

TP : IOC

Principe de l'inversion de contrôle et injection des dépendances

Contents

Introduction :.....	2
Couplage fort :.....	2
Couplage faible :.....	2
CONCEPTION :.....	3
Partie de code :.....	4
Inversion of Control container (IoC)	5
Injection de dépendances	5
Injection des dépendances de manier statique :.....	6
Injection des dépendances de manier dynamique :.....	7
SPRING	9
Inversion of Control container (IoC) avec SPRING	9
Injection des dépendances avec SPRING (cas pratique)	10
Version Xml.....	10
Version annotation	12
Conclusion	15

Introduction :

Parmi les principes de conception on trouve :

- ❖ **Une application qui n'évolue pas meurt.**
- ❖ **Une application doit être fermée à la modification et ouverte à l'extension.**
- ❖ **Une application doit s'adapter aux changements**

Mais pour atteindre ces objectifs il faut suivre une démarche bien précise et connaître un ensemble de connaissances et pour commencer ce parcours en commençant par répondre à ces questions :

- Comment Créer une application fermée à la modification et ouverte à l'extension ?
- Comment faire l'injection des dépendances ?
- C'est quoi le principe de l'inversion de contrôle ?

Une application doit être fermée à la modification et ouverte à l'extension. Mais que signifie ça ?

Cela veut dire qu'ajouter de nouvelles fonctionnalités ne devrait pas casser les fonctionnalités déjà existantes.

Cela veut dire aussi qu'on ne touche pas le code source déjà écrit, on y ajoute seulement.

Une application ouverte à la modification doit se baser sur une bonne conception qui profite du polymorphisme pour établir **un couplage faible** entre les différentes entités, cela facilite par la suite l'étendue de l'application et l'ajout ou la modification des fonctionnalités.

Mais d'abord il faut comprendre la différence entre le couplage faible et fort :

Couplage fort :

Quand une classe A est liée à une classe B, on dit que la classe A est fortement couplée à la classe B.

La classe A ne peut fonctionner qu'en présence de la classe B.

Si une nouvelle version de la classe B (soit B2), est créée, on est obligé de modifier dans la classe A.

Couplage faible :

Pour utiliser le couplage faible, nous devons utiliser les interfaces.

Considérons une classe A qui implémente une interface IA, et une classe B qui implémente une interface IB.

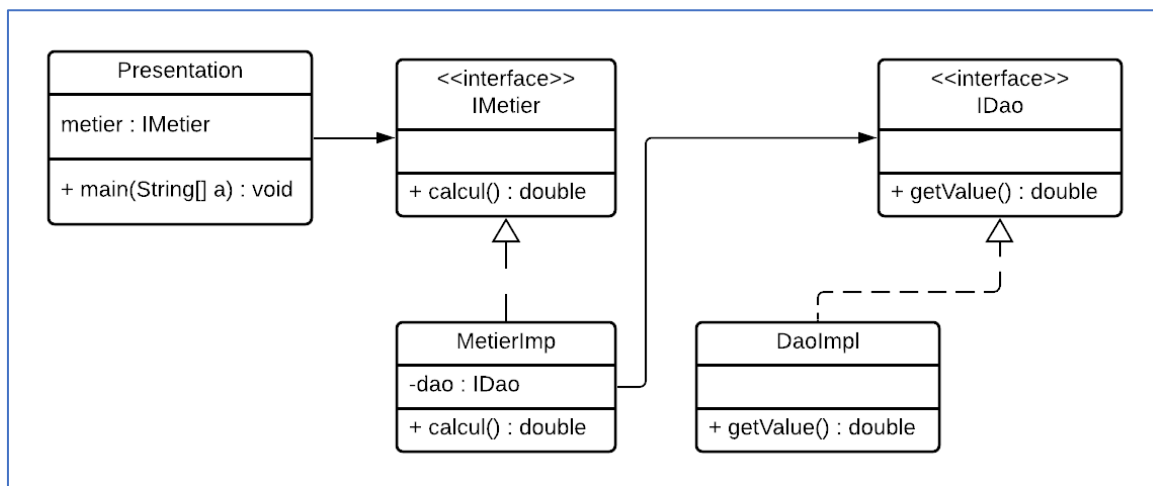
Si la classe A est liée à l'interface IB par une association, on dit que la classe A et la classe B sont liées par un couplage faible.

Cela signifie que la classe B peut fonctionner avec n'importe quelle classe qui implémente l'interface IA.

Remarque : pour illustrer tous les concepts dans ce rapport on va travailler sur un projet Maven dans laquelle on va implémenter une application qui nous retourne le degré de température selon différentes sources d'information

Remarque : Maven est un outil de construction de projets (build) open source développé par la fondation Apache, initialement pour les besoins du projet Jakarta Turbine. Il permet de faciliter et d'automatiser certaines tâches de la gestion d'un projet Java.

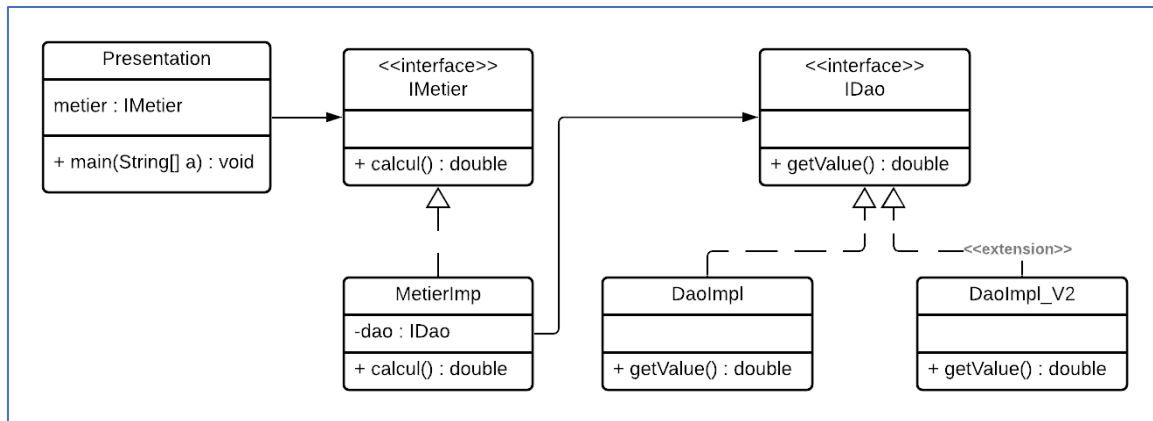
CONCEPTION :



La conception ci-dessous implémente les notions mentionnées dans l'introduction, alors maintenant si on veut modifier la méthode **getValue()** il suffit d'ajouter une nouvelle classe qui dépend de l'interface **IDao** et redéfinir la méthode **getValue()**.

Aucun problème sera rencontré puisque la classe **MetierImpl** peut fonctionner avec n'importe quelle classe qui implémente l'interface **IDao**.

Voici comment si facile étendre les fonctionnalités de notre application sans toucher le code déjà écrit.



Dans ce cas on peut facilement utiliser l'un des deux implémentations de Dao et on peut ajouter d'autres versions d'implémentations et sans modifier le code source initial de l'application mais seulement avec l'ajout des extensions

Partie de code :

La création de l'interface IDao qui contient la méthode `getData()` et une implémentation de cette interface

```
package dao;

public interface IDao {
    double getData();
}
```

```
package dao;

public class DaoImpl implements IDao{
    @Override
    public double getData() {
        /*
         * se connecter a la BD pour recuperer la temperature
         */
        System.out.println("version BD");
        double temp = Math.random()*40;
        return temp;
    }
}
```

DaoImpl est la première implémentation de l'interface IDao on va la nommée version DB (base de donnée) car il nous permet de récupérer la température a partir d'une base de donnée

La création de l'interface IMetier qui contient la méthode calcul() et une implémentation de cette interface

```
package metier;
```



```
public interface IMetier {  
    double calcul();  
}
```

```
package metier;
```

```
import ...
```

```
public class MetierImpl implements IMetier{  
    private IDao dao;  
    @Override  
    public double calcul() {  
        double temp = dao.getData();  
        double res=temp*540/Math.cos(temp*Math.PI);  
        return res;  
    }  
    public MetierImpl(IDao dao) {  
        this.dao = dao;  
    }  
    public void setDao(IDao dao) {  
        this.dao = dao;  
    }  
}
```

La méthode calcul() récupère les données de la méthode getData() et avec une formule il nous permet d'avoir le résultat qui est la température

Inversion of Control container (IoC)

Il s'agit d'un mécanisme qui facilite la mise en place des dépendances par l'injection automatique des objets. Il permet au développeur de se concentrer sur le code métier et le Framework va s'occuper du code technique

Injection de dépendances

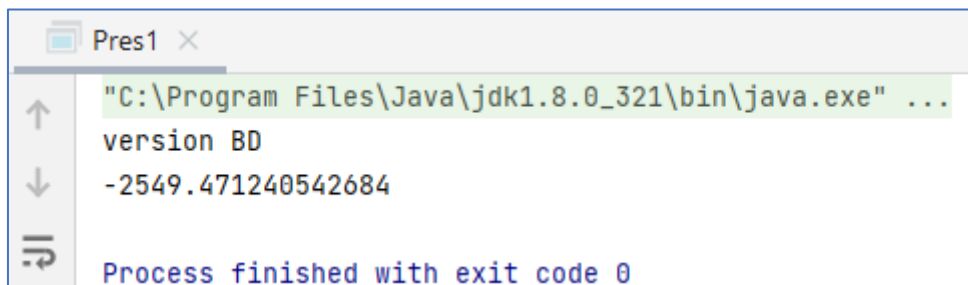
L'injection de dépendances est un mécanisme qui permet d'implémenter le principe de l'inversion de contrôle. Il consiste à créer dynamiquement les dépendances entre les différents objets en s'appuyant sur une description ou de manière programmatique

Injection des dépendances de manier statique :

La création de class présentation version 1

```
package pres;  
import ...  
public class Pres1 {  
    public static void main(String[] args) {  
        /*  
         * Injection des dependances par  
         * instantiation statique  
         */  
        try{  
            DaoImpl dao = new DaoImpl();  
            MetierImpl metier = new MetierImpl();  
            metier.setDao(dao);  
            System.out.println(metier.calcul());  
        }catch (Exception e){  
            e.printStackTrace();  
        }  
    }  
}
```

Voila le code de première méthode de l'injection des dépendances (méthode statique) en utilisant la mot clé **new**



Le résultat de la version base de données

Maintenant on veut récupérer la température en utilisant des capteurs et non pas les données de base de données donc on doit créer une nouvelle package extension qui contient la nouvelle méthode de calcul cette nouvelle class qui doit implémenter l'interface IDao on va la nommée DaoImplVC (Dao implémentation version capteur)

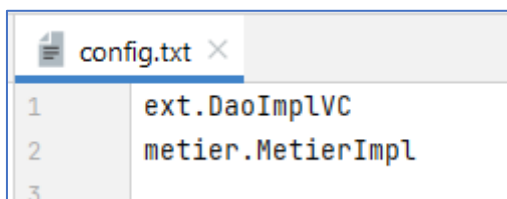
```
package ext;
import ...

public class DaoImplVC implements IDao {
    @Override
    public double getData() {
        System.out.println("versions capteurs");
        double temp = 80;
        return temp;
    }
}
```

Mais la question est : est ce que on peut utiliser la nouvelle version sans modifier le code source de notre application la réponse est non donc pour résoudre de problème on va utiliser instantiation dynamique

Injection des dépendances de manier dynamique :

La première action est de crier un fichier de configuration qui contient les classes qu'on veut injecter



```
config.txt x
1 ext.DaoImplVC
2 metier.MetierImpl
3
```

Et après dans la présentation on va lire ce fichier et instancier ces class de manier dynamique

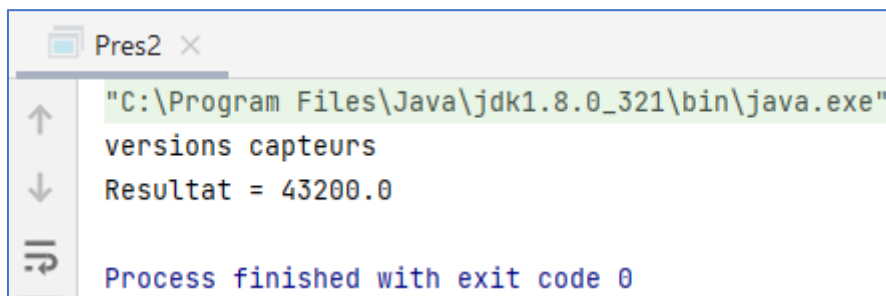
```
package pres;
import ...
public class Pres1 {
    public static void main(String[] args) {
        try{
            Scanner scanner = new Scanner(new File( pathname: "config.txt"));

            String daoClassName = scanner.nextLine();
            Class cDao = Class.forName(daoClassName);
            IDao dao = (IDao) cDao.newInstance();

            String metierClassName = scanner.nextLine();
            Class cMetier = Class.forName(metierClassName);
            IMetier metier =(IMetier) cMetier.newInstance();

            Method method = cMetier.getMethod( name: "setDao", IDao.class);
            method.invoke(metier,dao);

        }catch (Exception e){
            e.printStackTrace();
        }
    }
}
```



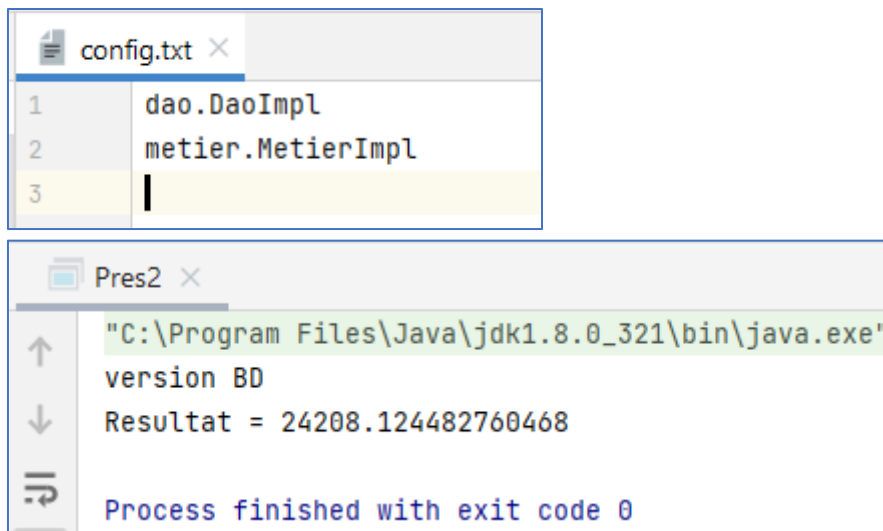
```
Pres2 x
"C:\Program Files\Java\jdk1.8.0_321\bin\java.exe"
versions capteurs
Resultat = 43200.0
Process finished with exit code 0
```

Donc en arriver à utiliser la version capteurs avec succès

Mais imaginons que notre client il veut revenir à la première version de base de données est ce que cette opération est possible et sans modification de code source

La réponse est oui. Avec l'instanciation dynamique on peut facilement revenir à la première version il faut seulement modifier le contenu du fichier config.txt et changer la ligne qui contient le nom de la

ext.DaoImplVC par la version que tu veux et dans notre cas est dao.DaoImpl



```
config.txt x
1 dao.DaoImpl
2 metier.MetierImpl
3

Pres2 x
"C:\Program Files\Java\jdk1.8.0_321\bin\java.exe"
version BD
Resultat = 24208.124482760468
Process finished with exit code 0
```

Donc jusqu'à maintenant on est arrivé à créer une application qui est fermée à la modification et ouverte à l'extension, mais il existe des Frameworks qui nous permettent de faciliter ce travail.

C'est le principe de l'Inversion de contrôle : ce dernier permet au développeur de s'occuper uniquement du code métier (Exigences fonctionnelles) et c'est le Framework qui s'occupe du code technique (Exigences Techniques).

Dans notre cas, on va utiliser Spring IOC.

SPRING

Spring est un Framework qui facilite la maîtrise du cycle de vie de l'application grâce à de nombreuses bonnes pratiques de développement.

Le concept fondamental du Spring est l'injection des dépendances (*dependency injection*). C'est une réponse à 3 facteurs du mauvais codage, définis en 1994 par Robert C. Martin :

- ✓ Rigidité : il est difficile de changer quoi que ce soit car chaque changement affecte trop le fonctionnement du système.
- ✓ Fragilité : un changement provoque le mauvais fonctionnement des parties qui devraient fonctionner correctement.
- ✓ Immobilité : il est difficile de réutiliser le code dans une autre application.

Ces 3 problèmes sont appelés *RFI*. L'Injection des dépendances est une solution à cette problématique.

Inversion of Control container (IoC) avec SPRING

Sous Spring, les objets s'appellent des beans. L'injection des dépendances automatique peut se faire à travers les fichiers XML aussi bien qu'à travers des annotations. Le rôle du container consiste donc à regrouper les beans avec le modèle des données. Ce regroupement produit ensuite le système prêt à être déployé. Le container est donc un contexte d'application (application context) qui gère des beans.

Injection des dépendances avec SPRING (cas pratique)

D'abord pour travailler avec il faut l'ajouter comme des dépendances dans le fichier pom.xml

Il faut 3 dépendances qui sont : spring-core , spring-context et spring-beans



```
m pom.xml (enset-ioc-2) x
15
16 <dependencies>
17 <!-- https://mvnrepository.com/artifact/org.springframework/spring-core -->
18 <dependency>
19 <groupId>org.springframework</groupId>
20 <artifactId>spring-core</artifactId>
21 <version>5.3.15</version>
22 </dependency>
23 <!-- https://mvnrepository.com/artifact/org.springframework/spring-context -->
24 <dependency>
25 <groupId>org.springframework</groupId>
26 <artifactId>spring-context</artifactId>
27 <version>5.3.15</version>
28 </dependency>
29 <!-- https://mvnrepository.com/artifact/org.springframework/spring-beans -->
30 <dependency>
31 <groupId>org.springframework</groupId>
32 <artifactId>spring-beans</artifactId>
33 <version>5.3.15</version>
34 </dependency>
```

Version Xml

Spring lit le fichier spring-ioc.xml et à base de lui il crée les nouveaux beans et injecter les dépendances entre eux.

Pour utiliser l'injection des dépendances avec spring version xml il faut crée un fichier xml nommée ApplicationContexte.xml



```
ApplicationContexte.xml x
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     <beans>
7         <bean id="dao" class="dao.DaoImpl"></bean>
8         <bean id="metier" class="metier.MetierImpl">
9             <property name="dao" ref="dao"></property>
10        </bean>
11    </beans>
12</beans>
```

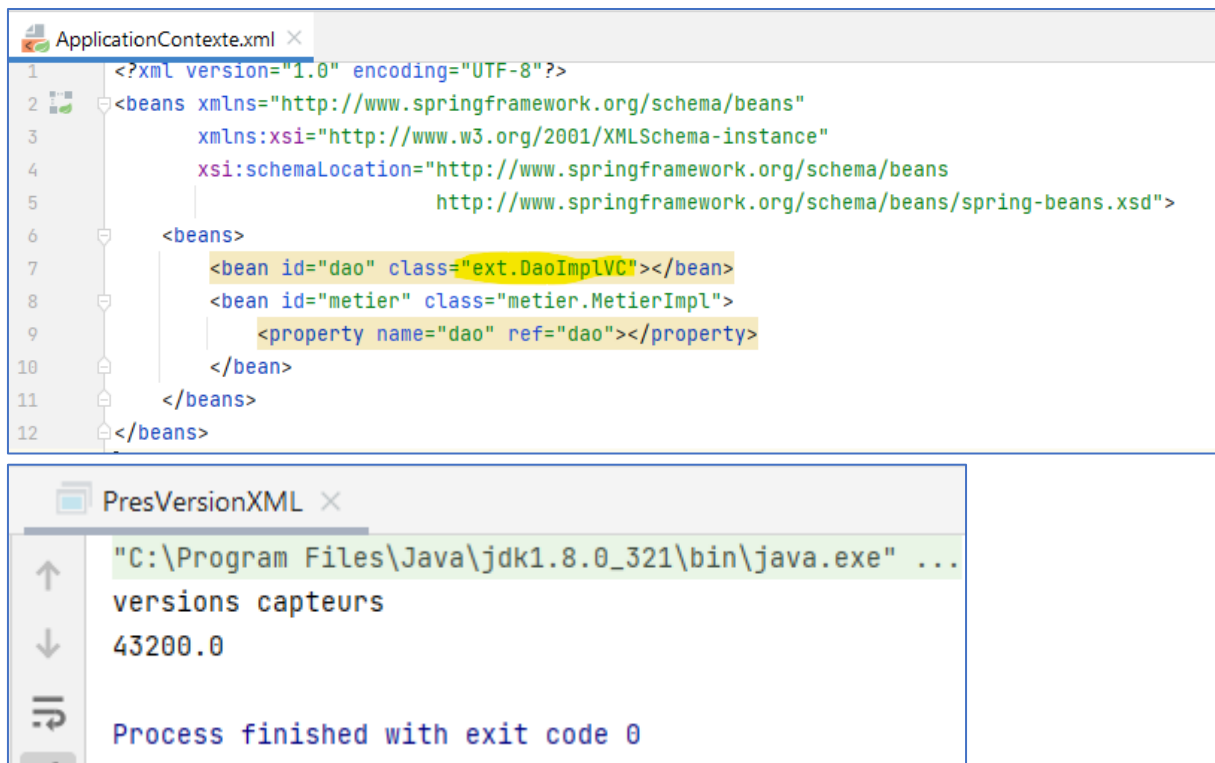
Maintenant on va une crée une présentation pour la version xml

```
PresVersionXML.java x
1 package pres;
2
3 import metier.IMetier;
4 import org.springframework.context.ApplicationContext;
5 import org.springframework.context.support.ClassPathXmlApplicationContext;
6
7 public class PresVersionXML {
8     public static void main(String[] args) {
9         ApplicationContext context=new
10             ClassPathXmlApplicationContext( configLocation: "ApplicationContexte.xml");
11         IMetier metier=(IMetier) context.getBean( s: "metier");
12         System.out.println(metier.calcul());
13     }
14 }
15
```

on peut facilement récupérer le bean désiré depuis le contexte de l'application via la méthode `getBean()`

```
PresVersionXML x
↑ "C:\Program Files\Java\jdk1.8.0_321\bin\java.exe" ...
version BD
↓ -13053.202353774606
↻ Process finished with exit code 0
```

Pour utiliser l'autre version de capteurs il faut seulement modifier le fichier `ApplicationContexte.xml`



```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans.xsd">
  <beans>
    <bean id="dao" class="ext.DaoImplVC"></bean>
    <bean id="metier" class="metier.MetierImpl">
      <property name="dao" ref="dao"></property>
    </bean>
  </beans>
</beans>
```

```
"C:\Program Files\Java\jdk1.8.0_321\bin\java.exe" ...
versions capteurs
43200.0
Process finished with exit code 0
```

Version annotation

Pour utiliser l'injection des dépendances avec spring version annotation on doit respecter quelques règles :

Ajouter l'annotation **@Component("nom")** a chaque class qu'on veut instancier. Le nom est facultatif si tu ne donnes pas un nome spring utilise le nom de la classe

Pour faire l'injection des dépendances on utilise l'annotation **@Autowired**

```
MetierImpl.java x
1 package metier;
2 import ...
6
7 @Component("metier")
8 public class MetierImpl implements IMetier{
9     // @Qualifier("dao")
10    @Autowired
11    private IDao dao;
12    @Override
13    public double calcul() {
14        double temp = dao.getData(); // n'importe quel source:classe
15        double res=temp*540/Math.cos(temp*Math.PI);
16        return res;
17    }
}
```

```
DaoImpl.java x
1 package dao;
2 import org.springframework.stereotype.Component;
3
4 @Component("dao")
5 public class DaoImpl implements IDao{
6     @Override
7     public double getData() {
8         /*
9          * se connecter a la BD pour recuperer la temperature
10         */
11         System.out.println("version BD");
12         double temp = Math.random()*40;
13         return temp;
14     }
15 }
```

Maintenant on va une crée une présentation pour la version annotation

```

PresVersionAnnotation.java x
1 package pres;
2 import ...
6
7 public class PresVersionAnnotation {
8     public static void main(String[] args) {
9         ApplicationContext context=
10             new AnnotationConfigApplicationContext( ...basePackages: "dao","metier");
11         IMetier metier= context.getBean(IMetier.class);
12         System.out.println(metier.calcul());
13     }
14 }

```

Il faut mentionner a spring les packages qui va scanner

Comme c'est le cas avec Spring XML, on peut facilement récupérer le bean désiré depuis le contexte de l'application via la méthode `getBean()`

```

PresVersionAnnotation x
↑ "C:\Program Files\Java\jdk1.8.0_321\bin\java.exe" ...
version BD
↓ 2518.275753059823
↻ Process finished with exit code 0

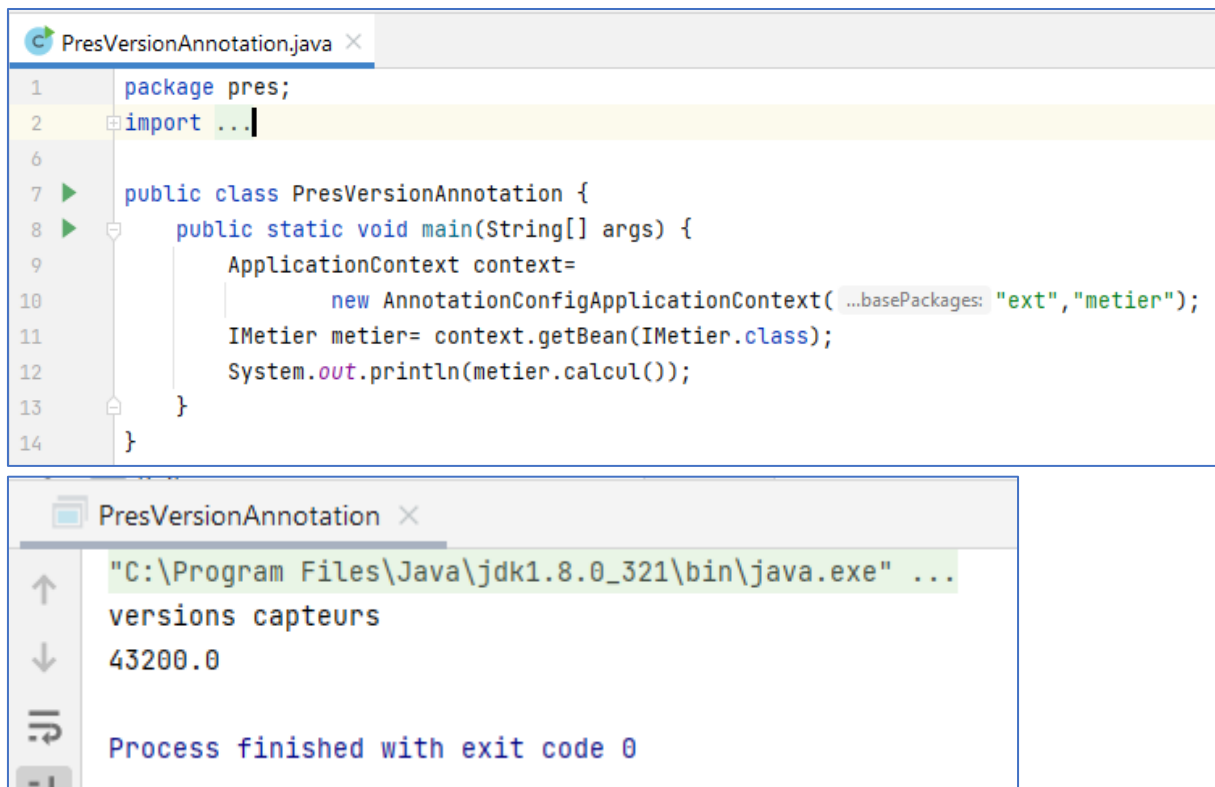
```

Pour utiliser l'autre version il faut seulement ajouter `@Component` a la nouvelle implémentation que tu veux ajouter au projet et aussi mettre le nom de son package dans `AnnotationConfigApplicationContext` pour permettre a spring de les scanner

```

DaoImplVC.java x
1 package ext;
2
3 import ...
5
6 @Component
7 public class DaoImplVC implements IDao {
8     @Override
9     public double getData() {
10         System.out.println("versions capteurs");
11         double temp = 80;
12
13         return temp;
14     }
15 }

```



The image shows two screenshots from an IDE. The top screenshot displays the source code of a Java class named `PresVersionAnnotation` in the `pres` package. The code includes a `main` method that creates an `ApplicationContext`, retrieves an `IMetier` bean, and prints its `calcul()` result. The bottom screenshot shows the command prompt output of running the program, displaying the command path, the output `versions capteurs` and `43200.0`, and a confirmation that the process finished with exit code 0.

```
PresVersionAnnotation.java x
1 package pres;
2 import ...
6
7 public class PresVersionAnnotation {
8     public static void main(String[] args) {
9         ApplicationContext context=
10             new AnnotationConfigApplicationContext( ...basePackages: "ext","metier");
11         IMetier metier= context.getBean(IMetier.class);
12         System.out.println(metier.calcul());
13     }
14 }
```

```
PresVersionAnnotation x
"C:\Program Files\Java\jdk1.8.0_321\bin\java.exe" ...
versions capteurs
43200.0
Process finished with exit code 0
```

Conclusion

Spring est un outil puissant et performant qui permet au développeur de manipuler son application avec beaucoup de facilité, il le permet de l'étendre on y ajoutant plus des fonctionnalités et ou en modifiant celles qui déjà existe. Cela d'un part, d'un autre part Spring permet au développeur de se développer le côté fonctionnelle de l'application et laisser la gestion du côté technique qui est répétitif au Framework.

