insérés
71.         categories = dao.addCategories(categories);
72.     }
73.
74.     @Test
75.     public void showDataBase() throws BeansException, JsonProcessingException {
76.         // liste des catégories
77.         log("Liste des catégories", 2);
78.         List<Categorie> categories = dao.getAllCategories();
79.         affiche(categories, jsonMapperCategorieWithoutProduits);
80.         // liste des produits
81.         log("Liste des produits", 2);
82.         List<Produit> produits = dao.getAllProduits();
83.         affiche(produits, jsonMapperProduitWithoutCategorie);
84.         // quelques vérifications
85.         Assert.assertEquals(2, categories.size());
86.         Assert.assertEquals(10, produits.size());
87.         Categorie categorie = findCategorieByName("categorie0", categories);
88.         Assert.assertNotNull(categorie);
89.         Produit produit = findProduitByName("produit03", produits);
90.         Assert.assertNotNull(produit);
91.         Long idCategorie = produit.getIdCategorie();
92.         Assert.assertEquals(categorie.getId(), idCategorie);
93.     }
94.
95.     @Test
96.     public void getCategorieByNameWithProduits() {
97.         log("getCategorieByNameWithProduits", 1);
98.         Categorie categorie1 = dao.getCategorieByNameWithProduits("categorie1");
99.         Assert.assertNotNull(categorie1);
100.        Assert.assertEquals(5, categorie1.getProduits().size());
101.    }
102.
103.    @Test
104.    public void getCategorieByNameWithoutProduits() {
105.        log("getCategorieByNameWithoutProduits", 1);
106.        Categorie categorie1 = dao.getCategorieByNameWithoutProduits("categorie1");
107.        Assert.assertNotNull(categorie1);
108.        Assert.assertEquals("categorie1", categorie1.getNom());
109.    }
110.
111.    @Test
112.    public void getProduitByIdWithCategorie() {
113.        log("getProduitByNameWithCategorie", 1);
114.        Produit produit = dao.getProduitByNameWithCategorie("produit03");
115.        Produit produit2 = dao.getProduitByIdWithCategorie(produit.getId());
116.        Assert.assertNotNull(produit2);
117.        Assert.assertEquals(produit2.getNom(), produit.getNom());
118.        Assert.assertEquals(produit2.getId(), produit.getId());
119.        Assert.assertEquals(produit.getCategorie().getId(), produit2.getCategorie().getId());
120.    }
121.
122.    @Test
123.    public void getProduitByIdWithoutCategorie() {
124.        log("getProduitByIdWithoutCategorie", 1);
125.        Produit produit = dao.getProduitByNameWithCategorie("produit03");
126.        Produit produit2 = dao.getProduitByIdWithoutCategorie(produit.getId());
127.        Assert.assertNotNull(produit2);
128.        Assert.assertEquals(produit2.getNom(), produit.getNom());
129.        Assert.assertEquals(produit2.getId(), produit.getId());
130.    }
131. ...
132.    // -------------- méthodes privées
133.    private Produit findProduitByName(String nom, List<Produit> produits) {
134.        for (Produit produit : produits) {
135.            if (produit.getNom().equals(nom)) {
136.                return produit;
137.            }
138.        }
139.        return null;
140.    }
141.
142.    private Categorie findCategorieByName(String nom, List<Categorie> categories) {
143.        for (Categorie categorie : categories) {
144.            if (categorie.getNom().equals(nom)) {
145.                return categorie;
146.            }
147.        }
148.        return null;
149.    }
150.
151.    // affichage d'un élément de type T
152.    static private <T> void affiche(T element, ObjectMapper jsonMapper) throws JsonProcessingException {
153.        System.out.println(jsonMapper.writeValueAsString(element));
154.    }
155.
156.    // affichage d'une liste d'éléments de type T
```

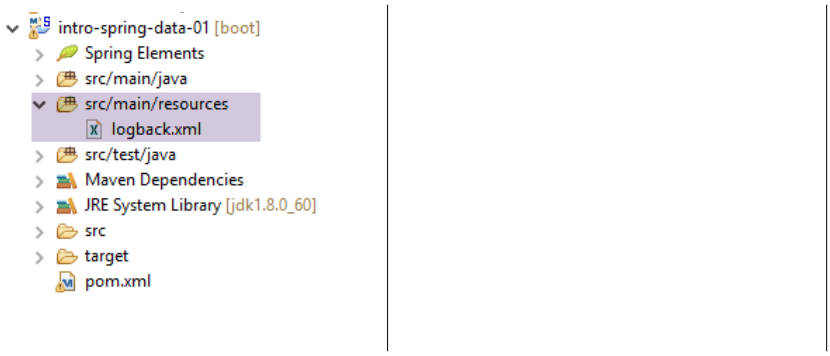
```
157.  static private <T> void affiche(List<T> elements, ObjectMapper jsonMapper) throws JsonProcessingException {
158.      for (T element : elements) {
159.          affiche(element, jsonMapper);
160.      }
161.  }
162.
163.  private static void log(String message, int mode) {
164.      // affiche message
165.      String toPrint = null;
166.      switch (mode) {
167.      case 1:
168.          toPrint = String.format("%s --------------------------------", message);
169.          break;
170.      case 2:
171.          toPrint = String.format("-- %s", message);
172.          break;
173.      }
174.      System.out.println(toPrint);
175.  }
176.
177.  private static void show(String title, List<String> messages) {
178.      // titre
179.      System.out.println(String.format("%s : ", title));
180.      // messages
181.      for (String message : messages) {
182.          System.out.println(String.format("- %s", message));
183.      }
184.  }
185.
186. }
```

- ligne 27 : le test unitaire est configuré par la classe [AppConfig] déjà présentée au paragraphe 11.3.8, page 196 ;
- lignes 32-33 : injection d'une référence sur la couche [DAO] ;
- lignes 36-50 : injection des cinq mappeurs jSON ;
- lignes 60-71 : après avoir vidé la base (ligne 57), on remplit la base de données avec 2 catégories contenant chacune 5 produits. Cette méthode est exécutée avant chaque test à cause de l'annotation [@Before] de la ligne 52 ;
- lignes 75-93 : affiche le contenu de la base ;
- lignes 95-101 : demande une catégorie avec ses produits, catégorie identifiée par son nom ;
- lignes 103-109 : demande une catégorie sans ses produits, catégorie identifiée par son nom ;
- lignes 111-120 : demande un produit avec sa catégorie, produit identifié par son n° ;
- lignes 122-130 : demande un produit sans sa catégorie, produit identifié par son n° ;
- lignes 133-184 : des méthodes privées partagées par les différents tests ;

Travail à faire : exécutez le test. Il doit réussir.

11.3.11 Gestion des logs

Les logs de l'application console ou du test JUnit sont configurés par le fichier [logback.xml] suivant :



Le fichier doit s'appeler [logback.xml] et être dans le Classpath du projet. Pour cela, il a été placé ici dans le dossier [src/main/resources] qui fait partie du Classpath. Son contenu est le suivant :

```
1. <configuration>
2.
3.   <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
4.     <!-- encoders are  by default assigned the type
5.          ch.qos.logback.classic.encoder.PatternLayoutEncoder -->
6.     <encoder>
```

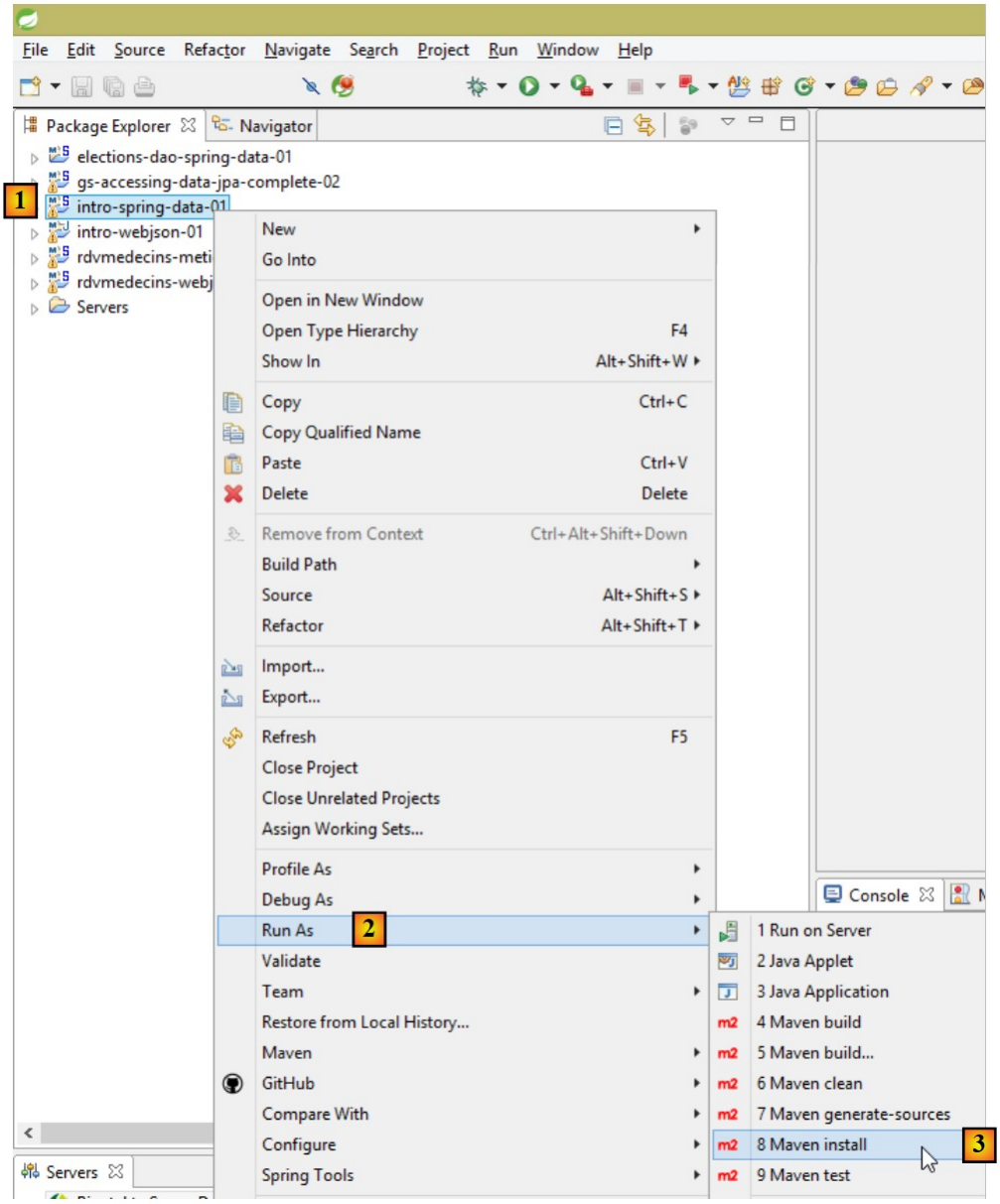
```
7.      <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>
8.    </encoder>
9.  </appender>
10.
11.   <!-- contrôle niveau des logs -->
12.   <root level="info"> <!-- info, debug, warn -->
13.     <appender-ref ref="STDOUT" />
14.   </root>
15. </configuration>
```

- ligne 12 : la balise [<root level="info">] affiche les logs de niveau [info]. A la place de [info], on peut mettre :
  - [debug] : c'est le niveau le plus détaillé de logs. Il est conseillé de l'utiliser pendant la phase de débogage du projet car il y a des logs très intéressants sur les échanges client / serveur. C'est une façon de comprendre ce qui se passe 'sous le capot' ;
  - [off] : pas de logs du tout ;
  - [warn] : un niveau de logs intermédiaire où Spring affiche des anomalies qui ne pas pour autant des erreurs. Il faut les regarder si on n'obtient pas le résultat escompté ;

Travail à faire : passez le niveau de la ligne 12 à [debug] puis exécutez le test unitaire. Regardez la différence de logs.

11.3.12 Génération de l'archive Maven du projet

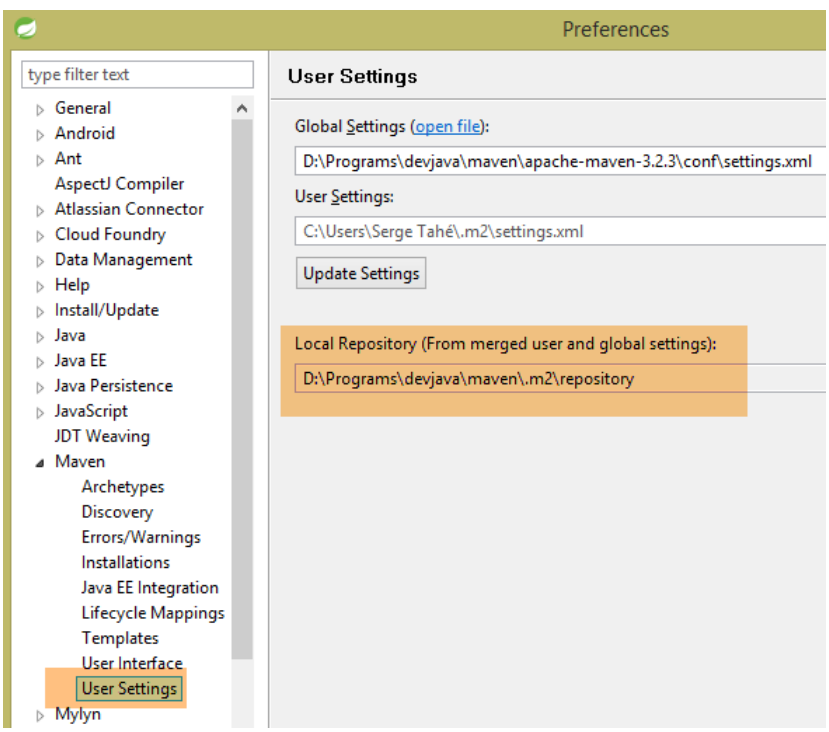
Pour installer l'archive du projet dans le dépôt local Maven, procédez comme suit [1-3] :



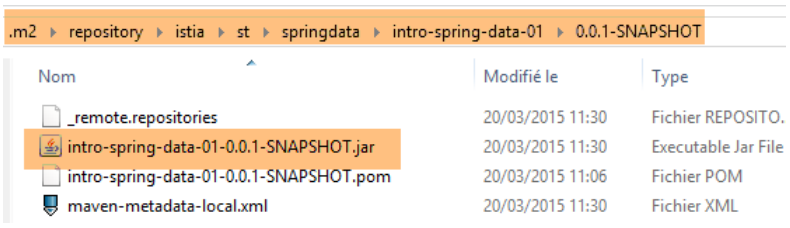
L'archive va être générée avec les identifiants trouvés dans le fichier [pom.xml] :

```
1.  <groupId>istia.st.springdata</groupId>
2.  <artifactId>intro-spring-data-01</artifactId>
3.  <version>0.0.1-SNAPSHOT</version>
4.  <packaging>jar</packaging>
```

La localisation du dépôt local Maven peut être trouvé dans la configuration d'Eclipse :



Il est alors possible de vérifier la bonne installation de l'artifact Maven :



Désormais, un autre projet Maven local pourra utiliser cette archive.

12 [TD] : Implémentation de la couche [DAO] du TD avec [Spring Data]

Mots clés : architecture multicouche, Spring, injection de dépendances, API JPA (Java Persistence API), Spring Data.

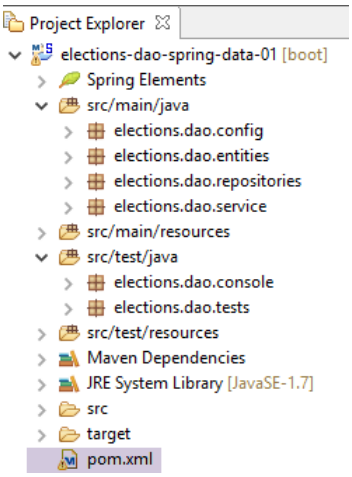
Nous allons suivre la même démarche que précédemment pour implémenter la couche [DAO] du TD.

12.1 Support

- en [1], le dossier [support / chap-12] contient le projet Eclipse de ce chapitre ;

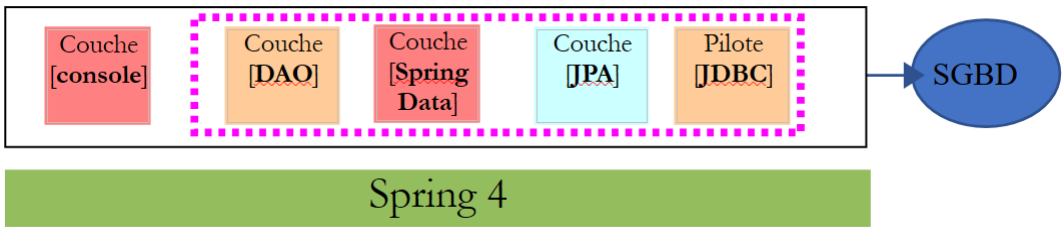
12.2 Le projet Eclipse

Le projet Eclipse sera le suivant :



- [elections.dao.config] : contient la classe de configuration du projet Spring ;
- [elections.dao.entities] : contient les entités JPA ainsi que la classe d'exception du projet ;
- [elections.dao.repositories] : contient les interfaces [CrudRepository] pour les tables [CONF] et [LISTES] ;
- [elections.dao.service] : contient l'implémentation de la couche [DAO]. C'est elle qu'il nous faut écrire ;
- [elections.dao.console] : contient une classe de test de type [console] ;
- [pom.xml] : le fichier de configuration du projet Maven ;

Ce projet implémente l'architecture suivante :

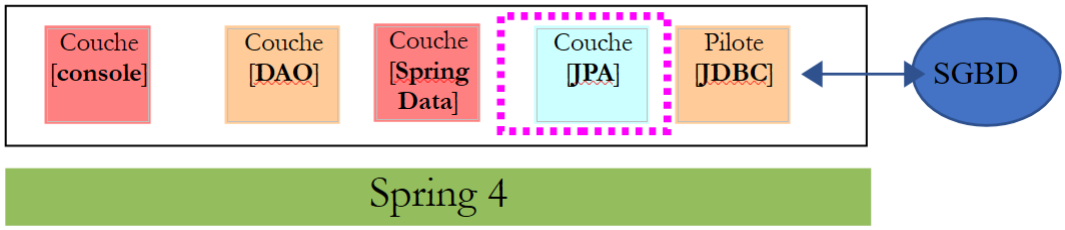


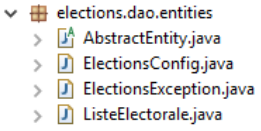
La couche [DAO] ne voit que la couche implémentée par [Spring Data].

12.3 Configuration Maven

Travail à faire : construisez le fichier [pom.xml] du projet.

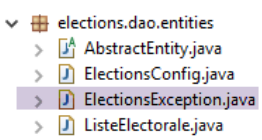
12.4 Les entités de la couche [JPA]





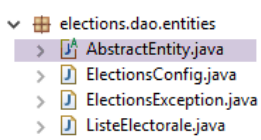
- [ElectionsConfig] est le modèle objet associé à une ligne de la table [CONF] ;
- [ListeElectorale] est le modèle objet associé à une ligne de la table [LISTES] ;
- [AbstactEntity] est la classe parent des deux classes précédentes. Elle factorise les champs [id, version] communs aux deux classes ;
- [ElectionsException] est la classe d'exception du projet ;

12.4.1 La classe [ElectionsException]



La classe [ElectionsException] est l'exception non contrôlée décrite au paragraphe 4.3, page 39.

12.4.2 La classe [AbstractEntity]

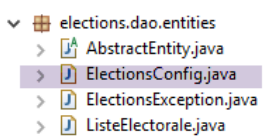


La classe [AbstractEntity] est la suivante :

```
1. package elections.dao.entities;
2.
3. import java.io.Serializable;
4.
5. import javax.persistence.Column;
6. import javax.persistence.GeneratedValue;
7. import javax.persistence.GenerationType;
8. import javax.persistence.Id;
9. import javax.persistence.MappedSuperclass;
10.  import javax.persistence.Version;
11.
12.  import com.fasterxml.jackson.core.JsonProcessingException;
13.  import com.fasterxml.jackson.databind.ObjectMapper;
14.
15.  @MappedSuperclass
16.  public abstract class AbstractEntity implements Serializable{
17.    private static final long serialVersionUID = 1L;
18.
19.    // propriétés
20.    @Id
21.    @GeneratedValue(strategy = GenerationType.IDENTITY)
22.    @Column(name = "ID")
23.    protected Long id;
24.    @Version
25.    @Column(name = "VERSION")
26.    protected Long version;
27.
28.    // constructeurs
29.    public AbstractEntity() {
30.
31.    }
32.
33.    public AbstractEntity(Long id, Long version) {
34.        this.id = id;
35.        this.version = version;
36.    }
37.
38.    // redéfinition [equals] et [hashcode]
39.    @Override
40.    public int hashCode() {
41.        return (id != null ? id.hashCode() : 0);
42.    }
43.
44.    @Override
45.    public boolean equals(Object entity) {
46.        if (!(entity instanceof AbstractEntity)) {
47.            return false;
48.        }
49.        String class1 = this.getClass().getName();
50.        String class2 = entity.getClass().getName();
51.        if (!class2.equals(class1)) {
52.            return false;
53.        }
54.        AbstractEntity other = (AbstractEntity) entity;
55.        return id != null && this.id == other.id;
56.    }
57.
58.    // signature jSON
59.    public String toString() {
60.        ObjectMapper mapper = new ObjectMapper();
61.        try {
62.            return mapper.writeValueAsString(this);
63.        } catch (JsonProcessingException e) {
64.            e.printStackTrace();
65.            return null;
66.        }
67.    }
68.
69.    // getters et setters
70.  ...
71.  }
```

C'est la classe décrite au paragraphe 11.3.5.2, page 188, sans les filtres jSON. Ici en effet, les tables [CONF] et [LISTES] ne sont pas liées par une relation de clé étrangère. Or c'est l'existence d'une telle relation avec le mode [lazy loading] qui induit la nécessité de filtres jSON.

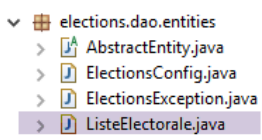
12.4.3 La classe [ElectionsConfig]



La classe [ElectionsConfig] est l'entité JPA associée à la table [CONF] ;

Travail à faire : construisez la classe [ElectionsConfig].

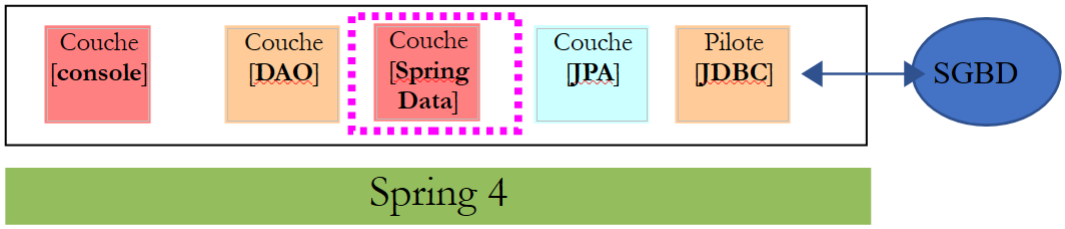
12.4.4 La classe [ListeElectorale]

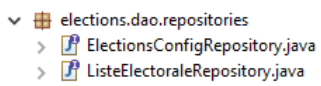


La classe [ListeElectorale] est l'entité JPA associée à la table [LISTES] ;

Travail à faire : construisez la classe [ListeElectorale].

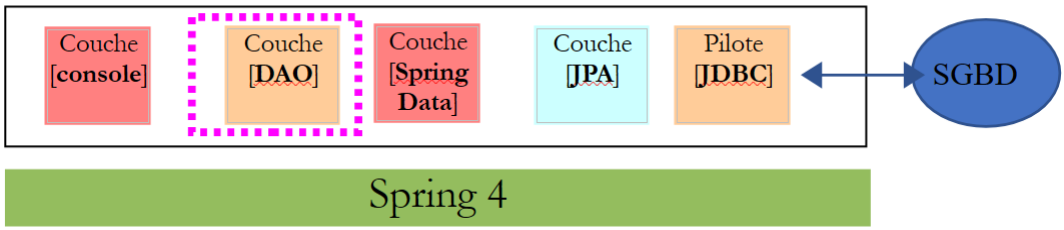
12.5 La couche [Spring Data]

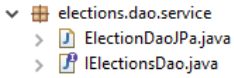




Travail à faire : écrivez les deux interfaces de [Spring Data] pour gérer les deux tables [CONF] et [LISTES] ;

12.6 La couche [DAO]



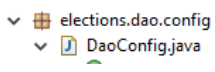


L'interface [IElectionsDao] de la couche [DAO] est la suivante :

```
1. package dao.service;
2.
3. import dao.entities.ElectionsConfig;
4. import dao.entities.ListeElectorale;
5.
6. public interface IElectionsDao {
7.
8.   // configuration de l'élection
9.   public ElectionsConfig getElectionsConfig();
10.
11.    // listes candidates
12.    public ListeElectorale[] getListesElectorales();
13.
14.    // mise à jour des listes candidates
15.    public void setListesElectorales(ListeElectorale[] listesElectorales);
16.  }
```

Travail à faire : écrivez l'implémentation [ElectionsDaoJpa] de l'interface [IElectionsDao].

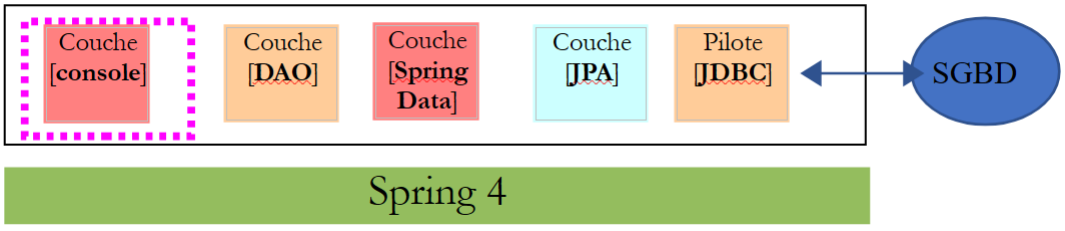
12.7 Configuration du projet Spring

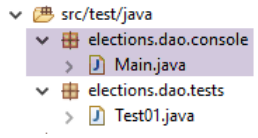


La classe [DaoConfig] configure le projet Spring.

Travail à faire : écrivez la classe [DaoConfig]

12.8 La couche [console]





La classe [Main] est la classe exécutable suivante :

```
1. package dao.console;
2.
3. import org.springframework.context.annotation.AnnotationConfigApplicationContext;
4.
5. import dao.config.AppConfig;
6. import dao.entities.ElectionsConfig;
7. import dao.entities.ListeElectorale;
8. import dao.service.IElectionsDao;
9.
10.  public class Main {
11.
12.    // source de données
13.    private static IElectionsDao dao;
14.
15.    public static void main(String[] args) {
16.        // récupération du contexte Spring
17.        AnnotationConfigApplicationContext ctx = new AnnotationConfigApplicationContext(AppConfig.class);
18.        // récupération de la source de données
19.        dao = ctx.getBean(IElectionsDao.class);
20.        // contenu des deux tables
21.        ElectionsConfig electionsConfig = dao.getElectionsConfig();
22.        ListeElectorale[] listes = dao.getListesElectorales();
23.        // affichage
24.        System.out.println(String.format("Nombre de sièges à pourvoir : %d", electionsConfig.getNbSiegesAPourvoir()));
25.        System.out.println(String.format("Seuil électoral : %5.2f", electionsConfig.getSeuilElectoral()));
26.        System.out.println("Listes candidates----------------");
27.        for (ListeElectorale liste : listes) {
28.            System.out.println(liste);
29.        }
30.        // fermeture contexte Spring
31.        ctx.close();
32.    }
33.
34.  }
```

La classe [Test01] est un test JUnit :

```
1. package dao.tests;
2.
3. import org.junit.Assert;
4. import org.junit.Before;
5. import org.junit.Test;
6. import org.junit.runner.RunWith;
7. import org.springframework.beans.factory.annotation.Autowired;
8. import org.springframework.boot.test.SpringApplicationConfiguration;
9. import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
10.
11.  import dao.config.AppConfig;
12.  import dao.entities.ElectionsConfig;
13.  import dao.entities.ListeElectorale;
14.  import dao.service.IElectionsDao;
15.
16.  @SpringApplicationConfiguration(classes = AppConfig.class)
17.  @RunWith(SpringJUnit4ClassRunner.class)
18.  public class Test01 {
19.
20.    // couche [DAO]
21.    @Autowired
22.    private IElectionsDao electionsDao;
23.
24.    @Before
25.    public void init() {
26.        // on nettoie la table [LISTES]
27.        // listes en compétition
28.        ListeElectorale[] listes = electionsDao.getListesElectorales();
29.        // on met à 0 les voix et sièges et elimine à false
30.        int voix = 0;
```

```
31.         int sièges = 0;
32.         boolean elimine = false;
33.         for (ListeElectorale liste : listes) {
34.             liste.setVoix(voix);
35.             liste.setSieges(sièges);
36.             liste.setElimine(elimine);
37.         }
38.         // on rend ces données persistantes grâce à la couche [dao]
39.         electionsDao.setListesElectorales(listes);
40.     }
41.
42.     @Test
43.     public void testElections01() {
44.         System.out.println("testElections01-----------------------------------");
45.         // récupération de la configuration des élections
46.         ElectionsConfig electionsConfig = electionsDao.getElectionsConfig();
47.         int nbSiegesAPourvoir = electionsConfig.getNbSiegesAPourvoir();
48.         double seuilElectoral = electionsConfig.getSeuilElectoral();
49.         Assert.assertEquals(6, nbSiegesAPourvoir);
50.         Assert.assertEquals(0.05, seuilElectoral, 1E-6);
51.
52.         // listes en compétition
53.         ListeElectorale[] listes = electionsDao.getListesElectorales();
54.         // affichage valeurs lues
55.         System.out.println("Nombre de sièges à pourvoir : " + nbSiegesAPourvoir);
56.         System.out.println("Seuil électoral : " + seuilElectoral);
57.         System.out.println("Listes en compétition ---------------------");
58.         for (int i = 0; i < listes.length; i++) {
59.             System.out.println(listes[i]);
60.         }
61.
62.         // on affecte des voix et des sièges aux listes
63.         int voix = 0;
64.         int sièges = 0;
65.         boolean elimine = false;
66.         for (ListeElectorale liste : listes) {
67.             liste.setVoix(voix);
68.             liste.setSieges(sièges);
69.             liste.setElimine(elimine);
70.             voix += 10;
71.             sièges += 1;
72.             elimine = !elimine;
73.         }
74.
75.         // on rend ces données persistantes grâce à la couche [dao]
76.         electionsDao.setListesElectorales(listes);
77.
78.         // on relit les données
79.         ListeElectorale[] listesElectorales2 = electionsDao.getListesElectorales();
80.         // on vérifie les données lues
81.         Assert.assertEquals(7, listesElectorales2.length);
82.         voix = 0;
83.         sièges = 0;
84.         elimine = false;
85.         for (ListeElectorale liste : listes) {
86.             Assert.assertEquals(voix, liste.getVoix());
87.             Assert.assertEquals(sièges, liste.getSieges());
88.             Assert.assertEquals(elimine, liste.isElimine());
89.             voix += 10;
90.             sièges += 1;
91.             elimine = !elimine;
92.         }
93.         System.out.println("Listes en compétition ---------------------");
94.         for (int i = 0; i < listes.length; i++) {
95.             System.out.println(listes[i]);
96.         }
97.     }
98. }
```

Travail à faire : passez les tests [console] et [JUnit] sur votre couche [DAO].

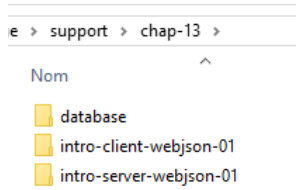
12.9 Génération de l'archive Maven du projet

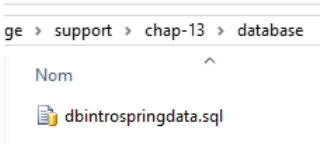
En suivant l'exemple du paragraphe 11.3.12, page 205, générez l'archive Maven du projet.

13 [Cours] : Exposer une base de données sur le web avec Spring MVC

Mots clés : architecture multicouche, Spring, injection de dépendances, service web / jSON, client / serveur

13.1 Support

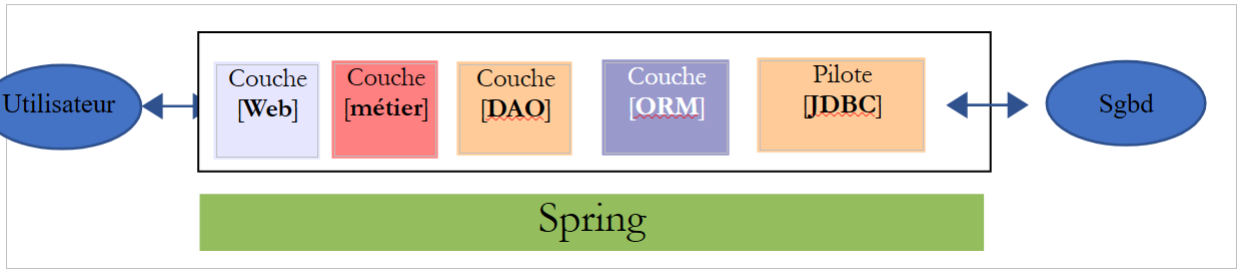




Les projets de ce chapitre seront trouvés dans le dossier [support / chap-13]. Le script SQL [dbintrospringdata.sql] permet de créer la base MySQL nécessaire aux tests.

13.2 La place de Spring MVC dans une application Web

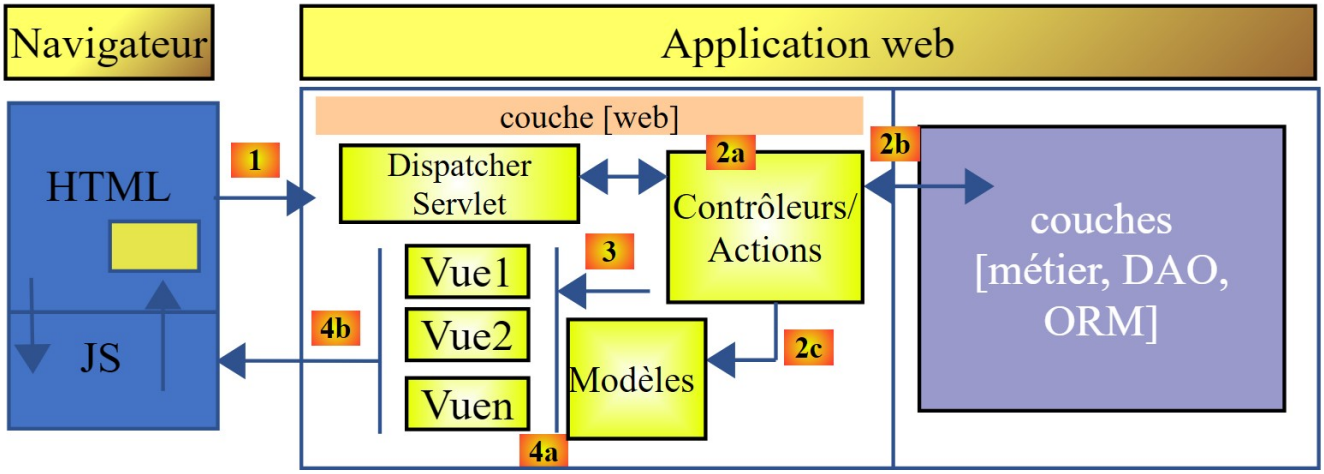
Situons Spring MVC dans le développement d'une application Web. Le plus souvent, celle-ci sera bâtie sur une architecture multicouche telle que la suivante :



- la couche [**Web**] est la couche en contact avec l'utilisateur de l'application Web. Celui-ci interagit avec l'application Web au travers de pages Web visualisées par un navigateur. **C'est dans cette couche que se situe Spring MVC et uniquement dans cette couche ;**
- la couche [**métier**] implémente les règles de gestion de l'application, tels que le calcul d'un salaire ou d'une facture. Cette couche utilise des données provenant de l'utilisateur via la couche [Web] et du SGBD via la couche [DAO] ;
- la couche [**DAO**] (Data Access Objects), la couche [**ORM**] (Object Relational Mapper) et le pilote JDBC gèrent l'accès aux données du SGBD. La couche [**ORM**] fait un pont entre les objets manipulés par la couche [DAO] et les lignes et les colonnes des tables d'une base de données relationnelle. La spécification JPA (Java Persistence API) permet de s'abstraire de l'ORM utilisé si celui-ci implémente ces spécifications. Ce sera le cas ici et nous appellerons désormais la couche ORM, la couche JPA ;
- l'intégration des couches est faite par le framework Spring ;

13.3 Le modèle de développement de Spring MVC

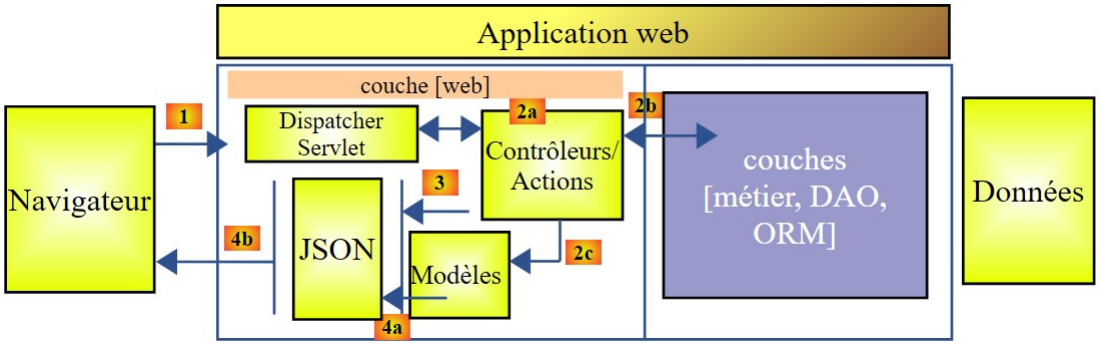
Spring MVC implémente le modèle d'architecture dit MVC (Modèle – Vue – Contrôleur) de la façon suivante :



Le traitement d'une demande d'un client se déroule de la façon suivante :

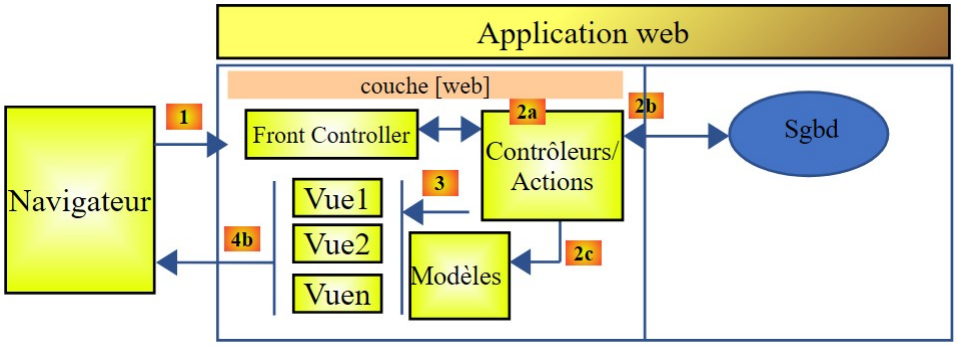
1. **demande** - les URL demandées sont de la forme *http://machine:port/contexte/Action/param1/param2/....?p1=v1&p2=v2&...* Le [Front Controller] utilise un fichier de configuration ou des annotations Java pour " router " la demande vers le bon contrôleur et la bonne action au sein de ce contrôleur. Pour cela, il utilise le champ [Action] de l'URL. Le reste de l'URL [*/param1/param2/...*] est formé de paramètres facultatifs qui seront transmis à l'action. Le **C** de MVC est ici la chaîne [Front Controller, Contrôleur, Action]. Si aucun contrôleur ne peut traiter l'action demandée, le serveur Web répondra que l'URL demandée n'a pas été trouvée.
2. **traitement**
    - l'action choisie peut exploiter les paramètres *paramI* que le [Front Controller] lui a transmis. Ceux-ci peuvent provenir de plusieurs sources :
        - du chemin [*/param1/param2/...*] de l'URL,
        - des paramètres [*p1=v1&p2=v2*] de l'URL,
        - de paramètres postés par le navigateur avec sa demande ;
    - dans le traitement de la demande de l'utilisateur, l'action peut avoir besoin de la couche [métier] [2b]. Une fois la demande du client traitée, celle-ci peut appeler diverses réponses. Un exemple classique est :
        - une page d'erreur si la demande n'a pu être traitée correctement
        - une page de confirmation sinon
    - l'action demande à une certaine vue de s'afficher [3]. Cette vue va afficher des données qu'on appelle le **modèle de la vue**. C'est le **M** de MVC. L'action va créer ce modèle M [2c] et demander à une vue **V** de s'afficher [3] ;
3. **réponse** - la vue **V** choisie utilise le modèle **M** construit par l'action pour initialiser les parties dynamiques de la réponse HTML qu'elle doit envoyer au client puis envoie cette réponse.

Pour un service web / jSON, l'architecture précédente est légèrement modifiée :



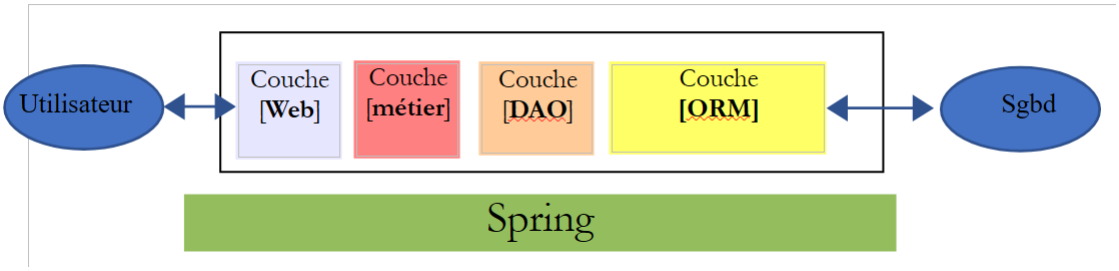
- en [4a], le modèle qui est une classe Java est transformé en chaîne jSON par une bibliothèque jSON ;
- en [4b], cette chaîne jSON est envoyée au navigateur ;

Maintenant, précisons le lien entre architecture web MVC et architecture en couches. Selon la définition qu'on donne au **modèle**, ces deux concepts sont liés ou non. Prenons une application web Spring MVC à une couche :



Si nous implémentons la couche [Web] avec Spring MVC, nous aurons bien une architecture web MVC mais pas une architecture multicouche. Ici, la couche [web] s'occupera de tout : présentation, métier, accès aux données. Ce sont les actions qui feront ce travail.

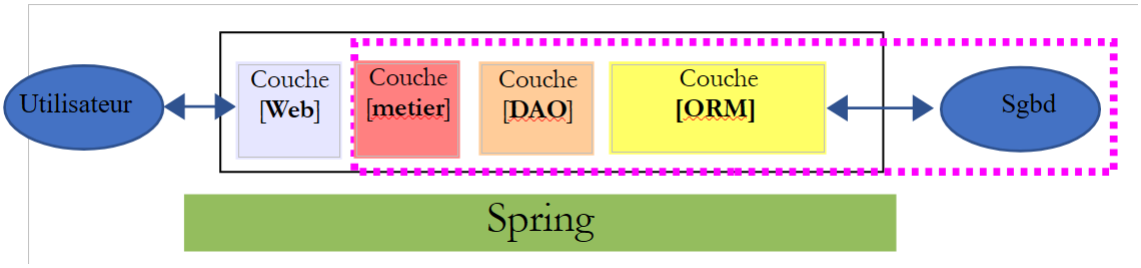
Maintenant, considérons une architecture Web multicouche :



La couche [Web] peut être implémentée sans framework et sans suivre le modèle MVC. On a bien alors une architecture multicouche mais la couche Web n'implémente pas le modèle MVC.

Par exemple, dans le monde .NET la couche [Web] ci-dessus peut être implémentée avec ASP.NET MVC et on a alors une architecture en couches avec une couche [Web] de type MVC. Ceci fait, on peut remplacer cette couche ASP.NET MVC par une couche ASP.NET classique (WebForms) tout en gardant le reste (métier, DAO, ORM) **à l'identique**. On a alors une architecture en couches avec une couche [Web] qui n'est plus de type MVC.

Dans MVC, nous avons dit que le modèle M était celui de la vue V, c.a.d. l'ensemble des données affichées par la vue V. Une autre définition du modèle M de MVC est donnée :



Beaucoup d'auteurs considèrent que ce qui est à droite de la couche [Web] forme le modèle M du MVC. Pour éviter les ambigüités on peut parler :

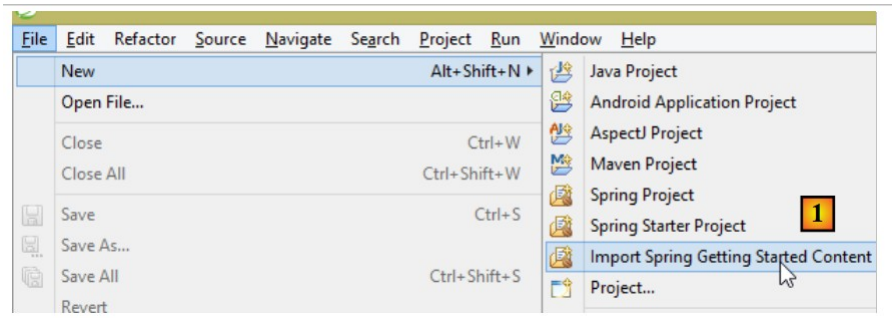
- du **modèle du domaine** lorsqu'on désigne tout ce qui est à droite de la couche [Web]
- du **modèle de la vue** lorsqu'on désigne les données affichées par une vue V

Dans la suite, le terme " modèle M " désignera exclusivement le **modèle d'une vue** V.

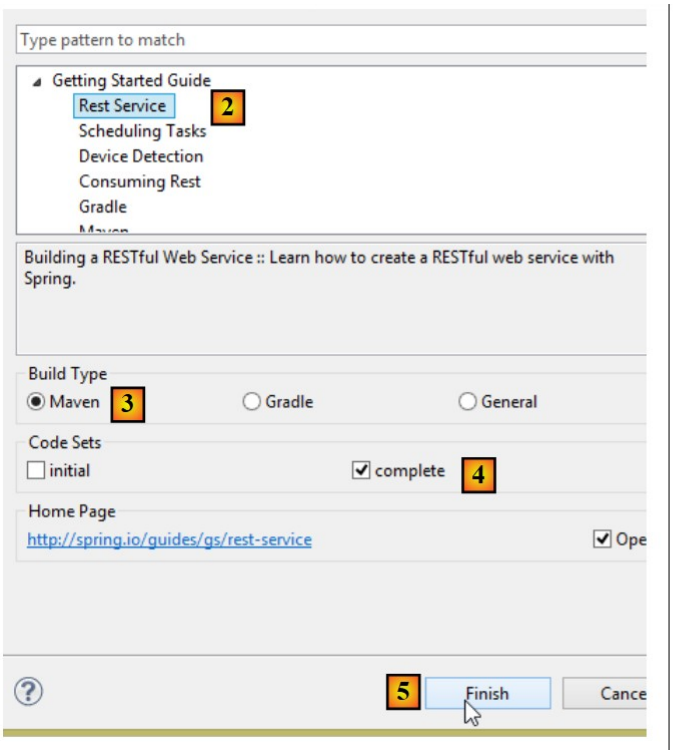
13.4 Un projet web / jSON avec Spring MVC

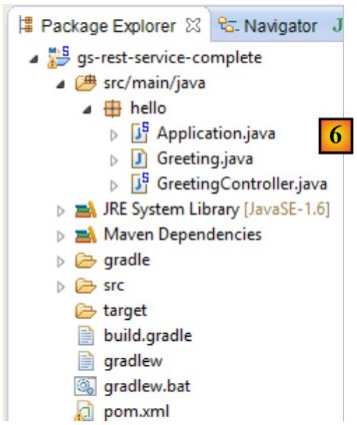
Le site [http://spring.io/guides] offre des tutoriels de démarrage pour découvrir l'écosystème Spring. Nous allons suivre l'un d'eux pour découvrir la configuration Maven nécessaire à un projet Spring MVC.

13.4.1 Le projet de démonstration



- en [1], nous importons l'un des guides Spring ;





- en [2], nous choisissons l'exemple [Rest Service] ;
- en [3], on choisit le projet Maven ;
- en [4], on prend la version finale du guide ;
- en [5], on valide ;
- en [6], le projet importé ;

Les services web accessibles via des URL standard et qui délivrent du texte jSON sont souvent appelés des services REST (REpresentational State Transfer). Un service est dit Restful s'il respecte certaines règles.

Examinons maintenant le projet importé, d'abord sa configuration Maven.

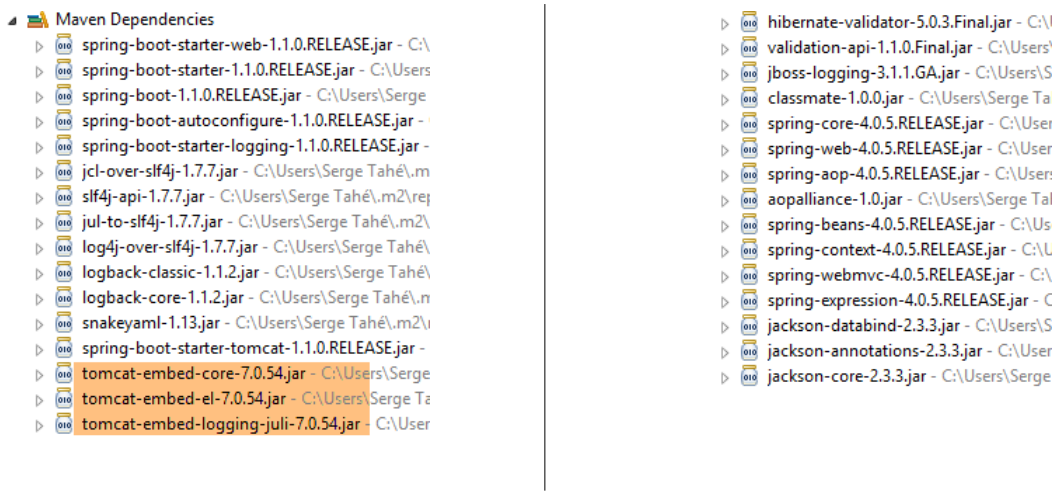
13.4.2 Configuration Maven

Le fichier [pom.xml] est le suivant :

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3.   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
4.   <modelVersion>4.0.0</modelVersion>
5.
6.   <groupId>org.springframework</groupId>
7.   <artifactId>gs-rest-service</artifactId>
8.   <version>0.1.0</version>
9.
10.    <parent>
11.        <groupId>org.springframework.boot</groupId>
12.        <artifactId>spring-boot-starter-parent</artifactId>
13.        <version>1.2.2.RELEASE</version>
14.    </parent>
15.
16.    <dependencies>
17.        <dependency>
18.            <groupId>org.springframework.boot</groupId>
19.            <artifactId>spring-boot-starter-web</artifactId>
20.        </dependency>
21.    </dependencies>
22.
23.    <properties>
24.        <start-class>hello.Application</start-class>
25.    </properties>
26.
27.    <build>
28.        <plugins>
29.            <plugin>
30.                <groupId>org.springframework.boot</groupId>
31.                <artifactId>spring-boot-maven-plugin</artifactId>
32.            </plugin>
33.        </plugins>
34.    </build>
35.
36.    <repositories>
37.        <repository>
38.            <id>spring-releases</id>
39.            <url>https://repo.spring.io/libs-release</url>
40.        </repository>
41.    </repositories>
42.    <pluginRepositories>
43.        <pluginRepository>
44.            <id>spring-releases</id>
45.            <url>https://repo.spring.io/libs-release</url>
46.        </pluginRepository>
47.    </pluginRepositories>
48. </project>
```

- lignes 6-8 : les propriétés du projet Maven. Manque une balise [<packaging>] indiquant le type du fichier produit par la compilation Maven. En l'absence de celle-ci, c'est le type [jar] qui est utilisé. L'application est donc une application exécutable de type console, et non une application web où le packaging serait alors [war] ;
- lignes 10-14 : le projet Maven a un projet parent [spring-boot-starter-parent]. C'est lui qui définit l'essentiel des dépendances du projet. Elles peuvent être suffisantes, auquel cas on n'en rajoute pas, ou pas, auquel cas on rajoute les dépendances manquantes ;
- lignes 17-20 : l'artifact [spring-boot-starter-web] amène avec lui les bibliothèques nécessaires à un projet Spring MVC de type service web où il n'y a pas de vues générées. Cet artifact amène avec lui un très grand nombre de bibliothèques dont celles d'un serveur Tomcat embarqué. C'est sur ce serveur que l'application sera exécutée ;

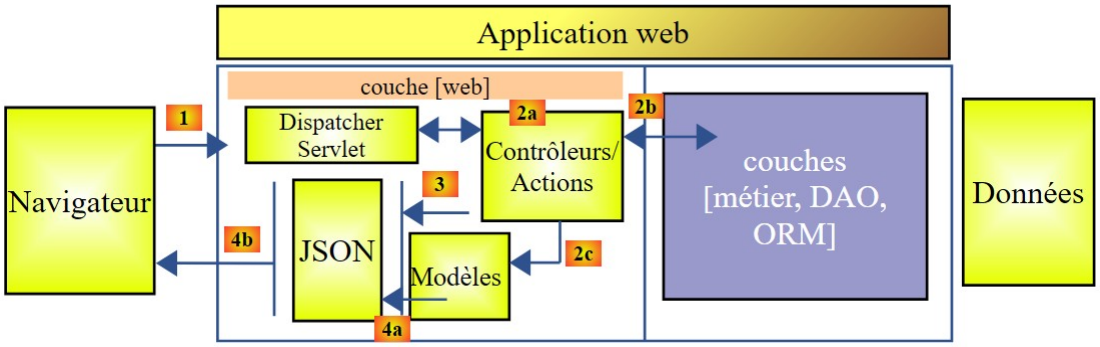
Les bibliothèques amenées par cette configuration sont très nombreuses :



Ci-dessus on voit les trois archives du serveur Tomcat.

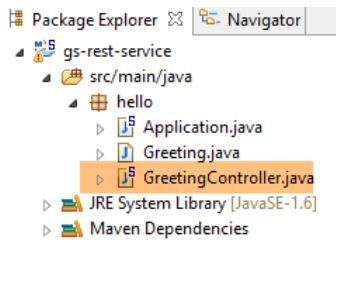
13.4.3 L'architecture d'un service Spring [web / jSON]

Pour un service web / jSON, Spring MVC implémente le modèle MVC de la façon suivante :



- en [4a], le modèle qui est une classe Java est transformé en chaîne jSON par une bibliothèque jSON ;
- en [4b], cette chaîne jSON est envoyée au navigateur ;

13.4.4 Le contrôleur C



L'application importée a le contrôleur suivant :

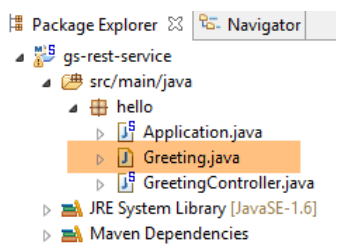
```
1. package hello;
2.
3. import java.util.concurrent.atomic.AtomicLong;
4. import org.springframework.web.bind.annotation.RequestMapping;
5. import org.springframework.web.bind.annotation.RequestParam;
6. import org.springframework.web.bind.annotation.RestController;
7.
8. @RestController
```

```
9. public class GreetingController {
10.
11.    private static final String template = "Hello, %s!";
12.    private final AtomicLong counter = new AtomicLong();
13.
14.    @RequestMapping("/greeting")
15.    public Greeting greeting(@RequestParam(value = "name", defaultValue = "World") String name) {
16.        return new Greeting(counter.incrementAndGet(), String.format(template, name));
17.    }
18. }
```

- ligne 9 : l'annotation [**@RestController**] fait de la classe [GreetingController] un contrôleur Spring, ç-à-d que ses méthodes sont enregistrées pour traiter des URL. Nous avons vu l'annotation similaire [**@Controller**]. Le résultat des méthodes de ce contrôleur était un type [String] qui était le nom de la vue à afficher. Ici c'est différent. Les méthodes d'un contrôleur de type [**@RestController**] rendent des objets qui sont sérialisés pour être envoyés au navigateur. Le type de sérialisation opérée dépend de la configuration de Spring MVC. Ici, ils seront sérialisés en jSON. C'est la présence d'une bibliothèque jSON dans les dépendances du projet qui fait que Spring Boot va, par autoconfiguration, configurer le projet de cette façon ;
- ligne 14 : l'annotation [**@RequestMapping**] indique l'URL que traite la méthode, ici l'URL [/greeting] ;
- ligne 15 : nous avons déjà expliqué l'annotation [@RequestParam]. Le résultat rendu par la méthode est un objet de type [Greeting].
- ligne 12 : un entier long de type atomique. Cela signifie qu'il supporte la concurrence d'accès. Plusieurs threads peuvent vouloir incrémenter la variable [counter] en même temps. Cela se fera proprement. Un thread ne peut lire la valeur du compteur que si le thread en train de le modifier a terminé sa modification.

13.4.5 Le modèle M

Le modèle M produit par la méthode précédente est l'objet [Greeting] suivant :




```
1. package hello;
2.
3. public class Greeting {
4.
5.     private final long id;
6.     private final String content;
7.
8.     public Greeting(long id, String content) {
9.         this.id = id;
10.          this.content = content;
11.     }
12.
13.     public long getId() {
14.         return id;
15.     }
16.
17.     public String getContent() {
18.         return content;
19.     }
20. }
```

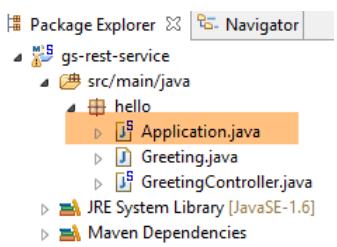
La transformation jSON de cet objet créera la chaîne de caractères **{"id":n,"content":"texte"}**. Au final, la chaîne jSON produite par la méthode du contrôleur sera de la forme :

```
{"id":2,"content":"Hello, World!"}
```

ou

```
{"id":2,"content":"Hello, John!"}
```

13.4.6 Exécution



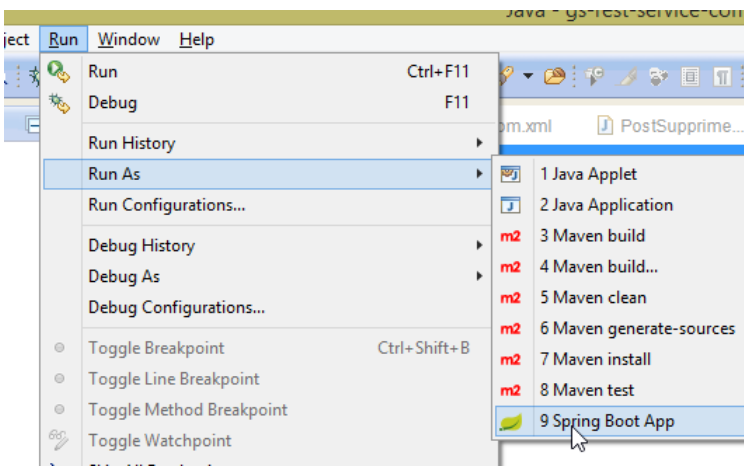
La classe [Application.java] est la classe exécutable du projet. Son code est le suivant :

```
1. package hello;
2.
3. import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
4. import org.springframework.boot.SpringApplication;
5. import org.springframework.context.annotation.ComponentScan;
6.
7. @ComponentScan
8. @EnableAutoConfiguration
9. public class Application {
10.
11.     public static void main(String[] args) {
12.         SpringApplication.run(Application.class, args);
13.     }
14.
15. }
```

Nous avons déjà rencontré et expliqué ce code dans l'exemple précédent.

13.4.7 Exécution du projet

Exécutons le projet :



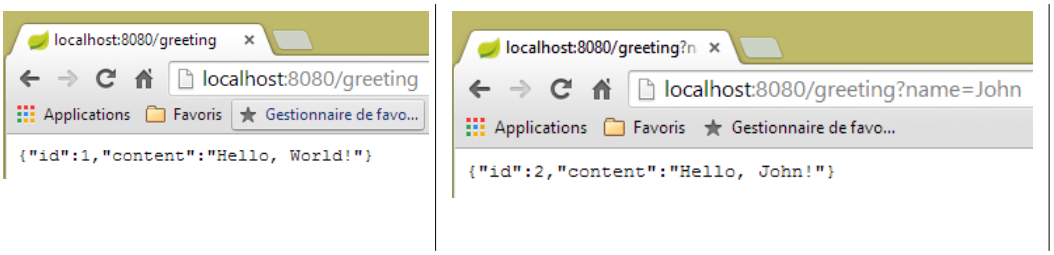
On obtient les logs console suivants :

```
1.   .   ____          _            __ _ _
2.  /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
3. ( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
4.  \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
5.   '  |____| .__|_| |_|_| |_\__, | / / / /
6.  =========|_|==============|___/=/_/_/_/
7.  :: Spring Boot ::        (v1.1.9.RELEASE)
8.
9. 2014-11-28 15:22:55.005  INFO 3152 --- [           main] hello.Application                        : Starting Application on
    Gportpers3 with PID 3152 (started by ST in D:\data\istia-1415\spring mvc\dvp-final\gs-rest-service)
```

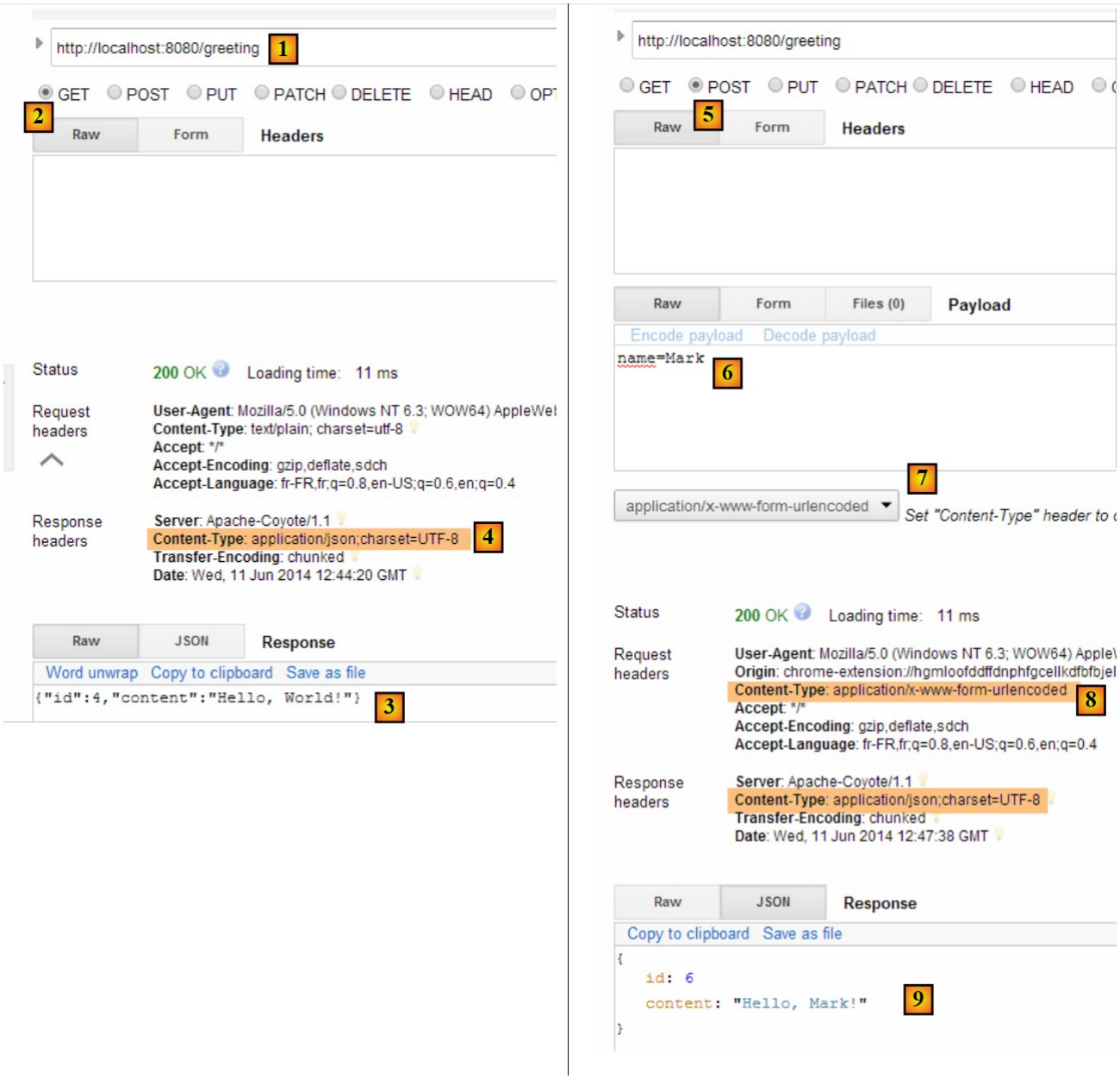
```
10. 2014-11-28 15:22:55.046  INFO 3152 --- [           main] ationConfigEmbeddedWebApplicationContext : Refreshing
    org.springframework.boot.context.embedded.AnnotationConfigEmbeddedWebApplicationContext@62e136d3: startup date [Fri Nov 28
    15:22:55 CET 2014]; root of context hierarchy
11. 2014-11-28 15:22:55.762  INFO 3152 --- [           main] o.s.b.f.s.DefaultListableBeanFactory     : Overriding bean
    definition for bean 'beanNameViewResolver': replacing [Root bean: class [null]; scope=; abstract=false; lazyInit=false;
    autowireMode=3; dependencyCheck=0; autowireCandidate=true; primary=false;
    factoryBeanName=org.springframework.boot.autoconfigure.web.ErrorMvcAutoConfiguration$WhitelabelErrorViewConfiguration;
    factoryMethodName=beanNameViewResolver; initMethodName=null; destroyMethodName=(inferred); defined in class path resource
    [org/springframework/boot/autoconfigure/web/ErrorMvcAutoConfiguration$WhitelabelErrorViewConfiguration.class]] with [Root
    bean: class [null]; scope=; abstract=false; lazyInit=false; autowireMode=3; dependencyCheck=0; autowireCandidate=true;
    primary=false;
    factoryBeanName=org.springframework.boot.autoconfigure.web.WebMvcAutoConfiguration$WebMvcAutoConfigurationAdapter;
    factoryMethodName=beanNameViewResolver; initMethodName=null; destroyMethodName=(inferred); defined in class path resource
    [org/springframework/boot/autoconfigure/web/WebMvcAutoConfiguration$WebMvcAutoConfigurationAdapter.class]]
12. 2014-11-28 15:22:56.567  INFO 3152 --- [           main] .t.TomcatEmbeddedServletContainerFactory : Server initialized with
    port: 8080
13. 2014-11-28 15:22:56.738  INFO 3152 --- [           main] o.apache.catalina.core.StandardService   : Starting service Tomcat
14. 2014-11-28 15:22:56.740  INFO 3152 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet
    Engine: Apache Tomcat/7.0.56
15. 2014-11-28 15:22:56.869  INFO 3152 --- [ost-startStop-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring
    embedded WebApplicationContext
16. 2014-11-28 15:22:56.870  INFO 3152 --- [ost-startStop-1] o.s.web.context.ContextLoader            : Root
    WebApplicationContext: initialization completed in 1827 ms
17. 2014-11-28 15:22:57.478  INFO 3152 --- [ost-startStop-1] o.s.b.c.e.ServletRegistrationBean        : Mapping servlet:
    'dispatcherServlet' to [/]
18. 2014-11-28 15:22:57.481  INFO 3152 --- [ost-startStop-1] o.s.b.c.embedded.FilterRegistrationBean  : Mapping filter:
    'hiddenHttpMethodFilter' to: [/*]
19. 2014-11-28 15:22:57.685  INFO 3152 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping  : Mapped URL path
    [/**/favicon.ico] onto handler of type [class org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
20. 2014-11-28 15:22:57.879  INFO 3152 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped
    "{[/greeting],methods=[],params=[],headers=[],consumes=[],produces=[],custom=[]}" onto public hello.Greeting
    hello.GreetingController.greeting(java.lang.String)
21. 2014-11-28 15:22:57.884  INFO 3152 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped
    "{[/error],methods=[],params=[],headers=[],consumes=[],produces=[],custom=[]}" onto public
    org.springframework.http.ResponseEntity<java.util.Map<java.lang.String, java.lang.Object>>
    org.springframework.boot.autoconfigure.web.BasicErrorController.error(javax.servlet.http.HttpServletRequest)
22. 2014-11-28 15:22:57.885  INFO 3152 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped
    "{[/error],methods=[],params=[],headers=[],consumes=[],produces=[text/html],custom=[]}" onto public
    org.springframework.web.servlet.ModelAndView
    org.springframework.boot.autoconfigure.web.BasicErrorController.errorHtml(javax.servlet.http.HttpServletRequest)
23. 2014-11-28 15:22:57.906  INFO 3152 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping  : Mapped URL path
    [/webjars/**] onto handler of type [class org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
24. 2014-11-28 15:22:57.907  INFO 3152 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping  : Mapped URL path [/**]
    onto handler of type [class org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
25. 2014-11-28 15:22:58.231  INFO 3152 --- [           main] o.s.j.e.a.AnnotationMBeanExporter        : Registering beans for
    JMX exposure on startup
26. 2014-11-28 15:22:58.318  INFO 3152 --- [           main] s.b.c.e.t.TomcatEmbeddedServletContainer : Tomcat started on
    port(s): 8080/http
27. 2014-11-28 15:22:58.319  INFO 3152 --- [           main] hello.Application                        : Started Application in
    3.788 seconds (JVM running for 4.424)
```

- ligne 13 : le serveur Tomcat démarre sur le port 8080 (ligne 12) ;
- ligne 17 : la servlet [DispatcherServlet] est présente ;
- ligne 20 : la méthode [GreetingController.greeting] a été découverte ;

Pour tester l'application web, on demande l'URL [http://localhost:8080/greeting] :



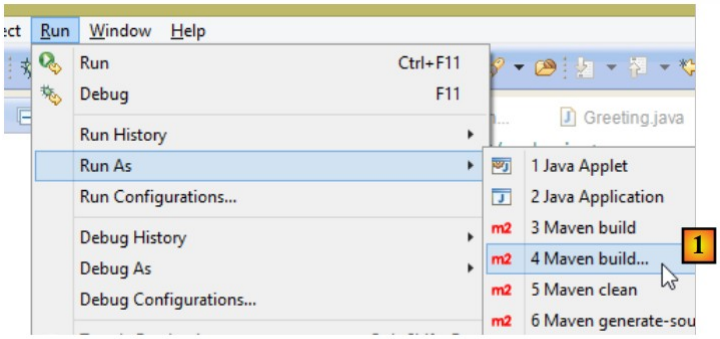
On reçoit bien la chaîne jSON attendue. Il peut être intéressant de voir les entêtes HTTP envoyés par le serveur. Pour cela, on va utiliser l'extension de Chrome appelée [Advanced Rest Client] (Chrome / Ctrl-T / Menu [Applications] / [Advanced Rest Client] - cf Annexes page 419) :

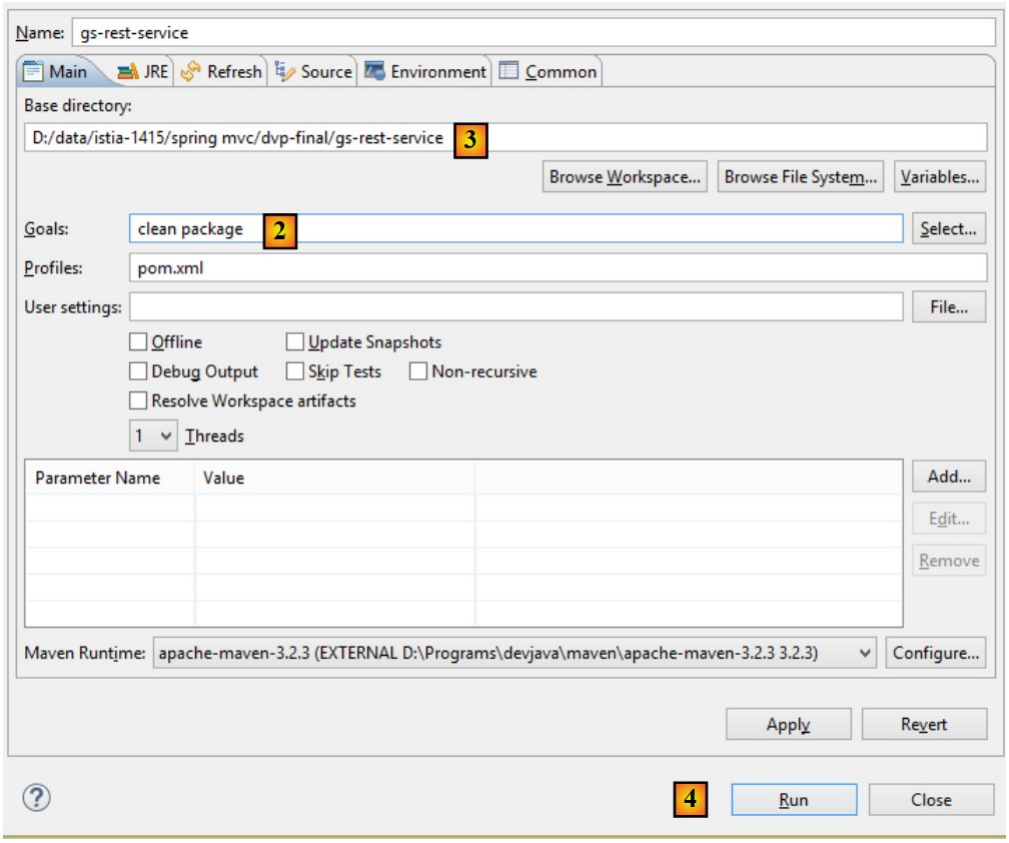


- en [1], l'URL demandée ;
- en [2], la méthode GET est utilisée ;
- en [3], la réponse jSON ;
- en [4], le serveur a indiqué qu'il envoyait une réponse au format jSON ;
- en [5], on demande la même URL mais cette fois-ci avec un POST ;
- en [7], les informations sont envoyées au serveur sous la forme [urlencoded] ;
- en [6], le paramètre **name** avec sa valeur ;
- en [8], le navigateur indique au serveur qu'il lui envoie des informations [urlencoded] ;
- en [9], la réponse jSON du serveur ;

13.4.8 Création d'une archive exécutable

Nous créons maintenant une archive exécutable :





- en [1] : on exécute une cible Maven ;
- en [2] : il y a deux cibles (goals) : [clean] pour supprimer le dossier [target] du projet Maven, [package] pour le régénérer ;
- en [3] : le dossier [target] généré, le sera dans ce dossier ;
- en [4] : on génère la cible ;

Dans les logs qui apparaissent dans la console, il est important de voir apparaître le plugin [`spring-boot-maven-plugin`]. C'est lui qui génère l'archive exécutable.

```
[INFO] --- spring-boot-maven-plugin:1.1.0.RELEASE:repackage (default) @ gs-rest-service ---
```

Avec une console, on se place dans le dossier généré :

```
1. D:\Temp\wksSTS\gs-rest-service\target>dir
2.  ...
3. 11/06/2014  15:30    <DIR>          classes
4. 11/06/2014  15:30    <DIR>          generated-sources
5. 11/06/2014  15:30        11 073 572 gs-rest-service-0.1.0.jar
6. 11/06/2014  15:30             3 690 gs-rest-service-0.1.0.jar.original
```

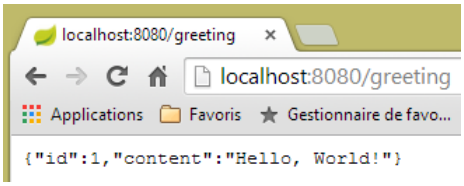
```
7. 11/06/2014  15:30    <DIR>          maven-archiver
8. 11/06/2014  15:30    <DIR>          maven-status
9. ...
```

- ligne 5 : l'archive générée ;

Cette archive est exécutée de la façon suivante :

```
1. D:\Temp\wksSTS\gs-rest-service-complete\target>java -jar gs-rest-service-0.1.0.jar
2.
3.   .   ____          _            __ _ _
4.  /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
5. ( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
6.  \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
7.   '  |____| .__|_| |_|_| |_\__, | / / / /
8.  =========|_|==============|___/=/_/_/_/
9.  :: Spring Boot ::        (v1.1.0.RELEASE)
10.
11.  2014-06-11 15:32:47.088  INFO 4972 --- [           main] hello.Application
12.                   : Starting Application on Gportpers3 with PID 4972 (D:\Temp\wk
13.  sSTS\gs-rest-service-complete\target\gs-rest-service-0.1.0.jar started by ST in
14.  D:\Temp\wksSTS\gs-rest-service-complete\target)
15.  ...
```

Maintenant que l'application web est lancée, on peut l'interroger avec un navigateur :



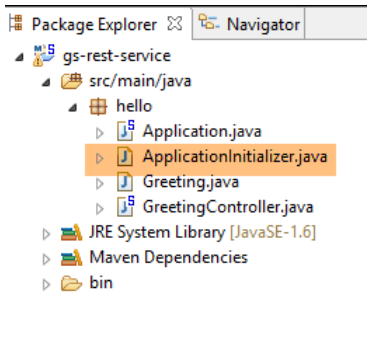
13.4.9 Déployer l'application sur un serveur Tomcat

Comme il a été fait pour le projet précédent, nous modifions le fichier [pom.xml] de la façon suivante :

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3.   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
4.   <modelVersion>4.0.0</modelVersion>
5.
6.   <groupId>org.springframework</groupId>
7.   <artifactId>gs-rest-service</artifactId>
8.   <version>0.1.0</version>
9.   <packaging>war</packaging>
10.
11.    ...
12.  </project>
```

- ligne 9 : il faut indiquer qu'on va générer une archive war (Web ARchive) ;

Il faut par ailleurs configurer l'application web. En l'absence de fichier [web.xml], cela se fait avec une classe héritant de [`SpringBootServletInitializer`] :

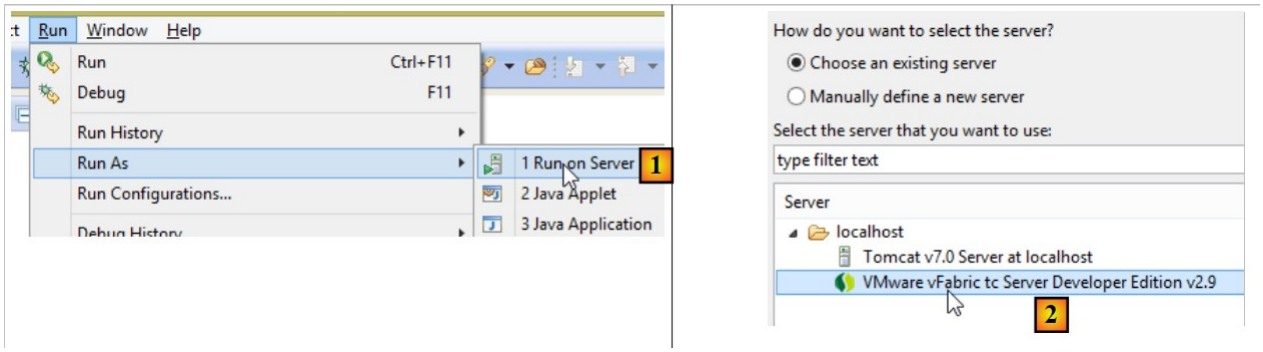


La classe [ApplicationInitializer] est la suivante :

```
1. package hello;
2.
3. import org.springframework.boot.builder.SpringApplicationBuilder;
4. import org.springframework.boot.context.web.SpringBootServletInitializer;
5.
6. public class ApplicationInitializer extends SpringBootServletInitializer {
7.
8.     @Override
9.     protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {
10.         return application.sources(Application.class);
11.     }
12.
13. }
```

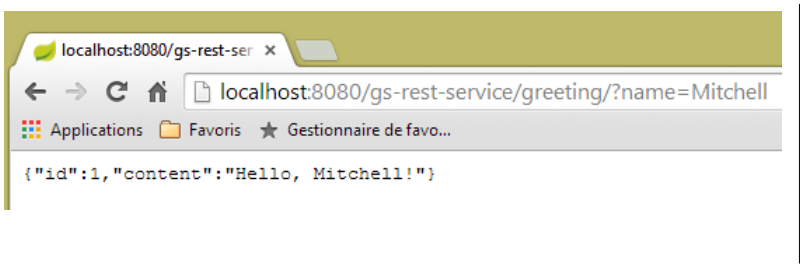
- ligne 6 : la classe [ApplicationInitializer] étend la classe [SpringBootServletInitializer] ;
- ligne 9 : la méthode [configure] est redéfinie (ligne 8) ;
- ligne 10 : on fournit la classe qui configure le projet ;

Pour exécuter le projet, on peut procéder ainsi :



- en [1-2], on exécute le projet sur l'un des serveurs enregistrés dans l'IDE Eclipse ;

Ceci fait, on peut demander l'URL [http://localhost:8080/gs-rest-service/greeting/?name=Mitchell] dans un navigateur :



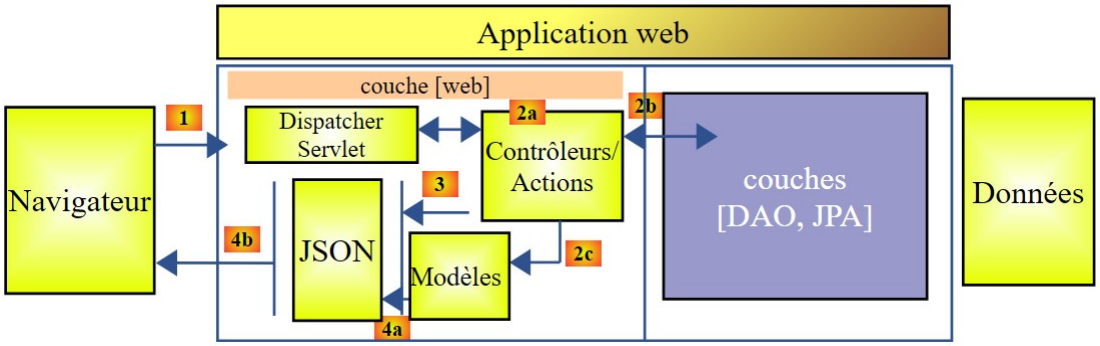
13.4.10 Conclusion

Nous avons introduit un type de projets Spring MVC où l'application web envoie un flux jSON au navigateur. Nous allons développer maintenant une application web / jSON pour exposer sur le web la base [dbintrospringdata] étudiée dans le tutoriel [Introduction à Spring Data].

13.5 Exposer la base [dbintrospringdata] sur le web

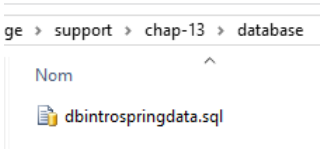
13.5.1 Architecture du service web / jSON

Nous allons mettre en place l'architecture suivante :



Les couches [DAO] et [JPA] sont implémentées par l'application écrite dans le tutoriel [Introduction à Spring Data].

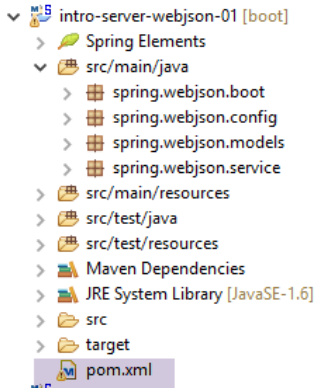
13.5.2 Installation de la base de données



Le script SQL [dbintrospringdata.sql] permet de créer la base MySQL nécessaire aux tests.

13.5.3 Le projet Eclipse du service web / jSON

Le projet Eclipse du service web / jSON est le suivant :



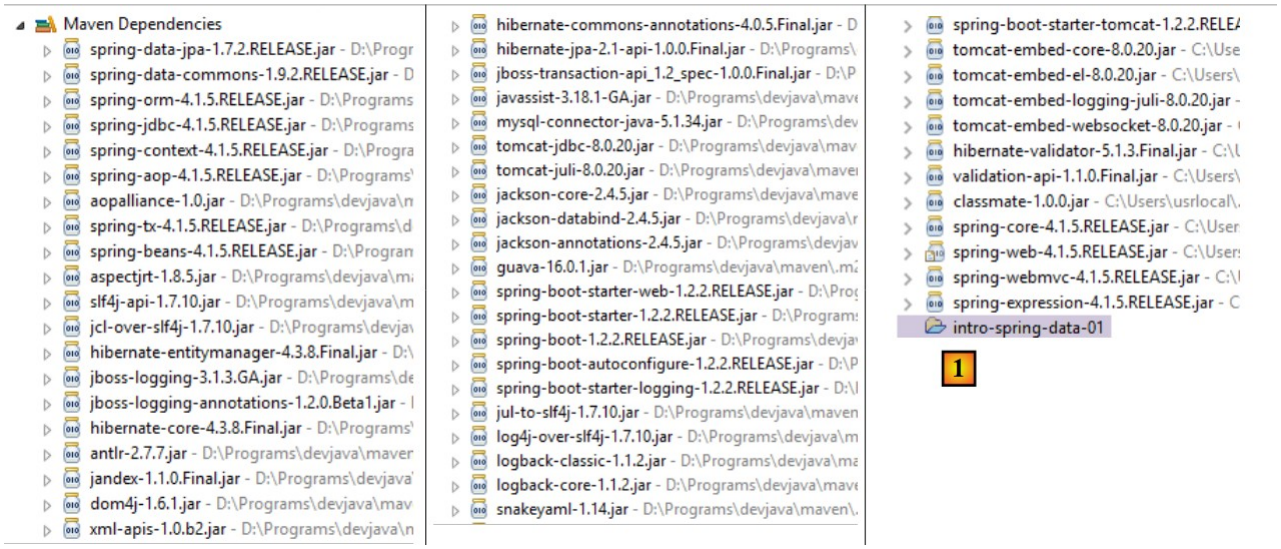
C'est un projet Maven dont le fichier [pom.xml] est le suivant :

```
1. <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2.   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
3.   <modelVersion>4.0.0</modelVersion>
4.   <groupId>istia.st.webjson</groupId>
5.   <artifactId>intro-server-webjson01</artifactId>
6.   <version>0.0.1-SNAPSHOT</version>
7.
8.   <name>intro-server-webjson01</name>
9.   <description>démo spring mvc</description>
10.
11.    <parent>
12.        <groupId>org.springframework.boot</groupId>
13.        <artifactId>spring-boot-starter-parent</artifactId>
14.        <version>1.2.7.RELEASE</version>
15.    </parent>
```

```
16.
17.   <dependencies>
18.       <dependency>
19.           <groupId>istia.st.springdata</groupId>
20.           <artifactId>intro-spring-data-01</artifactId>
21.           <version>0.0.1-SNAPSHOT</version>
22.       </dependency>
23.       <dependency>
24.           <groupId>org.springframework.boot</groupId>
25.           <artifactId>spring-boot-starter-web</artifactId>
26.       </dependency>
27.       <dependency>
28.           <groupId>org.springframework.boot</groupId>
29.           <artifactId>spring-boot-starter</artifactId>
30.       </dependency>
31.   </dependencies>
32.
33.   <build>
34.       <plugins>
35.           <plugin>
36.               <groupId>org.springframework.boot</groupId>
37.               <artifactId>spring-boot-maven-plugin</artifactId>
38.           </plugin>
39.           <plugin>
40.               <groupId>org.apache.maven.plugins</groupId>
41.               <artifactId>maven-surefire-plugin</artifactId>
42.               <version>2.18.1</version>
43.           </plugin>
44.       </plugins>
45.   </build>
46.
47. </project>
```

- lignes 11-15 : le projet Maven parent déjà utilisé pour la couche [DAO] ;
- lignes 18-22 : la dépendance sur la couche [DAO] ;
- lignes 23-26 : la dépendance sur l'artifact [spring-boot-starter-web]. Cet artifact amène avec lui toutes les dépendances nécessaires à la création d'un service web / jSON. Il amène aussi des bibliothèques inutiles. Une configuration plus précise serait donc nécessaire. Mais cette configuration est pratique pour démarrer ;
- lignes 28-30 : la dépendance sur l'artifact [spring-boot-starter] permet de gérer les annotation Spring Boot ;

Les dépendances amenées par cette configuration sont les suivantes :

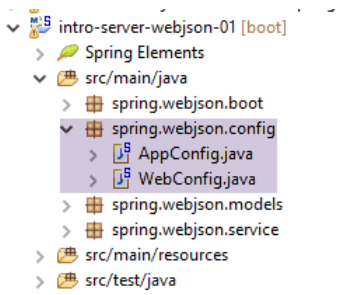


- en [1], on voit qu'Eclipse a vu la dépendance sur l'archive du projet [intro-spring-data-01] ;

Les dépendances ci-dessus sont à la fois celles de la couche [DAO] et de la couche [web].

13.5.3.1 Configuration de la couche [web]

La couche [web] est configurée par un fichier [AppConfig] :



La classe [WebConfig] configure la couche [web] :

```
1. package spring.webjson.config;
2.
3. import org.springframework.beans.factory.annotation.Autowired;
4. import org.springframework.boot.context.embedded.EmbeddedServletContainerFactory;
5. import org.springframework.boot.context.embedded.ServletRegistrationBean;
6. import org.springframework.boot.context.embedded.tomcat.TomcatEmbeddedServletContainerFactory;
7. import org.springframework.context.ApplicationContext;
8. import org.springframework.context.annotation.Bean;
9. import org.springframework.web.context.WebApplicationContext;
10.  import org.springframework.web.servlet.DispatcherServlet;
11.  import org.springframework.web.servlet.config.annotation.EnableWebMvc;
12.  import org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter;
13.
14.  import com.fasterxml.jackson.databind.ObjectMapper;
15.  import com.fasterxml.jackson.databind.ser.impl.SimpleBeanPropertyFilter;
16.  import com.fasterxml.jackson.databind.ser.impl.SimpleFilterProvider;
17.
18.  @EnableWebMvc
19.  public class WebConfig extends WebMvcConfigurerAdapter {
20.
21.    // ------------------------------- configuration couche [web]
22.    @Autowired
23.    private ApplicationContext context;
24.
25.    @Bean
26.    public DispatcherServlet dispatcherServlet() {
27.        DispatcherServlet servlet = new DispatcherServlet((WebApplicationContext) context);
28.        return servlet;
29.    }
30.
31.    @Bean
32.    public ServletRegistrationBean servletRegistrationBean(DispatcherServlet dispatcherServlet) {
33.        return new ServletRegistrationBean(dispatcherServlet, "/*");
34.    }
35.
36.    @Bean
37.    public EmbeddedServletContainerFactory embeddedServletContainerFactory() {
38.        return new TomcatEmbeddedServletContainerFactory("", 8080);
39.    }
40.
41.    // filtres jSON
42.    @Bean(name = "jsonMapper")
43.    public ObjectMapper jsonMapper() {
44.        return new ObjectMapper();
45.    }
46.
47.    @Bean(name = "jsonMapperCategorieWithProduits")
48.    public ObjectMapper jsonMapperCategorieWithProduits() {
49.        // mappeur jSON
50.        ObjectMapper mapper = new ObjectMapper();
51.        // filtres
52.        mapper.setFilters(
53.                new SimpleFilterProvider().addFilter("jsonFilterCategorie", SimpleBeanPropertyFilter.serializeAllExcept())
54.                        .addFilter("jsonFilterProduit", SimpleBeanPropertyFilter.serializeAllExcept("categorie")));
55.        // résultat
56.        return mapper;
57.    }
58.
59.    @Bean(name = "jsonMapperProduitWithCategorie")
60.    public ObjectMapper jsonMapperProduitWithCategorie() {
61.        // mappeur jSON
62.        ObjectMapper mapper = new ObjectMapper();
63.        // filtres
64.        mapper.setFilters(
65.                new SimpleFilterProvider().addFilter("jsonFilterProduit", SimpleBeanPropertyFilter.serializeAllExcept())
```

```
66.                     .addFilter("jsonFilterCategorie", SimpleBeanPropertyFilter.serializeAllExcept("produits")));
67.       // résultat
68.       return mapper;
69.   }
70.
71.   @Bean(name = "jsonMapperCategorieWithoutProduits")
72.   public ObjectMapper jsonMapperCategorieWithoutProduits() {
73.       // mappeur jSON
74.       ObjectMapper mapper = new ObjectMapper();
75.       // filtres
76.       mapper.setFilters(new SimpleFilterProvider().addFilter("jsonFilterCategorie",
77.               SimpleBeanPropertyFilter.serializeAllExcept("produits")));
78.       // résultat
79.       return mapper;
80.   }
81.
82.   @Bean(name = "jsonMapperProduitWithoutCategorie")
83.   public ObjectMapper jsonMapperProduitWithoutCategorie() {
84.       // mappeur jSON
85.       ObjectMapper mapper = new ObjectMapper();
86.       // filtres
87.       mapper.setFilters(new SimpleFilterProvider().addFilter("jsonFilterProduit",
88.               SimpleBeanPropertyFilter.serializeAllExcept("categorie")));
89.       // résultat
90.       return mapper;
91.   }
92. }
```

- ligne 18 : l'annotation [`@EnableWebMvc`] induit des configurations automatiques pour le framework Spring MVC ;
- ligne 19 : la classe [WebConfig] étend la classe Spring [WebMvcConfigurerAdapter] pour en redéfinir certains beans (lignes 26-40) ;
- lignes 22-23 : injection du contexte Spring ;
- lignes 25-29 : définition de la servlet du framework Spring MVC, celle qui route les requêtes HTTP vers le bon contrôleur et la bonne méthode. [DispatcherServlet] est une classe de Spring ;
- lignes 31-34 : on indique que cette servlet traite toutes les URL ;
- lignes 36-39 : c'est la présence de ce bean qui va activer le serveur Tomcat présent dans les archives du projet. Il attendra les requêtes sur le port 8080 ;
- lignes 42-91 : des beans qui seront utilisés pour gérer des filtres jSON ;
- lignes 42-45 : un mappeur jSON sans filtres ;
- lignes 47-57 : le mappeur jSON qui permet d'avoir une catégorie avec ses produits. On notera que lorsqu'on demande une catégorie avec ses produits, il faut à la fois configurer le filtre jSON de la classe [Categorie] et celui de la classe [Produit]. Il en est toujours ainsi. Lorsqu'on sérialise / désérialise une classe en jSON, il faut configurer le filtre jSON de la classe et ceux de toutes les dépendances à inclure dans celle-ci ;
- lignes 59-69 : le mappeur jSON qui permet d'avoir un produit avec sa catégorie ;
- lignes 71-80 : le mappeur jSON qui permet d'avoir une catégorie sans ses produits ;
- lignes 82-91 : le mappeur jSON qui permet d'avoir un produit sans sa catégorie ;

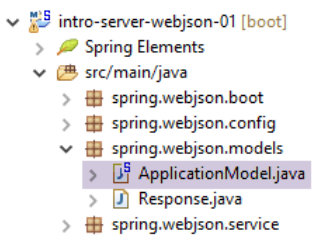
La classe [AppConfig] configure l'ensemble de l'application, ç-à-d les couches [web] et [DAO] :

```
1. package spring.webjson.config;
2.
3. import org.springframework.context.annotation.ComponentScan;
4. import org.springframework.context.annotation.Import;
5.
6. import spring.data.config.DaoConfig;
7.
8. @ComponentScan(basePackages = { "spring.webjson" })
9. @Import({ DaoConfig.class, WebConfig.class})
10.  public class AppConfig {
11.
12.  }
```

- ligne 9 : on importe les beans de la couche [DAO] et ceux de la couche [web] ;
- ligne 8 : indique dans quels packages trouver d'autres beans Spring ;

On notera que nulle part, nous n'avons utilisé l'annotation [@EnableAutoConfiguration]. Nous avons préféré contrôler la configuration nous-mêmes.

13.5.4 Le modèle de l'application



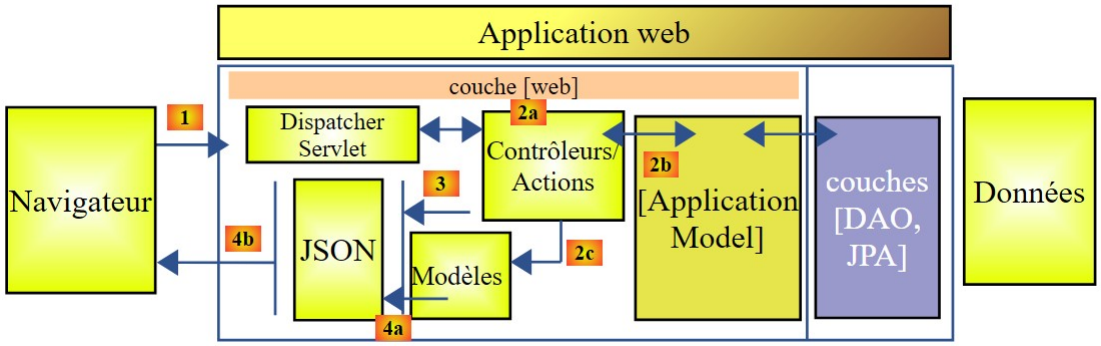
La classe [ApplicationModel] est la suivante :

```
1. package spring.webjson.models;
2.
3. import java.util.List;
4.
5. import org.springframework.beans.factory.annotation.Autowired;
6. import org.springframework.stereotype.Component;
7.
8. import spring.data.dao.IDao;
9. import spring.data.entities.Categorie;
10.  import spring.data.entities.Produit;
11.
12.  @Component
13.  public class ApplicationModel implements IDao {
14.
15.    // la couche [DAO]
16.    @Autowired
17.    private IDao dao;
18.
19.    @Override
20.    public void addProduits(List<Produit> produits) {
21.        dao.addProduits(produits);
22.    }
23.
24.    @Override
25.    public void deleteAllProduits() {
26.        dao.deleteAllProduits();
27.    }
28.
29.    @Override
30.    public void updateProduits(List<Produit> produits) {
31.        dao.updateProduits(produits);
32.    }
33.
34.    @Override
35.    public List<Produit> getAllProduits() {
36.        return dao.getAllProduits();
37.    }
38.
39.    @Override
40.    public void addCategories(List<Categorie> categories) {
41.        dao.addCategories(categories);
42.    }
43.
44.    @Override
45.    public void deleteAllCategories() {
46.        dao.deleteAllCategories();
47.    }
48.
49.    @Override
50.    public void updateCategories(List<Categorie> categories) {
51.        dao.updateCategories(categories);
52.    }
53.
54.    @Override
55.    public List<Categorie> getAllCategories() {
56.        return dao.getAllCategories();
57.    }
58.
59.    @Override
60.    public Produit getProduitByIdWithCategorie(Long idProduit) {
61.        return dao.getProduitByIdWithCategorie(idProduit);
62.    }
63.
64.    @Override
65.    public Produit getProduitByNameWithCategorie(String nom) {
66.        return dao.getProduitByNameWithCategorie(nom);
67.    }
68.
```

```
69.    @Override
70.    public Categorie getCategorieByIdWithProduits(Long idCategorie) {
71.        return dao.getCategorieByIdWithProduits(idCategorie);
72.    }
73.
74.    @Override
75.    public Categorie getCategorieByNameWithProduits(String nom) {
76.        return dao.getCategorieByNameWithProduits(nom);
77.    }
78.
79.    @Override
80.    public Produit getProduitByIdWithoutCategorie(Long idProduit) {
81.        return dao.getProduitByIdWithoutCategorie(idProduit);
82.    }
83.
84.    @Override
85.    public Categorie getCategorieByIdWithoutProduits(Long idCategorie) {
86.        return dao.getCategorieByIdWithoutProduits(idCategorie);
87.    }
88.
89.    @Override
90.    public Produit getProduitByNameWithoutCategorie(String nom) {
91.        return dao.getProduitByNameWithoutCategorie(nom);
92.    }
93.
94.    @Override
95.    public Categorie getCategorieByNameWithoutProduits(String nom) {
96.        return dao.getCategorieByNameWithoutProduits(nom);
97.    }
98.
99.  }
```

- ligne 12 : la classe est un singleton Spring ;
- ligne 13 : qui implémente l'interface [IDao] de la couche [DAO] ;
- lignes 16-17 : injection d'une référence sur la couche [DAO] ;
- lignes 19-99 : implémentation de l'interface [IDao] ;

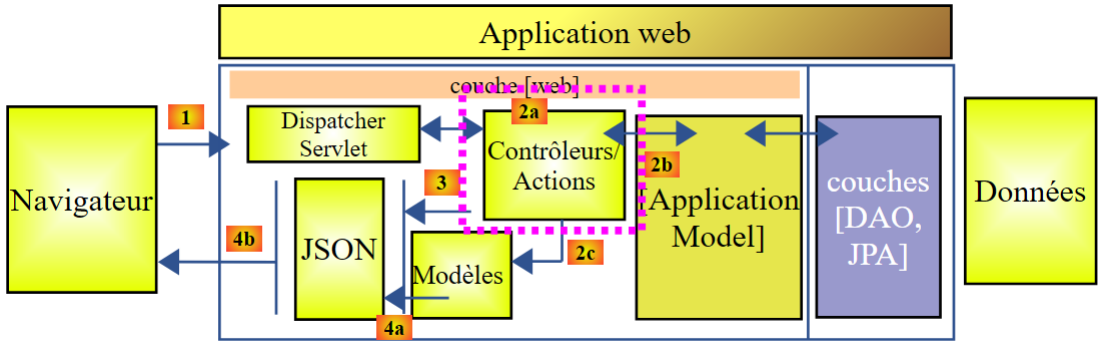
L'architecture de la couche web évolue comme suit :

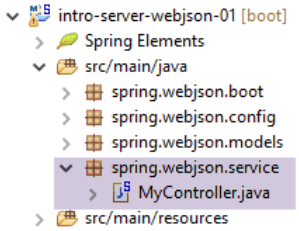


- en [2b], les méthodes du ou des contrôleurs communiquent avec le singleton [ApplicationModel] ;

Cette stratégie amène de la souplesse quant à la gestion d'un éventuel cache. La classe [ApplicationModel] peut servir à mémoriser des informations obtenues auprès de la couche [DAO] ou encore des données de configuration. Cela peut être utile lorsqu'on n'a pas la maîtrise de la couche [DAO]. Cette stratégie de cache peut évoluer au fil du temps. Les modifications n'auront aucun impact sur le code du ou des contrôleurs.

13.5.5 Le contrôleur





Nous n'avons ici qu'un contrôleur, la classe [MyController].

13.5.5.1 Les URL exposées

Les URL exposées par ce contrôleur sont les suivantes :

```
    @RequestMapping(value = "/addProduits", method = RequestMethod.POST, consumes =
"application/json; charset=UTF-8")
    public String addProduits(HttpServletRequest request) {
...
    }
```

Ajoute des produits dans la base. Ceux-ci sont postés. La réponse est la chaîne jSON la liste des produits ajoutés avec leur clé primaire.

```
    @RequestMapping(value = "/deleteAllProduits", method = RequestMethod.GET)
    public String deleteAllProduits() {
..
    }
```

Supprime tous les produits de la base.

```
    @RequestMapping(value = "/updateProduits", method = RequestMethod.POST, consumes =
"application/json; charset=UTF-8")
    public String updateProduits(HttpServletRequest request) {
..
    }
```

Met à jour des produits dans la base. Ceux-ci sont postés. La réponse est chaîne jSON de la liste des produits mis à jour.

```
    @RequestMapping(value = "/getAllProduits", method = RequestMethod.GET)
    public String getAllProduits() {
..
    }
```

Obtient la chaîne jSON de tous les produits.

```
    @RequestMapping(value = "/addCategories", method = RequestMethod.POST, consumes =
"application/json; charset=UTF-8")
    public String addCategories(HttpServletRequest request) {
..
    }
```

Ajoute des catégories dans la base. Ceux-ci sont postés. La réponse est la chaîne jSON de la liste des catégories ajoutées avec leur clé primaire. Si les catégories contiennent des produits, ceux-ci sont également ajoutés à la base.

```
    @RequestMapping(value = "/deleteAllCategories", method = RequestMethod.GET)
    public String deleteAllCategories() {
...
    }
```

Supprime toutes les catégories de la base ainsi que tous les produits de celles-ci. Après la base est vide.

`@RequestMapping(value = "/updateCategories", method = RequestMethod.POST, consumes = "application/json; charset=UTF-8")`
`public String updateCategories(HttpServletRequest request) {`
`...`
`}`
	Met à jour des catégories dans la base. Ceux-ci sont postés. La réponse est la liste des catégories mises à jour. Si les catégories contiennent des produits, ceux-ci sont également mis à jour dans la base. Rend la chaîne jSON des catégories modifiées ;

`@RequestMapping(value = "/getAllCategories", method = RequestMethod.GET)`
`public String getAllCategories() {`
`...`
`}`
	Obtient la chaîne jSON de toutes les catégories.

`@RequestMapping(value = "/getProduitByIdWithCategorie/{idProduit}", method = RequestMethod.GET)`
`public String getProduitByIdWithCategorie(@PathVariable("idProduit") Long idProduit) {`
`...`
`}`
	Obtient la chaîne jSON d'un produit désigné par son id, avec sa catégorie.

`@RequestMapping(value = "/getProduitByIdWithoutCategorie/{idProduit}", method = RequestMethod.GET)`
`public String getProduitByIdWithoutCategorie(@PathVariable("idProduit") Long idProduit) {`
`...`
`}`
	Obtient la chaîne jSON d'un produit désigné par son id, sans sa catégorie.

`@RequestMapping(value = "/getProduitByNameWithCategorie/{nom}", method = RequestMethod.GET)`
`public String getProduitByNameWithCategorie(@PathVariable("nom") String nom) {`
`...`
`}`
	Obtient la chaîne jSON d'un produit désigné par son nom, avec sa catégorie.

`@RequestMapping(value = "/getProduitByNameWithoutCategorie/{nom}", method = RequestMethod.GET)`
`public String getProduitByNameWithoutCategorie(@PathVariable("nom") String nom) {`
`...`
`}`
	Obtient la chaîne jSON d'un produit désigné par son nom, sans sa catégorie.

`@RequestMapping(value = "/getCategorieByIdWithProduits/{idCategorie}", method = RequestMethod.GET)`
`public String getCategorieByIdWithProduits(@PathVariable("idCategorie") Long idCategorie) {`
`...`
`}`
	Obtient la chaîne jSON d'une catégorie désignée par son id, avec ses produits.

`@RequestMapping(value = "/getCategorieByNameWithProduits/{nom}", method = RequestMethod.GET)`
`public String getCategorieByNameWithProduits(@PathVariable("nom") String nom) {`
`...`
`}`
	Obtient la chaîne jSON d'une catégorie désignée par son nom, avec ses produits.

`@RequestMapping(value = "/getCategorieByNameWithoutProduits/{nom}", method = RequestMethod.GET)`
`public String getCategorieByNameWithoutProduits(@PathVariable("nom") String nom) {`
`...`
`}`
	Obtient la chaîne jSON d'une catégorie désignée par son nom, sans ses produits.

`@RequestMapping(value = "/getCategorieByIdWithoutProduits/{idCategorie}", method = RequestMethod.GET)`
`public String getCategorieByIdWithoutProduits(@PathVariable("idCategorie") Long idCategorie) {`
`...`
`}`
	Obtient la chaîne jSON d'une catégorie désignée par son id sans ses produits.

Les URL exposées correspondent aux méthodes de l'interface [IDao] de la couche [DAO]. Les méthodes du service web / jSON sont toutes bâties sur le même modèle. Nous allons en examiner quelques unes.

13.5.5.2 Le squelette du contrôleur

Le squelette du contrôleur est le suivant :

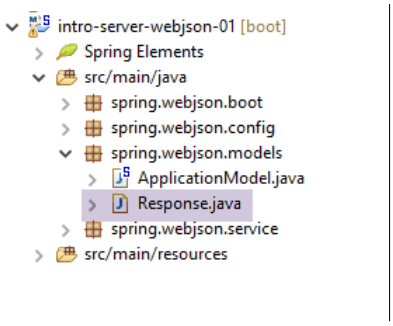
```
1. package spring.webjson.service;
2.
3. import java.util.ArrayList;
4. import java.util.List;
```

```
5. import java.util.Set;
6.
7. import javax.servlet.http.HttpServletRequest;
8.
9. import org.springframework.beans.factory.annotation.Autowired;
10.  import org.springframework.beans.factory.annotation.Qualifier;
11.  import org.springframework.stereotype.Controller;
12.  import org.springframework.web.bind.annotation.PathVariable;
13.  import org.springframework.web.bind.annotation.RequestMapping;
14.  import org.springframework.web.bind.annotation.RequestMethod;
15.  import org.springframework.web.bind.annotation.ResponseBody;
16.
17.  import com.fasterxml.jackson.core.JsonProcessingException;
18.  import com.fasterxml.jackson.core.type.TypeReference;
19.  import com.fasterxml.jackson.databind.ObjectMapper;
20.  import com.google.common.io.CharStreams;
21.
22.  import spring.data.dao.DaoException;
23.  import spring.data.entities.Categorie;
24.  import spring.data.entities.Produit;
25.  import spring.webjson.models.ApplicationModel;
26.  import spring.webjson.models.Response;
27.
28.  @Controller
29.  public class MyController {
30.
31.    // dépendances Spring
32.    @Autowired
33.    private ApplicationModel application;
34.
35.    // filtres jSON
36.    @Autowired
37.    @Qualifier("jsonMapper")
38.    private ObjectMapper jsonMapper;
39.    @Autowired
40.    @Qualifier("jsonMapperCategorieWithProduits")
41.    private ObjectMapper jsonMapperCategorieWithProduits;
42.    @Autowired
43.    @Qualifier("jsonMapperProduitWithCategorie")
44.    private ObjectMapper jsonMapperProduitWithCategorie;
45.    @Autowired
46.    @Qualifier("jsonMapperCategorieWithoutProduits")
47.    private ObjectMapper jsonMapperCategorieWithoutProduits;
48.    @Autowired
49.    @Qualifier("jsonMapperProduitWithoutCategorie")
50.    private ObjectMapper jsonMapperProduitWithoutCategorie;
51.
52.    // la classe [MyController] est un singleton et n'est instanciée qu'une fois le bean
53.
54.    public MyController() {
55.        // System.out.println("MyController");
56.    }
57.
58.    @RequestMapping(value = "/addProduits", method = RequestMethod.POST, consumes = "application/json; charset=UTF-8",
     produces = "application/json; charset=UTF-8")
59.    @ResponseBody
60.    public String addProduits(HttpServletRequest request) throws JsonProcessingException {
61.        ...
62.    }
63.
```

- ligne 28 : l'annotation [`@Controller`] fait de la classe un composant Spring ;
- lignes 32-33 : injection d'une référence sur la classe [ApplicationModel] ;
- lignes 36-50 : injections de références sur les mappeurs jSON ;
- ligne 58 : l'URL exposée est [/addProduits]. Le client doit utiliser une méthode [POST] pour faire sa requête (`method = RequestMethod.POST`). Il doit envoyer la valeur postée sous forme d'une chaîne jSON (`consumes = "application/json; charset=UTF-8"`). La méthode renvoie elle-même la réponse au client (ligne 59). Ce sera une chaîne de caractères (ligne 60). L'entête HTTP [Content-type : application/json; charset=UTF-8] sera envoyé au client pour lui indiquer qu'il va recevoir une chaîne jSON (ligne 58) ;
- ligne 60 : la méthode [`addProduits`] rend la chaîne jSON de la liste des produits ajoutés dans la base ;

13.5.5.3 La réponse des méthodes du contrôleur

Toutes les méthodes du contrôleur rendent la réponse de type [Response] suivant :




```
1. package spring.webjson.service;
2.
3. import java.util.List;
4.
5. public class Response<T> {
6.
7.   // ---------------- propriétés
8.   // statut de l'opération
9.   private int status;
10.    // les éventuels messages d'erreur
11.    private List<String> messages;
12.    // le corps de la réponse
13.    private T body;
14.
15.    // constructeurs
16.    public Response() {
17.
18.    }
19.
20.    public Response(int status, List<String> messages, T body) {
21.        this.status = status;
22.        this.messages = messages;
23.        this.body = body;
24.    }
25.
26.    // getters et setters
27.    ...
28. }
```

- ligne 5 : la réponse encapsule un type T ;
- ligne 13 : la réponse de type T ;
- lignes 9-11 : il est possible qu'une méthode rencontre une exception. Dans ce cas, elle rendra une réponse avec :
  - ligne 9 : status!=0 ;
  - ligne 11 : la liste des erreurs rencontrées ;

13.5.5.4 L'URL [/addProduits]

L'URL [/addProduits] est traitée par la méthode suivante :

```
1. @RequestMapping(value = "/addProduits", method = RequestMethod.POST, consumes = "application/json; charset=UTF-8",
    produces = "application/json; charset=UTF-8")
2.   @ResponseBody
3.   public String addProduits(HttpServletRequest request) throws JsonProcessingException {
4.     // réponse
5.     Response<List<Produit>> response;
6.     try {
7.         // on récupère la valeur postée
8.         String body = CharStreams.toString(request.getReader());
9.         List<Produit> produits = jsonMapperProduitWithoutCategorie.readValue(body, new TypeReference<List<Produit>>() {
10.            });
11.            // on rétablit le lien entre produits et catégories
12.            for (Produit produit : produits) {
13.                produit.setCategorie(application.getCategorieByIdWithoutProduits(produit.getIdCategorie()));
14.            }
15.            // on persiste les produits
16.            application.addProduits(produits);
17.            response = new Respon se<List<Produit>>(0, null, produits);
18.        } catch (DaoException e1) {
19.            response = new Response<List<Produit>>(1000, e1.getErreurs(), null);
20.        } catch (Exception e2) {
21.            response = new Response<List<Produit>>(1000, getErreursForException(e2), null);
22.        }
23.        // réponse jSON
24.        return jsonMapperProduitWithoutCategorie.writeValueAsString(response);
25.    }
```

- ligne 3 : la méthode admet pour paramètre [HttpServletRequest request] qui encapsule toutes les informations sur la requête du client ;
- ligne 5 : la réponse qui sera envoyée au client : une liste de produits ;
- ligne 8 : on récupère la valeur postée. La classe [CharStreams] appartient à la bibliothèque [Google Guava] dont on a ajouté la référence dans le fichier [pom.xml]. On obtient la chaîne jSON postée par le client. Il faut la désérialiser pour en faire quelque chose ;
- lignes 8-10 : la désérialisation est faite. On obtient une liste de produits où chaque produit a un champ [categorie=null] ;
- lignes 12-14 : on réinitialise le champ [categorie] de tous les produits de la liste. Pour cela, on utilise le champ [idCategorie] du produit qui lui, est initialisé ;
- ligne 16 : les produits sont insérés dans la base ;
- ligne 17 : l'objet [response] est initialisé avec la liste de produits ;
- lignes 18-19 : cas où la méthode rencontre une exception de la couche [DAO]. On initialise la réponse avec [status=1000] (code d'erreur) [messages=e1.getMessages()], ç-à-d qu'on transmet au client la liste des erreurs rencontrées côté serveur ;
- lignes 20-21 : cas où la méthode rencontre un autre type d'exception. On initialise la réponse avec [status=1000] (code d'erreur) [messages=getErreursForException(e)] où [getErreursForException] est une méthode privée de la classe qui rend la liste des erreurs associées aux exceptions de la pile d'exceptions de e, et [body=null] ;
- ligne 24 : on rend la chaîne jSON de la réponse ;

13.5.5.5 L'URL [/getAllProduits]

L'URL [/getAllProduits] est traitée par la méthode suivante :

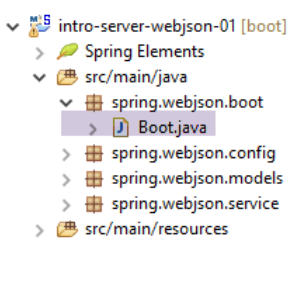
```
1.  @RequestMapping(value = "/getAllProduits", method = RequestMethod.GET, produces = "application/json; charset=UTF-8")
2.  @ResponseBody
3.  public String getAllProduits() throws JsonProcessingException {
4.    // réponse
5.    Response<List<Produit>> response;
6.    try {
7.        response = new Response<List<Produit>>(0, null, application.getAllProduits());
8.    } catch (DaoException e1) {
9.        response = new Response<List<Produit>>(1003, e1.getErreurs(), null);
10.       } catch (Exception e2) {
11.           response = new Response<List<Produit>>(1003, getErreursForException(e2), null);
12.       }
13.       // réponse jSON
14.       return jsonMapperProduitWithoutCategorie.writeValueAsString(response);
15. }
```

- ligne 1 : l'URL [/getAllProduits] est demandée avec une opération [GET]. Elle produit du jSON ;
- ligne 2 : la méthode envoie elle-même la réponse jSON au client ;
- ligne 5 : la méthode envoie la chaîne jSON d'un type [Response<List<Produit>>] ;
- ligne 7 : les produits sont demandés sans leur catégorie ;
- lignes 8-12 : en cas d'erreur, la réponse est initialisée avec un code et des messages d'erreur ;
- ligne 14 : la réponse jSON de la réponse est envoyée au client ;

13.5.5.6 Conclusion

Nous n'allons pas présenter les autres méthodes du contrôleur. Elles ressemblent à l'une ou l'autre des deux méthodes que nous enons de présenter.

13.5.6 La classe d'exécution du service web / jSON



La classe [Boot] est la classe exécutable du projet :

```
1. package spring.webjson.boot;
2.
3. import org.springframework.boot.SpringApplication;
4.
5. import spring.webjson.server.config.AppConfig;
6.
7. public class Boot {
8.
9.   public static void main(String[] args) {
10.       SpringApplication.run(AppConfig.class, args);
11.    }
12. }
```

- ligne 10 : la méthode statique [SpringApplication.*run*] est exécutée. La classe [SpringApplication] est une classe du projet [Spring Boot] (ligne 3). On lui passe deux paramètres :
  - [AppConfig.**class**] : la classe qui configure la totalité de l'application ;
  - [args] : les éventuels arguments passés à la méthode [main] ligne 9. Ce paramètre n'est pas utilisé ici ;

Lorsqu'on exécute cette classe, on a les logs suivants :

```
1.  .   ____          _            __ _ _
2.   /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
3.  ( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
4.   \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
5.    '  |____| .__|_| |_|_| |_\__, | / / / /
6.   =========|_|==============|___/=/_/_/_/
7.   :: Spring Boot ::        (v1.2.2.RELEASE)
8.
9.  2015-03-24 16:22:46.608  INFO 9492 --- [           main] spring.webjson.server.boot.Boot          : Starting Boot on
    Gportpers3 with PID 9492 (D:\data\istia-1415\eclipse\intro-web-json\intro-webjson-server-02\target\classes started by ST
    in D:\data\istia-1415\eclipse\intro-web-json\intro-webjson-server-02)
10. 2015-03-24 16:22:46.654  INFO 9492 --- [           main] ationConfigEmbeddedWebApplicationContext : Refreshing
    org.springframework.boot.context.embedded.AnnotationConfigEmbeddedWebApplicationContext@1d7acb34: startup date [Tue Mar
    24 16:22:46 CET 2015]; root of context hierarchy
11. 2015-03-24 16:22:47.521  INFO 9492 --- [           main] o.s.b.f.s.DefaultListableBeanFactory     : Overriding bean
    definition for bean 'beanNameViewResolver': replacing [Root bean: class [null]; scope=; abstract=false; lazyInit=false;
    autowireMode=3; dependencyCheck=0; autowireCandidate=true; primary=false;
    factoryBeanName=org.springframework.boot.autoconfigure.web.ErrorMvcAutoConfiguration$WhitelabelErrorViewConfiguration;
    factoryMethodName=beanNameViewResolver; initMethodName=null; destroyMethodName=(inferred); defined in class path
    resource [org/springframework/boot/autoconfigure/web/ErrorMvcAutoConfiguration$WhitelabelErrorViewConfiguration.class]]
    with [Root bean: class [null]; scope=; abstract=false; lazyInit=false; autowireMode=3; dependencyCheck=0;
    autowireCandidate=true; primary=false;
    factoryBeanName=org.springframework.boot.autoconfigure.web.WebMvcAutoConfiguration$WebMvcAutoConfigurationAdapter;
    factoryMethodName=beanNameViewResolver; initMethodName=null; destroyMethodName=(inferred); defined in class path
    resource [org/springframework/boot/autoconfigure/web/WebMvcAutoConfiguration$WebMvcAutoConfigurationAdapter.class]]
12. 2015-03-24 16:22:47.569  INFO 9492 --- [           main] o.s.b.f.s.DefaultListableBeanFactory     : Overriding bean
    definition for bean 'entityManagerFactory': replacing [Root bean: class [null]; scope=; abstract=false; lazyInit=false;
    autowireMode=3; dependencyCheck=0; autowireCandidate=true; primary=false; factoryBeanName=spring.data.config.DaoConfig;
    factoryMethodName=entityManagerFactory; initMethodName=null; destroyMethodName=(inferred); defined in class
    spring.data.config.DaoConfig] with [Root bean: class [null]; scope=; abstract=false; lazyInit=false; autowireMode=3;
    dependencyCheck=0; autowireCandidate=true; primary=true;
    factoryBeanName=org.springframework.boot.autoconfigure.orm.jpa.HibernateJpaAutoConfiguration;
    factoryMethodName=entityManagerFactory; initMethodName=null; destroyMethodName=(inferred); defined in class path
    resource [org/springframework/boot/autoconfigure/orm/jpa/HibernateJpaAutoConfiguration.class]]
13. 2015-03-24 16:22:48.137  INFO 9492 --- [           main] trationDelegate$BeanPostProcessorChecker : Bean
    'org.springframework.transaction.annotation.ProxyTransactionManagementConfiguration' of type [class
    org.springframework.transaction.annotation.ProxyTransactionManagementConfiguration$$EnhancerBySpringCGLIB$$405db6ba] is
    not eligible for getting processed by all BeanPostProcessors (for example: not eligible for auto-proxying)
14. 2015-03-24 16:22:48.162  INFO 9492 --- [           main] trationDelegate$BeanPostProcessorChecker : Bean
    'transactionAttributeSource' of type [class
    org.springframework.transaction.annotation.AnnotationTransactionAttributeSource] is not eligible for getting processed
    by all BeanPostProcessors (for example: not eligible for auto-proxying)
15. 2015-03-24 16:22:48.172  INFO 9492 --- [           main] trationDelegate$BeanPostProcessorChecker : Bean
    'transactionInterceptor' of type [class org.springframework.transaction.interceptor.TransactionInterceptor] is not
    eligible for getting processed by all BeanPostProcessors (for example: not eligible for auto-proxying)
16. 2015-03-24 16:22:48.178  INFO 9492 --- [           main] trationDelegate$BeanPostProcessorChecker : Bean
    'org.springframework.transaction.config.internalTransactionAdvisor' of type [class
    org.springframework.transaction.interceptor.BeanFactoryTransactionAttributeSourceAdvisor] is not eligible for getting
    processed by all BeanPostProcessors (for example: not eligible for auto-proxying)
17. 2015-03-24 16:22:48.586  INFO 9492 --- [           main] s.b.c.e.t.TomcatEmbeddedServletContainer : Tomcat initialized
    with port(s): 8080 (http)
18. 2015-03-24 16:22:48.850  INFO 9492 --- [           main] o.apache.catalina.core.StandardService   : Starting service
    Tomcat
19. 2015-03-24 16:22:48.852  INFO 9492 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet
    Engine: Apache Tomcat/8.0.20
20. 2015-03-24 16:22:48.992  INFO 9492 --- [ost-startStop-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring
    embedded WebApplicationContext
21. 2015-03-24 16:22:48.992  INFO 9492 --- [ost-startStop-1] o.s.web.context.ContextLoader            : Root
    WebApplicationContext: initialization completed in 2342 ms
22. 2015-03-24 16:22:49.645  INFO 9492 --- [ost-startStop-1] o.s.b.c.e.ServletRegistrationBean        : Mapping servlet:
    'dispatcherServlet' to [/]
23. 2015-03-24 16:22:49.650  INFO 9492 --- [ost-startStop-1] o.s.b.c.embedded.FilterRegistrationBean  : Mapping filter:
    'characterEncodingFilter' to: [/*]
```

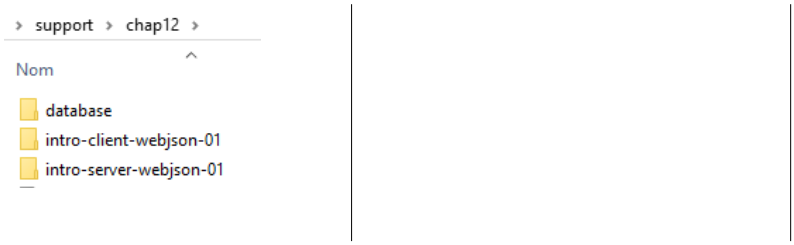
```
24. 2015-03-24 16:22:49.651  INFO 9492 --- [ost-startStop-1] o.s.b.c.embedded.FilterRegistrationBean  : Mapping filter:
    'hiddenHttpMethodFilter' to: [/*]
25. 2015-03-24 16:22:50.380  INFO 9492 --- [           main] j.LocalContainerEntityManagerFactoryBean : Building JPA
    container EntityManagerFactory for persistence unit 'default'
26. 2015-03-24 16:22:50.392  INFO 9492 --- [           main] o.hibernate.jpa.internal.util.LogHelper  : HHH000204: Processing
    PersistenceUnitInfo [
27.  name: default
28.  ...]
29. 2015-03-24 16:22:50.478  INFO 9492 --- [           main] org.hibernate.Version                    : HHH000412: Hibernate
    Core {4.3.8.Final}
30. 2015-03-24 16:22:50.480  INFO 9492 --- [           main] org.hibernate.cfg.Environment            : HHH000206:
    hibernate.properties not found
31. 2015-03-24 16:22:50.483  INFO 9492 --- [           main] org.hibernate.cfg.Environment            : HHH000021: Bytecode
    provider name : javassist
32. 2015-03-24 16:22:50.697  INFO 9492 --- [           main] o.hibernate.annotations.common.Version   : HCANN000001:
    Hibernate Commons Annotations {4.0.5.Final}
33. 2015-03-24 16:22:50.806  INFO 9492 --- [           main] org.hibernate.dialect.Dialect            : HHH000400: Using
    dialect: org.hibernate.dialect.MySQLDialect
34. 2015-03-24 16:22:51.058  INFO 9492 --- [           main] o.h.h.i.ast.ASTQueryTranslatorFactory    : HHH000397: Using
    ASTQueryTranslatorFactory
35. 2015-03-24 16:22:52.581  INFO 9492 --- [           main] s.w.s.m.m.a.RequestMappingHandlerAdapter : Looking for
    @ControllerAdvice: org.springframework.boot.context.embedded.AnnotationConfigEmbeddedWebApplicationContext@1d7acb34:
    startup date [Tue Mar 24 16:22:46 CET 2015]; root of context hierarchy
36. 2015-03-24 16:22:52.654  INFO 9492 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped
    "{[/addProduits],methods=[POST],params=[],headers=[],consumes=[application/json;charset=UTF-8],produces=[],custom=[]}"
    onto public spring.webjson.webjson.models.Response<java.util.List<spring.data.entities.Produit>>
    spring.webjson.server.service.Controller.addProduits(javax.servlet.http.HttpServletRequest)
37. 2015-03-24 16:22:52.655  INFO 9492 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped
    "{[/updateProduits],methods=[POST],params=[],headers=[],consumes=[application/
    json;charset=UTF-8],produces=[],custom=[]}" onto public
    spring.webjson.webjson.models.Response<java.util.List<spring.data.entities.Produit>>
    spring.webjson.server.service.Controller.updateProduits(javax.servlet.http.HttpServletRequest)
38. 2015-03-24 16:22:52.655  INFO 9492 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped
    "{[/getAllProduits],methods=[GET],params=[],headers=[],consumes=[],produces=[],custom=[]}" onto public
    spring.webjson.webjson.models.Response<java.util.List<spring.data.entities.Produit>>
    spring.webjson.server.service.Controller.getAllProduits()
39. 2015-03-24 16:22:52.655  INFO 9492 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped
    "{[/getAllCategories],methods=[GET],params=[],headers=[],consumes=[],produces=[],custom=[]}" onto public
    spring.webjson.webjson.models.Response<java.util.List<spring.data.entities.Categorie>>
    spring.webjson.server.service.Controller.getAllCategories()
40. 2015-03-24 16:22:52.655  INFO 9492 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped
    "{[/addCategories],methods=[POST],params=[],headers=[],consumes=[application/json;charset=UTF-8],produces=[],custom=[]}"
    onto public spring.webjson.webjson.models.Response<java.util.List<spring.data.entities.Categorie>>
    spring.webjson.server.service.Controller.addCategories(javax.servlet.http.HttpServletRequest)
41. 2015-03-24 16:22:52.655  INFO 9492 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped
    "{[/updateCategories],methods=[POST],params=[],headers=[],consumes=[application/
    json;charset=UTF-8],produces=[],custom=[]}" onto public
    spring.webjson.webjson.models.Response<java.util.List<spring.data.entities.Categorie>>
    spring.webjson.server.service.Controller.updateCategories(javax.servlet.http.HttpServletRequest)
42. 2015-03-24 16:22:52.656  INFO 9492 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped
    "{[/getCategorieByNameWithoutProduits/{nom}],methods=[GET],params=[],headers=[],consumes=[],produces=[],custom=[]}" onto
    public spring.webjson.webjson.models.Response<spring.data.entities.Categorie>
    spring.webjson.server.service.Controller.getCategorieByNameWithoutProduits(java.lang.String)
43. 2015-03-24 16:22:52.656  INFO 9492 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped
    "{[/getProduitByNameWithoutCategorie/{nom}],methods=[GET],params=[],headers=[],consumes=[],produces=[],custom=[]}" onto
    public spring.webjson.webjson.models.Response<spring.data.entities.Produit>
    spring.webjson.server.service.Controller.getProduitByNameWithoutCategorie(java.lang.String)
44. 2015-03-24 16:22:52.656  INFO 9492 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped
    "{[/getProduitByNameWithCategorie/{nom}],methods=[GET],params=[],headers=[],consumes=[],produces=[],custom=[]}" onto
    public spring.webjson.webjson.models.Response<spring.data.entities.Produit>
    spring.webjson.server.service.Controller.getProduitByNameWithCategorie(java.lang.String)
45. 2015-03-24 16:22:52.656  INFO 9492 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped
    "{[/getProduitByIdWithCategorie/{idProduit}],methods=[GET],params=[],headers=[],consumes=[],produces=[],custom=[]}" onto
    public spring.webjson.webjson.models.Response<spring.data.entities.Produit>
    spring.webjson.server.service.Controller.getProduitByIdWithCategorie(java.lang.Long)
46. 2015-03-24 16:22:52.656  INFO 9492 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped
    "{[/getCategorieByNameWithProduits/{nom}],methods=[GET],params=[],headers=[],consumes=[],produces=[],custom=[]}" onto
    public spring.webjson.webjson.models.Response<spring.data.entities.Categorie>
    spring.webjson.server.service.Controller.getCategorieByNameWithProduits(java.lang.String)
47. 2015-03-24 16:22:52.657  INFO 9492 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped
    "{[/getCategorieByIdWithProduits/{idCategorie}],methods=[GET],params=[],headers=[],consumes=[],produces=[],custom=[]}"
    onto public spring.webjson.webjson.models.Response<spring.data.entities.Categorie>
    spring.webjson.server.service.Controller.getCategorieByIdWithProduits(java.lang.Long)
48. 2015-03-24 16:22:52.657  INFO 9492 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped
    "{[/deleteAllCategories],methods=[GET],params=[],headers=[],consumes=[],produces=[],custom=[]}" onto public
    spring.webjson.webjson.models.Response<java.lang.Void> spring.webjson.server.service.Controller.deleteAllCategories()
49. 2015-03-24 16:22:52.657  INFO 9492 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped
    "{[/getCategorieByIdWithoutProduits/{idCategorie}],methods=[GET],params=[],headers=[],consumes=[],produces=[],custom=[]}
    " onto public spring.webjson.webjson.models.Response<spring.data.entities.Categorie>
    spring.webjson.server.service.Controller.getCategorieByIdWithoutProduits(java.lang.Long)
50. 2015-03-24 16:22:52.657  INFO 9492 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped
    "{[/deleteAllProduits],methods=[GET],params=[],headers=[],consumes=[],produces=[],custom=[]}" onto public
    spring.webjson.webjson.models.Response<java.lang.Void> spring.webjson.server.service.Controller.deleteAllProduits()
51. 2015-03-24 16:22:52.658  INFO 9492 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped
    "{[/getProduitByIdWithoutCategorie/{idProduit}],methods=[GET],params=[],headers=[],consumes=[],produces=[],custom=[]}"
    onto public spring.webjson.webjson.models.Response<spring.data.entities.Produit>
    spring.webjson.server.service.Controller.getProduitByIdWithoutCategorie(java.lang.Long)
```

```
52. 2015-03-24 16:22:52.659  INFO 9492 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped
    "{[/error],methods=[],params=[],headers=[],consumes=[],produces=[],custom=[]}" onto public
    org.springframework.http.ResponseEntity<java.util.Map<java.lang.String, java.lang.Object>>
    org.springframework.boot.autoconfigure.web.BasicErrorController.error(javax.servlet.http.HttpServletRequest)
53. 2015-03-24 16:22:52.659  INFO 9492 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped
    "{[/error],methods=[],params=[],headers=[],consumes=[],produces=[text/html],custom=[]}" onto public
    org.springframework.web.servlet.ModelAndView
    org.springframework.boot.autoconfigure.web.BasicErrorController.errorHtml(javax.servlet.http.HttpServletRequest)
54. 2015-03-24 16:22:52.691  INFO 9492 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping  : Mapped URL path
    [/webjars/**] onto handler of type [class org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
55. 2015-03-24 16:22:52.692  INFO 9492 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping  : Mapped URL path [/**]
    onto handler of type [class org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
56. 2015-03-24 16:22:52.742  INFO 9492 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping  : Mapped URL path
    [/**/favicon.ico] onto handler of type [class org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
57. 2015-03-24 16:22:53.001  INFO 9492 --- [           main] o.s.j.e.a.AnnotationMBeanExporter        : Registering beans for
    JMX exposure on startup
58. 2015-03-24 16:22:53.106  INFO 9492 --- [           main] s.b.c.e.t.TomcatEmbeddedServletContainer : Tomcat started on
    port(s): 8080 (http)
59. 2015-03-24 16:22:53.108  INFO 9492 --- [           main] spring.webjson.server.boot.Boot          : Started Boot in 6.752
    seconds (JVM running for 7.433)
```

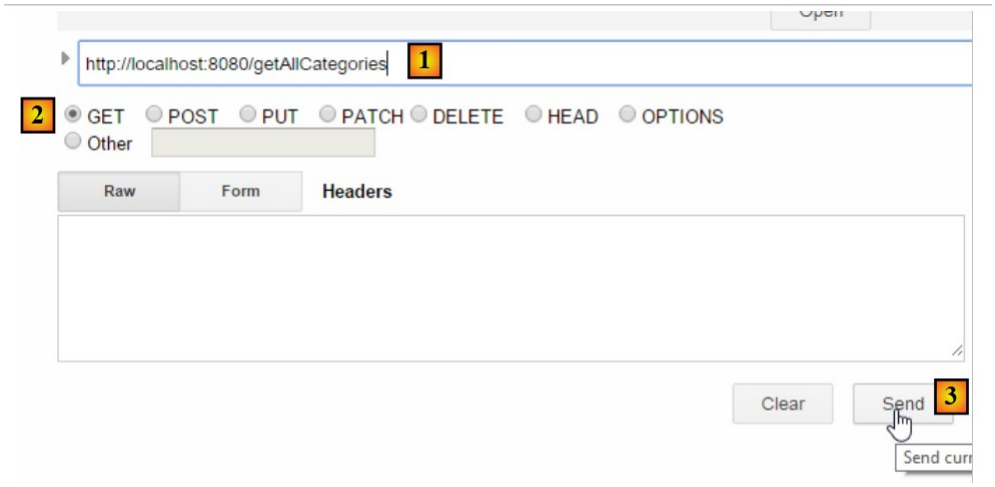
- lignes 17-19 : démarrage du serveur Tomcat qui va exécuter le service web / jSON ;
- lignes 25-33 : construction de la couche [DAO] ;
- lignes 32-51 : les URL exposées sont découvertes ;

13.5.7 Tests du service web / jSON

Pour faire les tests, nous générons la base de données MySQL [dbintrospringdata] à partir du script SQL [dbintrospringdata.sql] :



Ceci fait, nous utilisons le client [Advanced Rest Client] (cf page 419) pour interroger les URL exposées par le service web / jSON (le service web / jSON doit être lancé).



- en [1-3], nous demandons l'URL [/getAllCategories] via une commande HTTP GET ;

Nous obtenons la réponse suivante :

- en [1], la requête HTTP du client ;
- en [2], la réponse HTTP du serveur ;
- en [3], le statut [200 OK] indique que le serveur a correctement traité la demande ;
- en [4], la réponse jSON du serveur ;

La réponse jSON complète est la suivante :

```
1. {"status":0,"messages":null,"body":[{"id":415,"version":0,"nom":"categorie0","produits":
    [{"id":1849,"version":0,"nom":"produit00","idCategorie":415,"prix":100.0,"description":"desc00"},
    {"id":1850,"version":0,"nom":"produit01","idCategorie":415,"prix":101.0,"description":"desc01"},
    {"id":1851,"version":0,"nom":"produit02","idCategorie":415,"prix":102.0,"description":"desc02"},
    {"id":1852,"version":0,"nom":"produit03","idCategorie":415,"prix":103.0,"description":"desc03"},
    {"id":1853,"version":0,"nom":"produit04","idCategorie":415,"prix":104.0,"description":"desc04"}]},
    {"id":416,"version":0,"nom":"categorie1","produits":
    [{"id":1856,"version":0,"nom":"produit12","idCategorie":416,"prix":112.0,"description":"desc12"},
    {"id":1857,"version":0,"nom":"produit13","idCategorie":416,"prix":113.0,"description":"desc13"},
    {"id":1858,"version":0,"nom":"produit14","idCategorie":416,"prix":114.0,"description":"desc14"},
    {"id":1854,"version":0,"nom":"produit10","idCategorie":416,"prix":110.0,"description":"desc10"},
    {"id":1855,"version":0,"nom":"produit11","idCategorie":416,"prix":111.0,"description":"desc11"}]}]}
```

- **status:0** signifie qu'il n'y a pas eu d'erreurs côté serveur ;
- **messages : null** signifie qu'il n'y a pas de messages d'erreur ;
- **body** : est le corps de la réponse, ici la liste des catégories avec leurs produits. Il y a deux catégories avec chacune 5 produits ;

Nous allons ajouter à la catégorie [categorie1], le produit [produit15]. Pour cela nous allons utiliser l'URL [/addCategories] qui a le code suivant :

```
1. @RequestMapping(value = "/addCategories", method = RequestMethod.POST, consumes = "application/json; charset=UTF-8", produces
    = "application/json; charset=UTF-8")
2.   @ResponseBody
3.   public String addCategories(HttpServletRequest request) throws JsonProcessingException {
4.     Response<List<Categorie>> response;
5.     ObjectMapper mapper = context.getBean(ObjectMapper.class);
6.     // on persiste les catégories
7.     try {
8.         // on récupère la valeur postée
9.         String body = CharStreams.toString(request.getReader());
10.            mapper.setFilters(jsonFilterCategorieWithProduits);
11.            List<Categorie> categories = mapper.readValue(body, new TypeReference<List<Categorie>>() {
12.            });
13.            // on rétablit le lien entre produits et catégories
14.            for (Categorie categorie : categories) {
15.                Set<Produit> produits = categorie.getProduits();
```

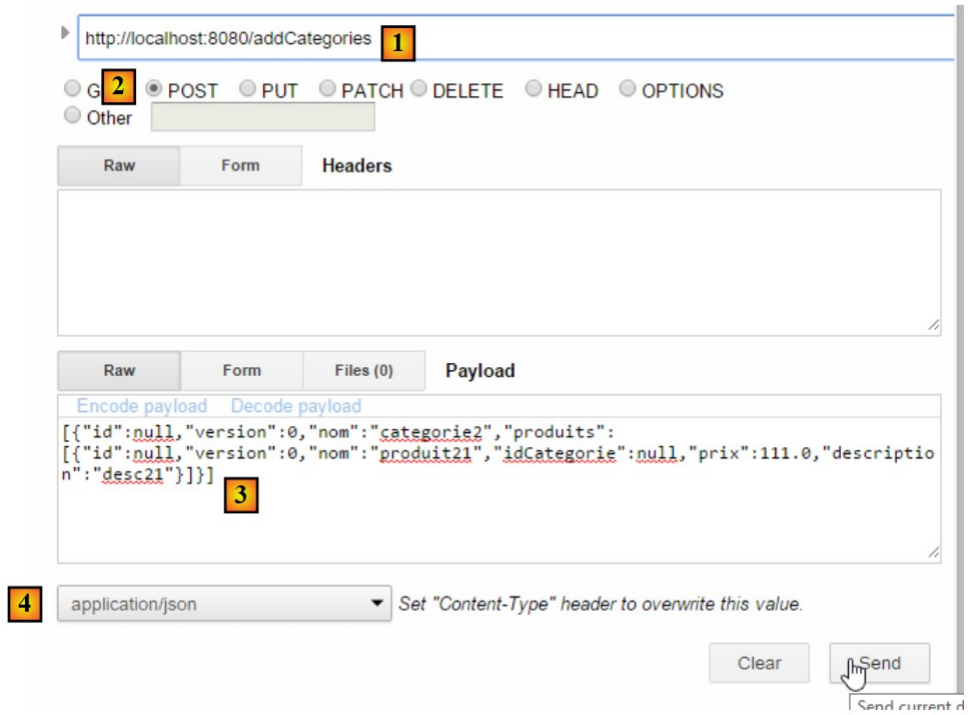
```
16.             if (produits != null) {
17.                 for (Produit produit : categorie.getProduits()) {
18.                     produit.setCategorie(categorie);
19.                 }
20.             }
21.         }
22.         // on persiste les catégories
23.         application.addCategories(categories);
24.         response = new Response<List<Categorie>>(0, null, categories);
25.       } catch (Exception e) {
26.         response = new Response<List<Categorie>>(1004, getErreursForException(e), null);
27.       }
28.       // réponse jSON
29.       return mapper.writeValueAsString(response);
30.   }
```

- ligne 1 : le client doit faire un POST et la valeur postée doit être une chaîne jSON ;
- lignes 9-12 : la valeur postée doit être une liste de catégories avec leurs produits associés ;

Nous allons créer une catégorie [categorie2] avec un produit [produit21]. La chaîne jSON à envoyer est alors la suivante :

```
[{"id":null,"version":0,"nom":"categorie2","produits":
[{"id":null,"version":0,"nom":"produit21","idCategorie":null,"prix":111.0,"description":"desc21"}]}]
```

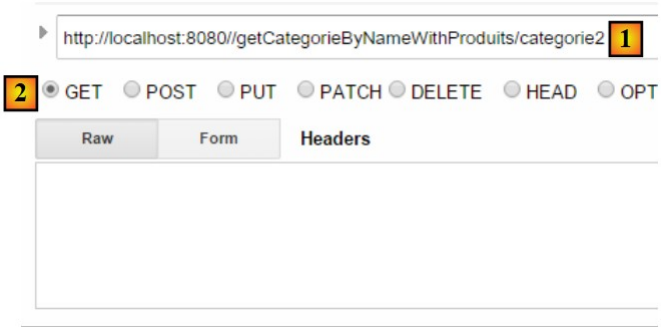
La requête au service web / jSON est faite de la façon suivante :



- en [1], l'URL demandée ;
- en [2], elle est demandée via une opération POST ;
- en [3], la chaîne jSON postée ;
- en [4], on indique au serveur qu'on va lui envoyer du jSON ;

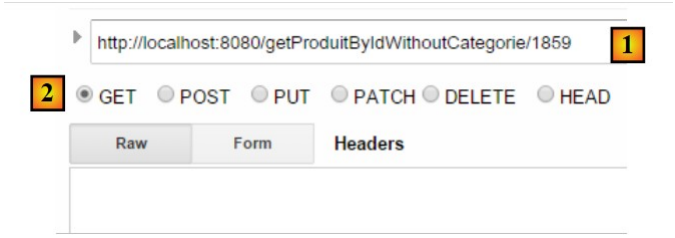
La réponse du serveur est la suivante :

- en [1], on voit que et la catégorie et son produit ont maintenant une clé primaire montrant par là qu'ils ont probablement été insérés dans la base. Nous allons le vérifier en utilisant l'URL [/getCategorieByNameWithProduits/categorie2] :



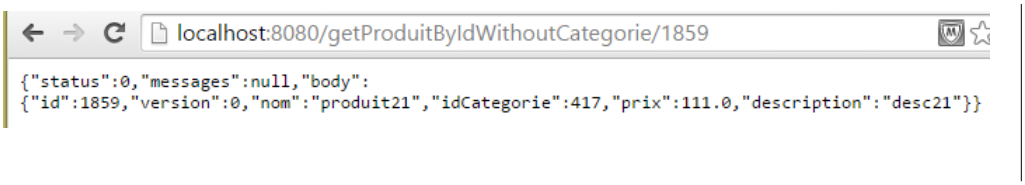
Nous obtenons le résultat suivant :

Nous avons bien obtenu la catégorie [categorie2] avec son unique produit [produit21]. On peut aussi demander uniquement le produit. Utilisons pour cela l'URL [/getProduitByIdWithoutCategorie/1859] :



Nous obtenons le résultat suivant :

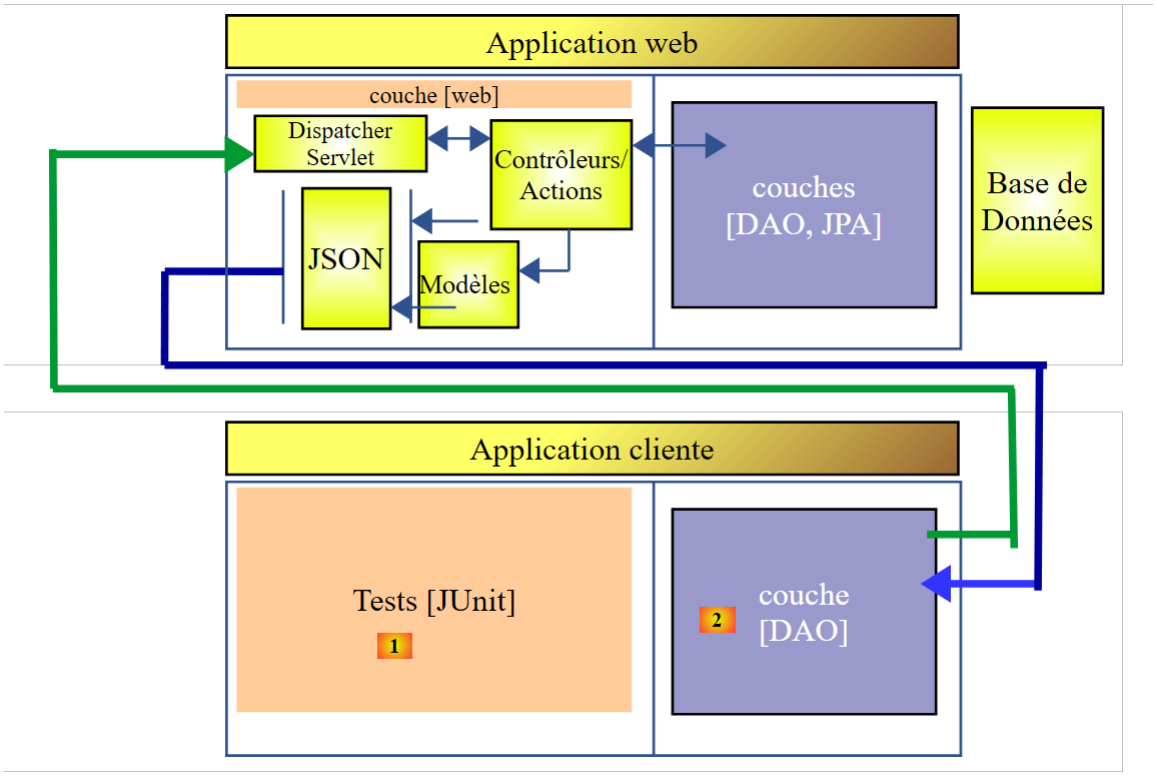
Toutes les opérations [GET] peuvent être faites dans un simple navigateur :



Le lecteur est invité à tester les autres URL du service web / json.

13.6 Un client programmé pour le service web / jSON

Maintenant que la base [dbintrospringdata] est disponible sur le web, nous allons écrire une application qui l'exploite. On aura alors l'architecture client / serveur suivante :

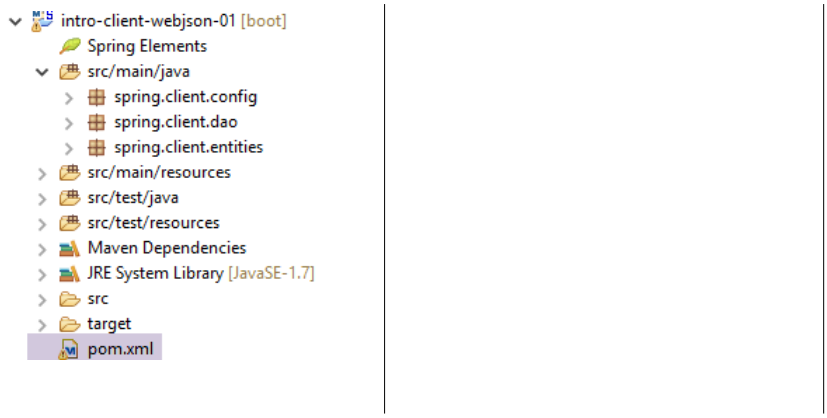


L'application cliente aura deux couches :

- une couche [DAO] [2] pour communiquer avec l'application web / jSON qui expose la base de données ;
- une couche de tests JUnit [1] pour vérifier que le client et le serveur font bien leur travail ;

13.6.1 Le projet Eclipse

Le projet Eclipse du client est le suivant :



- le dossier [src/main/java] implémente la couche [DAO] ;
- le dossier [src/test/java] implémente les tests JUnit ;

13.6.2 Configuration Maven du projet

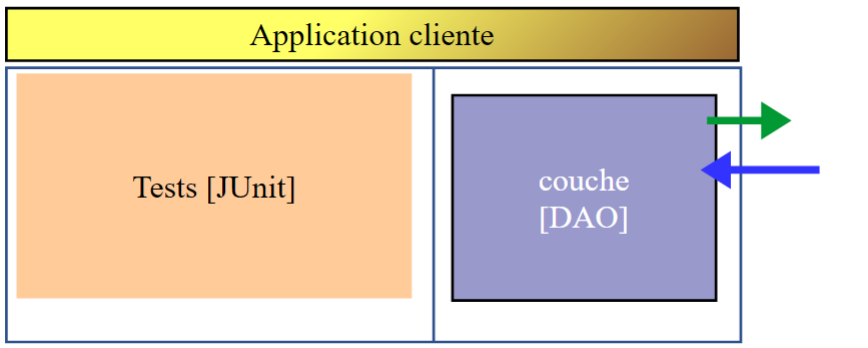
Le projet est un projet Maven configuré par le fichier [pom.xml] suivant :

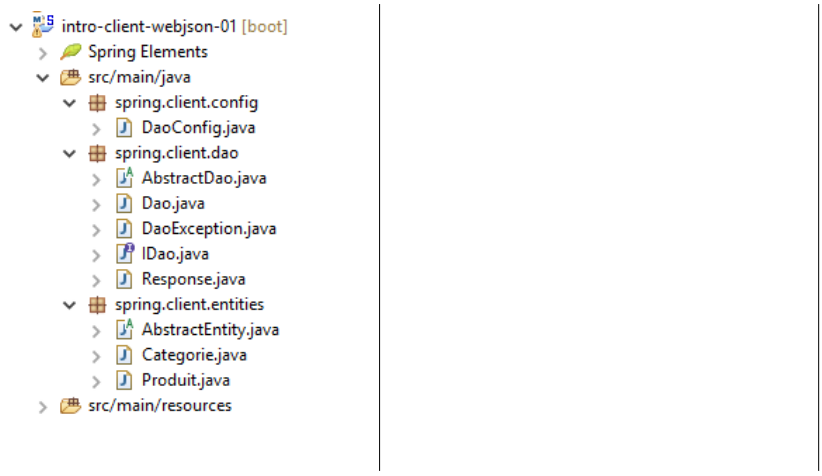
```
1. <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2.   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
3.   <modelVersion>4.0.0</modelVersion>
4.   <groupId>istia.st.webjson</groupId>
5.   <artifactId>intro-client-webjson-01</artifactId>
6.   <version>0.0.1-SNAPSHOT</version>
7.   <description>Client console du serveur web / jSON</description>
8.
9.   <properties>
10.       <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
11.       <java.version>1.8</java.version>
12.    </properties>
13.
14.    <parent>
15.       <groupId>org.springframework.boot</groupId>
16.       <artifactId>spring-boot-starter-parent</artifactId>
17.       <version>1.2.7.RELEASE</version>
18.    </parent>
19.
20.    <dependencies>
21.       <!-- Spring -->
22.       <dependency>
23.          <groupId>org.springframework</groupId>
24.          <artifactId>spring-web</artifactId>
25.       </dependency>
26.       <!-- librairie jSON utilisée par Spring -->
27.       <dependency>
28.          <groupId>com.fasterxml.jackson.core</groupId>
29.          <artifactId>jackson-core</artifactId>
30.       </dependency>
31.       <dependency>
32.          <groupId>com.fasterxml.jackson.core</groupId>
33.          <artifactId>jackson-databind</artifactId>
34.       </dependency>
35.       <!-- composant utilisé par Spring RestTemplate -->
36.       <dependency>
37.          <groupId>org.apache.httpcomponents</groupId>
38.          <artifactId>httpclient</artifactId>
39.       </dependency>
40.       <!-- Google Guava -->
41.       <dependency>
42.          <groupId>com.google.guava</groupId>
43.          <artifactId>guava</artifactId>
44.          <version>16.0.1</version>
45.          <scope>test</scope>
46.       </dependency>
47.       <!-- bibliothèque de logs -->
48.       <dependency>
49.          <groupId>org.springframework.boot</groupId>
50.          <artifactId>spring-boot-starter-logging</artifactId>
51.       </dependency>
52.       <!-- Spring Boot Test -->
53.       <dependency>
54.          <groupId>org.springframework.boot</groupId>
55.          <artifactId>spring-boot-starter-test</artifactId>
56.          <scope>test</scope>
```

```
57.         </dependency>
58.         <!-- Spring Boot -->
59.         <dependency>
60.             <groupId>org.springframework.boot</groupId>
61.             <artifactId>spring-boot</artifactId>
62.             <scope>test</scope>
63.         </dependency>
64.     </dependencies>
65.     <!-- plugins -->
66.     <build>
67.         <plugins>
68.             <plugin>
69.                 <artifactId>maven-assembly-plugin</artifactId>
70.                 <configuration>
71.                     <descriptorRefs>
72.                         <descriptorRef>jar-with-dependencies</descriptorRef>
73.                     </descriptorRefs>
74.                 </configuration>
75.             </plugin>
76.             <plugin>
77.                 <groupId>org.apache.maven.plugins</groupId>
78.                 <artifactId>maven-surefire-plugin</artifactId>
79.                 <version>2.18.1</version>
80.             </plugin>
81.         </plugins>
82.     </build>
83.
84.     <name>intro-client-webjson-01</name>
85. </project>
```

- lignes 14-18 : le projet Maven parent [`spring-boot-starter-parent`] qui nous permet de définir un certain nombre de dépendances sans leur versions, celle-ci étant définie dans le projet parent ;
- lignes 22-25 : bien que nous n'écrivions pas une application web, nous avons besoin de la dépendance [`spring-web`] qui amène avec elle la classe [RestTemplate] qui permet de s'interfacer aisément avec une application web / jSON ;
- lignes 27-34 : une bibliothèque jSON ;
- lignes 36-39 : une dépendance qui va nous permettre de fixer un *timeout* aux requêtes HTTP du client. Un *timeout* est un temps maximal d'attente de la réponse du serveur. Au-delà de ce temps, le client signale une erreur de *timeout* en jetant une exception ;
- lignes 41-46 : la bibliothèque Google Guava utilisée dans le test JUnit. Pour cette raison, nous avons mis sa portée à [test] (ligne 45). Cela signifie que cette dépendance n'est incluse que lors de l'exécution de codes de la branche [src/test/java] ;
- lignes 48-51 : la bibliothèque de logs ;
- lignes 52-63 : la dépendance pour les tests JUnit. Elle amène notamment la bibliothèque JUnit 4 nécessaire pour les tests. Ces dépendances ont l'attribut [<scope>test</scope>] indiquant qu'elles ne sont nécessaires que pour la phase de tests. Elles ne sont pas incluses dans l'archive finale du projet ;

13.6.3 Implémentation de la couche [DAO]





- le package [spring.client.config] contient la configation Spring de la couche [DAO] ;
- le package [spring.client.dao] contient l'implémentation de la couche [DAO] ;
- le package [spring.client.entities] contient les objets échangés avec le service web / jSON ;

13.6.3.1 Configuration

La classe [DaoConfig] fait la configuration Spring de la couche [DAO]. Son code est le suivant :

```
1. package spring.client.config;
2.
3. import org.springframework.context.annotation.Bean;
4. import org.springframework.context.annotation.ComponentScan;
5. import org.springframework.context.annotation.Configuration;
6. import org.springframework.http.client.HttpComponentsClientHttpRequestFactory;
7. import org.springframework.web.client.RestTemplate;
8.
9. import com.fasterxml.jackson.databind.ObjectMapper;
10.  import com.fasterxml.jackson.databind.ser.impl.SimpleBeanPropertyFilter;
11.  import com.fasterxml.jackson.databind.ser.impl.SimpleFilterProvider;
12.
13.  @ComponentScan({ "spring.client.dao" })
14.  public class DaoConfig {
15.
16.    // constantes
17.    static private final int TIMEOUT = 1000;
18.    static private final String URL_WEBJSON = "http://localhost:8080";
19.
20.    @Bean
21.    public RestTemplate restTemplate(int timeout) {
22.        // création du composant RestTemplate
23.        HttpComponentsClientHttpRequestFactory factory = new HttpComponentsClientHttpRequestFactory();
24.        RestTemplate restTemplate = new RestTemplate(factory);
25.        // timeout des échanges
26.        factory.setConnectTimeout(timeout);
27.        factory.setReadTimeout(timeout);
28.        // résultat
29.        return restTemplate;
30.    }
31.
32.    @Bean
33.    public int timeout() {
34.        return TIMEOUT;
35.    }
36.
```

```
37.    @Bean
38.    public String urlWebJson() {
39.        return URL_WEBJSON;
40.    }
41.
42.    // filtres jSON
43.    @Bean(name = "jsonMapper")
44.    public ObjectMapper jsonMapper() {
45.        return new ObjectMapper();
46.    }
47.
48.    @Bean(name = "jsonMapperCategorieWithProduits")
49.    public ObjectMapper jsonMapperCategorieWithProduits() {
50.        // mappeur jSON
51.        ObjectMapper mapper = new ObjectMapper();
52.        // filtres
53.        mapper.setFilters(
54.                new SimpleFilterProvider().addFilter("jsonFilterCategorie", SimpleBeanPropertyFilter.serializeAllExcept())
55.                        .addFilter("jsonFilterProduit", SimpleBeanPropertyFilter.serializeAllExcept("categorie")));
56.        // résultat
57.        return mapper;
58.    }
59.
60.    @Bean(name = "jsonMapperProduitWithCategorie")
61.    public ObjectMapper jsonMapperProduitWithCategorie() {
62.        // mappeur jSON
63.        ObjectMapper mapper = new ObjectMapper();
64.        // filtres
65.        mapper.setFilters(
66.                new SimpleFilterProvider().addFilter("jsonFilterProduit", SimpleBeanPropertyFilter.serializeAllExcept())
67.                        .addFilter("jsonFilterCategorie", SimpleBeanPropertyFilter.serializeAllExcept("produits")));
68.        // résultat
69.        return mapper;
70.    }
71.
72.    @Bean(name = "jsonMapperCategorieWithoutProduits")
73.    public ObjectMapper jsonMapperCategorieWithoutProduits() {
74.        // mappeur jSON
75.        ObjectMapper mapper = new ObjectMapper();
76.        // filtres
77.        mapper.setFilters(new SimpleFilterProvider().addFilter("jsonFilterCategorie",
78.                SimpleBeanPropertyFilter.serializeAllExcept("produits")));
79.        // résultat
80.        return mapper;
81.    }
82.
83.    @Bean(name = "jsonMapperProduitWithoutCategorie")
84.    public ObjectMapper jsonMapperProduitWithoutCategorie() {
85.        // mappeur jSON
86.        ObjectMapper mapper = new ObjectMapper();
87.        // filtres
88.        mapper.setFilters(new SimpleFilterProvider().addFilter("jsonFilterProduit",
89.                SimpleBeanPropertyFilter.serializeAllExcept("categorie")));
90.        // résultat
91.        return mapper;
92.    }
93. }
```

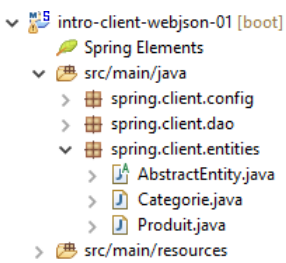
- ligne 13 : la classe est une classe de configuration Spring - des composants Spring sont à chercher dans le package [spring.client.dao] ;
- ligne 17 : on se fixe un *timeout* d'une seconde (1000 ms) ;
- lignes 32-35 : le bean qui rend cette valeur ;
- ligne 18 : l'URL du service web / jSON ;
- lignes 37-40 : le bean qui rend cette valeur ;
- lignes 20-30 : la configuration de la classe [RestTemplate] qui assure les échanges avec le service web / jSON. Lorsqu'on n'a pas à la configurer, on peut en disposer dans le code par un simple [new RestTemplate()]. Ici, nous voulons fixer le *timeout* des échanges avec le service web / jSON. Le bean [timeout] de la ligne 36 est passé en paramètre de la méthode [restTemplate] de la ligne 24 ;
- ligne 23 : le composant [HttpComponentsClientHttpRequestFactory] est le composant qui nous permet de fixer le *timeout* des échanges (lignes 29-30) ;
- ligne 24 : la classe [RestTemplate] est construite avec ce composant. Comme elle s'appuie sur celui-ci pour communiquer avec le service web / jSON, les échanges seront bien soumis au *timeout* ;
- le client et le serveur vont s'échanger des lignes de texte. Un convertisseur s'occupe de sérialiser un objet en texte et inversement de désérialiser un texte en objet. Il peut y avoir plusieurs convertisseurs associés à la classe [RestTemplate] et celui choisi à un moment donné dépend des entêtes HTTP envoyés par le serveur. Ici, nous n'aurons aucun convertisseur. Aussi, le composant [RestTemplate] ne cherchera pas à convertir d'une façon ou d'une autre les deux éléments suivants :
  - le texte posté ;
  - le texte reçu en réponse ;

Ces textes seront des chaînes jSON qui seront donc laissées en l'état par le composant [RestTemplate]. C'est nous développeur, qui ferons les séralisations / désérialisations jSON nécessaires. Ceci parce que les filtres à appliquer à la valeur postée et à la réponse reçue peuvent être différents et l'expérience montre qu'il est plus facile de les gérer soi-même que d'essayer de configurer le composant [RestTemplate] afin qu'il utilise le bon convertisseur jSON ;

- lignes 42-92 : définissent des filtres jSON. Ce sont les mêmes que ceux du serveur présentés et expliqués au paragraphe 13.5.3.1, page 228 ;
- lignes 43-46 : un mappeur jSON sans filtres ;
- lignes 64-68 : un mappeur jSON pour avoir une catégorie sans ses produits ;
- lignes 48-58 : un mappeur jSON pour avoir une catégorie avec ses produits ;
- lignes 83-92 : un mappeur jSON pour avoir un produit sans sa catégorie ;
- lignes 60-70 : un mappeur jSON pour avoir un produit avec sa catégorie ;

Tous ces beans vont être disponibles aux codes de la couche [DAO] ainsi qu'au test Junit.

13.6.3.2 Les entités



Les entités manipulées par la couche [DAO] sont celles qu'elle échange avec le service web / jSON. Ce sont les articles et les produits. Côté serveur, ces entités avaient des annotations de persistence JPA. Ici, ces annotations ont été enlevées. Nous redonnons le code des entités pour rappel :

[**AbstractEntity**]

```
1. package spring.client.entities;
2.
3. import com.fasterxml.jackson.core.JsonProcessingException;
4. import com.fasterxml.jackson.databind.ObjectMapper;
5.
6. public abstract class AbstractEntity {
7.   // propriétés
8.   protected Long id;
9.   protected Long version;
10.
11.    // constructeurs
12.    public AbstractEntity() {
13.
14.    }
15.
16.    public AbstractEntity(Long id, Long version) {
17.        this.id = id;
18.        this.version = version;
19.    }
20.
21.    // redéfinition [equals] et [hashcode]
22.    @Override
23.    public int hashCode() {
24.        return (id != null ? id.hashCode() : 0);
25.    }
26.
27.    @Override
28.    public boolean equals(Object entity) {
29.        if (!(entity instanceof AbstractEntity)) {
30.            return false;
31.        }
32.        String class1 = this.getClass().getName();
33.        String class2 = entity.getClass().getName();
34.        if (!class2.equals(class1)) {
35.            return false;
36.        }
37.        AbstractEntity other = (AbstractEntity) entity;
38.        return id != null && this.id == other.id.longValue();
```

```
    }

    // signature jSON
    public String toString() {
        ObjectMapper mapper = new ObjectMapper();
        try {
            return mapper.writeValueAsString(this);
        } catch (JsonProcessingException e) {
            e.printStackTrace();
            return null;
        }
    }

    // getters et setters
...
}
```

[**Categorie**]

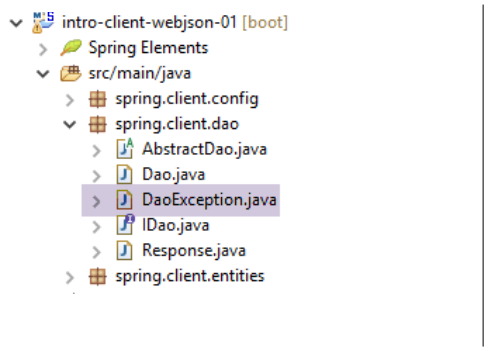
```
package spring.client.entities;

import java.util.HashSet;
import java.util.Set;

import com.fasterxml.jackson.annotation.JsonFilter;

@JsonFilter("jsonFilterCategorie")
public class Categorie extends AbstractEntity {

    // propriétés
    private String nom;

    // les produits associés
    public Set<Produit> produits = new HashSet<Produit>();

    // constructeurs
    public Categorie() {

    }

    public Categorie(String nom) {
        this.nom = nom;
    }

    // méthodes
    public void addProduit(Produit produit) {
        // on ajoute le produit
        produits.add(produit);
        // on fixe sa catégorie
        produit.setCategorie(this);
    }

    // getters et setters
    ...
}
```

[**Produit**]

```
package spring.webjson.client.entities;

import com.fasterxml.jackson.annotation.JsonFilter;

@JsonFilter("jsonFilterProduit")
public class Produit extends AbstractEntity {

   // le nom
   private String nom;
    // le n° de la catégorie
    private Long idCategorie;
    // le prix
    private double prix;
    // la description
    private String description;

    // la catégorie
    private Categorie categorie;

    // constructeurs
    public Produit() {

    }

    public Produit(String nom, double prix, String description) {
        this.nom = nom;
```

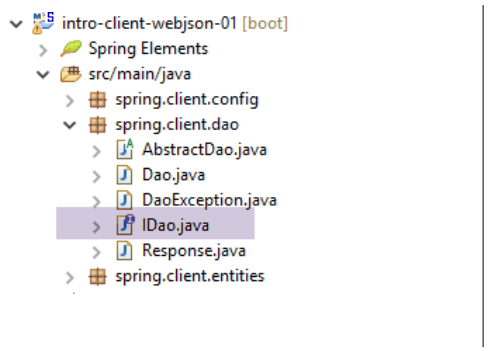
```
27.         this.prix = prix;
28.         this.description = description;
29.     }
30.
31.     // getters et setters
32.  ...
33.  }
```

13.6.3.3 La classe [DaoException]



Lorsque la couche [DAO] rencontrera une erreur, elle lancera une exception de type [DaoException]. Cette classe est celle utilisée côté serveur et décrite au paragraphe 11.3.7, page 191.

13.6.3.4 L'interface de la couche [DAO]



La couche [DAO] présente l'interface [IDao] décrite au paragraphe 11.3.7, page 191.

```
1. package spring.client.dao;
2.
3. import java.util.List;
4.
5. import spring.client.entities.Categorie;
6. import spring.client.entities.Produit;
7.
8. public interface IDao {
9.
10.     // insertion d'une liste de produits
11.     public List<Produit> addProduits(List<Produit> produits);
12.
13.     // suppression de tous les produits
14.     public void deleteAllProduits();
15.
16.     // mise à jour d'une liste de produits
17.     public List<Produit> updateProduits(List<Produit> produits);
18.
19.     // obtention de tous les produits
20.     public List<Produit> getAllProduits();
21.
22.     // insertion d'une liste de categories
23.     public List<Categorie> addCategories(List<Categorie> categories);
24.
25.     // suppression de tous les categories
26.     public void deleteAllCategories();
27.
28.     // mise à jour d'une liste de categories
```

```
29.    public List<Categorie> updateCategories(List<Categorie> categories);
30.
31.    // obtention de tous les categories
32.    public List<Categorie> getAllCategories();
33.
34.    // un produit particulier
35.    public Produit getProduitByIdWithCategorie(Long idProduit);
36.
37.    public Produit getProduitByIdWithoutCategorie(Long idProduit);
38.
39.    public Produit getProduitByNameWithCategorie(String nom);
40.
41.    public Produit getProduitByNameWithoutCategorie(String nom);
42.
43.    // une catégorie particulière
44.    public Categorie getCategorieByIdWithProduits(Long idCategorie);
45.
46.    public Categorie getCategorieByIdWithoutProduits(Long idCategorie);
47.
48.    public Categorie getCategorieByNameWithProduits(String nom);
49.
50.    public Categorie getCategorieByNameWithoutProduits(String nom);
51.
52. }
```

13.6.3.5 La réponse du service web / jSON

Nous avons vu que toutes les URL du service web / jSON rendaient un type [Response] défini au paragraphe 13.5.5.3, page 235. Nous reprenons ici cette classe :

```
1. package spring.client.dao;
2.
3. import java.util.List;
4.
5. public class Response<T> {
6.
7.   // ----------------- propriétés
8.   // statut de l'opération
9.   private int status;
10.    // les éventuels messages d'erreur
11.    private List<String> messages;
12.    // le corps de la réponse
13.    private T body;
14.
15.    // constructeurs
16.    public Response() {
17.
18.    }
19.
20.    public Response(int status, List<String> messages, T body) {
21.        this.status = status;
22.        this.messages = messages;
23.        this.body = body;
24.    }
25.
26.    // getters et setters
27.    ...
28. }
```

13.6.3.6 Implémentation des échanges avec le service web / jSON

La classe [AbstractDao] implémente les échanges avec le service web / jSON :

```
1. package spring.client.dao;
2.
3. import java.net.URI;
4. import java.net.URISyntaxException;
5.
6. import org.springframework.beans.factory.annotation.Autowired;
7. import org.springframework.core.ParameterizedTypeReference;
8. import org.springframework.http.MediaType;
9. import org.springframework.http.RequestEntity;
10.  import org.springframework.web.client.RestTemplate;
11.
12.  public abstract class AbstractDao {
13.
14.    // data
15.    @Autowired
16.    protected RestTemplate restTemplate;
17.    @Autowired
18.    protected String urlServiceWebJson;
19.
20.    // requête générique
21.    protected String getResponse(String url, String jsonPost) {
22.
23.        // url : URL à contacter
24.        // jsonPost : la valeur jSON à poster
25.        try {
26.            // exécution requête
27.            RequestEntity<?> request;
28.            if (jsonPost != null) {
29.                // requête POST
30.                request = RequestEntity.post(new URI(String.format("%s%s", urlServiceWebJson, url)))
31.                        .header("Content-Type", "application/json").accept(MediaType.APPLICATION_JSON).body(jsonPost);
32.            } else {
33.                // requête GET
34.                request = RequestEntity.get(new URI(String.format("%s%s", urlServiceWebJson, url)))
35.                        .accept(MediaType.APPLICATION_JSON).build();
36.            }
37.            // on exécute la requête
38.            return restTemplate.exchange(request, new ParameterizedTypeReference<String>() {
39.            }).getBody();
40.        } catch (URISyntaxException e1) {
41.            throw new DaoException(20, e1);
42.        } catch (RuntimeException e2) {
43.            throw new DaoException(21, e2);
44.        }
45.    }
46.
47.  }
48.
```

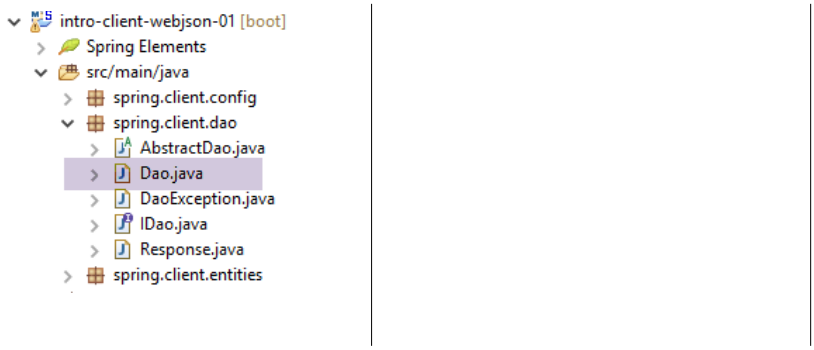
- lignes 15-16 : injection du composant [RestTemplate] qui assure la communication avec le serveur ;
- lignes 17-18 : injection de l'URL du service web / jSON ;

L'implémentation des méthodes de communication avec le serveur est factorisée dans la méthode [getResponse] :

- ligne 21 : la méthode reçoit 2 paramètres :
  - [url] : l'URL demandée ;
  - [jsonPost] : la chaîne jSON à poster, *null* sinon. Si [jsonPost==null], la requête de l'URL est faite avec un GET, sinon avec un POST ;

- ligne 38 : l'instruction qui fait la requête au serveur et reçoit sa réponse. Le composant [RestTemplate] offre un nombre important de méthodes d'échange avec le serveur. Nous avons choisi ici la méthode [exchange], mais il en existe d'autres ;
- lignes 27-36 : il nous faut construire la requête de type [RequestEntity]. Elle est différente selon que l'on utilise un GET ou un POST pour faire la requête ;
- lignes 30-31 : la requête pour un GET. La classe [RequestEntity] offre des méthodes statiques pour créer les requêtes GET, POST, HEAD,... La méthode [RequestEntity.get] permet de créer une requête GET en chaînant les différentes méthodes qui construisent celle-ci :
  - la méthode [RequestEntity.get] admet pour paramètre l'URL cible sous la forme d'une instance URI,
  - la méthode [accept] permet de définir les éléments de l'entête HTTP [Accept]. Ici, nous indiquons que nous acceptons le type [application/json] que va envoyer le serveur ;
  - la méthode [build] utilise ces différentes informations pour construire le type [RequestEntity] de la requête ;
- lignes 34-35 : la requête pour un POST. La méthode [RequestEntity.post] permet de créer une requête POST en chaînant les différentes méthodes qui construisent celle-ci :
  - la méthode [RequestEntity.post] admet pour paramètre l'URL cible sous la forme d'une instance URI,
  - la méthode [header] définit un entête HTTP. Ici on envoie au serveur l'entête [Content-Type: application/json] pour lui indiquer que la valeur postée va lui arriver sous la forme d'une chaîne jSON ;
  - la méthode [accept] permet d'indiquer que nous acceptons le type [application/json] que va envoyer le serveur ;
  - la méthode [body] fixe la valeur postée. Celle-ci est le 4ième paramètre de la méthode générique [getResponse] (ligne 1) ;
- ligne 38 : la méthode [RestTemplate].exchange rend un type [ResponseEntity<String>] qui encapsule la totalité de la réponse du serveur : entêtes HTTP et corps du document. La méthode [ResponseEntity].getBody() permet d'avoir ce corps qui représente la réponse du serveur, ici une chaîne de caractères ;

13.6.3.7 Implémentation de l'interface [IDao]



La classe [Dao] implémente l'interface [IDao] :

```
1. package spring.client.dao;
2.
3. import java.io.IOException;
4. import java.util.List;
5.
6. import org.springframework.beans.factory.annotation.Autowired;
7. import org.springframework.beans.factory.annotation.Qualifier;
8. import org.springframework.context.ApplicationContext;
9. import org.springframework.stereotype.Component;
10.
11.  import com.fasterxml.jackson.core.type.TypeReference;
12.  import com.fasterxml.jackson.databind.ObjectMapper;
13.
14.  import spring.client.entities.Categorie;
15.  import spring.client.entities.Produit;
16.
17.  @Component
18.  public class Dao extends AbstractDao implements IDao {
19.
20.    @Autowired
21.    private ApplicationContext context;
22.
23.    // filtres jSON
24.    @Autowired
25.    @Qualifier("jsonMapper")
26.    private ObjectMapper jsonMapper;
27.    @Autowired
28.    @Qualifier("jsonMapperCategorieWithProduits")
29.    private ObjectMapper jsonMapperCategorieWithProduits;
30.    @Autowired
31.    @Qualifier("jsonMapperProduitWithCategorie")
```

```
32.    private ObjectMapper jsonMapperProduitWithCategorie;
33.    @Autowired
34.    @Qualifier("jsonMapperCategorieWithoutProduits")
35.    private ObjectMapper jsonMapperCategorieWithoutProduits;
36.    @Autowired
37.    @Qualifier("jsonMapperProduitWithoutCategorie")
38.    private ObjectMapper jsonMapperProduitWithoutCategorie;
39.
40.    @Override
41.    public List<Produit> addProduits(List<Produit> produits) {
42.        // ----------- ajouter des produits (sans leur catégorie)
43.        ...
44.  }
```

- ligne 17 : la classe [Dao] est un composant Spring dans lequel on peut donc injecter d'autres composants Spring ;
- ligne 18 : la classe [Dao] étend la classe [AbstractDao] que nous venons de voir et implémente l'interface [IDao] ;
- lignes 20-21 : on injecte le contexte Spring afin d'avoir accès à ses beans ;
- lignes 24-38 : injection des mappeurs jSON définis dans la classe [AppConfig] présentée au paragraphe 13.6.2, page 246 ;

Les implémentations des différentes méthodes de l'interface [IDao] suivent toutes le même schéma. Nous allons présenter deux méthodes, l'une s'appuyant sur une opération [POST], l'autre sur une opération [GET].

Un exemple de [GET] : [getCategorieByNameWithProduits]

```
1. @Override
2.   public Categorie getCategorieByNameWithProduits(String nom) {
3.     // ----------- obtenir une catégorie désignée par son nom, avec ses produits
4.     try {
5.         // requête
6.         Response<Categorie> response = jsonMapperCategorieWithProduits.readValue(
7.                 getResponse(String.format("/getCategorieByNameWithProduits/%s", nom), null),
8.                 new TypeReference<Response<Categorie>>() {
9.                 });
10.           // erreur ?
11.           if (response.getStatus() != 0) {
12.               // on lance 1 exception
13.               throw new DaoException(response.getStatus(), response.getMessages());
14.           } else {
15.               // on rend le coeur de la réponse du serveur
16.               return response.getBody();
17.           }
18.       } catch (DaoException e1) {
19.           throw e1;
20.       } catch (RuntimeException | IOException e2) {
21.           throw new DaoException(113, e2);
22.       }
23.   }
```

- ligne 7 : on appelle la méthode [getResponse] de la classe parent. C'est cette méthode qui assure les échanges avec le service web / jSON. Ses paramètres sont les suivants :

```
getResponse(String.format("/getCategorieByNameWithProduits/%s", nom), null)
```

  - l'URL du service interrogée [/getCategorieByNameWithProduits/nom] ;
  - la valeur postée. Ici il n'y en a pas ;

  La méthode [getResponse] rend un type String qui est la réponse jSON envoyée par le serveur. On désérialise cette réponse jSON de la façon suivante :

```
1. jsonMapperCategorieWithProduits.readValue(
2.                 jsonResponse,
3.                 new TypeReference<Response<Categorie>>() {
4. });
```

  parce que la chaîne jSON est la sérialisation d'un type [Response<Categorie>] ;
- lignes 11-17 : on teste le statut de la réponse. Si le statut est différent de 0, alors c'est qu'il y a eu une erreur côté serveur. On lance alors une exception (ligne 13), en reprenant les informations contenues dans la réponse (statut et liste de messages d'erreur) ;
- ligne 16 : s'il n'y a pas eu d'erreur côté serveur, on rend le corps du type [Response<Categorie>], ç-à-d la catégorie demandée ;
- lignes 18-19 : on gère l'exception lancée ligne 16 ;
- lignes 20-22 : traitent toutes les autres exceptions ;

Un exemple de [POST] : [addCategories]

```
1. @Override
2.   public List<Categorie> addCategories(List<Categorie> categories) {
3.     // ---------- ajouter des catégories (avec leurs produits)
4.     try {
5.         // requête
6.         Response<List<Categorie>> response = jsonMapperCategorieWithProduits.readValue(
7.                 getResponse("/addCategories", jsonMapperCategorieWithProduits.writeValueAsString(categories)),
8.                 new TypeReference<Response<List<Categorie>>>() {
9.                 });
10.            // erreur ?
11.            if (response.getStatus() != 0) {
12.                // on lance 1 exception
13.                throw new DaoException(response.getStatus(), response.getMessages());
14.            } else {
15.                // on rend le coeur de la réponse du serveur
16.                return response.getBody();
17.            }
18.        } catch (DaoException e1) {
19.            throw e1;
20.        } catch (RuntimeException | IOException e2) {
21.            throw new DaoException(104, e2);
22.        }
23.    }
```

- ligne 2 : la méthode [addCategories] sert à persister en base de données les catégories passées en paramètre. Elle rend ces mêmes catégories enrichies de leurs clés primaires. Si les catégories sont passées avec des produits, ceux-ci sont également persistés ;
- ligne 7 : on appelle la méthode [getResponse] du parent pour faire les échanges avec le service web / jSON ;
  - le 1er paramètre est l'URL [/addCategories] ;
  - le second paramètre est la valeur postée, ici la liste des catégories à persister ;

```
getResponse("/addCategories", jsonMapperCategorieWithProduits.writeValueAsString(categories))
```

La chaîne jSON obtenue est ensuite désérialisée pour obtenir le type [Response<List<Categorie>] attendu :

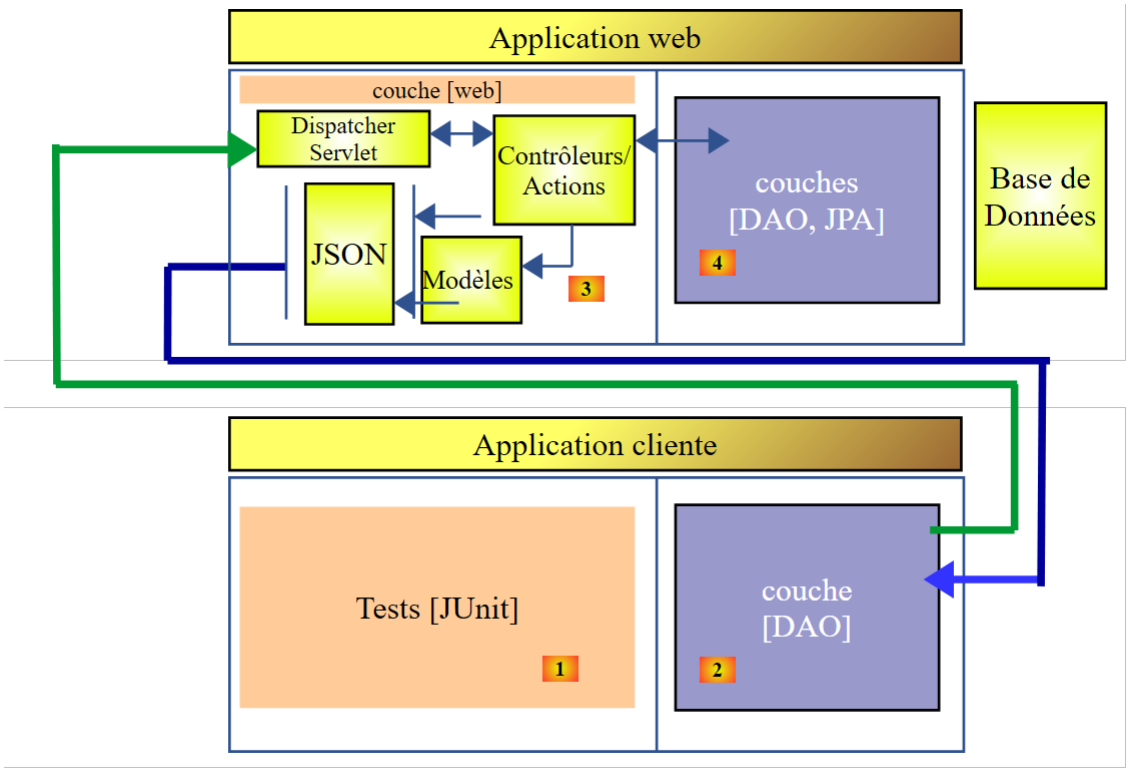
```
Response<List<Categorie>> response = jsonMapperCategorieWithProduits.readValue(
                jsonResponse,
                new TypeReference<Response<List<Categorie>>>() {
                });
```

- lignes 11-17 : gestion de la réponse du serveur (erreur ou pas) ;
- lignes 20-22 : gestion des exceptions ;

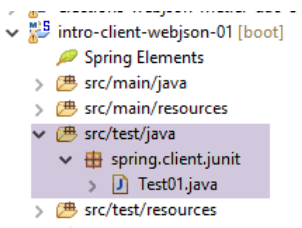
Toutes les autres méthodes suivent le canevas des deux méthodes présentées.

13.6.4 Le test JUnit

Revenons à l'architecture client / serveur en cours de construction :



Nous avons construit une couche [DAO] [2] avec la même interface que la couche [DAO] [4]. Pour tester la couche [DAO] [2], on peut donc utiliser le test JUnit qui a servi à tester la couche [DAO] [4]. Pour rappel, celui-ci est le suivant :




```
1. package spring.client.junit;
2.
3. import java.util.ArrayList;
4. import java.util.List;
5. import java.util.Set;
6.
7. import org.junit.Assert;
8. import org.junit.Before;
9. import org.junit.Test;
10.  import org.junit.runner.RunWith;
11.  import org.springframework.beans.BeansException;
12.  import org.springframework.beans.factory.annotation.Autowired;
13.  import org.springframework.beans.factory.annotation.Qualifier;
14.  import org.springframework.boot.test.SpringApplicationConfiguration;
15.  import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
16.
17.  import com.fasterxml.jackson.core.JsonProcessingException;
18.  import com.fasterxml.jackson.databind.ObjectMapper;
19.  import com.google.common.collect.Lists;
20.
21.  import spring.client.config.DaoConfig;
22.  import spring.client.dao.DaoException;
23.  import spring.client.dao.IDao;
24.  import spring.client.entities.Categorie;
25.  import spring.client.entities.Produit;
26.
27.  @SpringApplicationConfiguration(classes = DaoConfig.class)
28.  @RunWith(SpringJUnit4ClassRunner.class)
29.  public class Test01 {
```

```
30.
31.     // couche [DAO]
32.     @Autowired
33.     private IDao dao;
34.
35.     // filtres jSON
36.     @Autowired
37.     @Qualifier("jsonMapper")
38.     private ObjectMapper jsonMapper;
39.     @Autowired
40.     @Qualifier("jsonMapperCategorieWithProduits")
41.     private ObjectMapper jsonMapperCategorieWithProduits;
42.     @Autowired
43.     @Qualifier("jsonMapperProduitWithCategorie")
44.     private ObjectMapper jsonMapperProduitWithCategorie;
45.     @Autowired
46.     @Qualifier("jsonMapperCategorieWithoutProduits")
47.     private ObjectMapper jsonMapperCategorieWithoutProduits;
48.     @Autowired
49.     @Qualifier("jsonMapperProduitWithoutCategorie")
50.     private ObjectMapper jsonMapperProduitWithoutCategorie;
51.
52.     @Before
53.     public void cleanAndFill() {
54.         // on nettoie la base avant chaque test
55.         log("Vidage de la base de données", 1);
56.         // on vide la table [CATEGORIES] - par cascade la table [PRODUITS] va être vidée
57.         dao.deleteAllCategories();
58.         // ----------------------------------------------------------------------------------
59.         log("Remplissage de la base", 1);
60.         // on remplit les tables
61.         List<Categorie> categories = new ArrayList<Categorie>();
62.         for (int i = 0; i < 2; i++) {
63.             Categorie categorie = new Categorie(String.format("categorie%d", i));
64.             for (int j = 0; j < 5; j++) {
65.                 categorie.addProduit(new Produit(String.format("produit%d%d", i, j), 100 * (1 + (double) (i * 10 + j) / 100),
66.                         String.format("desc%d%d", i, j)));
67.             }
68.             categories.add(categorie);
69.         }
70.         // ajout de la catégorie - par cascade les produits vont eux aussi être insérés
71.         categories = dao.addCategories(categories);
72.     }
73.
74.     @Test
75.     public void showDataBase() throws BeansException, JsonProcessingException {
76.         // liste des catégories
77.         log("Liste des catégories", 2);
78.         List<Categorie> categories = dao.getAllCategories();
79.         affiche(categories, jsonMapperCategorieWithoutProduits);
80.         // liste des produits
81.         log("Liste des produits", 2);
82.         List<Produit> produits = dao.getAllProduits();
83.         affiche(produits, jsonMapperProduitWithoutCategorie);
84.         // quelques vérifications
85.         Assert.assertEquals(2, categories.size());
86.         Assert.assertEquals(10, produits.size());
87.         Categorie categorie = findCategorieByName("categorie0", categories);
88.         Assert.assertNotNull(categorie);
89.         Produit produit = findProduitByName("produit03", produits);
90.         Assert.assertNotNull(produit);
91.         Long idCategorie = produit.getIdCategorie();
92.         Assert.assertEquals(categorie.getId(), idCategorie);
93.     }
94.
95.     @Test
96.     public void getCategorieByNameWithProduits() {
97.         log("getCategorieByNameWithProduits", 1);
98.         Categorie categorie1 = dao.getCategorieByNameWithProduits("categorie1");
99.         Assert.assertNotNull(categorie1);
100.        Assert.assertEquals(5, categorie1.getProduits().size());
101.    }
102.
103.    @Test
104.    public void getCategorieByNameWithoutProduits() {
105.        log("getCategorieByNameWithoutProduits", 1);
106.        Categorie categorie1 = dao.getCategorieByNameWithoutProduits("categorie1");
107.        Assert.assertNotNull(categorie1);
108.        Assert.assertEquals("categorie1", categorie1.getNom());
109.    }
110.
111.    @Test
112.    public void getCategorieByIdWithProduits() {
113.        log("getCategorieByIdWithProduits", 1);
114.        Categorie categorie1 = dao.getCategorieByNameWithProduits("categorie1");
115.        Categorie categorie2 = dao.getCategorieByIdWithProduits(categorie1.getId());
116.        Assert.assertNotNull(categorie2);
```

```
117.      Assert.assertEquals(categorie1.getId(), categorie2.getId());
118.      Assert.assertEquals(categorie1.getNom(), categorie2.getNom());
119.  }
120.
121.  @Test
122.  public void getCategorieByIdWithoutProduits() {
123.      log("getCategorieByIdWithoutProduits", 1);
124.      Categorie categorie1 = dao.getCategorieByNameWithProduits("categorie1");
125.      Categorie categorie2 = dao.getCategorieByIdWithoutProduits(categorie1.getId());
126.      Assert.assertNotNull(categorie2);
127.      Assert.assertEquals(categorie1.getNom(), categorie2.getNom());
128.  }
129.
130.  @Test
131.  public void getProduitByNameWithCategorie() {
132.      log("getProduitByNameWithCategorie", 1);
133.      Produit produit = dao.getProduitByNameWithCategorie("produit03");
134.      Assert.assertNotNull(produit);
135.      Assert.assertNotNull(produit.getCategorie());
136.  }
137.
138.  @Test
139.  public void getProduitByNameWithoutCategorie() {
140.      log("getProduitByNameWithoutCategorie", 1);
141.      Produit produit = dao.getProduitByNameWithoutCategorie("produit03");
142.      Assert.assertNotNull(produit);
143.      Assert.assertEquals("produit03", produit.getNom());
144.  }
145.
146.  @Test
147.  public void getProduitByIdWithCategorie() {
148.      log("getProduitByNameWithCategorie", 1);
149.      Produit produit = dao.getProduitByNameWithCategorie("produit03");
150.      Produit produit2 = dao.getProduitByIdWithCategorie(produit.getId());
151.      Assert.assertNotNull(produit2);
152.      Assert.assertEquals(produit2.getNom(), produit.getNom());
153.      Assert.assertEquals(produit2.getId(), produit.getId());
154.      Assert.assertEquals(produit.getCategorie().getId(), produit2.getCategorie().getId());
155.  }
156.
157.  @Test
158.  public void getProduitByIdWithoutCategorie() {
159.      log("getProduitByIdWithoutCategorie", 1);
160.      Produit produit = dao.getProduitByNameWithCategorie("produit03");
161.      Produit produit2 = dao.getProduitByIdWithoutCategorie(produit.getId());
162.      Assert.assertNotNull(produit2);
163.      Assert.assertEquals(produit2.getNom(), produit.getNom());
164.      Assert.assertEquals(produit2.getId(), produit.getId());
165.  }
166.
167.  @Test
168.  public void doInsertsInTransaction() {
169.      log("Ajout d'une catégorie [cat1] avec deux produits de même nom", 1);
170.      // on fait l'insertion
171.      Categorie categorie = new Categorie("cat1");
172.      categorie.addProduit(new Produit("x", 1.0, ""));
173.      categorie.addProduit(new Produit("x", 1.0, ""));
174.      // ajout de la catégorie - par cascade les produits vont eux aussi être insérés
175.      try {
176.          categorie = dao.addCategories(Lists.newArrayList(categorie)).get(0);
177.      } catch (DaoException e) {
178.          show("Les erreurs suivantes se sont produites :", e.getErreurs());
179.      }
180.      // vérifications
181.      List<Categorie> categories = dao.getAllCategories();
182.      Assert.assertEquals(2, categories.size());
183.      List<Produit> produits = dao.getAllProduits();
184.      Assert.assertEquals(10, produits.size());
185.  }
186.
187.  @Test
188.  public void updateDataBase() {
189.      log("Mise à jour du prix des produits de [categorie1]", 1);
190.      Categorie categorie1 = dao.getCategorieByNameWithProduits("categorie1");
191.      Categorie categorie1Saved = dao.getCategorieByNameWithProduits("categorie1");
192.      Set<Produit> produits = categorie1.getProduits();
193.      for (Produit produit : produits) {
194.          produit.setPrix(1.1 * produit.getPrix());
195.      }
196.      List<Produit> produits2 = Lists.newArrayList(produits);
197.      produits2 = dao.updateProduits(produits2);
198.      // vérifications
199.      List<Produit> produitsSaved = Lists.newArrayList(categorie1Saved.getProduits());
200.      for (Produit produit2 : produits2) {
201.          Produit produit = findProduitByName(produit2.getNom(), produitsSaved);
202.          Assert.assertEquals(produit2.getPrix(), produit.getPrix() * 1.1, 1e-6);
203.      }
```

```
204.  }
205.
206.  @Test
207.  public void addProduits() throws BeansException, JsonProcessingException {
208.      log("Ajout de deux produits de catégorie [categorie0]", 1);
209.      Categorie categorie0 = dao.getCategorieByNameWithoutProduits("categorie0");
210.      Long idCategorie = categorie0.getId();
211.      Produit p1 = new Produit("x", 1, "");
212.      p1.setIdCategorie(idCategorie);
213.      p1.setCategorie(categorie0);
214.      Produit p2 = new Produit("y", 1, "");
215.      p2.setIdCategorie(idCategorie);
216.      p2.setCategorie(categorie0);
217.      List<Produit> produits = new ArrayList<Produit>();
218.      produits.add(p1);
219.      produits.add(p2);
220.      produits = dao.addProduits(produits);
221.      // vérification
222.      affiche(produits, jsonMapperProduitWithoutCategorie);
223.  }
224.
225.  // -------------- méthodes privées
226.  private Produit findProduitByName(String nom, List<Produit> produits) {
227.      for (Produit produit : produits) {
228.          if (produit.getNom().equals(nom)) {
229.              return produit;
230.          }
231.      }
232.      return null;
233.  }
234.
235.  private Categorie findCategorieByName(String nom, List<Categorie> categories) {
236.      for (Categorie categorie : categories) {
237.          if (categorie.getNom().equals(nom)) {
238.              return categorie;
239.          }
240.      }
241.      return null;
242.  }
243.
244.  // affichage d'un élément de type T
245.  static private <T> void affiche(T element, ObjectMapper jsonMapper) throws JsonProcessingException {
246.      System.out.println(jsonMapper.writeValueAsString(element));
247.  }
248.
249.  // affichage d'une liste d'éléments de type T
250.  static private <T> void affiche(List<T> elements, ObjectMapper jsonMapper) throws JsonProcessingException {
251.      for (T element : elements) {
252.          affiche(element, jsonMapper);
253.      }
254.  }
255.
256.  private static void log(String message, int mode) {
257.      // affiche message
258.      String toPrint = null;
259.      switch (mode) {
260.      case 1:
261.          toPrint = String.format("%s ----------------------------------", message);
262.          break;
263.      case 2:
264.          toPrint = String.format("-- %s", message);
265.          break;
266.      }
267.      System.out.println(toPrint);
268.  }
269.
270.  private static void show(String title, List<String> messages) {
271.      // titre
272.      System.out.println(String.format("%s : ", title));
273.      // messages
274.      for (String message : messages) {
275.          System.out.println(String.format("- %s", message));
276.      }
277.  }
278.
279. }
```

Son exécution réussit et donne les résultats suivants sur la console :

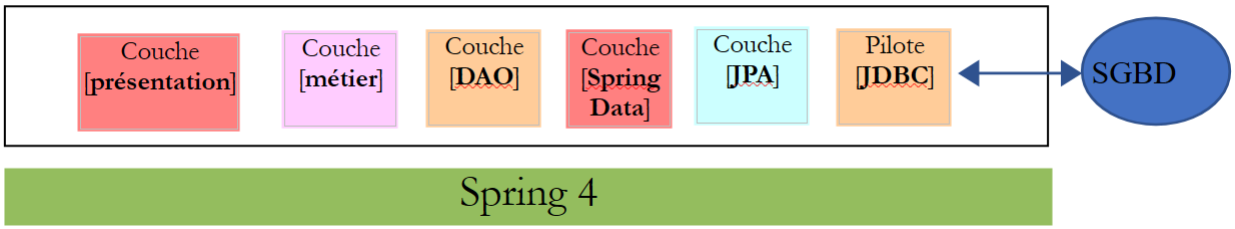
```
1. Vidage de la base de données ----------------------------------
2. Remplissage de la base ----------------------------------
3. Ajout de deux produits de catégorie [categorie0] ----------------------------------
4. {"id":6285,"version":0,"nom":"x","idCategorie":1319,"prix":1.0,"description":""}
5. {"id":6286,"version":0,"nom":"y","idCategorie":1319,"prix":1.0,"description":""}
6. Vidage de la base de données ----------------------------------
```

```
7. Remplissage de la base --------------------------------
8. Mise à jour du prix des produits de [categorie1] --------------------------------
9. Vidage de la base de données --------------------------------
10.  Remplissage de la base --------------------------------
11.  getCategorieByIdWithoutProduits --------------------------------
12.  Vidage de la base de données --------------------------------
13.  Remplissage de la base --------------------------------
14.  getProduitByNameWithoutCategorie --------------------------------
15.  Vidage de la base de données --------------------------------
16.  Remplissage de la base --------------------------------
17.  getCategorieByNameWithProduits --------------------------------
18.  Vidage de la base de données --------------------------------
19.  Remplissage de la base --------------------------------
20.  getCategorieByNameWithoutProduits --------------------------------
21.  Vidage de la base de données --------------------------------
22.  Remplissage de la base --------------------------------
23.  getProduitByNameWithCategorie --------------------------------
24.  Vidage de la base de données --------------------------------
25.  Remplissage de la base --------------------------------
26.  getProduitByNameWithCategorie --------------------------------
27.  Vidage de la base de données --------------------------------
28.  Remplissage de la base --------------------------------
29.  getProduitByIdWithoutCategorie --------------------------------
30.  Vidage de la base de données --------------------------------
31.  Remplissage de la base --------------------------------
32.  -- Liste des catégories
33.  {"id":1337,"version":0,"nom":"categorie0"}
34.  {"id":1338,"version":0,"nom":"categorie1"}
35.  -- Liste des produits
36.  {"id":6367,"version":0,"nom":"produit00","idCategorie":1337,"prix":100.0,"description":"desc00"}
37.  {"id":6368,"version":0,"nom":"produit01","idCategorie":1337,"prix":101.0,"description":"desc01"}
38.  {"id":6369,"version":0,"nom":"produit02","idCategorie":1337,"prix":102.0,"description":"desc02"}
39.  {"id":6370,"version":0,"nom":"produit03","idCategorie":1337,"prix":103.0,"description":"desc03"}
40.  {"id":6371,"version":0,"nom":"produit04","idCategorie":1337,"prix":104.0,"description":"desc04"}
41.  {"id":6372,"version":0,"nom":"produit10","idCategorie":1338,"prix":110.0,"description":"desc10"}
42.  {"id":6373,"version":0,"nom":"produit11","idCategorie":1338,"prix":111.0,"description":"desc11"}
43.  {"id":6374,"version":0,"nom":"produit12","idCategorie":1338,"prix":112.0,"description":"desc12"}
44.  {"id":6375,"version":0,"nom":"produit13","idCategorie":1338,"prix":113.0,"description":"desc13"}
45.  {"id":6376,"version":0,"nom":"produit14","idCategorie":1338,"prix":114.0,"description":"desc14"}
46.  Vidage de la base de données --------------------------------
47.  Remplissage de la base --------------------------------
48.  getCategorieByIdWithProduits --------------------------------
49.  Vidage de la base de données --------------------------------
50.  Remplissage de la base --------------------------------
51.  Ajout d'une catégorie [cat1] avec deux produits de même nom --------------------------------
52.  Les erreurs suivantes se sont produites :
53.  - org.hibernate.exception.ConstraintViolationException: could not execute statement
54.  - could not execute statement
55.  - Duplicate entry 'x' for key 'NOM'
56.  11:24:37.650 [Thread-1] INFO  o.s.c.a.AnnotationConfigApplicationContext - Closing
     org.springframework.context.annotation.AnnotationConfigApplicationContext@f8c1ddd: startup date [Fri Nov 20 11:24:34 CET
     2015]; root of context hierarchy
```

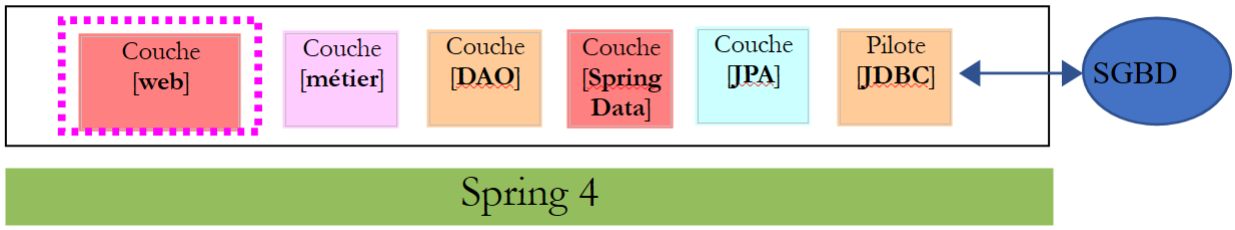
14 [TD] : Exposition sur le web de la couche [metier]

Mots clés : architecture multicouche, Spring, injection de dépendances, service web / jSON, client / serveur.

Revenons à l'architecture actuelle de l'application du TD :

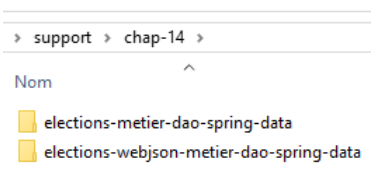


Nous allons faire évoluer cette architecture vers la suivante :



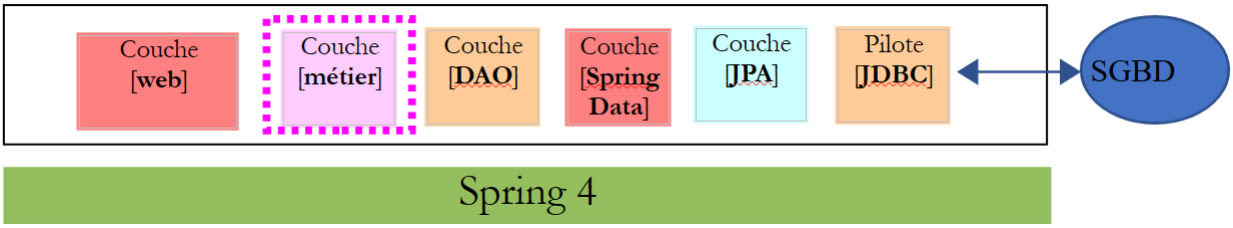
afin d'exposer sur le web l'interface [IMetier] de la couche métier. Pour cela nous allons suivre la méthodologie décrite au paragraphe 13.5, page 226.

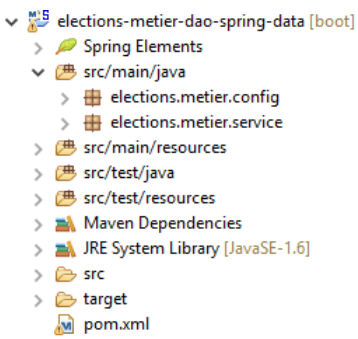
14.1 Support



Les projets de ce chapitre seront trouvés dans le dossier [support / chap-14].

14.2 Le projet Eclipse de la couche [métier]





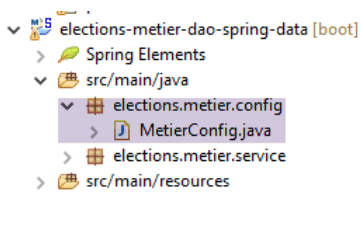
14.2.1 Configuration Maven

Le projet de la couche [métier] est un projet Maven configuré par le fichier [pom.xml] suivant :

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <project xmlns="http://maven.apache.org/POM/4.0.0"
3.   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd"
4.   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
5.   <modelVersion>4.0.0</modelVersion>
6.   <groupId>istia.st.elections</groupId>
7.   <artifactId>elections-metier-dao-spring-data</artifactId>
8.   <version>0.1.0</version>
9.
10.    <!-- dépendances -->
11.    <parent>
12.        <groupId>org.springframework.boot</groupId>
13.        <artifactId>spring-boot-starter-parent</artifactId>
14.        <version>1.2.7.RELEASE</version>
15.    </parent>
16.    <dependencies>
17.        <!-- couche [DAO] -->
18.        <dependency>
19.            <groupId>istia.st.elections</groupId>
20.            <artifactId>elections-dao-spring-data-01</artifactId>
21.            <version>0.0.1-SNAPSHOT</version>
22.        </dependency>
23.        <!-- Spring Boot -->
24.        <dependency>
25.            <groupId>org.springframework.boot</groupId>
26.            <artifactId>spring-boot</artifactId>
27.            <scope>test</scope>
28.        </dependency>
29.        <!-- Spring Boot Test -->
30.        <dependency>
31.            <groupId>org.springframework.boot</groupId>
32.            <artifactId>spring-boot-starter-test</artifactId>
33.            <scope>test</scope>
34.        </dependency>
35.    </dependencies>
36.
37.    <properties>
38.        <!-- use UTF-8 for everything -->
39.        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
40.        <java.version>1.8</java.version>
41.    </properties>
42.
43.    <build>
44.        <plugins>
45.            <plugin>
46.                <groupId>org.apache.maven.plugins</groupId>
47.                <artifactId>maven-surefire-plugin</artifactId>
48.                <version>2.18.1</version>
49.            </plugin>
50.        </plugins>
51.    </build>
52. </project>
```

- lignes 18-22 : la dépendance sur la couche [DAO] construite au paragraphe 12, page 207 ;
- lignes 23-34 : les dépendances nécessaires aux tests ;

14.2.2 Configuration Spring

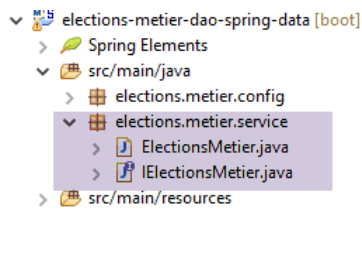


Le projet de la couche [métier] est un projet Spring configuré par le fichier [MetierConfig] suivant :

```
1. package elections.metier.config;
2.
3. import org.springframework.context.annotation.ComponentScan;
4. import org.springframework.context.annotation.Import;
5.
6. import elections.dao.config.DaoConfig;
7.
8. @Import({ DaoConfig.class })
9. @ComponentScan({ "elections.metier.service" })
10.  public class MetierConfig {
11.  }
```

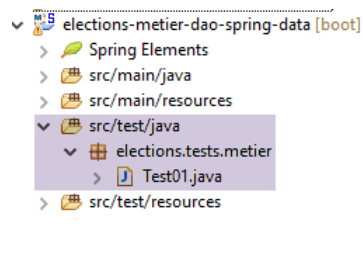
- nous n'utilisons pas ici la notation [@Configuration] qui fait de la classe une classe de configuration Spring. La présence des annotations [@Import, @ComponentScan] fait automatiquement d'elle une classe de configuration ;
- ligne 8 : on importe le fichier de configuration de la couche [DAO]. On dispose alors de tous les beans définis par ce fichier ;
- ligne 9 : d'autres beans Spring sont à chercher dans le dossier [elections.metier.service] ;

14.2.3 Implémentation de la couche [métier]



L'implémentation de la couche [métier] est celle qui a été définie au paragraphe 8.5, page 128.

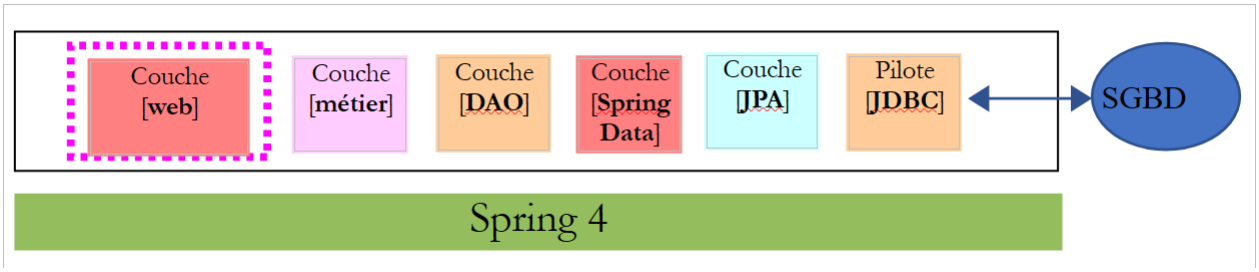
14.2.4 Le test de la couche [métier]



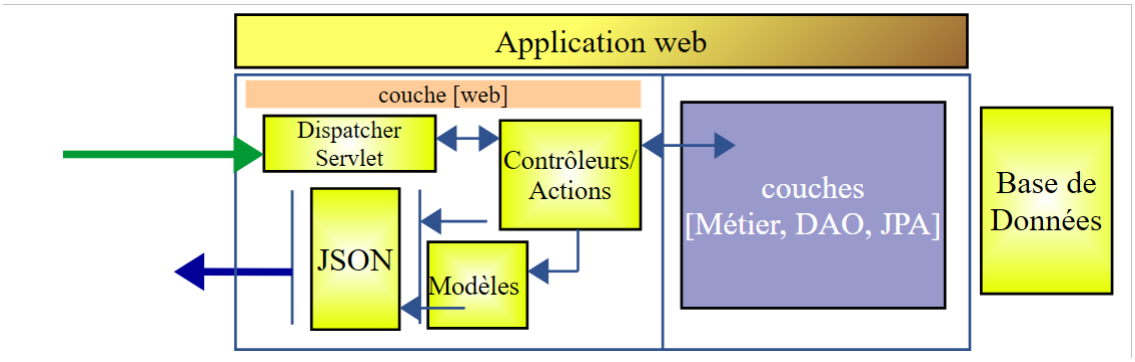
La classe de test est celle décrite au paragraphe 8.6, page 129.

Travail à faire : implémentez le projet de la couche [métier] et passer son test unitaire. Générez l'archive de la couche dans le dépôt Maven local (run as/ Maven / install).

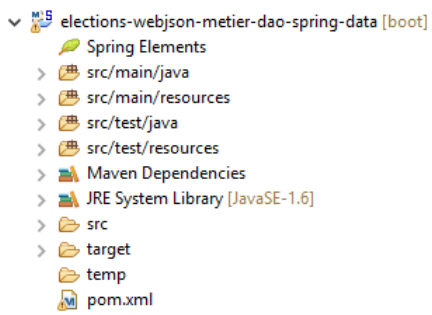
14.3 Le projet Eclipse de la couche [web]

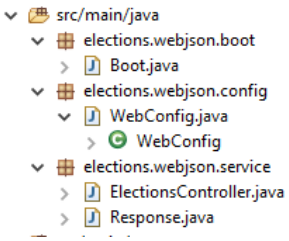


La couche web est une couche Spring MVC :



Le projet Eclipse a la structure suivante :





- [Boot.java] est la classe qui lance le service web ;
- [WebConfig.java] est la classe de configuration du service web ;
- [Response.java] est la réponse faite par les différentes URL du service web ;
- [ElectionsController] est la classe d'implémentation du service web ;

14.4 Configuration Maven

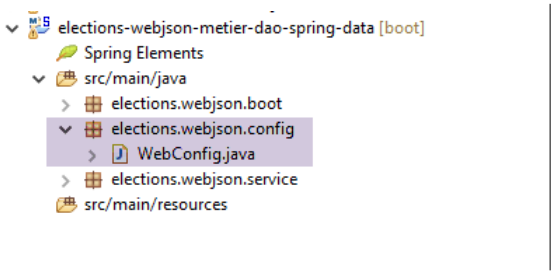
Le projet est un projet Maven configuré par le fichier [pom.xml] suivant :

```
1. <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2.   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
3.   <modelVersion>4.0.0</modelVersion>
```

```
4.  <groupId>istia.st.elections</groupId>
5.  <artifactId>elections-webjson-metier-dao-spring-data</artifactId>
6.  <version>0.0.1-SNAPSHOT</version>
7.
8.  <name>elections-webjson-metier-dao-spring-data</name>
9.  <description>couche métier exposée comme un service web / jSON</description>
10.
11.   <parent>
12.       <groupId>org.springframework.boot</groupId>
13.       <artifactId>spring-boot-starter-parent</artifactId>
14.       <version>1.2.7.RELEASE</version>
15.   </parent>
16.
17.   <dependencies>
18.       <!-- couche métier -->
19.       <dependency>
20.           <groupId>istia.st.elections</groupId>
21.           <artifactId>elections-metier-dao-spring-data</artifactId>
22.           <version>0.1.0</version>
23.       </dependency>
24.       <!-- couche MVC -->
25.       <dependency>
26.           <groupId>org.springframework.boot</groupId>
27.           <artifactId>spring-boot-starter-web</artifactId>
28.       </dependency>
29.   </dependencies>
30.
31.   <properties>
32.       <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
33.       <java.version>1.8</java.version>
34.   </properties>
35.
36.   <build>
37.       <plugins>
38.           <plugin>
39.               <groupId>org.apache.maven.plugins</groupId>
40.               <artifactId>maven-surefire-plugin</artifactId>
41.               <version>2.18.1</version>
42.           </plugin>
43.       </plugins>
44.   </build>
45.
46. </project>
```

- lignes 19-23 : la dépendance sur l'archive la couche [métier]. C'est celle que nous avons créé au paragraphe 14 page 263;
- lignes 25-28 : la dépendance pour avoir une application Spring MVC ;

14.5 Configuration Spring



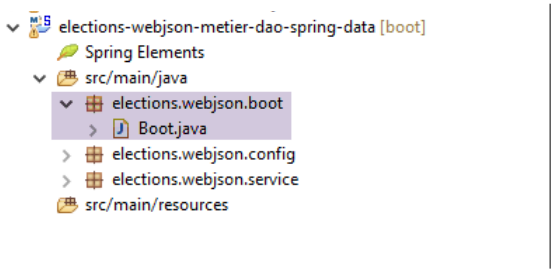
La classe [WebConfig] configure le service web :

```
1. package elections.webjson.config;
2.
3. import org.springframework.beans.factory.annotation.Autowired;
4. import org.springframework.beans.factory.config.ConfigurableBeanFactory;
5. import org.springframework.boot.context.embedded.EmbeddedServletContainerFactory;
6. import org.springframework.boot.context.embedded.ServletRegistrationBean;
7. import org.springframework.boot.context.embedded.tomcat.TomcatEmbeddedServletContainerFactory;
8. import org.springframework.context.ApplicationContext;
9. import org.springframework.context.annotation.Bean;
10.  import org.springframework.context.annotation.ComponentScan;
11.  import org.springframework.context.annotation.Import;
12.  import org.springframework.context.annotation.Scope;
13.  import org.springframework.web.context.WebApplicationContext;
14.  import org.springframework.web.servlet.DispatcherServlet;
15.  import org.springframework.web.servlet.config.annotation.EnableWebMvc;
```

```
16.
17.  import com.fasterxml.jackson.databind.ObjectMapper;
18.
19.  import elections.metier.config.MetierConfig;
20.
21.  @EnableWebMvc
22.  @Import({ MetierConfig.class })
23.  @ComponentScan({ "elections.webjson.service" })
24.  public class WebConfig {
25.    // ------------------------------- configuration couche [web]
26.    @Autowired
27.    private ApplicationContext context;
28.
29.    @Bean
30.    public DispatcherServlet dispatcherServlet() {
31.        DispatcherServlet servlet = new DispatcherServlet((WebApplicationContext) context);
32.        return servlet;
33.    }
34.
35.    @Bean
36.    public ServletRegistrationBean servletRegistrationBean(DispatcherServlet dispatcherServlet) {
37.        return new ServletRegistrationBean(dispatcherServlet, "/*");
38.    }
39.
40.    @Bean
41.    public EmbeddedServletContainerFactory embeddedServletContainerFactory() {
42.        return new TomcatEmbeddedServletContainerFactory("", 8080);
43.    }
44.    // mappeur jSON
45.    @Bean
46.    @Scope(value = ConfigurableBeanFactory.SCOPE_PROTOTYPE)
47.    public ObjectMapper jsonMapper() {
48.        return new ObjectMapper();
49.    }
50.
51.  }
```

- la signification de cette configuration a été donnée au paragraphe 13.5.3.1, page 228. Nous n'expliquons que les nouveautés :
- ligne 22 : on importe le fichier de configuration de la couche [métier] pour bénéficier de tous ses beans ;
- ligne 23 : on indique que d'autres beans seront trouvés dans le dossier [elections.webjson.server.service] ;

14.6 La classe de lancement du service web

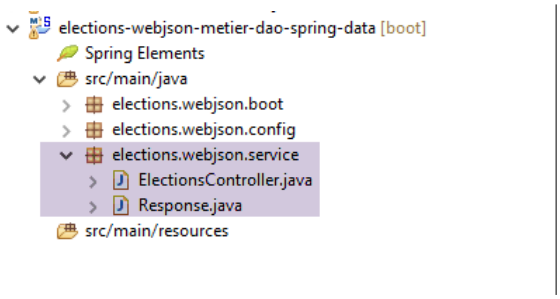


La classe [Boot] lance le service web de la façon suivante :

```
1. package elections.webjson.boot;
2.
3. import org.springframework.boot.SpringApplication;
4.
5. import elections.webjson.config.WebConfig;
6.
7. public class Boot {
8.
9.   public static void main(String[] args) {
10.       SpringApplication.run(WebConfig.class, args);
11.   }
12. }
```

- ligne 10 : la méthode statique [SpringApplication.*run*] va exploiter le fichier de configuration [WebConfig]. A cause de l'annotation [@EnableAutoConfiguration], Spring Boot va lancer le serveur Tomvat et déployer le service web dessus ;

14.7 La réponse des URL du service web

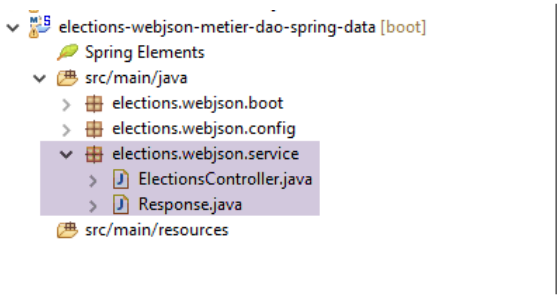


Toutes les URL du service web / jSON envoient le même type de réponse :

```
1. package elections.webjson.service;
2.
3. import java.util.List;
4.
5. public class Response<T> {
6.
7.   // ----------------- propriétés
8.   // statut de l'opération
9.   private int status;
10.    // les éventuels messages d'erreur
11.    private List<String> messages;
12.    // le corps de la réponse
13.    private T body;
14.
15.    // constructeurs
16.    public Response() {
17.
18.    }
19.
20.    public Response(int status, List<String> messages, T body) {
21.        this.status = status;
22.        this.messages = messages;
23.        this.body = body;
24.    }
25.
26.    // getters et setters
27.  ...
28.  }
```

Cette classe a été présentée et étudiée au paragraphe 13.5.5.3, page 235.

14.8 L'implémentation du service web / jSON



Le service web / jSON est implémenté par la classe [ElectionsController] suivante :

```
1. package elections.webjson.service;
2.
3. import java.util.ArrayList;
4. import java.util.List;
5.
6. import javax.servlet.http.HttpServletRequest;
7.
```

```java
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.ResponseBody;

import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;

import elections.dao.entities.ElectionsConfig;
import elections.dao.entities.ElectionsException;
import elections.metier.service.IElectionsMetier;

@Controller
public class ElectionsController {

  // dépendances Spring
  @Autowired
  private ObjectMapper jsonMapper;

  @Autowired
  private IElectionsMetier metier;

  @RequestMapping(value = "/getElectionsConfig", method = RequestMethod.GET, produces = "application/json; charset=UTF-8")
  @ResponseBody
  public String getElectionsConfig() throws JsonProcessingException {
      // réponse
      Response<ElectionsConfig> response;
      try {
          response = new Response<>(0, null,
                  new ElectionsConfig(metier.getNbSiegesAPourvoir(), metier.getSeuilElectoral()));
      } catch (ElectionsException e1) {
          response = new Response<>(e1.getCode(), e1.getErreurs(), null);
      } catch (RuntimeException e2) {
          response = new Response<>(1000, getErreursForException(e2), null);
      }
      // réponse
      return jsonMapper.writeValueAsString(response);
  }

  @RequestMapping(value = "/getListesElectorales", method = RequestMethod.GET, produces = "application/json; charset=UTF-8")
  @ResponseBody
  public String getListesElectorales() throws JsonProcessingException {
      throw new UnsupportedOperationException("Not supported yet");
  }

  @RequestMapping(value = "/setListesElectorales", method = RequestMethod.POST, consumes = "application/json; charset=UTF-8", produces = "application/json; charset=UTF-8")
  @ResponseBody
  public String setListesElectorales(HttpServletRequest request) throws JsonProcessingException {
      throw new UnsupportedOperationException("Not supported yet");
  }

  @RequestMapping(value = "/calculerSieges", method = RequestMethod.POST, consumes = "application/json; charset=UTF-8", produces = "application/json; charset=UTF-8")
  @ResponseBody
  public String calculerSieges(HttpServletRequest request) throws JsonProcessingException {
      throw new UnsupportedOperationException("Not supported yet");
  }

  // méthodes privées -----------------------------
  // liste des messages d'erreur d'une RuntimeException
  private List<String> getErreursForException(Exception e) {
      // on récupère la liste des messages d'erreur de l'exception
      Throwable cause = e;
      List<String> erreurs = new ArrayList<>();
      while (cause != null) {
          // on récupère le message seulement s'il est !=null et non blanc
          String message = cause.getMessage();
          if (message != null) {
              message = message.trim();
              if (message.length() != 0) {
                  erreurs.add(message);
              }
          }
          // cause suivante
          cause = cause.getCause();
      }
      return erreurs;
  }

}
```

Travail à faire : en suivant ce qui a été fait au paragraphe 13.5.5, page 233, complétez le code de la classe [ElectionsController].

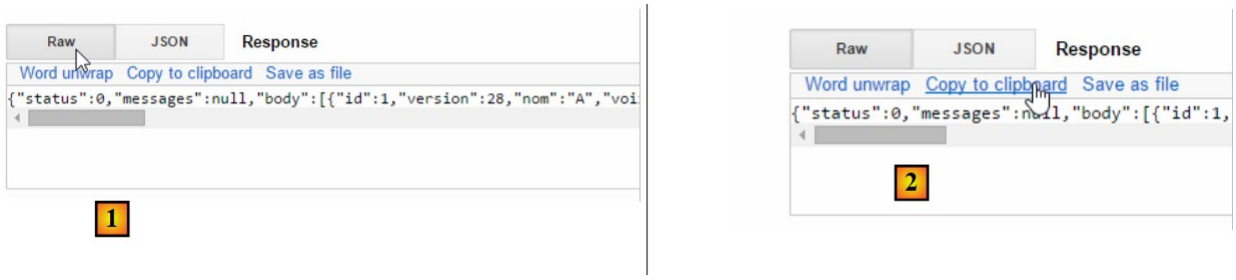
Notes :

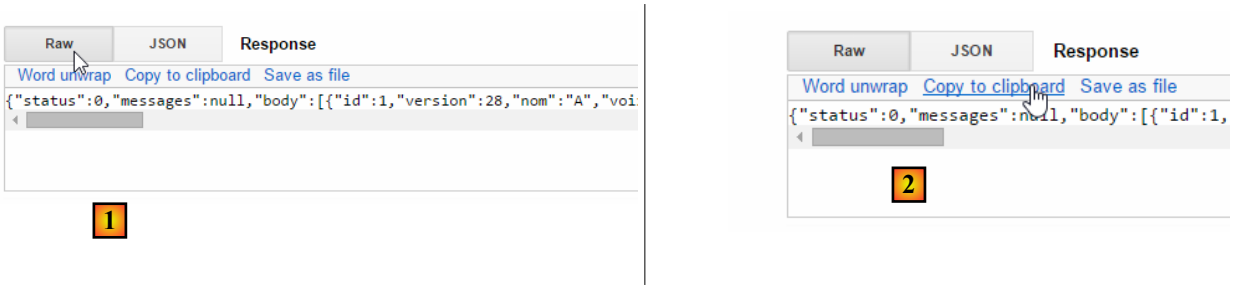
- il n'y a pas ici de filtres jSON car les tables [CONF] et [LISTES] ne sont pas liées entre-elles par une relation de clé étrangère, ce qui allège considérablement le code du service web ;
- ne pas oublier les différentes annotations Spring nécessaires ;
- on donnera aux URL le nom des méthodes associées ;
- la méthode [setListeElectorales] est appelée avec une opération [POST]. La valeur postée est le tableau des listes en compétition (de type *ListeElectorale[]*) avec leurs attributs [sieges, voix, elimine] qu'il faut enregistrer en base. Cette méthode rend un type [Response<Void>] avec un champ [status=0] s'il n'y a pas eu d'erreur, autre chose sinon ;
- la méthode [calculerSieges] est appelée avec une opération [POST]. La valeur postée est le tableau des listes en compétition (de type *ListeElectorale[]*) avec leurs attributs [nom, voix]. Cette méthode rend un type [Response<*ListeElectorale[]*>] avec comme corps, les listes électorales avec leurs champs [sieges, elimine] initialisés ;

14.9 Tests

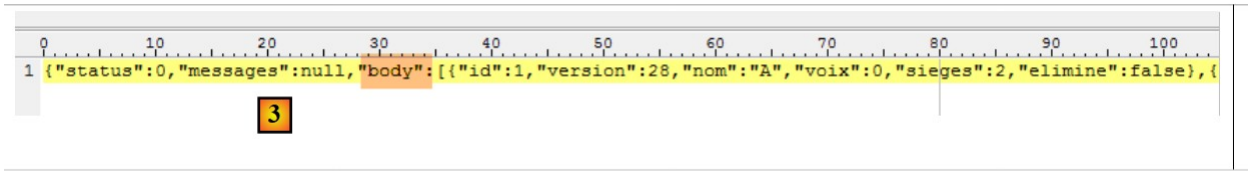
Aorès avoir lancé le service web, vous ferez les tests suivants pour vous assurer du bon fonctionnement du service web avec l'utilitaire [Advanced Rest Client] :

La réponse jSON à la demande précédente est la suivante [1] :

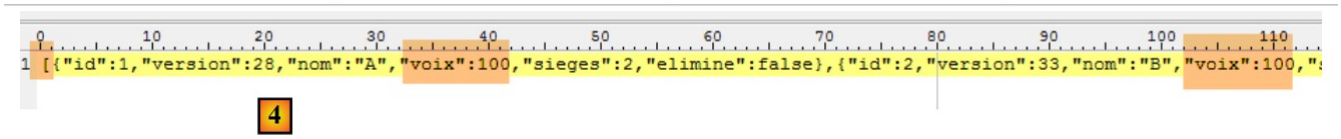




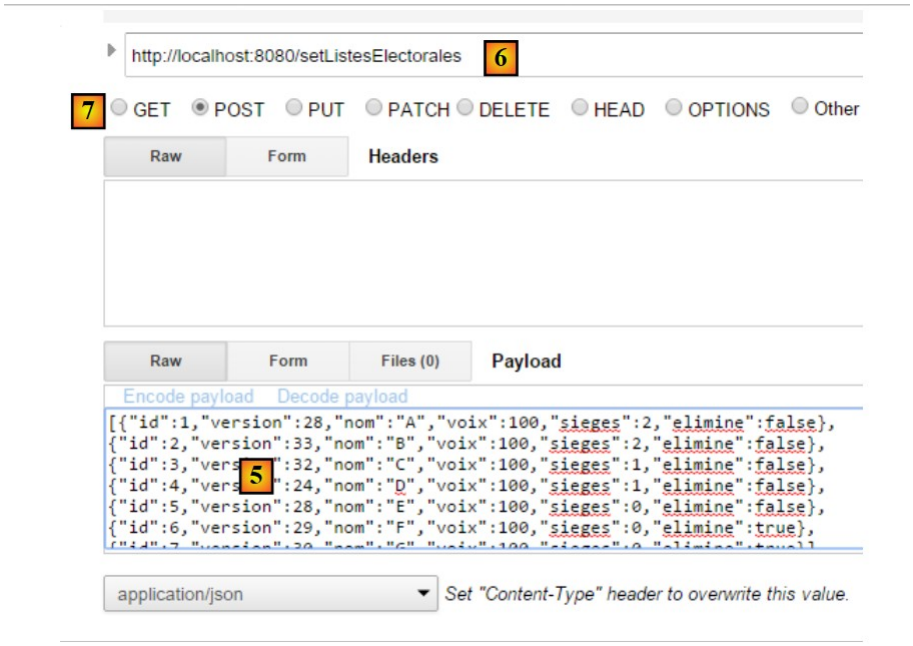
En [2], copier la réponse dans le presse-papiers puis copiez celui-ci dans un éditeur de texte quelconque [3] :



Isolez la valeur du champ [body] et changez par exemple les voix des listes. Ci-dessous [4], on passe à 100 les voix de toutes les listes :



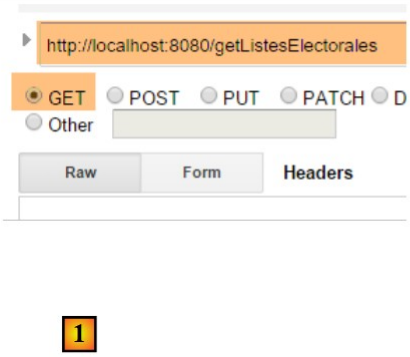
Vérifiez que votre chaîne jSON commence par [et se termine par]. Ces caractères servent à délimiter un tableau jSON. En [5], collez la chaîne jSON ci-dessus. Ce sera la valeur postée pour la prochaine URL. Pour cela, il faut sélectionner la méthode HTTP [POST] [7].



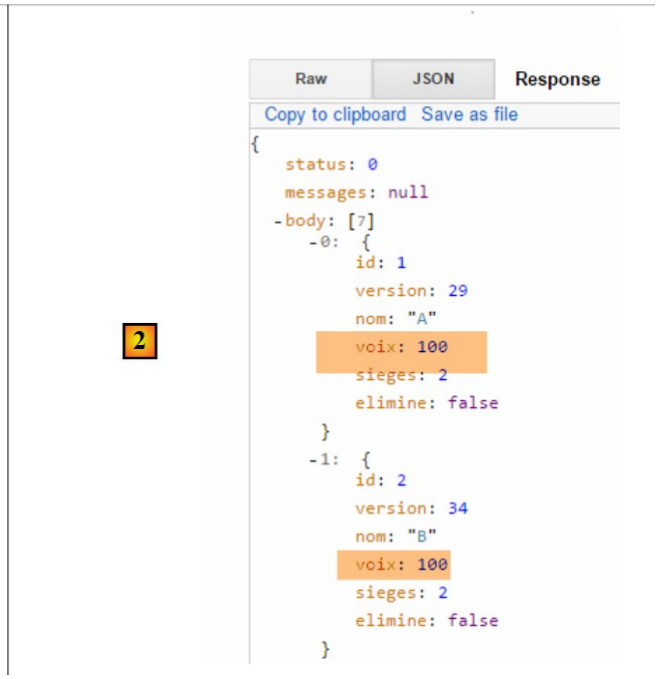
- en [6], demandez l'URL [setListesElectorales]. Cette URL se demande avec un POST. La valeur postée est le tableau jSON des listes en compétition dont il faut enregistrer les résultats en base ;

On obtient le résultat suivant :

Le champ [status=0] indique qu'il n'y a pas eu d'erreur. Pour le vérifier, redemandez les listes en compétition et vérifiez que les modifications que vous aviez faites sur les listes ont été prises en compte :

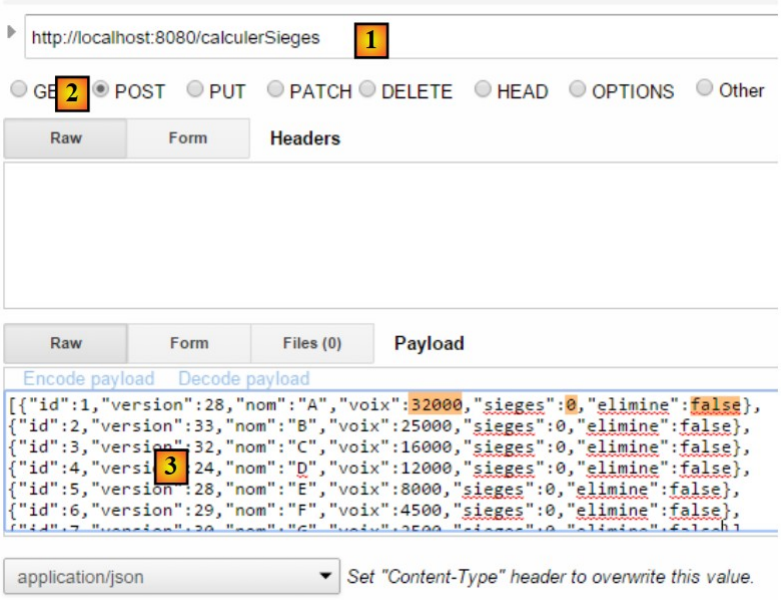


1



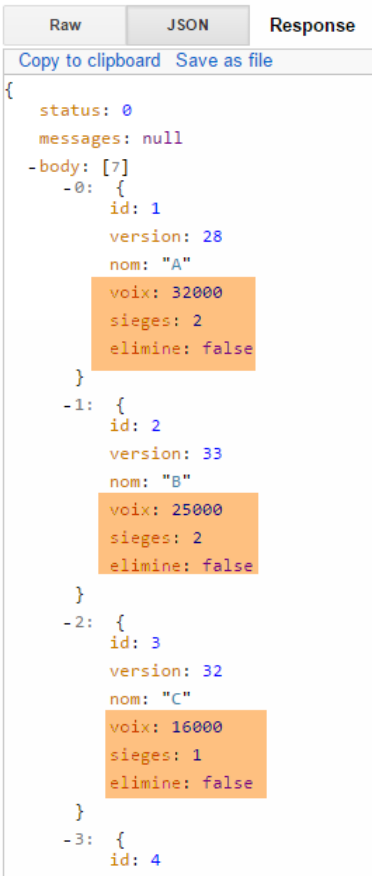
2

On refait un [POST] pour calculer les sièges obtenus par les listes :



- en [1] : l'URL du calcul des sièges ;
- en [2] : on fait un [POST] ;
- en [3] : les listes en compétition. On donne au champ [voix] les valeurs du TD, tous les [sieges] sont à 0, tous les champs [elimine] sont à false ;

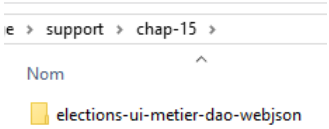
Le résultat obtenu est le suivant :



15 [TD] : création d'un client pour le service web

Mots clés : architecture multicouche, Spring, injection de dépendances, service web / jSON, client / serveur

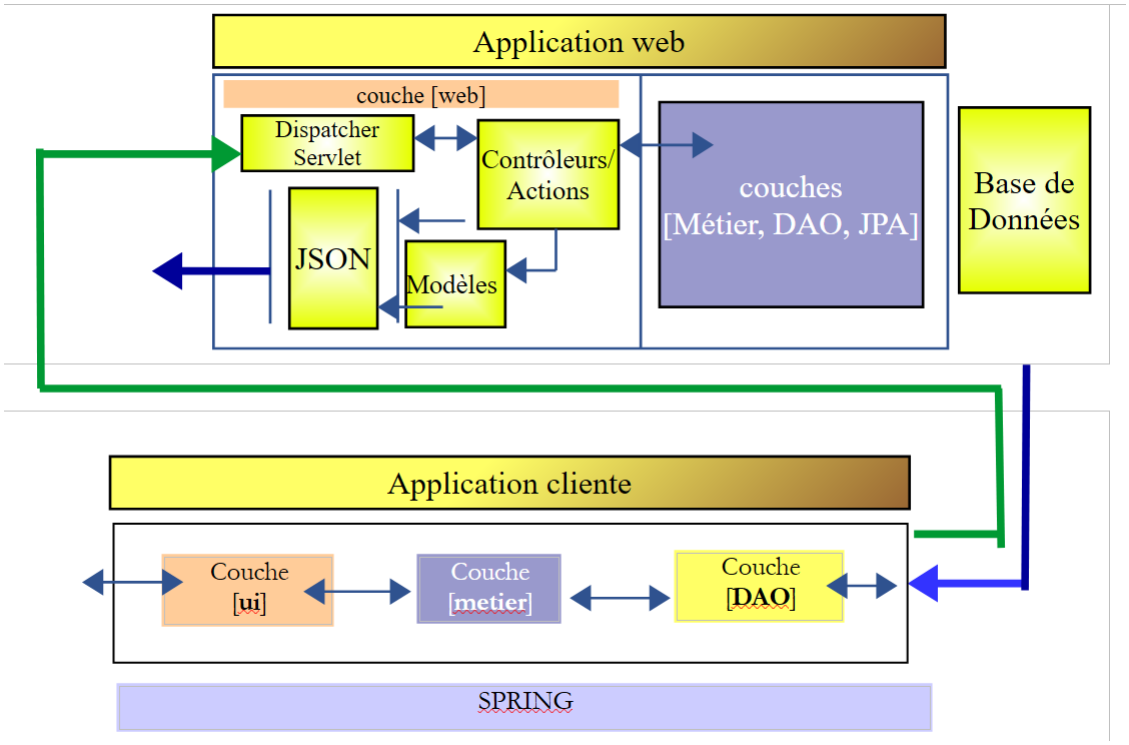
15.1 Support



Les projets de ce chapitre seront trouvés dans le dossier [support / chap-15].

15.2 L'architecture client / serveur

Nous voulons créer l'architecture client / serveur suivante :



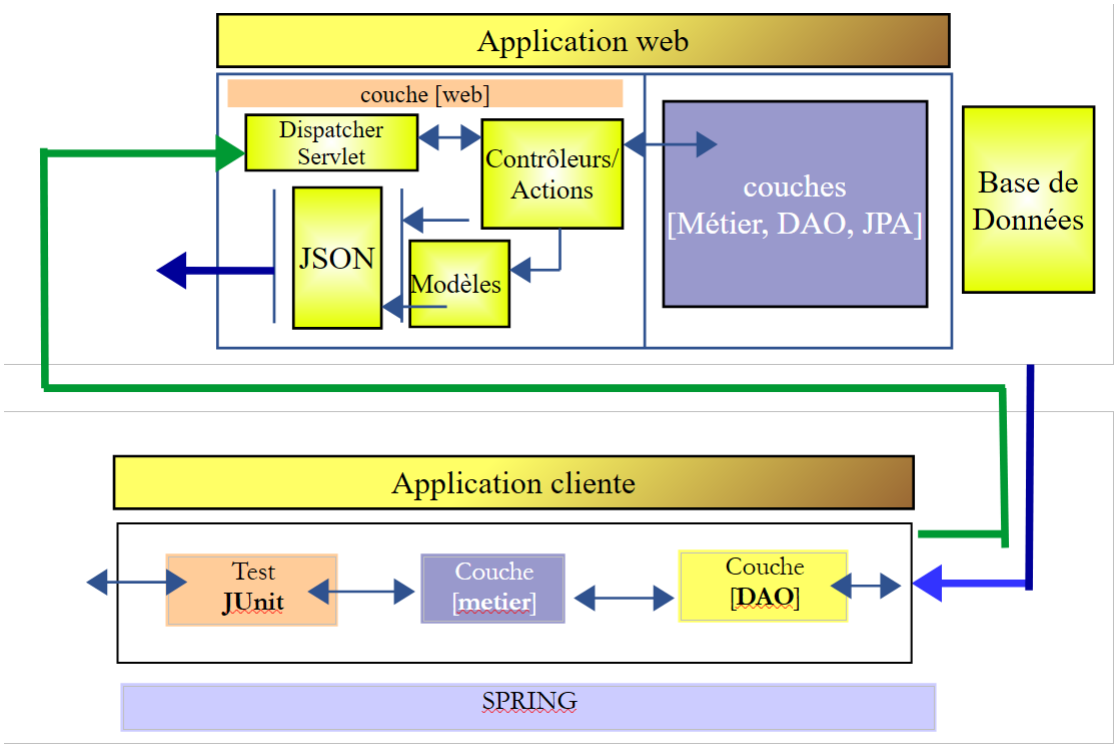
La couche [ui] sera celle déjà développée aux paragraphes 9 et 10. Cela sera possible parce que la couche [métier] ci-dessus implémentera la même interface [IElectionsMetier] que la couche [métier] du paragraphe 8 :

```
1. package elections.client.metier;
2.
3. import elections.client.entities.ListeElectorale;
```

```
4.
5. public interface IElectionsMetier {
6.
7.   // obtenir les listes en compétition
8.   public ListeElectorale[] getListesElectorales();
9.
10.   // le nombre de sièges à pourvoir
11.   public int getNbSiegesAPourvoir();
12.
13.   // le seuil électoral
14.   public double getSeuilElectoral();
15.
16.   // l'enregistrement des résultats
17.   public void recordResultats(ListeElectorale[] listesElectorales);
18.
19.   // le calcul des sièges
20.   public ListeElectorale[] calculerSieges(ListeElectorale[] listesElectorales);
21.
22. }
```

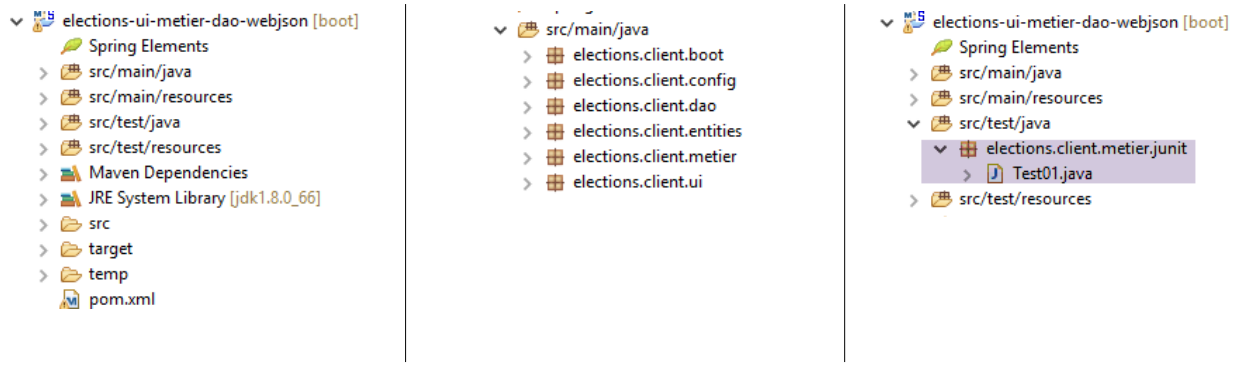
Dans le paragraphe 7, la couche [DAO] échangeait des données avec un SGBD. Ici la couche [DAO] échange des données avec un serveur web / jSON.

Dans un premier temps, nous nous intéresserons à l'architecture suivante :



15.3 Le projet Eclipse

Le projet Eclipse est le suivant :

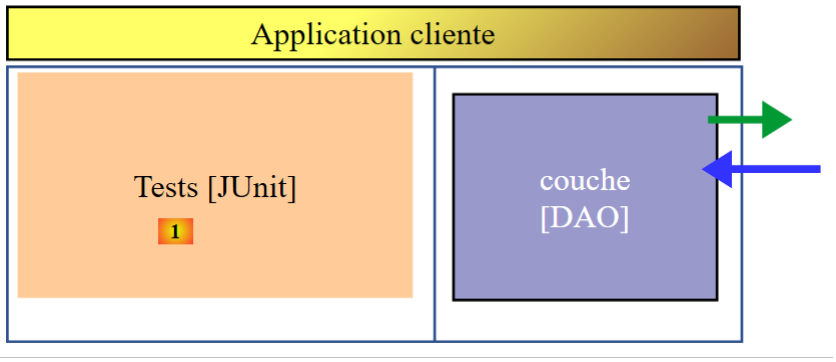


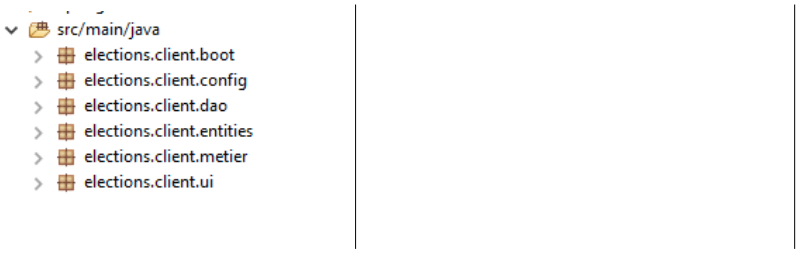
Cette structure reprend celle du projet exemple du paragraphe 13.6.1, page 245. Nous allons suivre la même démarche.

15.4 Configuration Maven

C'est celle décrite au paragraphe 13.6.2, page 246.

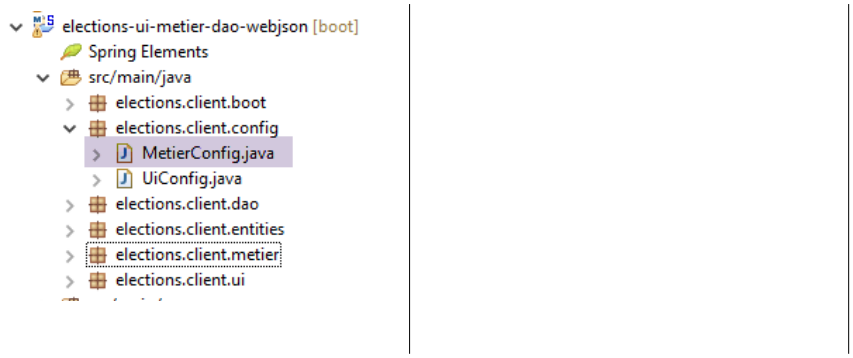
15.5 Implémentation de la couche [DAO]





- le package [elections.client.config] contient la configation Spring de la couche [DAO] ;
- le package [elections .client.dao] contient l'implémentation de la couche [DAO] ;
- le package [elections .client.entities] contient les objets échangés avec le service web / jSON ;
- le package [elections .client.metier] contient la couche [métier]
- le package [elections .client.ui] contient la couche [UI]

15.5.1 Configuration de la couche [métier]

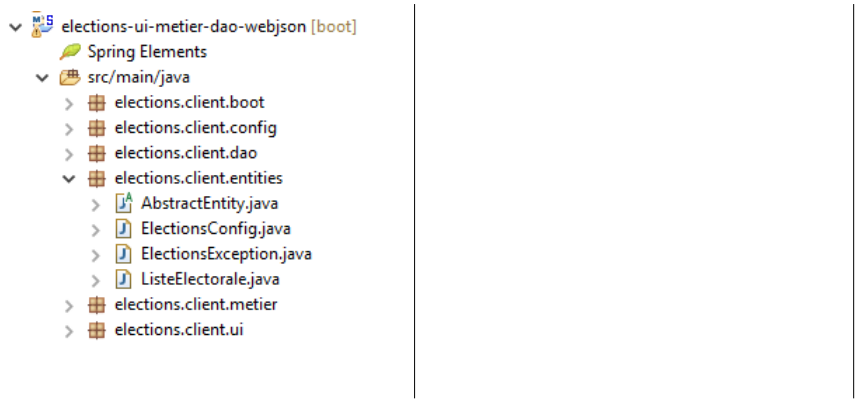


La classe [MetierConfig] fait la configuration Spring de la couche [métier]. Son code est le suivant :

```
1. package elections.client.config;
2.
3. import org.springframework.beans.factory.config.ConfigurableBeanFactory;
4. import org.springframework.context.annotation.Bean;
5. import org.springframework.context.annotation.ComponentScan;
6. import org.springframework.context.annotation.Scope;
7. import org.springframework.http.client.HttpComponentsClientHttpRequestFactory;
8. import org.springframework.web.client.RestTemplate;
9.
10.  import com.fasterxml.jackson.databind.ObjectMapper;
11.
12.  @ComponentScan({ "elections.client.dao","elections.client.metier" })
13.  public class MetierConfig {
14.
15.    // constantes
16.    static private final int TIMEOUT = 1000;
17.    static private final String URL_WEBJSON = "http://localhost:8080";
18.
19.    @Bean
20.    public RestTemplate restTemplate(int timeout) {
21.        // création du composant RestTemplate
22.        HttpComponentsClientHttpRequestFactory factory = new HttpComponentsClientHttpRequestFactory();
23.        RestTemplate restTemplate = new RestTemplate(factory);
24.        // timeout des échanges
25.        factory.setConnectTimeout(timeout);
26.        factory.setReadTimeout(timeout);
27.        // résultat
28.        return restTemplate;
29.    }
30.
31.    @Bean
32.    public int timeout() {
33.        return TIMEOUT;
34.    }
35.
36.    @Bean
37.    public String urlWebJson() {
38.        return URL_WEBJSON;
39.    }
40.
41.    // mappeur jSON
42.    @Bean
43.    @Scope(value = ConfigurableBeanFactory.SCOPE_PROTOTYPE)
44.    public ObjectMapper jsonMapper() {
45.        return new ObjectMapper();
46.    }
47.  }
```

Ce code a été expliqué au paragraphe 13.6.3.1, page 248. Il est plus simple car il n'y a pas ici de filtres jSON à gérer.

15.5.2 Les entités



Les entités manipulées par les couches [DAO] et [métier] sont celles qu'elles échangent avec le service web / jSON. Ce sont les objets de type [ElectionsConfig] et [ListeElectorale]. Côté serveur, ces entités avaient des annotations de persistence JPA. Ici, ces annotations ont été enlevées. Nous redonnons le code des entités pour rappel :

[AbstractEntity]

```
1. package spring.webjson.client.entities;
2.
3.
4. public abstract class AbstractEntity {
5.   // propriétés
6.   protected Long id;
7.   protected Long version;
8.
9.   // constructeurs
10.    public AbstractEntity() {
11.
12.    }
13.
14.    public AbstractEntity(Long id, Long version) {
15.        this.id = id;
16.        this.version = version;
17.    }
18.
19.    // redéfinition [equals] et [hashcode]
20.    @Override
21.    public int hashCode() {
22.        return (id != null ? id.hashCode() : 0);
23.    }
24.
25.    @Override
26.    public boolean equals(Object entity) {
27.        if (!(entity instanceof AbstractEntity)) {
28.            return false;
29.        }
30.        String class1 = this.getClass().getName();
31.        String class2 = entity.getClass().getName();
32.        if (!class2.equals(class1)) {
33.            return false;
34.        }
35.        AbstractEntity other = (AbstractEntity) entity;
36.        return id != null && this.id == other.id.longValue();
37.    }
38.
39.    // getters et setters
40.    ...
41. }
```

[ElectionsConfig]

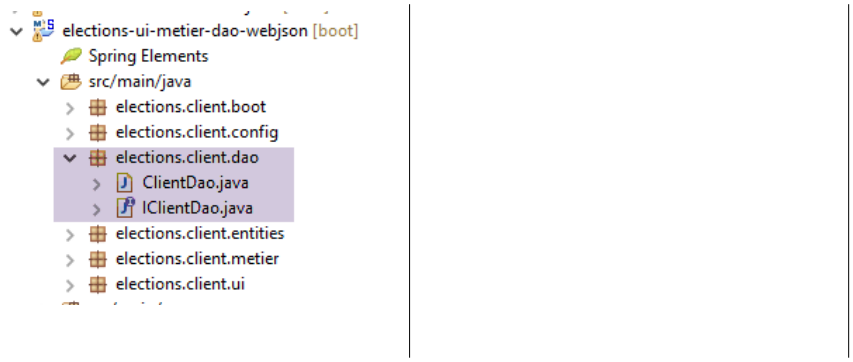
```
1. package elections.webjson.client.entities;
2.
3.
4. public class ElectionsConfig extends AbstractEntity {
5.
6.   // champs
7.   private int nbSiegesAPourvoir;
8.   private double seuilElectoral;
9.
10.    // constructeurs
11.    public ElectionsConfig() {
12.
```

```
13.    }
14.
15.    public ElectionsConfig(int nbSiegesAPourvoir, double seuilElectoral) {
16.        this.nbSiegesAPourvoir = nbSiegesAPourvoir;
17.        this.seuilElectoral = seuilElectoral;
18.    }
19.
20.    // getters et setters
21.    ...
22. }
```

[ListeElectorale]

```
1. package elections.webjson.client.entities;
2.
3.
4. public class ListeElectorale extends AbstractEntity {
5.
6.    // champs
7.    private String nom;
8.    private int voix;
9.    private int sieges;
10.    private boolean elimine;
11.
12.    // constructeurs
13.    public ListeElectorale() {
14.    }
15.
16.    public ListeElectorale(String nom, int voix, int sieges, boolean elimine) {
17.        setNom(nom);
18.        setVoix(voix);
19.        setSieges(sieges);
20.        setElimine(elimine);
21.    }
22.
23.    // getters et setters
24.    ...
25. }
```

15.5.3 L'interface de la couche [DAO]



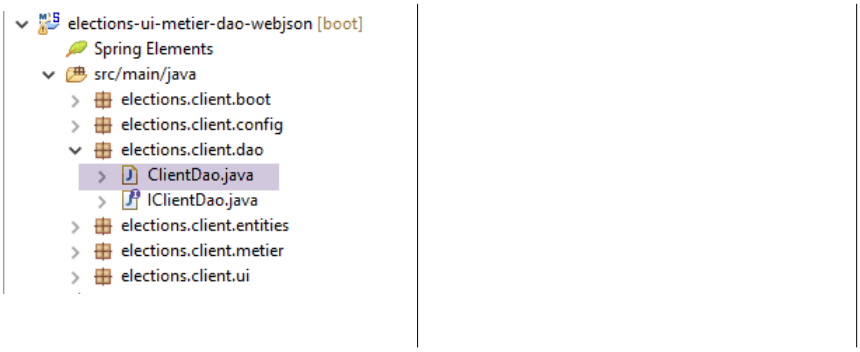
La couche [DAO] présente l'interface [IClientDao] suivante :

```
1. package elections.client.dao;
2.
3. public interface IClientDao {
4.
5.    // requête générique
6.    String getResponse(String url, String jsonPost);
7.
8. }
```

L'interface n'a qu'une unique méthode [getResponse] :

- le 1er paramètre est l'URL du serveur à interroger ;
- le second paramètre est la valeur jSON de la valeur à poster, *null* s'il n'y a rien à poster ;
- le résultat est la chaîne jSON d'un objet [Response<T>] où la classe [Response] a été décrite au paragraphe 14.7, page 269 ;

15.5.4 Implémentation des échanges avec le service web / jSON

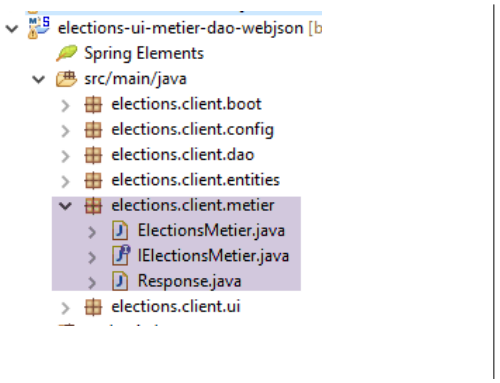


La classe [ClientDao] implémente l'interface [IClientDao] de la façon suivante :

```
1. package elections.client.dao;
2.
3. import java.net.URI;
4. import java.net.URISyntaxException;
5.
6. import org.springframework.beans.factory.annotation.Autowired;
7. import org.springframework.core.ParameterizedTypeReference;
8. import org.springframework.http.MediaType;
9. import org.springframework.http.RequestEntity;
10.  import org.springframework.stereotype.Component;
11.  import org.springframework.web.client.RestTemplate;
12.
13.  import elections.client.entities.ElectionsException;
14.
15.  @Component
16.  public class ClientDao implements IClientDao {
17.
18.    // data
19.    @Autowired
20.    protected RestTemplate restTemplate;
21.    @Autowired
22.    protected String urlServiceWebJson;
23.
24.    // requête générique
25.    @Override
26.    public String getResponse(String url, String jsonPost) {
27.
28.        try {
29.            // url : URL à contacter
30.            // jsonPost : la valeur jSON à poster
31.
32.            // exécution requête
33.            RequestEntity<?> request;
34.            if (jsonPost != null) {
35.                // requête POST
36.                request = RequestEntity.post(new URI(String.format("%s%s", urlServiceWebJson, url)))
37.                        .header("Content-Type", "application/json").accept(MediaType.APPLICATION_JSON).body(jsonPost);
38.            } else {
39.                // requête GET
40.                request = RequestEntity.get(new URI(String.format("%s%s", urlServiceWebJson, url)))
41.                        .accept(MediaType.APPLICATION_JSON).build();
42.            }
43.            // on exécute la requête
44.            return restTemplate.exchange(request, new ParameterizedTypeReference<String>() {
45.            }).getBody();
46.        } catch (URISyntaxException e1) {
47.            throw new ElectionsException(200, e1);
48.        } catch (RuntimeException e2) {
49.            throw new ElectionsException(201, e2);
50.        }
51.    }
52.  }
```

Ce code a été décrit page 254.

15.6 Implémentation de la couche [métier]



Comme il a été dit, la couche [métier] présente la même interface [IElectionsMetier] que dans le paragraphe 8.4, page 128 :

```
1. package elections.client.metier;
2.
3. import elections.client.entities.ListeElectorale;
4.
5. public interface IElectionsMetier {
6.
7.   // obtenir les listes en compétition
8.   public ListeElectorale[] getListesElectorales();
9.
10.    // le nombre de sièges à pourvoir
11.    public int getNbSiegesAPourvoir();
12.
13.    // le seuil électoral
14.    public double getSeuilElectoral();
15.
16.    // l'enregistrement des résultats
17.    public void recordResultats(ListeElectorale[] listesElectorales);
18.
19.    // le calcul des sièges
20.    public ListeElectorale[] calculerSieges(ListeElectorale[] listesElectorales);
21.
22. }
```

Cette interface est implémentée par la classe [ElectionsMetier] suivante :

```
1. package elections.client.metier;
2.
3. import java.util.ArrayList;
4. import java.util.List;
5.
6. import javax.annotation.PostConstruct;
7.
8. import org.springframework.beans.factory.annotation.Autowired;
9. import org.springframework.context.ApplicationContext;
10.  import org.springframework.stereotype.Component;
11.
12.  import com.fasterxml.jackson.core.type.TypeReference;
13.  import com.fasterxml.jackson.databind.ObjectMapper;
14.
15.  import elections.client.dao.IClientDao;
16.  import elections.client.entities.ElectionsConfig;
17.  import elections.client.entities.ElectionsException;
18.  import elections.client.entities.ListeElectorale;
19.
20.  @Component
21.  public class ElectionsMetier implements IElectionsMetier {
22.
23.    @Autowired
24.    private IClientDao dao;
25.    @Autowired
26.    private ApplicationContext context;
27.
28.    // configuration de l'élection
29.    private ElectionsConfig electionsConfig;
30.
31.    @PostConstruct
32.    public void init() {
33.        // mappeurs jSON
```

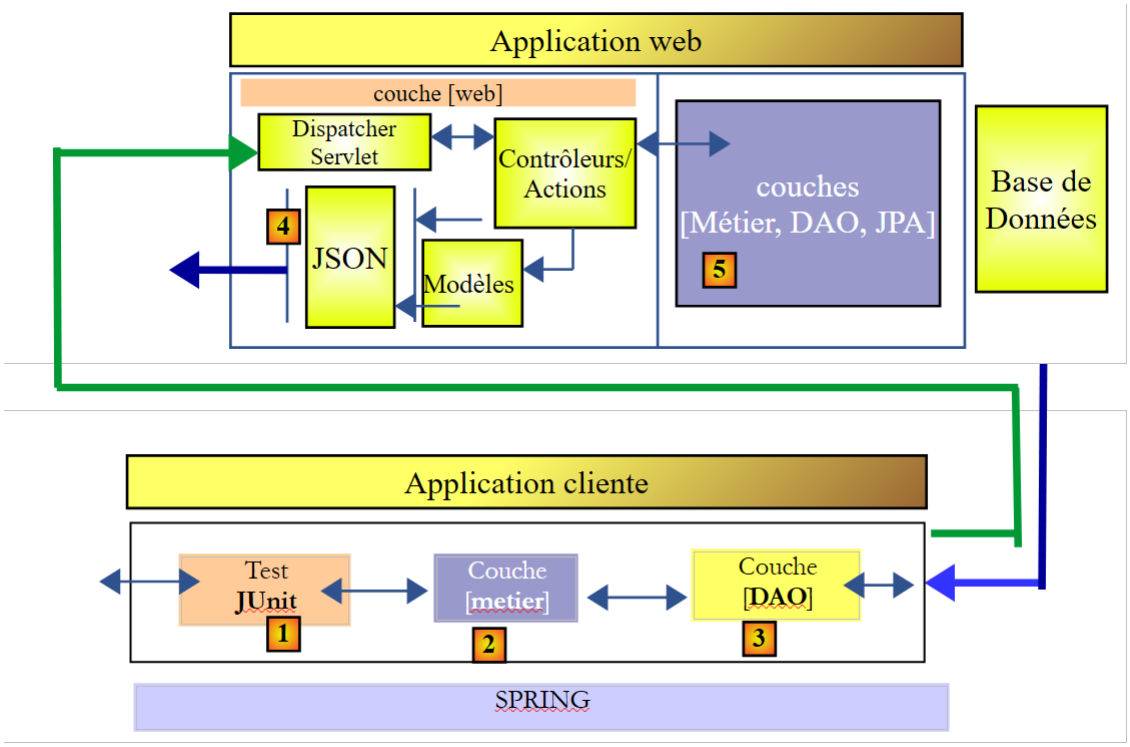
```
34.       ObjectMapper mapperResponse = context.getBean(ObjectMapper.class);
35.       try {
36.           // requête
37.           Response<ElectionsConfig> response = mapperResponse.readValue(dao.getResponse("/getElectionsConfig", null),
38.                   new TypeReference<Response<ElectionsConfig>>() {
39.                   });
40.           // erreur ?
41.           if (response.getStatus() != 0) {
42.               // on lance 1 exception
43.               throw new ElectionsException(response.getStatus(), response.getMessages());
44.           } else {
45.               electionsConfig = response.getBody();
46.           }
47.       } catch (ElectionsException e1) {
48.           throw e1;
49.       } catch (Exception e2) {
50.           throw new ElectionsException(100, getMessagesForException(e2));
51.       }
52.   }
53.
54.   @Override
55.   public ListeElectorale[] getListesElectorales() {
56.       ...
57.   }
58.
59.   @Override
60.   public int getNbSiegesAPourvoir() {
61.       return electionsConfig.getNbSiegesAPourvoir();
62.   }
63.
64.   @Override
65.   public double getSeuilElectoral() {
66.       return electionsConfig.getSeuilElectoral();
67.   }
68.
69.   @Override
70.   public void recordResultats(ListeElectorale[] listesElectorales) {
71.   ...
72.   }
73.
74.   @Override
75.   public ListeElectorale[] calculerSieges(ListeElectorale[] listesElectorales) {
76. ...
77.   }
78.
79.   // liste des messages d'erreur d'une exception
80.   private List<String> getMessagesForException(Exception exception) {
81.       // on récupère la liste des messages d'erreur de l'exception
82.       Throwable cause = exception;
83.       List<String> erreurs = new ArrayList<String>();
84.       while (cause != null) {
85.           // on récupère le message seulement s'il est !=null et non blanc
86.           String message = cause.getMessage();
87.           if (message != null) {
88.               message = message.trim();
89.               if (message.length() != 0) {
90.                   erreurs.add(message);
91.               }
92.           }
93.           // cause suivante
94.           cause = cause.getCause();
95.       }
96.       return erreurs;
97.   }
98. }
99.
```

Le type [Response] utilisé ligne 37 est la réponse du serveur web / jSON décrite au paragraphe 14.7, page 269 ;

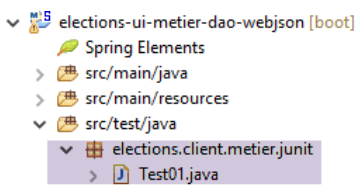
Travail à faire : en suivant le paragraphe 13.6.3.7, page 255, complétez la classe [ElectionsMetier] ;

15.7 Le test Junit

Revenons à l'architecture client / serveur en cours de construction :



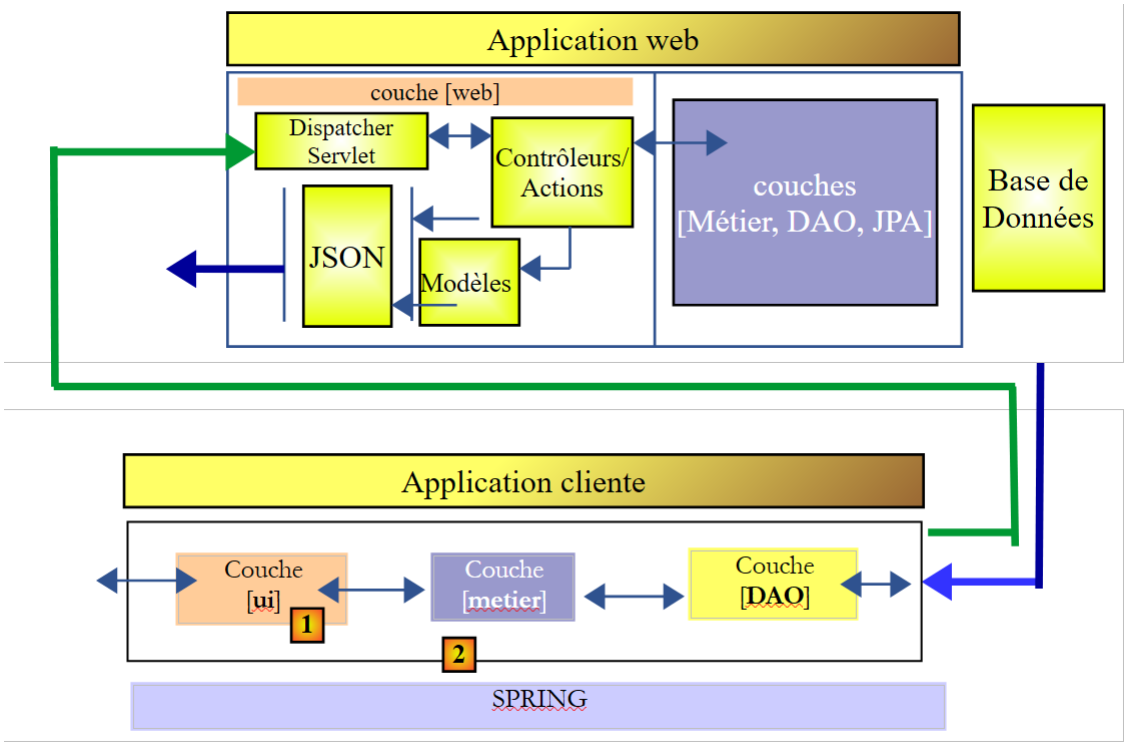
La couche [JUnit] [1] communique avec la couche [Métier] du serveur [5] au travers des couches [2-4]. En faisant en sorte que les couches [Métier] [2] et [5] aient la même interface, on rend les couches [2-4] transparentes. La couche [1] a l'impression de communiquer directement avec la couche [5]. Le point intéressant est qu'en [1] on va pouvoir utiliser le test JUnit qui avait été utilisé pour tester la couche [Métier] [5].



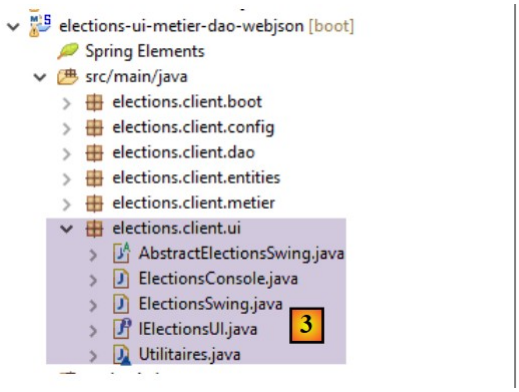
Travail à faire : passez le test JUnit du projet pour vérifier votre implémentation et du serveur et de son client.

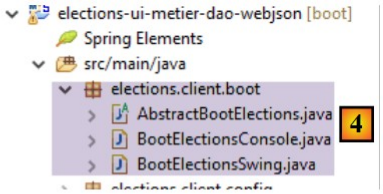
15.8 Implémentation de la couche [UI]

Revenons à l'architecture que nous voulons construire :



Maintenant que la couche [métier] [2] a été construite et testée, nous pouvons construire la couche [ui] [1].





Comme l'interface [IElectionsMetier] de la couche [métier] est identique à celle du projet décrit au paragraphe 8, page 125, nous pouvons en [3], copier le projet de la couches [ui] du paragraphe 10, page 139. Ce projet était un projet Netbeans. Il suffit de faire un copier / coller des classes Java concernées de Netbeans vers Eclipse. Ceci fait, il y a des ajustements de packages et d'imports à faire.

On fera de même pour les classes exécutables du package [elections.client.boot] [4].

La classe [AbstractBootElections] est la suivante :

```
1. package elections.client.boot;
2.
3. import org.springframework.context.annotation.AnnotationConfigApplicationContext;
4.
5. import elections.client.config.UiConfig;
6. import elections.client.entities.ElectionsException;
7. import elections.client.ui.IElectionsUI;
8.
9. public abstract class AbstractBootElections {
10.
```

```
11.    // récupération du contexte Spring
12.    protected AnnotationConfigApplicationContext ctx;
13.
14.    public void run() {
15.        // instanciation couche [ui]
16.        IElectionsUI electionsUI = null;
17.        try {
18.            // récupération du contexte Spring
19.            ctx = new AnnotationConfigApplicationContext(UiConfig.class);
20.            // récupération de la couche [ui]
21.            electionsUI = getUI();
22.
23.  ...
```

- ligne 19 : le contexte Spring défini par la classe de configuration [UiConfig] est instancié. Cette classe est la suivante :

La classe [UiConfig] est la suivante :

```
1. package elections.client.config;
2.
3. import org.springframework.context.annotation.ComponentScan;
4. import org.springframework.context.annotation.Import;
5.
6. @Import(MetierConfig.class)
7. @ComponentScan(basePackages = { "elections.client.ui" })
8. public class UiConfig {
9. }
```

- ligne 6 : on importe les beans de la couche [métier] ;
- ligne 7 : on indique qu'il y des beans Spring dans le package [elections.client.ui] ;

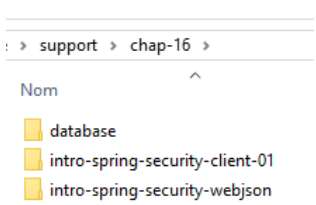
Travail à faire : vérifiez que les versions console et swing de la couche [ui] fonctionnent.

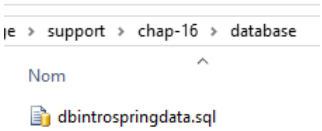
---

16 [Cours] : Sécuriser l'accès à un service web avec Spring Security

Mots clés : architecture multicouche, Spring, injection de dépendances, service web / jSON sécurisé, client / serveur

16.1 Support

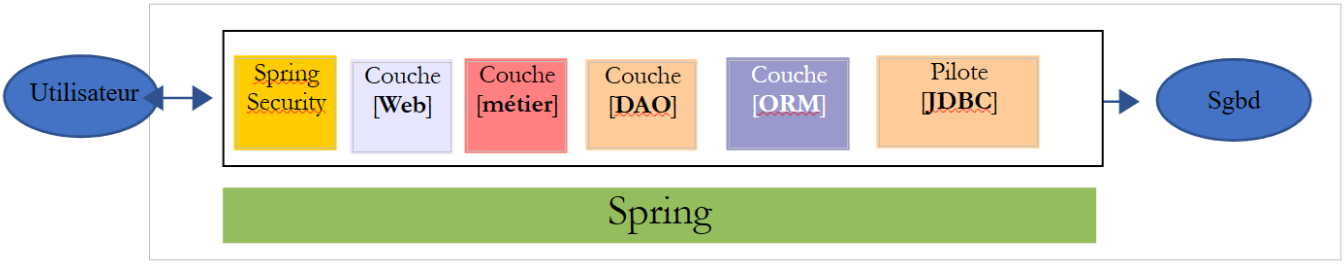




Les projets de ce chapitre seront trouvés dans le dossier [support / chap-16]. Le script SQL sert à générer la base de données nécessaire aux tests.

16.2 La place de Spring Security dans une application Web

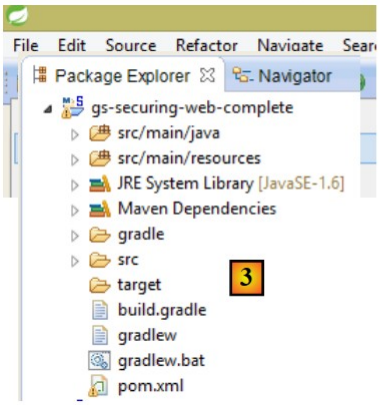
Situons Spring Security dans le développement d'une application Web. Le plus souvent, celle-ci sera bâtie sur une architecture multicouche telle que la suivante :

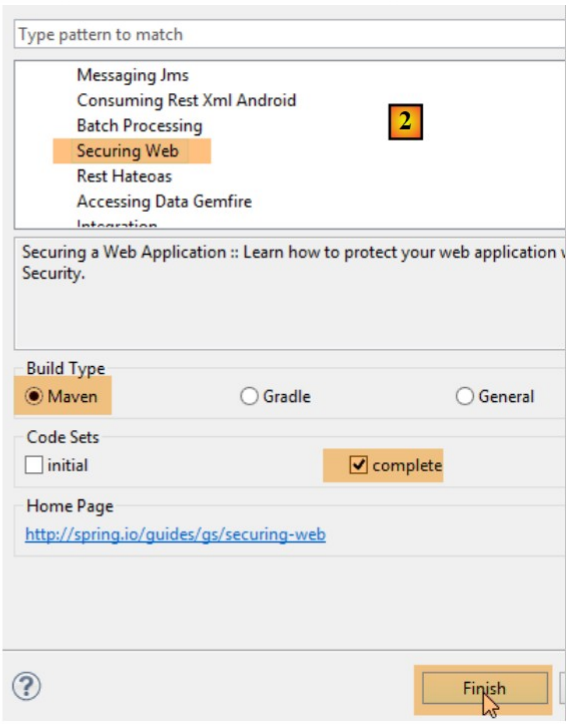


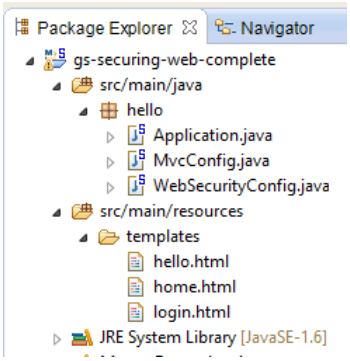
- la couche [**Spring Security**] n'accorde l'accès à la couche [web] qu'aux utilisateurs autorisés.

16.3 Un tutoriel sur Spring Security

Nous allons de nouveau importer un guide Spring en suivant les étapes 1 à 3 ci-dessous :







Le projet se compose des éléments suivants :

- dans le dossier [templates], on trouve les pages HTML du projet ;
- [Application] : est la classe exécutable du projet ;
- [MvcConfig] : est la classe de configuration de Spring MVC ;
- [WebSecurityConfig] : est la classe de configuration de Spring Security ;

16.3.1 Configuration Maven

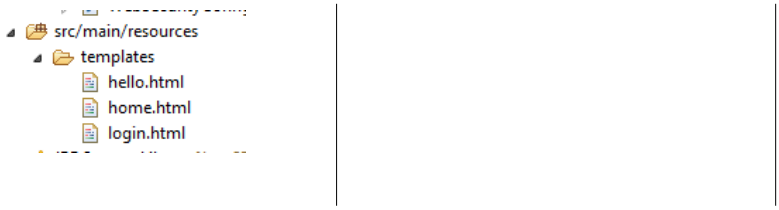
Le projet [3] est un projet Maven. Examinons son fichier [pom.xml] pour connaître ses dépendances :

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3.   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
4.   <modelVersion>4.0.0</modelVersion>
5.
6.   <groupId>org.springframework</groupId>
7.   <artifactId>gs-securing-web</artifactId>
8.   <version>0.1.0</version>
```

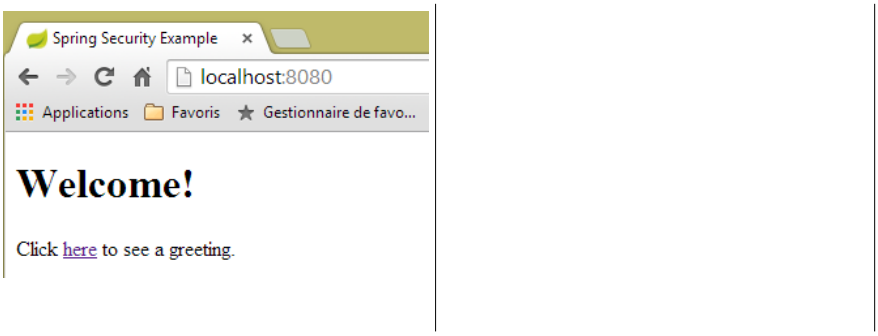
```
9.
10.   <parent>
11.      <groupId>org.springframework.boot</groupId>
12.      <artifactId>spring-boot-starter-parent</artifactId>
13.      <version>1.2.3.RELEASE</version>
14.   </parent>
15.
16.   <dependencies>
17.      <dependency>
18.         <groupId>org.springframework.boot</groupId>
19.         <artifactId>spring-boot-starter-thymeleaf</artifactId>
20.      </dependency>
21.      <!-- tag::security[] -->
22.      <dependency>
23.         <groupId>org.springframework.boot</groupId>
24.         <artifactId>spring-boot-starter-security</artifactId>
25.      </dependency>
26.      <!-- end::security[] -->
27.   </dependencies>
28.
29.   <properties>
30.      <start-class>hello.Application</start-class>
31.   </properties>
32.
33.   <build>
34.      <plugins>
35.         <plugin>
36.            <groupId>org.springframework.boot</groupId>
37.            <artifactId>spring-boot-maven-plugin</artifactId>
38.         </plugin>
39.      </plugins>
40.   </build>
41.
42. </project>
```

- lignes 10-14 : le projet est un projet Spring Boot ;
- lignes 17-20 : dépendance sur le framework [Thymeleaf] ;
- lignes 22-25 : dépendance sur le framework Spring Security ;

16.3.2 Les vues Thymeleaf



La vue [home.html] est la suivante :




```
1. <!DOCTYPE html>
```

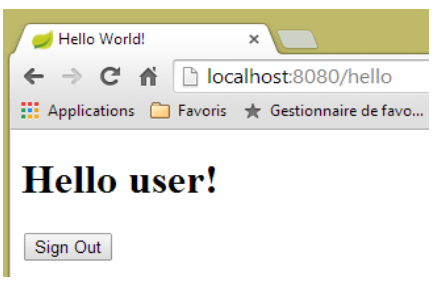
```
2. <html xmlns="http://www.w3.org/1999/xhtml"
3.   xmlns:th="http://www.thymeleaf.org"
4.   xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity3">
5. <head>
6. <title>Spring Security Example</title>
7. </head>
8. <body>
9.   <h1>Welcome!</h1>
10.
11.    <p>
12.        Click <a th:href="@{/hello}">here</a> to see a greeting.
13.    </p>
14.  </body>
15. </html>
```

- ligne 12 : l'attribut [th:href="@{/hello}"] va générer l'attribut [href] de la balise [<a>]. La valeur [@{/hello}] va générer le chemin [<context>/hello] où [context] est le contexte de l'application web ;

Le code HTML généré est le suivant :

```
1. <!DOCTYPE html>
2.
3. <html xmlns="http://www.w3.org/1999/xhtml" xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity3">
4.   <head>
5.     <title>Spring Security Example</title>
6.   </head>
7.   <body>
8.     <h1>Welcome!</h1>
9.
10.        <p>
11.            Click
12.            <a href="/hello">here</a>
13.            to see a greeting.
14.        </p>
15.    </body>
16. </html>
```

La vue [hello.html] est la suivante :




```
1. <!DOCTYPE html>
2. <html xmlns="http://www.w3.org/1999/xhtml"
3.   xmlns:th="http://www.thymeleaf.org"
4.   xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity3">
5. <head>
6. <title>Hello World!</title>
7. </head>
8. <body>
9.   <h1 th:inline="text">Hello [[${#httpServletRequest.remoteUser}]]!</h1>
10.    <form th:action="@{/logout}" method="post">
11.        <input type="submit" value="Sign Out" />
12.    </form>
13.  </body>
14. </html>
```

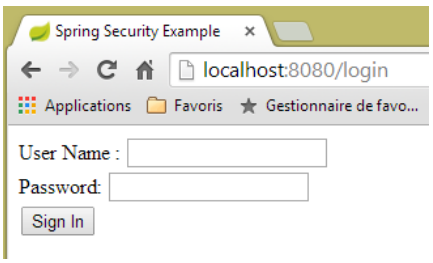
- ligne 9 : L'attribut [th:inline="text"] va générer le texte de la balise [<h1>]. Ce texte contient une expression $ qui doit être évaluée. L'élément [[${#httpServletRequest.remoteUser}]] est la valeur de l'attribut [RemoteUser] de la requête HTTP courante. C'est le nom de l'utilisateur connecté ;
- ligne 10 : un formulaire HTML. L'attribut [th:action="@{/logout}"] va générer l'attribut [action] de la balise [form]. La valeur [@{/logout}] va générer le chemin [<context>/logout] où [context] est le contexte de l'application web ;

Le code HTML généré est le suivant :

```
1. <!DOCTYPE html>
2.
3. <html xmlns="http://www.w3.org/1999/xhtml" xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity3">
4.   <head>
5.     <title>Hello World!</title>
6.   </head>
7.   <body>
8.     <h1>Hello user!</h1>
9.     <form method="post" action="/logout">
10.          <input type="submit" value="Sign Out" />
11.          <input type="hidden" name="_csrf" value="b152e5b9-d1a4-4492-b89d-b733fe521c91" />
12.      </form>
13.   </body>
14. </html>
```

- ligne 8 : la traduction de *Hello [[${#httpServletRequest.remoteUser}]]!*;
- ligne 9 : la traduction de *@{/logout}* ;
- ligne 11 : un champ caché appelé (attribut *name*) **_csrf** ;

La vue [login.html] est la suivante :




```
1. <!DOCTYPE html>
2. <html xmlns="http://www.w3.org/1999/xhtml"
3.   xmlns:th="http://www.thymeleaf.org"
4.   xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity3">
5. <head>
6. <title>Spring Security Example</title>
7. </head>
8. <body>
9.   <div th:if="${param.error}">Invalid username and password.</div>
10.    <div th:if="${param.logout}">You have been logged out.</div>
11.    <form th:action="@{/login}" method="post">
12.        <div>
13.            <label> User Name : <input type="text" name="username" />
14.            </label>
15.        </div>
16.        <div>
17.            <label> Password: <input type="password" name="password" />
18.            </label>
19.        </div>
20.        <div>
21.            <input type="submit" value="Sign In" />
22.        </div>
23.    </form>
24. </body>
25. </html>
```

- ligne 9 : l'attribut [th:if="${param.error}"] fait que la balise ne sera générée que si l'URL qui affiche la page de login contient le paramètre [error] (http://context/login?error);
- ligne 10 : l'attribut [th:if="${param.logout}"] fait que la balise ne sera générée que si l'URL qui affiche la page de login contient le paramètre [logout] (http://context/login?logout);
- lignes 11-23 : un formulaire HTML ;
- ligne 11 : le formulaire sera posté à l'URL [<context>/login] où <context> est le contexte de l'application web ;
- ligne 13 : un champ de saisie nommé [username] ;
- ligne 17 : un champ de saisie nommé [password] ;

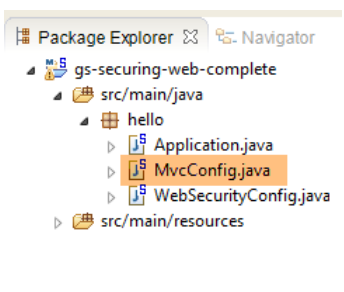
Le code HTML généré est le suivant :

```
1. <!DOCTYPE html>
2.
```

```
3. <html xmlns="http://www.w3.org/1999/xhtml" xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity3">
4.   <head>
5.     <title>Spring Security Example </title>
6.   </head>
7.   <body>
8.
9.     <div>
10.          You have been logged out.
11.      </div>
12.      <form method="post" action="/login">
13.          <div>
14.              <label>
15.                  User Name :
16.                  <input type="text" name="username" />
17.              </label>
18.          </div>
19.          <div>
20.              <label>
21.                  Password:
22.                  <input type="password" name="password" />
23.              </label>
24.          </div>
25.          <div>
26.              <input type="submit" value="Sign In" />
27.          </div>
28.          <input type="hidden" name="_csrf" value="ef809b0a-88b4-4db9-bc53-342216b77632" />
29.      </form>
30.  </body>
31. </html>
```

On notera ligne 28 que Thymeleaf a ajouté un champ caché nommé [**_csrf**].

16.3.3 Configuration Spring MVC



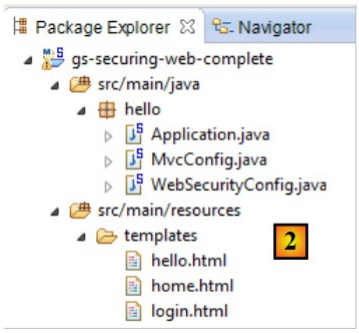
La classe [MvcConfig] configure le framework Spring MVC :

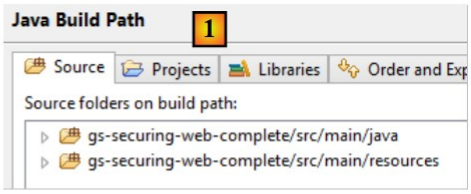
```
1. package hello;
2.
3. import org.springframework.context.annotation.Configuration;
4. import org.springframework.web.servlet.config.annotation.ViewControllerRegistry;
5. import org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter;
6.
7. @Configuration
8. public class MvcConfig extends WebMvcConfigurerAdapter {
9.
10.    @Override
11.    public void addViewControllers(ViewControllerRegistry registry) {
12.        registry.addViewController("/home").setViewName("home");
13.        registry.addViewController("/").setViewName("home");
14.        registry.addViewController("/hello").setViewName("hello");
15.        registry.addViewController("/login").setViewName("login");
16.    }
17.
18. }
```

- ligne 7 : l'annotation [@Configuration] fait de la classe [MvcConfig] une classe de configuration ;
- ligne 8 : la classe [MvcConfig] étend la classe [WebMvcConfigurerAdapter] pour en redéfinir certaines méthodes ;
- ligne 10 : redéfinition d'une méthode de la classe parent ;
- lignes 11- 16 : la méthode [addViewControllers] permet d'associer des URL à des vues HTML. Les associations suivantes y sont faites :

URL	vue
/, /home	/templates/home.html
/hello	/templates/hello.html
/login	/templates/login.html

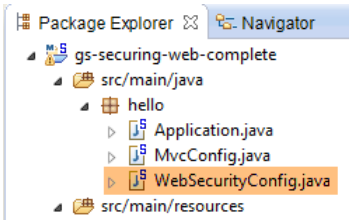
Le suffixe [html] et le dossier [templates] sont les valeurs par défaut utilisées par Thymeleaf. Elles peuvent être changées par configuration. Le dossier [templates] doit être à la racine du Classpath du projet :





Ci-dessus [1], les dossiers [java] et [resources] sont tous les deux des dossier source (source folders). Cela implique que leur contenu sera à la racine du Classpath du projet. Donc en [2], les dossiers [hello] et [templates] seront à la racine du Classpath.

16.3.4 Configuration Spring Security



La classe [WebSecurityConfig] configure le framework Spring Security :

```
1. package hello;
2.
3. import org.springframework.context.annotation.Configuration;
4. import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
5. import org.springframework.security.config.annotation.web.builders.HttpSecurity;
6. import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
7. import org.springframework.security.config.annotation.web.servlet.configuration.EnableWebMvcSecurity;
8.
9. @Configuration
10.  @EnableWebMvcSecurity
11.  public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
12.    @Override
13.    protected void configure(HttpSecurity http) throws Exception {
14.        http.authorizeRequests().antMatchers("/", "/home").permitAll().anyRequest().authenticated();
15.        http.formLogin().loginPage("/login").permitAll().and().logout().permitAll();
16.    }
17.
18.    @Override
19.    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
20.        auth.inMemoryAuthentication().withUser("user").password("password").roles("USER");
```

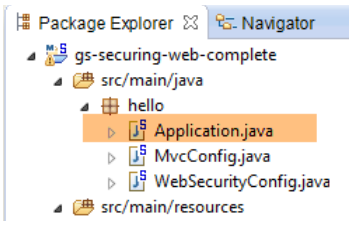
```
21.    }
22.  }
```

- ligne 9 : l'annotation [@Configuration] fait de la classe [WebSecurityConfig] une classe de configuration ;
- ligne 10 : l'annotation [@EnableWebSecurity] fait de la classe [WebSecurityConfig] une classe de configuration de Spring Security ;
- ligne 11 : la classe [WebSecurity] étend la classe [WebSecurityConfigurerAdapter] pour en redéfinir certaines méthodes ;
- ligne 12 : redéfinition d'une méthode de la classe parent ;
- lignes 13- 16 : la méthode [configure(HttpSecurity http)] est redéfinie pour définir les droits d'accès aux différentes URL de l'application ;
- ligne 14 : la méthode [http.authorizeRequests()] permet d'associer des URL à des droits d'accès. Les associations suivantes y sont faites :

URL	règle	code
/, /home	accès sans être authentifié	`http.authorizeRequests().antMatchers("/", "/home").permitAll()`
autres URL	accès authentifié uniquement	`http.anyRequest().authenticated();`

- ligne 15 : définit la méthode d'authentification. L'authentification se fait via un formulaire d'URL [/login] accessible à tous [http.formLogin().loginPage("/login").permitAll()]. La déconnexion (logout) est également accessible à tous ;
- lignes 19-21 : redéfinissent la méthode [configure(AuthenticationManagerBuilder auth)] qui gère les utilisateurs ;
- ligne 20 : l'autentification se fait avec des utilisateurs définis en " dur " [auth.inMemoryAuthentication()]. Un utilisateur est ici défini avec le login [user], le mot de passe [password] et le rôle [USER]. On peut accorder les mêmes droits à des utilisateurs ayant le même rôle ;

16.3.5 Classe exécutable



La classe [Application] est la suivante :

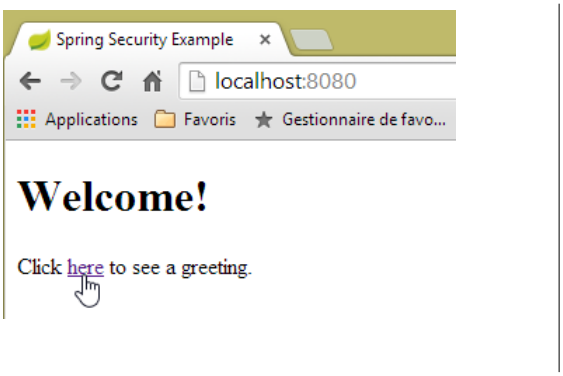
```
1. package hello;
2.
3. import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
4. import org.springframework.boot.SpringApplication;
5. import org.springframework.context.annotation.ComponentScan;
6. import org.springframework.context.annotation.Configuration;
7.
8. @EnableAutoConfiguration
9. @Configuration
10.  @ComponentScan
11.  public class Application {
12.
13.      public static void main(String[] args) throws Throwable {
14.          SpringApplication.run(Application.class, args);
15.      }
16.
17.  }
```

- ligne 8 : l'annotation [@EnableAutoConfiguration] demande à Spring Boot (ligne 3) de faire la configuration que le développeur n'aura pas fait explicitement ;
- ligne 9 : fait de la classe [Application] une classe de configuration Spring ;
- ligne 10 : demande le scan du dossier de la classe [Application] afin de rechercher des composants Spring. Les deux classes [MvcConfig] et [WebSecurityConfig] vont être ainsi découvertes car elles ont l'annotation [@Configuration] ;
- ligne 13 : la méthode [main] de la classe exécutable ;
- ligne 14 : la méthode statique [SpringApplication.run] est exécutée avec comme paramètre la classe de configuration [Application]. Nous avons déjà rencontré ce processus et nous savons que le serveur Tomcat embarqué dans les

dépendances Maven du projet va être lancé et le projet déployé dessus. Nous avons vu que quatre URL étaient gérées [/, /home, /login, /hello] et que certaines étaient protégées par des droits d'accès.

16.3.6 Tests de l'application

Commençons par demander l'URL [/] qui est l'une des quatre URL acceptées. Elle est associée à la vue [/templates/home.html] :



L'URL demandée [/] est accessible à tous. C'est pourquoi nous l'avons obtenue. Le lien [here] est le suivant :

```
Click <a href="/hello">here</a> to see a greeting.
```

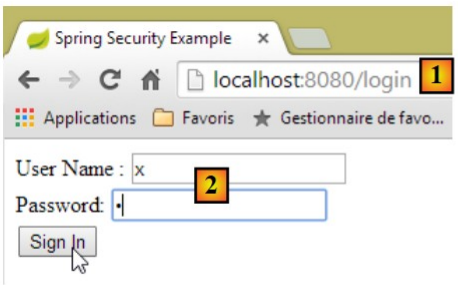
L'URL [/hello] va être demandée lorsqu'on va cliquer sur le lien. Celle-ci est protégée :

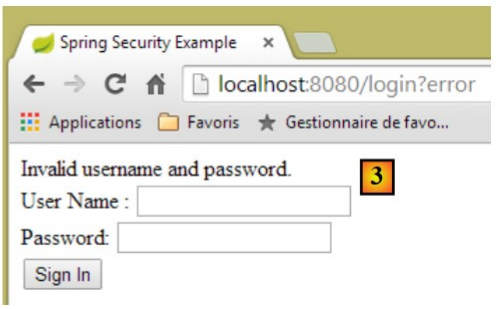
URL	règle	code
/, /home	accès sans être authentifié	`http.authorizeRequests().antMatchers("/", "/home").permitAll()`
autres URL	accès authentifié uniquement	`http.anyRequest().authenticated();`

Il faut être authentifié pour l'obtenir. Spring Security va alors rediriger le navigateur client vers la page d'authentification. D'après la configuration vue, c'est la page d'URL [/login]. Celle-ci est accessible à tous :

```
http.formLogin().loginPage("/login").permitAll().and().logout().permitAll();
```

Nous l'obtenons donc [1] :





Le code source de la page obtenue est le suivant :

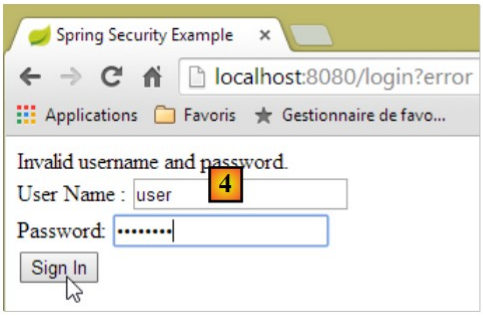
```
1. <!DOCTYPE html>
2.
3. <html xmlns="http://www.w3.org/1999/xhtml" xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity3">
4. ...
5.   <form method="post" action="/login">
6. ...
7.       <input type="hidden" name="_csrf" value="87bea06a-a177-459d-b279-c6068a7ad3eb" />
8.    </form>
9. </body>
10.  </html>
```

- ligne 7, un champ caché apparaît qui n'est pas dans la page [login.html] d'origine. C'est Thymeleaf qui l'a ajouté. Ce code appelé CSRF (Cross Site Request Forgery) vise à éliminer une faille de sécurité. Ce jeton doit être renvoyé à Spring Security avec l'authentification pour que cette dernière soit acceptée ;

Nous nous souvenons que seul l'utilisateur *user/password* est reconnu par Spring Security. Si nous entrons autre chose en [2], nous obtenons la même page avec un message d'erreur en [3]. Spring Security a redirigé le navigateur vers l'URL [http://localhost:8080/login?error]. La présence du paramètre [error] a déclenché l'affichage de la balise :

```
<div th:if="${param.error}">Invalid username and password.</div>
```

Maintenant, entrons les valeurs attendues user/password [4] :



- en [4], nous nous identifions ;
- en [5], Spring Security nous redirige vers l'URL [/hello] car c'est l'URL que nous demandions lorsque nous avons été redirigés vers la page de login. L'identité de l'utilisateur a été affichée par la ligne suivante de [hello.html] :

```
    <h1 th:inline="text">Hello [[${#httpServletRequest.remoteUser}]]!</h1>
```

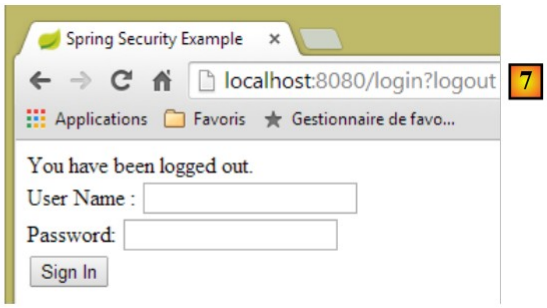
La page [5] affiche le formulaire suivant :

```
1.   <form th:action="@{/logout}" method="post">
2.     <input type="submit" value="Sign Out" />
3. </form>
```

Lorsqu'on clique sur le bouton [Sign Out], un POST va être fait sur l'URL [/logout]. Celle-ci comme l'URL [/login] est accessible à tous :

```
http.formLogin().loginPage("/login").permitAll().and().logout().permitAll();
```

Dans notre association URL / vues, nous n'avons rien défini pour l'URL [/logout]. Que va-t-il se passer ? Essayons :



- en [6], nous cliquons sur le bouton [Sign Out] ;
- en [7], nous voyons que nous avons été redirigés vers l'URL [http://localhost:8080/login?logout]. C'est Spring Security qui a demandé cette redirection. La présence du paramètre [logout] dans l'URL a fait afficher la ligne suivante de la vue :

```
<div th:if="${param.logout}">You have been logged out.</div>
```

16.3.7 Conclusion

Dans l'exemple précédent, nous aurions pu écrire l'application web d'abord puis la sécuriser ensuite. Spring Security n'est pas intrusif. On peut mettre en place la sécurité d'une application web déjà écrite. Par ailleurs, nous avons découvert les points suivants :

- il est possible de définir une page d'authentification ;
- l'authentification doit être accompagnée du jeton CSRF délivré par Spring Security ;
- si l'authentification échoue, on est redirigé vers la page d'authentification avec de plus un paramètre **error** dans l'URL ;
- si l'authentification réussit, on est redirigé vers la page demandée lorsque l'authentification a eu lieu. Si on demande directement la page d'authentification sans passer par une page intermédiaire, alors Spring Security nous redirige vers l'URL [/] (ce cas n'a pas été présenté) ;
- on se déconnecte en demandant l'URL [/logout] avec un POST. Spring Security nous redirige alors vers la page d'authentification avec le paramètre **logout** dans l'URL ;

Toutes ces conclusions reposent sur des comportements par défaut de Spring Security. Ces comportements peuvent être changés par configuration en redéfinissant certaines méthodes de la classe [`WebSecurityConfigurerAdapter`].

Le tutoriel précédent nous aidera peu dans la suite. Nous allons en effet utiliser :

- une base de données pour stocker les utilisateurs, leurs mots de passe et leurs rôles ;
- une authentification par entête HTTP ;

On trouve assez peu de tutoriels pour ce qu'on veut faire ici. La solution qui va être proposée est un assemblage de codes trouvés ici et là.

16.4 Mise en place de la sécurité sur le service web / json des produits

16.4.1 La base de données

La base de données [dbintrospringdata] évolue pour prendre en compte les utilisateurs, leurs mots de passe et leur rôles. Trois nouvelles tables apparaissent :

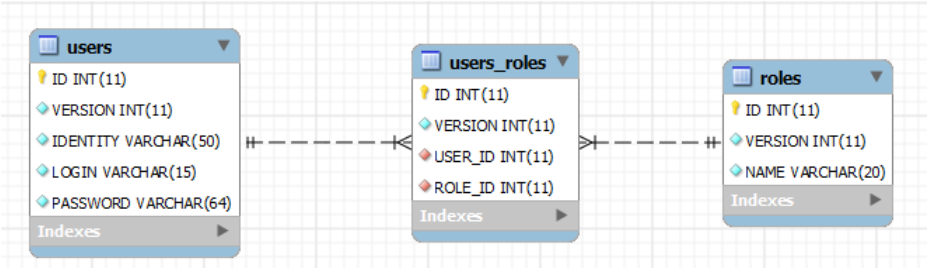


Table [USERS] : les utilisateurs

- ID : clé primaire ;
- VERSION : colonne de versioning de la ligne ;
- IDENTITY : une identité descriptive de l'utilisateur ;
- LOGIN : le login de l'utilisateur ;
- PASSWORD : son mot de passe ;

Dans la table USERS, les mots de passe ne sont pas stockés en clair :

ID	LOGIN	NAME	PASSWORD	VERSION
1	admin	admin	$2a$10$qB9fuPvC/qTzJbjhlYAmleWyaeVVPoeHNF68UQRFse1...	1
2	user	user	$2a$10$JmuEgAW/FcOlthOq9C6b0u5yNUhvdB.xOHOu/e48dEf...	1
3	guest	guest	$2a$10$i0RlLI01pt2xx9oKGxs8D.5n3/O.vvPw92d7Rp3eCfD...	1
4	y	y	$2a$10$E8TJhq4TCk1MW2ESyB.m/eWvvyH173P0LUaSHQtgm6k...	0

L'algorithme qui crypte les mots de passe est l'algorithme BCRYPT.

Table [ROLES] : les rôles

- ID : clé primaire ;
- VERSION : colonne de versioning de la ligne ;
- NAME : nom du rôle. Par défaut, Spring Security attend des noms de la forme ROLE_XX, par exemple ROLE_ADMIN ou ROLE_GUEST ;

ID	VERSION	NAME
1	1	ROLE_ADMIN
2	1	ROLE_USER

Table [USERS_ROLES] : table de jointure USERS / ROLES

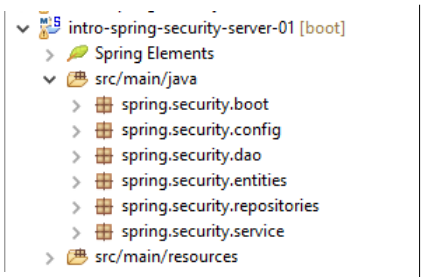
Un utilisateur peut avoir plusieurs rôles, un rôle peut rassembler plusieurs utilisateurs. On a une relation **plusieurs à plusieurs** matérialisée par la table [USERS_ROLES].

- ID : clé primaire ;
- VERSION : colonne de versioning de la ligne ;
- USER_ID : identifiant d'un utilisateur ;
- ROLE_ID : identifiant d'un rôle ;

ID	VERSION	USER_ID	ROLE_ID
1	1	1	2
2	1	2	1

16.4.2 Le projet Eclipse

Nous créons le projet Eclipse suivant :



- en [1] : le nouveau projet avec les paquetages suivants :
  - [spring.security.entities] : contient les entités JPA correspondant aux trois nouvelles tables de la base de données ;
  - [spring.security.repositories] : contient les [repositories] de Spring Data associés aux trois nouvelles tables ;
  - [spring.security.dao] : contient un service s'appuyant sur les [repositories] ;
  - [spring.security.config] : contient la configuration du projet et notamment celle des accès sécurisés au service web ;
  - [spring.security.boot] : conteint la classe de lancement du service web sécurisé ;

16.4.3 La configuration Maven

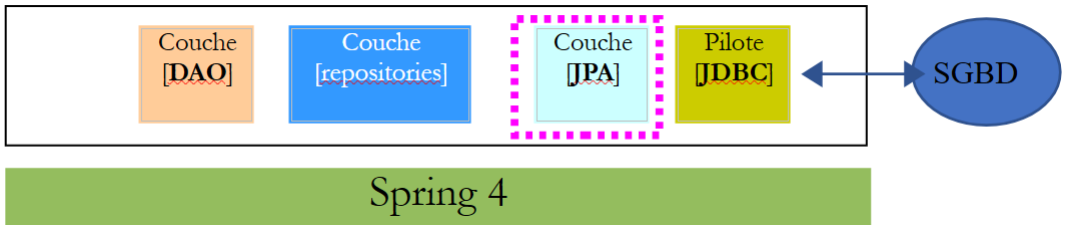
Le nouveau projet est un projet Maven configuré par le fichier [pom.xml] suivant :

```
1. <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2.   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
```

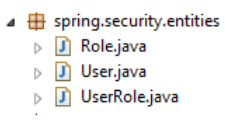
```
3.  <modelVersion>4.0.0</modelVersion>
4.  <groupId>istia.st.spring.security</groupId>
5.  <artifactId>intro-spring-security-server-01</artifactId>
6.  <version>0.0.1-SNAPSHOT</version>
7.
8.  <name>intro-spring-security-server-01</name>
9.  <description>démo spring security</description>
10.
11.   <properties>
12.       <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
13.       <java.version>1.8</java.version>
14.   </properties>
15.
16.   <parent>
17.       <groupId>org.springframework.boot</groupId>
18.       <artifactId>spring-boot-starter-parent</artifactId>
19.       <version>1.2.7.RELEASE</version>
20.   </parent>
21.
22.   <dependencies>
23.       <dependency>
24.           <groupId>istia.st.webjson</groupId>
25.           <artifactId>intro-server-webjson-01</artifactId>
26.           <version>0.0.1-SNAPSHOT</version>
27.       </dependency>
28.       <!-- Spring security -->
29.       <dependency>
30.           <groupId>org.springframework.boot</groupId>
31.           <artifactId>spring-boot-starter-security</artifactId>
32.       </dependency>
33.       <!-- Spring logs -->
34.       <dependency>
35.           <groupId>org.springframework.boot</groupId>
36.           <artifactId>spring-boot-starter-logging</artifactId>
37.       </dependency>
38.       <!-- Spring Boot -->
39.       <dependency>
40.           <groupId>org.springframework.boot</groupId>
41.           <artifactId>spring-boot</artifactId>
42.       </dependency>
43.       <!-- Spring Boot Test -->
44.       <dependency>
45.           <groupId>org.springframework.boot</groupId>
46.           <artifactId>spring-boot-starter-test</artifactId>
47.           <scope>test</scope>
48.       </dependency>
49.   </dependencies>
50.   <!-- plugins -->
51.   <build>
52.       <plugins>
53.           <plugin>
54.               <artifactId>maven-assembly-plugin</artifactId>
55.               <configuration>
56.                   <descriptorRefs>
57.                       <descriptorRef>jar-with-dependencies</descriptorRef>
58.                   </descriptorRefs>
59.               </configuration>
60.           </plugin>
61.           <plugin>
62.               <groupId>org.apache.maven.plugins</groupId>
63.               <artifactId>maven-surefire-plugin</artifactId>
64.               <version>2.18.1</version>
65.           </plugin>
66.       </plugins>
67.   </build>
68.
69. </project>
```

- lignes 23-27 : on reprend l'existant avec l'archive du service web / json étudié ;
- lignes 29-32 : la dépendance qui amène les classes de Spring Security ;
- lignes 34-37 : la bibliothèque de logs ;
- lignes 39-42 : la bibliothèque permettant d'utiliser les annotations Spring Boot ;
- lignes 44-48 : la bibliothèque nécessaire aux tests ;

16.4.4 Les nouvelles entités [JPA]



La couche JPA définit trois nouvelles entités :



La classe [User] est l'image de la table [USERS] :

```
1. package spring.security.entities;
2.
3. import javax.persistence.Column;
4. import javax.persistence.Entity;
5. import javax.persistence.Table;
6.
7. import spring.data.entities.AbstractEntity;
8.
9. @Entity
10.  @Table(name = "USERS")
11.  public class User extends AbstractEntity {
12.
13.    // propriétés
14.    @Column(name = "NAME")
15.    private String name;
16.    @Column(name = "LOGIN")
17.    private String login;
18.    @Column(name = "PASSWORD")
19.    private String password;
20.
21.    // constructeur
22.    public User() {
23.    }
24.
25.    public User(String name, String login, String password) {
26.        this.name = name;
27.        this.login = login;
28.        this.password = password;
29.    }
30.
31.    // getters et setters
32.  ...
33.  }
```

- ligne 11 : la classe étend la classe [AbstractEntity] déjà utilisée pour les autres entités ;

La classe [Role] est l'image de la table [ROLES] :

```
1. package spring.security.entities;
2.
3. import javax.persistence.Column;
4. import javax.persistence.Entity;
5. import javax.persistence.Table;
6.
7. import spring.data.entities.AbstractEntity;
8.
9. @Entity
10.  @Table(name = "ROLES")
11.  public class Role extends AbstractEntity {
12.
13.    // propriétés
14.    @Column(name="NAME")
15.    private String name;
```

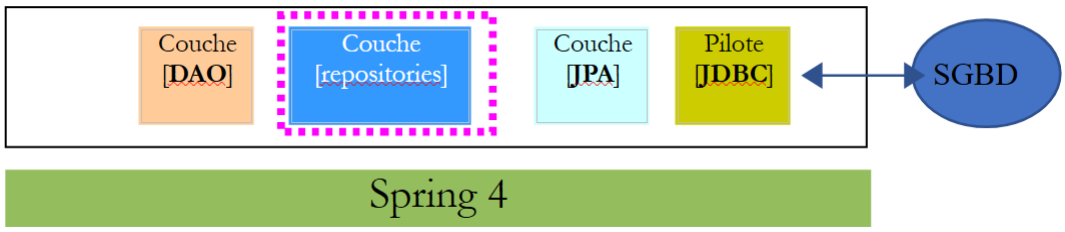
```
16.
17.    // constructeurs
18.    public Role() {
19.    }
20.
21.    public Role(String name) {
22.        this.name = name;
23.    }
24.
25.    // getters et setters
26.    public String getName() {
27.        return name;
28.    }
29.
30.    public void setName(String name) {
31.        this.name = name;
32.    }
33.
34.  }
```

La classe [UserRole] est l'image de la table [USERS_ROLES] :

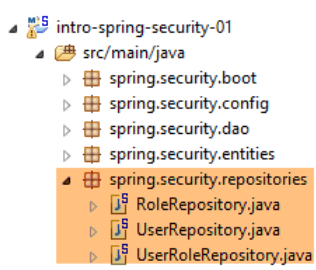
```
1. package spring.security.entities;
2.
3. import javax.persistence.Column;
4. import javax.persistence.Entity;
5. import javax.persistence.JoinColumn;
6. import javax.persistence.ManyToOne;
7. import javax.persistence.Table;
8.
9. import spring.data.entities.AbstractEntity;
10.
11.  @Entity
12.  @Table(name = "USERS_ROLES")
13.  public class UserRole extends AbstractEntity {
14.
15.    // les clés étrangères
16.    @Column(name = "USER_ID", insertable = false, updatable = false)
17.    private Long userId;
18.    @Column(name = "ROLE_ID", insertable = false, updatable = false)
19.    private Long roleId;
20.
21.    // un UserRole référence un User
22.    @ManyToOne
23.    @JoinColumn(name = "USER_ID")
24.    private User user;
25.
26.    // un UserRole référence un Role
27.    @ManyToOne
28.    @JoinColumn(name = "ROLE_ID")
29.    private Role role;
30.
31.    // constructeurs
32.    public UserRole() {
33.
34.    }
35.
36.    public UserRole(User user, Role role) {
37.        this.user = user;
38.        this.role = role;
39.    }
40.
41.    // getters et setters
42.  ...
43.  }
44.
```

- lignes 22-24 : matérialisent la clé étrangère de la table [USERS_ROLES] vers la table [USERS] ;
- lignes 27-29 : matérialisent la clé étrangère de la table [USERS_ROLES] vers la table [ROLES] ;

16.4.5 Les [repositories]



Chacune des entités JPA précédentes est gérée par un [repository] Spring Data :



L'interface [UserRepository] gère les accès aux entités [User] :

```
1. package spring.security.repositories;
2.
3. import org.springframework.data.jpa.repository.Query;
4. import org.springframework.data.repository.CrudRepository;
5.
6. import spring.security.entities.Role;
7. import spring.security.entities.User;
8.
9. public interface UserRepository extends CrudRepository<User, Long> {
10.
11.    // liste des rôles d'un utilisateur identifié par son id
12.    @Query("select ur.role from UserRole ur where ur.user.id=?1")
13.    Iterable<Role> getRoles(long id);
14.
15.    // liste des rôles d'un utilisateur identifié par son login unique
16.    @Query("select ur.role from UserRole ur where ur.user.login=?1 and ur.user.password=?2")
17.    Iterable<Role> getRoles(String login, String password);
18.
19.    // recherche d'un utilisateur via son login
20.    User findUserByLogin(String login);
21. }
```

- ligne 9 : l'interface [UserRepository] étend l'interface [CrudRepository] de Spring Data (ligne 4) ;
- lignes 12-13 : la méthode [getRoles(User user)] permet d'avoir tous les rôles d'un utilisateur identifié par son [id]
- lignes 16-17 : idem mais pour un utilisateur identifié pas ses login / mot de passe ;
- ligne 20 : pour trouver un utilisateur via son login ;

L'interface [RoleRepository] gère les accès aux entités [Role] :

```
1. package spring.security.repositories;
2.
3. import org.springframework.data.repository.CrudRepository;
4.
5. import spring.security.entities.Role;
6.
7. public interface RoleRepository extends CrudRepository<Role, Long> {
8.
9.   // recherche d'un rôle via son nom
10.    Role findRoleByName(String name);
11.
12. }
```

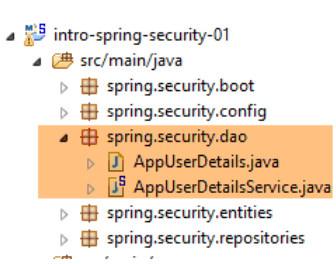
- ligne 7 : l'interface [RoleRepository] étend l'interface [CrudRepository] ;
- ligne 10 : on peut chercher un rôle via son nom ;

L'interface [UserRoleRepository] gère les accès aux entités [UserRole] :

```
1. package spring.security.repositories;
2.
3. import org.springframework.data.repository.CrudRepository;
4.
5. import spring.security.entities.UserRole;
6.
7. public interface UserRoleRepository extends CrudRepository<UserRole, Long> {
8.
9. }
```

- ligne 5 : l'interface [UserRoleRepository] se contente d'étendre l'interface [CrudRepository] sans lui ajouter de nouvelles méthodes ;

16.4.6 Les classes de gestion des utilisateurs et des rôles



Spring Security impose la création d'une classe implémentant l'interface [UsersDetail] suivante :

Method Summary

Collection<GrantedAuthority>	getAuthorities() Returns the authorities granted to the user.
String	getPassword() Returns the password used to authenticate the user.
String	getUsername() Returns the username used to authenticate the user.
boolean	isAccountNonExpired() Indicates whether the user's account has expired.
boolean	isAccountNonLocked() Indicates whether the user is locked or unlocked.
boolean	isCredentialsNonExpired() Indicates whether the user's credentials (password) has expired.
boolean	isEnabled() Indicates whether the user is enabled or disabled.

Cette interface est ici implémentée par la classe [AppUserDetails] :

```
1. package spring.security.dao;
2.
3. import java.util.ArrayList;
4. import java.util.Collection;
5.
6. import org.springframework.security.core.GrantedAuthority;
7. import org.springframework.security.core.authority.SimpleGrantedAuthority;
8. import org.springframework.security.core.userdetails.UserDetails;
9.
10.  import spring.security.entities.Role;
11.  import spring.security.entities.User;
```

```
12. import spring.security.repositories.UserRepository;
13.
14. public class AppUserDetails implements UserDetails {
15.
16.   private static final long serialVersionUID = 1L;
17.
18.   // propriétés
19.   private User user;
20.   private UserRepository userRepository;
21.
22.   // constructeurs
23.   public AppUserDetails() {
24.   }
25.
26.   public AppUserDetails(User user, UserRepository userRepository) {
27.       this.user = user;
28.       this.userRepository = userRepository;
29.   }
30.
31.   // ------------------------interface
32.   @Override
33.   public Collection<? extends GrantedAuthority> getAuthorities() {
34.       Collection<GrantedAuthority> authorities = new ArrayList<>();
35.       for (Role role : userRepository.getRoles(user.getId())) {
36.           authorities.add(new SimpleGrantedAuthority(role.getName()));
37.       }
38.       return authorities;
39.   }
40.
41.   @Override
42.   public String getPassword() {
43.       return user.getPassword();
44.   }
45.
46.   @Override
47.   public String getUsername() {
48.       return user.getLogin();
49.   }
50.
51.   @Override
52.   public boolean isAccountNonExpired() {
53.       return true;
54.   }
55.
56.   @Override
57.   public boolean isAccountNonLocked() {
58.       return true;
59.   }
60.
61.   @Override
62.   public boolean isCredentialsNonExpired() {
63.       return true;
64.   }
65.
66.   @Override
67.   public boolean isEnabled() {
68.       return true;
69.   }
70.
71.   // getters et setters
72.   ...
73. }
```

- ligne 14 : la classe [AppUserDetails] implémente l'interface [UserDetails] ;
- lignes 19-20 : la classe encapsule un utilisateur (ligne 19) et le repository qui permet d'avoir les détails de cet utilisateur (ligne 20) ;
- lignes 26-29 : le constructeur qui instancie la classe avec un utilisateur et son repository ;
- lignes 32-36 : implémentation de la méthode [getAuthorities] de l'interface [UserDetails]. Elle doit construire une collection d'éléments de type [GrantedAuthority] ou dérivé. Ici, nous utilisons le type dérivé [SimpleGrantedAuthority] (ligne 36) qui encapsule le nom d'un des rôles de l'utilisateur de la ligne 19 ;
- lignes 35-37 : on parcourt la liste des rôles de l'utilisateur de la ligne 19 pour construire une liste d'éléments de type [SimpleGrantedAuthority] ;
- lignes 42-44 : implémentent la méthode [getPassword] de l'interface [UserDetails]. On rend le mot de passe de l'utilisateur de la ligne 19 ;
- lignes 42-44 : implémentent la méthode [getUserName] de l'interface [UserDetails]. On rend le login de l'utilisateur de la ligne 19 ;
- lignes 51-54 : le compte de l'utilisateur n'expire jamais ;
- lignes 56-59 : le compte de l'utilisateur n'est jamais bloqué ;
- lignes 61-64 : les identifiants de l'utilisateur n'expirent jamais ;
- lignes 66-69 : le compte de l'utilisateur est toujours actif ;

Spring Security impose également l'existence d'une classe implémentant l'interface [AppUserDetailsService] :

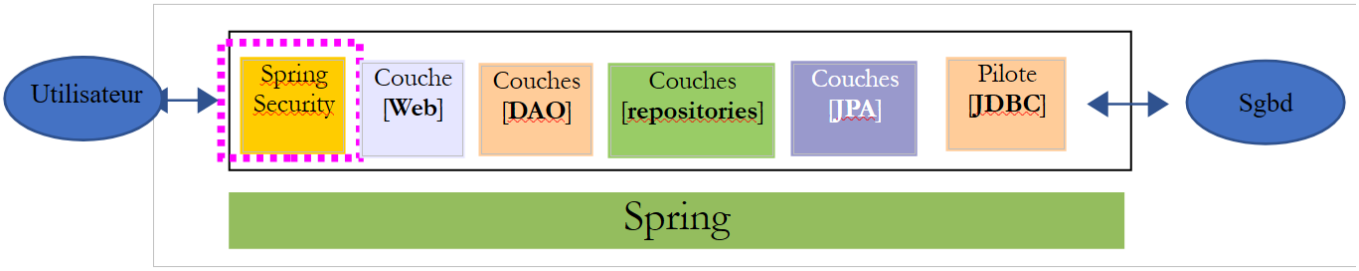
Method Summary	
UserDetails	loadUserByUsername(String username) Locates the user based on the username.

Cette interface est implémentée par la classe [AppUserDetailsService] suivante :

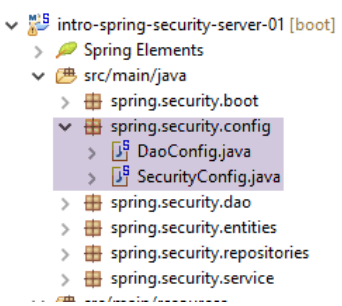
```
1. package spring.security.dao;
2.
3. import org.springframework.beans.factory.annotation.Autowired;
4. import org.springframework.security.core.userdetails.UserDetails;
5. import org.springframework.security.core.userdetails.UserDetailsService;
6. import org.springframework.security.core.userdetails.UsernameNotFoundException;
7. import org.springframework.stereotype.Service;
8.
9. import spring.security.entities.User;
10.  import spring.security.repositories.UserRepository;
11.
12.  @Service
13.  public class AppUserDetailsService implements UserDetailsService {
14.
15.    @Autowired
16.    private UserRepository userRepository;
17.
18.    @Override
19.    public UserDetails loadUserByUsername(String login) throws UsernameNotFoundException {
20.        // on cherche l'utilisateur via son login
21.        User user = userRepository.findUserByLogin(login);
22.        // trouvé ?
23.        if (user == null) {
24.            throw new UsernameNotFoundException(String.format("login [%s] inexistant", login));
25.        }
26.        // on rend les détails de l'utilsateur
27.        return new AppUserDetails(user, userRepository);
28.    }
29.
30.  }
```

- ligne 12 : la classe sera un composant Spring, donc disponible dans son contexte ;
- lignes 15-16 : le composant [UserRepository] sera injecté ici ;
- lignes 19-28 : implémentation de la méthode [loadUserByUsername] de l'interface [UserDetailsService] (ligne 10). Le paramètre est le login de l'utilisateur ;
- ligne 21 : l'utilisateur est recherché via son login ;
- lignes 23-25 : s'il n'est pas trouvé, une exception est lancée ;
- ligne 27 : un objet [AppUserDetails] est construit et rendu. Il est bien de type [UserDetails] (ligne 19) ;

16.4.7 La configuration du projet



Le projet est configuré par deux classes :



La classe [DaoConfig] configure la couche [DAO] amenée par le nouveau projet :

```
1. package spring.security.config;
2.
3. import org.springframework.context.annotation.Bean;
4. import org.springframework.context.annotation.ComponentScan;
5. import org.springframework.context.annotation.Import;
6. import org.springframework.data.jpa.repository.config.EnableJpaRepositories;
7.
8. @EnableJpaRepositories(basePackages = { "spring.security.repositories" })
9. @ComponentScan(basePackages = { "spring.security.dao" })
10.  @Import({ spring.data.config.DaoConfig.class })
11.  public class DaoConfig {
12.
13.    // constantes
14.    final static private String[] ENTITIES_PACKAGES = { "spring.data.entities", "spring.security.entities" };
15.
16.    @Bean
17.    public String[] packagesToScan() {
18.        return ENTITIES_PACKAGES;
19.    }
20.
21.  }
```

- ligne 10 : on importe la classe de configuration [spring.data.config.DaoConfig] du projet [intro-spring-data-01] qui implémente la couche [DAO] des produits et catégories ;
- ligne 8 : on désigne les dossiers du projet courant contenant des [repositories] Spring Data ;
- ligne 9 : on désigne les dossiers du projet courant contenant des composants Spring concernant la couche [DAO] ;
- ligne 14 : on désigne les dossiers contenant des entités JPA. Il y a celles du projet [intro-spring-data-01] et celles du projet du serveur sécurisé. Cette information fait l'objet du bean des lignes 16-19. Ce bean redéfinit le bean de même nom du projet [intro-spring-data-01] :

```
1.    final static private String[] ENTITIES_PACKAGES = { "spring.data.entities" };
2.
3.    // EntityManagerFactory
4.    @Bean
5.    public EntityManagerFactory entityManagerFactory(JpaVendorAdapter jpaVendorAdapter, DataSource dataSource) {
6.      LocalContainerEntityManagerFactoryBean factory = new LocalContainerEntityManagerFactoryBean();
7.      factory.setJpaVendorAdapter(jpaVendorAdapter);
8.      factory.setPackagesToScan(packagesToScan());
9.      factory.setDataSource(dataSource);
10.         factory.afterPropertiesSet();
11.         return factory.getObject();
12.     }
13.
14.       @Bean
15.     public String[] packagesToScan() {
16.         return ENTITIES_PACKAGES;
17.  }
```

Dans la couche [DAO], la ligne 8 scanne les dossiers indiqués ligne 1. A cause de la redéfinition du bean des lignes 14-17 dans le projet sécurisé (lignes 16-19), la ligne 8 ci-dessus va désormais scanner les dossiers ["spring.data.entities", "spring.security.entities"]. Il est à noter que la classe importée ligne 10 de la classe [spring.security.config.DaoConfig] doit comporter l'annotation [@Configuration], sinon le phénomène qui vient d'être expliqué ne fonctionne pas.

La classe [SecurityConfig] configure l'aspect sécurité du projet. Nous avons déjà rencontré une classe de configuration de Spring Security :

```
1. package hello;
2.
3. import org.springframework.context.annotation.Configuration;
4. import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
5. import org.springframework.security.config.annotation.web.builders.HttpSecurity;
6. import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
7. import org.springframework.security.config.annotation.web.servlet.configuration.EnableWebMvcSecurity;
8.
9. @Configuration
10.  @EnableWebMvcSecurity
11.  public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
12.    @Override
13.    protected void configure(HttpSecurity http) throws Exception {
14.        http.authorizeRequests().antMatchers("/", "/home").permitAll().anyRequest().authenticated();
15.        http.formLogin().loginPage("/login").permitAll().and().logout().permitAll();
16.    }
17.
18.    @Override
19.    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
20.        auth.inMemoryAuthentication().withUser("user").password("password").roles("USER");
21.    }
22.  }
```

Nous allons suivre la même démarche :

- ligne 11 : définir une classe qui étend la classe [WebSecurityConfigurerAdapter] ;
- ligne 13 : définir une méthode [configure(HttpSecurity http)] qui définit les droits d'accès aux différentes URL du service web ;
- ligne 19 : définir une méthode [configure(AuthenticationManagerBuilder auth)] qui définit les utilisateurs et leurs rôles ;

La classe [SecurityConfig] sera la suivante :

```
1. package spring.security.config;
2.
3. import org.springframework.beans.factory.annotation.Autowired;
4. import org.springframework.context.annotation.ComponentScan;
5. import org.springframework.context.annotation.Import;
6. import org.springframework.http.HttpMethod;
7. import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
8. import org.springframework.security.config.annotation.web.builders.HttpSecurity;
9. import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
10.  import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
11.  import org.springframework.security.config.http.SessionCreationPolicy;
12.  import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
13.
14.  import spring.security.dao.AppUserDetailsService;
15.
16.  @EnableWebSecurity
17.  @ComponentScan(basePackages = { "spring.security.service" })
18.  @Import({ spring.webjson.config.AppConfig.class, DaoConfig.class })
19.  public class SecurityConfig extends WebSecurityConfigurerAdapter {
20.
21.    @Autowired
22.    private AppUserDetailsService appUserDetailsService;
23.
24.    // sécurisation
25.    private boolean activateSecurity = true;
26.
27.    @Override
28.    protected void configure(AuthenticationManagerBuilder registry) throws Exception {
29.        // l'authentification est faite par le bean [appUserDetailsService]
30.        // le mot de passe est crypté par l'algorithme de hachage BCrypt
31.        registry.userDetailsService(appUserDetailsService).passwordEncoder(new BCryptPasswordEncoder());
32.    }
33.
34.    @Override
35.    protected void configure(HttpSecurity http) throws Exception {
36.        // CSRF
37.        http.csrf().disable();
38.        // application sécurisée ?
39.        if (activateSecurity) {
40.            // le mot de passe est transmis par le header Authorization: Basic xxxx
41.            http.httpBasic();
42.            // la méthode HTTP OPTIONS doit être autorisée pour tous
43.            http.authorizeRequests() //
44.                    .antMatchers(HttpMethod.OPTIONS, "/", "/**").permitAll();
45.            // seul le rôle ADMIN peut utiliser l'application
46.            http.authorizeRequests() //
47.                    .antMatchers("/", "/**") // toutes les URL
```

```
48.                 .hasRole("ADMIN");
49.         // pas de session
50.         http.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);
51.     }
52.   }
53. }
```

- ligne 16 : pour activer les éléments de Spring Security ;
- ligne 17 : on rajoute les composants Spring du package [spring.security.service] ;
- ligne 18 : on importe les beans de la couche [DAO] que l'on vient de présenter ainsi que ceux du serveur web / jSON non sécurisé ;
- lignes 21-22 : la classe [AppUserDetails] qui donne accès aux utilisateurs de l'application est injectée ;
- ligne 25 : un booléen qui sécurise (true) ou non (false) l'application web ;
- lignes 27-32 : la méthode [configure(HttpSecurity http)] définit les utilisateurs et leurs rôles. Elle reçoit en paramètre un type [AuthenticationManagerBuilder]. Ce paramètre est enrichi de deux informations (ligne 38) :
  - une référence sur le service [appUserDetailsService] de la ligne 22 qui donne accès aux utilisateurs enregistrés. On notera ici que le fait qu'ils soient enregistrés dans une base de données n'apparaît pas. Ils pourraient donc être dans un cache, délivrés par un service web, ...
  - le type de cryptage utilisé pour le mot de passe. On rappelle ici que nous avons utilisé l'algorithme BCrypt ;
- lignes 34-52 : la méthode [configure(HttpSecurity http)] définit les droits d'accès aux URL du service web ;
- ligne 37 : nous avons vu dans le projet d'introduction que par défaut Spring Security gérait un jeton CSRF (Cross Site Request Forgery) que l'utilisateur qui voulait s'authentifier devait renvoyer au serveur. Ici ce mécanisme est désactivé. Ceci allié au booléen (isSecured=false) permet d'utiliser l'application web sans sécurité ;
- ligne 41 : on active le mode d'authentification par entête HTTP. Le client devra envoyer l'entête HTTP suivant :

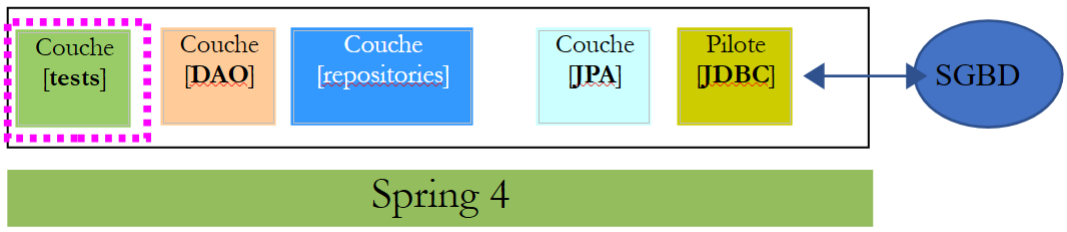
```
Authorization:Basic code
```

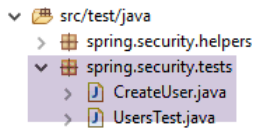
où code est le codage de la chaîne **login:password** par l'algorithme Base64. Par exemple, le codage Base64 de la chaîne *admin:admin* est *YWRtaW46YWRtaW4=*. Donc l'utilisateur de login [admin] et de mot de passe [admin] enverra l'entête HTTP suivant pour s'authentifier :

```
Authorization:Basic YWRtaW46YWRtaW4=
```

- lignes 46-48 : indiquent que toutes les URL du service web sont accessibles aux utilisateurs ayant le rôle [ROLE_ADMIN]. Cela veut dire qu'un utilisateur n'ayant pas ce rôle ne peut accéder au service web ;
- ligne 50 : en mode [session], un utilisateur qui s'est authentifié une fois n'a pas besoin de le faire pour ses accès suivants. Ici on désactive ce mode, aussi l'utilisateur devra-t-il s'authentifier à chaque accès ;

16.4.8 Tests de la couche [DAO]





Tout d'abord, nous créons une classe exécutable [CreateUser] capable de créer un utilisateur avec un rôle :

```
1. package sprin.security.tests;
2.
```

```
3. import org.springframework.context.annotation.AnnotationConfigApplicationContext;
4. import org.springframework.security.crypto.bcrypt.BCrypt;
5.
6. import spring.security.config.DaoConfig;
7. import spring.security.entities.Role;
8. import spring.security.entities.User;
9. import spring.security.entities.UserRole;
10.  import spring.security.repositories.RoleRepository;
11.  import spring.security.repositories.UserRepository;
12.  import spring.security.repositories.UserRoleRepository;
13.
14.  public class CreateUser {
15.
16.    public static void main(String[] args) {
17.        // syntaxe : login password roleName
18.
19.        // il faut trois paramètres
20.        if (args.length != 3) {
21.            System.out.println("Syntaxe : [pg] user password role");
22.            System.exit(0);
23.        }
24.        // on récupère les paramètres
25.        String login = args[0];
26.        String password = args[1];
27.        String roleName = String.format("ROLE_%s", args[2].toUpperCase());
28.        // contexte Spring
29.        AnnotationConfigApplicationContext context = new AnnotationConfigApplicationContext(DaoConfig.class);
30.        UserRepository userRepository = context.getBean(UserRepository.class);
31.        RoleRepository roleRepository = context.getBean(RoleRepository.class);
32.        UserRoleRepository userRoleRepository = context.getBean(UserRoleRepository.class);
33.        // le rôle existe-t-il déjà ?
34.        Role role = roleRepository.findRoleByName(roleName);
35.        // s'il n'existe pas on le crée
36.        if (role == null) {
37.            role = roleRepository.save(new Role(roleName));
38.        }
39.        // l'utilisateur existe-t-il déjà ?
40.        User user = userRepository.findUserByLogin(login);
41.        // s'il n'existe pas on le crée
42.        if (user == null) {
43.            // on hashe le mot de passe avec bcrypt
44.            String crypt = BCrypt.hashpw(password, BCrypt.gensalt());
45.            // on sauvegarde l'utilisateur
46.            user = userRepository.save(new User(login, login, crypt));
47.            // on crée la relation avec le rôle
48.            userRoleRepository.save(new UserRole(user, role));
49.        } else {
50.            // l'utilisateur existe déjà- a-t-il le rôle demandé ?
51.            boolean trouvé = false;
52.            for (Role r : userRepository.getRoles(user.getId())) {
53.                if (r.getName().equals(roleName)) {
54.                    trouvé = true;
55.                    break;
56.                }
57.            }
58.            // si pas trouvé, on crée la relation avec le rôle
59.            if (!trouvé) {
60.                userRoleRepository.save(new UserRole(user, role));
61.            }
62.        }
63.
64.        // fermeture contexte Spring
65.        context.close();
66.        // fin
67.        System.out.println("Travail terminé...");
68.    }
69.
70.  }
```

- ligne 17 : la classe attend trois arguments définissant un utilisateur : son login, son mot de passe, son rôle ;
- lignes 25-27 : les trois paramètres sont récupérés ;
- ligne 29 : le contexte Spring est construit à partir de la classe de configuration [AppConfig] ;
- lignes 30-32 : on récupère les références des trois [Repository] qui peuvent nous être utiles pour créer l'utilisateur ;
- ligne 34 : on regarde si le rôle existe déjà ;
- lignes 36-38 : si ce n'est pas le cas, on le crée en base. Il aura un nom du type [ROLE_XX] ;
- ligne 40 : on regarde si le login existe déjà ;
- lignes 42-49 : si le login n'existe pas, on le crée en base ;
- ligne 44 : on crypte le mot de passe. On utilise ici, la classe [BCrypt] de Spring Security (ligne 4). On a donc besoin des archives de ce framework. Le fichier [pom.xml] inclut cette dépendance :

```
1.      <dependency>
2.          <groupId>org.springframework.boot</groupId>
```

```
3.          <artifactId>spring-boot-starter-security</artifactId>
4. </dependency>
```

- ligne 46 : l'utilisateur est persisté en base ;
- ligne 48 : ainsi que la relation qui le lie à son rôle ;
- lignes 51-57 : cas où le login existe déjà – on regarde alors si parmi ses rôles se trouve déjà le rôle qu'on veut lui attribuer ;
- ligne 59-61 : si le rôle cherché n'a pas été trouvé, on crée une ligne dans la table [USERS_ROLES] pour relier l'utilisateur à son rôle ;
- on ne s'est pas protégé des exceptions éventuelles. C'est une classe de soutien pour créer rapidement un utilisateur avec un rôle.

Lorsqu'on exécute la classe avec les arguments [x x guest], on obtient en base les résultats suivants :

Table [USERS]

ID	LOGIN	NAME	PASSWORD	VERSION
1	admin	admin	$2a$10$qB9fuPvC/qTzJbjhlYAmleWyaeVVPoeHNF68UQRFse1...	1
2	user	user	$2a$10$JmuEgAW/FcOlthOq9C6b0u5yNUhvdB.xOHOu/e48dEf...	1
3	guest	guest	$2a$10$i0RlLI01pt2xx9oKGxs8D.5n3/O.vvPw92d7Rp3eCfD...	1

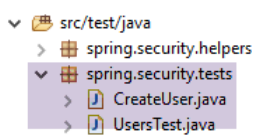
Table [ROLES]

ID	NAME	VERSION
1	ROLE_ADMIN	1
2	ROLE_USER	1
3	ROLE_GUEST	1

Table [USERS_ROLES]

ID	VERSION	ROLE_ID	USER_ID
1	1	1	1
2	1	2	2
3	1	3	3

Considérons maintenant la seconde classe [UsersTest] qui est un test JUnit :




```
1. package spring.security.tests;
2.
3. import java.util.List;
4.
5. import org.junit.Assert;
6. import org.junit.Test;
7. import org.junit.runner.RunWith;
8. import org.springframework.beans.factory.annotation.Autowired;
```

```
9. import org.springframework.boot.test.SpringApplicationConfiguration;
10.  import org.springframework.security.core.authority.SimpleGrantedAuthority;
11.  import org.springframework.security.crypto.bcrypt.BCrypt;
12.  import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
13.
14.  import com.fasterxml.jackson.core.JsonProcessingException;
15.  import com.fasterxml.jackson.databind.ObjectMapper;
16.  import com.google.common.collect.Lists;
17.
18.  import spring.security.config.DaoConfig;
19.  import spring.security.dao.AppUserDetails;
20.  import spring.security.dao.AppUserDetailsService;
21.  import spring.security.entities.Role;
22.  import spring.security.entities.User;
23.  import spring.security.repositories.UserRepository;
24.
25.  @SpringApplicationConfiguration(classes = DaoConfig.class)
26.  @RunWith(SpringJUnit4ClassRunner.class)
27.  public class UsersTest {
28.
29.    @Autowired
30.    private UserRepository userRepository;
31.    @Autowired
32.    private AppUserDetailsService appUserDetailsService;
33.
34.    // mappeur jSON
35.    private ObjectMapper mapper = new ObjectMapper();
36.
37.    @Test
38.    public void findAllUsersWithTheirRoles() throws JsonProcessingException {
39.        Iterable<User> users = userRepository.findAll();
40.        for (User user : users) {
41.            System.out.println(String.format("\n----------Utilisateur [%s]",mapper.writeValueAsString(user)));
42.            display("Roles :", userRepository.getRoles(user.getId()));
43.        }
44.    }
45.
46.    @Test
47.    public void findUserByLogin() {
48.        // on récupère l'utilisateur [admin]
49.        User user = userRepository.findUserByLogin("admin");
50.        // on vérifie que son mot de passe est [admin]
51.        Assert.assertTrue(BCrypt.checkpw("admin", user.getPassword()));
52.        // on vérifie le rôle de admin / admin
53.        List<Role> roles = Lists.newArrayList(userRepository.getRoles("admin", user.getPassword()));
54.        Assert.assertEquals(1L, roles.size());
55.        Assert.assertEquals("ROLE_ADMIN", roles.get(0).getName());
56.    }
57.
58.    @Test
59.    public void loadUserByUsername() {
60.        // on récupère l'utilisateur [admin]
61.        AppUserDetails userDetails = (AppUserDetails) appUserDetailsService.loadUserByUsername("admin");
62.        // on vérifie que son mot de passe est [admin]
63.        Assert.assertTrue(BCrypt.checkpw("admin", userDetails.getPassword()));
64.        // on vérifie le rôle de admin / admin
65.        @SuppressWarnings("unchecked")
66.        List<SimpleGrantedAuthority> authorities = (List<SimpleGrantedAuthority>) userDetails.getAuthorities();
67.        Assert.assertEquals(1L, authorities.size());
68.        Assert.assertEquals("ROLE_ADMIN", authorities.get(0).getAuthority());
69.    }
70.
71.    // méthode utilitaire - affiche les éléments d'une collection
72.    private void display(String message, Iterable<?> elements) throws JsonProcessingException {
73.        System.out.println(message);
74.        for (Object element : elements) {
75.            System.out.println(mapper.writeValueAsString(element));
76.        }
77.    }
78.  }
```

- lignes 37-44 : test visuel. On affiche tous les utilisateurs avec leurs rôles ;
- lignes 46-56 : on vérifie que l'utilisateur [admin] a le mot de passe [admin] et le rôle [ROLE_ADMIN] en utilisant le repository [UserRepository] ;
- ligne 51 : [admin] est le mot de passe en clair. En base, il est crypté selon l'algorithme BCrypt. La méthode [BCrypt.checkpw] permet de vérifier que le mot de passe en clair une fois crypté est bien égal à celui qui est en base ;
- lignes 58-69 : on vérifie que l'utilisateur [admin] a le mot de passe [admin] et le rôle [ROLE_ADMIN] en utilisant le service [appUserDetailsService] ;

L'exécution des tests réussit avec les logs suivants :

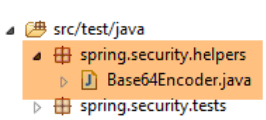
```
1. ----------Utilisateur
   [{"id":14,"version":0,"identity":"admin","login":"admin","password":"$2a$10$FN1LMKjPU46aPffh9Zaw4exJOLo51JJ
   PWrxqzak/eJrbt3CO9WzVG"}]
2. Roles :
3. {"id":6,"version":0,"name":"ROLE_ADMIN"}
4.
5. ----------Utilisateur
   [{"id":15,"version":0,"identity":"user","login":"user","password":"$2a$10$SJehR9Mv2VdyRZo9F0rXa.hKAoGLhJg6k
   SdyfExi40mEJrNOj0BTq"}]
6. Roles :
7. {"id":7,"version":0,"name":"ROLE_USER"}
8.
9. ----------Utilisateur [{"id":16,"version":0,"identity":"guest","login":"guest","password":"$2a$10$ubyWJb/
   vg2XZnUOAUjspZuz9jpHP3fIbPTbwQU115EtLdeSZ2PB7q"}]
10. Roles :
11. {"id":5,"version":0,"name":"ROLE_GUEST"}
12.
13. ----------Utilisateur
   [{"id":17,"version":0,"identity":"x","login":"x","password":"$2a$10$kEXA56wpKHFReVqwQTyWguKguK8I4uhA2zb6t3w
   Gxag8Dyv7AhLom"}]
14. Roles :
15. {"id":5,"version":0,"name":"ROLE_GUEST"}
```

16.4.9 Tests du service web

Nous allons tester le service web avec le client Chrome [Advanced Rest Client]. Nous allons avoir besoin de préciser l'entête HTTP d'authentification :

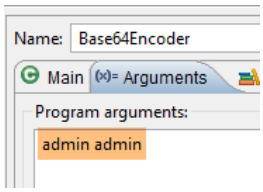
```
Authorization:Basic code
```

où [code] est le code Base64 de la chaîne [login:password]. Pour générer ce code, on peut utiliser le programme suivant :




```
1. package spring.security.helpers;
2.
3. import org.springframework.security.crypto.codec.Base64;
4.
5. public class Base64Encoder {
6.
7.   public static void main(String[] args) {
8.     // on attend deux arguments : login password
9.     if (args.length != 2) {
10.          System.out.println("Syntaxe : login password");
11.          System.exit(0);
12.      }
13.      // on récupère les deux arguments
14.      String chaîne = String.format("%s:%s", args[0], args[1]);
15.      // on encode la chaîne
16.      byte[] data = Base64.encode(chaîne.getBytes());
17.      // on affiche son encodage Base64
18.      System.out.println(new String(data));
19.   }
20.
21. }
```

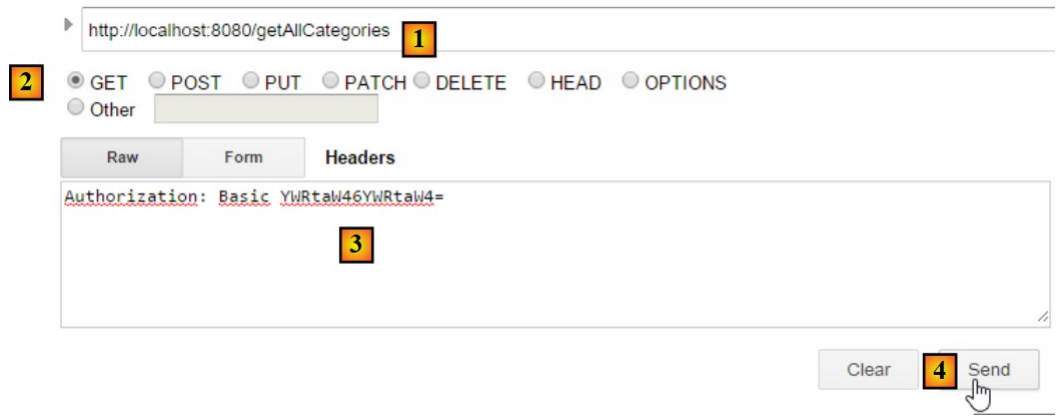
Si nous exécutons ce programme avec les deux arguments [admin admin] :



nous obtenons le résultat suivant :

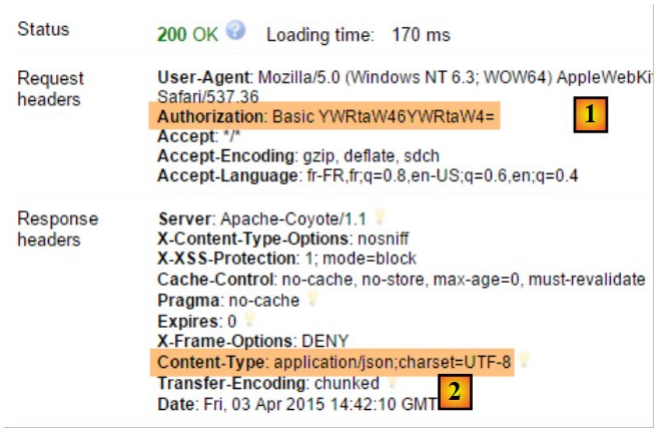
```
YWRtaW46YWRtaW4=
```

Maintenant que nous savons générer l'entête HTTP d'authentification, nous lançons le service web sécurisé, puis avec le client Chrome [Advanced Rest Client], nous demandons la liste des tous les produits :



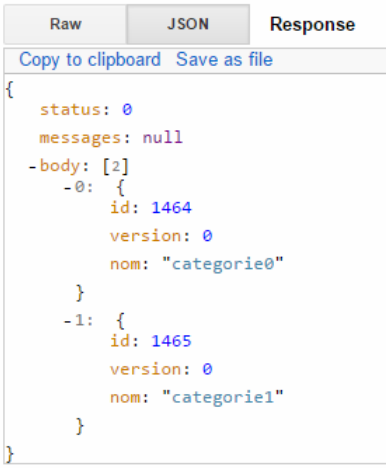
- en [1], nous demandons l'URL des catégories ;
- en [2], avec une méthode GET ;
- en [3], nous donnons l'entête HTTP de l'authentification. Le code [YWRtaW46YWRtaW4=] est le codage Base64 de la chaîne [admin:admin] ;
- en [4], nous envoyons la commande HTTP ;

La réponse du serveur est la suivante :

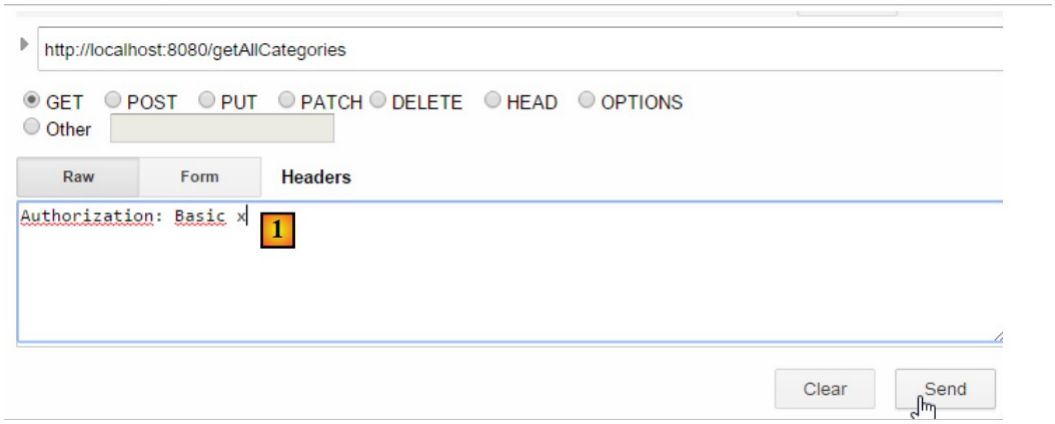


- en [1], l'entête HTTP d'authentification ;
- en [2], le serveur renvoie une réponse jSON ;

On obtient bien la liste des catégories :

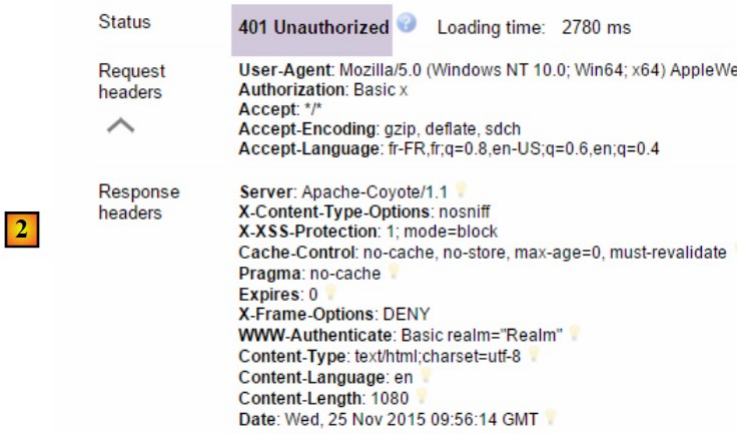


Tentons maintenant une requête HTTP avec un entête d'authentification incorrect. La réponse est alors la suivante :



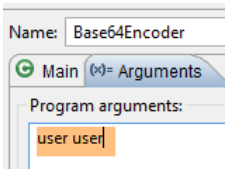
- en [1] : l'entête HTTP d'authentification ;

Nous obtenons la réponse suivante :



- en [2] : la réponse du service web ;

Maintenant, essayons l'utilisateur user / user. Il existe mais n'a pas accès au service web. Si nous exécutons le programme d'encodage Base64 avec les deux arguments [user user] :



nous obtenons le résultat suivant :

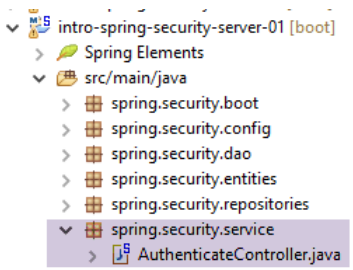
```
dXNlcjp1c2Vy
```

- en [1] : l'entête HTTP d'authentification erroné ;

- en [2] : la réponse du service web. Elle est différente de la précédente qui était [401 Unauthorized]. Cette fois-ci, l'utilisateur s'est authentifié correctement mais n'a pas les droits suffisants pour accéder à l'URL ;

Notre service web sécurisé est désormais opérationnel.

16.4.10 Une URL d'authentification



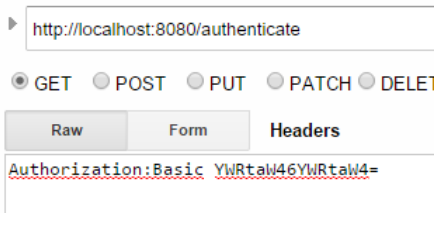
Nous allons créer une URL qui nous permettra de savoir si un utilisateur est autorisé ou non à accéder au service web. pour cela nous créons le nouveau contrôleur MVC [AuthenticateController] suivant :

```
1. package spring.security.service;
2.
3. import org.springframework.beans.factory.annotation.Autowired;
4. import org.springframework.context.ApplicationContext;
5. import org.springframework.stereotype.Controller;
6. import org.springframework.web.bind.annotation.RequestMapping;
7. import org.springframework.web.bind.annotation.RequestMethod;
8. import org.springframework.web.bind.annotation.ResponseBody;
9.
10.  import com.fasterxml.jackson.core.JsonProcessingException;
11.  import com.fasterxml.jackson.databind.ObjectMapper;
12.
13.  import spring.webjson.models.Response;
14.
15.  @Controller
16.  public class AuthenticateController {
17.
18.    // dépendances Spring
19.    @Autowired
20.    private ApplicationContext context;
21.
22.    @RequestMapping(value = "/authenticate", method = RequestMethod.GET, produces = "application/json; charset=UTF-8")
23.    @ResponseBody
24.    public String authenticate() throws JsonProcessingException {
25.        // réponse jSON
26.        ObjectMapper mapperResponse = context.getBean(ObjectMapper.class);
27.        return mapperResponse.writeValueAsString(new Response<Void>(0, null, null));
28.    }
29.
30.  }
```

- ligne 15 : la classe [AuthenticateController] est un contrôleur Spring. A ce titre elle expose des URL ;
- ligne 22 : expose l'URL [/authenticate] ;
- ligne 23 : le résultat de la méthode sera envoyé directement au client ;
- lignes 26-27 : la méthode se contente de renvoyer un objet [Response] vide mais avec un [status] égal à 0, montrant qu'il n'y a pas eu d'erreur ;

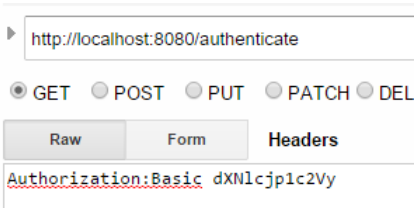
A quoi sert cette URL ? Lorsque nous voudrons simplement authentifier un utilisateur, nous la demanderons. Nous avons vu que si la couche de sécurité n'accepte pas cet utilisateur, elle renvoie une exception. Voici un exemple ;

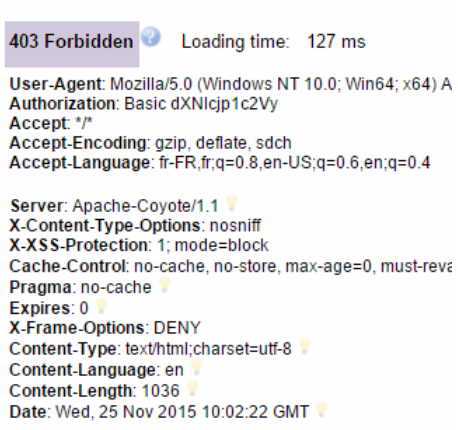
Avec l'utilisateur [admin:admin] :



On a une réponse vide mais pas d'exception.

Avec l'utilisateur [user:user] :





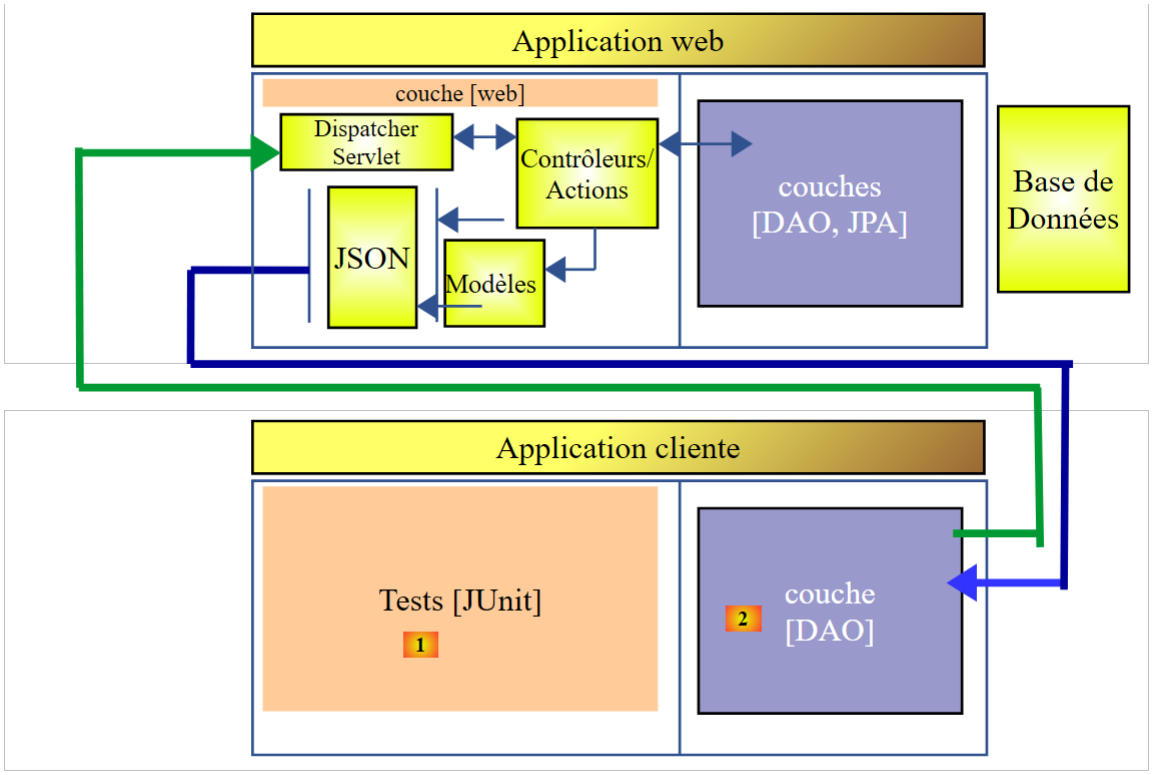
On a eu une exception.

16.4.11 Conclusion

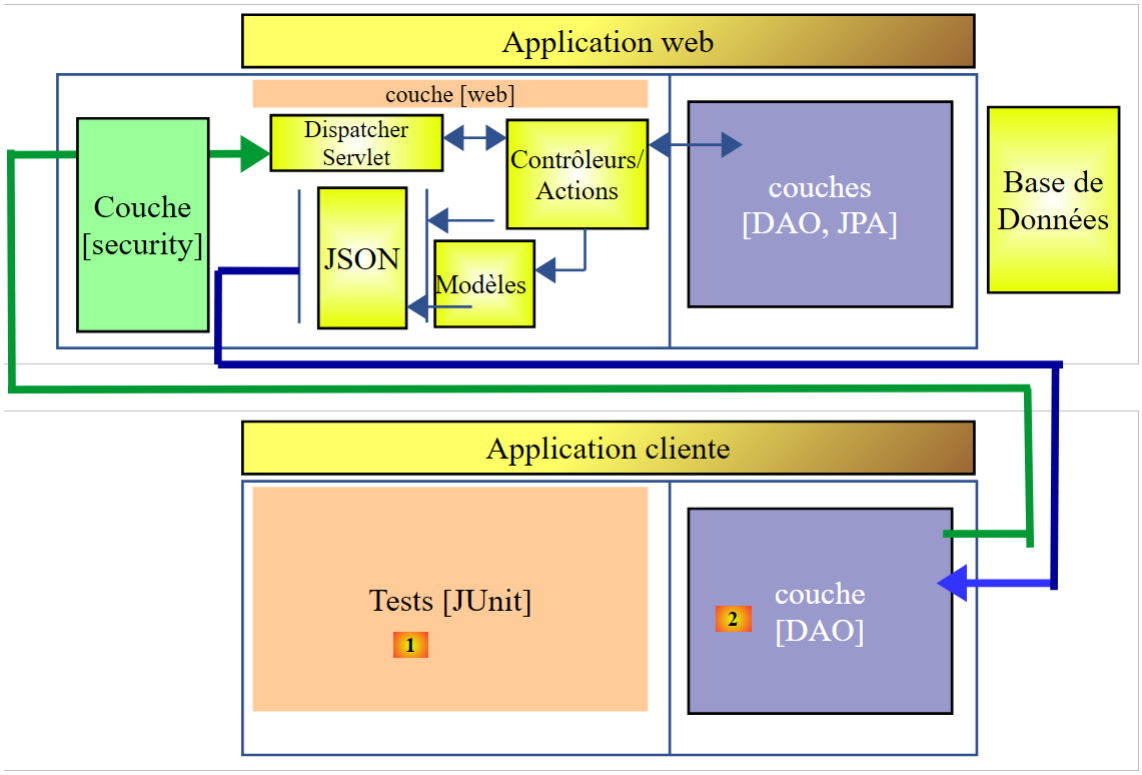
L'ajout des classes nécessaires à Spring Security a pu se faire sans modifications du projet web / json originel. Ce cas très favorable découle du fait que les trois tables ajoutées dans la base de données sont indépendantes des tables existantes. On aurait même pu les mettre dans une base de données séparée. Dans d'autres cas, les tables ajoutées peuvent avoir des relations avec les tables existantes. Il faut alors modifier les entités JPA ce qui en général impacte toutes les couches du projet.

16.5 Un client programmé pour le service web / jSON sécurisé

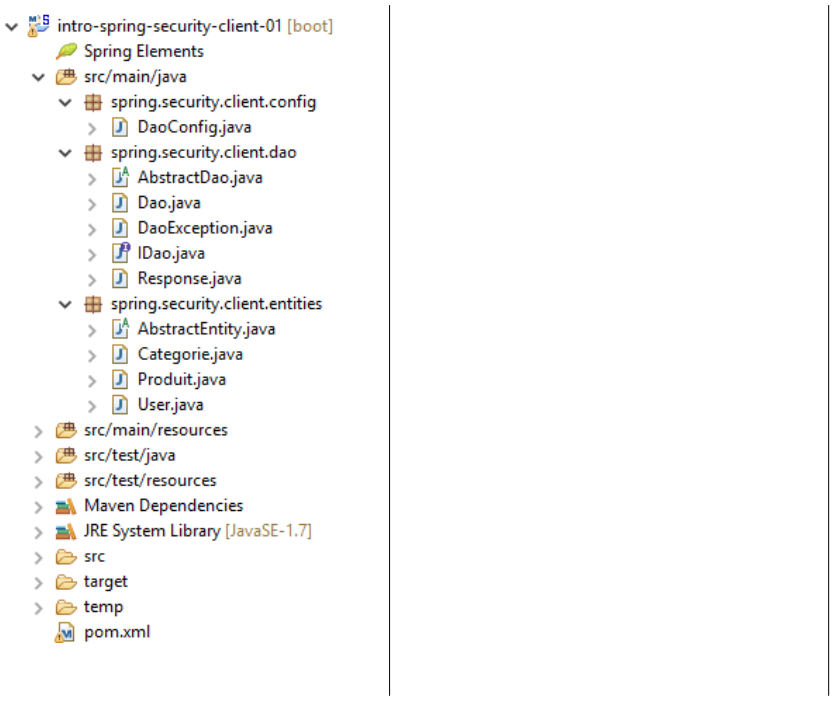
Nous avons déjà écrit un client pour le service web / jSON non sécurisé :



Nous allons maintenant créer un client programmé pour le service web sécurisé :



Nous dupliquons le projet déjà écrit [intro-webjson-client] dans un nouveau projet [intro-spring-security-client-01] :



16.5.1.1 La classe [AbstractDao]

La classe [AbstractDao] assure la communication HTTP avec le serveur web / jSON sécurisé. Comme nous venons de le voir, dans cette communication HTTP, le client doit désormais envoyer un entête d'authentification, par exemple :

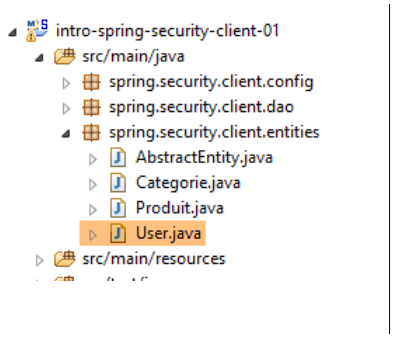
```
Authorization:Basic YWRtaW46YWRtaW4=
```

Cela se fait de la façon suivante :

```
1. package spring.security.client.dao;
2.
3. import java.net.URI;
4. ...
5.
6. public abstract class AbstractDao {
7.
8.   // data
9.   @Autowired
10.    protected RestTemplate restTemplate;
11.    @Autowired
12.    protected String urlServiceWebJson;
13.
14.    // requête générique
15.    protected String getResponse(User user, String url, String jsonPost) {
16.
17.  // url : URL à contacter
```

- ligne 15 : la méthode générique [getResponse] en charge de la communication HTTP avec le service web sécurisé, admet désormais comme premier paramètre l'utilisateur qui demande une URL. La classe [User] est la suivante :

Cette classe est la suivante :




```
1. package spring.security.client.entities;
2.
3. public class User {
4.
5.   // propriétés
6.   private String login;
7.   private String password;
8.
9.   // constructeur
10.    public User() {
11.    }
12.
13.    public User(String login, String password) {
14.        this.login = login;
15.        this.password = password;
16.    }
17.
18.    // getters et setters
19.  ...
20.  }
```

La méthode [getResponse] devient alors la suivante :

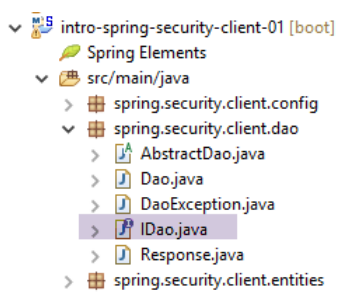
```
1. package spring.security.client.dao;
2.
3. import java.net.URI;
4. import java.net.URISyntaxException;
5. import java.util.Base64;
6.
7. import org.springframework.beans.factory.annotation.Autowired;
8. import org.springframework.core.ParameterizedTypeReference;
9. import org.springframework.http.MediaType;
10.  import org.springframework.http.RequestEntity;
11.  import org.springframework.http.RequestEntity.BodyBuilder;
12.  import org.springframework.http.RequestEntity.HeadersBuilder;
13.  import org.springframework.web.client.RestTemplate;
14.
15.  import spring.security.client.entities.User;
16.
17.  public abstract class AbstractDao {
18.
19.    // data
20.    @Autowired
21.    protected RestTemplate restTemplate;
22.    @Autowired
23.    protected String urlServiceWebJson;
24.
25.    private String getBase64(User user) {
26.        // on encode en base 64 l'utilisateur et son mot de passe - nécessite java 8
27.        String chaîne = String.format("%s:%s", user.getLogin(), user.getPassword());
28.        return String.format("Basic %s", new String(Base64.getEncoder().encode(chaîne.getBytes())));
29.    }
30.
31.    // requête générique
32.    protected String getResponse(User user, String url, String jsonPost) {
33.
34.        // url : URL à contacter
35.        // jsonPost : la valeur jSON à poster
36.        try {
37.            // exécution requête
38.            RequestEntity<?> request;
39.            if (jsonPost == null) {
40.                HeadersBuilder<?> headersBuilder = RequestEntity.get(new URI(String.format("%s%s", urlServiceWebJson, url)))
41.                        .accept(MediaType.APPLICATION_JSON);
42.                if (user != null) {
43.                    headersBuilder = headersBuilder.header("Authorization", getBase64(user));
44.                }
45.                request = headersBuilder.build();
```

```
46.           } else {
47.               BodyBuilder bodyBuilder = RequestEntity.post(new URI(String.format("%s%s", urlServiceWebJson, url)))
48.                       .header("Content-Type", "application/json").accept(MediaType.APPLICATION_JSON);
49.               if (user != null) {
50.                   bodyBuilder = bodyBuilder.header("Authorization", getBase64(user));
51.               }
52.               request = bodyBuilder.body(jsonPost);
53.           }
54.
55.           // on exécute la requête
56.           return restTemplate.exchange(request, new ParameterizedTypeReference<String>() {
57.           }).getBody();
58.       } catch (URISyntaxException e1) {
59.           throw new DaoException(20, e1);
60.       } catch (RuntimeException e2) {
61.           throw new DaoException(21, e2);
62.       }
63.   }
64.
65. }
66.
```

- lignes 42-44, 49-51 : si l'utilisateur [user] n'est pas **null**, alors on ajoute l'entête d'authentification. L'encodage *Base64* de l'utilisateur et de son mot de passe est assuré par la méthode [getBase64] des lignes 25-29. On fera attention au fait que cette méthode utilise une classe [Base64] appartenant au JDK 1.8.
- en-dehors des lignes précédentes, le code reste inchangé ;

16.5.1.2 L'interface [IDao]

Toutes les méthodes de l'interface [IDao] reçoivent un paramètre supplémentaire [User user] :




```
1. package spring.security.client.dao;
2.
3. import java.util.List;
4.
5. import spring.security.client.entities.Categorie;
6. import spring.security.client.entities.Produit;
7. import spring.security.client.entities.User;
8.
9. public interface IDaoClient {
10.
11.    // authentification
12.    public void authenticate(User user);
13.
14.    // insertion d'une liste de produits
15.    public List<Produit> addProduits(User user, List<Produit> produits);
16.
17.    // suppression de tous les produits
18.    public void deleteAllProduits(User user);
19.
20.    // mise à jour d'une liste de produits
21.    public List<Produit> updateProduits(User user, List<Produit> produits);
22.
23.    // obtention de tous les produits
24.    public List<Produit> getAllProduits(User user);
25.
26.    // insertion d'une liste de categories
27.    public List<Categorie> addCategories(User user, List<Categorie> categories);
28.
29.    // suppression de tous les categories
```

```
30.    public void deleteAllCategories(User user);
31.
32.    // mise à jour d'une liste de categories
33.    public List<Categorie> updateCategories(User user, List<Categorie> categories);
34.
35.    // obtention de tous les categories
36.    public List<Categorie> getAllCategories(User user);
37.
38.    // un produit particulier
39.    public Produit getProduitByIdWithCategorie(User user, Long idProduit);
40.
41.    public Produit getProduitByIdWithoutCategorie(User user, Long idProduit);
42.
43.    public Produit getProduitByNameWithCategorie(User user, String nom);
44.
45.    public Produit getProduitByNameWithoutCategorie(User user, String nom);
46.
47.    // une catégorie particulière
48.    public Categorie getCategorieByIdWithProduits(User user, Long idCategorie);
49.
50.    public Categorie getCategorieByIdWithoutProduits(User user, Long idCategorie);
51.
52.    public Categorie getCategorieByNameWithProduits(User user, String nom);
53.
54.    public Categorie getCategorieByNameWithoutProduits(User user, String nom);
55.
56. }
```

- ligne 12 : nous avons ajouté la méthode [authenticate(User user)] pour authentifier un utilisateur. Elle lance une exception si l'utilsateur n'a pas le droit d'accéder à l'URL [/authenticate] du service web ;

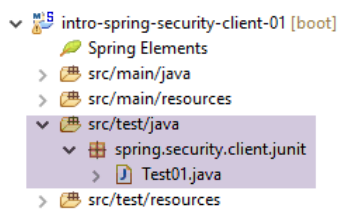
16.5.1.3 La classe [Dao]

Toutes les méthodes de la classe [Dao] reçoivent un paramètre supplémentaire [User user] qu'elles passent à la méthode générique [getResponse] de la classe [AbstractDao]. Voici deux exemples :

```
1. // authentification
2.   @Override
3.   public void authenticate(User user) {
4.     getResponse(user, "/authenticate", null);
5.   }
6.
7.   @Override
8.   public List<Produit> addProduits(User user, List<Produit> produits) {
9.     // ----------- ajouter des produits (sans leur catégorie)
10.       try {
11.           // mappeurs jSON
12.           ObjectMapper mapperPost = context.getBean(ObjectMapper.class);
13.           mapperPost.setFilters(jsonFilterProduitWithoutCategorie);
14.           ObjectMapper mapperResponse = mapperPost;
15.           // requête
16.           Response<List<Produit>> response = mapperResponse.readValue(
17.                   getResponse(user, "/addProduits", mapperPost.writeValueAsString(produits)),
18.                   new TypeReference<Response<List<Produit>>>() {
19.                   });
20.           // erreur ?
21.           if (response.getStatus() != 0) {
22.               // on lance 1 exception
23.               throw new DaoException(response.getStatus(), response.getMessages());
24.           } else {
25.               // on rend le coeur de la réponse du serveur
26.               return response.getBody();
27.           }
28.       } catch (DaoException e1) {
29.           throw e1;
30.       } catch (IOException | RuntimeException e2) {
31.           throw new DaoException(100, e2);
32.       }
33.   }
34.
```

16.5.1.4 Tests unitaires de la classe [Dao]

La classe [Test01] de tests unitaires de la classe [Dao] est modifiée de la façon suivante :




```
1. package client.tests.junit;
2.
3. ...
4.
5. @SpringApplicationConfiguration(classes = DaoConfig.class)
6. @RunWith(SpringJUnit4ClassRunner.class)
7. public class Test01 {
8.
9.   // contexte Spring
10.    @Autowired
11.    private ApplicationContext context;
12.    // couche [DAO]
13.    @Autowired
14.    private IDaoClient dao;
15.
16.    // utilisateurs
17.    static private User admin;
18.    static private User user;
19.    static private User unknown;
20.
21.    @BeforeClass
22.    public static void init() {
23.        admin = new User("admin", "admin");
24.        user = new User("user", "user");
25.        unknown = new User("x", "y");
26.    }
27.
28.    @Before
29.    public void cleanAndFill() {
30.        // on nettoie la base avant chaque test
31.        log("Vidage de la base de données", 1);
32.        // on vide la table [CATEGORIES] - par cascade la table [PRODUITS] va être vidée
33.        dao.deleteAllCategories(admin);
34.        // ---------------------------------------------------------------------------------
35.        log("Remplissage de la base", 1);
36.        // on remplit les tables
37.        List<Categorie> categories = new ArrayList<Categorie>();
38.        for (int i = 0; i < 2; i++) {
39.            Categorie categorie = new Categorie(String.format("categorie%d", i));
40.            for (int j = 0; j < 5; j++) {
41.                categorie.addProduit(new Produit(String.format("produit%d%d", i, j), 100 * (1 + (double) (i * 10 + j) / 100),
42.                        String.format("desc%d%d", i, j)));
43.            }
44.            categories.add(categorie);
45.        }
46.        // ajout de la catégorie - par cascade les produits vont eux aussi être insérés
47.        dao.addCategories(admin, categories);
48.    }
49.
50.    @Test
51.    public void showDataBase() throws BeansException, JsonProcessingException {
52.        // liste des catégories
53.        log("Liste des catégories", 2);
54.        List<Categorie> categories = dao.getAllCategories(admin);
55.        affiche(categories, context.getBean("jsonMapperCategorieWithoutProduits", ObjectMapper.class));
56.        // liste des produits
57.        log("Liste des produits", 2);
58.        List<Produit> produits = dao.getAllProduits(admin);
59.        affiche(produits, context.getBean("jsonMapperProduitWithoutCategorie", ObjectMapper.class));
60.        // quelques vérifications
61.        Assert.assertEquals(2, categories.size());
62.        Assert.assertEquals(10, produits.size());
63.        Categorie categorie = findCategorieByName("categorie0", categories);
64.        Assert.assertNotNull(categorie);
65.        Produit produit = findProduitByName("produit03", produits);
66.        Assert.assertNotNull(produit);
67.        Long idCategorie = produit.getIdCategorie();
68.        Assert.assertEquals(categorie.getId(), idCategorie);
69.    }
70. ...
71.    @Test()
72.    public void checkUserUser() {
```

```
73.         ServiceException se = null;
74.         try {
75.             dao.authenticate(user);
76.         } catch (ServiceException e) {
77.             se = e;
78.         }
79.         Assert.assertNotNull(se);
80.         Assert.assertEquals("403 Forbidden", se.getMessages().get(0));
81.     }
82.
83.     @Test()
84.     public void checkUserUnknown() {
85.         ServiceException se = null;
86.         try {
87.             dao.authenticate(unknown);
88.         } catch (ServiceException e) {
89.             se = e;
90.         }
91.         Assert.assertNotNull(se);
92.         Assert.assertEquals("401 Unauthorized", se.getMessages().get(0));
93.     }
94.
95.     @Test()
96.     public void checkUserAdmin() {
97.         ServiceException se = null;
98.         try {
99.             dao.authenticate(admin);
100.        } catch (ServiceException e) {
101.            se = e;
102.        }
103.        Assert.assertNull(se);
104.    }
105. ...
106. }
```

- lors de l'initialisation de la classe de test, lignes 21-26, trois utilisateurs sont créés :
  - l'utilisateur [admin] a accès aux URL du service web, test lignes 96-104 ;
  - l'utilisateur [user] existe mais n'est pas autorisé à utiliser les URL du service web, test lignes 71-81 ;
  - l'utilisateur [unknown] n'existe pas, test lignes 83-93 ;
- les méthodes de tests sont celles déjà vues pour le service web non sécurisé, si ce n'est que les méthodes de l'interface [IDaoClient] sont appelées avec comme premier paramètre, l'utilisateur [admin] qui a le droit d'utiliser les URL ;

Le test passe mais on peut constater qu'il est plus lent qu'avec le service web non sécurisé. La sécurisation d'une application augmente sensiblement ses temps de réponse. On peut noter un facteur important dans les performances du service web sécurisé : dans la classe [AppConfig] qui le configure, nous avons écrit :

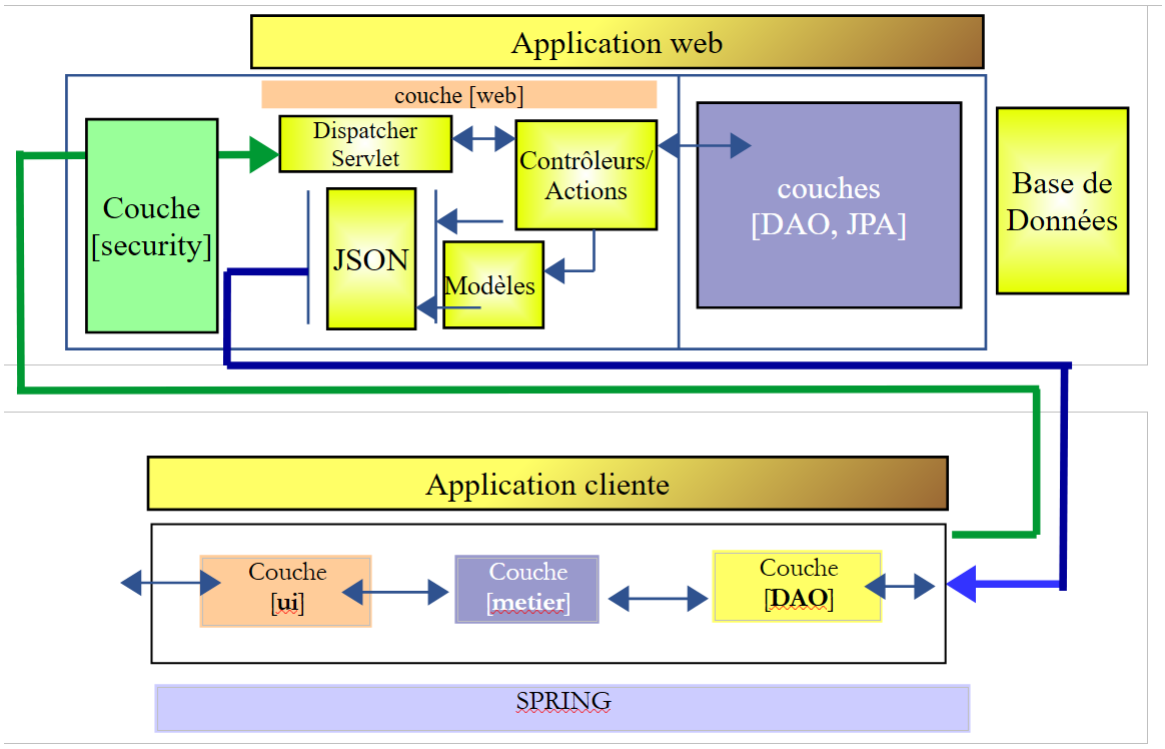
```
1.  @Override
2.  protected void configure(HttpSecurity http) throws Exception {
3.      // CSRF
4.      http.csrf().disable();
5.      // application sécurisée ?
6.      if (activateSecurity) {
7.          // le mot de passe est transmis par le header Authorization: Basic xxxx
8.          http.httpBasic();
9.          // la méthode HTTP OPTIONS doit être autorisée pour tous
10.         http.authorizeRequests() //
11.                 .antMatchers(HttpMethod.OPTIONS, "/", "/**").permitAll();
12.         // seul le rôle ADMIN peut utiliser l'application
13.         http.authorizeRequests() //
14.                 .antMatchers("/", "/**") // toutes les URL
15.                 .hasRole("ADMIN");
16.         // pas de session
17.         http.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);
18.     }
19. }
```

La ligne 17 a un coût. Elle force l'utilisateur à s'authentifier à chaque accès. Si on la met en commentaires, la durée du test JUnit précédent passe de 10,57 secondes à 4,21 secondes, ceci parce que l'utilisateur [admin] ne s'authentifie que pour le premier test et pas pour les suivants (même si l'entête HTTP d'authentification est envoyé par le client, le serveur lui ne revérifie pas le mot de passe de l'utilisateur). Avec un service web non sécurisé, la durée du test JUnit tombe à 2,33 secondes.

17 [TD] : sécurisation du serveur web / jSON des élections

Mots clés : architecture multicouche, Spring, injection de dépendances, service web / jSON sécurisé, client / serveur

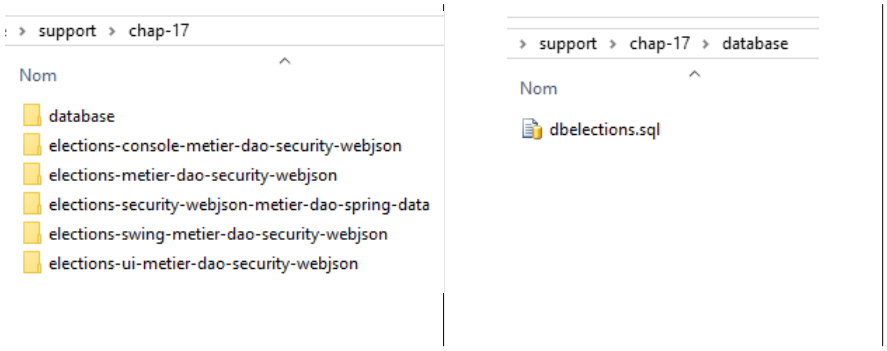
Nous appliquons maintenant ce que nous avons appris dans le chapitre précédent au TD des élections. L'architecture sera la suivante :



La progession se fera de la façon suivante :

- écriture du serveur ;
- écriture du client sans couche [ui] mais avec un test JUnit de la couche [métier] ;
- écriture du client avec la couche [ui] ;

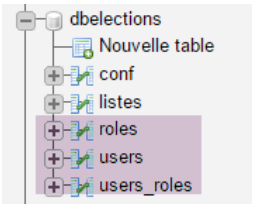
17.1 Support

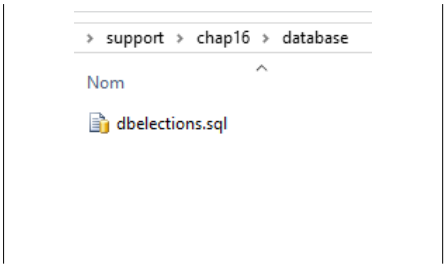


Les projets de ce chapitre seront trouvés dans le dossier [support / chap-17]. Le script SQL sert à générer la base de données nécessaire aux tests.

17.2 La base de données

La base de données du serveur sécurisé doit maintenant comporter les tables [USERS], [ROLES] et [USERS_ROLES] :

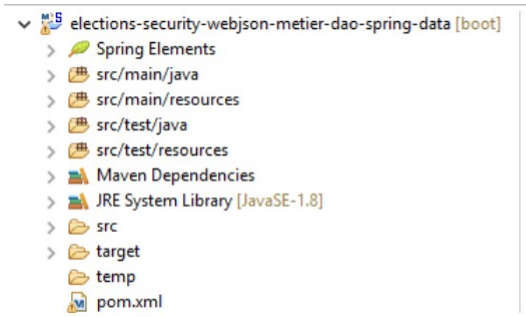


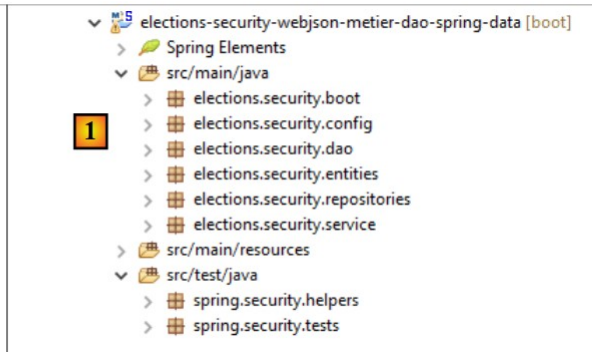


Le script SQL de la génération de la base est disponible dans le support du document.

17.3 Le serveur sécurisé

Pour mettre en place le serveur sécurisé des élections, nous nous contentons de faire un copy / paste du projet exemple [intro-spring-security-server-01] :





Travail à faire : renommez les packages comme montré en [1].

Ceci fait, nous modifions le fichier [pom.xml] de la façon suivante :

```
1. <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2.   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
3.   <modelVersion>4.0.0</modelVersion>
4.   <groupId>istia.st.elections</groupId>
5.   <artifactId>elections-security-webjson-metier-dao-spring-data</artifactId>
6.   <version>0.0.1-SNAPSHOT</version>
7.
8.   <name>elections-security-webjson-metier-dao-spring-data</name>
9.   <description>elections spring security</description>
10.
11.    <properties>
12.        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
13.        <java.version>1.8</java.version>
14.    </properties>
15.
16.    <parent>
17.        <groupId>org.springframework.boot</groupId>
18.        <artifactId>spring-boot-starter-parent</artifactId>
19.        <version>1.2.7.RELEASE</version>
20.    </parent>
21.
22.    <dependencies>
23.        <dependency>
24.            <groupId>istia.st.elections</groupId>
25.            <artifactId>elections-webjson-metier-dao-spring-data</artifactId>
26.            <version>0.0.1-SNAPSHOT</version>
27.        </dependency>
28.        <!-- Spring security -->
29.        ...
30.    </dependencies>
31.    <!-- plugins -->
32.    <build>
33.        ...
34.    </build>
```

```
35.
36.  </project>
```

- aux lignes 23-27, remplacez la dépendance sur le projet [intro-server-webjson-01] par celle sur le projet [elections-webjson-metier-dao-spring-data] étudié au paragraphe 12, page 207 ;
- aux lignes 4-7, mettre les caractéristiques du nouveau projet ;

Il nous reste à modifier les classes de configuration du projet :

[DaoConfig]

```
1. package elections.security.config;
2.
3. import org.springframework.context.annotation.Bean;
4. import org.springframework.context.annotation.ComponentScan;
5. import org.springframework.context.annotation.Import;
6. import org.springframework.data.jpa.repository.config.EnableJpaRepositories;
7.
8. @EnableJpaRepositories(basePackages = { "elections.security.repositories" })
9. @ComponentScan(basePackages = { "elections.security.dao" })
10.  @Import({ elections.dao.config.DaoConfig.class })
11.  public class DaoConfig {
12.
13.    // constantes
14.    final static private String[] ENTITIES_PACKAGES = { "elections.dao.entities", "elections.security.entities" };
15.
16.
17.    @Bean
18.    public String[] packagesToScan() {
19.        return ENTITIES_PACKAGES;
20.    }
21.
22.  }
```

Les modifications sont aux lignes suivantes :

- lignes 8, 9, 14 : mettre les bons noms de packages ;
- ligne 10 : la classe importée est maintenant [elections.dao.config.DaoConfig] du projet [elections-webjson-metier-dao-spring-data] ;

Note : comme il a été dit, cette configuration ne marche que si la classe [elections.dao.config.DaoConfig] a l'annotation [@Configuration]. Vérifiez ce point.

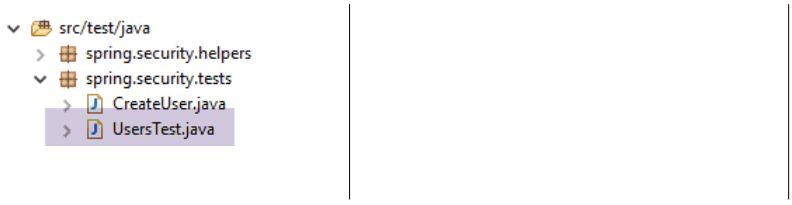
[SecurityConfig]

```
1. package elections.security.config;
2.
3. ...
4.
5. @EnableWebSecurity
6. @ComponentScan(basePackages = { "elections.security.service" })
7. @Import({ WebConfig.class, DaoConfig.class })
8. public class SecurityConfig extends WebSecurityConfigurerAdapter {
9.
10.    ...
11.  }
```

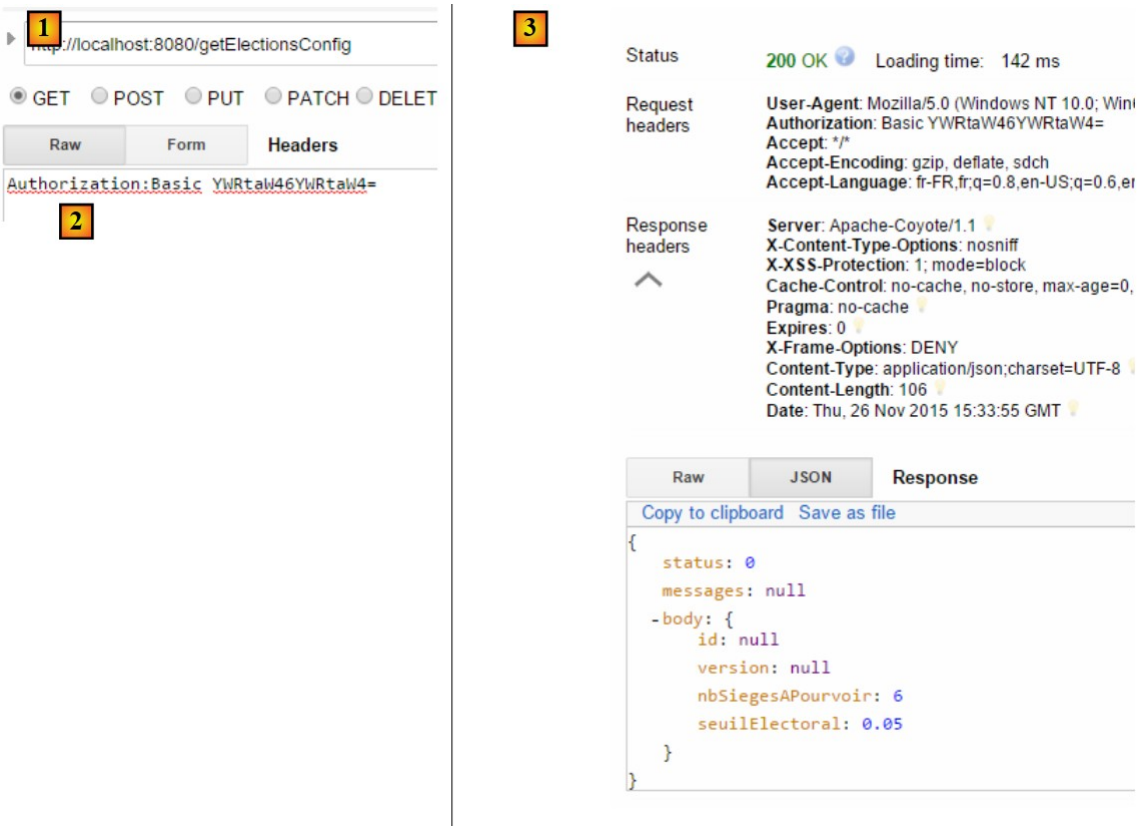
Les modifications sont aux lignes suivantes :

- ligne 6 : changez le nom du package ;

C'est tout. Faites un premier test en exécutant le test JUnit [] :

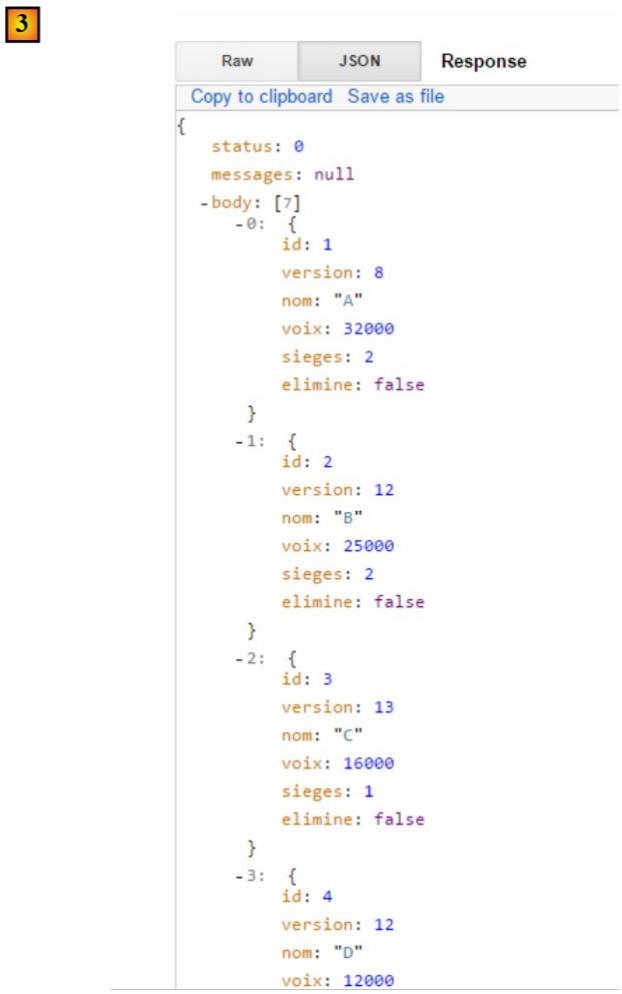


Exécutez la classe de boot [Boot] puis avec l'extension Chrome [Advanced Rest Client] faites le test suivant :

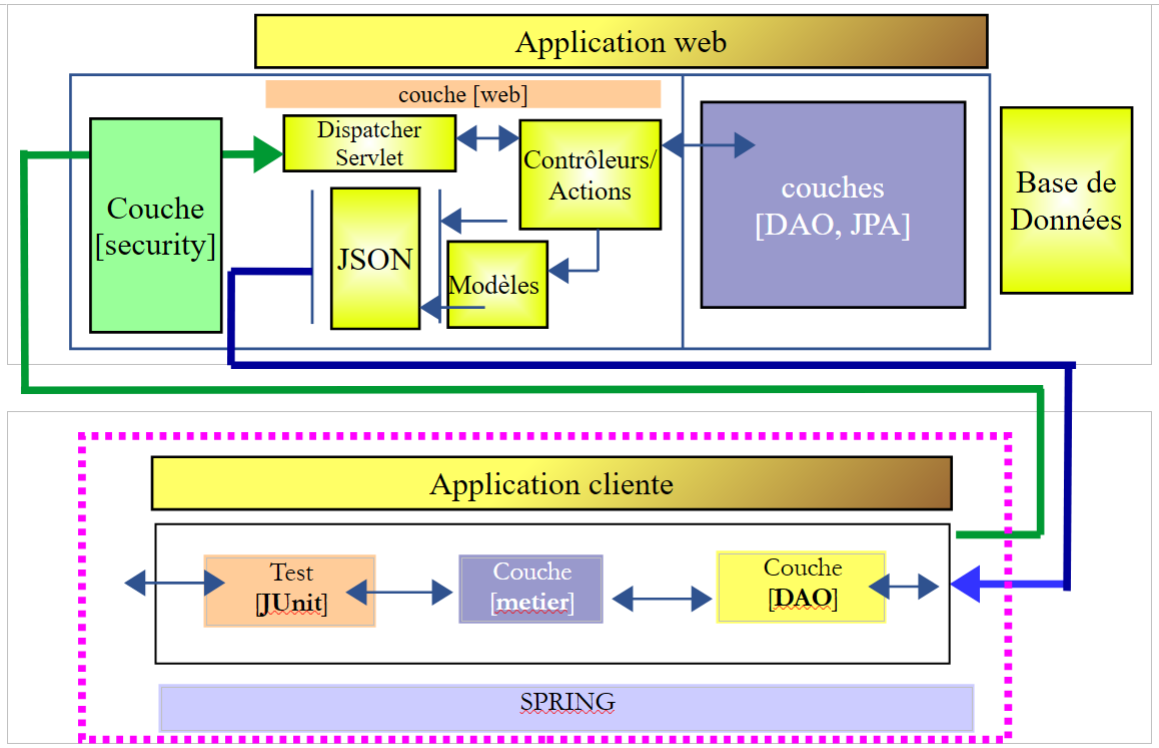


En [1], la demande. En [3], la réponse. En [2], l'entête HTTP est l'entête d'authentification de l'utilisateur [admin, admin] : **Authorization:Basic YWRtaW46YWRtaW4=**

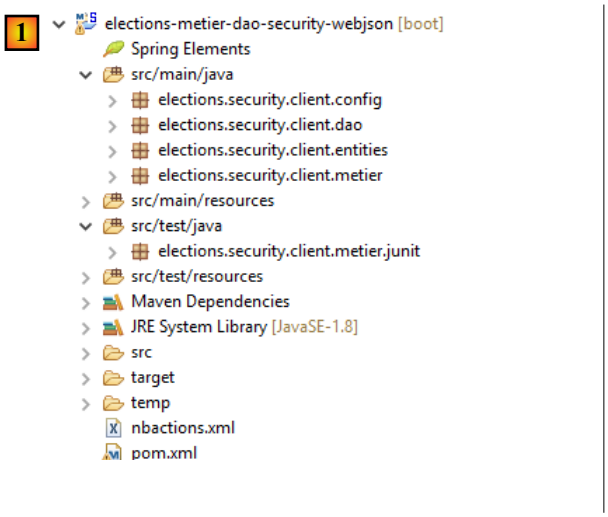
Demandez maintenant, les listes en compétition :



17.4 Le client du serveur sécurisé sans la couche [ui]



Faites un copy / paste du projet [elections-ui-metier-dao-webjson] dans le projet [elections-metier-dao-security-webjson].



Travail à faire : en [1], renommez les packages si nécessaire et supprimez ceux de la couche [ui] et des classes de démarrage [boot].

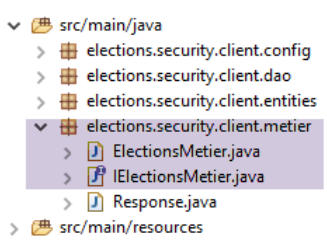
Nous allons procéder maintenant comme il a été fait au paragraphe 16.5, page 318.

17.4.1 Configuration Maven

Seule la partie identification du projet change :

```
1. <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2.   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
3.   <modelVersion>4.0.0</modelVersion>
4.   <groupId>istia.st.elections</groupId>
5.   <artifactId>elections-metier-dao-security-webjson</artifactId>
6.   <version>0.0.1-SNAPSHOT</version>
7.   <description>Client jUnit du serveur web / jSON</description>
8.     <name>elections-metier-dao-security-webjson</name>
9. ...
```

17.4.2 Refactorisation de la couche [métier]



L'interface [IElectionsMetier] évolue de la façon suivante :

```
1. package elections.security.client.metier;
2.
3. import elections.security.client.entities.ListeElectorale;
4. import elections.security.client.entities.User;
5.
6. public interface IElectionsMetier {
7.
8.   // authentification
9.   public void authenticate(User user);
10.
11.    // obtenir les listes en compétition
12.    public ListeElectorale[] getListesElectorales(User user);
13.
14.    // le nombre de sièges à pourvoir
15.    public int getNbSiegesAPourvoir(User user);
16.
17.    // le seuil électoral
18.    public double getSeuilElectoral(User user);
19.
20.    // l'enregistrement des résultats
21.    public void recordResultats(User user, ListeElectorale[] listesElectorales);
22.
23.    // le calcul des sièges
24.    public ListeElectorale[] calculerSieges(User user, ListeElectorale[] listesElectorales);
25.
26. }
```

Toutes les méthodes ont pour premier paramètre, l'utilisateur qui veut utiliser les ressources du serveur sécurisé.

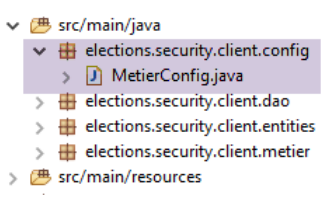
L'implémentation [ElectionsMetier] évolue de la façon suivante :

```
1. package elections.security.client.metier;
2.
3. import java.io.IOException;
4.
5. import org.springframework.beans.factory.annotation.Autowired;
6. import org.springframework.context.ApplicationContext;
7. import org.springframework.stereotype.Component;
8.
9. import com.fasterxml.jackson.core.type.TypeReference;
10.  import com.fasterxml.jackson.databind.ObjectMapper;
11.
12.  import elections.security.client.dao.IClientDao;
13.  import elections.security.client.entities.ElectionsConfig;
14.  import elections.security.client.entities.ElectionsException;
15.  import elections.security.client.entities.ListeElectorale;
16.  import elections.security.client.entities.User;
17.
18.  @Component
19.  public class ElectionsMetier implements IElectionsMetier {
20.
21.    @Autowired
22.    private IClientDao dao;
23.    @Autowired
24.    private ApplicationContext context;
25.
26.    // configuration de l'élection
27.    private ElectionsConfig electionsConfig;
28.
29.    private ElectionsConfig getElectionsConfig(User user) {
30.      if(electionsConfig!=null){
31.        return electionsConfig;
32.      }
```

```
33.     // mappeurs jSON
34.     ObjectMapper mapperResponse = context.getBean(ObjectMapper.class);
35.     try {
36.       // requête
37.       Response<ElectionsConfig> response = mapperResponse.readValue(dao.getResponse(user, "/getElectionsConfig", null),
38.               new TypeReference<Response<ElectionsConfig>>() {
39.       });
40.       // erreur ?
41.       if (response.getStatus() != 0) {
42.         // on lance 1 exception
43.         throw new ElectionsException(response.getStatus(), response.getMessages());
44.       } else {
45.         electionsConfig = response.getBody();
46.         return electionsConfig;
47.       }
48.     } catch (ElectionsException e1) {
49.       throw e1;
50.     } catch (IOException | RuntimeException e2) {
51.       throw new ElectionsException(100, e2);
52.     }
53.   }
54.
55.   @Override
56.   public ListeElectorale[] getListesElectorales(User user) {
57.    ...
58.   }
59.
60.   @Override
61.   public int getNbSiegesAPourvoir(User user) {
62.     return getElectionsConfig(user).getNbSiegesAPourvoir();
63.   }
64.
65.   @Override
66.   public double getSeuilElectoral(User user) {
67.     return getElectionsConfig(user).getSeuilElectoral();
68.   }
69.
70.   @Override
71.   public void recordResultats(User user, ListeElectorale[] listesElectorales) {
72.     ...
73.   }
74.
75.   @Override
76.   public ListeElectorale[] calculerSieges(User user, ListeElectorale[] listesElectorales) {
77.     ...
78.   }
79.
80.   @Override
81.   public void authenticate(User user) {
82.     dao.getResponse(user, "/authenticate", null);
83.   }
84. }
```

Travail à faire : complétez le code ci-dessus.

17.4.3 Configuration Spring

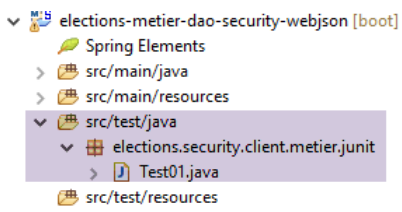


La classe [MetierConfig] évolue comme suit :

```
1. package elections.security.client.config;
2.
3. ...
4.
5. @ComponentScan({ "elections.security.client.dao", "elections.security.client.metier" })
6. public class MetierConfig {
```

Ligne 5, les packages à explorer doivent être mis à jour.

17.4.4 Le test JUnit de la couche [métier]



La classe de test [Test01] évolue de la façon suivante :

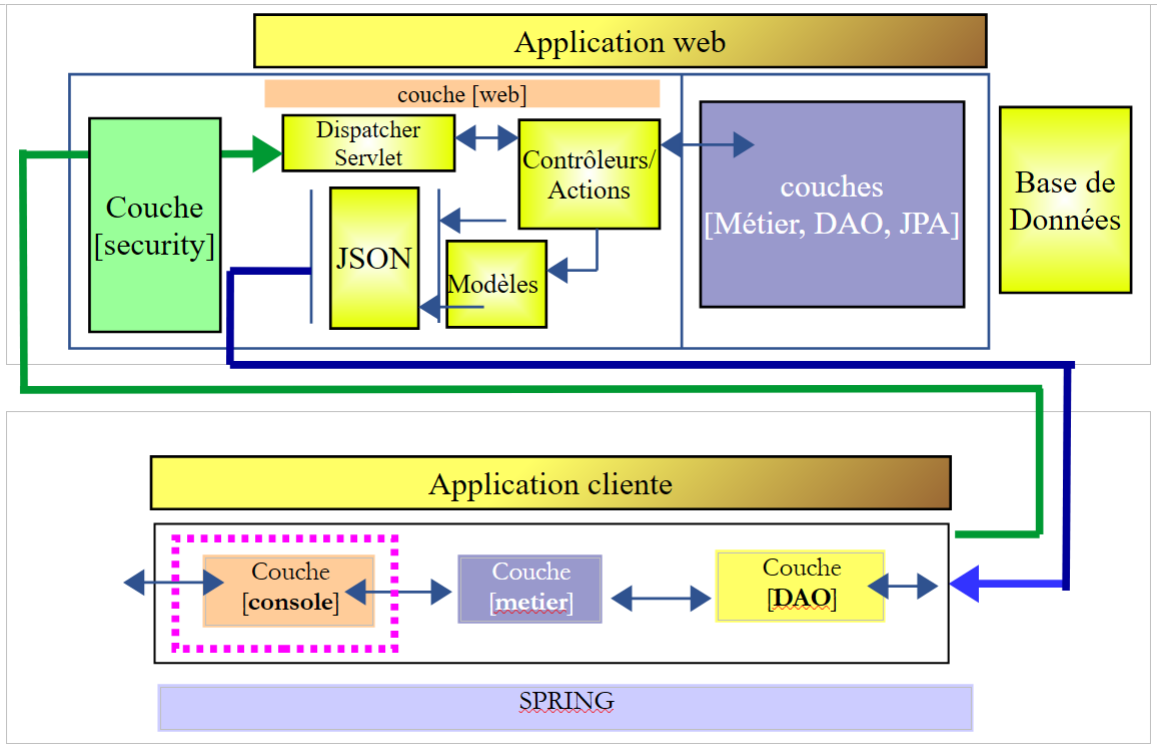
```
1. package elections.security.client.metier.junit;
2.
3. import org.junit.Assert;
4. import org.junit.BeforeClass;
5. import org.junit.Test;
6. import org.junit.runner.RunWith;
7. import org.springframework.beans.factory.annotation.Autowired;
8. import org.springframework.boot.test.SpringApplicationConfiguration;
9. import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
10.
11.  import com.fasterxml.jackson.core.JsonProcessingException;
12.  import com.fasterxml.jackson.databind.ObjectMapper;
13.
14.  import elections.security.client.config.MetierConfig;
15.  import elections.security.client.entities.ElectionsException;
16.  import elections.security.client.entities.ListeElectorale;
17.  import elections.security.client.entities.User;
18.  import elections.security.client.metier.IElectionsMetier;
19.
20.  @SpringApplicationConfiguration(classes = MetierConfig.class)
21.  @RunWith(SpringJUnit4ClassRunner.class)
22.  public class Test01 {
23.
24.    // couche [electionsMetier]
25.    @Autowired
26.    private IElectionsMetier electionsMetier;
27.
28.    // mappeur jSON
29.    private final ObjectMapper mapper = new ObjectMapper();
30.
31.    // utilisateurs
32.    static private User admin;
33.    static private User user;
34.    static private User unknown;
35.
36.    @BeforeClass
37.    public static void initTest() {
38.        admin = new User("admin", "admin");
39.        user = new User("user", "user");
40.        unknown = new User("x", "y");
41.    }
42.
43.    @Test()
44.    public void checkUserUser() {
45.        ElectionsException se = null;
46.        try {
47.            electionsMetier.authenticate(user);
48.        } catch (ElectionsException e) {
49.            se = e;
50.        }
51.        Assert.assertNotNull(se);
52.        Assert.assertEquals("403 Forbidden", se.getErreurs().get(0));
53.    }
54.
55.    @Test()
56.    public void checkUserUnknown() {
57.        ElectionsException se = null;
58.        try {
59.            electionsMetier.authenticate(unknown);
60.        } catch (ElectionsException e) {
61.            se = e;
62.        }
63.        Assert.assertNotNull(se);
64.        Assert.assertEquals("401 Unauthorized", se.getErreurs().get(0));
65.    }
66.
67.    @Test()
```

```
68.    public void checkUserAdmin() {
69.        ElectionsException se = null;
70.        try {
71.            electionsMetier.authenticate(admin);
72.        } catch (ElectionsException e) {
73.            se = e;
74.        }
75.        Assert.assertNull(se);
76.    }
77.
78.    /**
79.     * vérification 1 : méthode de calcul des sièges on fixe en dur les listes
80.     */
81.    @Test
82.    public void calculSieges1() {
83.        // on crée le tableau des 7 listes candidates
84.        ListeElectorale[] listes = new ListeElectorale[7];
85.        listes[0] = new ListeElectorale("A", 32000, 0, false);
86.        listes[1] = new ListeElectorale("B", 25000, 0, false);
87.        listes[2] = new ListeElectorale("C", 16000, 0, false);
88.        listes[3] = new ListeElectorale("D", 12000, 0, false);
89.        listes[4] = new ListeElectorale("E", 8000, 0, false);
90.        listes[5] = new ListeElectorale("F", 4500, 0, false);
91.        listes[6] = new ListeElectorale("G", 2500, 0, false);
92.        // on calcule les sièges de chacune des listes
93.        listes = electionsMetier.calculerSieges(admin,listes);
94.        // on vérifie les résultats
95.        Assert.assertEquals(2, listes[0].getSieges());
96.        Assert.assertFalse(listes[0].isElimine());
97.        Assert.assertEquals(2, listes[1].getSieges());
98.        Assert.assertFalse(listes[1].isElimine());
99.        Assert.assertEquals(1, listes[2].getSieges());
100.       Assert.assertFalse(listes[2].isElimine());
101.       Assert.assertEquals(1, listes[3].getSieges());
102.       Assert.assertFalse(listes[3].isElimine());
103.       Assert.assertEquals(0, listes[4].getSieges());
104.       Assert.assertFalse(listes[4].isElimine());
105.       Assert.assertEquals(0, listes[5].getSieges());
106.       Assert.assertTrue(listes[5].isElimine());
107.       Assert.assertEquals(0, listes[6].getSieges());
108.       Assert.assertTrue(listes[6].isElimine());
109.   }
110.
111.   /**
112.    * vérification 2 : méthode de calcul des sièges on demande les listes à la couche [metier] puis on fixe en dur les
113.    * voix
114.    */
115.   @Test
116.   public void calculSieges2() {
117.       // on crée le tableau des 7 listes candidates
118.       ListeElectorale[] listes = electionsMetier.getListesElectorales(admin);
119.       // on fixe en dur les voix
120.       listes[0].setVoix(32000);
121.       listes[1].setVoix(25000);
122.       listes[2].setVoix(16000);
123.       listes[3].setVoix(12000);
124.       listes[4].setVoix(8000);
125.       listes[5].setVoix(4500);
126.       listes[6].setVoix(2500);
127.       // on calcule les sièges obtenus par chacune des listes
128.       listes = electionsMetier.calculerSieges(admin,listes);
129.       // on vérifie les résultats
130.       Assert.assertEquals(2, listes[0].getSieges());
131.       Assert.assertFalse(listes[0].isElimine());
132.       Assert.assertEquals(2, listes[1].getSieges());
133.       Assert.assertFalse(listes[1].isElimine());
134.       Assert.assertEquals(1, listes[2].getSieges());
135.       Assert.assertFalse(listes[2].isElimine());
136.       Assert.assertEquals(1, listes[3].getSieges());
137.       Assert.assertFalse(listes[3].isElimine());
138.       Assert.assertEquals(0, listes[4].getSieges());
139.       Assert.assertFalse(listes[4].isElimine());
140.       Assert.assertEquals(0, listes[5].getSieges());
141.       Assert.assertTrue(listes[5].isElimine());
142.       Assert.assertEquals(0, listes[6].getSieges());
143.       Assert.assertTrue(listes[6].isElimine());
144.   }
145.
146.   /**
147.    * vérification 3 méthode de calcul des sièges on provoque une exception
148.    */
149.   @Test(expected = ElectionsException.class)
150.   public void calculSieges3() {
151.       // on crée un tableau de 24 listes candidates avec chacune 1 voix
152.       ListeElectorale[] listes = new ListeElectorale[25];
153.       // les 25 listes auront le même nombre de voix (4%)
154.       for (int i = 0; i < listes.length; i++) {
```

```
155.          listes[i] = new ListeElectorale("Liste" + (i + 1), 1, 0, false);
156.      }
157.      // calcul des sièges - normalement on doit avoir une ElectionsException
158.      // avec un seuil électoral de 5%
159.      electionsMetier.calculerSieges(admin,listes);
160.  }
161.
162.  /**
163.   * enregistrement des résultats de l'élection
164.   *
165.   * @throws JsonProcessingException
166.   */
167.  @Test
168.  public void ecritureResultatsElections() throws JsonProcessingException {
169.      // on crée le tableau des 7 listes candidates
170.      ListeElectorale[] listes = electionsMetier.getListesElectorales(admin);
171.      // on fixe en dur les voix
172.      listes[0].setVoix(32000);
173.      listes[1].setVoix(25000);
174.      listes[2].setVoix(16000);
175.      listes[3].setVoix(12000);
176.      listes[4].setVoix(8000);
177.      listes[5].setVoix(4500);
178.      listes[6].setVoix(2500);
179.      // on calcule les sièges obtenus par chacune des listes
180.      listes = electionsMetier.calculerSieges(admin,listes);
181.      // on affiche les résultats
182.      for (int i = 0; i < listes.length; i++) {
183.          System.out.println(mapper.writeValueAsString(listes[i]));
184.      }
185.      // on enregistre les résultats dans la base de données
186.      electionsMetier.recordResultats(admin,listes);
187.      // on vérifie les résultats
188.      listes = electionsMetier.getListesElectorales(admin);
189.      // on affiche les résultats
190.      for (int i = 0; i < listes.length; i++) {
191.          System.out.println(mapper.writeValueAsString(listes[i]));
192.      }
193.      Assert.assertEquals(2, listes[0].getSieges());
194.      Assert.assertFalse(listes[0].isElimine());
195.      Assert.assertEquals(2, listes[1].getSieges());
196.      Assert.assertFalse(listes[1].isElimine());
197.      Assert.assertEquals(1, listes[2].getSieges());
198.      Assert.assertFalse(listes[2].isElimine());
199.      Assert.assertEquals(1, listes[3].getSieges());
200.      Assert.assertFalse(listes[3].isElimine());
201.      Assert.assertEquals(0, listes[4].getSieges());
202.      Assert.assertFalse(listes[4].isElimine());
203.      Assert.assertEquals(0, listes[5].getSieges());
204.      Assert.assertTrue(listes[5].isElimine());
205.      Assert.assertEquals(0, listes[6].getSieges());
206.      Assert.assertTrue(listes[6].isElimine());
207.  }
208. }
```

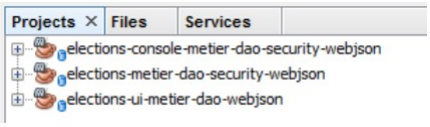
Travail à faire : comprenez ce test et vérifiez qu'il passe.

17.5 Le client du serveur sécurisé avec une couche [console]



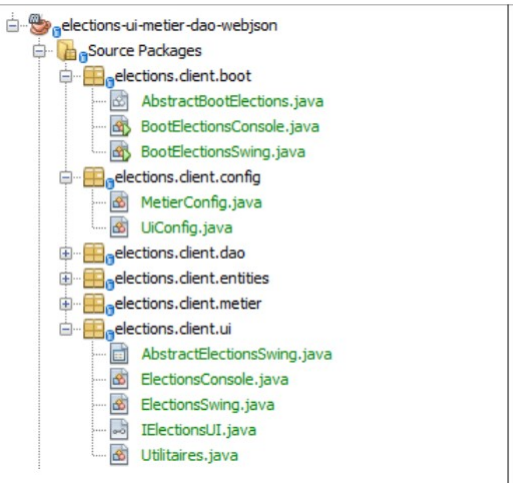
Avec **Netbeans**, nous ouvrons les projets Maven [elections-metier-dao-security-webjson] et [elections-ui-metier-dao-webjson] puis nous construisons un nouveau projet [elections-console-metier-dao-security-webjson] par un copy / paste du projet [elections-ui-metier-dao-webjson] :

1

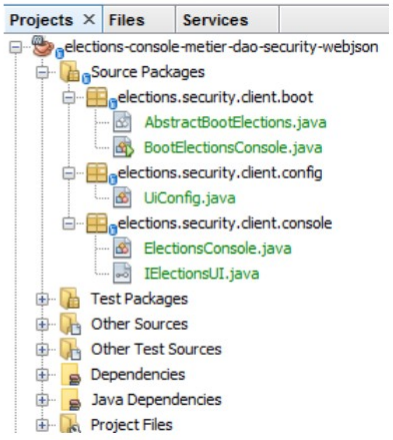


La plupart des classes du nouveau projet [3] proviennent du projet [elections-ui-metier-dao-webjson] [2] qui était le client du serveur web / jSON non sécurisé :

2



3



Procédez ainsi pour construire le nouveau projet Maven :

- faites un copy / paste du projet [elections-ui-metier-dao-webjson] dans le projet [elections-ui-metier-dao-security-webjson] ;

- dans le nouveau projet :
  - supprimez les packages [elections.client.dao, elections.client.metier, elections.client.entities] ;
  - dans le package [elections.client.ui], gardez seulement la classe console [ElectionsConsole] et son interface [IElectionsUI] ;
  - renommez les packages comme indiqué en [3] ;
  - réglez les problèmes d'import dans les diverses classes par [clic droit / Fix Imports] ;

A ce stade, le nouveau projet présente de nombreuses erreurs.

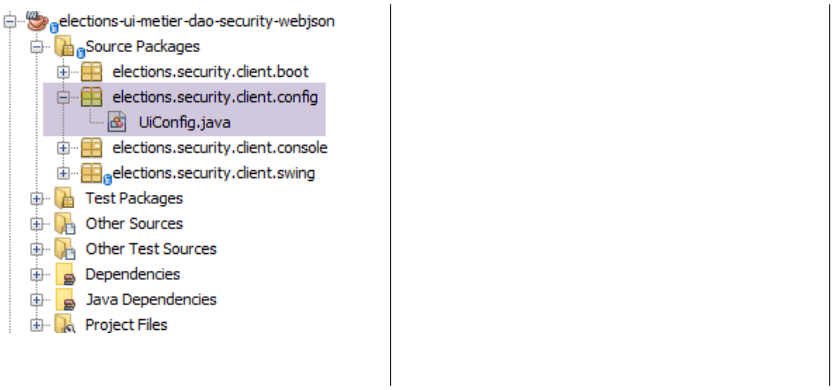
17.5.1 Configuration Maven

Le fichier [pom.xml] du nouveau projet est le suivant :

```
1. <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2.   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
3.   <modelVersion>4.0.0</modelVersion>
4.   <groupId>istia.st.elections</groupId>
5.   <artifactId>elections-console-metier-dao-security-webjson</artifactId>
6.   <version>0.0.1-SNAPSHOT</version>
7.   <name>elections-console-metier-dao-security-webjson</name>
8.   <description>couche console du client web / jSON</description>
9.
10.    <properties>
11.        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
12.        <java.version>1.8</java.version>
13.    </properties>
14.
15.    <parent>
16.        <groupId>org.springframework.boot</groupId>
17.        <artifactId>spring-boot-starter-parent</artifactId>
18.        <version>1.2.7.RELEASE</version>
19.    </parent>
20.
21.    <dependencies>
22.        <dependency>
23.            <groupId>istia.st.elections</groupId>
24.            <artifactId>elections-metier-dao-security-webjson</artifactId>
25.            <version>0.0.1-SNAPSHOT</version>
26.        </dependency>
27.    </dependencies>
28.
29.    <build>
30.        <plugins>
31.            <plugin>
32.                <groupId>org.apache.maven.plugins</groupId>
33.                <artifactId>maven-surefire-plugin</artifactId>
34.                <version>2.18.1</version>
35.            </plugin>
36.        </plugins>
37.    </build>
38.
39. </project>
```

- lignes 4-8 : l'identité Maven du nouveau projet ;
- lignes 22-26 : la dépendance sur le projet [elections-metier-dao-security-webjson] que nous venons de construire au paragraphe 17.4, page 330, et qui fournit les couches [DAO] et [métier] ;

17.5.2 Configuration Spring

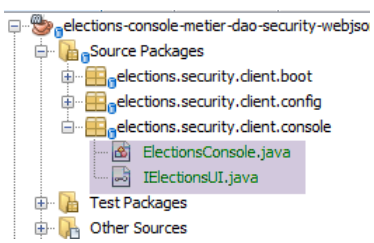


La classe [UiConfig] est la suivante :

```
1. package elections.security.client.config;
2.
3. import elections.security.client.entities.User;
4. import org.springframework.context.annotation.Bean;
5. import org.springframework.context.annotation.ComponentScan;
6. import org.springframework.context.annotation.Import;
7.
8. @Import(MetierConfig.class)
9. @ComponentScan(basePackages = {"elections.security.client.console","elections.security.client.swing"})
10.  public class UiConfig {
11.
12.    // administrateur
13.    private final User ADMIN = new User("admin", "admin");
14.
15.    @Bean
16.    private User admin() {
17.      return ADMIN;
18.    }
19.
20.  }
```

- ligne 8 : on importe la classe [elections.security.client.config.MetierConfig] du projet [elections-metier-dao-security-webjson] étudié au paragraphe 17.4, page 330 ;
- ligne 9 : on déclare les packages où se trouvent les beans Spring ;
- lignes 12-18 : le bean [admin] est l'utilisateur [admin, admin] qui sera utilisé par l'application console. On se rappelle que c'est le seul utilisateur autorisé à interroger el serveur web / jSON sécurisé ;

17.5.3 L'application de boot de la console



La classe [ElectionsConsole] évolue de la façon suivante :

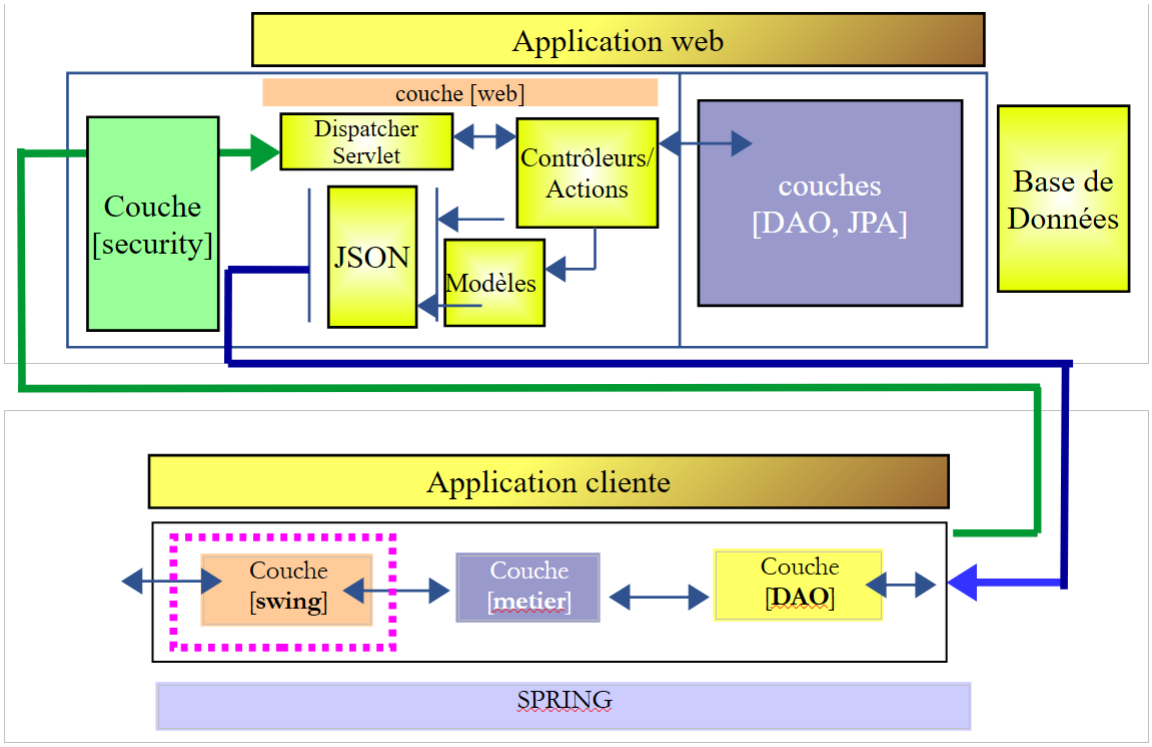
```
1. package elections.security.client.console;
2.
3. import java.util.Arrays;
4. import java.util.Comparator;
5. import java.util.Scanner;
6.
7. import org.springframework.beans.factory.annotation.Autowired;
8. import org.springframework.stereotype.Component;
9.
10.  import elections.security.client.entities.ElectionsException;
11.  import elections.security.client.entities.ListeElectorale;
12.  import elections.security.client.entities.User;
13.  import elections.security.client.metier.IElectionsMetier;
14.
15.  @Component
16.  public class ElectionsConsole implements IElectionsUI {
17.
18.    @Autowired
19.    private IElectionsMetier electionsMetier;
20.
21.    @Autowired
22.    private User admin;
23.
24.    @Override
25.    public void run() {
26.        // les listes en compétition
27.        ListeElectorale[] listes;
28.        // saisie des données
29.        try (Scanner clavier = new Scanner(System.in)) {
30.            // on demande les listes en compétition à la couche [metier]
31.            listes = electionsMetier.getListesElectorales(admin);
32.            // on fait la saisie des voix
33.            System.out.println("Il y a " + listes.length
```

```
34.                 + " listes en compétition. Veuillez indiquer le nombre de voix de chacune d'elles :");
35.         for (int i = 0; i < listes.length; i++) {
36.             boolean saisieOK = false;
37.             while (!saisieOK) {
38.                 System.out.print("Nombre de voix de la liste [" + listes[i].getNom() + "] : ");
39.                 try {
40.                     listes[i].setVoix(Integer.parseInt(clavier.nextLine()));
41.                     saisieOK = true;
42.                 } catch (ElectionsException | NumberFormatException ex) {
43.                     System.out.println("Nombre de voix incorrect. Veuillez recommencer");
44.                 }
45.             }
46.         }
47.     }
48.     // on fait le calcul des sièges
49.     listes=electionsMetier.calculerSieges(admin,listes);
50.     // on enregistre les résultats
51.     electionsMetier.recordResultats(admin,listes);
52.     // tri des listes dans l'ordre décroissant des voix
53.     Arrays.sort(listes, new CompareListesElectorales());
54.     // on les affiche
55.     System.out.println("\nRésultats de l'élection\n");
56.     for (int i = 0; i < listes.length; i++) {
57.         System.out.println(listes[i]);
58.     }
59. }
60.
61. // classe de comparaison de listes électorales
62. class CompareListesElectorales implements Comparator<ListeElectorale> {
63.
64.     // comparaison de deux listes candidates selon le nombre de voix
65.     @Override
66.     public int compare(ListeElectorale listeElectorale1, ListeElectorale listeElectorale2) {
67.         // on compare les voix de ces deux listes
68.         int nbVoix1 = listeElectorale1.getVoix();
69.         int nbVoix2 = listeElectorale2.getVoix();
70.         if (nbVoix1 < nbVoix2) {
71.             return +1;
72.         } else {
73.             if (nbVoix1 > nbVoix2)
74.                 return -1;
75.             else
76.                 return 0;
77.         }
78.     }
79. }
80.
81. }
```

La classe ElectionsConsole bouge très peu. Il faut simplement se rappeler que maintenant les méthodes de la couche [métier] réclament comme premier paramètre, un utilisateur (lignes 31, 49, 51). Celui-ci est fourni par l'injection des lignes 21-22. L'utilisateur injecté est celui autorisé à interroger le serveur web / jSON.

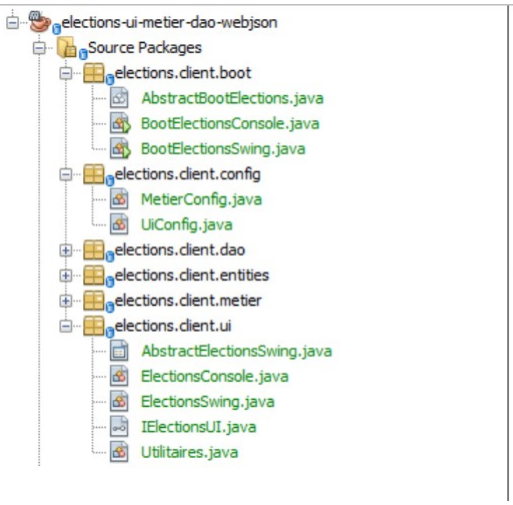
Travail à faire : testez l'application console (pensez à lancer auparavant le serveur sécurisé).

17.6 Le client du serveur sécurisé avec une couche [swing]

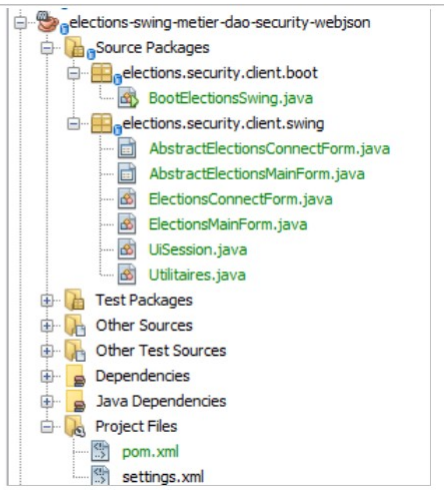


Nous créons un nouveau projet Netbeans [elections-swing-metier-dao-security-webjson] par un copy / paste du projet [elections-ui-metier-dao-webjson] :

1



2



Dans le nouveau projet [2] :

- supprimez les packages [elections.client.dao, elections.client.metier, elections.client.entities] ;
- dans le package [elections.client.ui], supprimez les classes [ElectionsConsole, IElectionsUI] ;
- renommez les packages comme indiqué en [2] ;
- dans le package [elections.security.client.swing], renommez la classe [AbstractElectionsSwing] qui implémente l'interface graphique en [AbstractElectionsMainForm] et la classe [ElectionsSwing] qui implémente les gestionnaires des événements de l'interface graphque en [ElectionsMainForm] ;
- réglez les problèmes d'import dans les diverses classes par [clic droit / Fix Imports] ;

A ce stade, il y a de nombreuses erreurs.

17.6.1 Configuration Maven

Le fichier [pom.xml] évolue de la façon suivante :

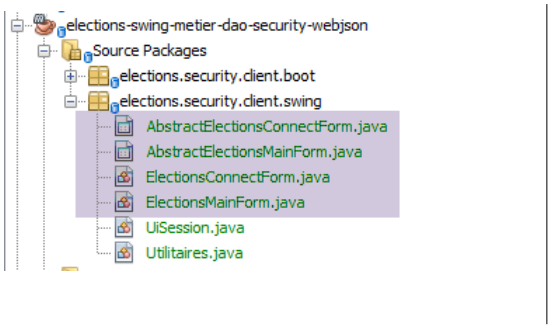
```
1. <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2.   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
3.   <modelVersion>4.0.0</modelVersion>
4.   <groupId>istia.st.elections</groupId>
5.   <artifactId>elections-swing-metier-dao-security-webjson</artifactId>
6.   <version>0.0.1-SNAPSHOT</version>
7.   <name>elections-swing-metier-dao-security-webjson</name>
8.   <description>couche swing du client web / jSON</description>
9.
10.    <properties>
11.        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
12.        <java.version>1.8</java.version>
13.    </properties>
14.
15.    <parent>
16.        <groupId>org.springframework.boot</groupId>
17.        <artifactId>spring-boot-starter-parent</artifactId>
18.        <version>1.2.7.RELEASE</version>
19.    </parent>
20.
21.    <dependencies>
22.        <dependency>
23.            <groupId>istia.st.elections</groupId>
24.            <artifactId>elections-console-metier-dao-security-webjson</artifactId>
25.            <version>0.0.1-SNAPSHOT</version>
26.        </dependency>
27.    </dependencies>
28.
29.    <build>
30.        <plugins>
31.            <plugin>
32.                <groupId>org.apache.maven.plugins</groupId>
33.                <artifactId>maven-surefire-plugin</artifactId>
34.                <version>2.18.1</version>
35.            </plugin>
36.        </plugins>
37.    </build>
38.
39.  </project>
```

- lignes 4-8 : l'identité Maven du nouveau projet ;
- lignes 22-26 : une dépendance sur le projet console que nous venons d'étudier au paragraphe 17.5, page 336 ;

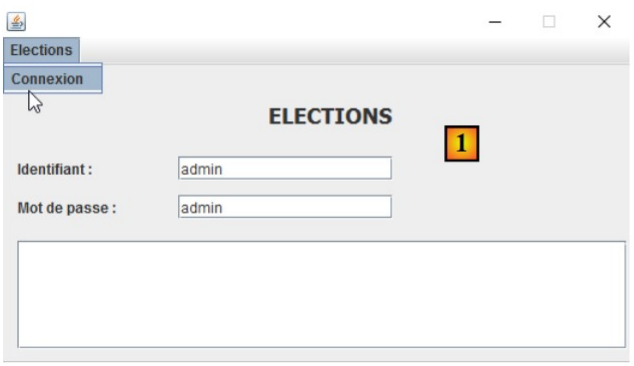
17.6.2 Configuration Spring

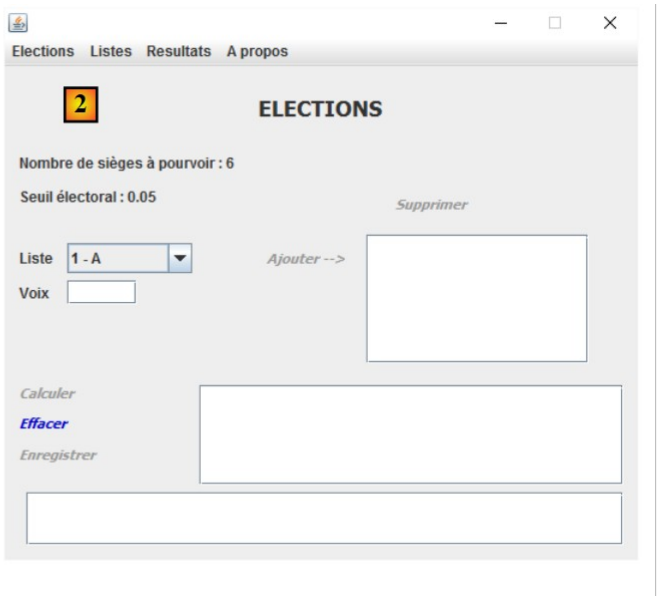
Ce projet utilise la configuration Spring du projet console qu'il importe (cf paragraphe 17.5.2, page 338).

17.6.3 Les vues de l'application Swing



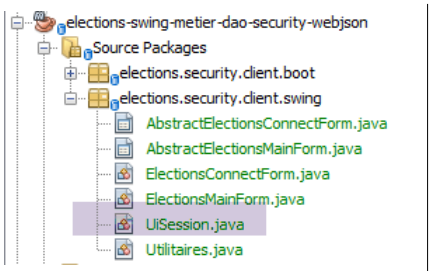
Ce projet va utiliser deux vues :





- la vue [1] de connexion est nouvelle. Elle sert à authentifier l'utilisateur qui veut utiliser l'application. Elle utilise les classes :
  - [AbstractElectionsConnectForm] qui implémente la vue ;
  - [ElectionsConnectForm] qui gère les événements de la vue ;
- la vue [2] est connue. C'est celle utilisée jusqu'à maintenant. Elle utilise les classes :
  - [AbstractElectionsMainForm] qui implémente la vue ;
  - [ElectionsMainForm] qui gère les événements de la vue ;

17.6.4 La session

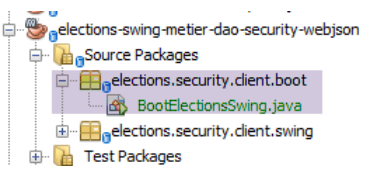


Lorsqu'une application Swing a plusieurs vues, il faut trouver un moyen pour qu'une vue puisse passer de l'information à une autre vue. Nous utilisons un concept utilisé en développement web : la session qui permet aux vues associées à un utilisateur précis de partager de l'information. Ce concept sera implémenté ici à l'aide d'un singleton Spring qui sera injecté dans les deux classes qui gèrent les événements des deux vues :

```
1. package elections.security.client.swing;
2.
3. import elections.security.client.entities.User;
4. import org.springframework.stereotype.Component;
5.
6. @Component
7. public class UiSession {
8.
9.   // l'utilisateur connecté
10.    private User user;
11.
12.    // getters et setters
13.  ...
14.  }
```

- ligne 6 : la classe [UiSession] est un composant Spring ;
- ligne 10 : elle ne sert qu'à une chose : mémoriser l'utilisateur qui se connecte avec la vue [1]. La vue [2] qui a besoin de connaître celui-ci ira le chercher là ;

17.6.5 La classe de boot



La classe de boot de l'application graphique est la suivante :

```
1. package elections.security.client.boot;
2.
3. import elections.security.client.console.IElectionsUI;
4.
5. public class BootElectionsSwing extends AbstractBootElections {
6.   public static void main(String[] arguments) {
7.     new BootElectionsSwing().run();
8.   }
9.
10.    @Override
11.    protected IElectionsUI getUI() {
12.        return ctx.getBean("electionsConnectForm", IElectionsUI.class);
13.    }
14.  }
```

- ligne 5 : la classe [BootElectionsSwing] étend la classe [AbstractBootElections] définit dans le projet console présent dans les dépendances du projet ;
- lignes 10-13 : la classe [BootElectionsSwing] fera afficher la vue de connexion [ElectionsConnectForm] ;
- lignes 6-8 : c'est la méthode [run] de cette classe qui sera exécutée ;

17.6.6 La classe [ElectionsMainForm]

La classe [ElectionsMainForm] est celle qui s'appelait auparavant [ElectionsSwing]. Elle gère les événements de la vue [AbstractElectionsMainForm]. Son code évolue de la façon suivante :

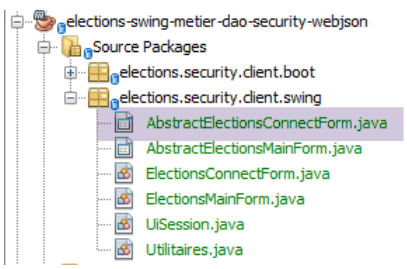
```
1. package elections.security.client.swing;
2.
3. import elections.security.client.console.IElectionsUI;
4. import java.awt.Dimension;
5. import java.awt.Toolkit;
6. import java.util.ArrayList;
7. import java.util.List;
8.
9. import javax.swing.DefaultListModel;
10.  import javax.swing.JLabel;
11.  import javax.swing.JMenuItem;
12.  import javax.swing.JOptionPane;
13.
14.  import org.springframework.beans.factory.annotation.Autowired;
15.  import org.springframework.stereotype.Component;
16.
17.  import elections.security.client.entities.ElectionsException;
18.  import elections.security.client.entities.ListeElectorale;
19.  import elections.security.client.entities.User;
20.  import elections.security.client.metier.IElectionsMetier;
21.
22.  @Component
23.  public class ElectionsMainForm extends AbstractElectionsMainForm implements IElectionsUI {
24.
25.    private static final long serialVersionUID = 1L;
26.
27.    // référence sur la couche [métier]
28.    @Autowired
29.    private IElectionsMetier metier;
30.
31.    // session UI
32.    @Autowired
33.    private UiSession uiSession;
34.
35.    // utilisateur connecté
36.    private User user;
37.
38.    // modèles des listes JList
39.    private DefaultListModel<String> modèleNomsVoix = null;
40.    private DefaultListModel<String> modèleRésultats = null;
```

```
41.
42.    // les listes en compétition
43.    private ListeElectorale[] listes;
44.
45.    // listes saisies par l'utilisateur
46.    private final List<ListeElectorale> listesSaisies = new ArrayList<>();
47.    private ListeElectorale[] tListesSaisies;
48.
49.    // initialisations
50.    @Override
51.    protected void init() {
52.      // génération des composants par la classe parent
53.      super.init();
54.      // initialisations locales
55.      modèleNomsVoix = new DefaultListModel<>();
56.      jListNomsVoix.setModel(modèleNomsVoix);
57.      modèleRésultats = new DefaultListModel<>();
58.      jListResultats.setModel(modèleRésultats);
59.      String info;
60.      boolean erreur = false;
61.      try {
62.        // on demande les listes à la couche [métier]
63.        listes = metier.getListesElectorales(user);
64.        // on associe les noms des listes au combo jComboBoxNomsListes
65.        for (int i = 0; i < listes.length; i++) {
66.          jComboBoxNomsListes.addItem(String.format("%s - %s", listes[i].getId(), listes[i].getNom()));
67.        }
68.        // ainsi que les paramètres de l'élection
69.        int nbSiegesAPourvoir = metier.getNbSiegesAPourvoir(user);
70.        double seuilElectoral = metier.getSeuilElectoral(user);
71.        // on initialise les labels liés à ces deux informations
72.        jLabelSAP.setText(jLabelSAP.getText() + nbSiegesAPourvoir);
73.        jLabelSE.setText(jLabelSE.getText() + seuilElectoral);
74.        // message de succès
75.        info = "Source de données lue avec succès";
76.      } catch (ElectionsException ex1) {
77.        // on note l'erreur
78.        info = getInfoForException("Les erreurs suivantes se sont produites :", ex1);
79.        erreur = true;
80.      } catch (RuntimeException ex2) {
81.        // on note l'erreur
82.        info = getInfoForException("Les erreurs suivantes se sont produites :", ex2);
83.        erreur = true;
84.      }
85.      // erreur ?
86.      if (erreur) {
87.        // on affiche l'info
88.        jTextPaneMessages.setText(info);
89.        jTextPaneMessages.setCaretPosition(0);
90.        return;
91.      } else {
92.        jTextPaneMessages.setText("");
93.      }
94.      // état formulaire
95.      Utilitaires.setEnabled(new JLabel[]{jLabelAjouter, jLabelCalculer, jLabelEnregistrer, jLabelSupprimer}, false);
96.      Utilitaires.setEnabled(
97.              new JMenuItem[]{jMenuItemAjouter, jMenuItemCalculer, jMenuItemEnregistrer, jMenuItemSupprimer}, false);
98.      // centrer la fenêtre
99.      Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
100.     Dimension frameSize = getSize();
101.     if (frameSize.height > screenSize.height) {
102.       frameSize.height = screenSize.height;
103.     }
104.     if (frameSize.width > screenSize.width) {
105.       frameSize.width = screenSize.width;
106.     }
107.     setLocation((screenSize.width - frameSize.width) / 2, (screenSize.height - frameSize.height) / 2);
108.
109.   }
110. ...
111.
112.   @Override
113.   public void run() {
114.     // mémorisation utilisateur
115.     user = uiSession.getUser();
116.     // initialisation fenêtre
117.     init();
118.     setVisible(true);
119.   }
120. }
```

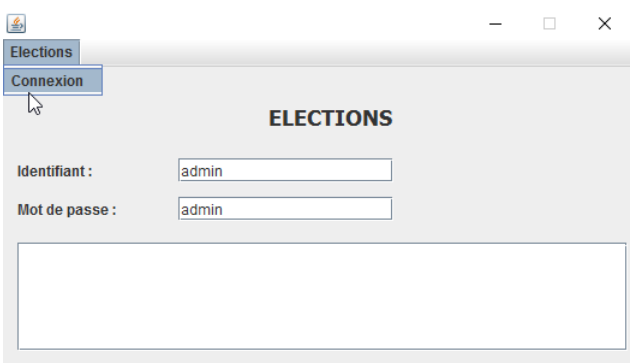
- lignes 32-33 : injection de la session de l'application ;
- lignes 112-119 : la vue sera activée comme précédemment par sa méthode [run] ;

- ligne 115 : on récupère l'utilisateur dans la session. Celui-ci y aura été mis par le code de la vue de connexion [ElectionsConnectForm]. Le code ensuite est modifié pour que le 1er paramètre des appels à la couche [métier] soient cet utilisateur (lignes 63, 69-70 par exemple) ;

17.6.7 La vue [AbstractElectionsConnectForm]

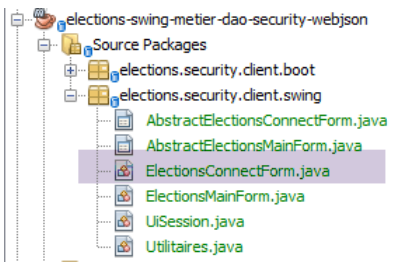


Construisez avec Netbeans le formulaire suivant :



La classe n'implémentera pas elle-même la méthode gérant le clic sur l'option de menu [Connexion]. Elle fera appel à une méthode [doConnecter] déclarée abstraite et la classe de la vue sera elle-même déclarée abstraite. On supprimera la méthode statique [main] qui a été générée automatiquement.

17.6.8 La gestion des événements de la vue de connexion

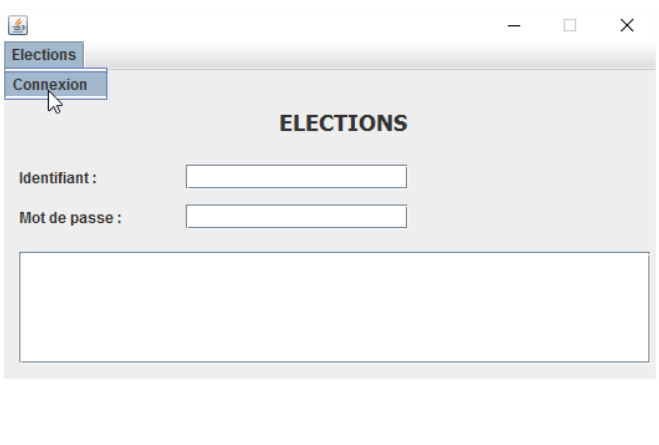


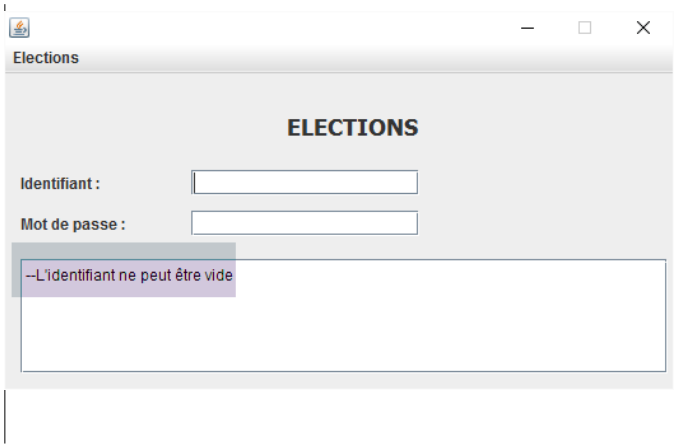
La classe [ElectionsConnectForm] implémente la gestion des événements de la vue de connexion de la façon suivante :

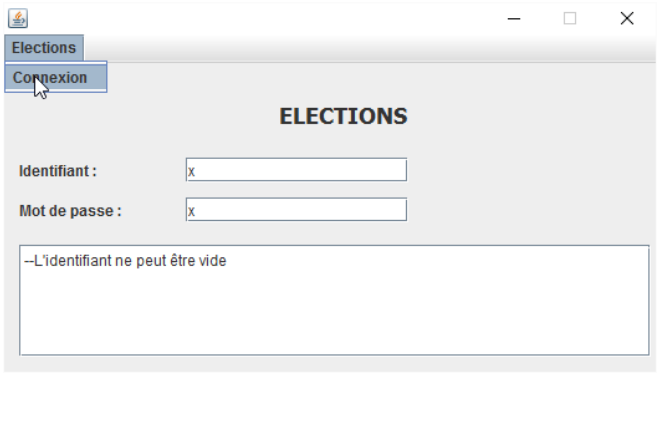
```
1. package elections.security.client.swing;
2.
3. import elections.security.client.console.IElectionsUI;
4. import elections.security.client.entities.ElectionsException;
5. import elections.security.client.entities.User;
6. import elections.security.client.metier.IElectionsMetier;
7. import java.awt.Dimension;
8. import java.awt.Toolkit;
9. import javax.swing.SwingUtilities;
```

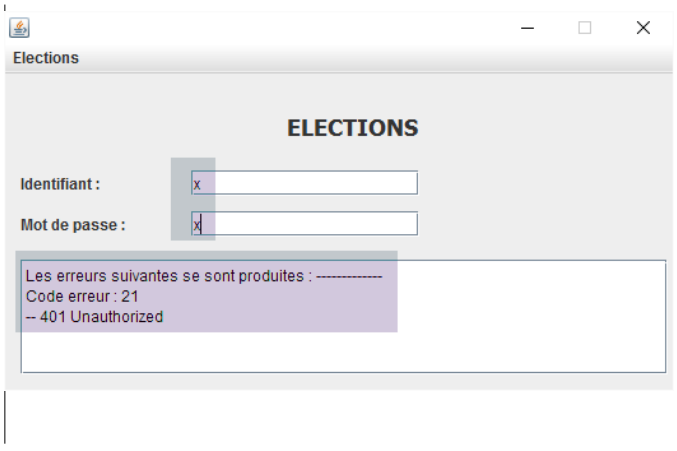
```
10. import org.springframework.beans.factory.annotation.Autowired;
11. import org.springframework.stereotype.Component;
12.
13. @Component
14. public class ElectionsConnectForm extends AbstractElectionsConnectForm implements IElectionsUI {
15.
16.   private static final long serialVersionUID = 1L;
17.
18.   // référence sur la couche [métier]
19.   @Autowired
20.   private IElectionsMetier metier;
21.
22.   // utilisateur connecté
23.   private User user;
24.
25.   // formulaire principal
26.   @Autowired
27.   private ElectionsMainForm electionsMainForm;
28.
29.   // session UI
30.   @Autowired
31.   private UiSession uiSession;
32.
33.   @Override
34.   protected void doConnect() {
35.     String info = null;
36.     try {
37.       if (isPageValid()) {
38.         // authentification de l'utilisateur
39.         metier.authenticate(user);
40.         // on mémorise l'utilisateur dans la session
41.         uiSession.setUser(user);
42.         // la vue de connexion est cachée
43.         setVisible(false);
44.         // la vue principale est affichée
45.         electionsMainForm.run();
46.       }
47.     } catch (ElectionsException ex1) {
48.       // on note l'erreur
49.       info = getInfoForException("Les erreurs suivantes se sont produites :", ex1);
50.     } catch (RuntimeException ex2) {
51.       // on note l'erreur
52.       info = getInfoForException("Les erreurs suivantes se sont produites :", ex2);
53.     }
54.     // erreur ?
55.     if (info != null) {
56.       // on affiche l'info
57.       jTextPaneErreurs.setText(info);
58.       jTextPaneErreurs.setCaretPosition(0);
59.     }
60.   }
61.
62.   // initialisations
63.   @Override
64.   protected void init() {
65.     // génération des composants par la classe parent
66.     super.init();
67.     // initialisations locales
68. ...
69.     // centrer la fenêtre
70. ...
71.   }
72.
73.   @Override
74.   public void run() {
75.     // on affiche l'interface graphique
76.     SwingUtilities.invokeLater(new Runnable() {
77.       public void run() {
78.         init();
79.         setVisible(true);
80.       }
81.     });
82.   }
83.
84.   private boolean isPageValid() {
85.     ...
86.   }
87.
88.   private String getInfoForException(String message, ElectionsException ex) {
89. ...
90.   }
91.
92.   private String getInfoForException(String message, RuntimeException ex) {
93. ...
94.   }
95.
96. }
```

- lignes 73-82 : la méthode [run] qui sera appelée par la classe de boot [BootElectionsSwing] ;
- lignes 26-27 : injection d'une référence sur la vue n° 2 qui devra être affichée si l'utilisateur est reconnu ;
- lignes 30-31 : injection de la session de l'application ;
- ligne 84 : la méthode [isPageValid] fait deux choses :
  - elle vérifie que l'identifiant n'est pas vide (le mot de passe peut l'être lui) ;
  - instancie avec les saisies le champ User de la ligne 23 ;









Travail à faire : complétez le code de la classe.

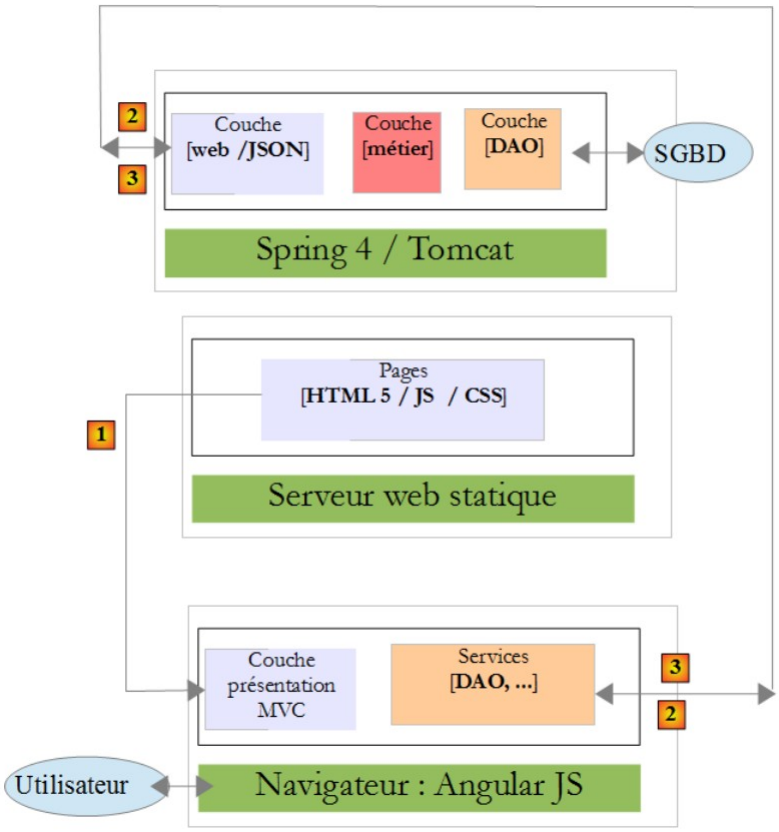
Travail à faire : Testez l'application.

18 [Cours] : Gestion des accès inter-domaines

Mots clés : CORS (Cross-Origin Resource Sharing).

Ce chapitre est un peu à l'écart du TD. Il a été gardé parce qu'il introduit la programmation web et la programmation Javascript. Il faut se rappeler ici que l'un des objectifs de ce TD est de présenter les concepts fréquemment utilisés dans le dévelopement JEE, ç-à-d le développement web s'appuyant sur des frameworks Java. On complète ici, le serveur web utilisé dans l'étude de la base de données de produits et de catégories pour lui permettre d'accepter des requêtes inter-domaines.

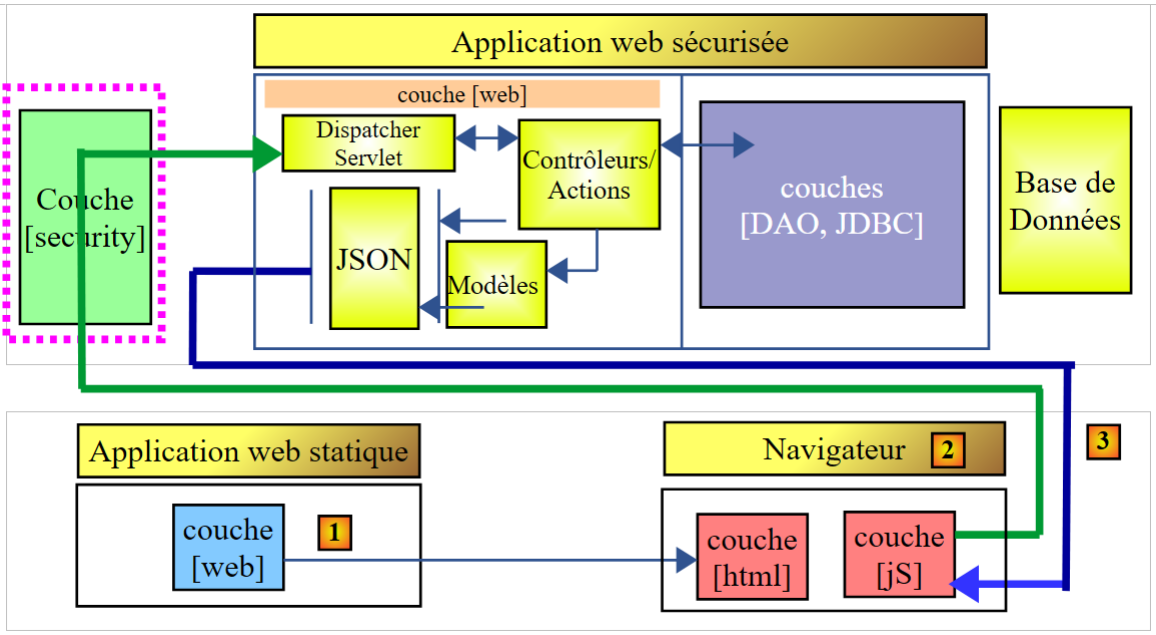
Dans le document [Tutoriel AngularJS / Spring 4], on développe une application client / serveur où le client est une application AngularJS :



- les pages HTML / CSS / JS de l'application Angular viennent du serveur [1] ;
- en [2], le service [dao] fait une requête à un autre serveur, le serveur [2]. Et bien ça, c'est interdit par le navigateur qui exécute l'application Angular parce que c'est une faille de sécurité. L'application ne peut interroger que le serveur d'où elle vient, ç-à-d le serveur [1] ;

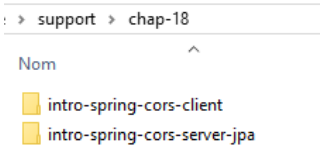
En fait, il est inexact de dire que le navigateur interdit à l'application Angular d'interroger le serveur [2]. Elle l'interroge en fait pour lui demander s'il autorise un client qui ne vient pas de chez lui à l'interroger. On appelle cette technique de partage, le **CORS** (Cross-Origin Resource Sharing). Le serveur [2] donne son accord en envoyant des entêtes HTTP précis.

Nous allons créer l'architecture suivante :



- en [1], une application web délivre des pages HTML / jS ;
- en [2], le navigateur exécute le Javascript embarqué dans les pages HTML pour interroger le service web sécurisé [3] ;

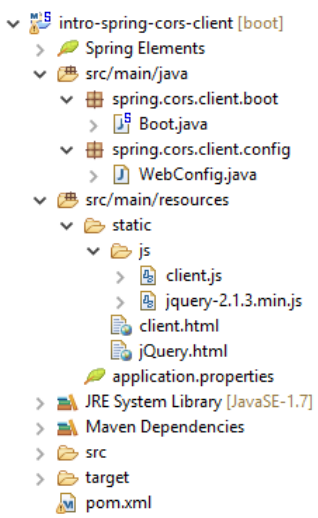
18.1 Support



Les projets de ce chapitre seront trouvés dans le dossier [support / chap-18].

18.2 Le projet du client

On crée le projet Eclipse suivant :



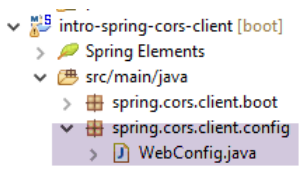
18.3 Configuration Maven

Le projet est un projet Maven avec le fichier [pom.xml] suivant :

```
1. <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2.   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
3.   <modelVersion>4.0.0</modelVersion>
4.   <groupId>istia.st.webjson</groupId>
5.   <artifactId>intro-server-webjson-01</artifactId>
6.   <version>0.0.1-SNAPSHOT</version>
7.
8.   <name>intro-server-webjson-01</name>
9.   <description>démo spring mvc</description>
10.
11.    <parent>
12.        <groupId>org.springframework.boot</groupId>
13.        <artifactId>spring-boot-starter-parent</artifactId>
14.        <version>1.2.7.RELEASE</version>
15.    </parent>
16.
17.    <dependencies>
18.        <dependency>
19.            <groupId>istia.st.springdata</groupId>
20.            <artifactId>intro-spring-data-01</artifactId>
21.            <version>0.0.1-SNAPSHOT</version>
22.        </dependency>
23.        <dependency>
24.            <groupId>org.springframework.boot</groupId>
25.            <artifactId>spring-boot-starter-web</artifactId>
26.        </dependency>
27.    </dependencies>
28.
29.    <build>
30.        <plugins>
31.            <plugin>
32.                <groupId>org.springframework.boot</groupId>
33.                <artifactId>spring-boot-maven-plugin</artifactId>
34.            </plugin>
35.            <plugin>
36.                <groupId>org.apache.maven.plugins</groupId>
37.                <artifactId>maven-surefire-plugin</artifactId>
38.                <version>2.18.1</version>
39.            </plugin>
40.        </plugins>
41.    </build>
42.
43. </project>
```

- lignes 11-15 : c'est un projet Spring Boot ;
- lignes 23-26 : on utilise la dépendance [spring-boot-starter-web] qui amène avec elle un serveur Tomcat et Spring MVC ;

18.4 Configuration Spring

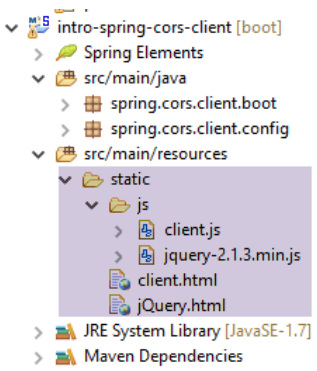


La classe [WebConfig] qui configure le projet Spring est la suivante :

```
1. package spring.cors.client.config;
2.
3. import org.springframework.beans.factory.annotation.Autowired;
4. import org.springframework.boot.context.embedded.EmbeddedServletContainerFactory;
5. import org.springframework.boot.context.embedded.ServletRegistrationBean;
6. import org.springframework.boot.context.embedded.tomcat.TomcatEmbeddedServletContainerFactory;
7. import org.springframework.context.ApplicationContext;
8. import org.springframework.context.annotation.Bean;
9. import org.springframework.web.context.WebApplicationContext;
10.  import org.springframework.web.servlet.DispatcherServlet;
11.  import org.springframework.web.servlet.config.annotation.EnableWebMvc;
12.  import org.springframework.web.servlet.config.annotation.ResourceHandlerRegistry;
13.  import org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter;
14.
15.  @EnableWebMvc
16.  public class WebConfig extends WebMvcConfigurerAdapter {
17.
18.    // ------------------------------ configuration couche [web]
19.    @Autowired
20.    private ApplicationContext context;
21.
22.    @Bean
23.    public DispatcherServlet dispatcherServlet() {
24.        DispatcherServlet servlet = new DispatcherServlet((WebApplicationContext) context);
25.        return servlet;
26.    }
27.
28.    @Bean
29.    public ServletRegistrationBean servletRegistrationBean(DispatcherServlet dispatcherServlet) {
30.        return new ServletRegistrationBean(dispatcherServlet, "/*");
31.    }
32.
33.    @Bean
34.    public EmbeddedServletContainerFactory embeddedServletContainerFactory() {
35.        return new TomcatEmbeddedServletContainerFactory("", 8081);
36.    }
37.
38.    @Override
39.    public void addResourceHandlers(ResourceHandlerRegistry registry) {
40.        registry.addResourceHandler("/*.html").addResourceLocations("classpath:/static/");
41.        registry.addResourceHandler("/*.js").addResourceLocations("classpath:/static/js/");
42.    }
43.  }
```

- ligne 15 : la classe configure un projet Spring MVC ;
- ligne 16 : la classe étend la classe [WebMvcConfigurerAdapter] pour redéfinir certaines de ses méthodes ;
- lignes 18-36 : nous avons déjà rencontré ces beans, par exemple au paragraphe 13.5.3.1, page 228 . On notera, ligne 35, que le service web fonctionnera sur le port 8081 ;
- lignes 38-42 : la méthode [addResourceHandlers] permet de définir des ressources statiques, ç-à-d des ressources non traitées par la [DispatcherServlet] de la ligne 23 ;
- ligne 40 : toute requête d'une ressource ayant un suffixe .html aura pour réponse le fichier demandé par la requête et trouvé dans le dossier [static] du Classpath du projet ;
- ligne 41 : toute requête d'une ressource ayant un suffixe .js aura pour réponse le fichier Javascript demandé par la requête et trouvé dans le dossier [static/js] du Classpath du projet ;

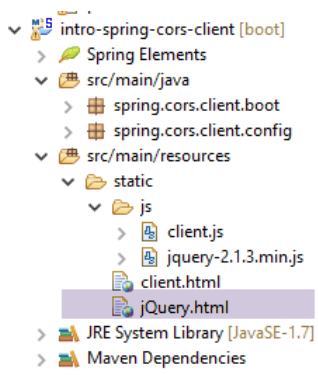
https://tahe.developpez.com - Ce cours tutoriel écrit par Serge Tahé est mis à disposition du public selon les termes de la Licence Creative Commons Attribution – Pas d'Utilisation Commerciale – Partage dans les Mêmes Conditions 3.0 non transposé.



18.5 Rudiments de jQuery et de Javascript

La page HTML du client sera la suivante :

Elle embarquera avec elle du code Javascript (jS) exécuté dans le navigateur. Nous allons présenter quelques rudiments de Javascript qui vont nous permettre de comprendre le code. Le client va faire des appels HTTP à l'aide de la bibliothèque jQuery [https://jquery.com/] qui apporte de nombreuses fonctions facilitant le développement Javascript. Nous créons un fichier statique HTML [jQuery.html] que l'on place dans le dossier [static] :

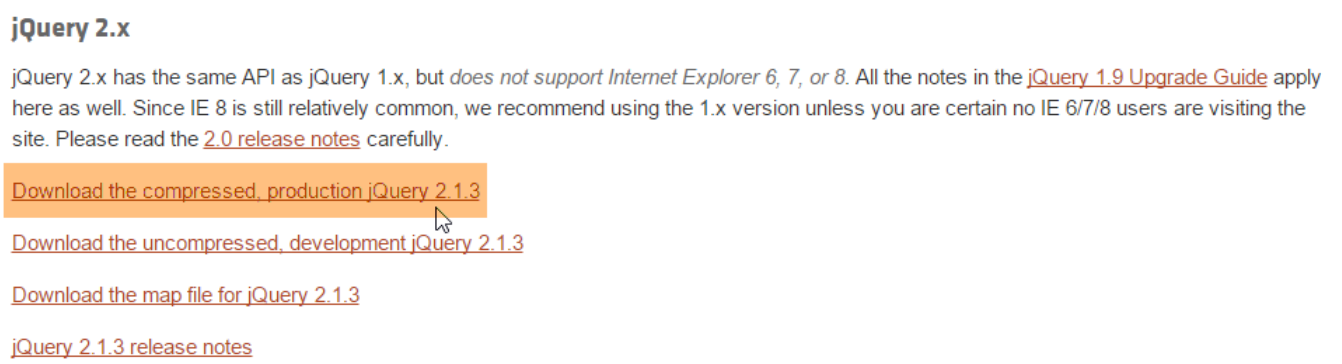


Ce fichier aura le contenu suivant :

```
1. <!DOCTYPE html>
2. <html xmlns="http://www.w3.org/1999/xhtml">
3. <head>
4. <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
5. <title>JQuery-01</title>
6. <script type="text/javascript" src="/jquery-2.1.3.min.js"></script>
7. </head>
8. <body>
9.   <h3>Rudiments de JQuery</h3>
10.    <div id="element1">Elément 1</div>
11.  </body>
12.  </html>
```

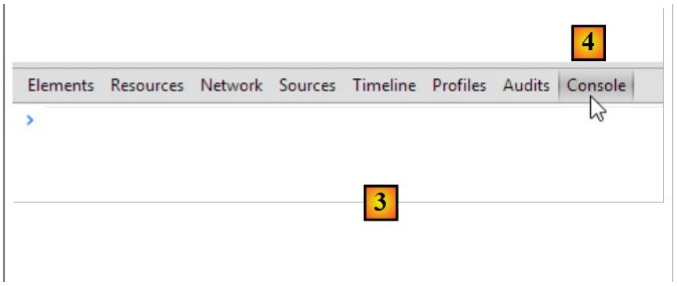
- ligne 6 : importation de jQuery ;
- lignes 10-12 : un élément de la page d'**id** [element1]. Nous allons jouer avec cet élément.

Il nous faut télécharger le fichier [*jquery-2.1.3.min.js*]. On trouvera la dernière version de jQuery à l'URL [http://jquery.com/download/] :

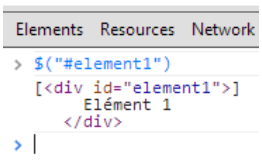
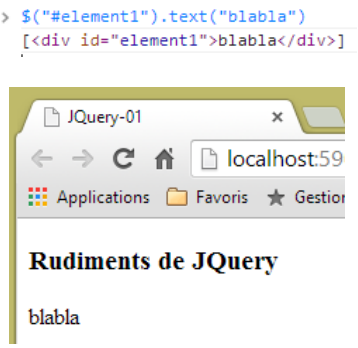
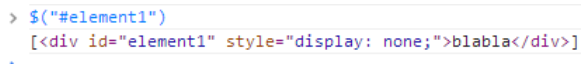


On placera le fichier téléchargé dans le dossier [static / js] et on changera la ligne 6 du fichier HTML en fonction de la version installée.

Ceci fait, on demande la vue statique [jQuery.html] avec Chrome [1-2] :



Avec Google Chrome, faire [Ctrl-Maj-I] pour faire apparaître les outils de développement [3]. L'onglet [Console] [4] permet d'exécuter du code Javascript. Nous donnons dans ce qui suit des commandes Javascript à taper et nous en donnons une explication.

JS	résultat
`$("#element1")` : rend la collection de tous les éléments d'**id** [element1], donc normalement une collection de 0 ou 1 élément parce qu'on ne peut avoir deux **id** identiques dans une page HTML.	
`$("#element1").text("blabla")` : affecte le texte [blabla] à tous les éléments de la collection. Ceci a pour effet de changer le contenu affiché par la page	
`$("#element1").hide()` cache les éléments de la collection. Le texte [blabla] n'est plus affiché.	
`$("#element1")` : affiche de nouveau la collection. Cela nous permet de voir que l'élément d'id [element1] a l'attribut CSS **style='display : none;'** qui fait que l'élément est caché.	
`$("#element1").show()` : affiche les éléments de la collection. Le texte [blabla] apparaît de nouveau. C'est l'attribut CSS **style='display : block;'** qui assure cet affichage.	

`$("#element1").attr('style','color: red')`
: fixe un attribut à tous les éléments de la collection. L'attribut est ici [style] et sa valeur [color: red]. Le texte [blabla] passe en rouge.

```
> $("#element1").attr('style','color: red')
[<div id="element1" style="color: red">blabla</div>]
```

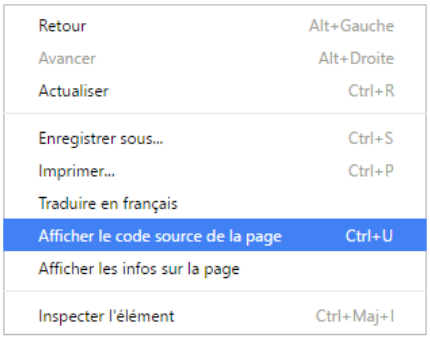
`Tableau`

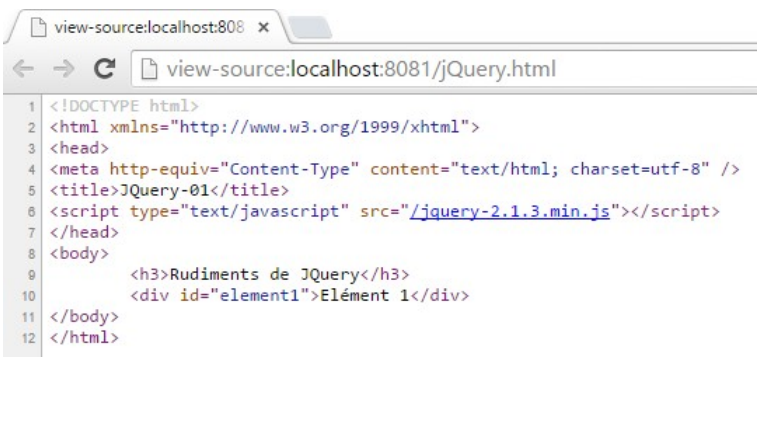
```
> var data=["zéro",1,"deux"]
undefined
> data[0]
"zéro"
> data[2]
"deux"
> data.length
3
> data[3]="trois"
"trois"
> data
["zéro", 1, "deux", "trois"]
> data[1]="un"
"un"
> data
["zéro", "un", "deux", "trois"]
```

`Dictionnaire`

```
> data={"zéro":0,"un":1,"erreur":"msg"}
Object {zéro: 0, un: 1, erreur: "msg"}
> data["zéro"]
0
> data.zéro
0
> data.msg="msg2"
"msg2"
> data
Object {zéro: 0, un: 1, erreur: "msg", msg: "msg2"}
> data.erreur="autre msg"
"autre msg"
> data
Object {zéro: 0, un: 1, erreur: "autre msg", msg: "msg2"}
```

On notera que l'URL du navigateur n'a pas changé pendant toutes ces manipulations. Il n'y a pas eu d'échanges avec le serveur web. Tout se passe à l'intérieur du navigateur. Maintenant, visualisons le code source de la page :

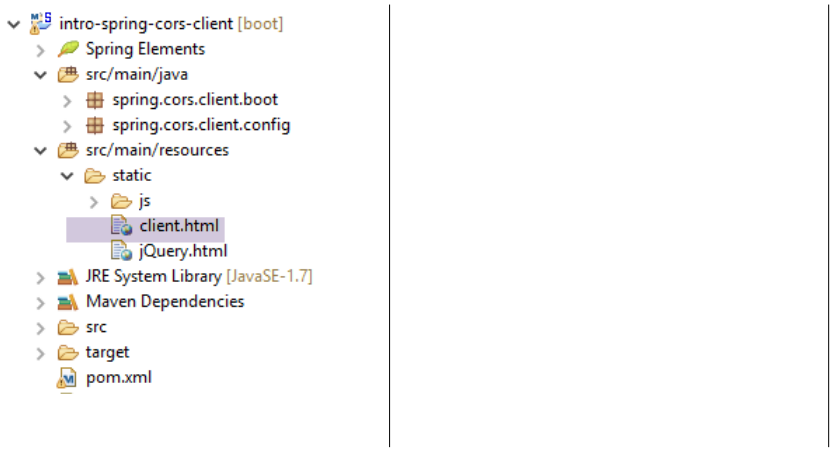




C'est le texte initial. Il ne reflète en rien les manipulations que l'on a faites sur l'élément des lignes 10-12. Il est important de s'en souvenir lorsqu'on fait du débogage Javascript. Il est alors souvent inutile de visualiser le code source de la page affichée.

18.6 Le code Javascript de l'application

Revenons à la page de l'application cliente qui va interroger le service web / jSON :



Le code HTML de cette page est le suivant :

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Spring MVC</title>
<script type="text/javascript" src="/jquery-2.1.3.min.js"></script>
<script type="text/javascript" src="/client.js"></script>
</head>
<body>
   <h2>Client du service web / jSON</h2>
   <form id="formulaire">
      <!--  identifiant -->
      Identifiant :
      <!--  -->
      <input type="text" id="identifiant" name="identifiant" value="" />
      <!--  mot de passe -->
      <br /> <br /> Mot de passe :
      <!--  -->
      <input type="text" id="password" name="password" value="" />
      <!--  méthode HTTP -->
      <br /> <br /> Méthode HTTP :
      <!--  -->
```

```
23.       <input type="radio" id="get" name="method" value="get"
24.           checked="checked" />GET
25.       <!--  -->
26.       <input type="radio" id="post" name="method" value="post" />POST
27.       <!--  URL -->
28.       <br /> <br />URL cible (commençant par /): <input type="text"
29.           id="url" size="30"><br />
30.       <!-- valeur postée -->
31.       <br /> Chaîne jSON à poster : <input type="text" id="posted"
32.           size="50" />
33.       <!-- bouton de validation -->
34.       <br /> <br /> <input type="button" value="Valider"
35.           onclick="javascript:requestServer()"></input>
36.   </form>
37.   <hr />
38.   <h2>Réponse du serveur</h2>
39.   <div id="response"></div>
40. </body>
41. </html>
```

- ligne 6 : on importe la bibliothèque jQuery ;
- ligne 7 : on importe un code que nous allons écrire ;
- lignes 15, 19, 26, 29, 31 : on notera les identifiants [id] des composants de la page. Le javascript référence ces composants via ces identifiants ;

Le code [client.js] est le suivant :

```
1. // données globales
2. var url;
3. var posted;
4. var response;
5. var method;
6. var baseUrl = 'http://localhost:8080';
7. var identifiant;
8. var password;
9. var authorizationHeader;
10.
11.  function requestServer() {
12.    // on récupère les informations
13.    var urlValue = url.val();
14.    var postedValue = posted.val();
15.    var identifiantValue = identifiant.val();
16.    var passwordValue = password.val();
17.    var method = document.forms[0].elements['method'].value;
18.    authorizationCode = btoa(identifiantValue + ':' + passwordValue);
19.    // on efface la réponse précédente
20.    response.text("");
21.    // on fait un appel Ajax à la main
22.    if (method === "get") {
23.        doGet(urlValue);
24.    } else {
25.        doPost(urlValue, postedValue);
26.    }
27.  }
28.
29.  function doGet(url) {
30.    // on fait un appel Ajax à la main
31.    $.ajax({
32.        headers : {
33.            'Authorization':'Basic '+authorizationCode
34.        },
35.        url : baseUrl + url,
36.        type : 'GET',
37.        dataType : 'text',
38.        beforeSend : function() {
39.        },
40.        success : function(data) {
41.            // résultat texte
42.            response.text(data);
43.        },
44.        complete : function() {
45.        },
46.        error : function(jqXHR) {
47.            // erreur système
48.            response.text(JSON.stringify(jqXHR.statusCode()));
49.        }
50.    })
51.  }
52.
53.  function doPost(url, posted) {
54.    // on fait un appel Ajax à la main
55.    $.ajax({
56.        headers : {
57.            'Authorization':'Basic '+authorizationCode
```

```
58.       },
59.       url : baseUrl + url,
60.       type : 'POST',
61.       contentType : 'application/json; charset=UTF-8',
62.       data : posted,
63.       dataType : 'text',
64.       beforeSend : function() {
65.       },
66.       success : function(data) {
67.           // résultat texte
68.           response.text(data);
69.       },
70.       complete : function() {
71.       },
72.       error : function(jqXHR) {
73.           // erreur système
74.           response.text(JSON.stringify(jqXHR.statusCode()));
75.       }
76.   })
77. }
78.
79. // au chargement du document
80. $(document).ready(function() {
81.   // on récupère les références des composants de la page
82.   identifiant = $("#identifiant");
83.   password = $("#password");
84.   url = $("#url");
85.   posted = $("#posted");
86.   response = $("#response");
87. });
```

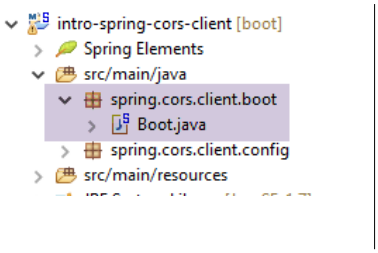
- lignes 80-87 : du code jS exécuté à la fin du chargement du document dans le navigateur ;
- lignes 81-86 : on récupère les références des différents éléments du document HTML, via leur identifiant [id] ;
- lignes 2-9 : des variables globales connues dans toute les fonctions définies dans le fichier jS ;
- ligne 13 : on récupère l'URL tapée par l'utilisateur ;
- ligne 14 : on récupère la valeur qu'il veut poster (vide si opération GET) ;
- ligne 15 : on récupère l'identifiant tapé par l'utilisateur ;
- ligne 16 : on récupère sont mot de passe ;
- ligne 17 : on récupère la façon [get] ou [post] à utiliser pour demander l'URL de la ligne 9 :
  - [**document**] désigne le document chargé par le navigateur, ce qu'on appelle le DOM (Document Object Model),
  - [**document.forms[0]**] désigne le 1er formulaire du document, un document pouvant en avoir plusieurs. Ici, il n'y en qu'un,
  - [**document.forms[0].elements['method']**] désigne l'élément du formulaire qui a l'attribut [name='method']. Il y en a deux :

```
1. <input type="radio" id="get" name="method" value="get" checked="checked" />GET
2. <input type="radio" id="post" name="method" value="post" />POST
```

  - [**document.forms[0].elements['method'].value**] est la valeur qui va être postée pour le composant qui a l'attribut [name='method']. On sait que la valeur postée est la valeur de l'attribut [value] du bouton radio coché. Ici, ce sera donc l'une des chaînes ['get', 'post'] ;
- ligne 18 : on construit l'encodage Base74 de la chaîne identifiant:password. Cet chaîne codée va servir dans l'entête HTTP [Authorization] que nous allons envoyer au serveur pour authentifier la requête ;
- lignes 22-26 : selon la méthode HTTP à utiliser, on exécute la méthode [doGet] ou [doPost] ;
- la méthode jQuery [$.ajax] effectue un appel HTTP ;
- lignes 32-34 : on s'adresse à un serveur qui exige un entête HTTP [Authorization: Basic code] ;
- ligne 35 : l'utilisateur saisira des URL du type [/cors-getAllCategories,/cors-addProduits, ...]. Il faut donc compléter ces URL avec l'URL du serveur de la ligne 6 ;
- ligne 36 : méthode HTTP à utiliser ;
- ligne 37 : le serveur renvoie du jSON. On indique le type [text] comme type de résultat afin de l'afficher tel qu'il a été reçu ;
- ligne 42 : affichage de la réponse texte du serveur ;
- lignes 48-49 : affichage du message d'erreur éventuel ;
- ligne 53 : la méthode [doPost] reçoit un second paramètre qui est la valeur à poster ;
- ligne 61 : pour indiquer que la valeur postée va l'être sous la forme d'une chaîne jSON ;

18.7 Exécution du client

L'application cliente est une application Spring Boot lancée par la classe exécutable [Boot] suivante :




```
1. package spring.cors.client.boot;
2.
3. import org.springframework.boot.SpringApplication;
4.
5. import spring.cors.client.config.WebConfig;
6.
7. public class Boot {
8.
9.   public static void main(String[] args) {
10.       SpringApplication.run(WebConfig.class, args);
11.   }
12. }
```

- ligne 10 : la méthode [SpringApplication.*run*] utilise le fichier de configuration [WebConfig]. La page [client.html] va être déployée sur le serveur Tomcat présent dans le Classpath du projet ;

18.8 L'URL [/getAllCategories]

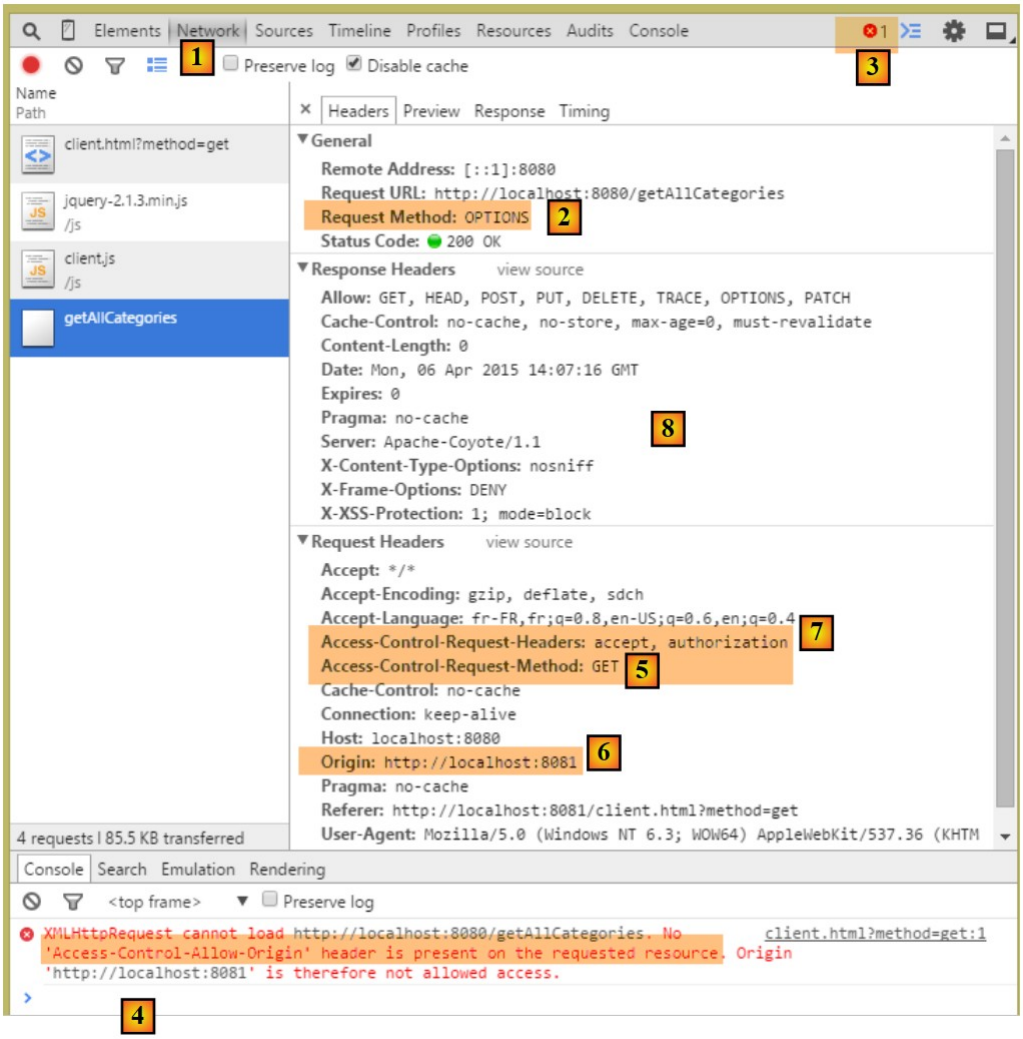
Nous lançons :

- le serveur web / json sur le port 8080 ;
- le client de ce serveur sur le port 8081 ;

puis nous demandons l'URL [http://localhost:8081/client.html] [1] :

- en [2], nous faisons un GET sur l'URL [http://localhost:8080/getAllCategories];

Nous n'obtenons pas de réponse du serveur. Lorsqu'on regarde la console de développement de Chrome (Ctrl-Maj-I) on découvre une erreur :

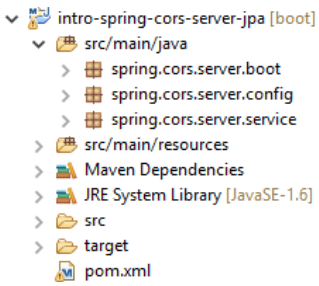


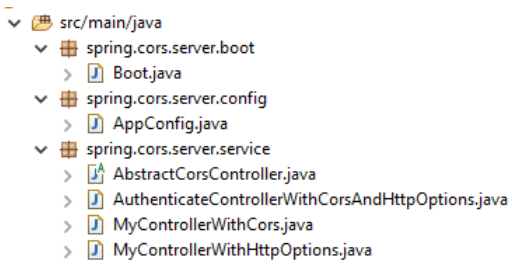
- en [1], on est dans l'onglet [Network] ;
- en [2], on voit que la requête HTTP qui a été faite n'est pas [GET] mais [OPTIONS]. Dans le cas d'une requête inter-domaines, le navigateur vérifie auprès du serveur qu'un certain nombre de conditions sont vérifiées en lui envoyant une requête HTTP [OPTIONS]. En l'occurrence, les requêtes sont celles pointées par les pastilles [5-6] ;
- en [5], le navigateur demande si l'URL cible peut être atteinte avec un GET. L'entête de la requête [Access-Control-Request-Method] demande une réponse avec un entête HTTP [Access-Control-Allow-Methods] indiquant que la méthode demandée est acceptée ;
- en [6], le navigateur envoie l'entête HTTP [Origin: http://localhost:8081]. Cet entête demande une réponse dans un entête HTTP [Access-Control-Allow-Origin] indiquant que l'origine indiquée est acceptée ;
- en [7], le navigateur demande si les entêtes HTTP [accept] et [authorization] sont acceptés. L'entête de la requête [Access-Control-Request-Headers] attend une réponse avec un entête HTTP [Access-Control-Allow-Headers] indiquant que les entêtes demandés sont acceptés ;
- on a une erreur en [3]. En cliquant sur l'icône, on a l'erreur [4] ;
- en [4], le message indique que le serveur n'a pas envoyé l'entête HTTP [Access-Control-Allow-Origin] qui indique si l'origine de la requête est acceptée ;
- en [8], on peut constater que le serveur n'a effectivement pas envoyé cet entête. Du coup le navigateur a refusé de faire la requête HTTP GET demandée initialement ;

Il nous faut modifier le serveur web / jSON.

18.9 Le nouveau service web / json

Nous créons un nouveau projet Maven [intro-spring-cors-server-jpa] :





18.9.1 Configuration Maven

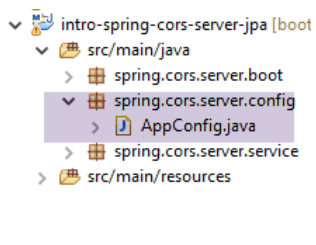
La configuration Maven du nouveau service web est la suivante :

```
1. <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2.   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
3.   <modelVersion>4.0.0</modelVersion>
4.   <groupId>istia.st.cors</groupId>
5.   <artifactId>spring-cors-server-jpa</artifactId>
6.   <version>0.0.1-SNAPSHOT</version>
7.
8.   <name>spring-cors-server-jpa</name>
9.   <description>démo spring cors</description>
10.
11.    <properties>
12.        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
13.        <java.version>1.8</java.version>
14.    </properties>
15.
16.    <parent>
17.        <groupId>org.springframework.boot</groupId>
18.        <artifactId>spring-boot-starter-parent</artifactId>
19.        <version>1.2.7.RELEASE</version>
20.    </parent>
21.
22.    <dependencies>
23.        <dependency>
24.            <groupId>istia.st.spring.security</groupId>
25.            <artifactId>intro-spring-security-server-01</artifactId>
26.            <version>0.0.1-SNAPSHOT</version>
27.        </dependency>
28.    </dependencies>
29.
30.    <!-- plugins -->
31.    <build>
32.        <plugins>
33.            <plugin>
34.                <groupId>org.springframework.boot</groupId>
35.                <artifactId>spring-boot-maven-plugin</artifactId>
36.            </plugin>
37.            <plugin>
38.                <groupId>org.apache.maven.plugins</groupId>
39.                <artifactId>maven-surefire-plugin</artifactId>
40.                <version>2.18.1</version>
41.            </plugin>
42.        </plugins>
43.    </build>
44.
45. </project>
```

- lignes 23-27 : nous récupérons tout l'acquis du travail fait jusqu'à maintenant en nous appuyant sur l'archive du serveur web / json sécurisé ;

18.9.2 Configuration Spring

La classe de configuration [AppConfig] est la suivante :

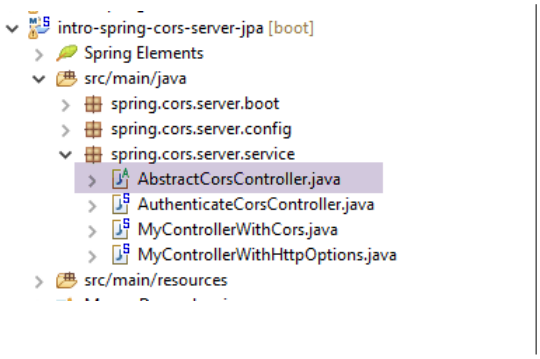



```
1. package spring.cors.server.config;
2.
3. import org.springframework.context.annotation.Bean;
4. import org.springframework.context.annotation.ComponentScan;
5. import org.springframework.context.annotation.Configuration;
6. import org.springframework.context.annotation.Import;
7.
8. import spring.security.config.SecurityConfig;
9.
10.  @Configuration
11.  @ComponentScan(basePackages = { "spring.cors.server.service" })
12.  @Import({ SecurityConfig.class })
13.  public class AppConfig {
14.
15.    // requêtes inter-domaines
16.    @Bean
17.    public boolean isCorsEnabled() {
18.        return true;
19.    }
20.  }
```

- ligne 10 : la classe est une classe de configuration Spring ;
- ligne 11 : d'autres composants Spring sont à chercher dans le paquetage [spring.cors.server.service] ;
- lignes 16-19 : nous créons un composant Spring nommé [isCorsEnabled] qui indique si on accepte ou non les clients étrangers au domaine du serveur ;

18.9.3 La classe [AbstractCorsController]

La classe [AbstractCorsController] qui sera la classe parente de tous les contrôleurs de cette application :



Son code est le suivant :

```
1. package spring.cors.server.service;
2.
3. import javax.servlet.http.HttpServletResponse;
4.
5. import org.springframework.beans.factory.annotation.Autowired;
6.
7. public abstract class AbstractCorsController {
8.
9.   @Autowired
10.    private boolean isCorsEnabled;
11.
12.    // envoi des options au client
13.    public void setHeaders(String origin, HttpServletResponse response) {
14.        // Cors allowed ?
15.        if (!isCorsEnabled || origin == null || !origin.startsWith("http://localhost")) {
16.            return;
17.        }
18.        // on fixe le header CORS
```

```
19.         response.addHeader("Access-Control-Allow-Origin", origin);
20.         // on autorise certains headers
21.         response.addHeader("Access-Control-Allow-Headers", "accept, authorization");
22.         // on autorise le GET
23.         response.addHeader("Access-Control-Allow-Methods", "GET");
24.     }
25. }
```

- ligne 7 : la classe [CorsController] est abstraite car elle est conçue pour être étendue et non instanciée ;
- lignes 13-24 : la méthode [setHeaders] met dans la réponse [HttpServletResponse response] (ligne 13) faite au client, les entêtes HTTP réclamés par les requêtes inter-domaines ;
- ligne 33 : la méthode [/setHeaders] admet pour paramètres :
  - la chaîne [origin] présent dans l'entête HTTP [Origin] des requêtes inter-domaines :

    ```
    Origin:http://localhost:8081
    ```

    Ici le paramètre [origin] de la ligne 13 aurait la valeur [http://localhost:8081]. Au cas, où la demande ne contient pas l'entête HTTP [Origin], on s'arrangera pour avoir [origin==null] ;
  - l'objet [HttpServletResponse response] qui va être renvoyé au client qui a fait la demande ;

  Ces deux paramètres sont injectés par Spring ;

- lignes 15-175 : si l'application est configurée pour accepter les requêtes inter-domaines et si l'émetteur a envoyé l'entête HTTP [Origin] et si cette origine commence par [http://localhost], alors on va accepter la requête inter-domaines, sinon on la rejette ;
- ligne 19 : si le client est dans le domaine [http://localhost:port], on envoie l'entête HTTP :

  ```
  Access-Control-Allow-Origin:  http://localhost:port
  ```

  qui signifie que le serveur accepte l'origine du client ;
- ligne 21 : nous avons signalé deux entêtes HTTP particuliers dans la requête HTTP [OPTIONS] :

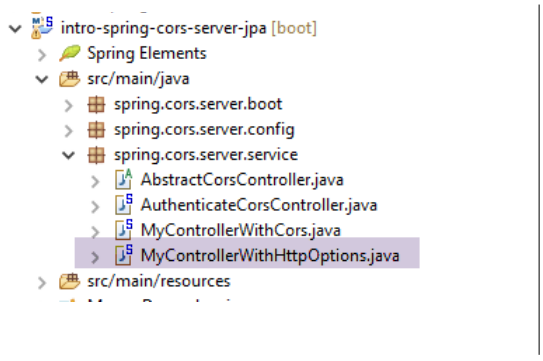
  ```
  Access-Control-Request-Method: GET
  Access-Control-Request-Headers: accept, authorization
  ```

  A l'entête HTTP [Access-Control-Request-X], le serveur répond avec un entête HTTP [Access-Control-Allow-X] dans lequel il indique ce qui est autorisé. Les lignes 20-23 se contentent de reprendre la demande du client pour indiquer qu'elle est acceptée ;

18.9.4 Le contrôleur [MyControllerWithHttpOptions]

Afin de ne pas avoir à modifier le serveur web / jSON non sécurisé [intro-server-webjson-01] étudié au paragraphe 13.5.3, page 227 nous allons créer un nouveau contrôleur qui là où le serveur non sécurisé traite l'URL [/url], le nouveau contrôleur traitera l'URL [/cors-url] et cette URL acceptera les requêtes inter-domaines.

La classe [MyControllerWithHttpOptions] est le contrôleur qui va traiter les requêtes HTTP de type [OPTIONS] :



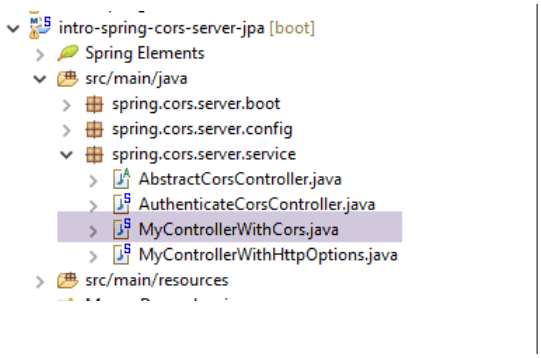

```
1. package spring.cors.server.service;
2.
3. import javax.servlet.http.HttpServletRequest;
4. import javax.servlet.http.HttpServletResponse;
5.
6. import org.springframework.stereotype.Controller;
7. import org.springframework.web.bind.annotation.PathVariable;
8. import org.springframework.web.bind.annotation.RequestHeader;
```

```
9. import org.springframework.web.bind.annotation.RequestMapping;
10.  import org.springframework.web.bind.annotation.RequestMethod;
11.
12.  import com.fasterxml.jackson.core.JsonProcessingException;
13.
14.  @Controller
15.  public class MyControllerWithHttpOptions extends AbstractCorsController {
16.
17.    @RequestMapping(value = "/cors-getAllCategories", method = RequestMethod.OPTIONS)
18.    public void getAllCategories(@RequestHeader(value = "Origin", required = false) String origin,
19.            HttpServletResponse httpServletResponse){
20.        // entêtes CORS
21.        setHeaders(origin, httpServletResponse);
22.    }
23.  ...
```

- ligne 14 : la classe est un contrôleur Spring MVC ;
- ligne 15 : la classe [MyControllerWithHttpOptions] étend la classe [AbstractCorsController] que nous venons de décrire ;
- lignes 17-18 : la méthode [getAllCategories] (ligne 18) traite l'URL ["/cors-getAllCategories"] lorsqu'elle est demandée avec la méthode HTTP [OPTIONS] ;
- ligne 18 : la méthode [getAllCategories] admet deux paramètres :
  - [@RequestHeader(value = "Origin", required = **false**) String origin] pour récupérer la valeur de l'entête HTTP [**Origin**:http://localhost:8081] lorsqu'il est présent. Dans cet exemple, le paramètre [String origin] recevra la valeur [http://localhost:8081]. Cet entête n'est pas obligatoire [required = **false**]. Lorsqu'il n'est pas présent, le paramètre [String origin] aura la valeur *null* ;
  - [HttpServletResponse httpServletResponse] : la réponse qui sera envoyée au client ;
- ligne 21 : on envoie les entêtes HTTP qui permettent les requêtes inter-domaines. La méthode [setHeaders] est définie dans la classe parent [AbstractCorsController] ;

Il est fait ainsi pour toutes les URL exposées par le serveur web / jSON non sécurisé [intro-server-webjson-01] étudié au paragraphe 13.5.3, page 227. Lorsque ce service expose l'URL [/url], la classe [MyControllerWithHttpOptions] ci-dessus expose l'URL [/**cors-**url].

18.9.5 Le contrôleur [MyControllerWithCors]



La classe [MyControllerWithCors] est le contrôleur qui va traiter les requêtes HTTP de type [GET] et [POST] :

```
1. package spring.cors.server.service;
2.
3. import javax.servlet.http.HttpServletRequest;
4. import javax.servlet.http.HttpServletResponse;
5.
6. import org.springframework.beans.factory.annotation.Autowired;
7. import org.springframework.web.bind.annotation.PathVariable;
8. import org.springframework.web.bind.annotation.RequestHeader;
9. import org.springframework.web.bind.annotation.RequestMapping;
10.  import org.springframework.web.bind.annotation.RequestMethod;
11.  import org.springframework.web.bind.annotation.ResponseBody;
12.
13.  import com.fasterxml.jackson.core.JsonProcessingException;
14.
15.  import spring.webjson.service.MyController;
16.
17.  @Controller
18.  public class MyControllerWithCors extends AbstractCorsController {
19.
20.    // dépendances Spring
21.    @Autowired
22.    private MyController myController;
23.
```

```
24.  ...
25.    @RequestMapping(value = "/cors-getAllCategories", method = RequestMethod.GET, produces = "application/json;
     charset=UTF-8")
26.    @ResponseBody
27.    public String getAllCategories(@RequestHeader(value = "Origin", required = false) String origin,
28.            HttpServletResponse httpServletResponse) throws JsonProcessingException {
29.        // réponse
30.        return myController.getAllCategories();
31.    }
32.  ...
```

- ligne 17 : la classe [MyControllerWithCors] est un contrôleur Spring MVC
- ligne 18 : elle étend la classe [AbstractCorsController] ;
- lignes 21-22 : injection du contrôleur [MyController] du serveur web / jSON non sécurisé [intro-server-webjson-01] étudié au paragraphe 13.5.3, page 227 ;
- lignes 25-27 : la méthode [getAllCategories] traite l'URL [/cors-getAllCategories] (ligne 28) lorsqu'elle demandée avec la méthode HTTP [GET] ;
- ligne 26 : le résultat de la méthode [getAllCategories] sera envoyé au client. Ce résultat est un flux jSON (attribut [produces] de la ligne 27 et type [String] du résultat ligne 25) ;
- ligne 27 : la méthode reçoit les mêmes paramètres que la méthode [getAllCategories] du contrôleur [MyControllerWithHttpOptions] que nous venons d'étudier ;
- ligne 30 : on demande à la méthode [myController.getAllCategories()] d'envoyer la réponse ;

Au final, c'est la méthode [myController.getAllCategories()] du serveur non sécurisé qui envoie la réponse. On a simplement enrichi sa réponse avec les entêtes nécessaires aux requêtes inter-domaines.

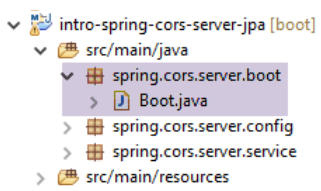
Il est fait ainsi pour toutes les URL exposées par le serveur web / jSON non sécurisé [intro-server-webjson-01] étudié au paragraphe 13.5.3, page 227. Lorsque ce service expose l'URL [/url], la classe [MyControllerWithCors] ci-dessus expose l'URL [/**cors-**url].

Une requête inter-domaine se déroulera de la façon suivante :

- le code JS du client demande l'URL [/cors-url] avec une requête HTTP GET ou un POST ;
- le navigateur qui exécute ce code intercepte cette demande et demande d'abord l'URL [/cors-url] avec une requête HTTP OPTIONS pour vérifier que le service web cible accepte les requêtes inter-domaines ;
- l'une des méthodes du contrôleur [MyControllerWithHttpOptions] envoie les entêtes inter-domaines attendus par le navigateur ;
- le navigateur demande alors l'URL initiale [/cors-url] avec une requête HTTP GET ou un POST ;
- l'une des méthodes du contrôleur [MyControllerWithCors] lui répond alors ;

18.9.6 Tests

La classe de boot du projet [intro-spring-cors-server-jpa] est la suivante :



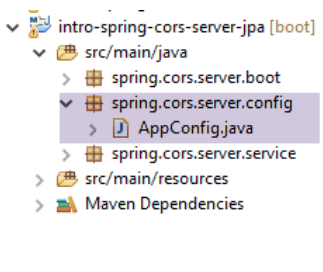

```
1. package spring.cors.server.boot;
2.
3. import org.springframework.boot.SpringApplication;
4.
5. import spring.cors.server.config.AppConfig;
6.
7. public class Boot {
8.
9.   public static void main(String[] args) {
10.       SpringApplication.run(AppConfig.class, args);
11.   }
12. }
```

- ligne 10 : la méthode statique [SpringApplication.*run*] est exécutée avec la configuration Spring [AppConfig]. A cause de cette configuration, le serveur Tomcat embarqué dans les archives du projet est exécuté et l'application web [intro-spring-cors-server-jpa] est déployée dessus. L'application web du serveur non sécurisé [intro-server-webjson-01] qui fait partie des archives du projet est également déployée dessus. Comme le projet [intro-spring-security-server-01] fait également partie des archives, deux types d'URL sont finalement exposées :
  - celles du service web sécurisé : /url ;
  - celles du service web acceptant les requêtes inter-domaines : /cors-url ;

Nous sommes désormais prêts pour de nouveaux tests. Nous lançons la nouvelle version du service web et nous découvrons que le problème reste entier. Rien n'a changé. Si ligne 7 ci-dessous, on met un affichage console, celui-ci n'est jamais affiché montrant par là que la méthode [getAllCategories] de la classe [MyControllerWithHttpOptions] n'est jamais appelée ;

```
1. @Controller
2. public class MyControllerWithHttpOptions extends AbstractCorsController {
3.
4.   @RequestMapping(value = "/cors-getAllCategories", method = RequestMethod.OPTIONS)
5.   public void getAllCategories(@RequestHeader(value = "Origin", required = false) String origin,
6.         HttpServletResponse httpServletResponse){
7.          System.out.println(un_texte) ;
8.    // entêtes CORS
9.    setHeaders(origin, httpServletResponse);
10.   }
11.
```

Après quelques recherches, on découvre que par défaut Spring MVC traite lui-même les commandes HTTP [OPTIONS]. Aussi c'est toujours Spring qui répond et jamais la méthode [getAllCategories] de la ligne 5 ci-dessus. Ce comportement par défaut de Spring MVC peut être changé. Nous modifions la classe [AppConfig] existante :

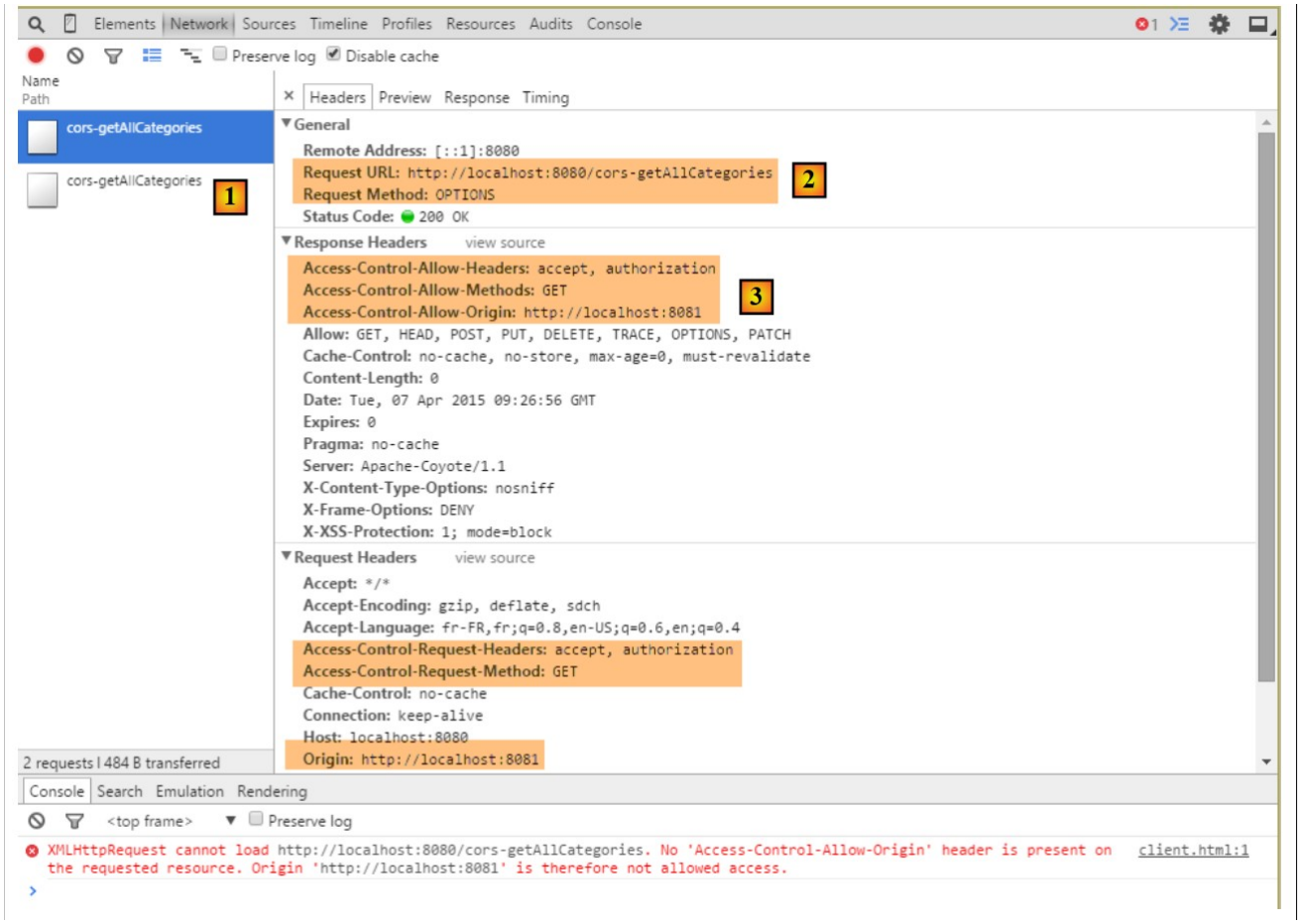



```
1. package spring.cors.server.config;
2.
3. import javax.annotation.PostConstruct;
4.
5. import org.springframework.beans.factory.annotation.Autowired;
6. import org.springframework.context.annotation.Bean;
7. import org.springframework.context.annotation.ComponentScan;
8. import org.springframework.context.annotation.Configuration;
9. import org.springframework.context.annotation.Import;
10.  import org.springframework.web.servlet.DispatcherServlet;
11.
12.  import spring.security.config.SecurityConfig;
13.
14.  @Configuration
15.  @ComponentScan(basePackages = { "spring.cors.server.service" })
16.  @Import({ SecurityConfig.class })
17.  public class AppConfig {
18.
19.    // requêtes inter-domaines
20.    @Bean
21.    public boolean isCorsEnabled() {
22.        return true;
23.    }
24.
25.    @Autowired
26.    private DispatcherServlet dispatcherServlet;
27.
28.    @PostConstruct
29.    public void init() {
30.        // l'application traite elle-même les demandes HTTP [OPTIONS]
31.        dispatcherServlet.setDispatchOptionsRequest(true);
32.    }
33.  }
```

- lignes 25-26 : injection du bean [dispatcherServlet] qui gère les demandes des clients. Ce bean a été défini dans la configuration du serveur web / jSON non sécurisé [intro-server-webjson-01] étudié au paragraphe 13.5.3, page 227 ;

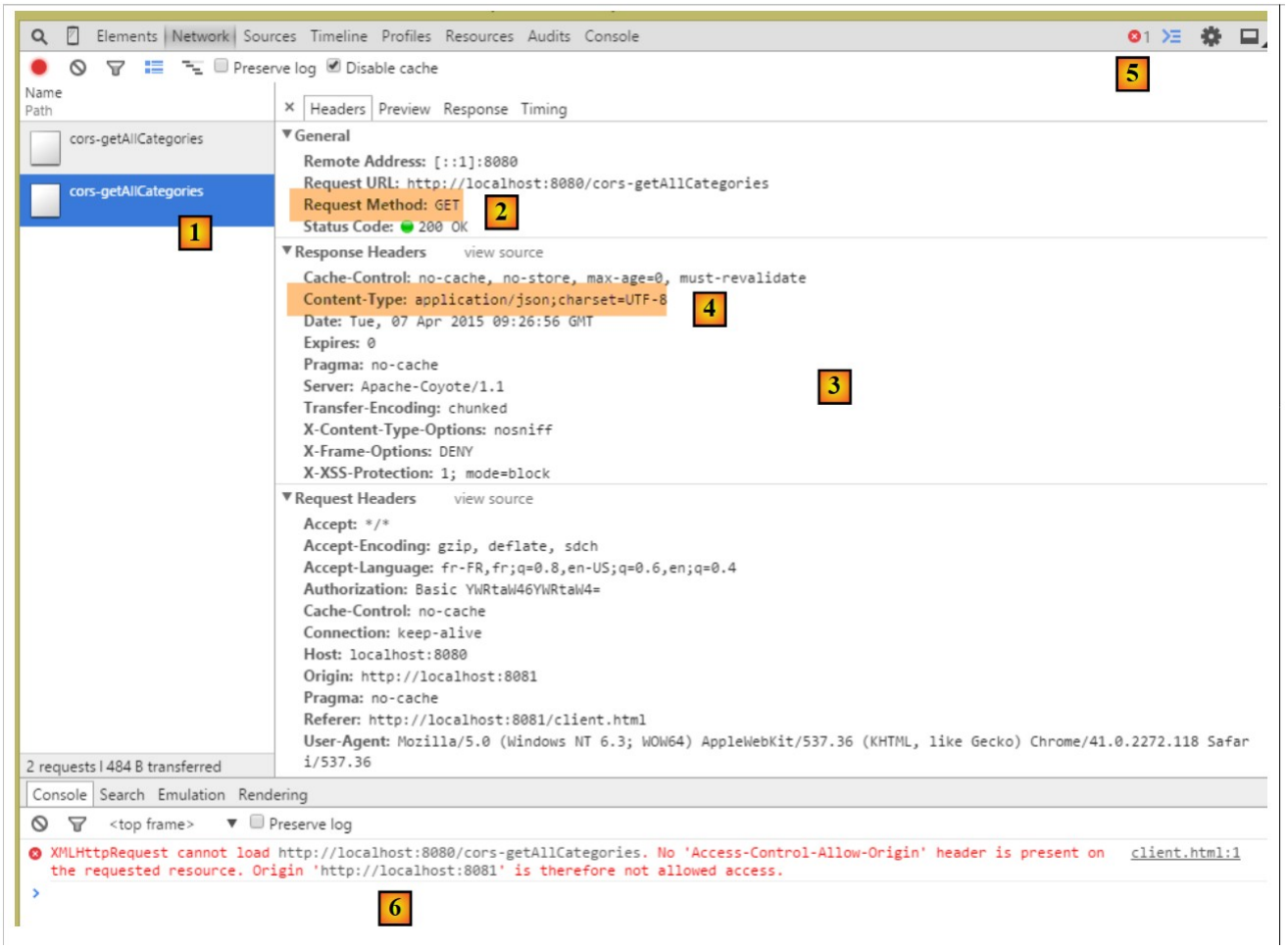
- lignes 28-29 : la méthode [init] (ligne 29) sera exécutée dès que la classe [AppConfig] aura été instanciée et les injections Spring faites. Donc lorsqu'elle s'exécute, le champ de la ligne 26 a été initialisé ;
- ligne 31 : on configure le bean [dispatcherServlet] pour qu'il laisse l'application web traiter elle-même les commandes HTTP [OPTIONS] ;

Nous refaisons les tests avec cette nouvelle configuration. On obtient le résultat suivant :



- en [1], nous voyons qu'il y a deux requêtes HTTP vers l'URL [http://localhost:8080/cors-getAllCategories];
- en [2], la requête [OPTIONS] ;
- en [3], les trois entêtes HTTP que nous venons de configurer dans la réponse du serveur ;

Examinons maintenant la seconde requête :



- en [1], la requête examinée ;
- en [2], c'est la requête GET. Grâce à la première requête [OPTIONS], le navigateur a reçu les informations qu'il demandait. Il réalise maintenant la requête [GET] demandée initialement ;
- en [3], la réponse du serveur ;
- en [4], le serveur envoie du jSON ;
- en [5], une erreur s'est produite ;
- en [6], le message d'erreur ;

Il est plus difficile d'expliquer ce qui s'est passé ici. La réponse [3] du serveur est normale [HTTP/1.1 200 OK]. On devrait donc avoir le document demandé. Il est possible que le serveur ait bien envoyé le document mais que c'est le navigateur qui empêche son utilisation parce qu'il veut que pour la requête GET également, la réponse comporte l'entête HTTP [Access-Control-Allow-Origin:http://localhost:8081].

Nous modifions alors le contrôleur [MyControllerWithCors] pour que lui aussi envoie les entêtes nécessaires aux requêtes inter-domaines :

```
1.  @RequestMapping(value = "/cors-getAllCategories", method = RequestMethod.GET, produces = "application/json;
    charset=UTF-8")
2.  @ResponseBody
3.  public String getAllCategories(@RequestHeader(value = "Origin", required = false) String origin,
4.          HttpServletResponse httpServletResponse) throws JsonProcessingException {
5.    // entêtes CORS
6.    setHeaders(origin, httpServletResponse);
7.    // réponse
8.    return myController.getAllCategories();
9. }
```

- ligne 6 : les entêtes nécessaires aux requêtes inter-domaines sont inclus dans la réponse ;

Après cette modification, les résultats sont les suivants :

Nous avons bien obtenu la liste des catégories.

18.10 Les autres URL [GET]

Dans les contrôleurs [MyControllerWithCors, MyControllerWithHttpOptions], le code des actions qui traitent les URL demandées avec un [GET] suit le modèle des actions qui ont traité précédemment l'URL [/cors-getAllCategories]. Le lecteur peut vérifier le code dans les exemples livrés avec ce document. Voici un exemple pour l'URL [/cors-getAllProduits] :

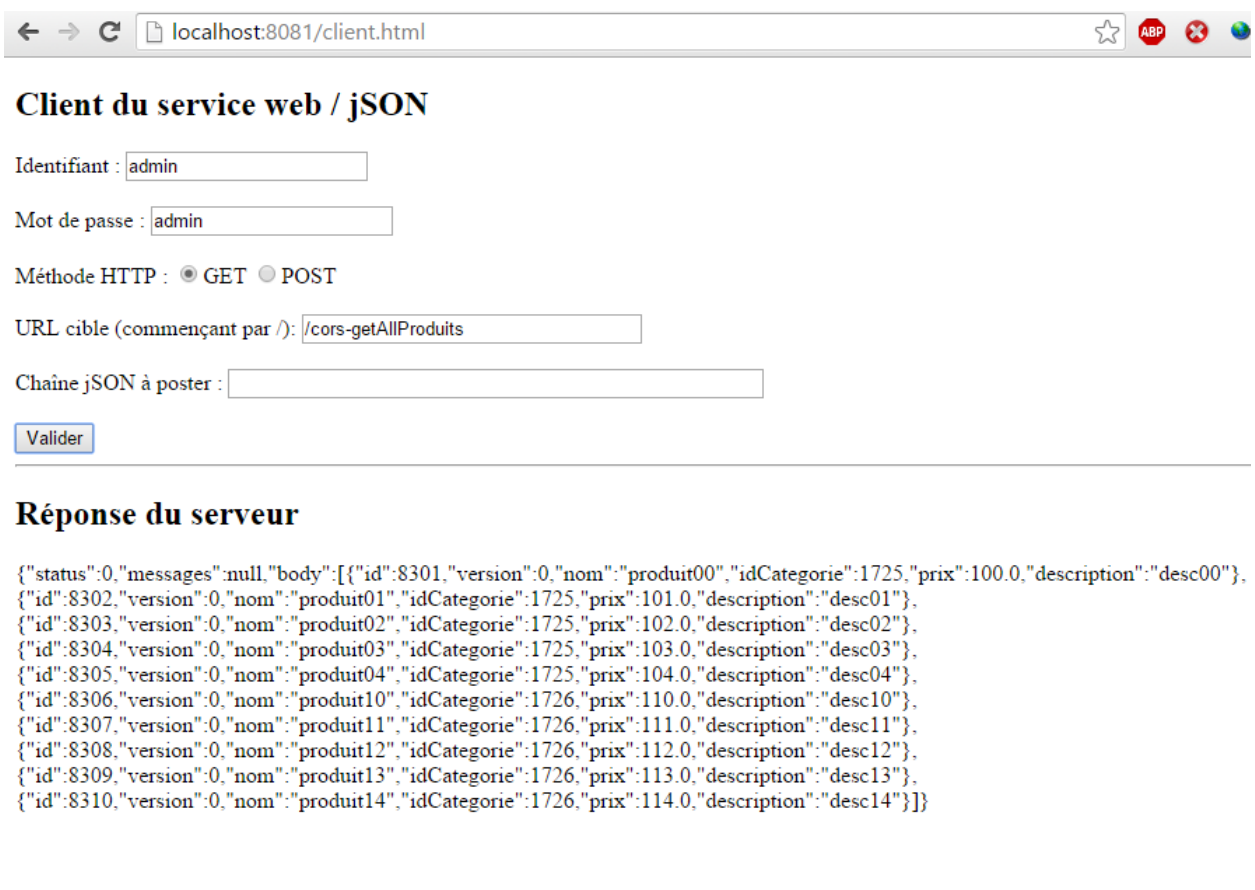
dans [MyControllerWithHttpOptions]

```
1.  @RequestMapping(value = "/cors-getAllProduits", method = RequestMethod.OPTIONS)
2.  public void getAllProduits(@RequestHeader(value = "Origin", required = false) String origin,
3.        HttpServletResponse httpServletResponse) {
4.    // entêtes CORS
5.    setHeaders(origin, httpServletResponse);
6. }
```

dans [MyControllerWithCors]

```
1.  @RequestMapping(value = "/cors-getAllProduits", method = RequestMethod.GET, produces = "application/json;
    charset=UTF-8")
2.  @ResponseBody
3.  public String getAllProduits(@RequestHeader(value = "Origin", required = false) String origin,
4.        HttpServletResponse httpServletResponse) throws JsonProcessingException {
5.    // entêtes CORS
6.    setHeaders(origin, httpServletResponse);
```

```
7.     // réponse
8.     return myController.getAllProduits();
9. }
```



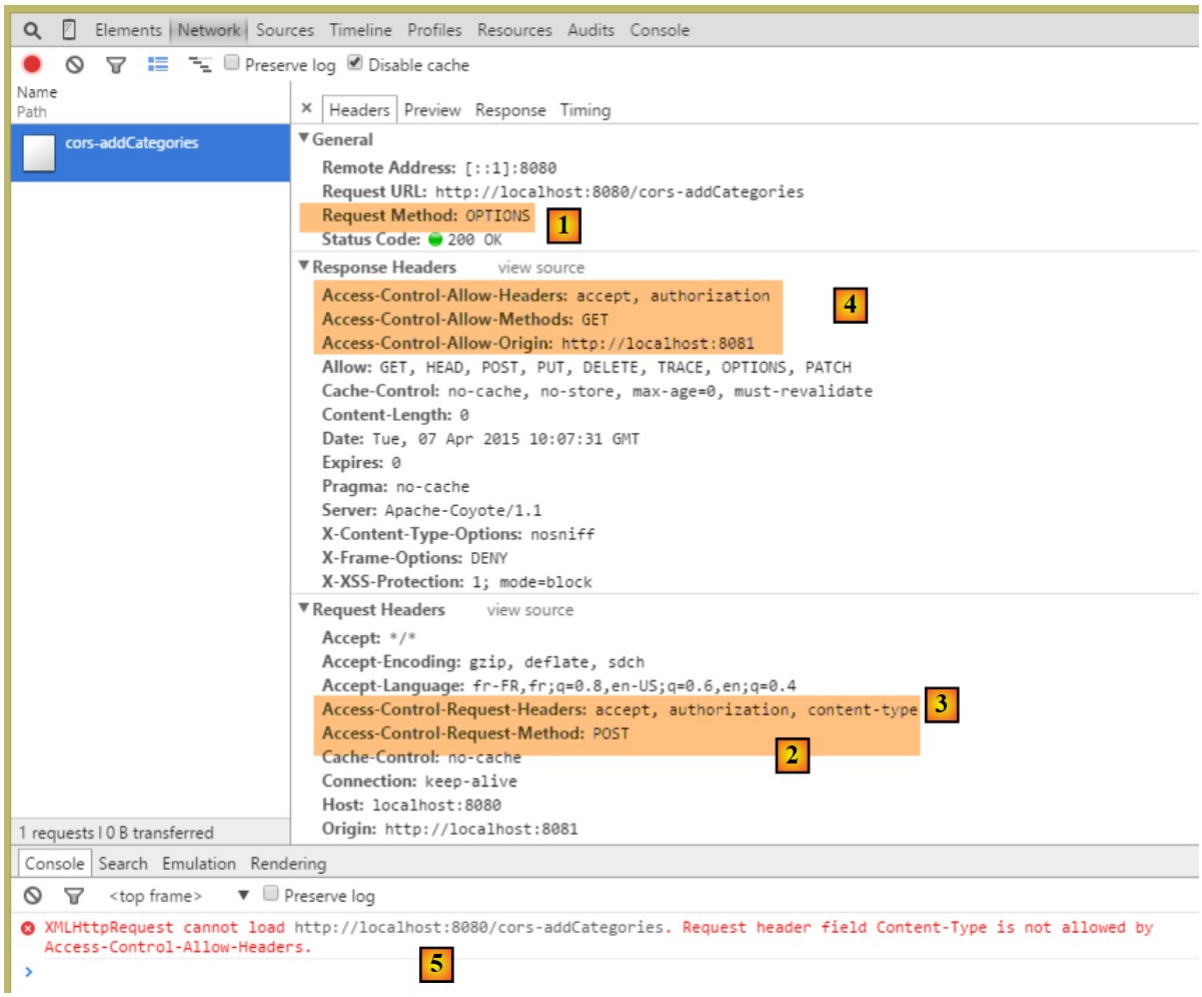
18.11 Les URL [POST]

Examinons le cas suivant :

- on fait un POST [1] vers l'URL [2] ;
- en [3], la valeur postée. Il s'agit d'une chaîne jSON ;

- au total, on cherche à créer une catégorie appelée [categorie2] ;

Nous ne modifions pour l'instant aucun code. Le résultat obtenu est le suivant :



- en [1], comme pour les requêtes [GET], une requête [OPTIONS] est faite par le navigateur ;
- en [2], il demande une autorisation d'accès pour une requête [POST]. Auparavant c'était [GET] ;
- en [3], il demande une autorisation d'envoyer les entêtes HTTP [accept, authorization, content-type]. Auparavant, on avait seulement les deux premiers entêtes ;
- en [4], le service web ne donne pas toutes les autorisations demandées ce qui provoque l'erreur [5] ;

Nous modifions la méthode [AbstractController.sendHeaders] de la façon suivante :

```
1. package spring.cors.server.service;
2.
3. import javax.servlet.http.HttpServletResponse;
4.
5. import org.springframework.beans.factory.annotation.Autowired;
6.
7. public abstract class AbstractCorsController {
8.
9.   @Autowired
10.   private boolean isCorsEnabled;
11.
12.   // envoi des options au client
13.   public void setHeaders(String origin, HttpServletResponse response) {
14.       // Cors allowed ?
15.       if (!isCorsEnabled || origin == null || !origin.startsWith("http://localhost")) {
16.           return;
17.       }
18.       // on fixe le header CORS
19.       response.addHeader("Access-Control-Allow-Origin", origin);
20.       // on autorise certains headers
```

```
21.       response.addHeader("Access-Control-Allow-Headers", "accept, authorization, content-type");
22.       // on autorise le GET et le POST
23.       response.addHeader("Access-Control-Allow-Methods", "GET, POST");
24.    }
25. }
```

- ligne 21 : on a ajouté l'entête HTTP [Content-Type] (la casse n'a pas d'importance) ;
- ligne 23 : on a ajouté la méthode HTTP [POST] ;

Ceci fait les méthodes [POST] sont traitées de la même façon que les requêtes [GET]. Voici l'exemple de l'URL [/cors-addArticles] :

dans [MyControllerWithCors]

```
1.  @RequestMapping(value = "/cors-addCategories", method = RequestMethod.POST, consumes = "application/json;
    charset=UTF-8", produces = "application/json; charset=UTF-8")
2.  @ResponseBody
3.  public String addCategories(HttpServletRequest request,
4.        @RequestHeader(value = "Origin", required = false) String origin, HttpServletResponse httpServletResponse)
5.              throws JsonProcessingException {
6.    // entêtes CORS
7.    setHeaders(origin, httpServletResponse);
8.    // réponse
9.    return myController.addCategories(request);
10. }
```

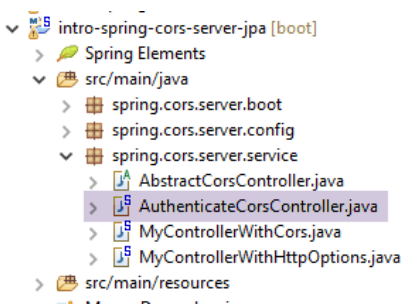
dans [MyControllerWithHttpOptions]

```
1.  @RequestMapping(value = "/cors-addCategories", method = RequestMethod.OPTIONS)
2.  public void addCategories(HttpServletRequest request,
3.        @RequestHeader(value = "Origin", required = false) String origin, HttpServletResponse httpServletResponse)
4.              throws JsonProcessingException {
5.    // entêtes CORS
6.    setHeaders(origin, httpServletResponse);
7. }
```

Le résultat obtenu est le suivant :

La catégorie [categorie2] a bien été ajoutée dans la base de données. Le SGBD lui a attribué la clé primaire 1729.

18.12 Le contrôleur [AuthenticateCorsController]



Le contrôleur [AuthenticateCorsController] est là pour fournir l'URL [/cors-authenticate] qui permet d'appeler l'URL [/authenticate] déjà existante, avec une requête inter-domaines. Son code est le suivant :

```
1. package spring.cors.server.service;
2.
3. import javax.servlet.http.HttpServletResponse;
4.
5. import org.springframework.beans.factory.annotation.Autowired;
6. import org.springframework.stereotype.Controller;
7. import org.springframework.web.bind.annotation.RequestHeader;
8. import org.springframework.web.bind.annotation.RequestMapping;
9. import org.springframework.web.bind.annotation.RequestMethod;
10.  import org.springframework.web.bind.annotation.ResponseBody;
11.
12.  import com.fasterxml.jackson.core.JsonProcessingException;
13.
14.  import spring.security.service.AuthenticateController;
15.
16.  @Controller
17.  public class AuthenticateCorsController extends AbstractCorsController {
18.    @Autowired
19.    private AuthenticateController authenticateController;
20.
21.    @RequestMapping(value = "/cors-authenticate", method = RequestMethod.GET)
22.    @ResponseBody
23.    public String authenticate(@RequestHeader(value = "Origin", required = false) String origin,
24.            HttpServletResponse response) throws JsonProcessingException {
25.        // entêtes CORS
26.        setHeaders(origin, response);
27.        // méthode d'origine
28.        return authenticateController.authenticate();
29.    }
30.
31.    @RequestMapping(value = "/cors-authenticate", method = RequestMethod.OPTIONS)
32.    public void corsAuthenticate(@RequestHeader(value = "Origin", required = false) String origin,
33.            HttpServletResponse response) {
34.        // entêtes CORS
35.        setHeaders(origin, response);
36.    }
37.
38.  }
```

Voici deux exemples :

- les réponses affichées le sont par le code jS suivant :

```
1. function doGet(url) {
2.   // on fait un appel Ajax à la main
3.   $.ajax({
4.     headers : {
5.         'Authorization':'Basic '+authorizationCode
6.     },
7.     url : baseUrl + url,
8.     type : 'GET',
9.     dataType : 'text',
10.        beforeSend : function() {
11.        },
12.        success : function(data) {
13.            // résultat texte
14.            response.text(data);
15.        },
16.        complete : function() {
17.        },
18.        error : function(jqXHR) {
19.            // erreur système
20.            response.text(JSON.stringify(jqXHR.statusCode()));
21.        }
22.   })
23. }
```

- la réponse [1] est affichée par la ligne 14 de la fonction [success] ;
- la réponse [2] est affichée par la ligne 20 de la fonction [error]. La fonction [JSON.stringify] crée la chaîne jSON de l'objet [jqXHR.statusCode()] qui est l'objet encapsulant l'erreur qui s'est produite. Cet objet donne peu d'informations. Il est possible d'exploiter d'autres méthodes de l'objet [jqXHR] pour avoir, par exemple, les entêtes HTTP renvoyés par le serveur ;

18.13 Conclusion

Notre application supporte désormais les requêtes inter-domaines. Celles-ci peuvent être autorisées ou non par configuration dans la classe [AppConfig] :

```
1. @ComponentScan(basePackages = { "spring.cors.server.service" })
2. @Import({ SecurityConfig.class })
3. public class AppConfig {
4.
5.   // requêtes inter-domaines
6.   @Bean
7.   public boolean isCorsEnabled() {
8.     return true;
9.   }
10.
```

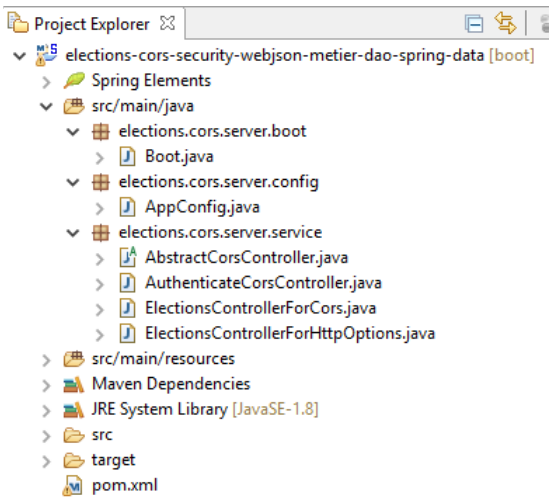
```
11.    @Autowired
12.    private DispatcherServlet dispatcherServlet;
13.
14.    @PostConstruct
15.    public void init() {
16.        // l'application traite elle-même les demandes HTTP [OPTIONS]
17.        dispatcherServlet.setDispatchOptionsRequest(true);
18.    }
19.  }
```

19 [TD] Elections avec des requêtes inter-domaines

Mots clés : CORS (Cross-Origin Resource Sharing).

Nous allons ici créer un nouveau projet pour que le service web sécurisé des élections accepte des requêtes inter-domaines.

Le projet Eclipse est le suivant :



Travail à faire : en suivant le processus décrit au paragraphe 18.9, page 361, construisez ce projet.

Lorsque ce projet est construit, voici ce qu'on peut faire avec le client HTML décrit au paragraphe 18.2, page 350. Au départ des requêtes, la base de données [dbelections] est la suivante :

id	version	sap	seuilelectoral
1	0	6	0.05

id	version	nom	voix	sieges	elimine
1	8	A	0	0	0
2	12	B	0	0	0
3	13	C	0	0	0
4	12	D	0	0	0
5	13	E	0	0	0
6	12	F	0	0	0
7	12	G	0	0	0

Client du service web / jSON

Identifiant : admin

Mot de passe : admin

Méthode HTTP : GET POST

URL cible (commençant par /): /cors-getElectionsConfig

Chaîne jSON à poster :

Valider

Réponse du serveur

{"status":0,"messages":null,"body":{"id":null,"version":null,"nbSiegesAPourvoir":6,"seuilElectoral":0.05}}

Client du service web / jSON

Identifiant : admin

Mot de passe : admin

Méthode HTTP : GET POST

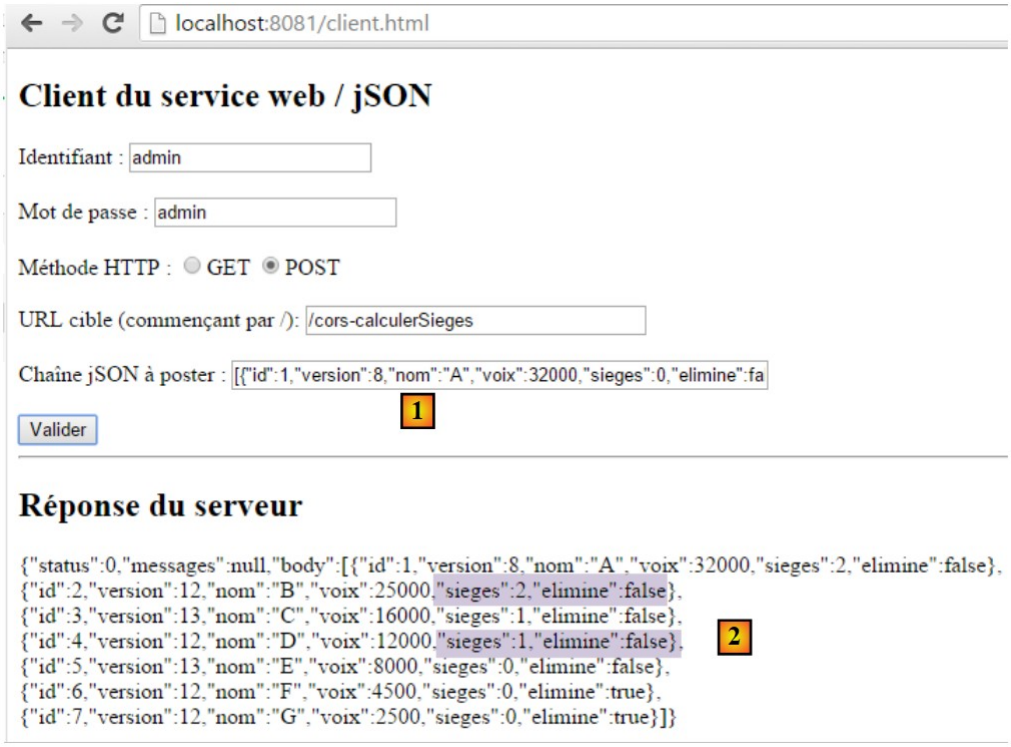
URL cible (commençant par /): /cors-getListesElectorales

Chaîne jSON à poster :

Valider

Réponse du serveur

{"status":0,"messages":null,"body":[{"id":1,"version":8,"nom":"A","voix":0,"sieges":0,"elimine":false},
{"id":2,"version":12,"nom":"B","voix":0,"sieges":0,"elimine":false},{"id":3,"version":13,"nom":"C","voix":0,"sieges":0,"elimine":false},
{"id":4,"version":12,"nom":"D","voix":0,"sieges":0,"elimine":false},{"id":5,"version":13,"nom":"E","voix":0,"sieges":0,"elimine":false},
{"id":6,"version":12,"nom":"F","voix":0,"sieges":0,"elimine":false},{"id":7,"version":12,"nom":"G","voix":0,"sieges":0,"elimine":false}]}



En [1], la valeur jSON postée est la suivante :

```
[{"id":1,"version":8,"nom":"A","voix":32000,"sieges":0,"elimine":false},
{"id":2,"version":12,"nom":"B","voix":25000,"sieges":0,"elimine":false},
{"id":3,"version":13,"nom":"C","voix":16000,"sieges":0,"elimine":false},
{"id":4,"version":12,"nom":"D","voix":12000,"sieges":0,"elimine":false},
{"id":5,"version":13,"nom":"E","voix":8000,"sieges":0,"elimine":false},
{"id":6,"version":12,"nom":"F","voix":4500,"sieges":0,"elimine":false},
{"id":7,"version":12,"nom":"G","voix":2500,"sieges":0,"elimine":false}]
```

En [2], les champs [sieges] et [elimine] ont été calculés. Nous copions le champ [body] de [2] en [3] ci-dessous :

La base de données [dbelections] est alors la suivante :

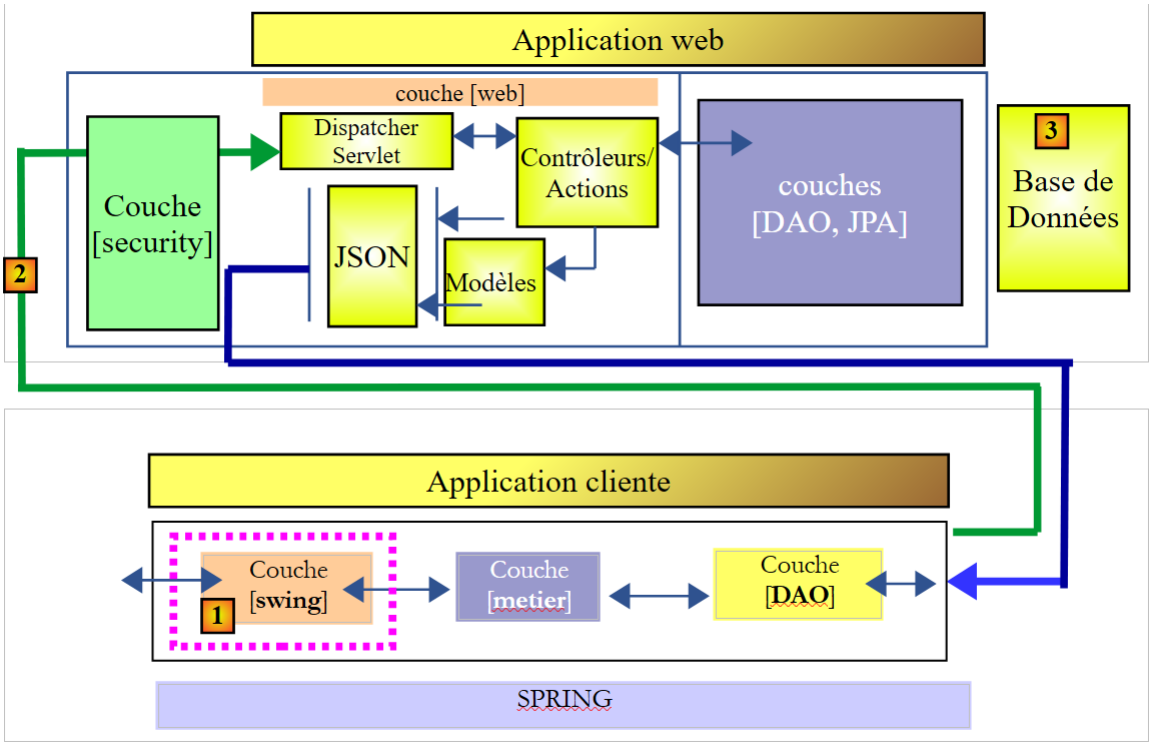
id	version	sap	seuilelectoral
1	0	6	0.05

id	version	nom	voix	sieges	elimine
1	9	A	32000	2	0
2	13	B	25000	2	0
3	14	C	16000	1	0
4	13	D	12000	1	0
5	14	E	8000	0	0
6	13	F	4500	0	1
7	13	G	2500	0	1

20 Programmation asynchrone avec RxJava

Document à lire : [Introduction à RxJava. Application aux environnements Swing et Android.]

Dans ce chapitre, nous revenons sur le chapitre 17.6, page 340 où nous avions construit une application client / serveur avec l'architecture suivante :



Certaines actions de l'utilisateur sur l'interface Swing en [1] déclenchent des actions jusque dans la base de données en [3] au-travers d'un réseau HTTP [2]. A cause de celui-ci, la réponse à l'action de l'utilisateur peut être plus ou moins longue à venir. Ce serait bien de pouvoir mettre un indicateur d'attente sur l'interface utilisateur avec une option d'annulation de l'opération lancée si celle-ci venait à être trop longue. Dans le chapitre 17.6, page 340, chaque action de l'utilisateur nécessitant d'échanger des informations avec le serveur est synchrone. Le gestionnaire d'événement exécuté par le code n'est terminé que lorsque la réponse est reçue. Pendant tout ce temps, l'interface graphique est gelée : elle ne répond pas aux nouvelles actions de l'utilisateur. Celles-ci sont simplement mises en file d'attente pour être traitées lorsque le gestionnaire d'événement qui s'exécute actuellement soit terminé. Ainsi, si on faisait apparaître un bouton d'annulation, l'utilisateur pourrait cliquer dessus mais il ne se passerait rien tant que l'opération en cours ne serait pas terminée. Le bouton d'annulation n'aurait alors aucun intérêt.

Pour que le clic sur le bouton d'annulation soit suivi d'effet, il faut que l'opération en cours soit terminée. Pour cela, elle doit lancer l'opération potentiellement longue de façon asynchrone :

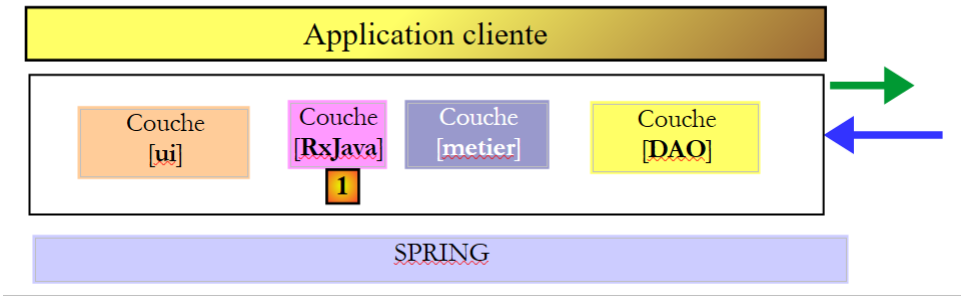
- le gestionnaire d'événement lance l'opération longue mais n'attend pas son résultat et rend la main au thread de l'UI qui gère les événements de l'interface graphique. L'opération longue est lancée sur un thread différent de celui de l'UI ce qui ne bloque pas ce dernier ;
- si l'utilisateur clique sur le bouton d'annulation avant la fin de l'opération longue, le thread de l'UI qui est inoccupé peut traiter cet événement. On peut alors abandonner l'opération longue en ignorant son résultat ;
- si l'opération longue n'a pas été annulée, l'arrivée de la réponse va provoquer un événement dans le thread de l'UI. Celui-ci, s'il est inoccupé, va alors exécuter le code lié à cet événement qui va exploiter la réponse ;

L'interface utilisateur va fonctionner comme précédemment. Si les temps de réponse du serveur sont rapides, l'utilisateur ne verra pas la différence. S'ils sont perceptibles, l'utilisateur verra un bouton d'annulation apparaître et aura la possibilté d'interrompre l'opération en cours.

La bibliothèque [Rx] permet de faire de la programmation asynchrone. Son grand intérêt réside dans le fait qu'elle a été portée dans de nombreux environnements (Java, .NET, JS, ...) et que sa maîtrise dans un environnement peut être transposée facilement dans un autre environnement. Nous allons nous appuyer ici sur le chapitre 2 du document [Introduction à RxJava. Application aux

environnements Swing et Android]. Le lecteur est invité à le lire. Dans la suite, nous reprenons du code issu des exemples de ce chapitre.

Nous allons faire évoluer l'architecture de l'application de la façon suivante :



- en [1], nous intercalons une couche [RxJava] entre la couche [swing] et la couche [métier]. Les méthodes de celle-ci vont désormais être appelées de façon asynchrone ;

Nous allons procéder en plusieurs étapes :

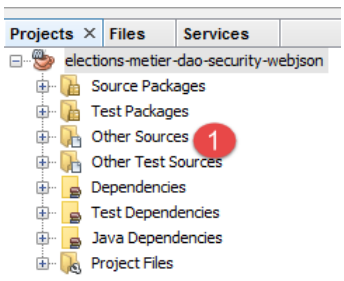
- **étape 1** : la couche [metier, DAO] présente pour l'instant une interface synchrone à la couche [ui]. Nous allons la transformer en une couche asynchrone [RxJava, metier, DAO] ;
- **étape 2** : nous porterons l'application console synchrone en une application toujours synchrone mais utilisant l'interface asynchrone [RxJava, metier, DAO] ;
- **étape 3** : nous porterons l'application swing synchrone en une application swing asynchrone ;

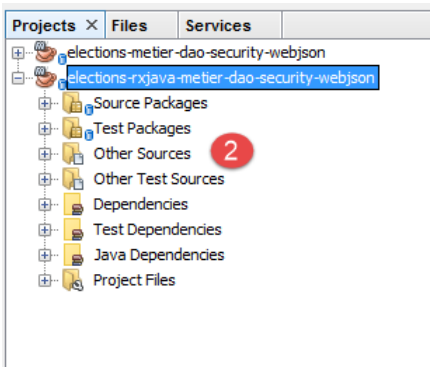
20.1 étape 1

Nous transformons la couche synchrone actuelle [metier, DAO] en une couche asynchrone [RxJava, metier, DAO].

20.1.1 Création

Nous partons du projet Maven du chapitre 17.4, page 330 que nous ouvrons avec **Netbeans** :





Nous dupliquons ce projet [1] (copy / paste) dans un nouveau projet [elections-rxjava-metier-dao-security-webjson] [2].

20.1.2 Configuration Maven

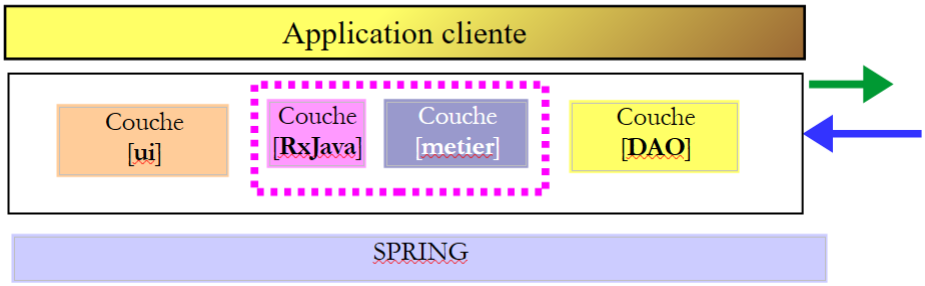
Nous faisons évoluer le fichier [pom.xml] du nouveau projet pour ajouter la dépendance sur la bibliothèque [RxJava] :

```
1. <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2.          xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
3.   <modelVersion>4.0.0</modelVersion>
4.   <groupId>istia.st.elections</groupId>
5.   <artifactId>elections-metier-dao-security-rxjava-webjson</artifactId>
```

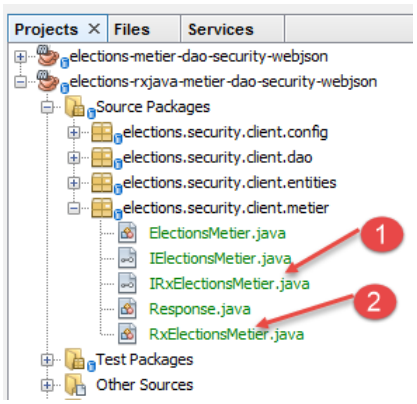
```
6.  <version>0.0.1-SNAPSHOT</version>
7.  <description>Client jUnit du serveur web / jSON</description>
8.  <name>elections-metier-dao-security-rxjava-webjson</name>
9.
10.   <properties>
11.     <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
12.     <java.version>1.8</java.version>
13.   </properties>
14.
15.   <parent>
16.     <groupId>org.springframework.boot</groupId>
17.     <artifactId>spring-boot-starter-parent</artifactId>
18.     <version>1.2.7.RELEASE</version>
19.   </parent>
20.
21.   <dependencies>
22.     <!-- Spring -->
23.     <dependency>
24.       <groupId>org.springframework</groupId>
25.       <artifactId>spring-web</artifactId>
26.     </dependency>
27.     <!-- librairie jSON utilisée par Spring -->
28.     <dependency>
29.       <groupId>com.fasterxml.jackson.core</groupId>
30.       <artifactId>jackson-core</artifactId>
31.     </dependency>
32.     <dependency>
33.       <groupId>com.fasterxml.jackson.core</groupId>
34.       <artifactId>jackson-databind</artifactId>
35.     </dependency>
36.     <!-- composant utilisé par Spring RestTemplate -->
37.     <dependency>
38.       <groupId>org.apache.httpcomponents</groupId>
39.       <artifactId>httpclient</artifactId>
40.     </dependency>
41.     <!-- Google Guava -->
42.     <dependency>
43.       <groupId>com.google.guava</groupId>
44.       <artifactId>guava</artifactId>
45.       <version>16.0.1</version>
46.       <scope>test</scope>
47.     </dependency>
48.     <!-- bibliothèque de logs -->
49.     <dependency>
50.       <groupId>org.springframework.boot</groupId>
51.       <artifactId>spring-boot-starter-logging</artifactId>
52.     </dependency>
53.     <!-- Spring Boot Test -->
54.     <dependency>
55.       <groupId>org.springframework.boot</groupId>
56.       <artifactId>spring-boot-starter-test</artifactId>
57.       <scope>test</scope>
58.     </dependency>
59.     <!-- Spring Boot -->
60.     <dependency>
61.       <groupId>org.springframework.boot</groupId>
62.       <artifactId>spring-boot</artifactId>
63.       <scope>test</scope>
64.     </dependency>
65.     <!-- https://mvnrepository.com/artifact/io.reactivex/rxjava -->
66.     <dependency>
67.       <groupId>io.reactivex</groupId>
68.       <artifactId>rxjava</artifactId>
69.       <version>1.2.0</version>
70.     </dependency>
71.
72.   </dependencies>
73.
74.   <build>
75.     <plugins>
76.       <plugin>
77.         <groupId>org.apache.maven.plugins</groupId>
78.         <artifactId>maven-surefire-plugin</artifactId>
79.         <version>2.18.1</version>
80.       </plugin>
81.     </plugins>
82.   </build>
83. </project>
```

- lignes 65-70 : nous avons ajouté la dépendance sur la bibliothèque RxJava ;

20.1.3 Implémentation asynchrone de la couche [métier]



Pour implémenter la couche [RxJava, métier], nous ajoutons une interface asynchrone [IRxElectionsMetier] [1] et son implémentation [RxElectionsMetier] [2] au projet :



L'interface [IRxElectionsMetier] est l'interface asynchrone de la couche [RxJava, métier]. Son code est le suivant :

```
1. package elections.security.client.metier;
2.
3. import elections.security.client.entities.ListeElectorale;
4. import elections.security.client.entities.User;
5. import rx.Observable;
6.
7. public interface IRxElectionsMetier {
8.
9.   // authentification
10.    Observable<Void> authenticate(User user);
11.
12.    // obtenir les listes en compétition
13.    Observable<ListeElectorale[]> getListesElectorales(User user);
14.
15.    // le nombre de sièges à pourvoir
16.    Observable<Integer> getNbSiegesAPourvoir(User user);
17.
18.    // le seuil électoral
19.    Observable<Double> getSeuilElectoral(User user);
20.
21.    // l'enregistrement des résultats
22.    Observable<Void> recordResultats(User user, ListeElectorale[] listesElectorales);
23.
24.    // le calcul des sièges
25.    Observable<ListeElectorale[]> calculerSieges(User user, ListeElectorale[] listesElectorales);
26.
27. }
```

L'interface [IRxElectionsMetier] reprend les méthodes de l'interface [IElectionsMetier] mais là où une méthode M de l'interface [IElectionsMetier] rendait un résultat de type **T**, la méthode M de l'interface [IRxElectionsMetier] rend un résultat de type **Observable<T>**. Le type [Observable] est fourni par la bibliothèque RxJava. Un type Observable<T> fournit la méthode [subscribe] qui va obtenir le type T de façon asynchrone. Sont associés à cette méthode trois événements :

- **onSuccess**(T result) qui avertit qu'un résultat de type T est disponible. L'opération asynchrone peut fournir plusieurs résultats ;
- **onError**(Throwable th) qui avertit que l'opération asynchrone a rencontré une erreur ;

- **onCompleted**() qui avertit que l'opération asynchrone est terminée ;

Tant que la méthode [Observable.subscribe] n'est pas appelée, l'opération asynchrone liée à l'observable n'est pas lancée. Le code qui appelle une méthode M de l'interface [IRxElectionsMetier] n'obtient pas le résultat T attendu, mais un type Observable<T> qui lui permettra ultérieurement d'obtenir le résultat T en appelant la méthode [Observable.subscribe].

L'implémentation [RxElectionsMetier] de l'interface [IRxElectionsMetier] est la suivante :

```
1. package elections.security.client.metier;
2.
3. import elections.security.client.entities.ListeElectorale;
4. import elections.security.client.entities.User;
5. import org.springframework.beans.factory.annotation.Autowired;
6. import org.springframework.stereotype.Component;
7. import rx.Observable;
8.
9. @Component
10.  public class RxElectionsMetier implements IRxElectionsMetier {
11.
12.    @Autowired
13.    private IElectionsMetier metier;
14.
15.    @Override
16.    public Observable<Void> authenticate(User user) {
17.      ...
18.    }
19.
20.    @Override
21.    public Observable<ListeElectorale[]> getListesElectorales(User user) {
22.      return Observable.create(subscriber -> {
23.        try {
24.          // appel méthode synchrone puis réponse au souscripteur
25.          subscriber.onNext(metier.getListesElectorales(user));
26.          // on signale la fin de l'observable
27.          subscriber.onCompleted();
28.        } catch (Exception e) {
29.          // on fait suivre l'exception
30.          subscriber.onError(e);
31.        }
32.      });
33.    }
34.
35.    @Override
36.    public Observable<Integer> getNbSiegesAPourvoir(User user) {
37.      ...
38.    }
39.
40.    @Override
41.    public Observable<Double> getSeuilElectoral(User user) {
42.      ...
43.    }
44.
45.    @Override
46.    public Observable<Void> recordResultats(User user, ListeElectorale[] listesElectorales) {
47.      ...
48.    }
49.
50.    @Override
51.    public Observable<ListeElectorale[]> calculerSieges(User user, ListeElectorale[] listesElectorales) {
52.      ...
53.    }
54.  }
```

- lignes 12-13 : injection Spring de la couche métier **synchrone** ;
- lignes 20-34 : nous allons commenter la méthode [getListesElectorales] qui au lieu de rendre un type [ListeElectorale[]] rend un type [Observable<ListeElectorale[]>] ;
- lignes 22-32 : la méthode statique [Observable.create] permet de créer un Observable à partir d'un type [**Subscriber**]. Le type [Subscriber] représente un abonné aux flux de résultats produits par le processus observé (l'Observable). Il fournit trois méthodes :
  - [Subscriber.**onNext**] (ligne 25) pour recevoir un résultat du processus observé ;
  - [Subscriber.**onError**] (ligne 30) pour recevoir une exception du processus observé. Après une exception, le type [Observable] n'émet plus de résultats ;
  - [Subscriber.**onCompleted**] (ligne 27) pour recevoir le signal de fin d'émission du processus observé. Ici, le processus observé n'émet qu'un élément. On remarquera ici que ce signal n'est pas émis s'il se produit une exception. C'est le comportement par défaut des Observables : l'émission d'une exception signale également la fin des émissions. Les souscripteurs le savent ;

- lignes 22-34 : la méthode [Observable.create] admet pour paramètre un type [Observable.OnSubscribe]. Ce type est une **interface fonctionnelle**. Cette notion a été introduite avec Java 8 et désigne une interface ayant une unique méthode. Ici, la méthode unique de l'interface [Observable.OnSubscribe] est la suivante :

```
T call(Subscriber<T> subscriber)
```

Pour implémenter une interface fonctionnelle de méthode unique **m(param1, param2, ..., paramn)**, on peut utiliser la syntaxe simplifiée suivante :

```
(param1, param2, ..., paramn) -> { code de la méthode m}
```

C'est ce qui est fait aux lignes 22-34 :
  - [subscriber] est le paramètre de la méthode [Observable.OnSubscribe.**call**] ;
  - lignes 23-32 : le code qu'on veut donner à la méthode [call] ;

- ligne 25 : on demande les listes électorales de façon synchrone à la couche [métier] injectée en ligne 13. Il va donc y avoir attente du résultat. Lorsque celui-ci va être reçu, il est passé à la méthode [onNext] du souscripteur ;
- ligne 28 : en cas d'erreur, l'exception est passée à la méthode [onError] du souscripteur ;
- ligne 31 : on n'attend qu'un résultat. Lorsque celui-ci a été obtenu (les listes électorales ou une exception), on indique au souscripteur que le processus observé a terminé d'émettre des résultats ;

On se rappellera bien que la méthode [RxElectionsMetier] rend un type Observable<ListeElectorale[]> et non le type ListeElectorale[] lui-même. Il faudra que le code appelant appelle la méthode Observable<ListeElectorale[]>.**subscribe** pour que le code des lignes 23-33 soit exécuté et rende les listes électorales au moyen de la ligne 25.

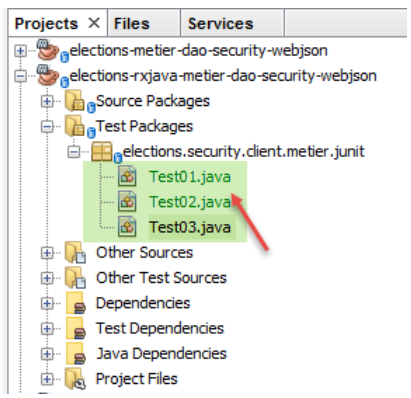
Le code des autres méthodes est analogue :

```
1. package elections.security.client.metier;
2.
3. import elections.security.client.entities.ListeElectorale;
4. import elections.security.client.entities.User;
5. import org.springframework.beans.factory.annotation.Autowired;
6. import org.springframework.stereotype.Component;
7. import rx.Observable;
8.
9. @Component
10.  public class RxElectionsMetier implements IRxElectionsMetier {
11.
12.    @Autowired
13.    private IElectionsMetier metier;
14.
15.    @Override
16.    public Observable<Void> authenticate(User user) {
17.      return Observable.create(subscriber -> {
18.        try {
19.          // appel méthode synchrone
20.          metier.authenticate(user);
21.          // on signale la fin de l'observable
22.          subscriber.onCompleted();
23.        } catch (Exception e) {
24.          // on fait suivre l'exception
25.          subscriber.onError(e);
26.        }
27.      });
28.
29.    }
30.
31.    @Override
32.    public Observable<ListeElectorale[]> getListesElectorales(User user) {
33.      return Observable.create(subscriber -> {
34.        try {
35.          // appel méthode synchrone puis réponse au souscripteur
36.          subscriber.onNext(metier.getListesElectorales(user));
37.          // on signale la fin de l'observable
38.          subscriber.onCompleted();
39.        } catch (Exception e) {
40.          // on fait suivre l'exception
41.          subscriber.onError(e);
42.        }
43.      });
44.    }
45.
46.    @Override
47.    public Observable<Integer> getNbSiegesAPourvoir(User user) {
48.      return Observable.create(subscriber -> {
49.        try {
```

```
50.         // appel méthode synchrone puis réponse au souscripteur
51.         subscriber.onNext(metier.getNbSiegesAPourvoir(user));
52.         // on signale la fin de l'observable
53.         subscriber.onCompleted();
54.       } catch (Exception e) {
55.         // on fait suivre l'exception
56.         subscriber.onError(e);
57.       }
58.     });
59.   }
60.
61.   @Override
62.   public Observable<Double> getSeuilElectoral(User user) {
63.     return Observable.create(subscriber -> {
64.       try {
65.         // appel méthode synchrone puis réponse au souscripteur
66.         subscriber.onNext(metier.getSeuilElectoral(user));
67.         // on signale la fin de l'observable
68.         subscriber.onCompleted();
69.       } catch (Exception e) {
70.         // on fait suivre l'exception
71.         subscriber.onError(e);
72.       }
73.     });
74.   }
75.
76.   @Override
77.   public Observable<Void> recordResultats(User user, ListeElectorale[] listesElectorales) {
78.     return Observable.create(subscriber -> {
79.       try {
80.         // appel méthode synchrone
81.         metier.recordResultats(user, listesElectorales);
82.         // on signale la fin de l'observable
83.         subscriber.onCompleted();
84.       } catch (Exception e) {
85.         // on fait suivre l'exception
86.         subscriber.onError(e);
87.       }
88.     });
89.   }
90.
91.   @Override
92.   public Observable<ListeElectorale[]> calculerSieges(User user, ListeElectorale[] listesElectorales) {
93.     return Observable.create(subscriber -> {
94.       try {
95.         // appel méthode synchrone puis réponse au souscripteur
96.         subscriber.onNext(metier.calculerSieges(user, listesElectorales));
97.         // on signale la fin de l'observable
98.         subscriber.onCompleted();
99.       } catch (Exception e) {
100.        // on fait suivre l'exception
101.        subscriber.onError(e);
102.      }
103.    });
104.  }
105. }
```

- lignes 20 et 81 : la méthode [onNext] du souscripteur n'est pas appelée parce que celui-ci n'attend pas de résultats ;

20.1.4 Les tests JUnit de la couche [métier]



20.1.4.1 Test01

Nous reprenons le test unitaire [Test01] étudié au paragraphe 17.4.4, page 334. Il a été prévu pour faire des appels synchrones à l'interface [IElectionsMetier]. Nous le modifions pour qu'il fasse des appels synchrones à la nouvelle interface [IRxElectionsMetier]. Il est en effet possible de faire des appels synchrones à une interface asynchrone RxJava. Le code devient le suivant :

```
1. package elections.security.client.metier.junit;
2.
3. import org.junit.Assert;
4. import org.junit.BeforeClass;
5. import org.junit.Test;
6. import org.junit.runner.RunWith;
7. import org.springframework.beans.factory.annotation.Autowired;
8. import org.springframework.boot.test.SpringApplicationConfiguration;
9. import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
10.
11.  import com.fasterxml.jackson.core.JsonProcessingException;
12.  import com.fasterxml.jackson.databind.ObjectMapper;
13.
14.  import elections.security.client.config.MetierConfig;
15.  import elections.security.client.entities.ElectionsException;
16.  import elections.security.client.entities.ListeElectorale;
17.  import elections.security.client.entities.User;
18.  import elections.security.client.metier.IRxElectionsMetier;
19.  import rx.observables.BlockingObservable;
20.
21.  @SpringApplicationConfiguration(classes = MetierConfig.class)
22.  @RunWith(SpringJUnit4ClassRunner.class)
23.  public class Test01 {
24.
25.    // couche [electionsMetier]
26.    @Autowired
27.    private IRxElectionsMetier electionsMetier;
28.
29.    // mappeur jSON
30.    private final ObjectMapper mapper = new ObjectMapper();
31.
32.    // utilisateurs
33.    static private User admin;
34.    static private User user;
35.    static private User unknown;
36.
37.    @BeforeClass
38.    public static void initTest() {
39.      admin = new User("admin", "admin");
40.      user = new User("user", "user");
41.      unknown = new User("x", "y");
42.    }
43.
44.    @Test()
45.    public void checkUserUser() {
46.      ElectionsException se = null;
47.      try {
48.        BlockingObservable.from(electionsMetier.authenticate(user)).firstOrDefault(null);
49.      } catch (ElectionsException e) {
50.        se = e;
51.      }
52.      Assert.assertNotNull(se);
53.      Assert.assertEquals("403 Forbidden", se.getErreurs().get(0));
54.    }
55.
56.    @Test()
57.    public void checkUserUnknown() {
58.      ElectionsException se = null;
59.      try {
60.        BlockingObservable.from(electionsMetier.authenticate(unknown)).firstOrDefault(null);
61.      } catch (ElectionsException e) {
62.        se = e;
63.      }
64.      Assert.assertNotNull(se);
65.      Assert.assertEquals("401 Unauthorized", se.getErreurs().get(0));
66.    }
67.
68.    @Test()
69.    public void checkUserAdmin() {
70.      ElectionsException se = null;
71.      try {
72.        BlockingObservable.from(electionsMetier.authenticate(admin)).firstOrDefault(null);
73.      } catch (ElectionsException e) {
74.        se = e;
75.      }
76.      Assert.assertNull(se);
77.    }
78.
```

```
79.    /**
80.     * vérification 1 : méthode de calcul des sièges on fixe en dur les listes
81.     */
82.    @Test
83.    public void calculSieges1() {
84.      // on crée le tableau des 7 listes candidates
85.      ListeElectorale[] listes = new ListeElectorale[7];
86.      listes[0] = new ListeElectorale("A", 32000, 0, false);
87.      listes[1] = new ListeElectorale("B", 25000, 0, false);
88.      listes[2] = new ListeElectorale("C", 16000, 0, false);
89.      listes[3] = new ListeElectorale("D", 12000, 0, false);
90.      listes[4] = new ListeElectorale("E", 8000, 0, false);
91.      listes[5] = new ListeElectorale("F", 4500, 0, false);
92.      listes[6] = new ListeElectorale("G", 2500, 0, false);
93.      // on calcule les sièges de chacune des listes
94.      listes = BlockingObservable.from(electionsMetier.calculerSieges(admin, listes)).first();
95.      // on vérifie les résultats
96.      Assert.assertEquals(2, listes[0].getSieges());
97.      Assert.assertFalse(listes[0].isElimine());
98.      Assert.assertEquals(2, listes[1].getSieges());
99.      Assert.assertFalse(listes[1].isElimine());
100.     Assert.assertEquals(1, listes[2].getSieges());
101.     Assert.assertFalse(listes[2].isElimine());
102.     Assert.assertEquals(1, listes[3].getSieges());
103.     Assert.assertFalse(listes[3].isElimine());
104.     Assert.assertEquals(0, listes[4].getSieges());
105.     Assert.assertFalse(listes[4].isElimine());
106.     Assert.assertEquals(0, listes[5].getSieges());
107.     Assert.assertTrue(listes[5].isElimine());
108.     Assert.assertEquals(0, listes[6].getSieges());
109.     Assert.assertTrue(listes[6].isElimine());
110.   }
111.
112.   /**
113.    * vérification 2 : méthode de calcul des sièges on demande les listes à la couche [metier] puis on fixe en dur les
114.    * voix
115.    */
116.   @Test
117.   public void calculSieges2() {
118.     // on crée le tableau des 7 listes candidates
119.     ListeElectorale[] listes = BlockingObservable.from(electionsMetier.getListesElectorales(admin)).first();
120.     // on fixe en dur les voix
121.     listes[0].setVoix(32000);
122.     listes[1].setVoix(25000);
123.     listes[2].setVoix(16000);
124.     listes[3].setVoix(12000);
125.     listes[4].setVoix(8000);
126.     listes[5].setVoix(4500);
127.     listes[6].setVoix(2500);
128.     // on calcule les sièges obtenus par chacune des listes
129.     listes = BlockingObservable.from(electionsMetier.calculerSieges(admin, listes)).first();
130.     // on vérifie les résultats
131.     Assert.assertEquals(2, listes[0].getSieges());
132.     Assert.assertFalse(listes[0].isElimine());
133.     Assert.assertEquals(2, listes[1].getSieges());
134.     Assert.assertFalse(listes[1].isElimine());
135.     Assert.assertEquals(1, listes[2].getSieges());
136.     Assert.assertFalse(listes[2].isElimine());
137.     Assert.assertEquals(1, listes[3].getSieges());
138.     Assert.assertFalse(listes[3].isElimine());
139.     Assert.assertEquals(0, listes[4].getSieges());
140.     Assert.assertFalse(listes[4].isElimine());
141.     Assert.assertEquals(0, listes[5].getSieges());
142.     Assert.assertTrue(listes[5].isElimine());
143.     Assert.assertEquals(0, listes[6].getSieges());
144.     Assert.assertTrue(listes[6].isElimine());
145.   }
146.
147.   /**
148.    * vérification 3 méthode de calcul des sièges on provoque une exception
149.    */
150.   @Test(expected = ElectionsException.class)
151.   public void calculSieges3() {
152.     // on crée un tableau de 24 listes candidates avec chacune 1 voix
153.     ListeElectorale[] listes = new ListeElectorale[25];
154.     // les 25 listes auront le même nombre de voix (4%)
155.     for (int i = 0; i < listes.length; i++) {
156.       listes[i] = new ListeElectorale("Liste" + (i + 1), 1, 0, false);
157.     }
158.     // calcul des sièges - normalement on doit avoir une ElectionsException
159.     // avec un seuil électoral de 5%
160.     BlockingObservable.from(electionsMetier.calculerSieges(admin, listes)).first();
161.   }
162.
163.   /**
164.    * enregistrement des résultats de l'élection
165.    *
```

```
166.    * @throws JsonProcessingException
167.    */
168.   @Test
169.   public void ecritureResultatsElections() throws JsonProcessingException {
170.     // on crée le tableau des 7 listes candidates
171.     ListeElectorale[] listes = BlockingObservable.from(electionsMetier.getListesElectorales(admin)).first();
172.     // on fixe en dur les voix
173.     listes[0].setVoix(32000);
174.     listes[1].setVoix(25000);
175.     listes[2].setVoix(16000);
176.     listes[3].setVoix(12000);
177.     listes[4].setVoix(8000);
178.     listes[5].setVoix(4500);
179.     listes[6].setVoix(2500);
180.     // on calcule les sièges obtenus par chacune des listes
181.     listes = BlockingObservable.from(electionsMetier.calculerSieges(admin, listes)).first();
182.     // on affiche les résultats
183.     for (int i = 0; i < listes.length; i++) {
184.       System.out.println(mapper.writeValueAsString(listes[i]));
185.     }
186.     // on enregistre les résultats dans la base de données
187.     BlockingObservable.from(electionsMetier.recordResultats(admin, listes)).firstOrDefault(null);
188.     // on vérifie les résultats
189.     listes = BlockingObservable.from(electionsMetier.getListesElectorales(admin)).first();
190.     // on affiche les résultats
191.     for (int i = 0; i < listes.length; i++) {
192.       System.out.println(mapper.writeValueAsString(listes[i]));
193.     }
194.     Assert.assertEquals(2, listes[0].getSieges());
195.     Assert.assertFalse(listes[0].isElimine());
196.     Assert.assertEquals(2, listes[1].getSieges());
197.     Assert.assertFalse(listes[1].isElimine());
198.     Assert.assertEquals(1, listes[2].getSieges());
199.     Assert.assertFalse(listes[2].isElimine());
200.     Assert.assertEquals(1, listes[3].getSieges());
201.     Assert.assertFalse(listes[3].isElimine());
202.     Assert.assertEquals(0, listes[4].getSieges());
203.     Assert.assertFalse(listes[4].isElimine());
204.     Assert.assertEquals(0, listes[5].getSieges());
205.     Assert.assertTrue(listes[5].isElimine());
206.     Assert.assertEquals(0, listes[6].getSieges());
207.     Assert.assertTrue(listes[6].isElimine());
208.   }
209. }
```

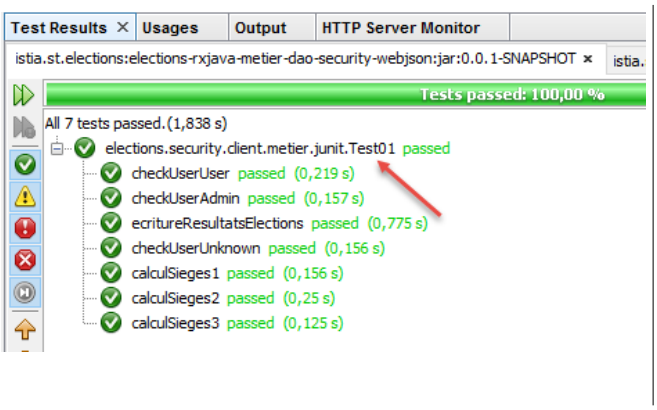
Etudions les modifications :

- ligne 48 : la méthode statique [BlockingObservable.from(Observable).first] :
  - s'abonne à l'observable paramètre de [from] ;
  - lance l'exécution du code associé à l'observable ;
  - attend de recevoir le 1er résultat. C'est donc une opération synchrone ;

  Nous utilisons ici la méthode [firstOrDefault(null)] parce que l'observable [metier.authenticate] ne rend pas de résultat lorsqu'il est exécuté. Le résultat de la méthode [firstOrDefault(null)] sera donc *null*, valeur inexploitée ici ;

  Nous reprenons ce schéma dans le reste du code à chaque fois que nous voulons faire appel à la couche [métier].

Le test unitaire [Test01] doit passer :



Travail à faire : vérifier que le test [Test01] passe.

20.1.4.2 Test02

Nous transformons le test [Test01] afin de tester désormais l'interface asynchrone [IRxElectionsMetier] en faisant des appels asynchrones à ses méthodes.

Etudions un premier test :

```
1.  // sémaphore de synchronisation de threads
2.  private CountDownLatch latch;
3.
4.  // ----------------------------------
5.  private ElectionsException checkUserUserException;
6.
7.  @Test()
8.  public void checkUserUser() throws InterruptedException {
9.    // sémaphore à 1
10.     latch = new CountDownLatch((1));
11.     // opération asynchrone
12.     electionsMetier.authenticate(user).subscribeOn(Schedulers.io())
13.             .subscribe((result) -> {
14.             },
15.                     (th) -> {
16.                       checkUserUserException = (ElectionsException) th;
17.                       latch.countDown();
18.                     },
19.                     () -> {
20.                       latch.countDown();
21.                     });
22.     // attente sémaphore
23.     latch.await();
24.     // vérification résultats
25.     Assert.assertNotNull(checkUserUserException);
26.     Assert.assertEquals("403 Forbidden", checkUserUserException.getErreurs().get(0));
27. }
```

- ligne 2 : un sémaphore est un outil utilisé pour synchroniser des threads entre eux. Des threads sont des flux d'exécution s'exécutant en parallèle. Pour exécuter une tâche T1, le thread [Thread1] peut avoir besoin qu'une tâche T2 exécutée par un thread [Thread2] soit terminée. Il attend alors que le thread [Thread2] lui envoie un signal indiquant que la tâche T2 est terminée. Il y a diverses façons de gérer cette synchronisation entre deux threads. La méthode utilisée ici est la suivante ;
  - ligne 10 : le thread [Thread1] crée un sémaphore avec la valeur 1 ;
  - ligne 12 : le thread [Thread1] crée et lance un thread [Thread2]. Ceci est obtenu par la syntaxe :

    ```
    electionsMetier.authenticate(user).subscribeOn(Schedulers.io())
    ```

    La méthode [Observable.subscribeOn] fixe le thread sur lequel s'exécutera le processus observé. Le paramètre de [subscribeOn] est un pool de threads. La bibliothèque RxJava en fournit plusieurs adaptés à différentes situations. Le pool [Schedulers.io()] est celui qui est conseillé pour les opérations réseau ;
  - lignes 12-13 : l'opération

    ```
    electionsMetier.authenticate(user).subscribeOn(Schedulers.io()).subscribe(...)
    ```

    exécute l'opération synchrone encapsulée dans l'observable [authenticate(user)]. Mais parce que cette opération synchrone est lancée sur un autre thread que le thread [Thread1], ce dernier n'attend pas la réponse de la méthode [subscribe] et passe à l'instruction suivante ;
  - ligne 23 : le thread [Thread1] s'arrête et attend que le sémaphore passe à 0 (il est à 1 pour l'instant) ;
- lignes 13-21 : la méthode [subscribe] admet comme paramètres trois fonctions lambda :
  - la première [**(result)->{...}**] est appelée à chaque fois que l'observable [authenticate(user)] émet un résultat [result]. Ici nous avons un observable [authenticate(user)] qui fait quelque chose mais n'émet aucun résultat. Le lambda [**(result)->{}**] ne sera donc jamais appelé. C'est pourquoi son code est ici vide [**{}**] ;
  - la seconde [**(th)->{...}**] reçoit comme paramètre un type [Throwable]. Elle est appelée lorsque l'exécution de l'observable rencontre une exception. Ici, nous traitons le paramètre [Throwable th] de la façon suivante :
    - ligne 16 : nous le mémorisons dans un champ de la classe de test de type [ElectionsException] car l'observable exécuté n'émet que ce type d'exception ;
    - ligne 17 : nous passons le sémaphore à 0 pour indiquer que le thread [Thread2] a terminé son travail ;
  - la troisième [**()->{...}**] est appelée lorsque l'observable n'a plus d'éléments à émettre. Nous traitons cet événement de la façon suivante :
    - ligne 20 : nous passons le sémaphore à 0 pour indiquer que le thread [Thread2] a terminé son travail ;

Il faut noter que le troisième lambda n'est pas appelé si une exception se produit. C'est pourquoi, on a été obligé de mettre le sémaphore à 0 également ligne 17 ;

- ligne 25 : lorsqu'on arrive à cette ligne, l'observable a terminé son travail. On peut alors faire les mêmes vérifications que dans le test [Test01] ;

Examinons un autre test :

```
1.  // ----------------------------------
2.    private ElectionsException calculSieges1Exception;
3.    private ListeElectorale[] listesCalculSieges1;
4.
5.    @Test
6.    public void calculSieges1() throws InterruptedException {
7.      // on crée le tableau des 7 listes candidates
8.      ListeElectorale[] listes = new ListeElectorale[7];
9.      listes[0] = new ListeElectorale("A", 32000, 0, false);
10.     listes[1] = new ListeElectorale("B", 25000, 0, false);
11.     listes[2] = new ListeElectorale("C", 16000, 0, false);
12.     listes[3] = new ListeElectorale("D", 12000, 0, false);
13.     listes[4] = new ListeElectorale("E", 8000, 0, false);
14.     listes[5] = new ListeElectorale("F", 4500, 0, false);
15.     listes[6] = new ListeElectorale("G", 2500, 0, false);
16.     // sémaphore à 1
17.     latch = new CountDownLatch((1));
18.     // opération asynchrone
19.     // on calcule les sièges de chacune des listes
20.     electionsMetier.calculerSieges(admin, listes).subscribeOn(Schedulers.io())
21.             .subscribe((result) -> {
22.               listesCalculSieges1 = result;
23.             },
24.                     (th) -> {
25.                       calculSieges1Exception = (ElectionsException) th;
26.                       latch.countDown();
27.                     },
28.                     () -> {
29.                       latch.countDown();
30.                     });
31.     // attente sémaphore
32.     latch.await();
33.     // on vérifie les résultats
34.     Assert.assertNull(calculSieges1Exception);
35.     Assert.assertEquals(2, listesCalculSieges1[0].getSieges());
36.     Assert.assertFalse(listesCalculSieges1[0].isElimine());
37.     Assert.assertEquals(2, listesCalculSieges1[1].getSieges());
38.     Assert.assertFalse(listesCalculSieges1[1].isElimine());
39.     Assert.assertEquals(1, listesCalculSieges1[2].getSieges());
40.     Assert.assertFalse(listesCalculSieges1[2].isElimine());
41.     Assert.assertEquals(1, listesCalculSieges1[3].getSieges());
42.     Assert.assertFalse(listesCalculSieges1[3].isElimine());
43.     Assert.assertEquals(0, listesCalculSieges1[4].getSieges());
44.     Assert.assertFalse(listesCalculSieges1[4].isElimine());
45.     Assert.assertEquals(0, listesCalculSieges1[5].getSieges());
46.     Assert.assertTrue(listesCalculSieges1[5].isElimine());
47.     Assert.assertEquals(0, listesCalculSieges1[6].getSieges());
48.     Assert.assertTrue(listesCalculSieges1[6].isElimine());
49.   }
```

- lignes 20-30 : exécution asynchrone de l'observable [electionsMetier.calculerSieges(admin, listes)] ;
- lignes 21-23 : l'exécution de l'observable rend un type [ListeElectorale[]] qu'on mémorise dans un champ de la classe de test, ligne 3 ;
- lignes 34-48 : ces vérifications sont celles du test [Test01] auxquelles on a ajouté la vérification de la ligne 34 qui s'assure qu'il n'y a pas eu d'exception ;

L'ensemble du test [Test02] est disponible dans le support de cours.

Travail à faire : passer le test [Test02] et vérifier qu'il passe.

20.1.4.3 Test03

Le test [Test03] fait la même chose que le test [Test01] : il teste l'interface [IRxElectionsMetier] par des appels synchrones à cette interface. C'est une copie du test [Test02] à deux détails près :

- les observables ne sont plus exécutés dans un thread différent de celui qui exécute les tests. Lorsque le thread [Thread1] exécute la méthode [subscribe] d'un observable, celle-ci démarre une opération HTTP vers le serveur également sur le thread [Thread1]. Toute la méthode [subscribe] devient alors synchrone ;

- puisqu'il n'y a plus qu'un thread, la synchronisation de threads devient inutile et le sémaphore disparaît ;

Voici deux exemples de tests :

```
1.    // ----------------------------------
2.    private ElectionsException checkUserUserException;
3.
4.    @Test()
5.    public void checkUserUser() throws InterruptedException {
6.      // opération synchrone
7.      electionsMetier.authenticate(user)
8.              .subscribe((result) -> {
9.              },
10.                     (th) -> {
11.                       checkUserUserException = (ElectionsException) th;
12.                     },
13.                     () -> {
14.                     });
15.     // vérification résultats
16.     Assert.assertNotNull(checkUserUserException);
17.     Assert.assertEquals("403 Forbidden", checkUserUserException.getErreurs().get(0));
18. }
```

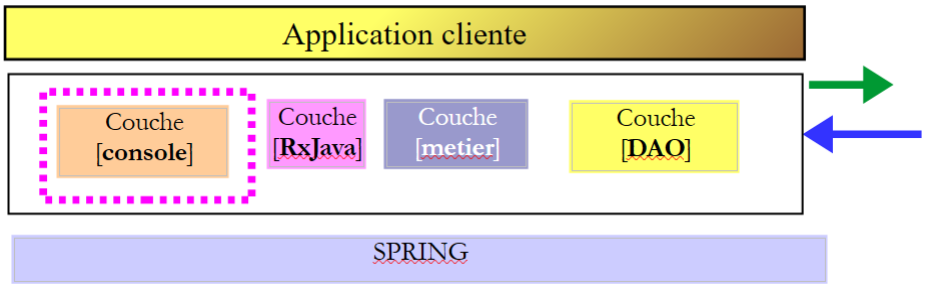
- ligne 7 : par défaut, la méthode [electionsMetier.authenticate(user).subscribe] s'exécute dans le thread du code appelant. On a donc une opération synchrone ;

```
1.    // ----------------------------------
2.    private ElectionsException calculSieges1Exception;
3.    private ListeElectorale[] listesCalculSieges1;
4.
5.    @Test
6.    public void calculSieges1() throws InterruptedException {
7.      // on crée le tableau des 7 listes candidates
8.      ListeElectorale[] listes = new ListeElectorale[7];
9.      listes[0] = new ListeElectorale("A", 32000, 0, false);
10.     listes[1] = new ListeElectorale("B", 25000, 0, false);
11.     listes[2] = new ListeElectorale("C", 16000, 0, false);
12.     listes[3] = new ListeElectorale("D", 12000, 0, false);
13.     listes[4] = new ListeElectorale("E", 8000, 0, false);
14.     listes[5] = new ListeElectorale("F", 4500, 0, false);
15.     listes[6] = new ListeElectorale("G", 2500, 0, false);
16.     // opération synchrone
17.     // on calcule les sièges de chacune des listes
18.     electionsMetier.calculerSieges(admin, listes)
19.             .subscribe((result) -> {
20.               listesCalculSieges1 = result;
21.             },
22.                     (th) -> {
23.                       calculSieges1Exception = (ElectionsException) th;
24.                     },
25.                     () -> {
26.                     });
27.     // on vérifie les résultats
28.     Assert.assertNull(calculSieges1Exception);
29.     Assert.assertEquals(2, listesCalculSieges1[0].getSieges());
30.     Assert.assertFalse(listesCalculSieges1[0].isElimine());
31.     Assert.assertEquals(2, listesCalculSieges1[1].getSieges());
32.     Assert.assertFalse(listesCalculSieges1[1].isElimine());
33.     Assert.assertEquals(1, listesCalculSieges1[2].getSieges());
34.     Assert.assertFalse(listesCalculSieges1[2].isElimine());
35.     Assert.assertEquals(1, listesCalculSieges1[3].getSieges());
36.     Assert.assertFalse(listesCalculSieges1[3].isElimine());
37.     Assert.assertEquals(0, listesCalculSieges1[4].getSieges());
38.     Assert.assertFalse(listesCalculSieges1[4].isElimine());
39.     Assert.assertEquals(0, listesCalculSieges1[5].getSieges());
40.     Assert.assertTrue(listesCalculSieges1[5].isElimine());
41.     Assert.assertEquals(0, listesCalculSieges1[6].getSieges());
42.     Assert.assertTrue(listesCalculSieges1[6].isElimine());
43.   }
```

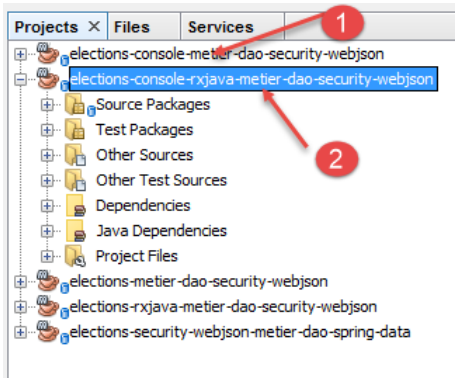
Travail à faire : passer le test [Test03] et vérifier qu'il passe.

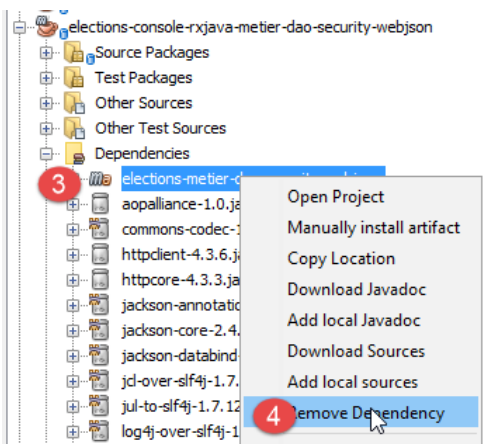
20.2 étape 2

Nous portons maintenant l'application console synchrone du chapitre 17.5, page 336, en une application toujours synchrone mais utilisant l'interface asynchrone [RxJava, metier, DAO] ;

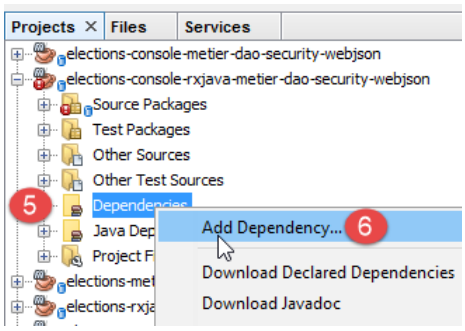


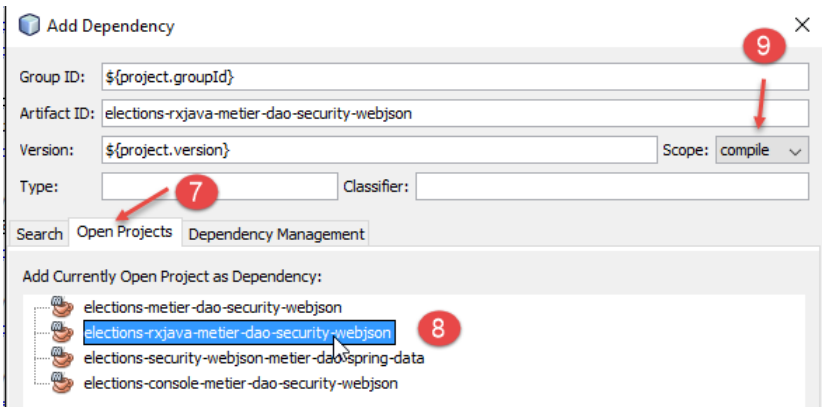
Nous partons du projet [elections-console-metier-dao-security-webjson] [1] du chapitre 17.5, page 336 que nous dupliquons dans un nouveau projet [elections-console-**rxjava-** metier-dao-security-webjson] [2] :



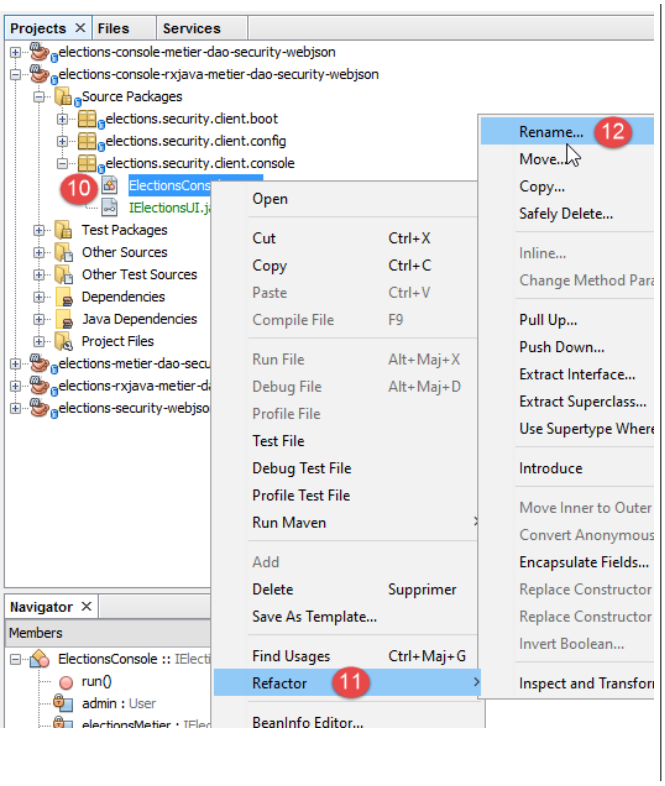


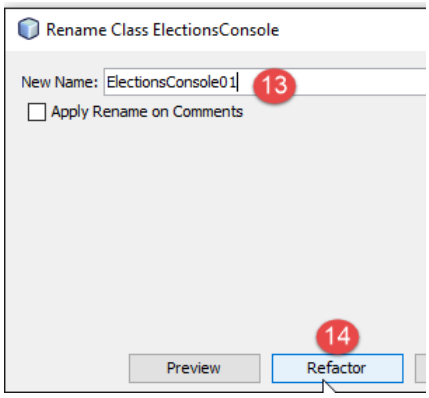
- en [3-4], dans le nouveau projet on supprime la dépendance sur l'ancienne couche [métier] synchrone ;





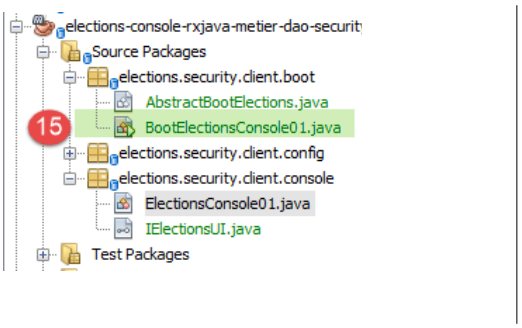
- en [5-9], on ajoute une dépendance sur la nouvelle couche [métier] asynchrone ;





- en [10-14], on renomme la classe [ElectionsConsole] en [ElectionsConsole01] ;

De même, on renomme la classe [BootElectionsConsole] en [BootElectionsConsole01] :



Le code actuel de la classe [BootElectionsConsole01] est le suivant :

```
1. package elections.security.client.boot;
2.
3. import elections.security.client.console.IElectionsUI;
4.
5.
6. public class BootElectionsConsole01 extends AbstractBootElections{
7.   public static void main(String[] arguments) {
8.     new BootElectionsConsole01().run();
9.   }
10.
11.    @Override
12.    protected IElectionsUI getUI() {
13.        return ctx.getBean("electionsConsole",IElectionsUI.class);
14.    }
15.  }
```

- ligne 13 : parce qu'on a changé le nom de la classe [ElectionsConsole] en [ElectionsConsole01], il faut désormais écrire :

```
        return ctx.getBean("electionsConsole01",IElectionsUI.class);
```

Revenons au code de la classe [ElectionsConsole01] :

```
1. @Component
2. public class ElectionsConsole01 implements IElectionsUI {
3.
4.   @Autowired
5.   private IElectionsMetier electionsMetier;
6.
7.   @Autowired
8.   private User admin;
9.
10.    @Override
11.    public void run() {
12.        // les listes en compétition
13.        ListeElectorale[] listes;
14.        // saisie des données
15.        try (Scanner clavier = new Scanner(System.in)) {
16.            // on demande les listes en compétition à la couche [metier]
17.            listes = electionsMetier.getListesElectorales(admin);
18.            ...
19.        // on fait le calcul des sièges
20.        listes=electionsMetier.calculerSieges(admin,listes);
21.        // on enregistre les résultats
22.        electionsMetier.recordResultats(admin,listes);
23.        ...
24.  }
```

Si on suit l'exemple du test [Test01] du paragraphe 20.1.4.1, page 388, les lignes 5, 17, 20 et 22 vont évoluer de la façon suivante :

```
1. @Component
2. public class ElectionsConsole01 implements IElectionsUI {
3.
4.   @Autowired
5.   private IRxElectionsMetier electionsMetier;
6.
7.   @Autowired
8.   private User admin;
9.
10.    @Override
11.    public void run() {
12.      // les listes en compétition
13.      ListeElectorale[] listes;
14.      // saisie des données
15.      try (Scanner clavier = new Scanner(System.in)) {
16.        // on demande les listes en compétition à la couche [metier]
17.        listes = BlockingObservable.from(electionsMetier.getListesElectorales(admin)).first();
18.        ...
19.      // on fait le calcul des sièges
20.      listes = BlockingObservable.from(electionsMetier.calculerSieges(admin, listes)).first();
21.      // on enregistre les résultats
22.      BlockingObservable.from(electionsMetier.recordResultats(admin, listes));
23.      ...
24.    }
```

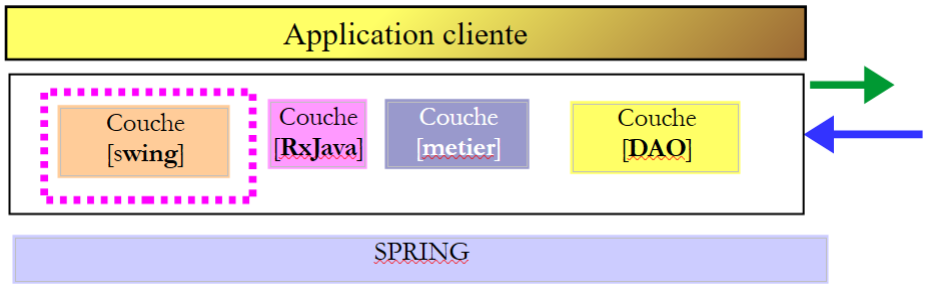
Travail à faire : configurez le projet pour exécuter la classe [BootElectionsConsole01] avec les trois paramètres [SS, Heures travaillées, Jours travaillés] et vérifiez que l'exécution du projet ainsi configuré donne les résultats attendus.

Travail à faire : configurez le projet pour exécuter le couple [BootElectionsConsole02, ElectionsConsole02] où la classe [ElectionsConsole02] aura été écrite en suivant le modèle du test [Test02] du paragraphe 20.1.4.2, page 391.

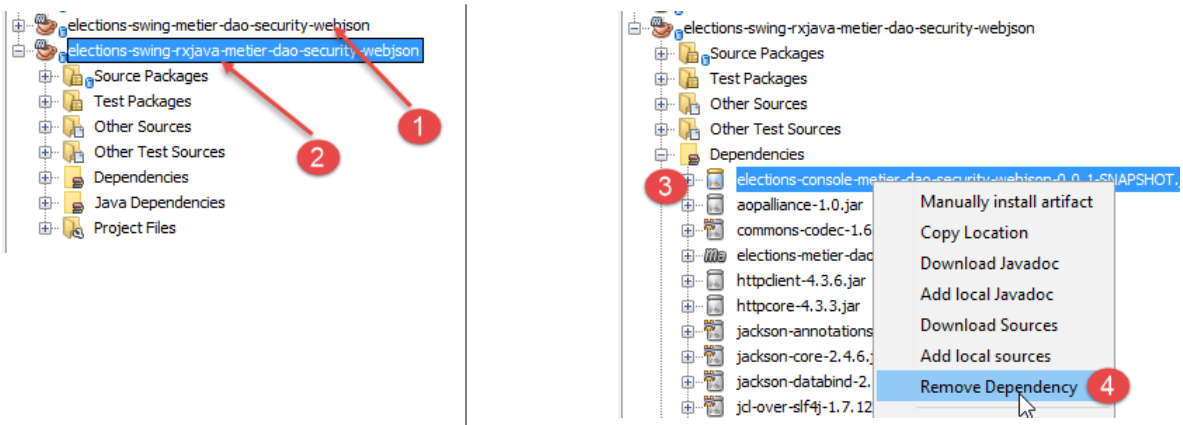
Travail à faire : configurez le projet pour exécuter le couple [BootElectionsConsole03, ElectionsConsole03] où la classe [ElectionsConsole03] aura été écrite en suivant le modèle du test [Test03] du paragraphe 20.1.4.3, page 392.

20.3 étape 3

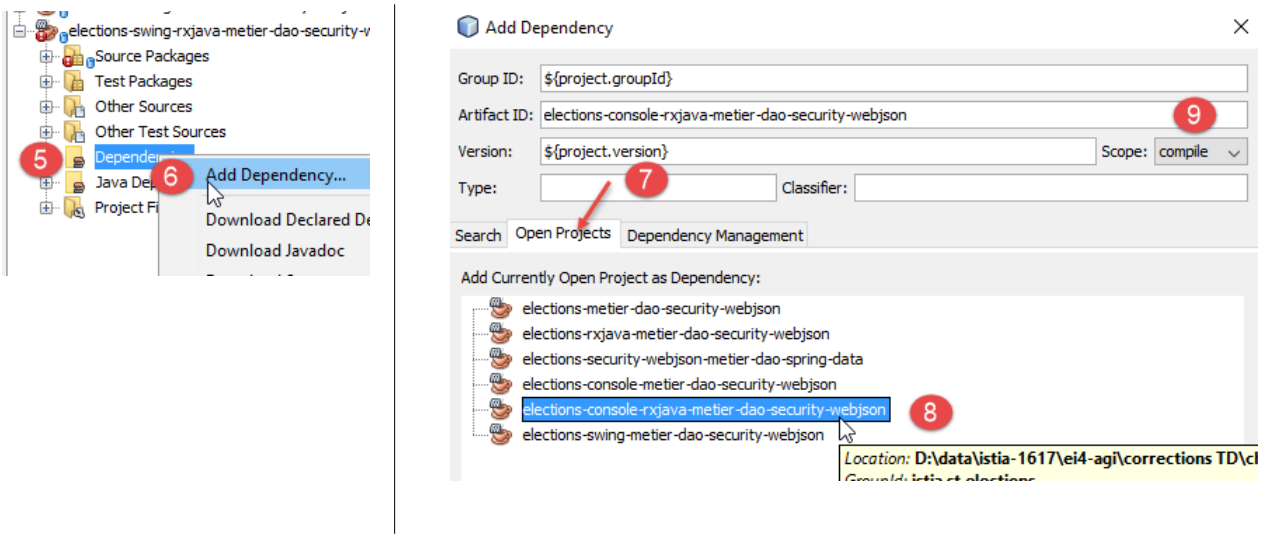
Nous passons maintenant au portage de l'application swing dans un environnement asynchrone.



Nous commençons par dupliquer le projet [elections-swing-metier-dao-security-webjson] [1] du chapitre 17.6, page 340, dans un nouveau projet [elections-swing-rxjava-metier-dao-security-webjson] [2] :



- en [3, 4], nous supprimons la dépendance sur la couche [console] synchrone ;

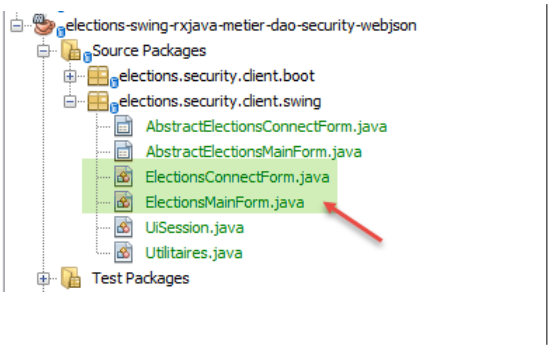


- en [5-9], nous ajoutons une dépendance sur la couche console asynchrone ;

La couche [swing] va faire de vrais appels asynchrones à la couche [métier]. Lors de l'appel à une méthode de celle-ci, il y aura deux threads :

- le thread de l'UI, celui qui gère les événements ;
- un thread d'E/S qui exécutera l'appel HTTP au serveur ;

Pendant toute la durée de l'appel asynchrone, nous devrions afficher une image d'attente ainsi qu'un bouton d'annulation. Nous ne le ferons pas ici et cela vous sera proposé comme amélioration de l'application. Les modifications ont lieu dans les deux classes qui font des appels à la couche [métier] :



20.3.1 Configuration Maven

Nous allons utiliser ici la bibliothèque [RxSwing] qui ajoute à la bibliothèque [RxJava] des fonctionnalités disponibles seulement dans un environnement Swing. Pour cela, nous modifions le fichier [pom.xml] de la façon suivante :

```
1. <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2.          xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
3.   <modelVersion>4.0.0</modelVersion>
4.   <groupId>istia.st.elections</groupId>
5.   <artifactId>elections-swing-rxjava-metier-dao-security-webjson</artifactId>
6.   <version>0.0.1-SNAPSHOT</version>
7.   <name>elections-swing-rxjava-metier-dao-security-webjson</name>
8.   <description>couche swing asynchrone du client web / jSON</description>
9.
10.    <properties>
11.      <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
12.      <java.version>1.8</java.version>
13.      <maven.compiler.source>1.8</maven.compiler.source>
14.      <maven.compiler.target>1.8</maven.compiler.target>
15.    </properties>
16.
17.    <dependencies>
18.      <!-- RxSwing -->
19.      <!-- https://mvnrepository.com/artifact/io.reactivex/rxswing -->
20.      <dependency>
21.        <groupId>io.reactivex</groupId>
22.        <artifactId>rxswing</artifactId>
23.        <version>0.27.0</version>
24.      </dependency>
25.      <!-- couches basses -->
26.      <dependency>
27.        <groupId>istia.st.elections</groupId>
28.        <artifactId>elections-console-rxjava-metier-dao-security-webjson</artifactId>
29.        <version>0.0.1-SNAPSHOT</version>
30.      </dependency>
31.    </dependencies>
32.
33.  </project>
```

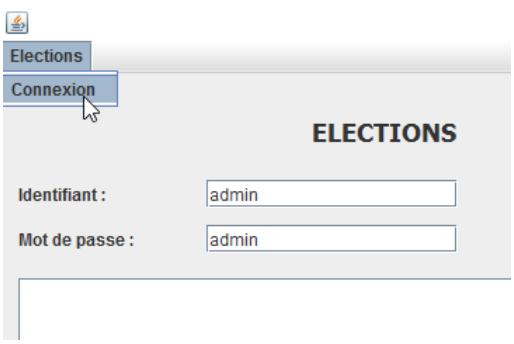
20.3.2 La classe [ElectionsConnectForm]

Dans un fonctionnement asynchrone, la classe [ElectionsConnectForm] devient la suivante :

```
1. package elections.security.client.swing;
2.
3. import elections.security.client.console.IElectionsUI;
4. import elections.security.client.entities.User;
5.
6. import java.awt.Dimension;
7. import java.awt.Toolkit;
8. import javax.swing.SwingUtilities;
9.
10.  import elections.security.client.metier.IRxElectionsMetier;
11.  import org.springframework.beans.factory.annotation.Autowired;
12.  import org.springframework.stereotype.Component;
13.  import rx.schedulers.Schedulers;
14.  import rx.schedulers.SwingScheduler;
```

```
15.
16. @Component
17. public class ElectionsConnectForm extends AbstractElectionsConnectForm implements IElectionsUI {
18.
19.   private static final long serialVersionUID = 1L;
20.
21.   // référence sur la couche [métier] asynchrone
22.   @Autowired
23.   private IRxElectionsMetier metier;
24.
25.   // utilisateur connecté
26.   private User user;
27.
28.   // formulaire principal
29.   @Autowired
30.   private ElectionsMainForm electionsMainForm;
31.
32.   // session UI
33.   @Autowired
34.   private UiSession uiSession;
35.
36.   @Override
37.   protected void doConnect() {
38.     if (isPageValid()) {
39.       // authentification de l'utilisateur
40.       metier.authenticate(user).subscribeOn(Schedulers.io()).observeOn(SwingScheduler.getInstance()).subscribe(
41.         // il n'y a pas de réponse
42.         (result) -> {
43.         },
44.         // gestion de l'exception
45.         (th) -> {
46.           // on note l'erreur
47.           String info = getInfoForException("Les erreurs suivantes se sont produites :", th);
48.           // on affiche l'info
49.           jTextPaneErreurs.setText(info);
50.           jTextPaneErreurs.setCaretPosition(0);
51.
52.         },
53.         // l'authentification est terminée
54.         () -> {
55.           // on mémorise l'utilisateur dans la session
56.           uiSession.setUser(user);
57.           // la vue de connexion est cachée
58.           setVisible(false);
59.           // la vue principale est affichée
60.           electionsMainForm.run();
61.         });
62.     }
63.   }
64.
65.   // initialisations
66.   @Override
67.   protected void init() {
68.     ...
69.   }
70.
71.   @Override
72.   public void run() {
73.     // on affiche l'interface graphique
74.     SwingUtilities.invokeLater(new Runnable() {
75.       public void run() {
76.         init();
77.         setVisible(true);
78.       }
79.     });
80.   }
81.
82.   private boolean isPageValid() {
83.     ...
84.   }
85.
86.   private String getInfoForException(String message, Throwable ex) {
87.     ...
88.   }
89.
90. }
```

- lignes 36-63 : la méthode [doConnect] est exécutée lorsque l'utilisateur appuie sur l'option de menu [Connexion] :



Tout est dans la ligne 40 :

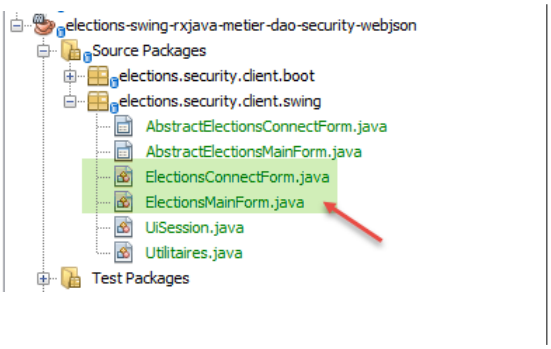
```
metier.authenticate(user).subscribeOn(Schedulers.io()).observeOn(SwingScheduler.getInstance()).subscribe(...)
```

- le processus observé est [metier.authenticate(user)] ;
- il sera exécuté sur un thread d'E/S pris dans le pool [Schedulers.io()] ;
- il sera observé dans le thread de l'UI, celui qui gère les événements de l'interface Swing [observeOn(SwingScheduler.getInstance())]. Ce thread est obtenu par la méthode [SwingScheduler.getInstance()] où [SwingScheduler] est une classe fournie par la bibliothèque [RxSwing]. Ceci est obligatoire. A l'obtention du résultat de l'opration asynchrone, celui-ci est souvent utilisé pour modifier des éléments de l'interface Swing. Or celle-ci ne peut être modifiée que dans le thread de l'UI, sinon on a une exception. Il faut donc que les lignes 41-61 s'exécutent dans le thread de l'UI. C'est assuré ici par la méthode [observeOn(SwingScheduler.getInstance())] ;

Commentons le reste du code :

- lignes 42-43 : ces lignes sont là pour respecter la syntaxe de la méthode [subscribe]. Elles ne seront jamais exécutées car le processus [metier.authenticate(user)] ne rend aucun résultat ;
- lignes 35-52 : à réception d'une exception, celle-ci est affichée ;
- lignes 54-61 : exécutées lorsque le processus [metier.authenticate(user)] signale la fin de ses émissions ;

20.3.3 La classe [ElectionsMainForm]

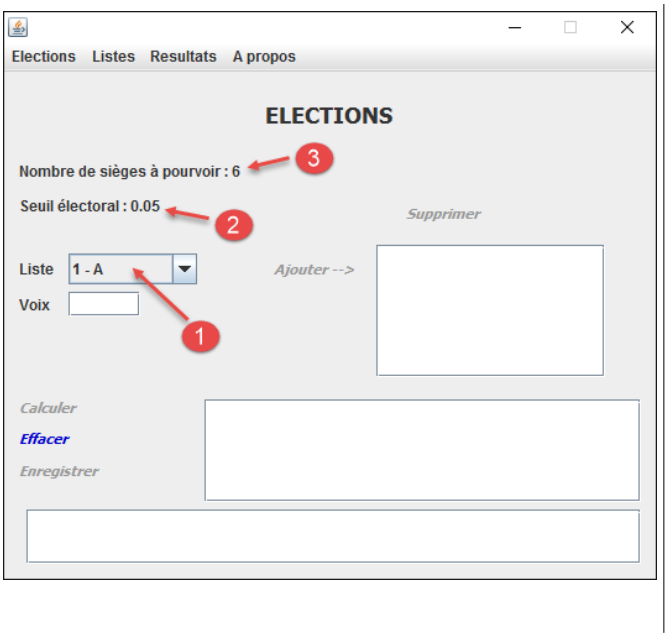


20.3.3.1 Initialisation de l'interface graphique

```
1. package elections.security.client.swing;
2.
3. import elections.security.client.console.IElectionsUI;
4. import elections.security.client.entities.ListeElectorale;
5. import elections.security.client.entities.User;
6. import elections.security.client.metier.IRxElectionsMetier;
7. import org.springframework.beans.factory.annotation.Autowired;
8. import org.springframework.stereotype.Component;
9. import rx.schedulers.Schedulers;
10.  import rx.schedulers.SwingScheduler;
11.
12.  import javax.swing.*;
13.  import java.awt.*;
14.  import java.util.ArrayList;
```

```
15.  import java.util.List;
16.
17.  @Component
18.  public class ElectionsMainForm extends AbstractElectionsMainForm implements IElectionsUI {
19.
20.    private static final long serialVersionUID = 1L;
21.
22.    // référence sur la couche [métier] asynchrone
23.    @Autowired
24.    private IRxElectionsMetier metier;
25.
26.    // session UI
27.    @Autowired
28.    private UiSession uiSession;
29.
30.    // utilisateur connecté
31.    private User user;
32.
33.    // modèles des listes JList
34.    private DefaultListModel<String> modèleNomsVoix = null;
35.    private DefaultListModel<String> modèleRésultats = null;
36.
37.    // les listes en compétition
38.    private ListeElectorale[] listes;
39.
40.    // listes saisies par l'utilisateur
41.    private final List<ListeElectorale> listesSaisies = new ArrayList<>();
42.    private ListeElectorale[] tListesSaisies;
43.
44.    // initialisations
45.    @Override
46.    protected void init() {
47.      // génération des composants par la classe parent
48.      super.init();
49.      // état formulaire
50.      Utilitaires.setEnabled(new JLabel[]{jLabelAjouter, jLabelCalculer, jLabelEnregistrer, jLabelSupprimer}, false);
51.      Utilitaires.setEnabled(
52.              new JMenuItem[]{jMenuItemAjouter, jMenuItemCalculer, jMenuItemEnregistrer, jMenuItemSupprimer}, false);
53.      // centrer la fenêtre
54.      Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
55.      Dimension frameSize = getSize();
56.      if (frameSize.height > screenSize.height) {
57.        frameSize.height = screenSize.height;
58.      }
59.      if (frameSize.width > screenSize.width) {
60.        frameSize.width = screenSize.width;
61.      }
62.      setLocation((screenSize.width - frameSize.width) / 2, (screenSize.height - frameSize.height) / 2);
63.      // utilisateur connecté
64.      user = uiSession.getUser();
65.      // initialisations locales
66.      modèleNomsVoix = new DefaultListModel<>();
67.      jListNomsVoix.setModel(modèleNomsVoix);
68.      modèleRésultats = new DefaultListModel<>();
69.      jListResultats.setModel(modèleRésultats);
70.      // on demande les listes à la couche [métier]
71.      metier.getListesElectorales(user).subscribeOn(Schedulers.io()).observeOn(SwingScheduler.getInstance())
72.              .subscribe(
73.                      // réponse
74.                      listesElectorales -> {
75.                        // on mémorise les listes
76.                        listes = listesElectorales;
77.                      },
78.                      // exception
79.                      (th) -> showException(th),
80.                      // fin observable
81.                      () -> {
82.                        // étape suivante
83.                        doInitStep2();
84.                      });
85.    }
86.  ...
```

- ligne 46 : la méthode [init] est exécutée lorsque la fenêtre associée va s'afficher. Elle a pour but d'initialiser les composants [1-3] ci-dessous :



- lignes 71-85 : on demande de façon asynchrone les listes candidates (composant [1]) ;
- ligne 71 : le processus observé est [metier.getListesElectorales(user)]. Il est exécuté sur un thread d'E/S [subscribeOn(Schedulers.io())] et observé sur le thread de l'UI [observeOn(SwingScheduler.getInstance()] ;
- lignes 74-77 : le résultat renvoyé par le processus observé est mémorisé dans le champ [listes] de la ligne 38 ;
- ligne 79 : l'éventuelle exception est traitée par la méthode suivante :

```
1.    private void showException(Throwable th) {
2.      // on affiche l'exception
3.      jTextPaneMessages.setText(getInfoForException("Les erreurs suivantes se sont produites : ", th));
4.      jTextPaneMessages.setCaretPosition(0);
5.    }
```

- lignes 81-84 : à la fin du processus observé, on exécute les lignes 81-84. Ces lignes ne sont pas exécutées s'il y a eu exception. La méthode [doInitStep2] assure l'étape 2 de l'initialisation de la façon suivante :

```
1.    private void doInitStep2() {
2.      // on associe les noms des listes au combo jComboBoxNomsListes
3.      for (int i = 0; i < listes.length; i++) {
4.        jComboBoxNomsListes.addItem(String.format("%s - %s", listes[i].getId(), listes[i].getNom()));
5.      }
6.      // nombre de sièges à pourvoir
7.      metier.getNbSiegesAPourvoir(user).subscribeOn(Schedulers.io()).observeOn(SwingScheduler.getInstance())
8.              .subscribe(
9.                      // réponse
10.                     nbSiegesAPourvoir -> {
11.                       // on initialise le label lié à cette information
12.                       jLabelSAP.setText(jLabelSAP.getText() + nbSiegesAPourvoir);
13.                     },
14.                     // exception
15.                     (th) -> showException(th),
16.                     // fin observable
17.                     () -> {
18.                       // étape suivante
19.                       doInitStep3();
20.                     });
21.   }
```

- lignes 3-5 : on utilise le résultat de l'étape précédente pour remplir la liste déroulante avec les noms des listes candidates ;
- lignes 7-20 : on demande le nombre de sièges à pourvoir de façon asynchrone ;
- ligne 7 : le processus observé est [metier.getNbSiegesAPourvoir(user)]. Il est exécuté sur un thread d'E/S [subscribeOn(Schedulers.io())] et observé sur le thread de l'UI [observeOn(SwingScheduler.getInstance()] ;
- lignes 10-13 : le résultat renvoyé par le processus est utilisé pour mettre à jour l'interface graphique ;
- ligne 15 : on affiche l'éventuelle exception ;
- lignes 17-20 : à réception du signal de fin de l'observable, on passe à l'étape 3 du processus d'initialisation ;

L'étape 3 de l'initialisation est assurée par le code suivant :

```
1.    private void doInitStep3() {
2.      // seuil électoral
3.      metier.getSeuilElectoral(user).subscribeOn(Schedulers.io()).observeOn(SwingScheduler.getInstance())
4.              .subscribe(
5.                      // réponse
6.                      seuilElectoral -> {
7.                        // on initialise le label lié à cette information
8.                        jLabelSE.setText(jLabelSE.getText() + seuilElectoral);
9.                      },
10.                     // exception
11.                     (th) -> showException(th),
12.                     // fin observable
13.                     () -> {
14.                     });
15. }
```

- lignes 3-4 : on demande le seuil électoral de façon asynchrone ;
- ligne 3 : le processus observé est [metier.getSeuilElectoral(user)]. Il est exécuté sur un thread d'E/S [subscribeOn(Schedulers.io())] et observé sur le thread de l'UI [observeOn(SwingScheduler.getInstance()] ;
- lignes 6-9 : le résultat renvoyé par le processus est utilisé pour mettre à jour l'interface graphique ;
- ligne 11 : on affiche l'éventuelle exception ;
- lignes 13-14 : à réception du signal de fin de l'observable, rien n'est fait : le processus d'initialisation de l'interface graphique est terminé ;

20.3.3.2 Calcul des sièges obtenus par les différentes listes

La méthode [doCalculer] a pour fonction de calculer le nombre de sièges obtenus par les différentes listes :

```
1.    @Override
2.    protected void doCalculer() {
3.      tListesSaisies = listesSaisies.toArray(new ListeElectorale[0]);
4.      // calcul des sièges
5.      String info = null;
6.      metier.calculerSieges(user, tListesSaisies).subscribeOn(Schedulers.io()).observeOn(SwingScheduler.getInstance())
7.              .subscribe(
8.                      // traitement résultat
9.                      result -> consumeResultSieges(result),
10.                     // traitement exception
11.                     th -> showException(th),
12.                     // fin observable
13.                     () -> {
14.                     }
15.             );
16. }
```

- lignes 6-15 : on calcule de façon asynchrone les sièges obtenus par les différentes listes ;
- ligne 6 : le processus observé est [metier.calculerSieges(user, tListesSaisies)]. Il est exécuté sur un thread d'E/S [subscribeOn(Schedulers.io())] et observé sur le thread de l'UI [observeOn(SwingScheduler.getInstance()] ;
- ligne 9 : le résultat renvoyé par le processus est utilisé par la méthode [consumeResultSieges] ;
- ligne 11 : on affiche l'éventuelle exception ;
- lignes 13-14 : à réception du signal de fin de l'observable, rien n'est fait ;

Ligne 9, la méthode [consumeResultSieges] exploite le résultat renvoyé par le processus observé, les listes candidates avec leurs champs [sieges, elimine] mis à jour :

```
1.    private void consumeResultSieges(ListeElectorale[] tListesSaisies) {
2.      // on mémorise le résultat
3.      this.tListesSaisies = tListesSaisies;
4.      // affichage des résultats
5.      modèleRésultats.clear();
6.      for (int i = 0; i < tListesSaisies.length; i++) {
7.        modèleRésultats.addElement(tListesSaisies[i].toString());
8.      }
9.      // maj état formulaire
10.     Utilitaires.setEnabled(new JLabel[]{jLabelEnregistrer}, true);
11.     Utilitaires.setEnabled(new JLabel[]{jLabelCalculer}, false);
12.     Utilitaires.setEnabled(new JMenuItem[]{jMenuItemEnregistrer}, true);
13.     Utilitaires.setEnabled(new JMenuItem[]{jMenuItemCalculer}, false);
14.     jTextPaneMessages.setText("Calcul terminé");
15. }
```

- lignes 4-14 : le résultat obtenu est utilisé pour mettre à jour l'interface graphique ;

20.3.3.3 Enregistrement des résultats de l'élection

L'enregistrement des résultats de l'élection est fait par la méthode [doEnregistrer] suivante :

```
1.    @Override
2.    protected void doEnregistrer() {
3.      // on demande l'enregistrement à la couche [métier]
4.      metier.recordResultats(user, tListesSaisies).subscribeOn(Schedulers.io()).observeOn(SwingScheduler.getInstance())
5.              .subscribe(
6.                      // traitement du résultat - il n'y en a pas ici
7.                      (param) -> {
8.                      },
9.                      // traitement de l'exception
10.                     (th) -> showException(th),
11.                     // fin observable
12.                     () -> {
13.                       // maj du formulaire
14.                       Utilitaires.setEnabled(new JLabel[]{jLabelEnregistrer}, false);
15.                       Utilitaires.setEnabled(new JMenuItem[]{jMenuItemEnregistrer}, false);
16.                       jTextPaneMessages.setText("Enregistrement des résultats réalisé");
17.                     }
18.             );
19.   }
```

- lignes 4-17 : on enregistre les résultats de l'élection de façon asynchrone ;
- ligne 4 : le processus observé est [metier.recordResultats(user, tListesSaisies)]. Il est exécuté sur un thread d'E/S [subscribeOn(Schedulers.io())] et observé sur le thread de l'UI [observeOn(SwingScheduler.getInstance()] ;
- lignes 7-8 : ces lignes ne seront jamais exécutées car le processus observé ne rend pas de résultat ;
- ligne 10 : on affiche l'éventuelle exception ;
- lignes 14-16 : à réception du signal de fin de l'observable, on met à jour l'interface graphique ;

Travail à faire : vérifier que l'application Swing fonctionne. Puis faites évoluer l'interface graphique et le code pour que lors d'une opération asynchrone avec le serveur web / jSON, une image d'attente apparaisse ainsi qu'une option d'annulation de l'opération en cours.

---

21 Conclusion

Il a été écrit en introduction de ce document :

Son objectif est d'enseigner le langage Java dans une optique professionnelle. Pour cette raison, nous nous appuyons fortement sur le framework Spring [http://spring.io/] très utilisé dans le développement JEE (Java Enterprise Edition). Logiquement, ce cours devrait être suivi par un cours JEE.

Le lecteur intéressé pourra poursuivre avec divers documents enseignant JEE :

- [Introduction à Struts 2 par l'exemple] (2012). Struts a été le premier framework MVC dans le monde JEE, il y a plus de 10 ans (~2005). Struts 2 est une évolution du framework Struts initial ;
- [Introduction à JSF2, Primefaces et Primefaces mobile] (2012). Ce document présente un autre framework MVC du mode JEE, la version 2 de JSF (Java Server Faces) ainsi que les EJB3 (Enterprise Java Bean) qui offrent des possibilités se rapprochant de ce qu'offre Spring. Il présente également [Primefaces] une bibliothèque de composants Ajax (Asynchronous Javascript And Xml) pour JSF et sa version [Primefaces mobile] à destination des smartphones et tablettes ;
- [Tutoriel Angular JS et Spring 4] (2014) qui présente une autre architecture web, celle du client / serveur. Ici,
  - le client est construit avec le framework jS Angular ;
  - le serveur est construit avec Spring MVC ;
- [Spring MVC et Thymeleaf] (2015) qui présente les détails de Spring MVC et associe celui-ci au générateur de pages Thymeleaf. L'application construite est la même que celle du document [Tutoriel Angular JS et Spring 4], ce qui permet de comparer les deux architectures ;
- [Exploiter une base de données relationnelle avec l'écosystème Spring] (2015). Ce document est celui à partir duquel a été écrit ce TD. Il n'apporte rien de plus si ce n'est que les exemples ont été traités avec 6 SGBD (MySQL, Oracle, SQL Server, PostgreSQL, IBM DB2, Firebird) et 3 implémentation JPA (Hibernate, EclipseLink, OpenJPA). Son objectif est de montrer comment architecturer une application afin qu'elle soit portable d'un SGBD à un autre et d'une implémentation JPA à une autre ;

22 Annexes

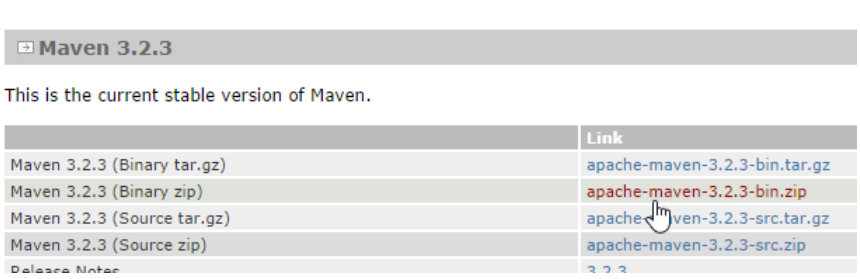
Nous présentons ici comment installer les outils utilisés dans ce document sur des machines windows 7 ou 8. Les copies d'écran désignent généralement les versions 64 bits des SGBD et outils installés. Le lecteur s'adaptera à son propre environnement.

22.1 Installation d'un JDK

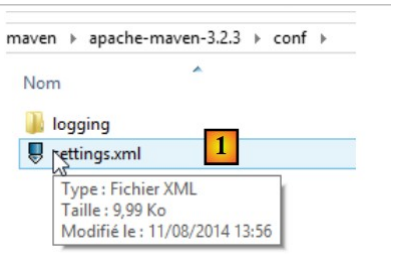
On trouvera à l'URL [http://www.oracle.com/technetwork/java/javase/downloads/index.html] (octobre 2014), le JDK le plus récent. On nommera par la suite <jdk-install> le dossier d'installation du JDK.

22.2 Installation de Maven

Maven est un outil de gestion des dépendances d'un projet Java et plus encore. Il est disponible (octobre 2014) à l'URL [http://maven.apache.org/download.cgi].



Téléchargez et dézippez l'archive. Nous appellerons <maven-install> le dossier d'installation de Maven.



- en [1], le fichier [conf / settings.xml] configure Maven ;

On y trouve les lignes suivantes :

```
1.    <!-- localRepository
2.     | The path to the local repository maven will use to store artifacts.
3.     |
4.     | Default: ${user.home}/.m2/repository
5.    <localRepository>/path/to/local/repo</localRepository>
6.  -->
```

La valeur par défaut de la ligne 4 peut poser problème à certains logiciels utilisant Maven, si comme moi votre {user.home} a un espace dans son chemin (par exemple [C:\Users\Serge Tahé]). On désignera (ligne 7) un autre dossier pour le dépôt local Maven :

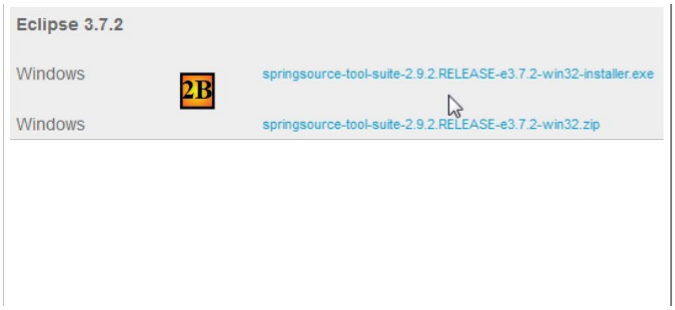
```
1.    <!-- localRepository
2.     | The path to the local repository maven will use to store artifacts.
3.     |
4.     | Default: ${user.home}/.m2/repository
5.    <localRepository>/path/to/local/repo</localRepository>
6.    -->
7.  <localRepository>D:\Programs\devjava\maven\.m2\repository</localRepository>
```

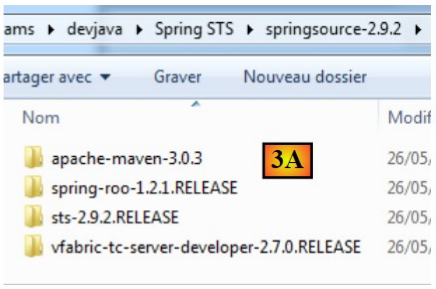
On évitera, ligne 7, un chemin qui contient des espaces.

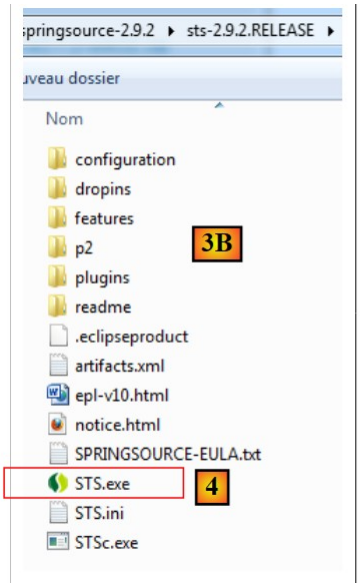
22.3 Installation de STS (Spring Tool Suite)

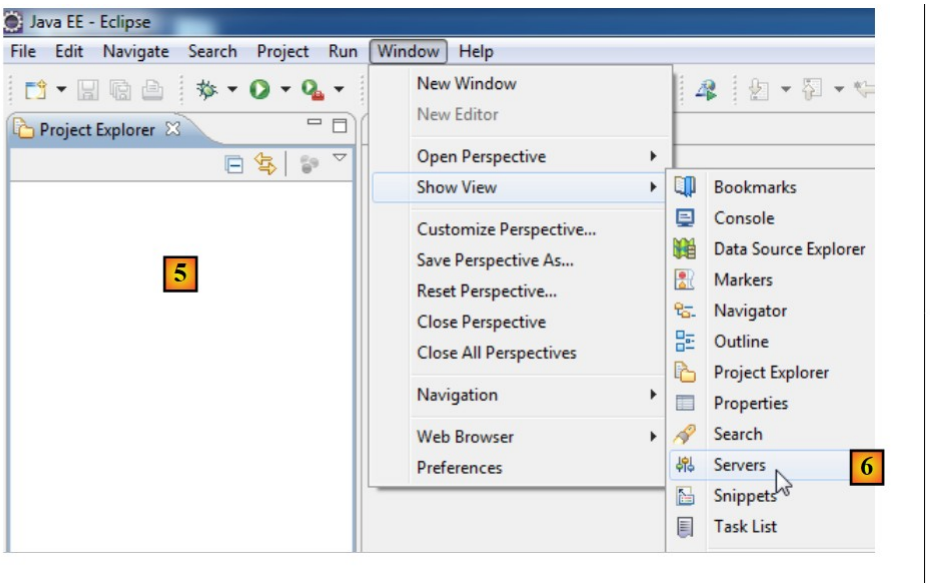
Nous allons installer **SpringSource Tool Suite** [http://www.springsource.com/developer/sts] (octobre 2014), un Eclipse pré-équipé avec de nombreux plugins liés au framework Spring et également avec une configuration Maven pré-installée.

- aller sur le site de **SpringSource Tool Suite** (STS) [1], pour télécharger la version courante de STS [2A] [2B],

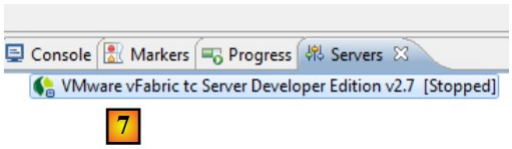






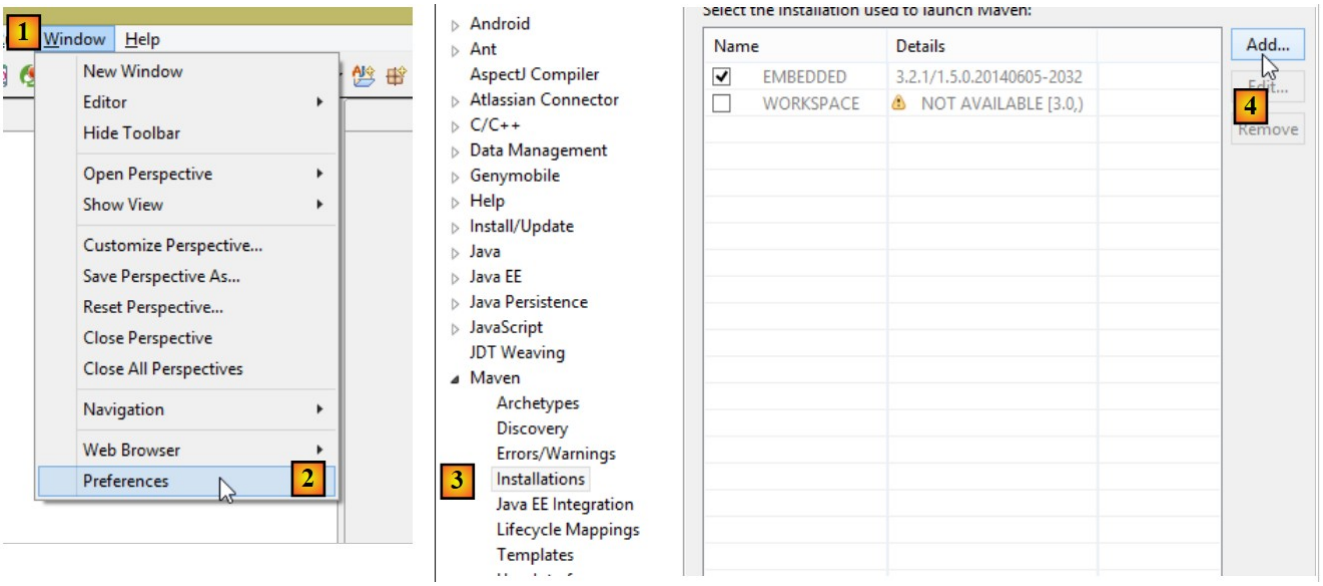


- le fichier téléchargé est un installateur qui crée l'arborescence de fichiers [3A] [3B]. En [4], on lance l'exécutable,
- en [5], la fenêtre de travail de l'IDE après avoir fermé la fenêtre de bienvenue. En [6], on fait afficher la fenêtre des serveurs d'applications,

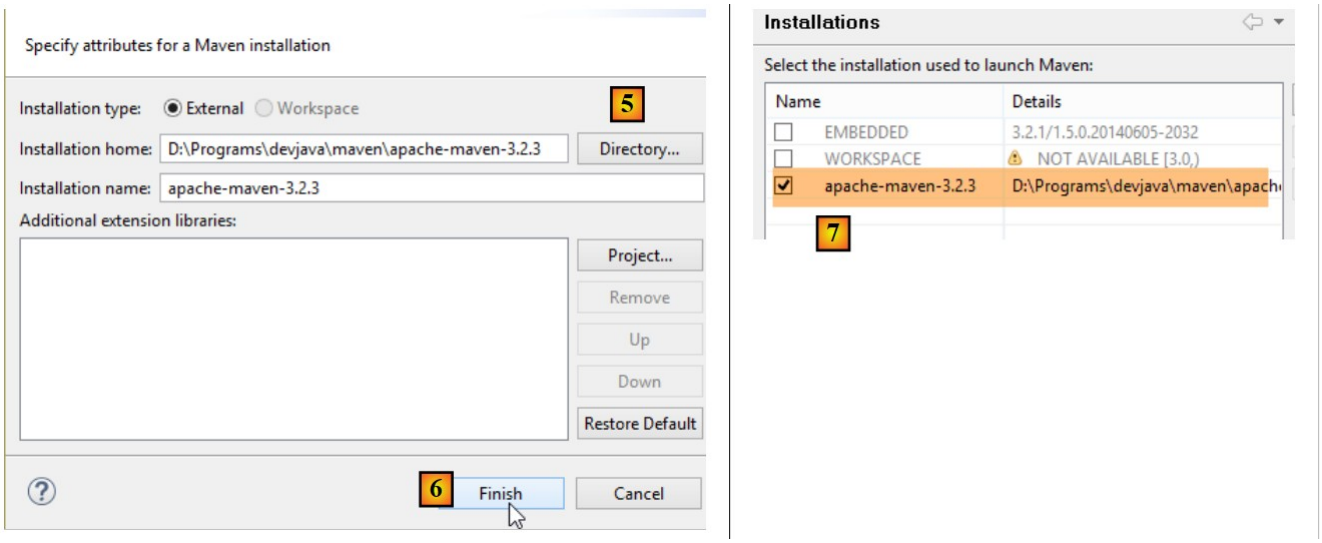


- en [7], la fenêtre des serveurs. Un serveur est enregistré. C'est un serveur VMware compatible Tomcat.

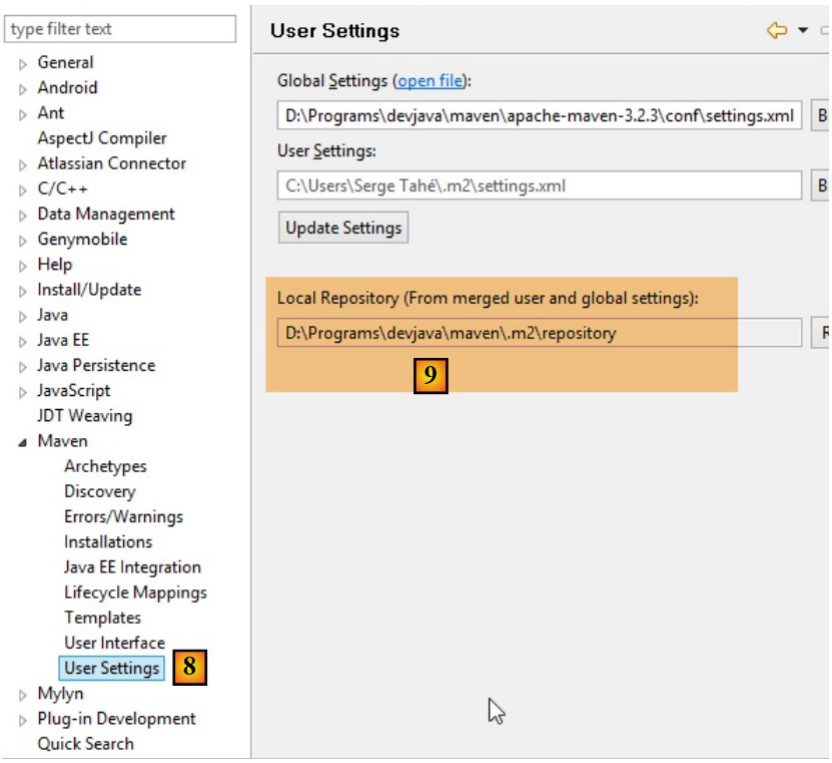
Il faut indiquer à STS le dossier d'installation de Maven :



- en [1-2], on configure STS ;
- en [3-4], on ajoute une nouvelle installation Maven ;

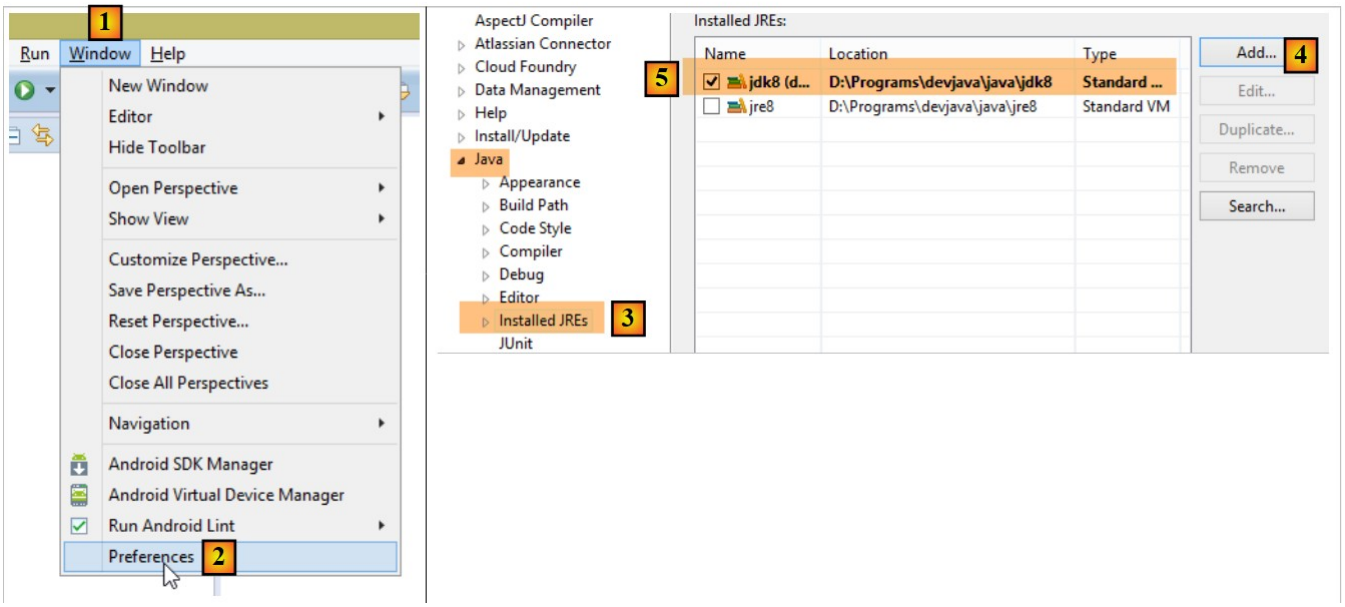


- en [5], on désigne le dossier d'installation de Maven ;
- en [6], on termine l'assistant ;
- en [7], on fait de la nouvelle installation Maven, l'installation par défaut ;



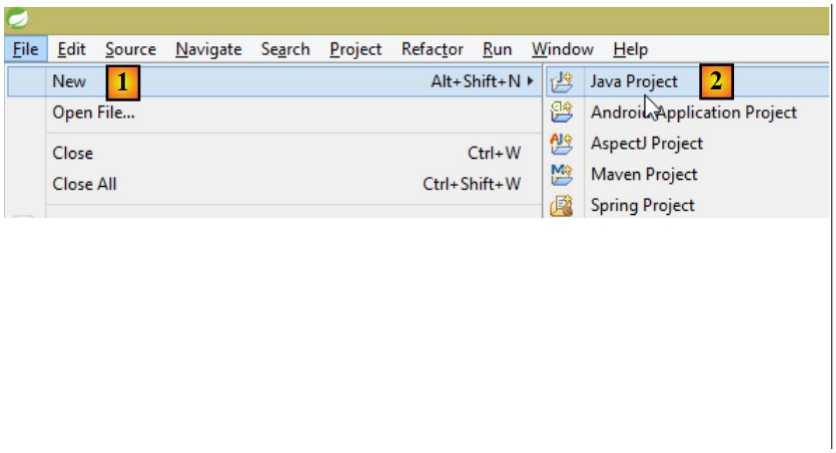
- en [8-9], on vérifie le dépôt local de Maven, le dossier où il mettra les dépendances qu'il téléchargera et où STS mettra les artifacts qui seront construits ;

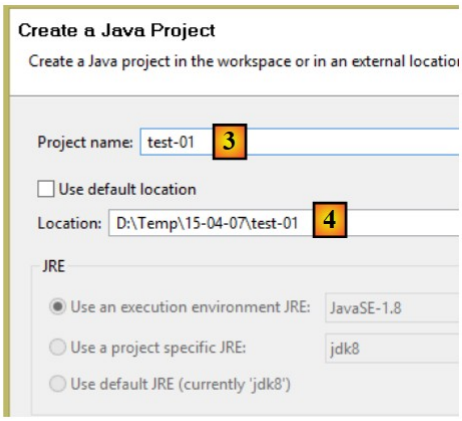
Il faut également choisir un JDK (Java Development Kit) pour exécuter à la fois les projets Eclipse sans et avec Maven [1-5].



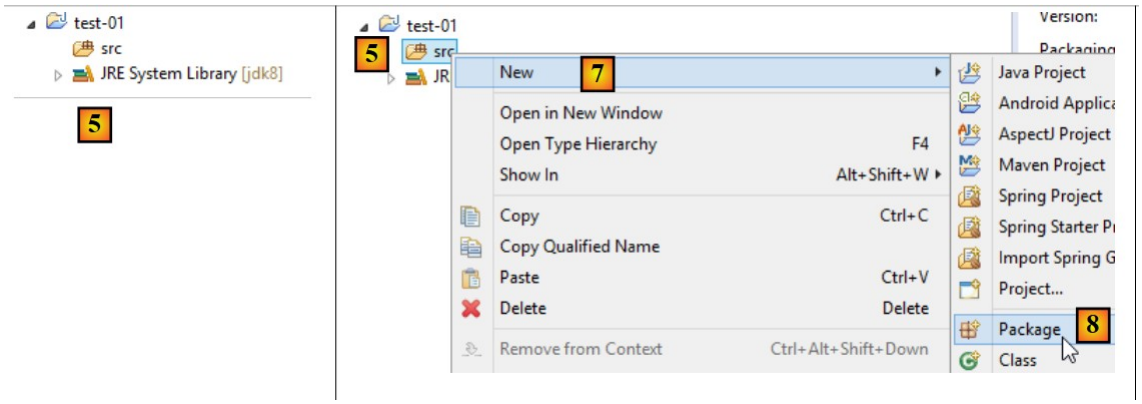
Avec [4], on peut ajouter des JDK (Java Development Kit) ou des JRE (Java Runtime Environment). Ce dernier sait exécuter des fichiers .class mais ne sait pas compiler les .java pour les produire. Le JDK sait faire les deux. On choisira un JDK parce que certaines opérations Maven ont besoin d'un JDK.

Pour créer un projet Eclipse, on procèdera de la façon suivante :

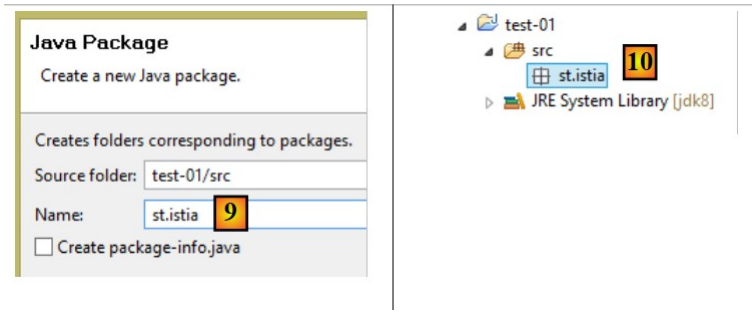




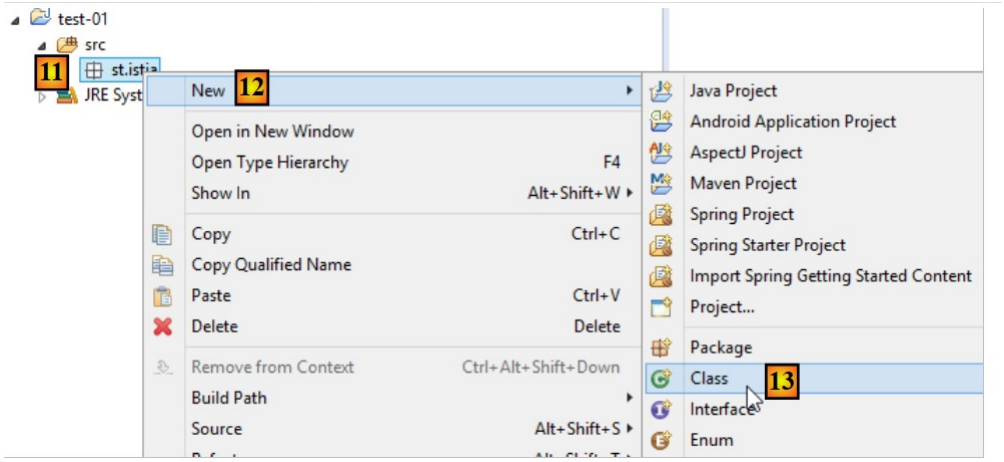
- en [3], donnez un nom au projet ;
- en [4], désignez un dossier existant et vide ;



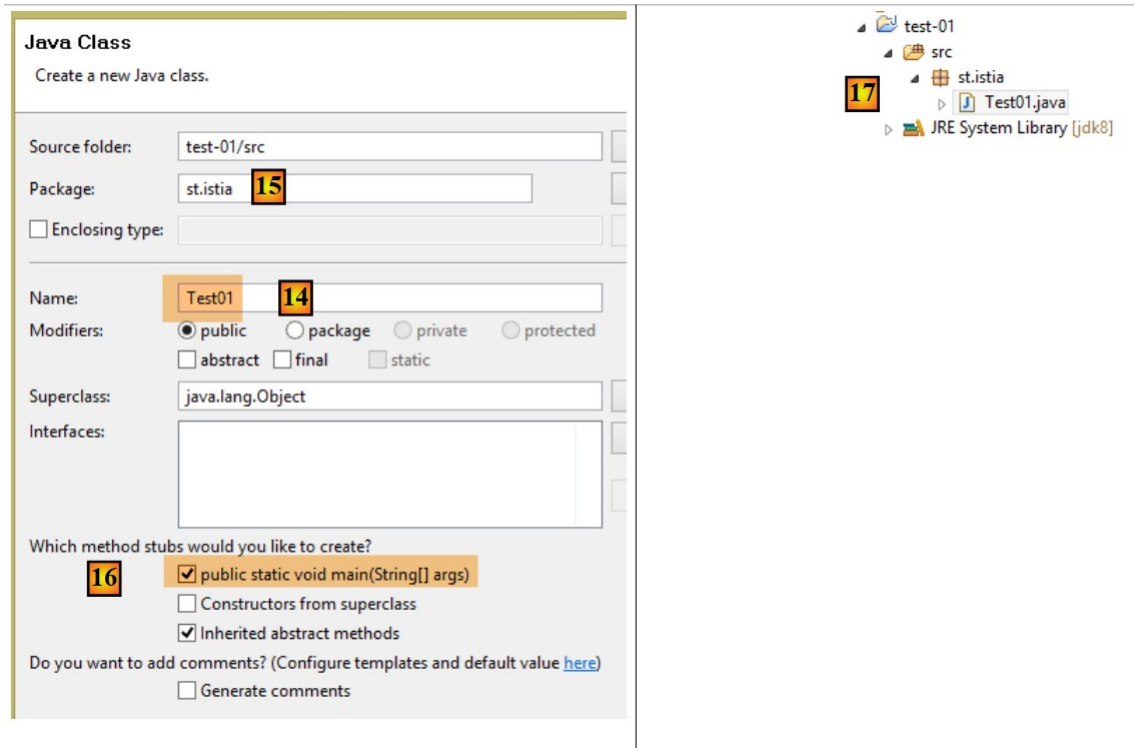
- en [5], le projet créé ;
- en [5-8], créez un package. Un package est un dossier qui contient du code Java. Deux classes peuvent porter le même nom si elles appartiennent à des packages différents. Dans un projet, il ne peut y avoir deux packages de même nom. Ainsi on ne peut utiliser un nom de package qui existerait dans une des dépendances du projet. Une entreprise utilisera comme nom de package, un nom précisant l'entreprise, le projet et les différentes branches de celui-ci ;



- en [9], donnez un nom au package ;
- en [10], le package créé ;



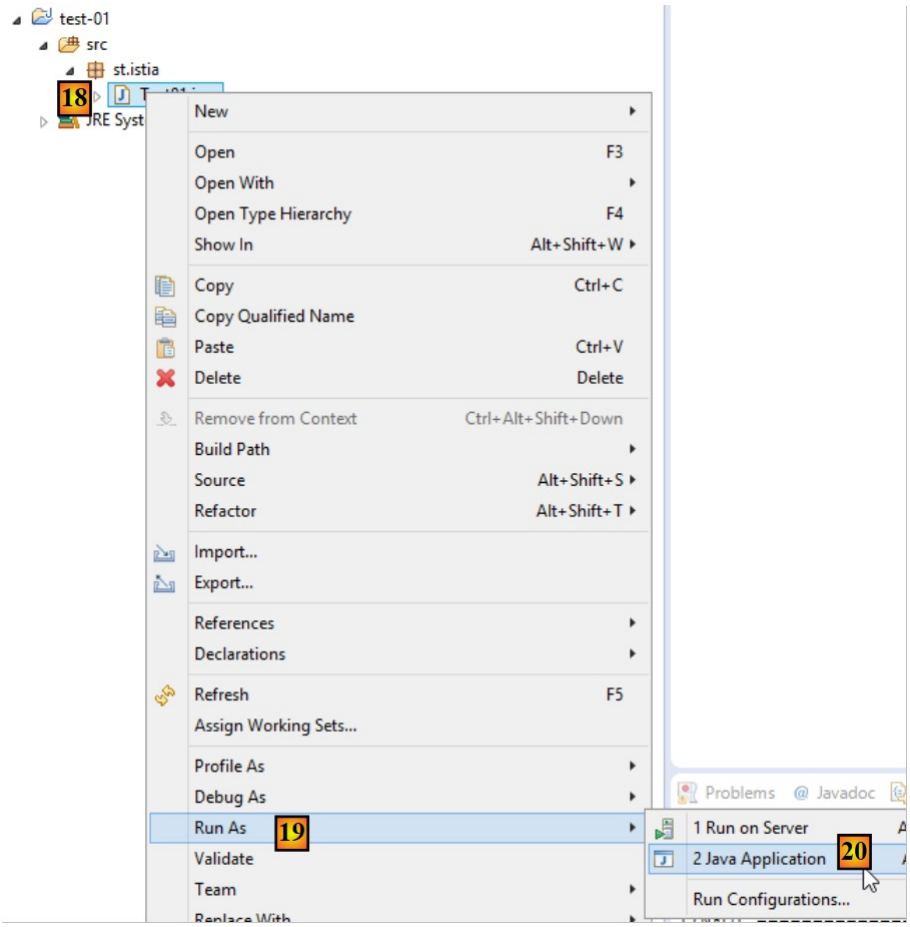
- en [11-13], on crée une classe dans le package qui a été créé ;



- en [14], donnez un nom à la classe (doit respecter la norme CamelCase - pour chaque mot du nom, commencer par une majuscule suivie de minuscules) ;
- en [15], vérifiez le package ;
- en [16], cochez la case. Elle demande à ce que la méthode statique [main] soit générée. Cette méthode rend une classe exécutable, ç-à-d la première classe à exécuter dans un projet ;
- en [17], la classe ainsi créée ;

Tapez dans la méthode [main] le code suivant qui affiche un texte sur la console :

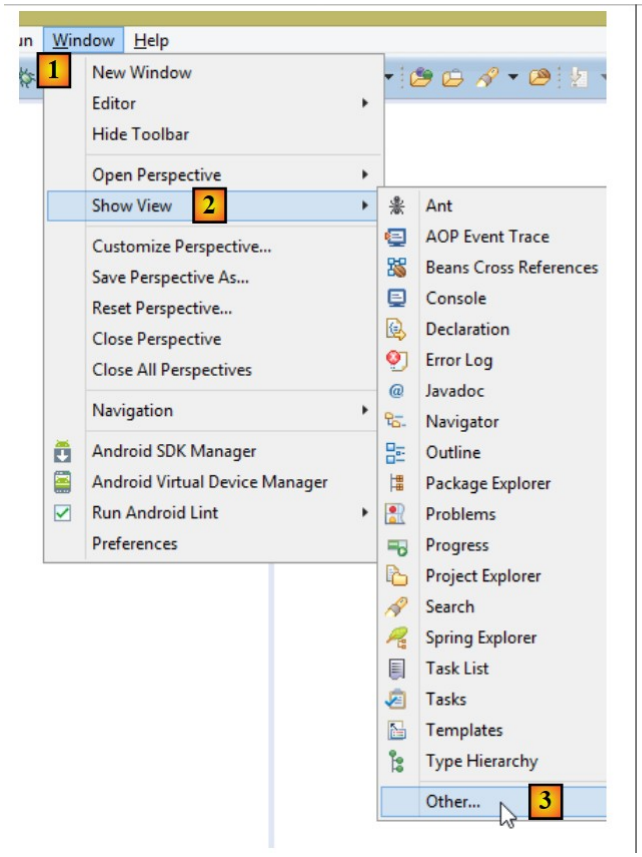
```
1. package st.istia;
2.
3. public class Test01 {
4.
5.   public static void main(String[] args) {
6.     System.out.println("test01");
7.   }
8.
9. }
```

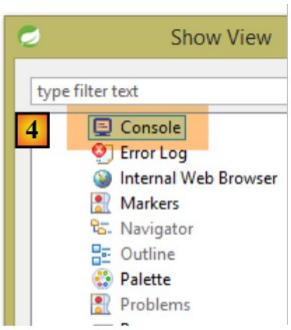


- en [18-20], exécutez la classe. Sa méthode [main] va alors être exécutée ;

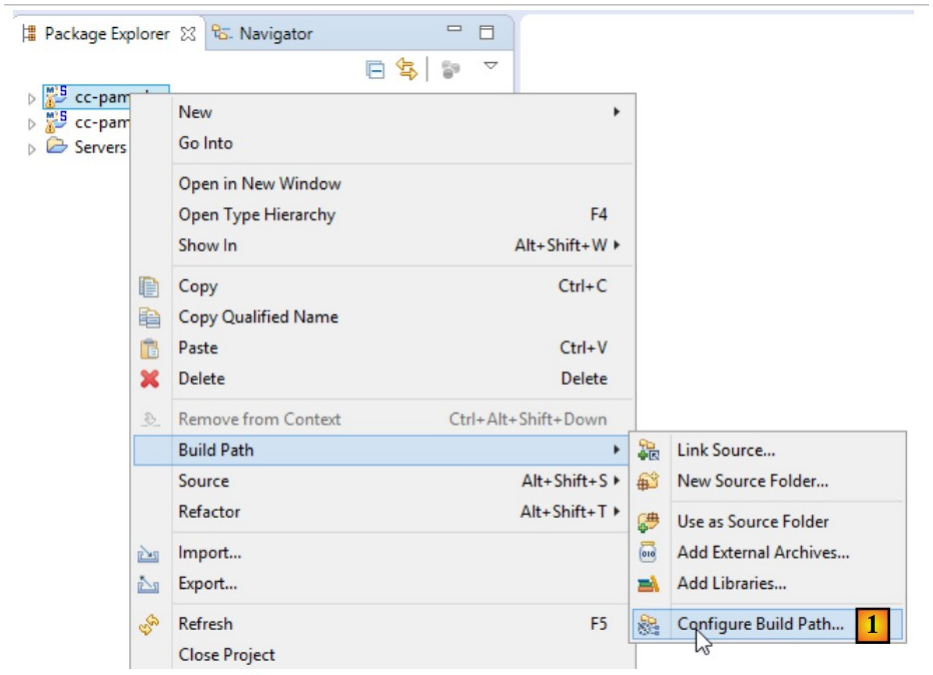
- en [21-22], le résultat de l'application ;

Si la vue [Console] n'est pas présente, procédez comme suit [1-4] :

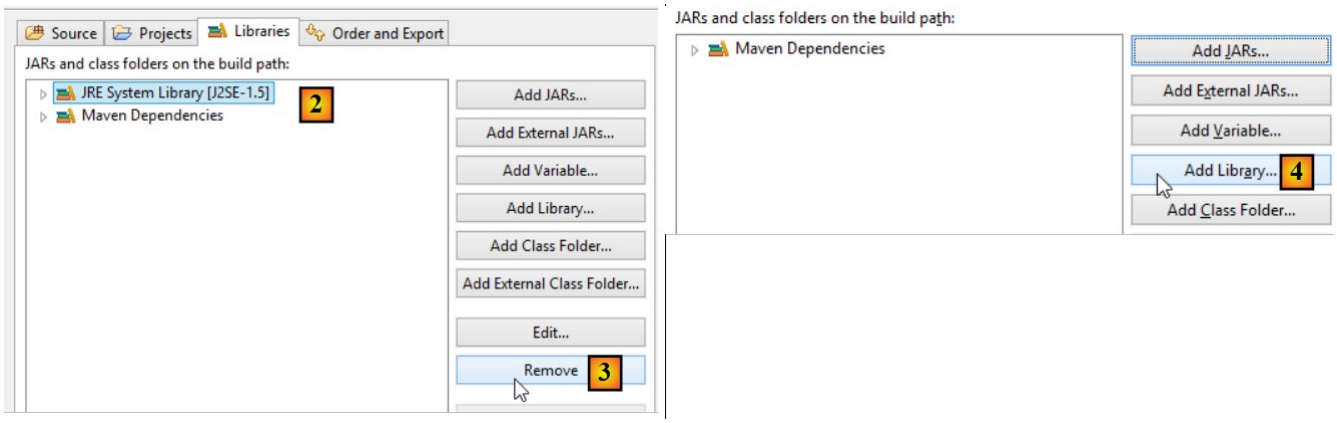




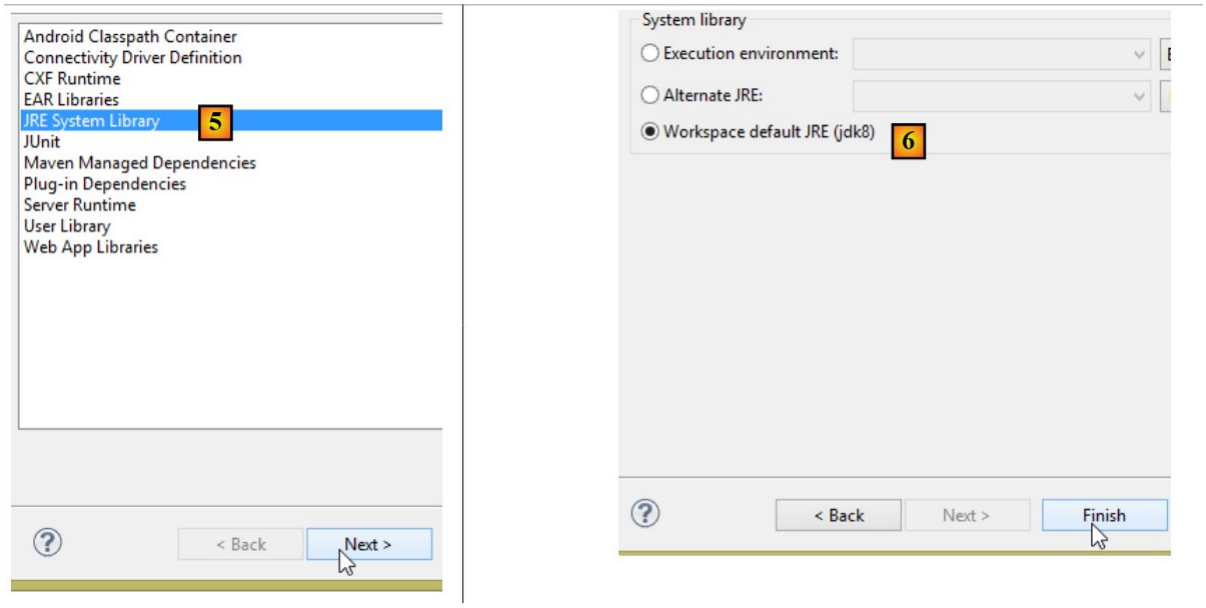
Il se peut qu'à l'importation d'un projet Eclipse, celui-ci présente des erreurs. Cela peut être dû à une configuration incorrecte du projet. Pour corriger l'erreur (si erreur il y a), procédez ainsi :



- en [1], modifiez le [Build Path] du projet ;

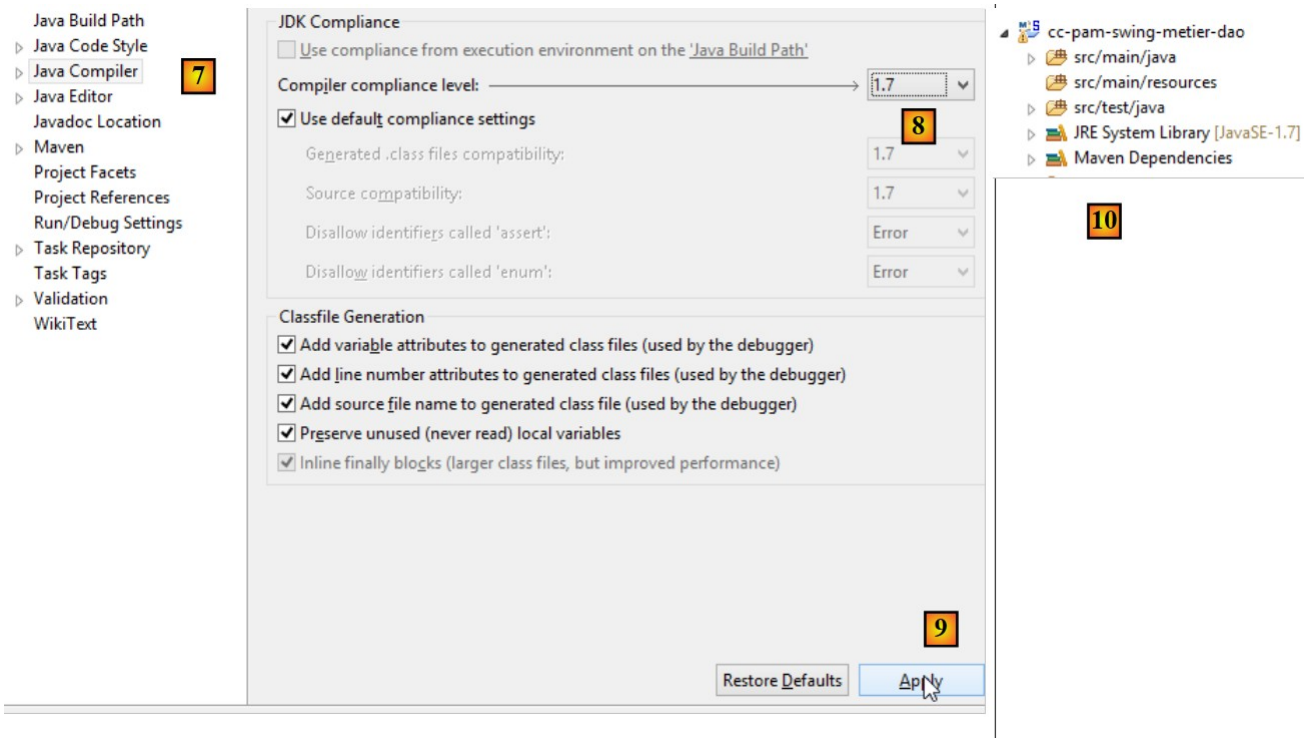


- en [2], le projet est configuré pour utiliser une JVM 1.5 ;
- en [3], supprimez cette dépendance ;
- en [4], ajoutez une nouvelle dépendance ;



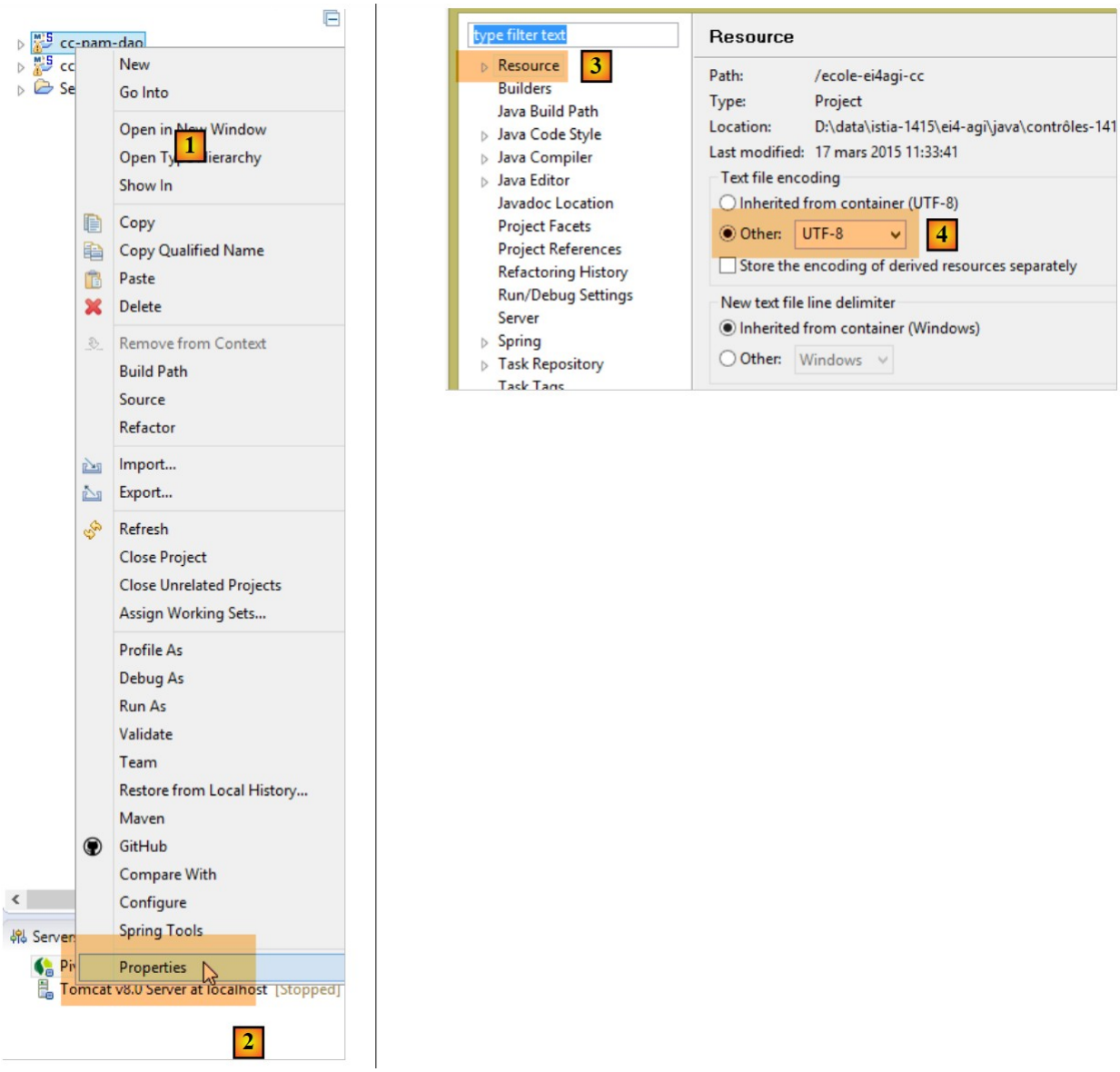
- en [5], on ajoute une JVM ;
- en [6], on choisit la JVM du poste ;

Ceci fait, on valide le tout puis on passe à la propriété [Java compiler] du projet [7] :

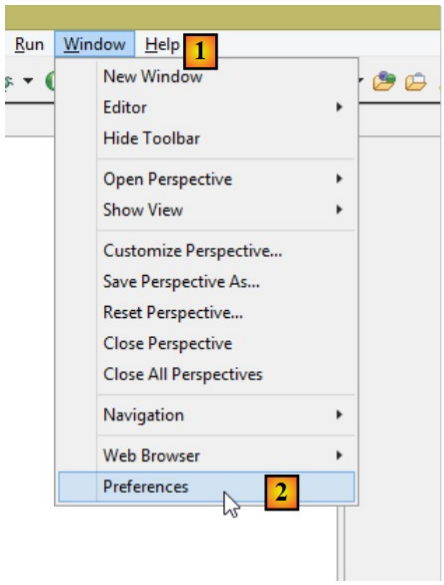


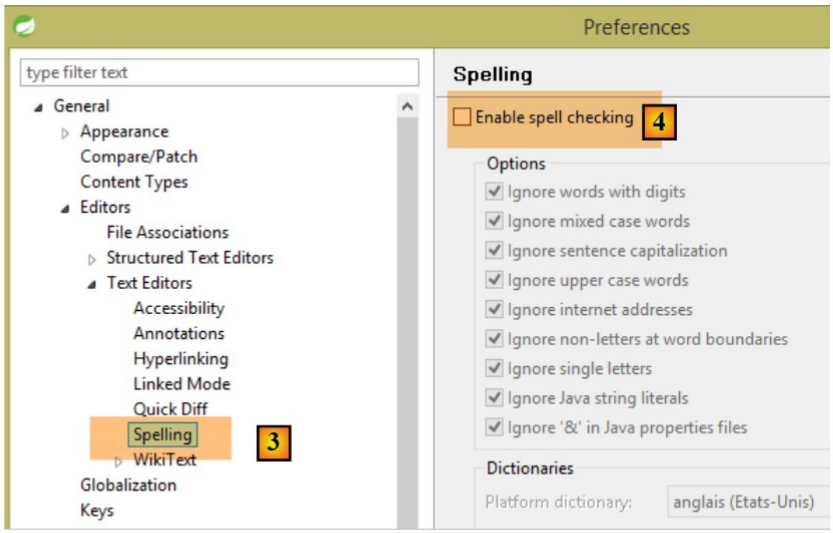
- en [8], on demande au compilateur d'accepter toutes les caractéristiques du langage Java jusqu'à sa version 1.7 (ou 1.8) comprise ;
- en [9], on valide ;
- en [10], le projet ainsi reconfiguré ne doit plus présenter d'erreurs ;

Par ailleurs, le projet importé peut utiliser un encodage UTF-8 des caractères. Procédez ainsi pour fixer cet encodage dans le projet importé [1-4] :



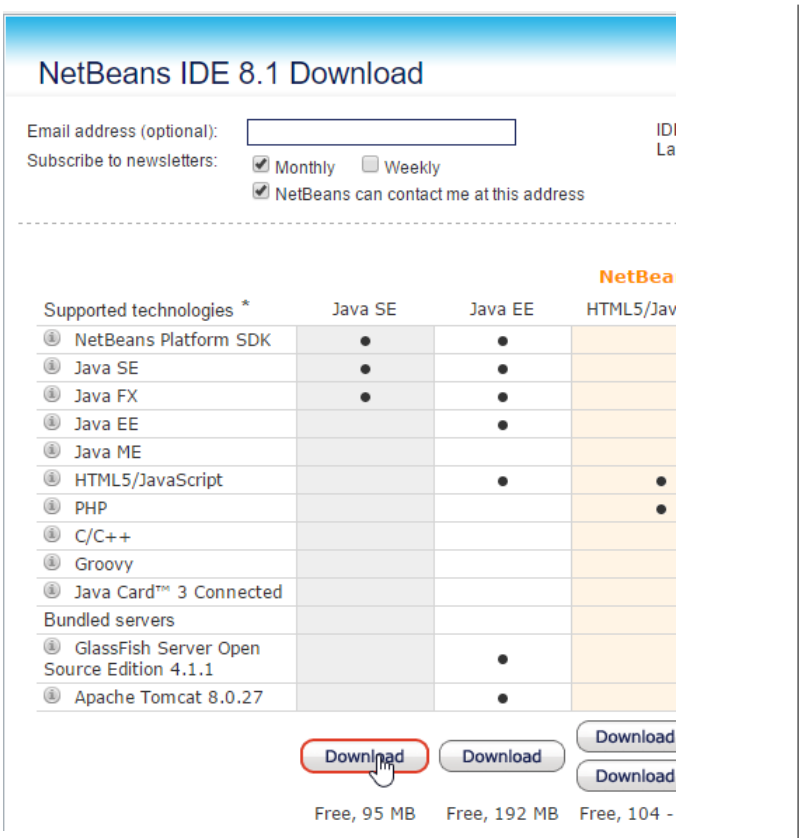
Par ailleurs, il peut être utile d'inhiber la vérification orthographique dans le projet pour éviter que les commentaires en français soient soulignés comme étant incorrects. Suivez les étapes [1-4] ci-dessous :





22.4 Installation de l'IDE Netbeans

Netbeans est disponible à l'URL [http://netbeans.org/downloads/].

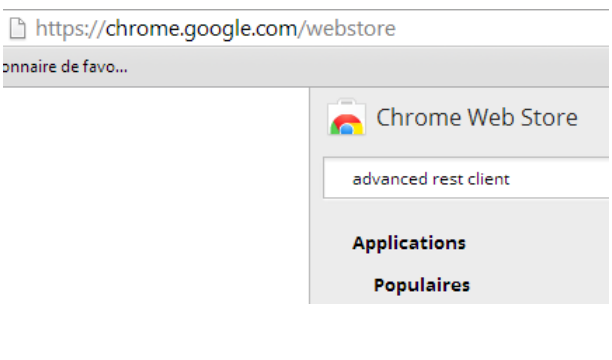


On pourra prendre ci-dessus, la version Java SE (Standard Edition).

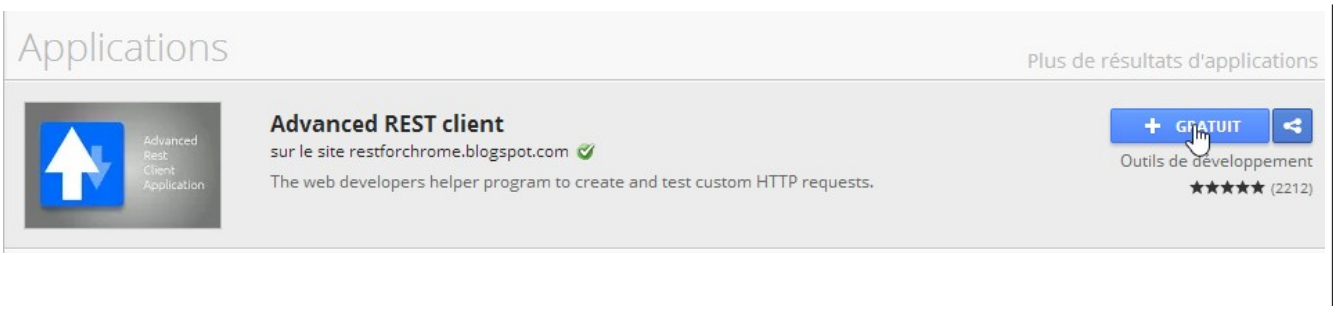
22.5 Installation du plugin Chrome [Advanced Rest Client]

Dans ce document, on utilise le navigateur **Chrome** de Google (http://www.google.fr/intl/fr/chrome/browser/). On lui ajoutera l'extension [Advanced Rest Client]. On pourra procéder ainsi :

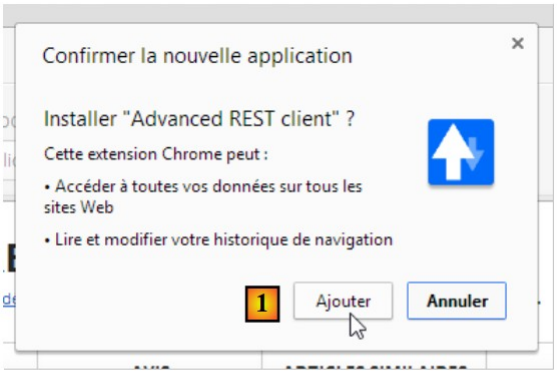
- aller sur le site de [Google Web store] (https://chrome.google.com/webstore) avec le navigateur Chrome ;
- chercher l'application [Advanced Rest Client] :

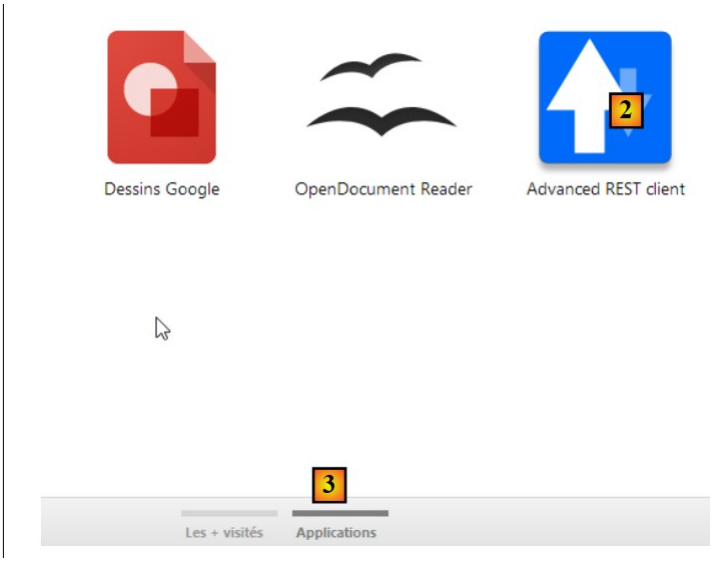


- l'application est alors disponible au téléchargement :



- pour l'obtenir, il vous faudra créer un compte Google. [Google Web Store] demande ensuite confirmation [1] :





- en [2], l'extension ajoutée est disponible dans l'option [Applications] [3]. Cette option est affichée sur chaque nouvel onglet que vous créez (CTRL-T) dans le navigateur.

22.6 Gestion du jSON en Java

De façon transparente pour le développeur le framework [Spring MVC] utilise la bibliothèque jSON [Jackson]. Pour illustrer ce qu'est le jSON (JavaScript Object Notation), nous présentons ici un programme qui sérialise des objets en jSON et fait l'inverse en désérialisant les chaînes jSON produites pour recréer les objets initiaux.

La bibliothèque 'Jackson' permet de construire :

- la chaîne jSON d'un objet : *new ObjectMapper().writeValueAsString(object)* ;
- un objet à partir d'un chaîne jSON : *new ObjectMapper().readValue(jsonString, Object.class)*.

Les deux méthodes sont susceptibles de lancer une *IOException*. Voici un exemple.

Le projet ci-dessus est un projet Maven avec le fichier [pom.xml] suivant ;

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <project xmlns="http://maven.apache.org/POM/4.0.0"
3.          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4.          xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
5.   <modelVersion>4.0.0</modelVersion>
6.
7.   <groupId>istia.st.pam</groupId>
8.   <artifactId>json</artifactId>
9.   <version>1.0-SNAPSHOT</version>
10.
11.    <dependencies>
12.      <dependency>
13.        <groupId>com.fasterxml.jackson.core</groupId>
14.        <artifactId>jackson-databind</artifactId>
15.        <version>2.3.3</version>
16.      </dependency>
17.    </dependencies>
18. </project>
```

- lignes 12-16 : la dépendance qui amène la bibliothèque 'Jackson' ;

La classe [Personne] est la suivante :

```
1. package istia.st.json;
2.
3. public class Personne {
4.   // data
5.   private String nom;
6.   private String prenom;
7.   private int age;
8.
9.   // constructeurs
10.    public Personne() {
11.
12.    }
13.
14.    public Personne(String nom, String prénom, int âge) {
15.        this.nom = nom;
16.        this.prenom = prénom;
17.        this.age = âge;
18.    }
19.
20.    // signature
21.    public String toString() {
22.        return String.format("Personne[%s, %s, %d]", nom, prenom, age);
23.    }
24.
25.    // getters et setters
26.  ...
27. }
```

La classe [Main] est la suivante :

```
1. package istia.st.json;
2.
3. import com.fasterxml.jackson.databind.ObjectMapper;
4.
5. import java.io.IOException;
6. import java.util.HashMap;
7. import java.util.Map;
8.
9. public class Main {
```

```
10.    // l'outil de sérialisation / désérialisation
11.    static ObjectMapper mapper = new ObjectMapper();
12.
13.    public static void main(String[] args) throws IOException {
14.      // création d'une personne
15.      Personne paul = new Personne("Denis", "Paul", 40);
16.      // affichage jSON
17.      String json = mapper.writeValueAsString(paul);
18.      System.out.println("Json=" + json);
19.      // instanciation Personne à partir du Json
20.      Personne p = mapper.readValue(json, Personne.class);
21.      // affichage personne
22.      System.out.println("Personne=" + p);
23.      // un tableau
24.      Personne virginie = new Personne("Radot", "Virginie", 20);
25.      Personne[] personnes = new Personne[]{paul, virginie};
26.      // affichage Json
27.      json = mapper.writeValueAsString(personnes);
28.      System.out.println("Json personnes=" + json);
29.      // dictionnaire
30.      Map<String, Personne> hpersonnes = new HashMap<String, Personne>();
31.      hpersonnes.put("1", paul);
32.      hpersonnes.put("2", virginie);
33.      // affichage Json
34.      json = mapper.writeValueAsString(hpersonnes);
35.      System.out.println("Json hpersonnes=" + json);
36.    }
37.  }
```

L'exécution de cette classe produit l'affichage écran suivant :

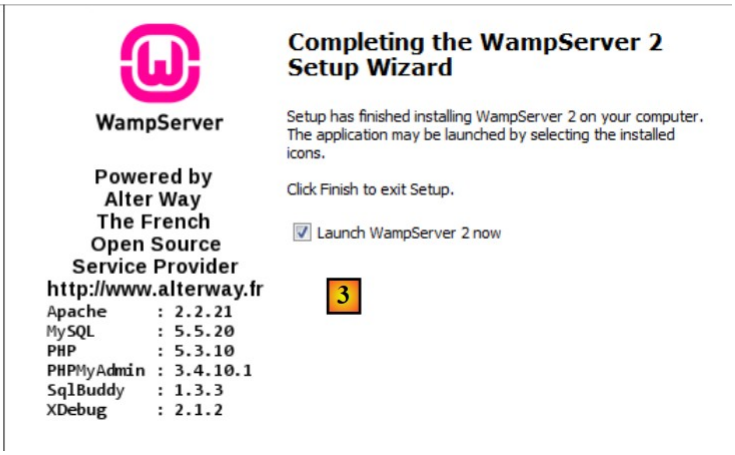
```
1. Json={"nom":"Denis","prenom":"Paul","age":40}
2. Personne=Personne[Denis, Paul, 40]
3. Json personnes=[{"nom":"Denis","prenom":"Paul","age":40},{"nom":"Radot","prenom":"Virginie","age":20}]
4. Json hpersonnes={"2":{"nom":"Radot","prenom":"Virginie","age":20},"1":{"nom":"Denis","prenom":"Paul","age":40}}
```

De l'exemple on retiendra :

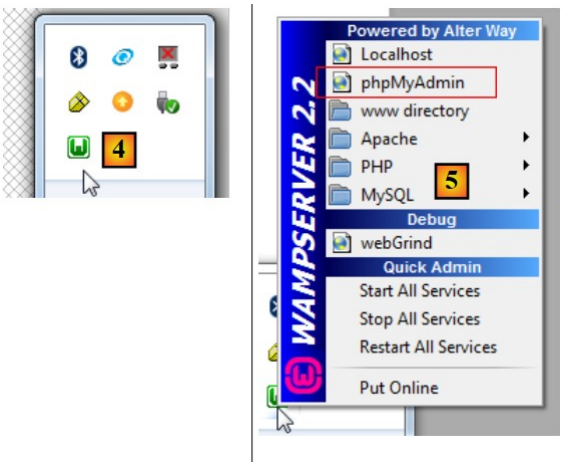
- l'objet [ObjectMapper] nécessaire aux transformations jSON / Object : ligne 11 ;
- la transformation [Personne] --> jSON : ligne 17 ;
- la transformation jSON --> [Personne] : ligne 20 ;
- l'exception [IOException] lancée par les deux méthodes : ligne 13.

22.7 Installation de [WampServer]

[WampServer] est un ensemble de logiciels pour développer en PHP / MySQL / Apache sur une machine Windows. Nous l'utiliserons uniquement pour le SGBD MySQL.



- sur le site de [WampServer] [1], choisir la version qui convient [2],
- l'exécutable téléchargé est un installateur. Diverses informations sont demandées au cours de l'installation. Elles ne concernent pas MySQL. On peut donc les ignorer. La fenêtre [3] s'affiche à la fin de l'installation. On lance [WampServer],



- en [4], l'icône de [WampServer] s'installe dans la barre des tâches en bas et à droite de l'écran [4],
- lorsqu'on clique dessus, le menu [5] s'affiche. Il permet de gérer le serveur Apache et le SGBD MySQL. Pour gérer celui-ci, on utiliser l'option [PhpPmyAdmin],
- on obtient alors la fenêtre ci-dessous,

Nous donnerons peu de détails sur l'utilisation de [PhpMyAdmin]. Nous montrons au paragraphe 6.4.2, page 85, comment l'utiliser pour créer une base de données à partir d'un script SQL.