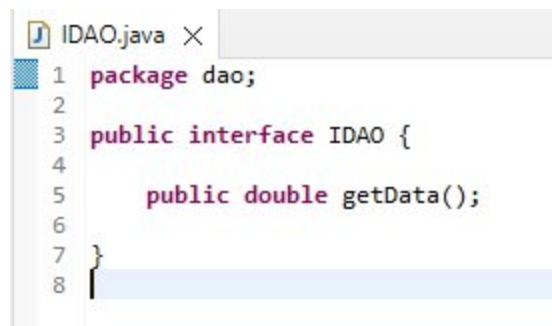


Compte rendu : Activity 1 IOC et Injection des dépendances

l'injection de dépendances a pour but de rendre votre application plus maintenable en utilisant le concept du couplage faible.

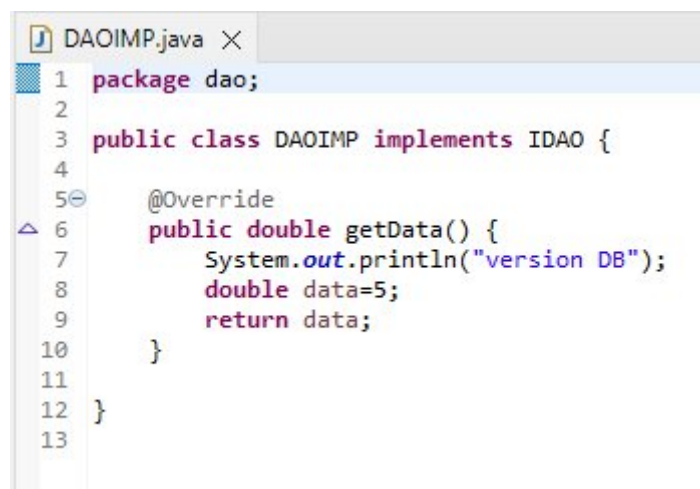
1 – création des interfaces :

Interface IDao :

A screenshot of a code editor showing the IDAO.java file. The code defines a package 'dao' and a public interface 'IDAO' with a single method 'getData()' that returns a 'double' value.

```
1 package dao;
2
3 public interface IDAO {
4
5     public double getData();
6
7 }
8
```

Une implémentation de cette interface :

A screenshot of a code editor showing the DAOIMP.java file. The code defines a package 'dao' and a public class 'DAOIMP' that implements the 'IDAO' interface. It overrides the 'getData()' method to print 'version DB' and return the value 5.

```
1 package dao;
2
3 public class DAOIMP implements IDAO {
4
5     @Override
6     public double getData() {
7         System.out.println("version DB");
8         double data=5;
9         return data;
10    }
11
12 }
13
```

Interface IMetier :

```
IMetier.java X
1 package metier;
2
3 public interface IMetier {
4     public double calcul();
5 }
6
7
8
```

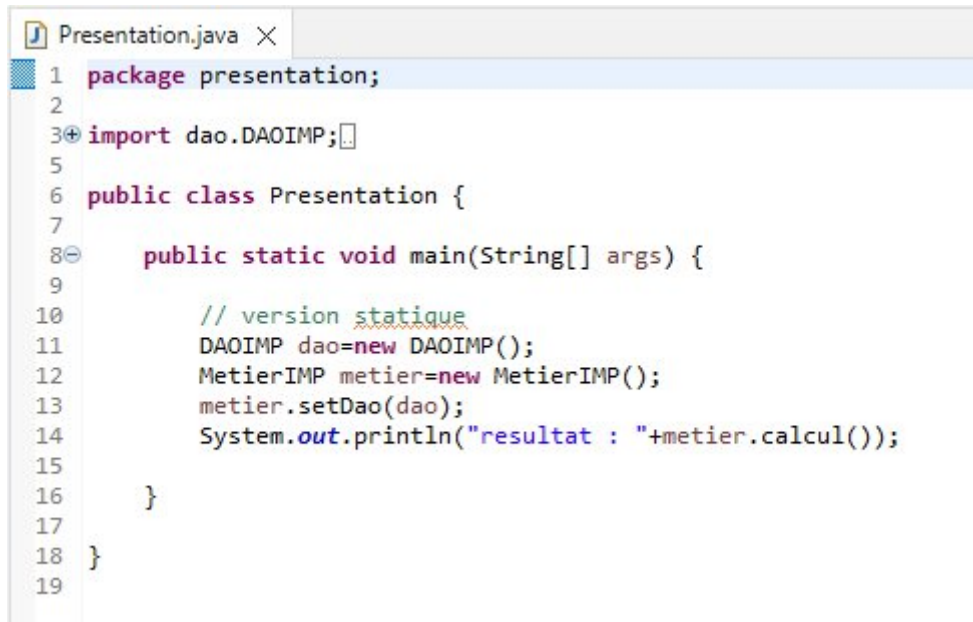
Une implémentation de l'interface IMetier :

```
MetierIMP.java X
1 package metier;
2
3 import dao.IDAO;
4
5 public class MetierIMP implements IMetier {
6     private IDAO dao;
7
8     @Override
9     public double calcul() {
10         double data=dao.getData();
11         double res=data*2;
12         return res;
13     }
14
15     public void setDao(IDAO dao) {
16         this.dao = dao;
17     }
18
19 }
20
21
22
23
```

Pour utiliser le couplage faible, le type de l'objet dont cette implémentation dépend, doit être l'interface implémenté par la classe de cet objet.

2 - Injection des dépendances :

A – instantiation statique :

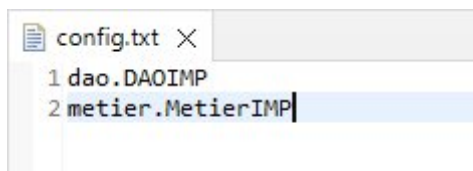


```
1 package presentation;
2
3 import dao.DAOIMP;
4
5
6 public class Presentation {
7
8     public static void main(String[] args) {
9
10         // version statique
11         DAOIMP dao=new DAOIMP();
12         MetierIMP metier=new MetierIMP();
13         metier.setDao(dao);
14         System.out.println("resultat : "+metier.calcul());
15
16     }
17
18 }
19
```

B – instantiation dynamique :

Pour instancier les objets d'une manière dynamique les noms des classes doivent être stockées dans un fichier de configuration.

Fichier de configuration :



```
1 dao.DAOIMP
2 metier.MetierIMP
```

Cette méthode d'instanciation va nous permettre d'injecter les dépendances a partir de ce fichier de configuration.

Si par la suite on veut effectuer des modification, il suffit de changer le nom de l'implémentation par la nouvelle implémentation dans le fichier.

Instanciation dynamique a partir d'un fichier de configuration :

```
Presentation2.java X
1 package presentation;
2
3 import java.io.File;
4
11
12 public class Presentation2 {
13
14     public static void main(String[] args) {
15         // version dynamique
16         try {
17             Scanner input=new Scanner(new File("src/config.txt"));
18             String daoName=input.nextLine();
19             String metierName=input.nextLine();
20
21             Class classDAO=Class.forName(daoName);
22             Class classMetier=Class.forName(metierName);
23
24             IDAO dao=(IDAO)classDAO.newInstance();
25             IMetier metier=(IMetier)classMetier.newInstance();
26
27             Method method=classMetier.getMethod("setDao", IDAO.class);
28             method.invoke(metier, dao);
29             System.out.println(metier.calcul());
30
31         } catch (Exception e) {
32             e.printStackTrace();
33         }
34     }
35 }
36
37
38 }
```

C – Spring Framework :

- Version XML :

Fichier XML :

```
*applicationContext.xml X
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
7     <bean id="dao" class="dao.DAOIMP2"></bean>
8     <bean id="metier" class="metier.MetierIMP">
9         <property name="dao" ref="dao"></property>
10    </bean>
11
12 </beans>
13
```

Présentation Version Spring XML :

```
PresentationSpring.java X
1 package presentation;
2
3 import org.springframework.context.ApplicationContext;
4
5
6
7
8 public class PresentationSpring {
9
10     public static void main(String[] args) {
11         ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");
12         IMetier metier=(IMetier) context.getBean("metier");
13         System.out.println(metier.calcul());
14     }
15 }
16
17
```

– Version Annotations :

Les Annotations :

```
PresentationSpring.java MetierIMP.java DAOIMP.java X
1 package dao;
2
3 import org.springframework.stereotype.Component;
4
5 @Component("dao")
6 public class DAOIMP implements IDAO {
7
8
9     public double getData() {
10         System.out.println("version DB");
11         double data=5;
12         return data;
13     }
14
15 }
16
```

```
PresentationSpring.java X MetierIMP.java X DAOIMP.java
1 package metier;
2
3+ import org.springframework.beans.factory.annotation.Autowired;
7
8 @Component
9 public class MetierIMP implements IMetier {
10-   @Autowired
11   private IDAO dao;
12
13
14-   /*
15   public MetierIMP(IDAO dao) {
16       this.dao = dao;
17   }*/
18
19-   public double calcul() {
20       double data=dao.getData();
21       double res=data*2;
22       return res;
23   }
24
25-   public void setDao(IDAO dao) {
26       this.dao = dao;
27   }
28
```

Présentation Spring Annotations :

```
MetierIMP.java DAOIMP.java PresentationSpringAnnotation.java X
1 package presentation;
2
3+ import javax.swing.plaf.synth.SynthOptionPaneUI;
9
10 public class PresentationSpringAnnotation {
11
12-   public static void main(String[] args) {
13       ApplicationContext context=new AnnotationConfigApplicationContext("dao","metier");
14       IMetier metier=context.getBean(IMetier.class);
15       System.out.println(metier.calcul());
16
17   }
18
19 }
```

Pour assurer que votre application soit maintenable et faciliter ce processus, il est fortement conseillé d'utiliser la notion d'injection des dépendances en utilisant le couplage faible.