

**DEPARTEMENT MATHÉMATIQUES ET INFORMATIQUE**

# **Activité Pratique N° 1**

**Filière :**

**« Génie du Logiciel et des Systèmes Informatiques Distribués »**

**GLSID**

## **Inversion de contrôle et Injection des dépendances**

Réalisé par :

Najat ES-SAYYAD

**Année Universitaire : 2022-2023**

# Introduction

L'inversion de contrôle est un motif de conception logicielle (ou *design pattern* en anglais) commun à tous les frameworks, devenu populaire avec l'adoption des conteneurs dits "légers".

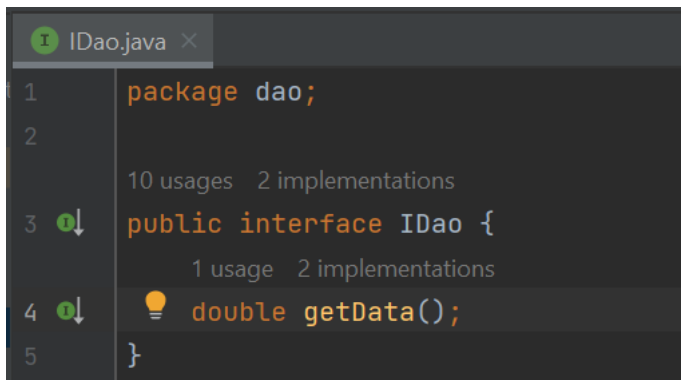
Cette notion fait partie des principes de la programmation orientée aspect.

Selon le mécanisme du *design pattern*, ce n'est plus l'application qui appelle les fonctions d'une librairie, mais un framework qui à l'aide d'une couche abstraite mettant en œuvre un comportement propre, va appeler l'application en l'implémentant. L'inversion de contrôle s'utilise par héritage de classes du framework ou par le biais d'un mécanisme de plug-in.

# Partie I.

---

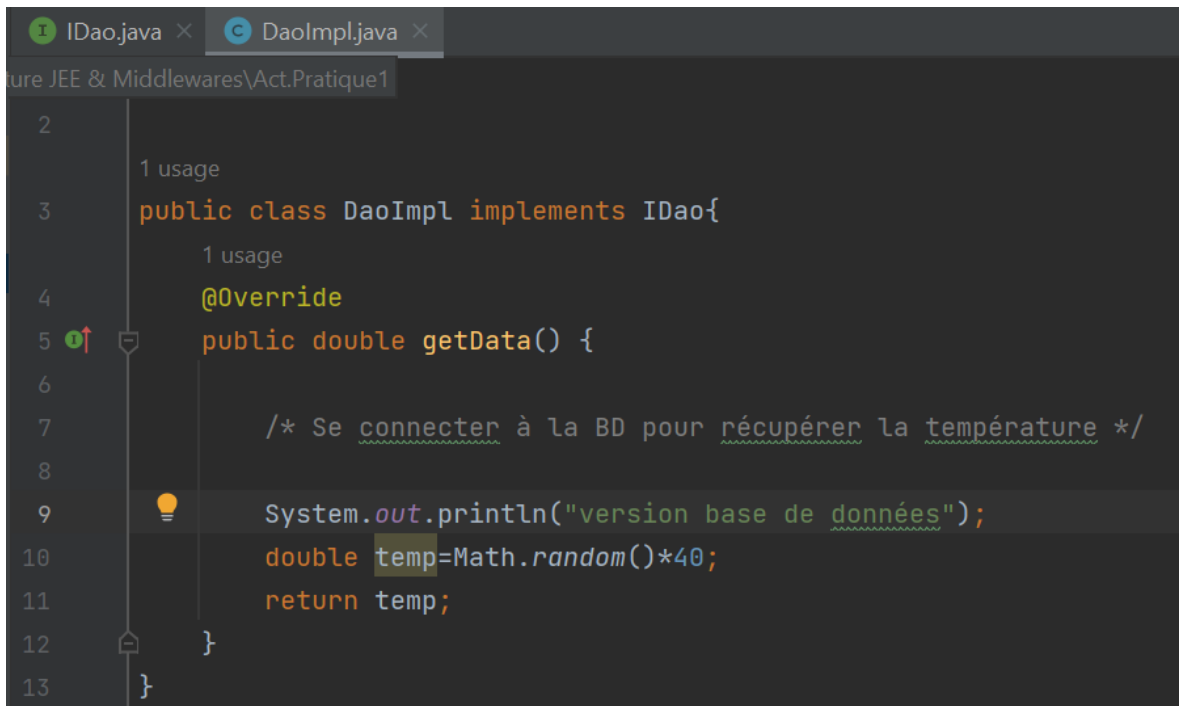
1. Créer l'interface IDao avec une méthode getData



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

2. Créer une implémentation de cette interface

Dans cette implémentation on se connecte à la base de données pour récupérer la température



```
2
3 public class DaoImpl implements IDao{
4     @Override
5     public double getData() {
6
7         /* Se connecter à la BD pour récupérer la température */
8
9         System.out.println("version base de données");
10        double temp=Math.random()*40;
11        return temp;
12    }
13 }
```

3. Créer l'interface IMetier avec une méthode calcul

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

4. Créer une implémentation de cette interface en utilisant le couplage faible

La classe MetierImpl ne dépend que de l'interface IDao

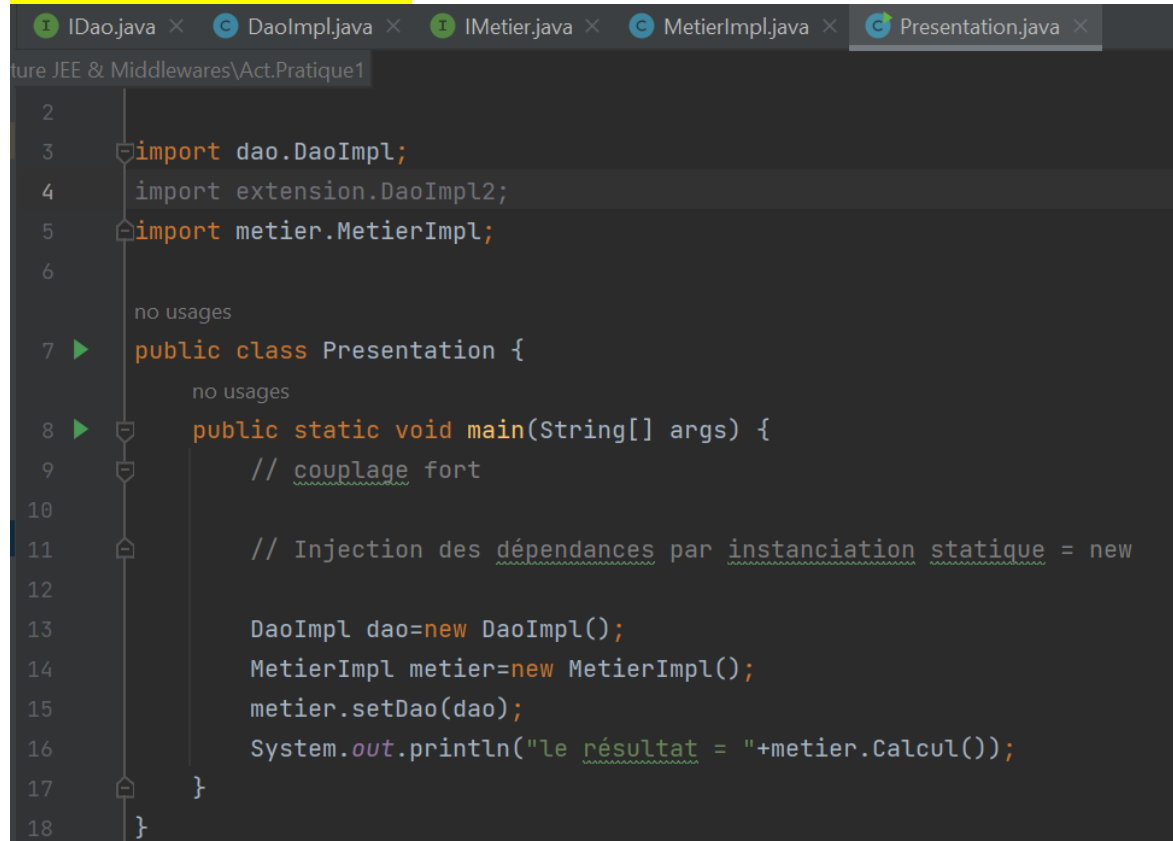
```
1 package metier;
2
3 import dao.IDao;
4
5 public class MetierImpl implements IMetier {
6     /* couplage faible */
7     private IDao dao;
8     @Override
9     public double Calcul() {
10         double temp=dao.getData();
11         double res=temp/Math.cos(Math.PI);
12         return res;
13     }
14
15     /* Injecter dans la variable dao un objet
16     d'une classe qui implémente l'interface IDao */
17
18     // L'injection des dépendances
19     public void setDao(IDao dao) { this.dao = dao; }
20
21
22 }
```

5. Faire l'injection des dépendances :

a. Par instantiation statique

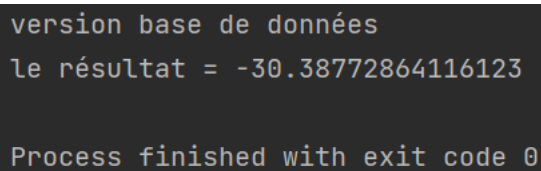
Injection des dépendances par instantiation statique = new

## Version 1 : la base de données



```
2
3 import dao.DaoImpl;
4 import extension.DaoImpl2;
5 import metier.MetierImpl;
6
7 no usages
8 public class Presentation {
9     no usages
10     public static void main(String[] args) {
11         // couplage fort
12
13         // Injection des dépendances par instanciation statique = new
14
15         DaoImpl dao=new DaoImpl();
16         MetierImpl metier=new MetierImpl();
17         metier.setDao(dao);
18         System.out.println("le résultat = "+metier.Calcul());
19     }
20 }
```

Exécution :



```
version base de données
le résultat = -30.38772864116123

Process finished with exit code 0
```

## Version 2 : ajouter une extension et prendre la valeur du capteur

```
IDao.java x DaoImpl.java x IMetier.java x MetierImpl.java x Presentation.java x
1 package pres;
2
3 import dao.DaoImpl;
4 import extension.DaoImpl2;
5 import metier.MetierImpl;
6
7 no usages
8 public class Presentation {
9     no usages
10     public static void main(String[] args) {
11         // couplage fort
12
13         // Injection des dépendances par instanciation statique = new
14
15         DaoImpl2 dao=new DaoImpl2();
16         MetierImpl metier=new MetierImpl();
17         metier.setDao(dao);
18         System.out.println("le résultat = "+metier.Calcul());
19     }
20 }
```

```
IDao.java x DaoImpl.java x IMetier.java x MetierImpl.java x Presentation.java x DaoImpl2.java x
1 package extension;
2
3 import dao.IDao;
4
5 2 usages
6 public class DaoImpl2 implements IDao {
7     1 usage
8     @Override
9     public double getData() {
10         System.out.println("la version capteurs");
11         double temp=100;
12         return temp;
13     }
14 }
```

Exécution :

```
la version capteurs
le résultat = -100.0

Process finished with exit code 0
```

b. Par instanciation dynamique  
J'ai créé un fichier config.xml pour faire la maintenance en modifiant ce fichier sans toucher le code source

```
DaoImpl.java x config.txt x IMetier.java x MetierImpl.java x Presentation.java x presentation2.java x
1 package pres;
2
3 import dao.IDao;
4 import metier.IMetier;
5
6 import java.io.File;
7 import java.io.FileNotFoundException;
8 import java.lang.reflect.Method;
9 import java.util.Scanner;
10
11 no usages
12 public class presentation2 {
13     no usages
14     public static void main(String[] args) throws Exception {
15         Scanner scanner=new Scanner(new File( pathname: "config.txt"));
16
17         String daoClassName=scanner.nextLine(); // connaitre le nom de la classe
18         Class cDao=Class.forName(daoClassName); // charger la classe au mémoire
19         IDao dao= (IDao) cDao.newInstance(); // instancier la classe ou créer l'objet
20
21         String metierClassName=scanner.nextLine();
22         Class cMetier=Class.forName(metierClassName);
23         IMetier metier= (IMetier) cMetier.newInstance();
24
25         Method method=cMetier.getMethod( name: "setDao", IDao.class);
26         // metier.setDao(dao)
27         method.invoke(metier,dao);
28         System.out.println("le résultat= "+metier.Calcul());
29     }
30 }
```

Version 1 : la base de données

```
IDao.java x DaoImpl.java x config.txt x
1 dao.DaoImpl
2 metier.MetierImpl
```

Exécution :

```
version base de données
le résultat= -33.31028639501347

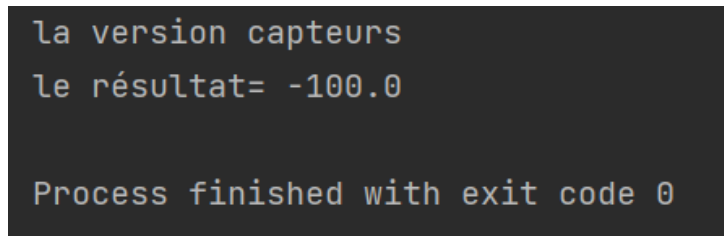
Process finished with exit code 0
```

Version 2 : ajouter une extension et prendre la valeur du capteur



```
1 extension.DaoImpl2
2 metier.MetierImpl
```

Exécution :



```
la version capteurs
le resultat= -100.0

Process finished with exit code 0
```