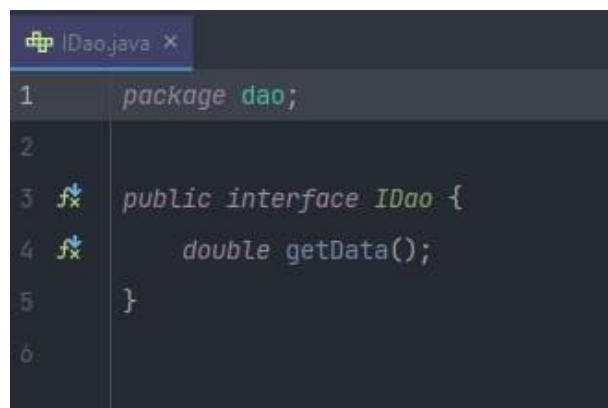


Compte Rendu de l'Activité Pratique 1

Inversion de Contrôle et Injection des Dépendances

1) Créer l'interface IDao :

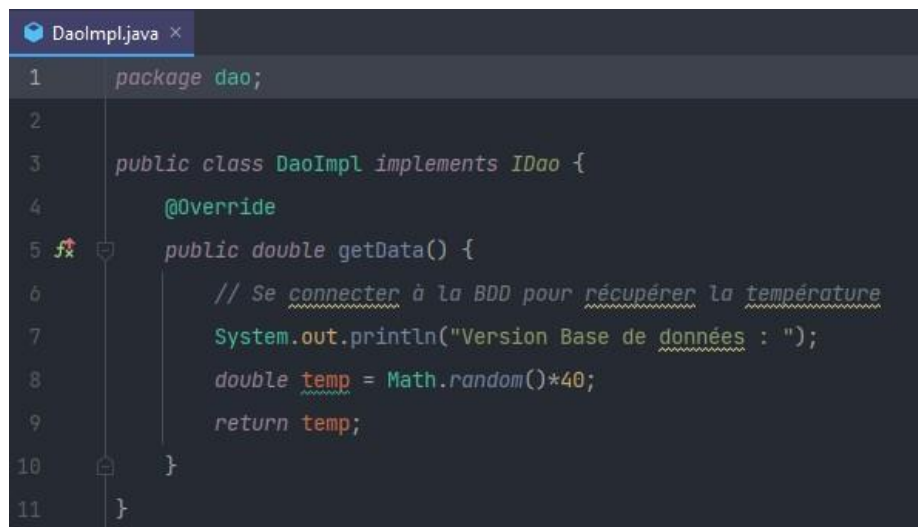
Nous avons commencé notre activité par la création d'une interface appelée IDao, qui contient la méthode abstraite getData() :

A screenshot of an IDE showing the code for IDao.java. The code is as follows:

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

2) Créer une implémentation de cette interface :

Après avoir créé l'interface IDao, nous avons créé la classe DaoImpl qui l'implémente :

A screenshot of an IDE showing the code for DaoImpl.java. The code is as follows:

```
1 package dao;  
2  
3 public class DaoImpl implements IDao {  
4     @Override  
5     public double getData() {  
6         // Se connecter à la BDD pour récupérer la température  
7         System.out.println("Version Base de données : ");  
8         double temp = Math.random()*40;  
9         return temp;  
10    }  
11 }
```

3) Créer l'interface IMetier :

Nous avons par la suite créé l'interface IMetier qui contient les méthodes abstraites calcul() et setDao(IDao dao) :

```
IMetier.java x
1 package metier;
2
3 import dao.IDao;
4
5 public interface IMetier {
6     double calcul();
7     void setDao(IDao dao);
8 }
```

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

Ensuite, nous avons créé l'implémentation MetierImpl de l'interface IMetier en utilisant le couplage faible :

```
MetierImpl.java x
1 package metier;
2
3 import dao.IDao;
4
5 public class MetierImpl implements IMetier {
6     private IDao dao; // Couplage faible
7     @Override
8     public double calcul() {
9         double tmp = dao.getData();
10        double res = tmp * Math.random() * 50 / Math.PI;
11        return res;
12    }
13
14    // Injecter dans la variable dao un objet d'une classe qui implémente l'interface IDAO
15    public void setDao(IDao dao) { this.dao = dao; }
16
17 }
```

5) Faire l'injection des dépendances :

a. Par instantiation statique :

Maintenant, nous allons passer à l'injection des dépendances en utilisant l'instanciation statique :

```
Presentation.java x
1  package pres;
2
3  import dao.DaoImpl;
4  import metier.MetierImpl;
5
6  public class Presentation {
7      public static void main(String[] args) {
8          // Instantiation statique
9          /*
10             MetierImpl metier = new MetierImpl();
11             System.out.println(metier.calcul());
12             !! Ceci nous donnera une Exception de type NullPointerException vu que :
13             le paramètre 'dao' déclaré dans la classe MetierImpl a 'null' comme valeur
14             */
15
16             DaoImpl dao = new DaoImpl();
17             MetierImpl metier = new MetierImpl();
18             metier.setDao(dao);
19             System.out.println("Résultat : " + metier.calcul());
20         }
21     }
```

b. Par instantiation dynamique :

Pour faire l'injection de dépendance par instantiation dynamique, nous allons créer un fichier config.txt dans lequel nous allons ajouter les classes qu'on veut utiliser, et qui vont être appelé dans la couche présentation de manière dynamique, chose qui va rendre notre code fermé à la modification.

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

Ainsi qu'on va ajouter une autre classe d'implémentation de l'interface IDao pour pouvoir permuter entre les deux classes d'implémentation :

```
DaoImpl2.java x
1 package dao;
2
3 public class DaoImpl2 implements IDao {
4     @Override
5     public double getData() {
6         System.out.println("Version Capteurs : ");
7         double temp = 6000;
8         return temp;
9     }
10 }
```

Enfin, nous allons créer la classe de présentation qui appelle les classes d'implémentation de l'interface IDao de manière dynamique à partir du fichierconfig.txt :

```
Presentation2.java x
1 package pres;
2
3 import dao.IDao;
4 import metier.IMetier;
5 import java.io.File;
6 import java.util.Scanner;
7
8 public class Presentation2 {
9     public static void main(String[] args) throws Exception {
10         Scanner scanner = new Scanner(new File( pathname: "config.txt"));
11
12         String daoClassName = scanner.nextLine();
13         Class cDao = Class.forName(daoClassName);
14         IDao dao = (IDao) cDao.newInstance();
15
16         String metierClassName = scanner.nextLine();
17         Class cMetier = Class.forName(metierClassName);
18         IMetier metier = (IMetier) cMetier.newInstance();
19         metier.setDao(dao);
20         System.out.println("Résultat : " + metier.calcul());
21     }
22 }
```

C. En utilisant le Framework Spring :

- Version XML :

Nous avons créé le fichier applicationContext.xml dans le dossier ressources après avoir ajouté toutes les dépendances Spring nécessaires dans le fichier pom.xml :

```
applicationContext.xml x
Application context not configured for this file
1  <?xml version="1.0" encoding="UTF-8"?>
2  <beans xmlns="http://www.springframework.org/schema/beans"
3      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4      xsi:schemaLocation="http://www.springframework.org/schema/beans
5          http://www.springframework.org/schema/beans/spring-beans.xsd">
6      <bean id="dao" class="dao.DaoImpl"/>
7      <bean id="metier" class="metier.MetierImpl">
8          <property name="dao" ref="dao"/>
9      </bean>
10 </beans>
```

Ensuite, nous avons créé la classe PresentationSpringXML :

```
PresentationSpringXML.java x
1  package pres;
2
3  import metier.IMetier;
4  import org.springframework.context.ApplicationContext;
5  import org.springframework.context.support.ClassPathXmlApplicationContext;
6
7  public class PresentationSpringXML {
8      public static void main(String[] args) {
9          ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");
10         IMetier metier = (IMetier) context.getBean("metier");
11         System.out.println(metier.calcul());
12     }
13 }
```

```
PresentationSpringXML x
↑ "C:\Program Files\Java\jdk1.8.0_321\bin\java.exe" ...
↓ Version Base de données :
93.2883381851913
Process finished with exit code 0
```

Si on change la classe d'implémentation dans le fichier xml :

```
6 <bean id="dao" class="dao.DaoImpl2"/>
```

```
PresentationSpringXML x
↑ "C:\Program Files\Java\jdk1.8.0_321\bin\java.exe" ...
↓ Version Capteurs :
82898.63823716182
Process finished with exit code 0
```

- **Version Spring :**

On peut utiliser le framework Spring pour faire l'injection de dépendances. Pour ce faire, on peut utiliser 3 techniques : soit l'instanciation par setter, soit par l'annotation Autowired, soit par un constructeur.

```
DaoImpl.java x
1 package dao;
2
3 import org.springframework.stereotype.Component;
4
5 @Component("dao")
6 public class DaoImpl implements IDao {
7     @Override
8     public double getData() {
9         // Se connecter à la BDD pour récupérer la température
10        System.out.println("Version Base de données : ");
11        double temp = Math.random()*40;
12        return temp;
13    }
14 }
```

```
DaoImpl2.java x
1 package dao;
2
3 import org.springframework.stereotype.Component;
4
5 @Component("dao2")
6 public class DaoImpl2 implements IDao {
7     @Override
8     public double getData() {
9         System.out.println("Version Capteurs : ");
10        double temp = 6000;
11        return temp;
12    }
13 }
```

```
DaoImplWS.java x
1 package dao;
2
3 import org.springframework.stereotype.Component;
4
5 @Component("daoWS")
6 public class DaoImplWS implements IDao{
7     @Override
8     public double getData() {
9         System.out.println("Version Web Service : ");
10        return 100;
11    }
12 }
```

```

MetierImpl.java x
1  package metier;
2
3  import dao.IDao;
4  import org.springframework.beans.factory.annotation.Autowired;
5  import org.springframework.beans.factory.annotation.Qualifier;
6  import org.springframework.stereotype.Component;
7
8  @Component
9  public class MetierImpl implements IMetier {
10
11     @Autowired
12     @Qualifier("daoWS")
13     private IDao dao; // Couplage faible
14
15     @Override
16     public double calcul() {
17         double tmp = dao.getData();
18         double res = tmp * Math.random() * 50 / Math.PI;
19         return res;
20     }
21
22     // Injecter dans la variable dao un objet d'une classe qui implémente l'interface IDAO
23     public void setDao(IDao dao) { this.dao = dao; }
24
25 }

```

```

MetierImpl.java x  PresentationSpringAnnotations.java x
1  package pres;
2
3  import metier.IMetier;
4  import org.springframework.context.ApplicationContext;
5  import org.springframework.context.annotation.AnnotationConfigApplicationContext;
6
7  public class PresentationSpringAnnotations {
8
9     public static void main(String[] args) {
10         ApplicationContext context = new AnnotationConfigApplicationContext( ...basePackages: "dao", "metier");
11         IMetier metier = context.getBean(IMetier.class);
12         System.out.println("Résultat = " + metier.calcul());
13     }
14 }

```

```

PresentationSpringAnnotations x
↑ "C:\Program Files\Java\jdk1.8.0_321\bin\java.exe" ...
↓ Version Web Service :
⇌ Résultat = 741.8856589895548
⏮
🖨 Process finished with exit code 0

```