



**ECOLE MAROCAINE DES  
SCIENCES DE L'INGENIEUR**  
*Membre de*   
**HONORIS UNITED UNIVERSITIES**

# Compte Rendu J2EE: Couplage faible

**RÉALISÉ PAR:  
AZRHILIL JIHANE**

---

**2023**

# Objectif du TP

---

L'objectif de ce TP est de savoir comment rendre un projet **fermé à la modification et ouvert à l'extension** en se basant sur le principe de couplage faible qui met l'accent sur les interfaces.

Donc à la fin de ce TP, on sera capable à comprendre comment **injecter des dépendances** avec les deux méthodes (**sans framework spring et avec le spring**).

Notre projet va être composé en 3 couches:

- **Couche accès aux données (DAO)**: permet de définir les classes et interagir avec la BD.
- **Couche Métier**: c'est la partie du traitement métier de l'application. Il permet de récupérer des données depuis la couche DAO afin d'implémenter un ensemble des méthodes et fonctions puis envoyer le résultat à la couche présentation.
- **Couche Présentation**: c'est la partie d'affichage.

# Injection des dépendances : instanciation statique

Pour injecter les dépendances d'une manière statique et sans utiliser le framework Spring. Premièrement, on doit créer un projet java sur IntelliJ IDEA. Puis, on va créer 3 packages (dao, metier et presentation) qui représentent les 3 couches de notre application. L'**instanciation statique** reposera sur les setters et constructeurs...

## Package dao

```
1 package dao;
2
3 public interface IDao {
4     double getData();
5 }
6
```



On a créé un interface IDao qui contient les méthodes public que notre classe doit implémenter

```
1 package dao;
2
3 public class DaoImpl implements IDao {
4     @Override
5     public double getData() {
6         System.out.println("From SQL DB");
7         return 7;
8     }
9 }

```



La classe DaoImpl implémente la méthode getData de l'interface IDao. Cette méthode doit permettre de se connecter à la BD pour récupérer les données.

## Package metier

```
1 package metier;
2
3 public interface IMetier {
4     double calcul();
5 }
6
```



L'interface IMetier doit contenir que les méthodes qui permettent d'atteindre les besoins fonctionnels.

# Injection des dépendances : instanciation statique

---

```
1 package metier;
2 import dao.IDao;
3
4 public class MetierImpl implements IMetier{
5     private IDao dao;
6     @Override
7     public double calcul() {
8         double data=dao.getData();
9         return data*10;
10    }
11    public void setDao(IDao dao) { this.dao=dao; }
12 }
```

- La déclaration d'un objet de type IDao implique le principe du couplage faible. Cet objet n'est pas initialisé alors il est null par défaut.
- La méthode calcul permet de récupérer les données depuis la couche DAO puis faire le calcul.
- DAO c'est une variable privé donc pour injecter à cette variable un objet d'une classe qui implémente l'interface IDao on va utiliser les setters. Cette méthode d'injection est appelée injection statique.

```
1 package dao;
2
3 public class DaoSQL implements IDao{
4     @Override
5     public double getData() {
6         System.out.println("From No SQL DB");
7         return 10;
8     }
9 }
```



On ajoute une extension DaoSQL dans la couche dao pour éviter de modifier au code.

# Injection des dépendances : instanciation statique

---

## Package presentation

```
1 package presentation;
2 import dao.DaoNSQL;
3 import metier.MetierImpl;
4
5 public class Presentation {
6     public static void main(String[] args){
7
8         MetierImpl metier= new MetierImpl();
9         DaoNSQL nosql= new DaoNSQL();
10        metier.setDao(nosql);
11        double resultat=metier.calcul();
12        System.out.println("Résultat est: "+resultat);
13    }
14 }
15
```

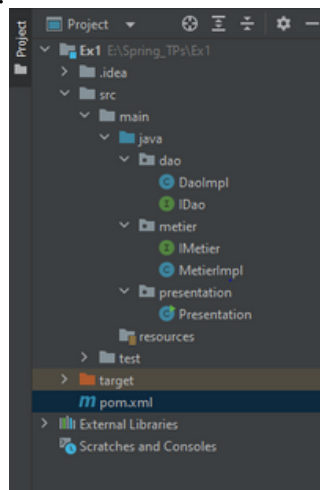
- Pour faire l'injection des dépendances on va travailler avec les setter qui permet de lier deux objets par une association.
- On fait appel à la couche metier pour faire le traitement afin d'afficher le résultat final dans la console.

## Exécution

```
Run: Presentation
C:\Users\WPV\jdk\corretto-1.8.0_362\bin\java.exe ...
From No SQL DB
Résultat est: 10.0
Process finished with exit code 0
```

# Injection des dépendances : Les annotations

Pour injecter les dépendances avec le framework Spring en utilisant les annotations. Premièrement, on doit créer un **projet maven** (logiciel d'automatisation de gestion de projet), c'est un projet java qui contient un fichier **pom.xml**, ce dernier regroupe l'ensemble des dépendances.



Après la création du projet maven on ajoute les dépendances suivantes dans le fichier pom.xml :

- 1.Spring Core
- 2.Spring context
- 3.Spring Beans

On trouve les dépendances au dessus dans le lien suivant:

<https://mvnrepository.com/artifact/org.springframework>

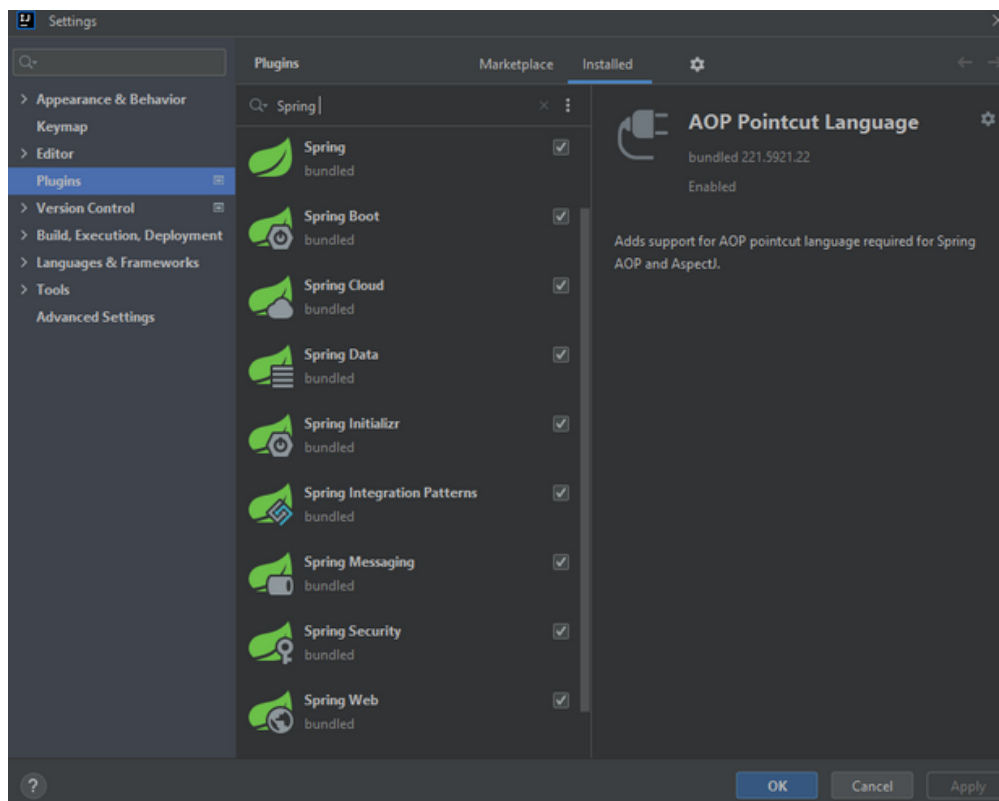
```
16 <!-- https://mvnrepository.com/artifact/org.springframework/spring-core -->
17 <dependency>
18   <groupId>org.springframework</groupId>
19   <artifactId>spring-core</artifactId>
20   <version>5.2.0.RELEASE</version>
21 </dependency>
22 <!-- https://mvnrepository.com/artifact/org.springframework/spring-context -->
23 <dependency>
24   <groupId>org.springframework</groupId>
25   <artifactId>spring-context</artifactId>
26   <version>5.2.0.RELEASE</version>
27 </dependency>
28 <!-- https://mvnrepository.com/artifact/org.springframework/spring-beans -->
29 <dependency>
30   <groupId>org.springframework</groupId>
31   <artifactId>spring-beans</artifactId>
32   <version>5.2.0.RELEASE</version>
33 </dependency>
34 </dependencies>
35
36 </project>
```

# Injection des dépendances : Les annotations

---

Après l'ajout des dépendances on doit faire un **reload** du projet maven pour puisse recharger les dépendances et qu'on puisse travailler avec le framework Spring.

L'ajout des Spring Tools

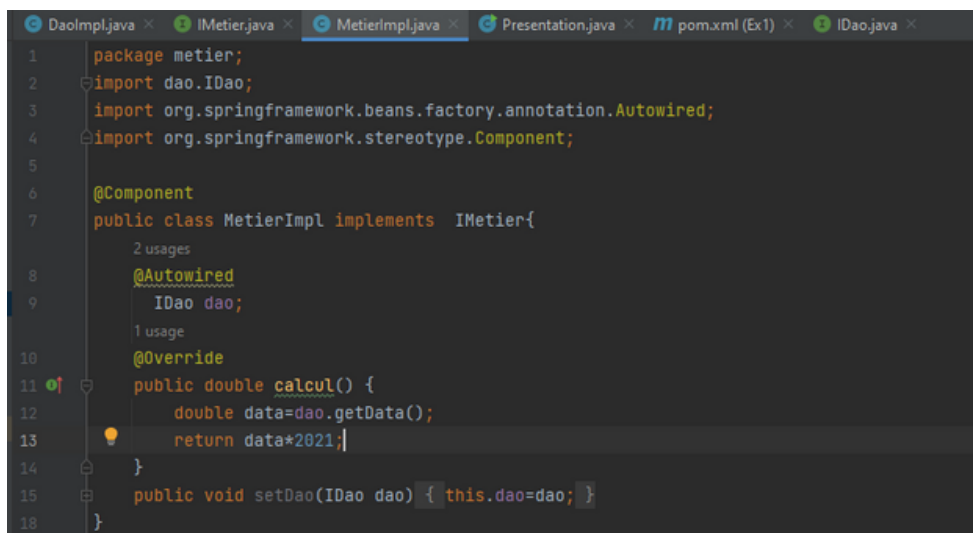


Pour utiliser la version annotations qui sont **appliquées sur les classes**, on va travailler avec l'annotation **@Component** qui permet de dire que la classe c'est un composant de Spring. Cette annotation va prendre par défaut le nom de la classe comme identifiant de Bean.

# Injection des dépendances : Les annotations

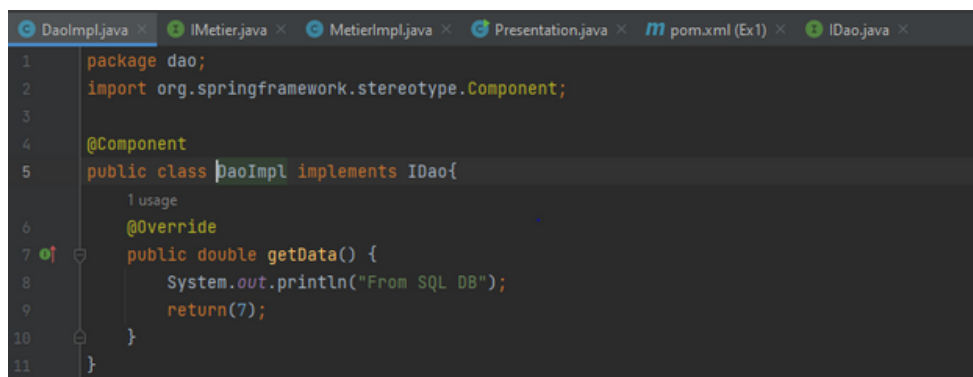
Pour utiliser les annotations qui **sont appliquées sur les champs**, on va travailler avec **@Autowired** qui permet d'activer l'injection des dépendances sur un objet d'une manière automatique.

L'injection à travers cette méthode peut être faite soit par un constructeur ou un setter.



```
1 package metier;
2 import dao.IDao;
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.stereotype.Component;
5
6 @Component
7 public class MetierImpl implements IMetier{
8     2 usages
9     @Autowired
10     IDao dao;
11     1 usage
12     @Override
13     public double calcul() {
14         double data=dao.getData();
15         return data*2021;
16     }
17     public void setDao(IDao dao) { this.dao=dao; }
18 }
```

Au moment le Spring va instancier un objet de la classe MetierImpl, il va chercher un objet de type IDao puis l'injecter à la variable dao.



```
1 package dao;
2 import org.springframework.stereotype.Component;
3
4 @Component
5 public class IDao{
6     1 usage
7     @Override
8     public double getData() {
9         System.out.println("From SQL DB");
10         return(7);
11     }
12 }
```



# Injection des dépendances : Les annotations

---

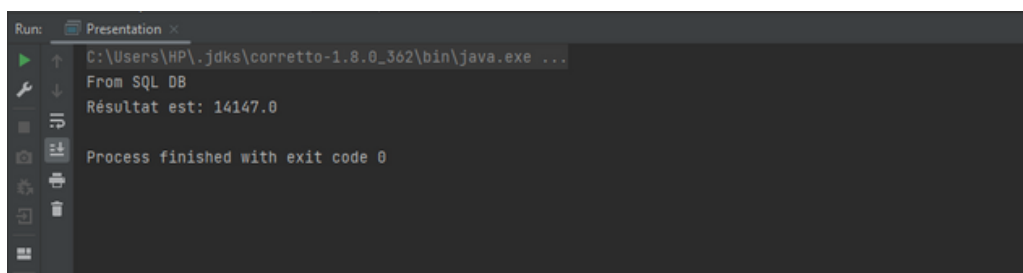
Pour la couche presentation, on doit travailler avec un objet **ApplicationContext** qui gère l'ensemble des fichiers xml de base de configuration.

**AnnotationConfigApplicationContext** permet de scanner toutes les classes des packages dao et metier.

**getBean** permet de charger un Bean qui implémente l'interface IMetier.

```
1 package presentation;
2 import metier.IMetier;
3 import org.springframework.context.ApplicationContext;
4 import org.springframework.context.annotation.AnnotationConfigApplicationContext;
5
6 public class Presentation {
7     public static void main(String[] args){
8
9         ApplicationContext context= new AnnotationConfigApplicationContext( "dao","metier");
10        IMetier metier= context.getBean(IMetier.class);
11        System.out.println("Résultat est: "+ metier.calcul() );
12    }
13 }
```

## Exécution



```
Run: Presentation
C:\Users\HP\.jdk\corretto-1.8.0_362\bin\java.exe ...
From SQL DB
Résultat est: 14147.0
Process finished with exit code 0
```

# Conclusion

---

- La couche dao est fermée à la modification par ce que les classes de cette couche ne dépend pas des autres classes donc il est fermée à la modification.
- La couche dao est ouverte à l'extension par ce qu'on peut toujours créer une nouvelle implémentation de l'interface IDao.
- La couche metier est fermée à la modification par ce qu'elle dépend que des interfaces. Elle est ouverte à l'extension par ce qu'on peut ajouter une autre classe qui implémente l'interface IMetier.
- La couche presentation n'est pas fermée à la modification par ce qu'on utilise le couplage fort.
- Lorsqu'une classe dépend d'une autre classe directement c'est un **couplage fort** et lorsqu'une classe dépend que des interfaces c'est un **couplage faible**.