



Rapport Java JEE

Objet : *Inversion de controle et injection des dependances*

❖ Code source :

<https://github.com/DAHMANE-git/InjectionDesDependances>

❖ Présentation :

1 - Methode statique avec fichier texte :

On crée un fichier texte nommé config.txt où on mentionne les noms des classes qu'on souhaite traiter. Dans ce cas, on a les deux classes dao.DaoImpl et metier.MetierImpl. Au niveau de la classe Présentation on déclare un scanner qui se charge de la lecture des lignes à partir de notre fichier de configuration config.txt. Après il commence à stocker les noms des classes récupérés ligne par ligne dans des objets Class en utilisant la méthode Class.forName() et qui implémentent par la suite les interfaces correspondantes. Ensuite, on fait appel à la fonction getMethod() afin d'invoquer la méthode setDao() de la classe MetierImpl, donc elle prend en paramètres le nom de la méthode (une chaîne de caractères) et l'objet. Finalement, on invoque cette méthode sur l'objet metier et on lui transmet comme argument l'objet dao. Par la suite, on peut afficher le résultat de la méthode calcul().

```
DaoImpl.java  IDao.java  DaoImpl.java  DaoImplV
1 dao.DaoImpl
2 metier.MetierImpl|
```

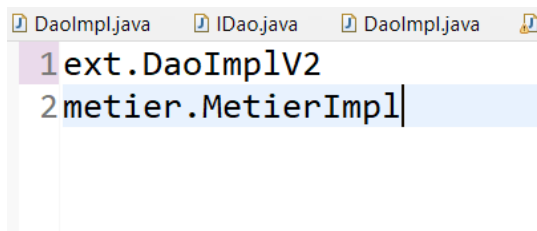
On peut utiliser une autre classe nommée DaoImplV2 comme une deuxième version et il suffit juste de changer le nom de la classe au niveau du fichier de configuration sans avoir besoin de changer le code.

Nom & prénom : DAHMANE walid

Classe : 4IIR G2

Site : CENTRE

```
3*import org.springframework.stereotype.Component;
7
8 @Repository
9 public class DaoImplV2 implements IDao{
10
11     @Override
12     public double getData() {
13         /*
14          * Version WS
15          */
16         System.out.println("version web service");
17         double data=12;
18         return data;
19     }
20 }
21 }
```



➔ Synthèse :

Le Framework **Spring** permet de construire et définir l'infrastructure d'une application Java, d'où il nous facilite le développement. Il est effectivement un conteneur léger. Il prend en charge la création d'objets et la mise en relation d'objets par l'intermédiaire d'un fichier de configuration qui décrit les objets à fabriquer et les relations de dépendances entre ces objets ou bien à travers les annotations. Le gros avantage par rapport aux serveurs d'application est qu'avec **Spring**, les classes n'ont pas besoin d'implémenter une quelconque interface pour être prises en charge par le Framework. Il s'appuie principalement sur l'intégration de trois concepts clés :

- ❖ L'inversion de contrôle est assurée de deux façons différentes : la recherche de dépendances et l'injection de dépendances.
- ❖ La programmation orientée aspect.
- ❖ Une couche d'abstraction.

2 - Methode dynamique avec fichier XML :

Spring utilise un système de Bean. Un Bean est un objet qui est instancié, assemblé et géré par Spring IoC Container.

Nom & prénom : DAHMANE walid

Classe : 4IIR G2







Site : CENTRE

IoC (Inversion of control), est un processus qui définit les dépendances d'un objet sans avoir à les créer. C'est lors de la création des objets, que Spring va Injecter les Beans entre eux afin d'avoir toutes leurs dépendances. Chaque bean représente un objet et on lui attribue un Id ainsi que le nom de la classe. Pour faire l'injection des dépendances on aura besoin de la méthode setDao, chose pour laquelle on rajoute une propriété afin de pouvoir invoquer cette méthode.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xsi:schemaLocation="http://www.springframework.org/schema/beans
5     http://www.springframework.org/schema/beans/spring-beans.xsd">
6     <bean id="d" class="dao.DaoImpl"></bean>
7     <bean id="metier" class="metier.MetierImpl">
8         <property name="dao" ref="d"></property>
9     </bean>
10 </beans>
```

Par contre puisque on travaille dans le contexte d'une application JAVA normale sans maven on doit installer quelques jars pour que notre application fonctionne.

Referenced Libraries

- >  spring-beans-4.3.13.RELEASE.jar
- >  spring-context-4.3.13.RELEASE.jar
- >  spring-core-4.3.13.RELEASE.jar
- >  commons-logging-1.1.3.jar
- >  spring-expression-4.3.13.RELEASE.jar
- >  spring-aop-4.3.13.RELEASE.jar

➔ Test applicatif

```
public static void main(String[] args) {
    ApplicationContext ctx=
        new ClassPathXmlApplicationContext("config.xml");
    IMetier metier= (IMetier) ctx.getBean(IMetier.class);
    System.out.println(metier.calcul());
}
```

Nom & prénom : DAHMANE walid

Classe : 4IIR G2

Site : CENTRE

3 - Methode dynamique avec fichier les annotations :

@Repository : marque toutes les classes qui remplissent le role d'un référentiel, également noté objet d'accès aux données ou DAO

```
1 package dao;
2
3 import org.springframework.stereotype.Component;
4
5 @Component("dao")
6 public class DaoImpl implements IDao{
7
8     @Override
9     public double getData() {
10         /*
11          * je me connecte à la bd
12          */
13         double data=98;
14         return data;
15     }
16
17     public void init() {
18         System.out.println("Instanciation de DaoImpl");
19     }
20
21 }
```

```
1 package ext;
2
3 import org.springframework.stereotype.Component;
4
5 @Repository
6 public class DaoImplV2 implements IDao{
7
8     @Override
9     public double getData() {
10         /*
11          * Version WS
12          */
13         System.out.println("version web service");
14         double data=12;
15         return data;
16     }
17
18 }
```

@Service : definie les Bean qui contiennent la logique metier et les méthodes d'appel dans la couche de référentiel. Son injection peut de faire a l'aide de l'annotation @autowired.

```
9 @Service("metier")
10 public class MetierImpl implements IMetier{
11     /*
12      * Couplage faible
13      */
14     @Autowired
15     private IDao dao=null;
16     @Override
17
18     public double calcul() {
19         double d=dao.getData();
20         double res=d*23;
21         return res;
22     }
23
24     public void setDao(IDao dao) {
25         this.dao = dao;
26         //System.out.println("Injection des dépendances");
27     }
28 }
```