

2021-2022

Département Mathématiques et Informatique
GLSID 2 – S4

Architecture JEE et Middlewares

Rapport
TP 1 IOC manuel

Préparé par : Fatima Zahra HASBI
ENCADRE PAR : MR MOHAMED
YOUSSEFI

Introduction

Ce tp consiste à utiliser à mettre en place l'inversion de contrôle avec une programmation manuelle sans utiliser de Framework et cela dans le but d'avoir une application fermée à la modification et ouverte à l'extension. Et pour pouvoir atteindre ce but on va utiliser un couplage faible qui consiste à utiliser les interfaces d'une sorte d'avoir des classes qui dépendent aux interfaces.

Le p est sous forme d'une simple application qui fait un petit calcul.

Dans ce rapport je montre un ensemble des captures d'écran de réalisation de ce tp.

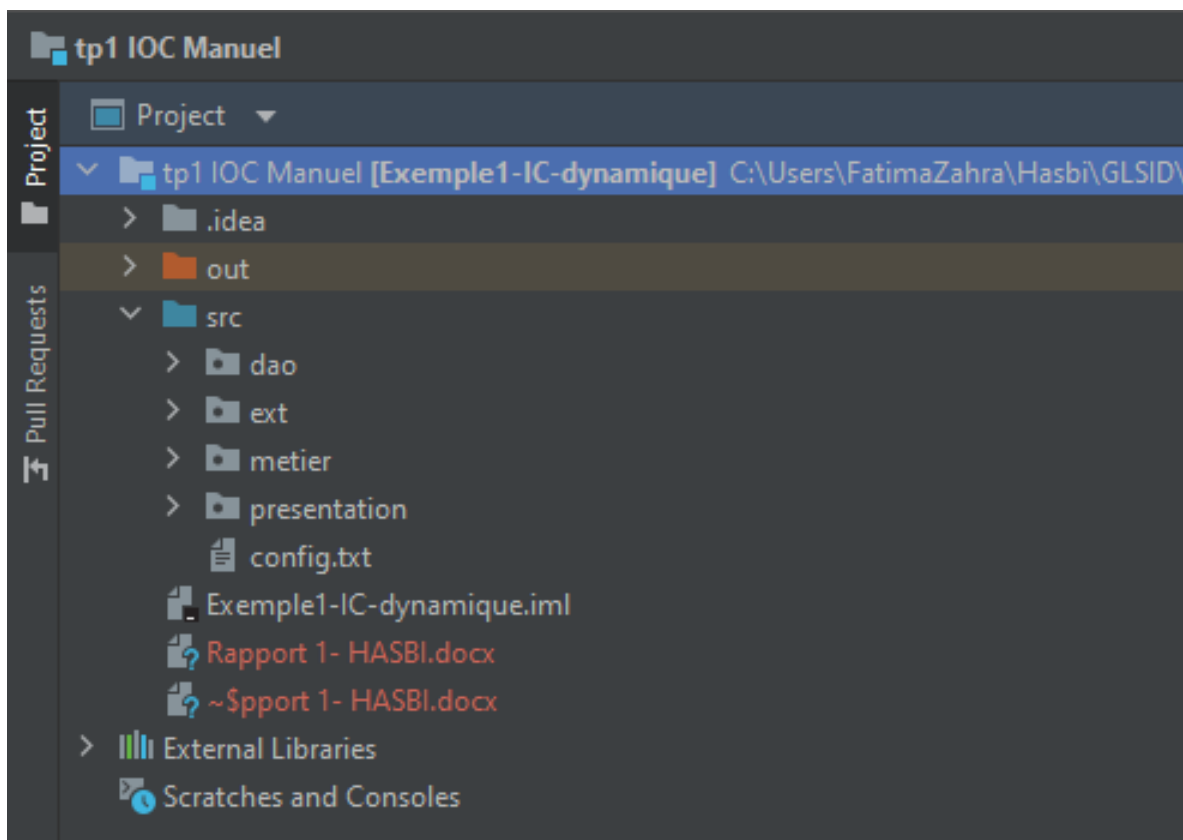
La structure du code

Pour une bonne gestion de l'application on travaille avec trois couches :

- Couche dao
- Couche metier
- Couche presentation

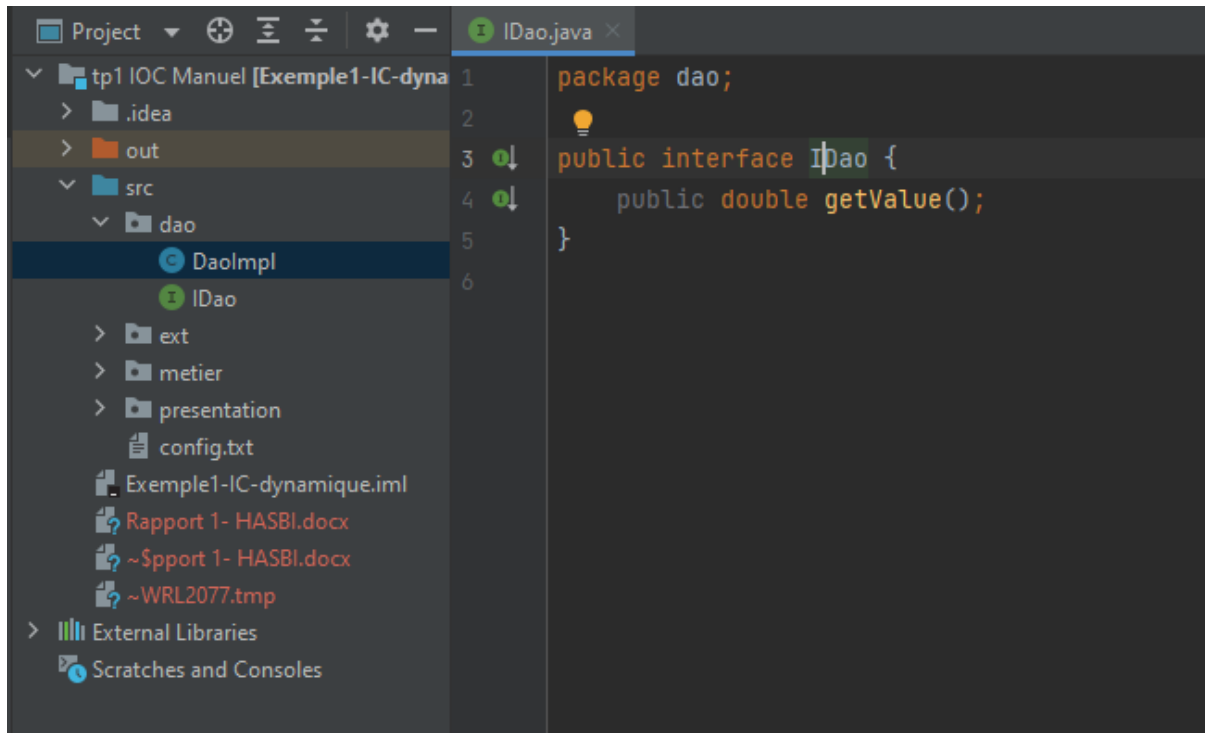
On a ajouté un dossier ext qui contient les extensions qu'on va ajouter à la fin.

Le fichier config.txt est l'endroit où on va modifier le travail de l'application si besoin, afin de ne pas modifier dans le code source.

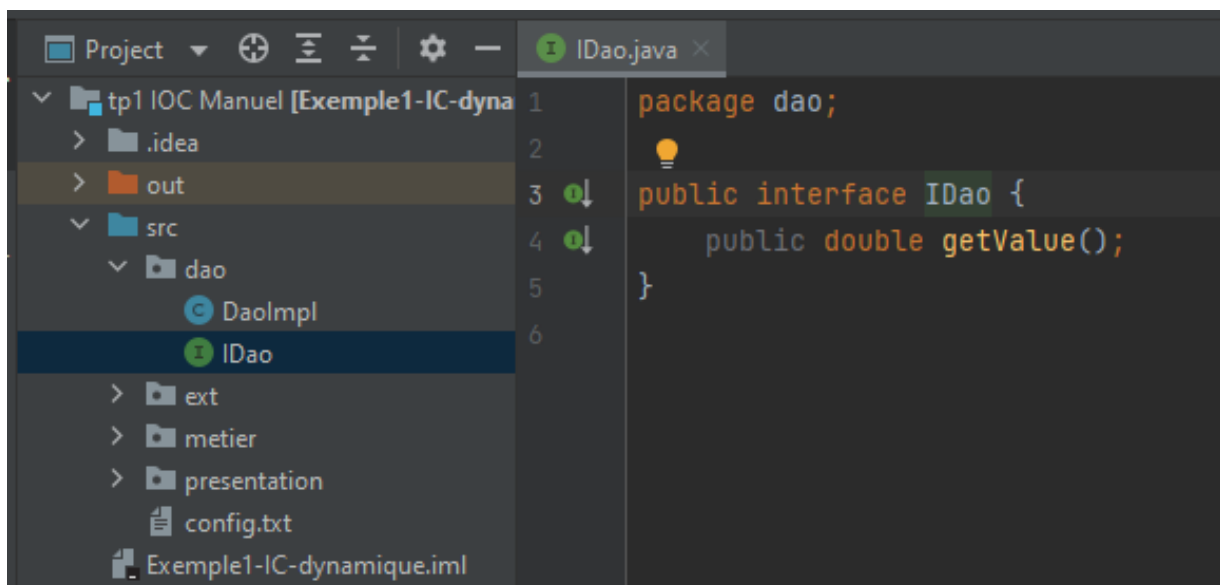


La couche dao

- L'interface IDao

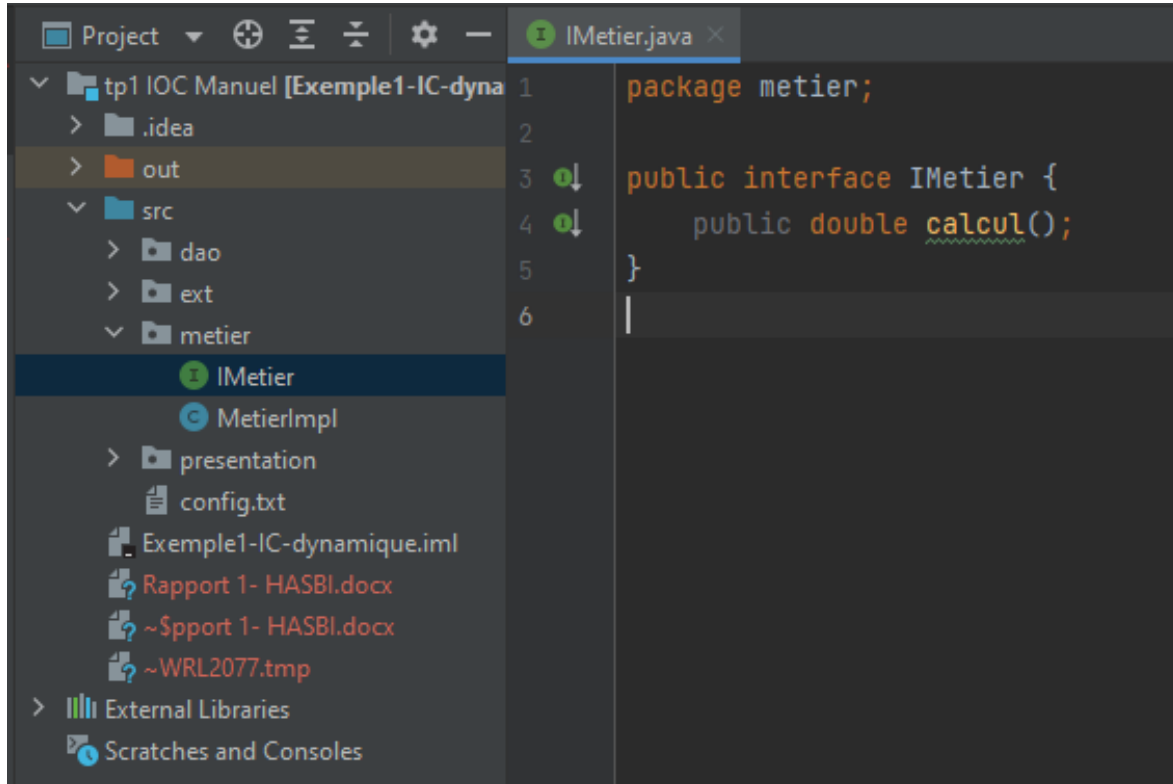


- L'implémentation de l'interface IDao par la classe DaoImpl



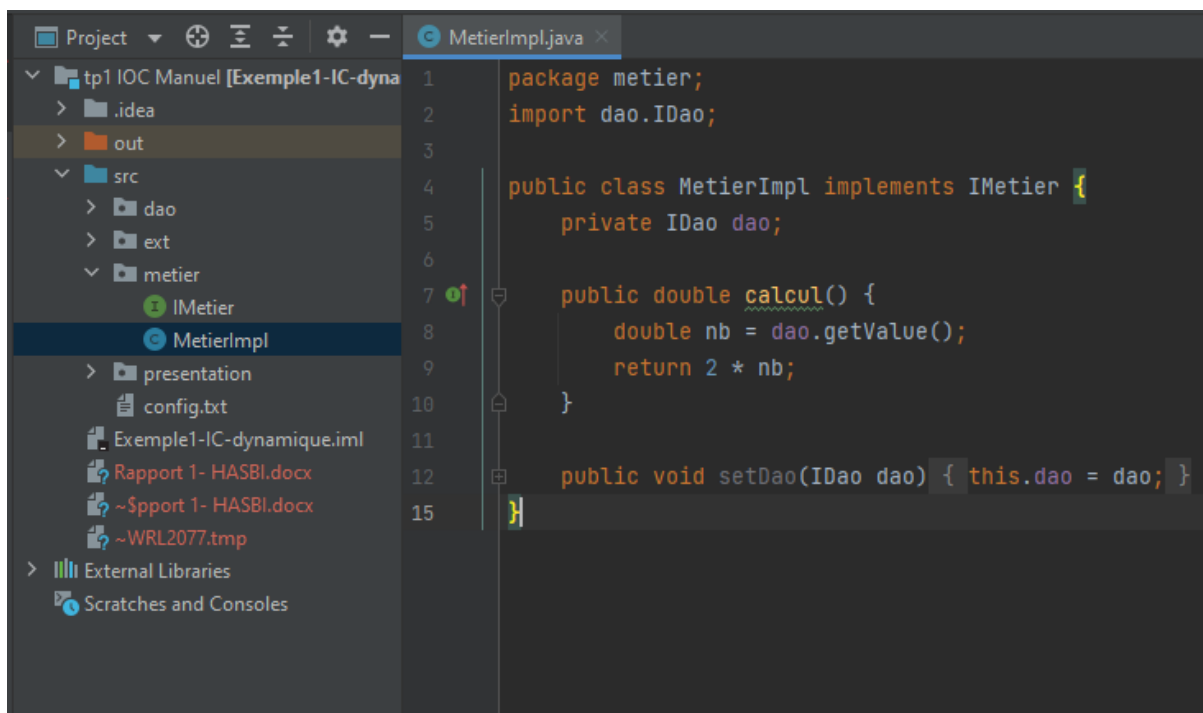
La couche metier

- L'interface IMetier



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

- L'implémentation de l'interface IMetier par la classe MetierImpl



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

La couche presentation

```
Presentation.java X
1 package presentation;
2 import ...
5 public class Presentation {
6     public static void main(String[] args) {
7         //methode classique
8         DaoImpl dao=new DaoImpl();
9         MetierImpl metier=new MetierImpl();
10        metier.setDao(dao);
11        System.out.println(metier.calcul());
12        //methode dynamique en lisant les noms des classes à utiliser depuis config.txt
13        try {
14            Scanner scanner=new Scanner(new File( pathname: "src/config.txt"));
15            String daoClassname=scanner.next();
16            String metierClassName=scanner.next();
17            Class cdao=Class.forName(daoClassname);
18            IDao dao= (IDao) cdao.newInstance();
19            Class cmetier=Class.forName(metierClassName);
20            IMetier metier=(IMetier) cmetier.newInstance();
21            Method meth=cmetier.getMethod( name: "setDao", IDao.class);
22            meth.invoke(metier,dao);
23            System.out.println(metier.calcul());
24        } catch (Exception e) {
```

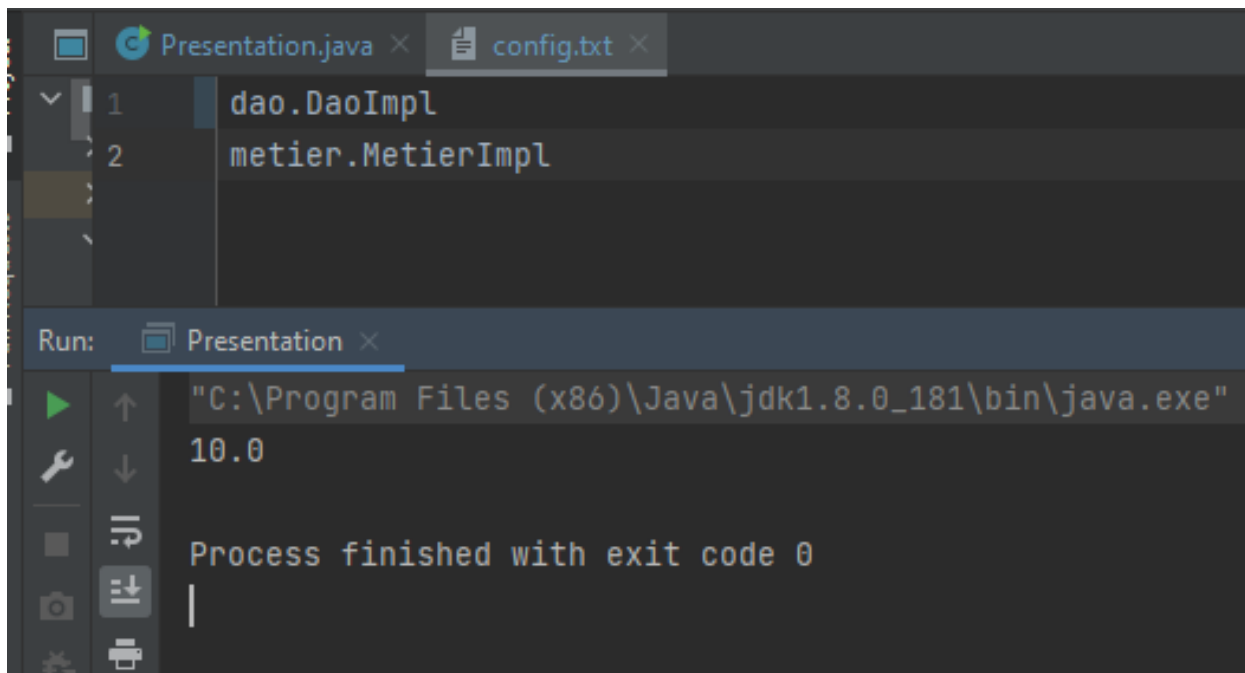
Le fichier config.txt

Il contient les noms des classes qui implémentent les interfaces.

En premier lieu on a les classes dao.DaoImpl et metier.MetierImpl

```
config.txt X
1 dao.DaoImpl
2 metier.MetierImpl
```

Résultat de l'exécution avec cette configuration du fichier config.txt



```
1 dao.DaoImpl
2 metier.MetierImpl
```

Run: Presentation ×

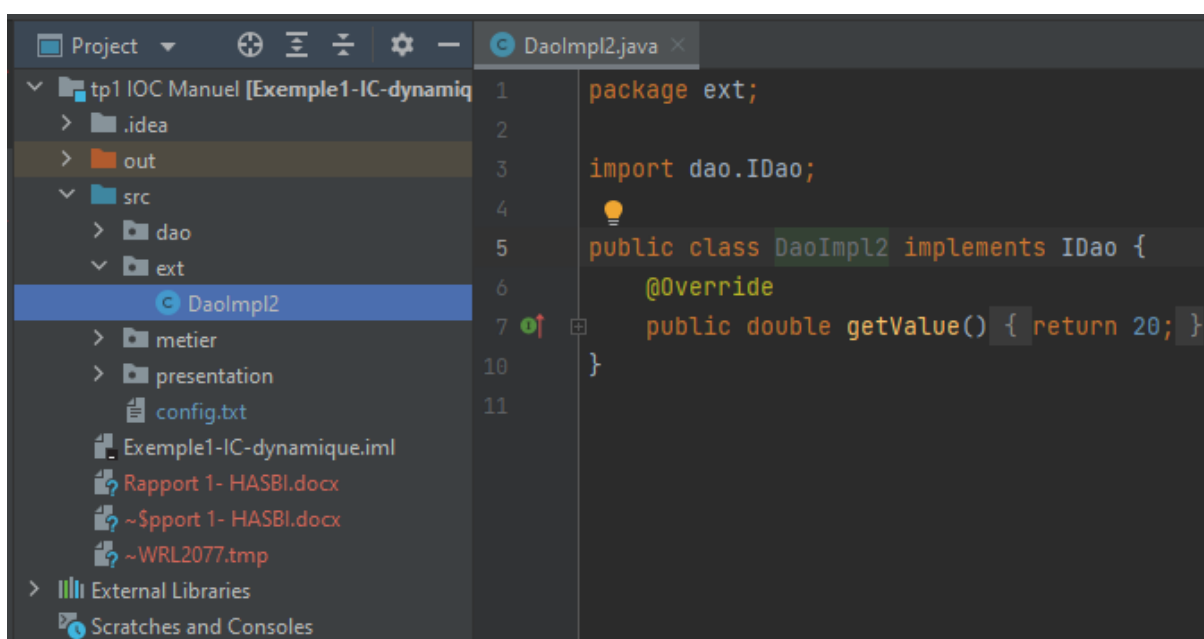
"C:\Program Files (x86)\Java\jdk1.8.0_181\bin\java.exe"

10.0

Process finished with exit code 0

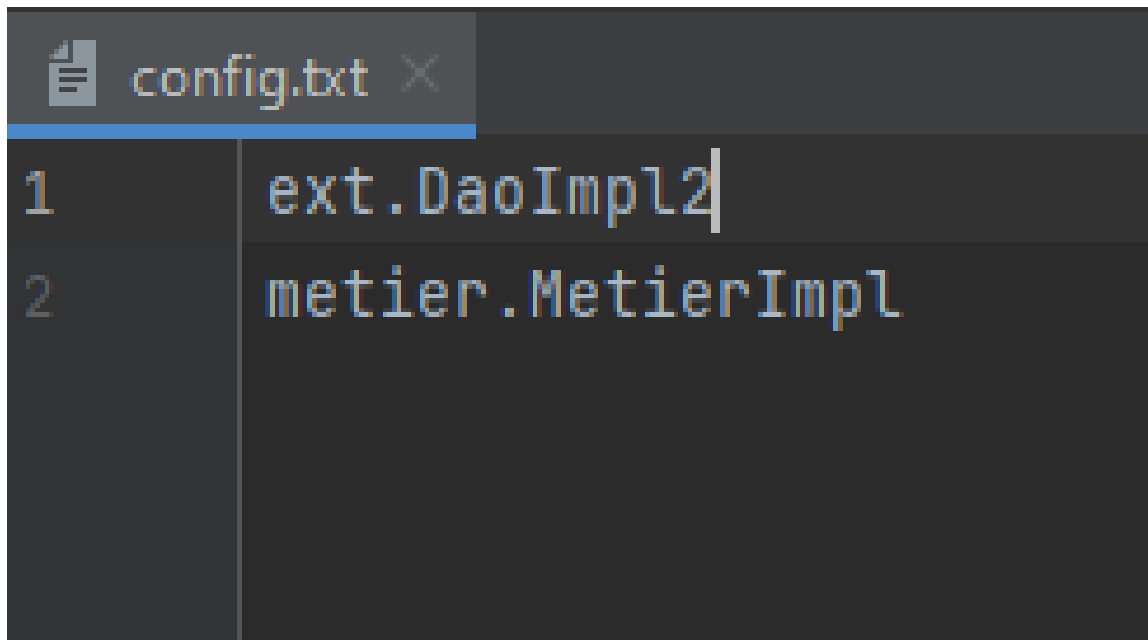
On veut modifier le travail du logiciel. Au lieu de modifier dans le code on va ajouter une autre implémentation de l'interface IDao dans le dossier ext.

La couche ext



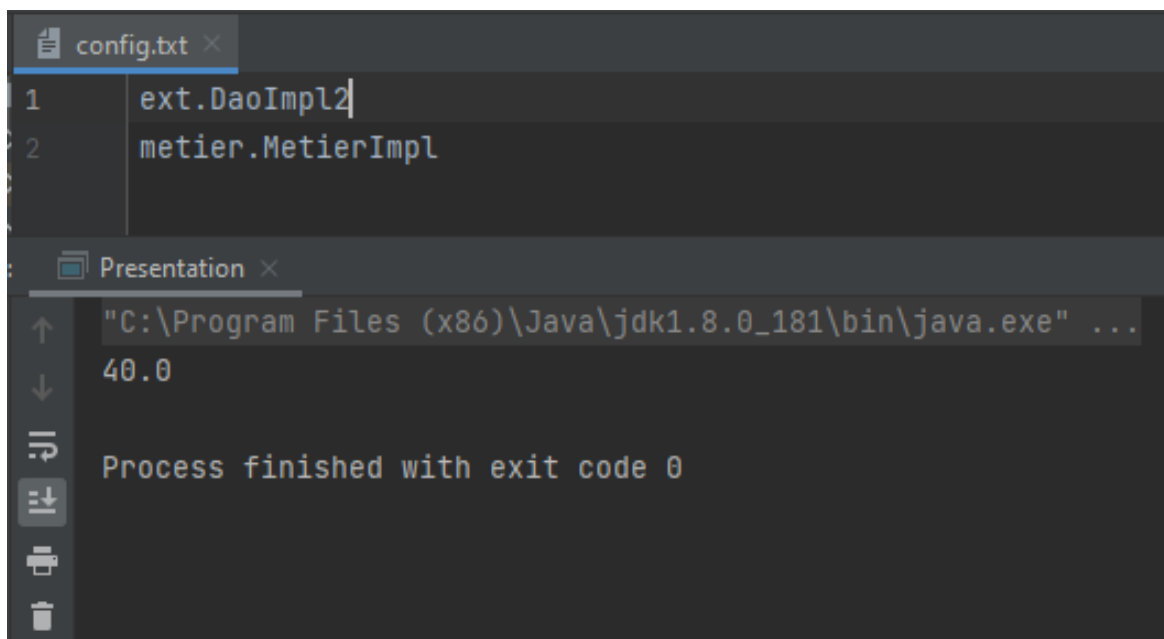
```
1 package ext;
2
3 import dao.IDao;
4
5 public class DaoImpl2 implements IDao {
6     @Override
7     public double getValue() { return 20; }
8 }
9
10
11
```

Modifiant maintenant le nom de la classe à utiliser dans le fichier config.txt



```
config.txt X
1  ext.DaoImpl2
2  metier.MetierImpl
```

Résultat



```
config.txt X
1  ext.DaoImpl2
2  metier.MetierImpl

Presentation X
"C:\Program Files (x86)\Java\jdk1.8.0_181\bin\java.exe" ...
40.0

Process finished with exit code 0
```

FIN