



ECOLE MAROCAINE DES
SCIENCES DE L'INGENIEUR
Membre de

HONORIS UNITED UNIVERSITIES

Compte Rendu : J2EE

Réalisée par :
MOUAFFAK Nisrine

Objectifs du TP

- ✓ Chaque application doit être fermée à la modification et ouverte à l'extension lors de sa conception et aussi doit respecter les exigences (fonctionnelles, techniques et financières).
- ✓ Le rôle de l'inversion de contrôle IoC.
- ✓ Savoir la différence entre le couplage fort et le couplage faible.
- ✓ L'injection des dépendances.

Le principe d'ouvert-fermé

En programmation orientée objet, le principe ouvert/fermé affirme qu'une classe doit être à la fois ouverte (à l'extension) et fermée (à la modification). « Ouverte » signifie qu'elle a la capacité d'être étendue. « Fermée » signifie qu'elle ne peut être modifiée que par extension, sans modification de son code source.

En pratique, le principe ouvert/fermé oblige à faire bon usage de l'abstraction et l'utilisation des interfaces.

Inversion de contrôle

À la base du Spring Framework, on trouve un unique principe de conception : l'inversion de contrôle.

L'inversion de contrôle est une façon de concevoir l'architecture d'une application en se basant sur le mécanisme objet de l'injection de dépendance.

Mais son rôle principale : Il 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)

Couplage fort et Couplage faible

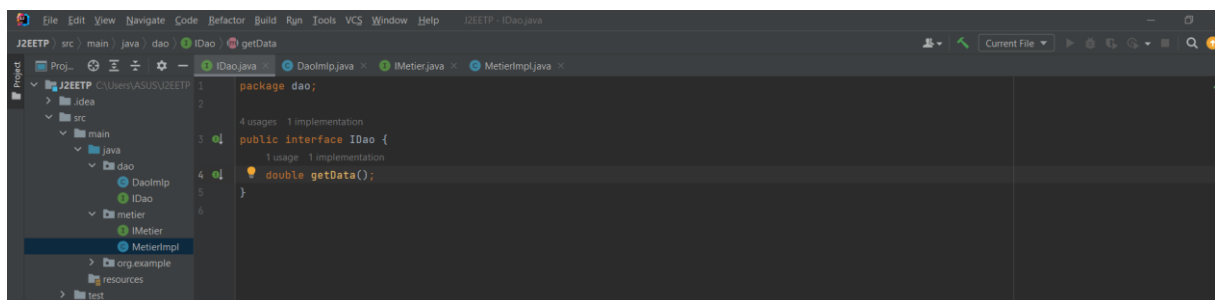
Un couplage fort signifie que les classes et les objets dépendent les uns des autres. En général, le couplage fort n'est pas bon car il réduit la flexibilité et la réutilisation du code, tandis que le couplage faible signifie la réduction des dépendances d'une classe qui utilise directement les différentes classes.

L'injection des dépendances

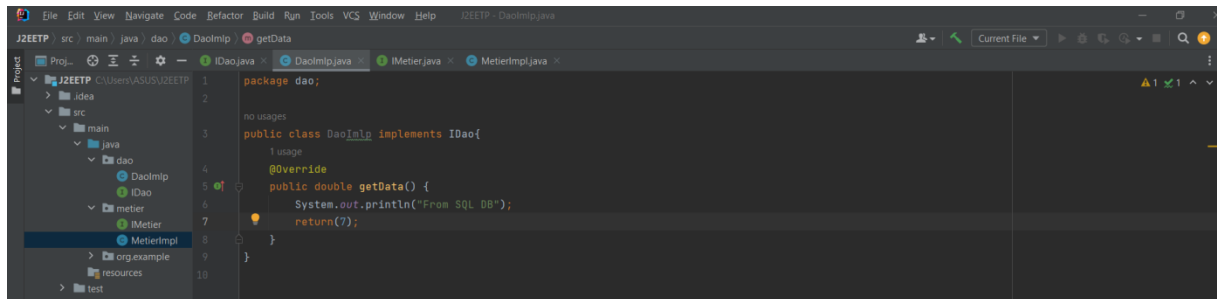
Sans utiliser Spring par instantiation statique :

Alors 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.

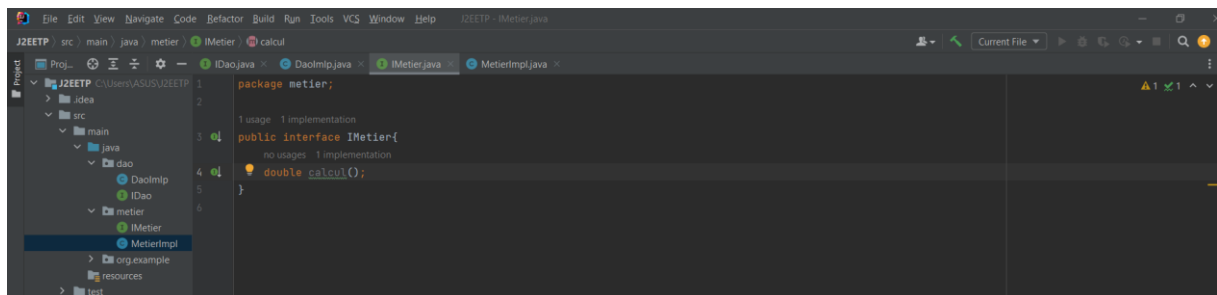
L'interface IDao avec la méthode getData()



La classe DaoImpl qui implémente la méthode getDate de l'interface IDao. A partir de cette méthode on peut se connecter à la BDD pour récupérer les données.

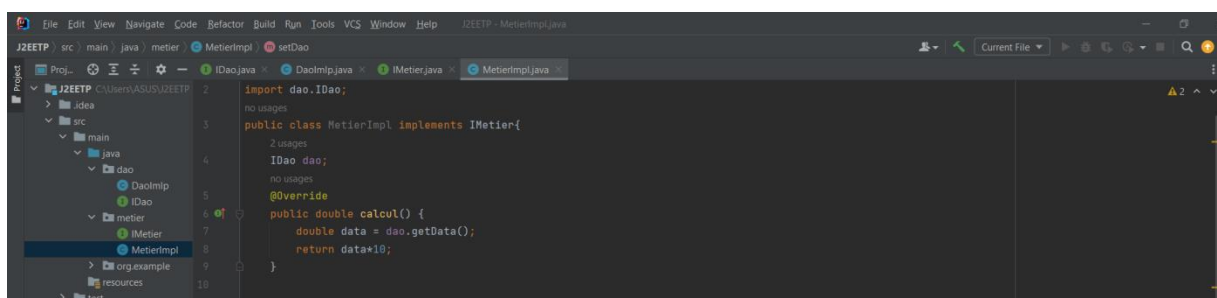


On passe maintenant à la couche métier :



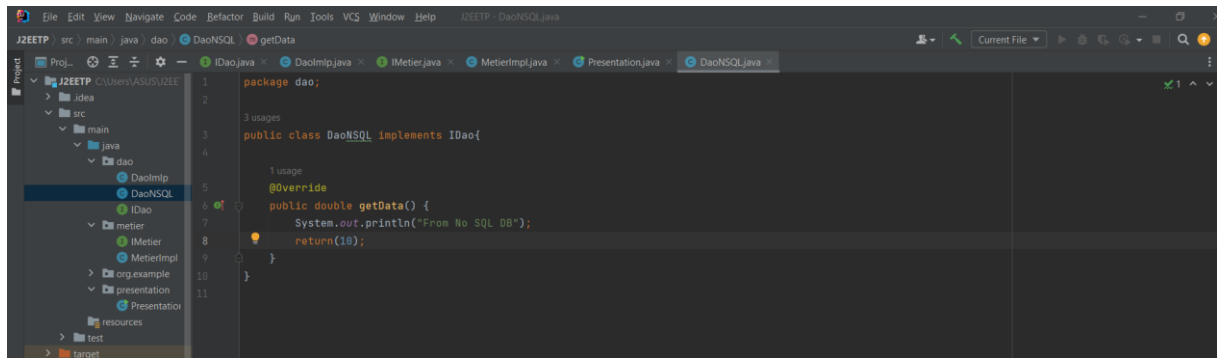
On trouve dans cette interface que les méthodes fonctionnelles.

La méthode calcul représente le fonctionnement réaliser pour ce TP.



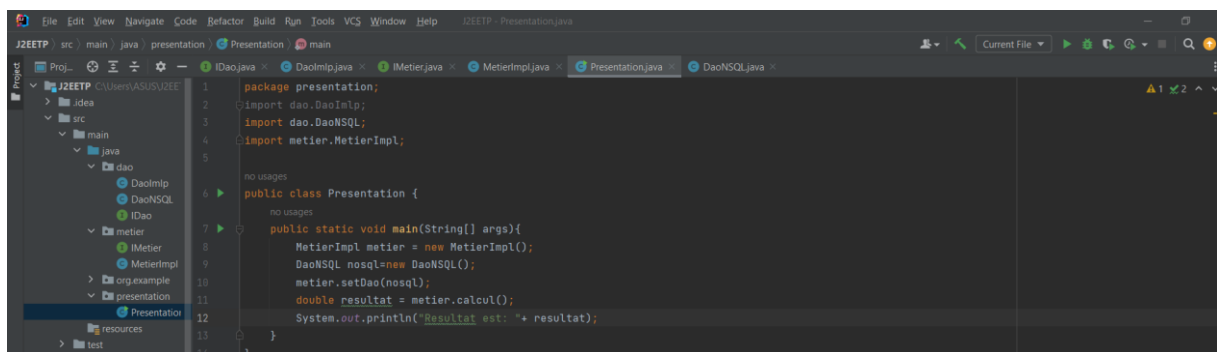
La déclaration d'un objet de type IDao implique le principe du couplage faible.

La méthode calcul permet de récupérer les données depuis la couche DAO puis faire le calcul.



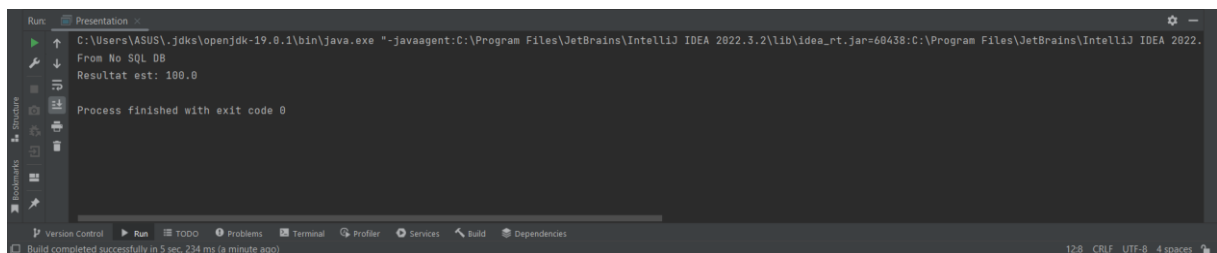
L'ajout de cette extension DaoNSQL nous a permis d'éviter de modifier le code.

La couche présentation :



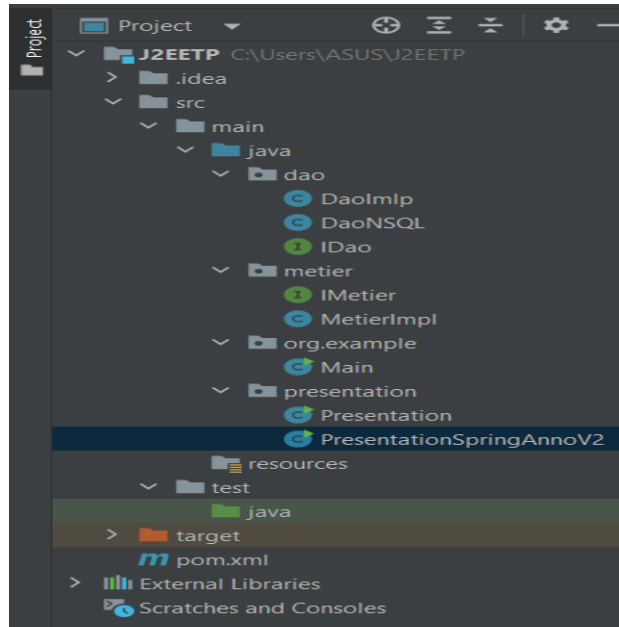
On fait appel à la couche metier pour faire le traitement afin d'afficher le résultat final dans la console.

Exécution :



Avec l'utilisation de Spring En utilisant les annotations :

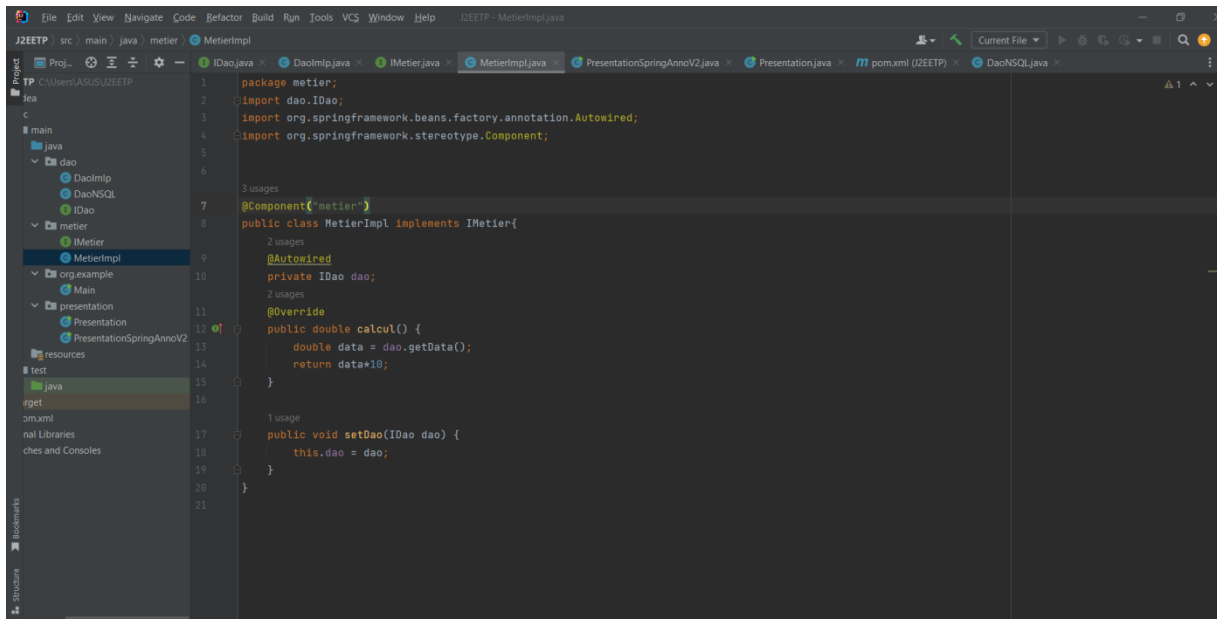
On doit créer un projet maven (logiciel d'automatisation de gestion de projet), c'est un projet java qui contient un fichier pom.xml, ce dernier regroupe l'ensemble des dépendances.



Après on ajoute les trois dépendances : Spring Core, Spring Context et Spring Beans.

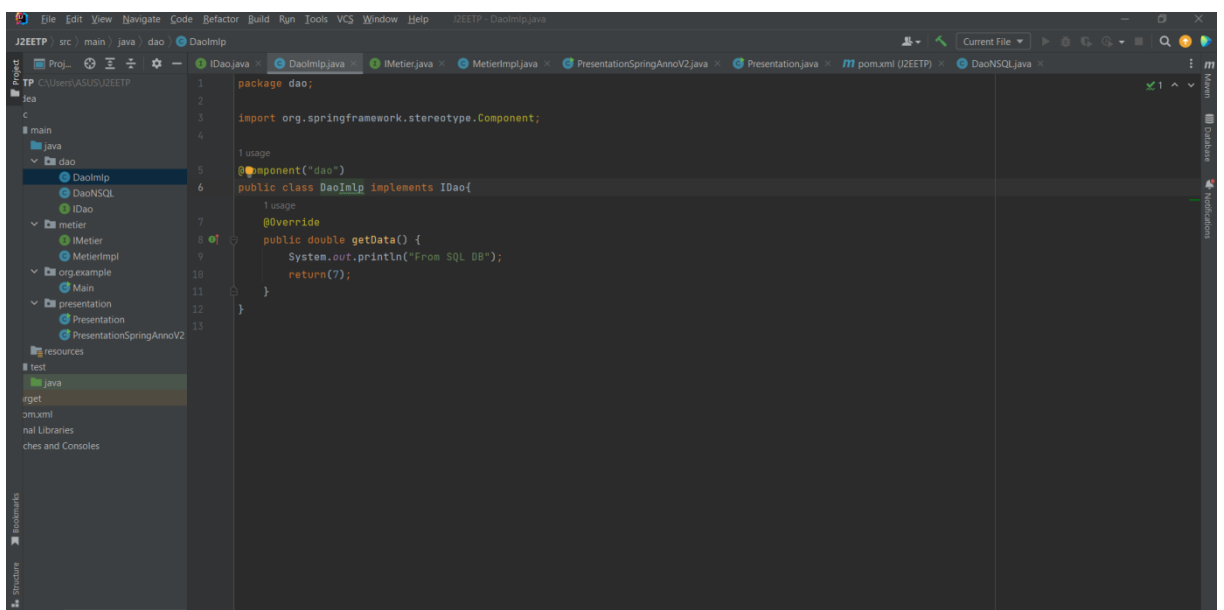
Pour utiliser les annotations qui sont appliquées sur les champs, on va travailler avec `@Autowired` qui permet d'activer l'injection des dépendances sur un objet d'une manière automatique.

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
7 @Component("metier")
8 public class MetierImpl implements IMetier{
9
10     @Autowired
11     private IDao dao;
12
13     @Override
14     public double calcul() {
15         double data = dao.getData();
16         return data*10;
17     }
18
19     public void setDao(IDao dao) {
20         this.dao = dao;
21     }
22 }
```

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.

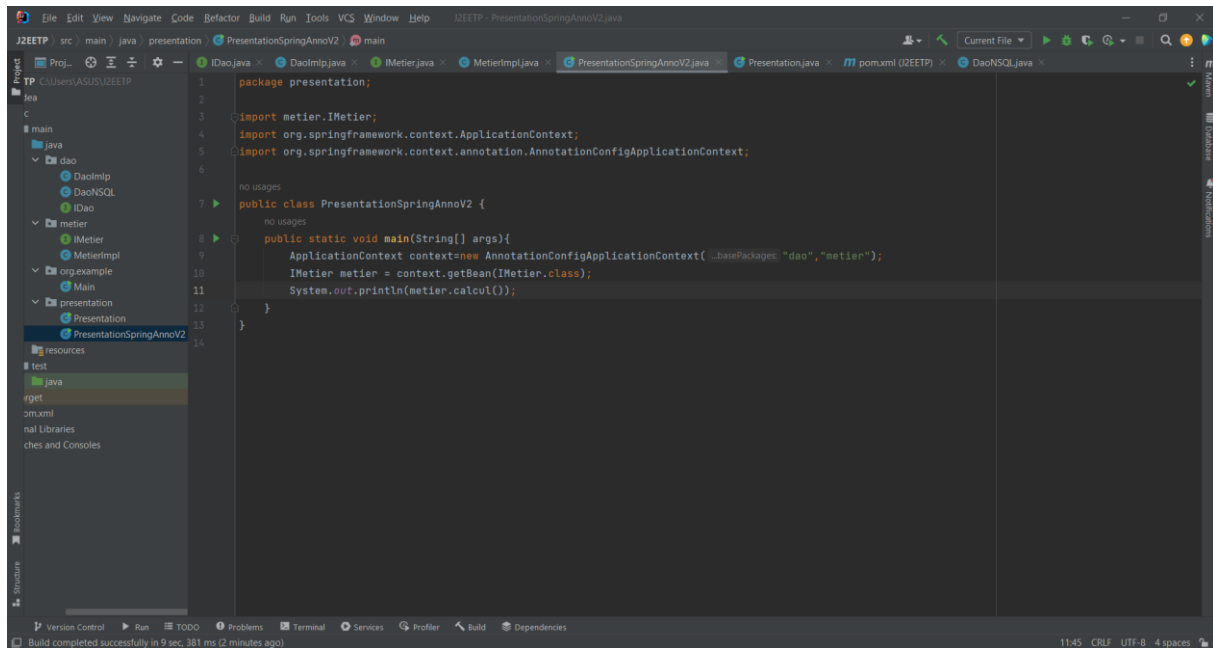


```
1 package dao;
2
3 import org.springframework.stereotype.Component;
4
5
6 @Component("dao")
7 public class DaoImpl implements IDao{
8
9     @Override
10     public double getData() {
11         System.out.println("From SQL DB");
12         return 7;
13     }
14 }
```

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

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

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



Exécution :

