



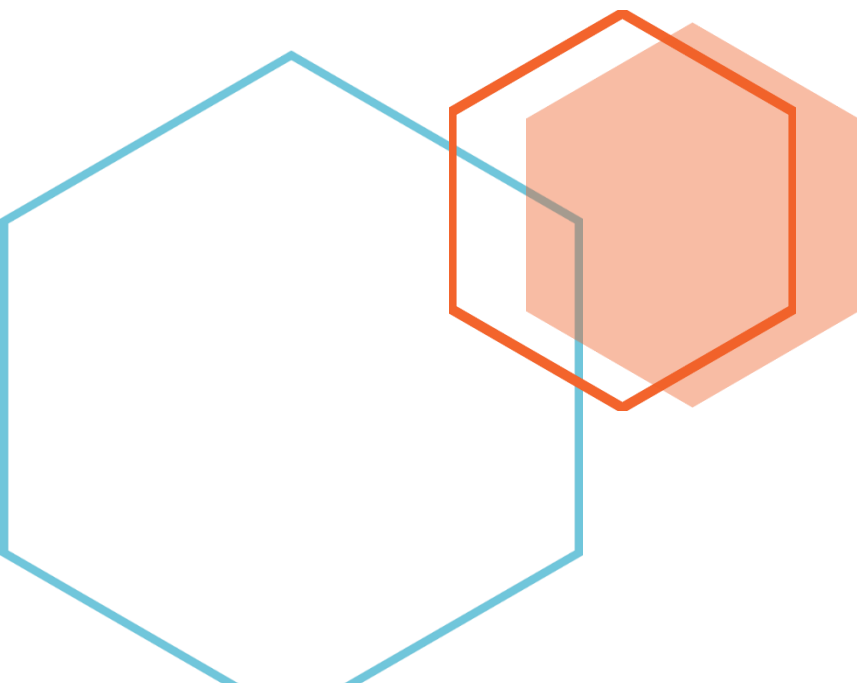
# [Compte Rendu]

---

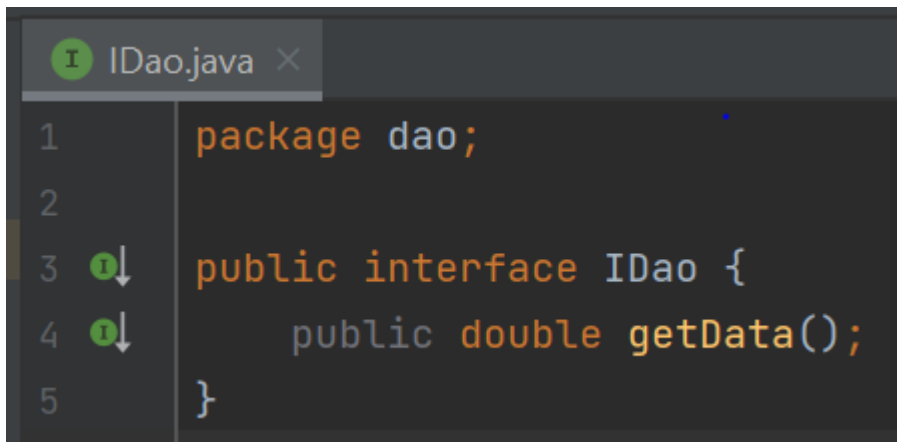
## Inversion de contrôle et Injection des dépendances

Architecture JEE et Middlewares

L'Ecole Normale Supérieure de l'Enseignement Technique de  
Mohammedia (ENSET)



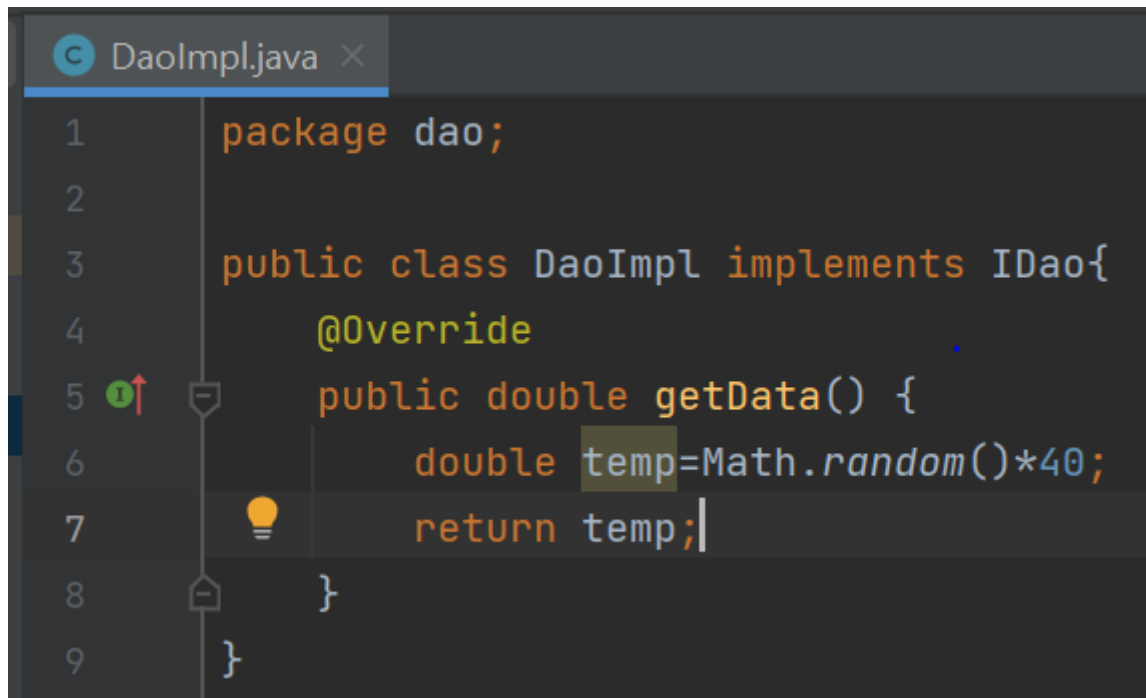
### 1. Créer l'interface IDao



The screenshot shows a code editor with a tab labeled 'IDao.java'. The code defines a package 'dao' and a public interface 'IDao' with a single method 'getData()' that returns a 'double'.

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

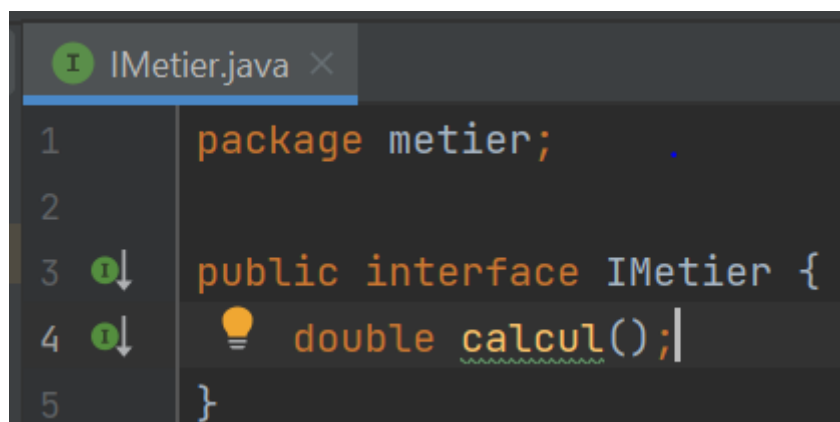
### 2. Créer une implémentation de cette interface



The screenshot shows a code editor with a tab labeled 'DaoImpl.java'. The code defines a package 'dao' and a public class 'DaoImpl' that implements 'IDao'. It overrides the 'getData()' method to return a random double value between 0 and 40.

```
1 package dao;  
2  
3 public class DaoImpl implements IDao{  
4     @Override  
5     public double getData() {  
6         double temp=Math.random()*40;  
7         return temp;  
8     }  
9 }
```

### 3. Créer l'interface IMetier



The screenshot shows a code editor with a tab labeled 'IMetier.java'. The code defines a package 'metier' and a public interface 'IMetier' with a single method 'calcul()' that returns a 'double'.

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

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

```
MetierImpl.java x
2  import dao.IDao;
3  public class MetierImpl implements IMetier{
4      //Couplage faible
5      //new == couplage fort
6      private IDao dao;
7      @Override
8      public double calcul() {
9          double tmp= dao.getData();
10         double res=tmp*540/Math.cos(tmp*Math.PI);
11         return res;
12     }
13     //affecter une valeur à dao
14     /** Injecter dans la variable dao un objet d'une
15      * classe qu'implémente l'interface IDao
16      */
17     public void setDao(IDao dao) {
18         this.dao = dao;
19     }
20 }
```

#### 5. Faire l'injection des dépendances :

a. Par instanciation statique

```
Presentation.java x
1  package pres;
2  import dao.DaoImpl;
3  import ext.DaoImpl2;
4  import metier.MetierImpl;
5  public class Presentation {
6      public static void main(String[] args) {
7          /* Injection des dépendances par instanciation statique
8           * statique => new qu'est le vrai probleme de la maintenance
9           */
10         DaoImpl dao=new DaoImpl();
11         //DaoImpl2 dao=new DaoImpl2();
12         MetierImpl metier=new MetierImpl();
13         metier.setDao(dao);
14         System.out.println("Résultats = "+metier.calcul());
15     }
16 }
```

```
Presentation x
"C:\Program Files\Java\jdk1.8.0_321\bin\java.exe" ...
DaoImpl 1
Résultats = 80806.46976433831
Process finished with exit code 0
```

Avec l'extension DaoImpl 2 :

```
Presentation x
"C:\Program Files\Java\jdk1.8.0_321\bin\java.exe" ...
DaoImpl 2
Résultats = 3240000.0
Process finished with exit code 0
```

## b. Par instantiation dynamique

```
package pres;
import ...
public class Pres2 {
    public static void main(String[] args) throws FileNotFoundException, ClassNotFoundException,
        InstantiationException, IllegalAccessException,
        NoSuchMethodException, InvocationTargetException {

        Scanner scanner=new Scanner(new File( pathname: "config.txt"));
        /**Lire le premier ligne du fichier config.txt*/
        String daoClassName=scanner.nextLine();
        /**Charger la classe dans la mémoire*/
        Class cDao=Class.forName(daoClassName);
        /**Instancier la classe fait appel au constructeur par défaut
         * newInstance() retourne une classe de type Object*/
        IDao dao =(IDao) cDao.newInstance();

        String metierClassName= scanner.nextLine();
        Class cMetier=Class.forName(metierClassName);
        IMetier metier=(IMetier) cMetier.newInstance();

        Method method=cMetier.getMethod( name: "setDao",IDao.class);
        /** Injection des dépendances d'une manière dynamique*/
        method.invoke(metier,dao);//exécuter la methode
        System.out.println("Résultat=>" +metier.calcul());
    }
}
```



### c. En utilisant le Framework Spring - Version XML

On crée un projet maven et dans le fichier pom.xml on ajoute 3 modules de Spring Spring-core, Spring-context et Spring-beans.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www
  <modelVersion>4.0.0</modelVersion>
  <groupId>ma.enset</groupId>
  <artifactId>TP1_Maven</artifactId>
  <version>1.0-SNAPSHOT</version>
  <properties>
    <maven.compiler.source>8</maven.compiler.source>
    <maven.compiler.target>8</maven.compiler.target>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-core</artifactId>
      <version>5.3.16</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-context</artifactId>
      <version>5.3.16</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-beans</artifactId>
      <version>5.3.16</version>
    </dependency>
  </dependencies>
</project>
```

Après on crée un fichier de configuration XML.

```
<?xml version="1.0" encoding="UTF-8"?>|
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans"
       >
    <!-- Créer un objet de la classe DaoImpl avec le nom dao-->
    <bean id="dao" class="dao.DaoImpl"></bean>
    <!-- lorsqu'on crée un objet metier on fait appelle à set
         pour faire l'injection des dépendances via le setter-->
    <bean id="metier" class="metier.MetierImpl">
        <property name="dao" ref="dao"></property>
    </bean>
</beans>
```

```
IOC_Spring_XML.java x
1  package pres;
2  import metier.IMetier;
3  import org.springframework.context.ApplicationContext;
4  import org.springframework.context.support.ClassPathXmlApplicationContext;
5
6  public class IOC_Spring_XML {
7  public static void main(String[] args) {
8      ApplicationContext context=
9          new ClassPathXmlApplicationContext( "applicationContext.xml");
10     // Objet de type IMetier
11     IMetier metier=(IMetier) context.getBean( s: "metier");
12     System.out.println(metier.calcul());
13 }
14 }
```

Exécution pour les 3 versions :

class="dao.DaoImpl"	class="ext.DaoImpl2"	class="ext.DaoImplVWS"
IOC_Spring_XML x "C:\Program Files\Java DaoImpl 1 4528.052501175052	IOC_Spring_XML x "C:\Program Files\Java\jdk1 DaoImpl 2 3240000.0	"C:\Program Files\Java Version web service 48600.0

## - Version annotations

```
package dao;
import org.springframework.stereotype.Component;

/** L'annotation @Component pour dire à Spring au démarrage
 * à chaque fois qu'il trouve une classe précédé par
 * @Component il va l'instancier et lui donner comme nom dao
 */
@Component("dao")
public class DaoImpl implements IDao{
    @Override
    public double getData() {
        System.out.println("DaoImpl 1");
        double temp=Math.random()*40;
        return temp;
    }
}
```

```
package metier;
import ...
@Component
public class MetierImpl implements IMetier{
    /**@Autowired pour faire l'injection des dépendances
     * on demande à spring au moment qu'on il instancie la classe
     * MetierImpl de chercher parmi les objets qu'il a crée un objet
     * de type IDao et l'injecter dans la variable dao
     */
    @Autowired
    private IDao dao;
    @Override
    public double calcul() {
        double tmp= dao.getData();
        double res=tmp*540/Math.cos(tmp*Math.PI);
        return res;
    }
}
```

```

1 package pres;
2 import metier.IMetier;
3 import org.springframework.context.ApplicationContext;
4 import org.springframework.context.annotation.AnnotationConfigApplicationContext;
5 public class IOC8Spring_Annotation {
6     public static void main(String[] args) {
7         //Scanner les packages dao & metier
8         ApplicationContext context=
9             new AnnotationConfigApplicationContext( ...basePackages: "dao","metier");
10        IMetier metier=context.getBean(IMetier.class);
11        System.out.println(metier.calcul());
12    }
13 }

```

Pour le conflit des beans :

```

package pres;
import metier.IMetier;
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
public class IOC8Spring_Annotation {
    public static void main(String[] args) {
        //Scanner les packages dao & metier
        ApplicationContext context=
            new AnnotationConfigApplicationContext( ...basePackages: "dao","metier","ext");
        IMetier metier=context.getBean(IMetier.class);
        System.out.println(metier.calcul());
    }
}

```

On donne pour chaque extension un nom de bean et on utilise l'annotation `@Qualifier` ou on indique le nom de la classe à instancier.

```

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

```



```
MetierImpl.java x DaolImpl.java x
1 package metier;
2 import ...
6 @Component
7 public class MetierImpl implements IMetier{
8     /**@Autowired pour faire l'injection des dépendances
9      * on demande à spring au moment qu'on il instancie la classe
10     * MetierImpl de chercher parmi les objets qu'il a crée un objet
11     * de type IDao et l'injecter dans la variable dao
12     */
13     @Autowired
14     @Qualifier("dao2")
15     private IDao dao;
16     /*public MetierImpl(IDao dao) {
17         this.dao = dao;
18     }*/
19     @Override
20     public double calcul() {
21         double tmp= dao.getData();
22         double res=tmp*540/Math.cos(tmp*Math.PI);
23         return res;
24     }
```