

*Architecture JEE et Middlewares*

# **Compte-rendu**

*Activité Pratique N1*

*Inversion de contrôle et Injection des dépendances*

## **Partie 1**

**Nom et Prénom :** BOUSSAIRI Ikram

**E-mail :** [ikramboussairi12@gmail.com](mailto:ikramboussairi12@gmail.com)

**ENSET GLSID – S4**

**Professeur :** Mohamed YOUSSEF



**Java EE™**

## Sommaire

<b>Avant-propos .....</b>	<b>3</b>
<b>Partie 1 .....</b>	<b>4</b>
A. Mise en contexte .....	4
B. Réalisation .....	4
1. Créer l'interface IDao avec une méthode getDate .....	4
2. Créer une implémentation de cette interface .....	4
3. Créer l'interface IMetier avec une méthode calcul .....	5
<b>Conclusion .....</b>	<b>9</b>

## Avant-propos

Il est difficile de développer un système logiciel qui respecte l'ensemble des exigences fonctionnelles, satisfaire les besoins métiers de projet, ainsi que les exigences techniques à savoir : la performance ; le temps de réponse de notre système, l'équilibrage de charges et tolérance aux pannes, le Problème de montée en charge. La maintenance, c'est-à-dire notre application doit être facile à maintenir et pour se faire l'application doit évoluer dans le temps ainsi qu'il doit être fermée à la modification et ouverte à l'extension. Sans oublier la sécurité et la persistance des données dans des SGBD appropriés, etc.

En effet, afin de pouvoir relever ce défi il va falloir utiliser l'expérience des autres ; autrement dit bâtir notre application sur une architecture d'entreprise. L'inversion de contrôle alors prend en charge du flot d'exécution de notre logiciel, c'est-à-dire le flot d'exécution d'un logiciel n'est plus sous le contrôle direct de l'application elle-même mais du Framework. D'autre part, le développeur s'occupe uniquement du code métier (exigences fonctionnelles).

Dans ce contexte, J'ai eu l'occasion d'appliquer l'ensemble de concepts de bases de l'inversion de contrôle et l'injection de dépendance à travers ces deux activités pratiques.

Ce compte rendu se compose de deux parties, la première partie consiste sur la création des différentes classes et interfaces en utilisant le couplage faible et en utilisant l'injection des dépendances de différentes manières ; instanciation statique, instanciation dynamique et en utilisant le Framework Spring (Version XML et version annotations). La deuxième partie consiste sur la réalisation d'un Mini Projet ; un Framework de l'injection des dépendances.

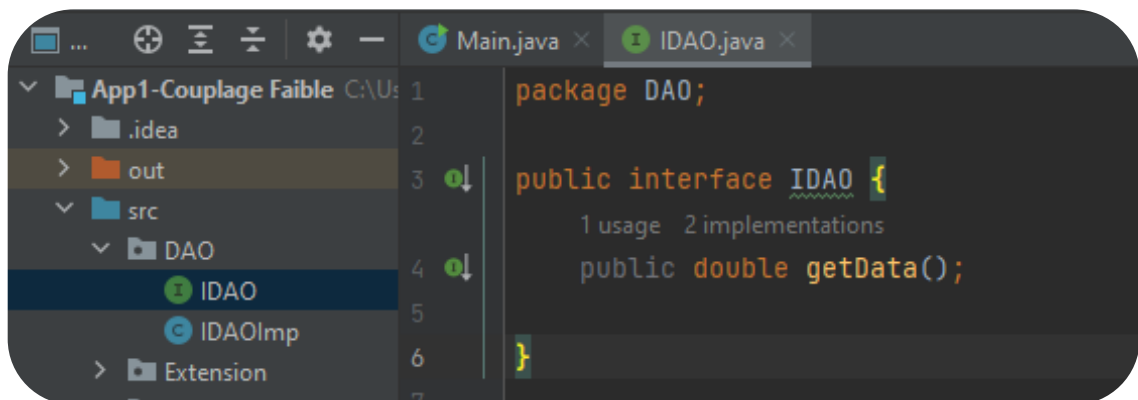
## Partie 1

### A. Mise en contexte

1. Créer l'interface IDao avec une méthode getDate
2. Créer une implémentation de cette interface
3. Créer l'interface IMetier avec une méthode calcul
4. Créer une implémentation de cette interface en utilisant le couplage faible
5. Faire l'injection des dépendances :
  - a. Par instanciation statique
  - b. Par instanciation dynamique
  - c. En utilisant le Framework Spring
    - Version XML
    - Version annotations

### B. Réalisation

#### 1. Créer l'interface IDao avec une méthode getDate

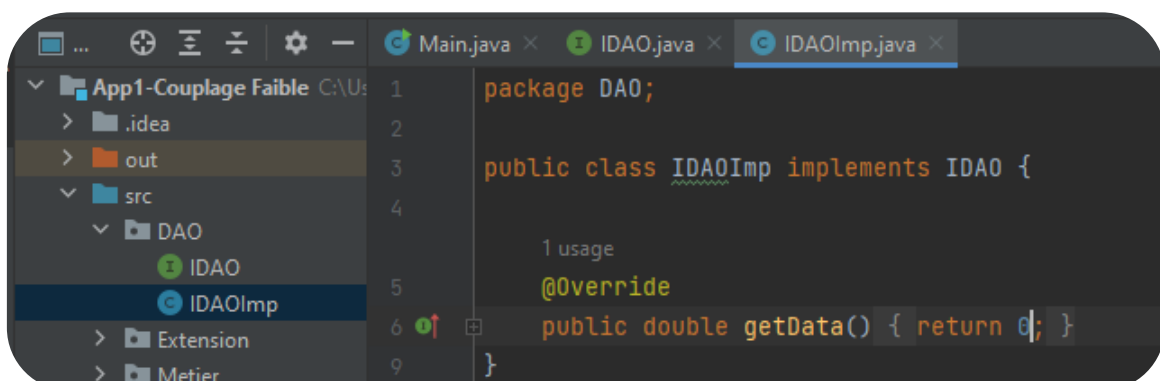


The screenshot shows an IDE with a project named 'App1-Couplage Faible'. The file explorer on the left shows the directory structure: 'src' > 'DAO' > 'IDAO'. The 'IDAO.java' file is open in the editor. The code defines a package 'DAO' and a public interface 'IDAO' with a method 'public double getData();'. The IDE shows 1 usage and 2 implementations for this interface.

```
package DAO;

public interface IDAO {
    1 usage 2 implementations
    public double getData();
}
```

#### 2. Créer une implémentation de cette interface

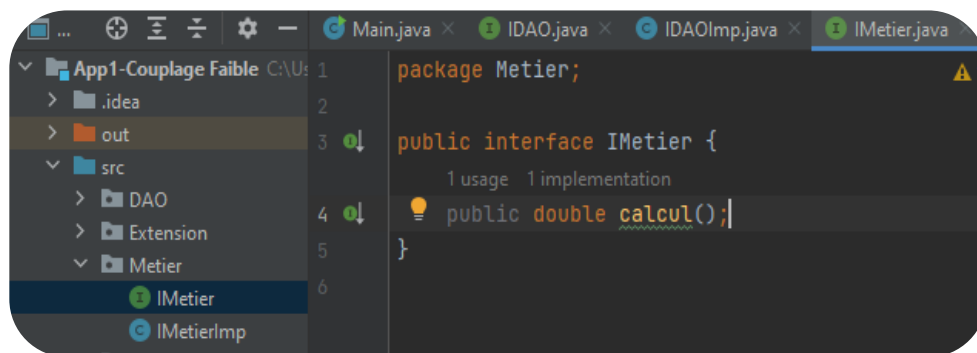


The screenshot shows the same IDE with the 'IDAOImp.java' file open. The code defines a package 'DAO' and a public class 'IDAOImp' that implements 'IDAO'. It includes an '@Override' annotation and a method 'public double getData() { return 0; }'. The IDE shows 1 usage for this implementation.

```
package DAO;

public class IDAOImp implements IDAO {
    1 usage
    @Override
    public double getData() { return 0; }
}
```

### 3. Créer l'interface IMetier avec une méthode calcul

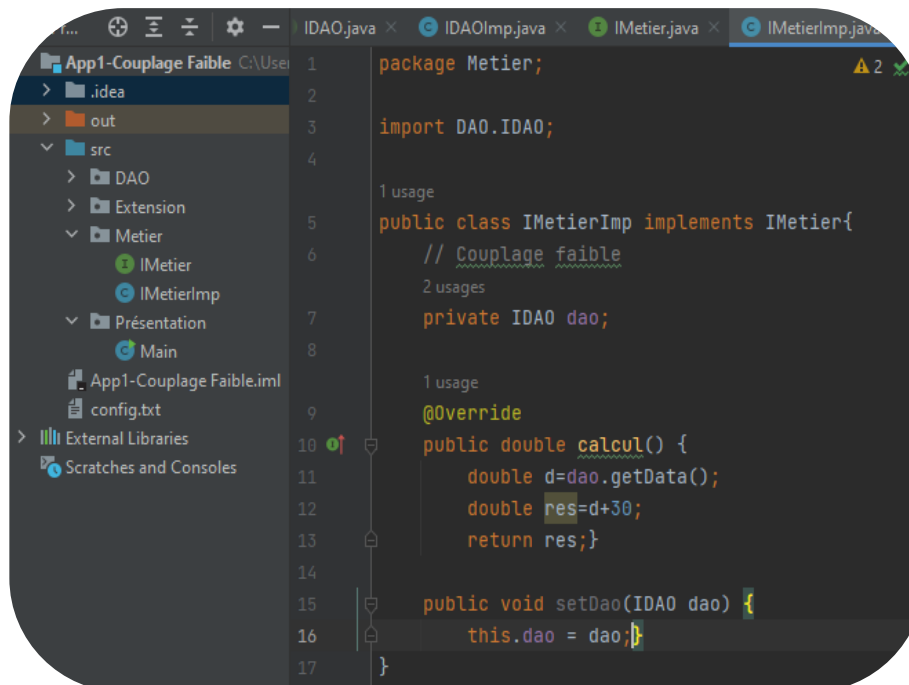


The screenshot shows an IDE with a project named 'App1-Couplage Faible'. The file explorer on the left shows the project structure: .idea, out, src (containing DAO, Extension, and Metier). The 'Metier' package is selected, and the 'IMetier' interface is being created. The code in the editor is as follows:

```
package Metier;

public interface IMetier {
    1 usage 1 implementation
    public double calcul();
}
```

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



The screenshot shows the 'IMetierImp' class being implemented. The file explorer on the left shows the project structure, with 'IMetierImp' selected under the 'Metier' package. The code in the editor is as follows:

```
package Metier;

import DAO.IDAO;

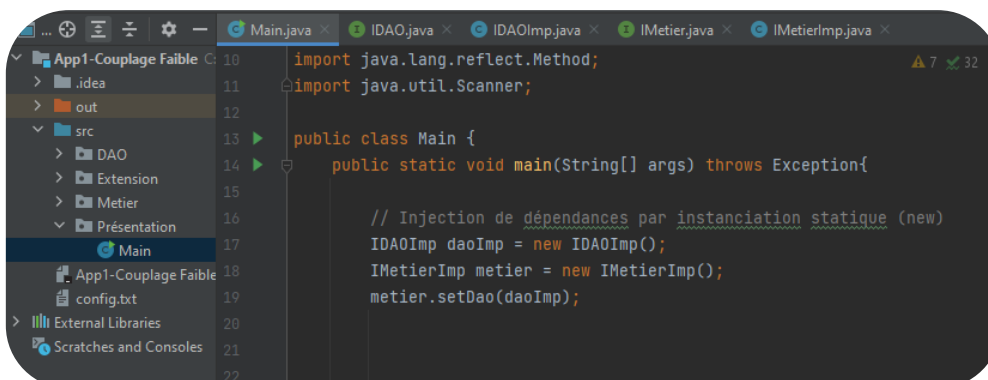
1 usage
public class IMetierImp implements IMetier{
    // Couplage faible
    2 usages
    private IDAO dao;

    1 usage
    @Override
    public double calcul() {
        double d=dao.getData();
        double res=d+30;
        return res;}

    public void setDao(IDAO dao) {
        this.dao = dao;}
}
```

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

#### a. Par instantiation statique



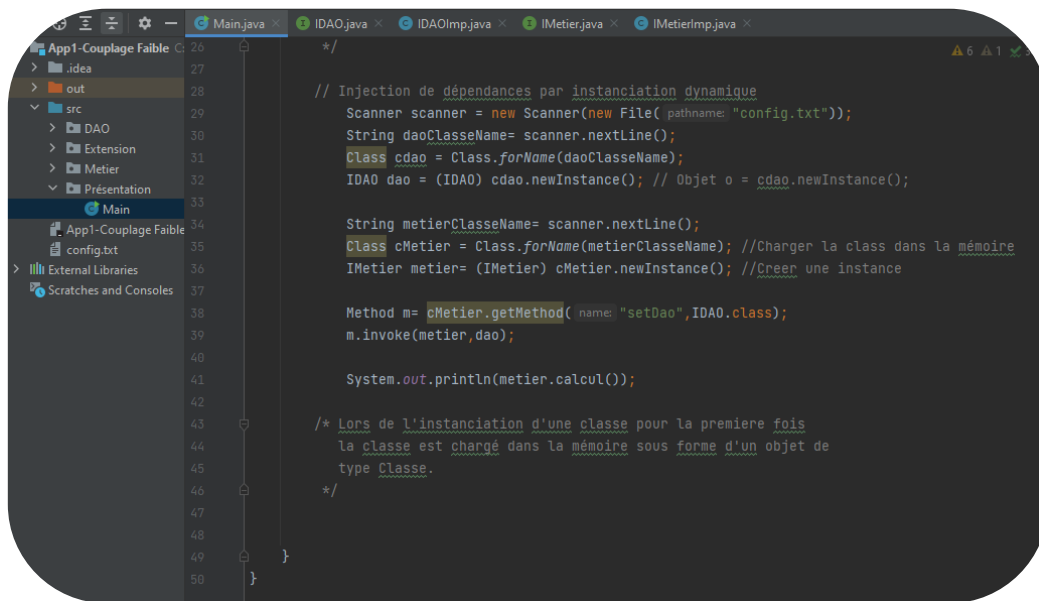
The screenshot shows the 'Main' class with static instantiation of dependencies. The file explorer on the left shows the project structure, with 'Main' selected under the 'Présentation' package. The code in the editor is as follows:

```
import java.lang.reflect.Method;
import java.util.Scanner;

public class Main {
    public static void main(String[] args) throws Exception{

        // Injection de dépendances par instantiation statique (new)
        IDAOImp daoImp = new IDAOImp();
        IMetierImp metier = new IMetierImp();
        metier.setDao(daoImp);
    }
}
```

## b. Par instantiation dynamique



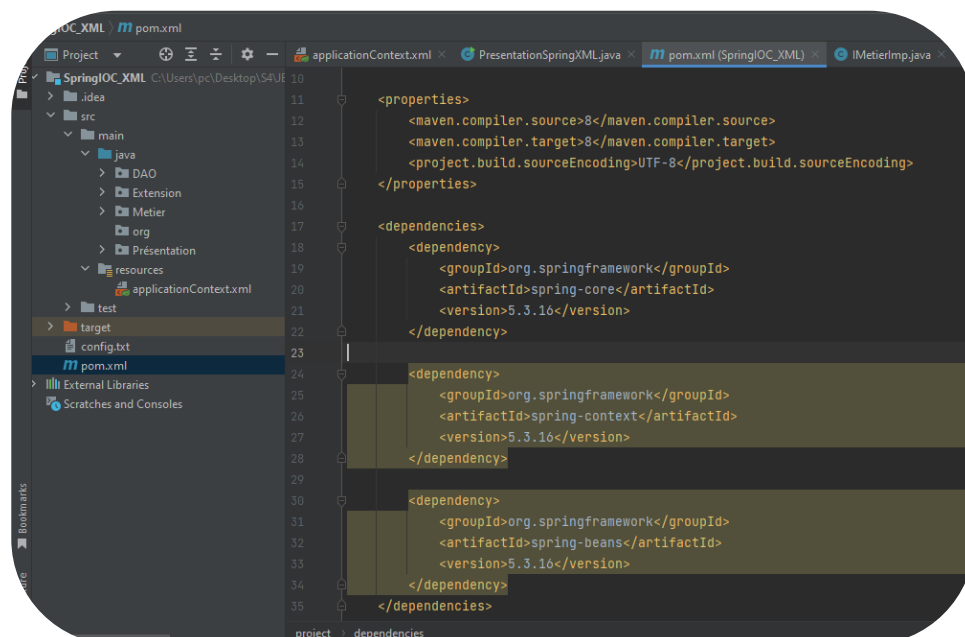
```
26 */
27
28 // Injection de dépendances par instantiation dynamique
29 Scanner scanner = new Scanner(new File( pathname: "config.txt"));
30 String daoClassName= scanner.nextLine();
31 Class cdao = Class.forName(daoClassName);
32 IDAO dao = (IDAO) cdao.newInstance(); // Objet o = cdao.newInstance();
33
34 String metierClassName= scanner.nextLine();
35 Class cMetier = Class.forName(metierClassName); //Charger la class dans la mémoire
36 IMetier metier= (IMetier) cMetier.newInstance(); //Créer une instance
37
38 Method m= cMetier.getMethod( name: "setDao", IDAO.class);
39 m.invoke(metier,dao);
40
41 System.out.println(metier.calcul());
42
43 /* Lors de l'instanciation d'une classe pour la première fois
44 la classe est chargée dans la mémoire sous forme d'un objet de
45 type Classe.
46 */
47
48
49 }
50 }
```

## c. En utilisant le Framework Spring

Dans cette partie j'ai utilisé un projet Maven ; un outil de gestion et d'automatisation de production des projets logiciels Java en général et Java EE en particulier. En effet Maven va nous aider à gérer les différentes dépendances de notre projet Spring.

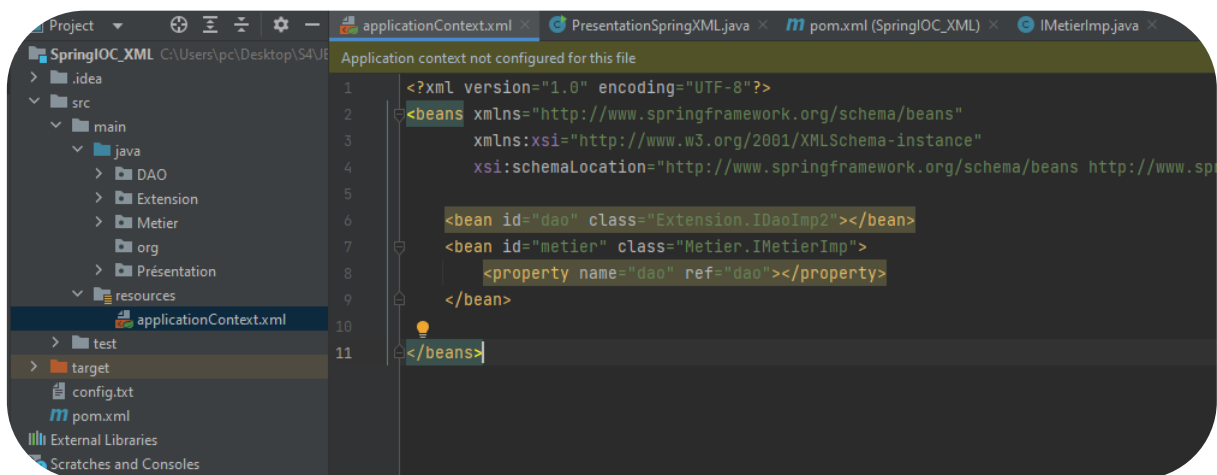
### - Version XML

**Etape 1 :** Ajouter les dépendances nécessaires dans POM.xml



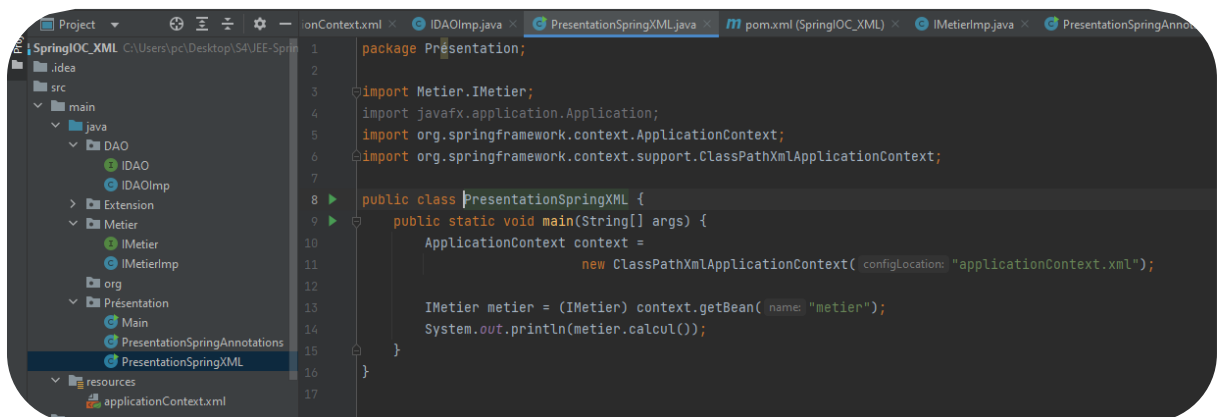
```
10
11 <properties>
12   <maven.compiler.source>8</maven.compiler.source>
13   <maven.compiler.target>8</maven.compiler.target>
14   <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
15 </properties>
16
17 <dependencies>
18   <dependency>
19     <groupId>org.springframework</groupId>
20     <artifactId>spring-core</artifactId>
21     <version>5.3.16</version>
22   </dependency>
23
24   <dependency>
25     <groupId>org.springframework</groupId>
26     <artifactId>spring-context</artifactId>
27     <version>5.3.16</version>
28   </dependency>
29
30   <dependency>
31     <groupId>org.springframework</groupId>
32     <artifactId>spring-beans</artifactId>
33     <version>5.3.16</version>
34   </dependency>
35 </dependencies>
```

## Etape 2 : Création du fichier XML



```
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 http://www.sp
5
6       <bean id="dao" class="Extension.IDaoImp2"></bean>
7       <bean id="metier" class="Metier.IMetierImp">
8         <property name="dao" ref="dao"></property>
9       </bean>
10
11 </beans>
```

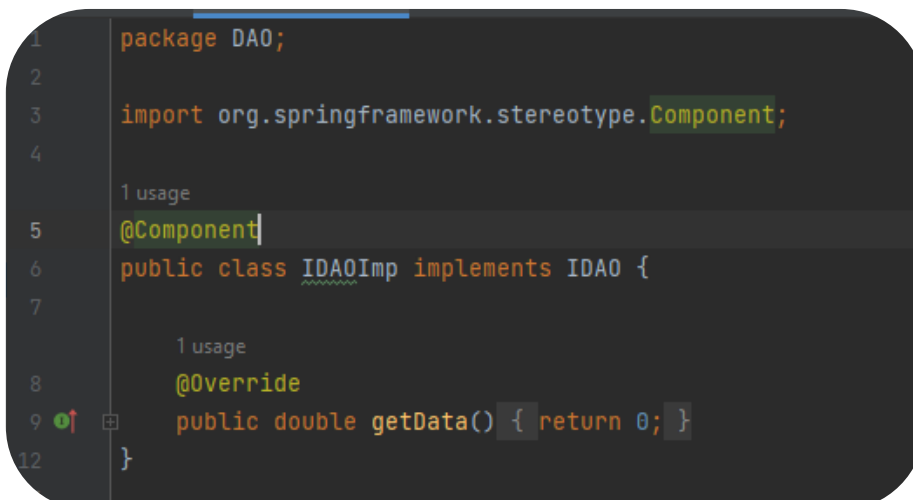
## Etape 3 : main



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

### - Version annotations

## Etape 1 : Ajouter les annotations



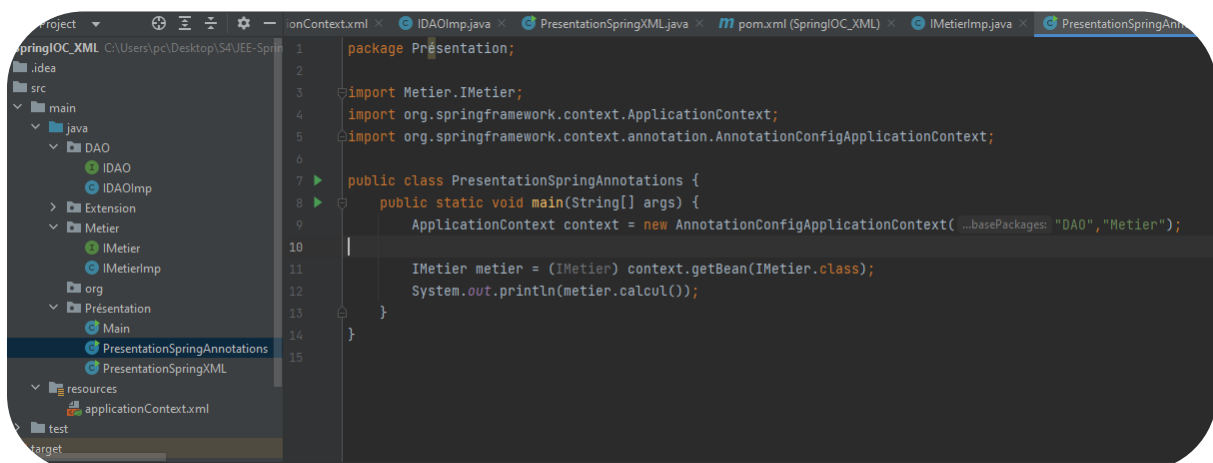
```
1 package DAO;
2
3 import org.springframework.stereotype.Component;
4
5 1 usage
6 @Component
7 public class IDAOImp implements IDAO {
8
9     1 usage
10     @Override
11     public double getData() { return 0; }
12 }
```

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

2 usages
@Component
public class IMetierImp implements IMetier{
    // Couplage faible
    2 usages
    @Autowired
    private IDAO dao;

    3 usages
    @Override
    public double calcul() {
        double d=dao.getData();
        double res=d+30;
        return res;}
}
```

## Etape 2 : main



The screenshot shows an IDE with a project structure on the left and a code editor on the right. The project structure includes a 'src' directory with 'main' and 'resources' subdirectories. The 'main' directory contains 'java' and 'resources' subdirectories. The 'java' directory contains 'DAO', 'Extension', 'Metier', and 'Presentation' subdirectories. The 'Presentation' directory contains 'Main', 'PresentationSpringAnnotations', and 'PresentationSpringXML' classes. The 'resources' directory contains 'applicationContext.xml'. The code editor shows the 'PresentationSpringAnnotations' class with the following code:

```
package Presentation;

import Metier.IMetier;
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class PresentationSpringAnnotations {
    public static void main(String[] args) {
        ApplicationContext context = new AnnotationConfigApplicationContext("DAO", "Metier");

        IMetier metier = (IMetier) context.getBean(IMetier.class);
        System.out.println(metier.calcul());
    }
}
```



## Conclusion

Grace à cette activité, j'ai eu l'occasion d'appliquer l'ensemble de concepts de bases de l'inversion de contrôle et l'injection de dépendance. En effet, j'ai pu comprendre le concept de l'inversion de contrôle et l'injection des dépendances qui se base sur l'utilisation de l'expérience des autres ; autrement dit bâtir notre application sur une architecture d'entreprise. L'inversion de contrôle alors prend en charge du flot d'exécution de notre logiciel, c'est-à-dire le flot d'exécution d'un logiciel n'est plus sous le contrôle direct de l'application elle-même mais du Framework. D'autre part, le développeur s'occupe uniquement du code métier (exigences fonctionnelles).

Ce compte rendu alors est le résumer de la première partie qui consiste sur la création des différentes classe et interface en utilisant le couplage faible et en utilisant l'injection des dépendances de différente manière ; instanciation statique, instanciation dynamique et en utilisant le Framework Spring (Version XML et version annotations).