



Compte Rendu du TP JAVA EE

RÉALISÉ PAR :
LAMIAA TAOUNTE

Objectif

Dans ce TP, on va avoir :

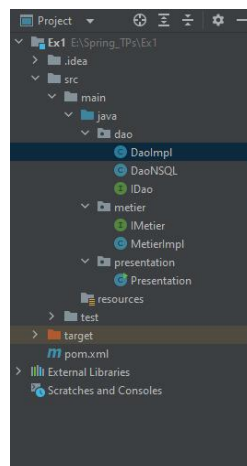
- ➔ Un projet doit être fermé à la modification et ouvert à l'extension.
- ➔ Les 2 méthodes de couplage Fort et Faible
- ➔ Les méthodes d'injection des dépendances en utilisant le Framework Spring et sans Spring.

Introduction

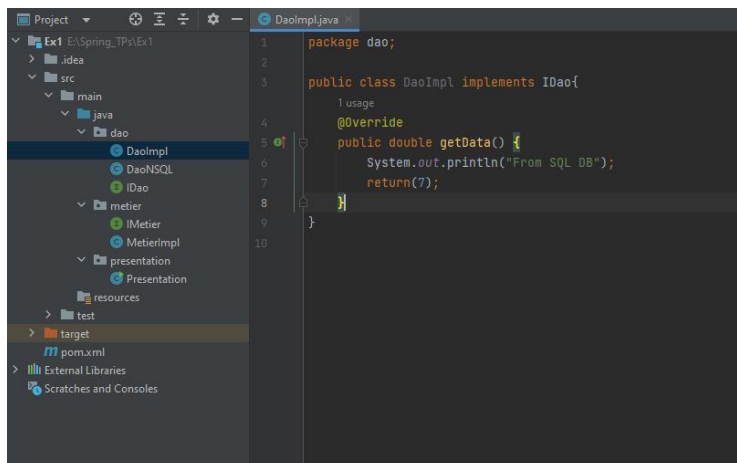
- JEE (Java Enterprise Edition) est une norme qui va spécifier à la fois l'infrastructure de gestion de vos applications et les API des services utilisées pour concevoir ces applications.
- Spring est un Framework open source pour construire et définir l'infrastructure d'une application Java3, dont il facilite le développement et les tests.
- Un projet JEE contient 3 couches :
 - **Couche Présentation** = Cette couche est responsable de la présentation de l'application à l'utilisateur final.
 - **Couche Métier** = Cette couche est responsable de la logique de l'application. Elle comprend les classes Java qui effectuent des opérations sur les données et les transforment en réponse aux requêtes des utilisateurs.
 - **Couche Dao** = Cette couche est responsable de la gestion des données et de l'interaction avec la base de données.

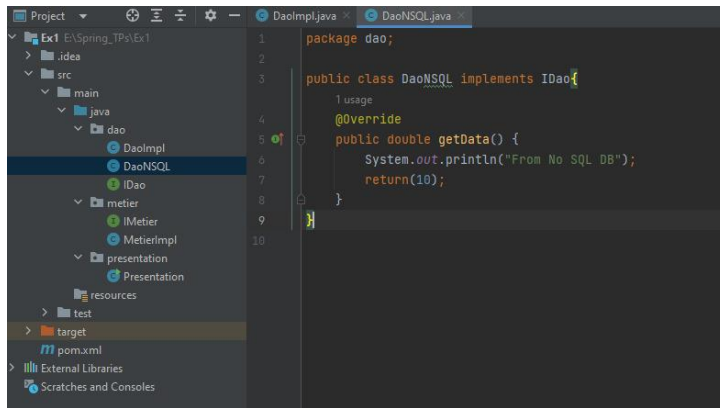
Injection des dépendances (statique) :

- Structure d'un projet JEE



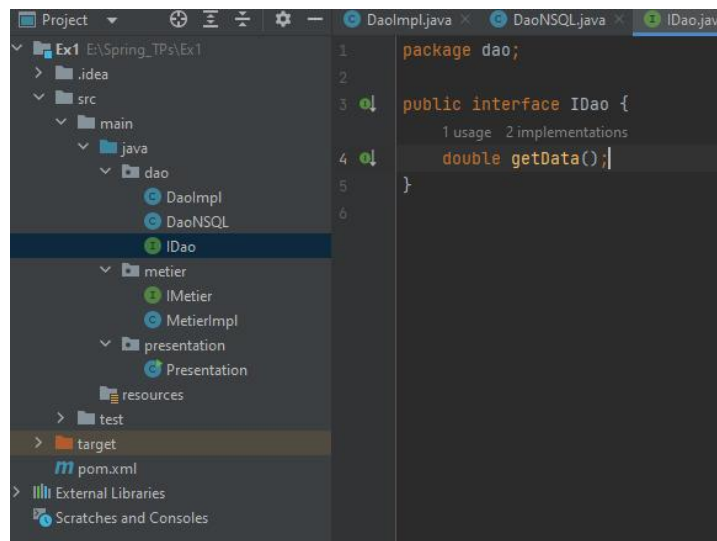
- Le dossier JAVA contient les 3 couches :
 - La couche dao





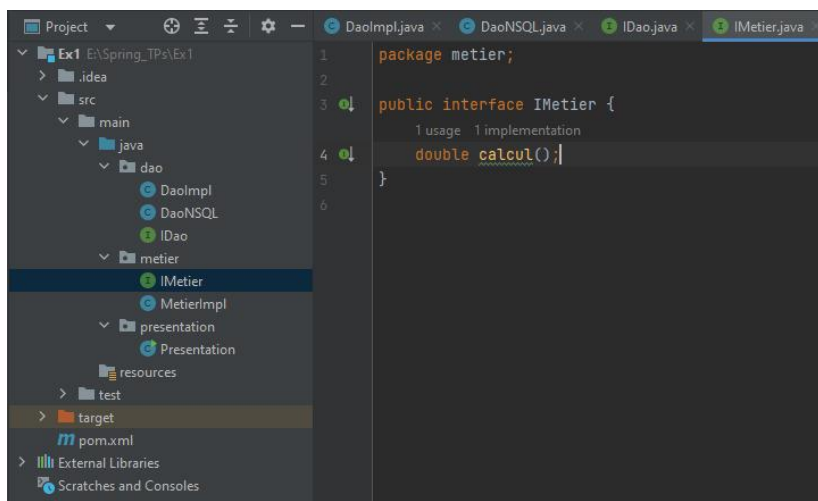
```
1 package dao;
2
3 public class DaoNSQL implements IDao {
4     @Override
5     public double getData() {
6         System.out.println("From No SQL DB");
7         return 10;
8     }
9 }
```

➤ L'ajout d'une extension DaoNSQL



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

- Couche Métier



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

```
1 package metier;
2 import dao.IDao;
3
4 public class MetierImpl implements IMetier{
5     private IDao dao;
6
7     @Override
8     public double calcul() {
9         double data=dao.getData();
10        return data=10;
11    }
12
13    public void setDao(IDao dao) { this.dao=dao; }
14 }
15
```

- Le couplage Faible est impliqué par la déclaration d'un objet de type DAO.
- La méthode Calcul() récupère les données depuis la couche DAO.

- Couche Présentation

```
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
```

- Importation du classe MetierImpl de la couche Métier pour faire le traitement.

➤ Résultat

```
Run: Presentation
C:\Users\HP\.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 avec Spring :

➔ Créer un projet Maven et on modifier le fichier pom.xml

```
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>
```

➔ @Component qui permet de dire que la classe c'est un composant de Spring.

➔ @Autowired qui permet d'activer l'injection des dépendances sur un objet d'une manière automatique.

```
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 }
```

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

➔ on doit utiliser objet ApplicationContext et AnnotationConfigApplicationContext qui 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(...basePackages: "dao","metier");
10        IMetier metier= context.getBean(IMetier.class);
11        System.out.println("Résultat est: "+ metier.calcul());
12    }
13 }
14
```

➔ Résultat

```
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
```