

Compte rendu part 1 : Inversion de contrôle et Injection des dépendances

Master de recherche :
Systèmes Distribués et Intelligence Artificielle
(SDIA)

Réalisé par :

EZACCANI salma

Introduction

Dans le développement logiciel, la conception d'une architecture robuste et modulaire est cruciale pour assurer la flexibilité, la maintenance et l'évolutivité d'une application. Une approche courante pour atteindre ces objectifs consiste à utiliser des interfaces, des implémentations, et à mettre en œuvre l'injection de dépendances. Cette approche favorise le couplage faible entre les composants, facilitant ainsi la gestion des dépendances et la réutilisation du code.

Dans cette exploration, nous allons créer une architecture logicielle modulaire en suivant quelques étapes clés :

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

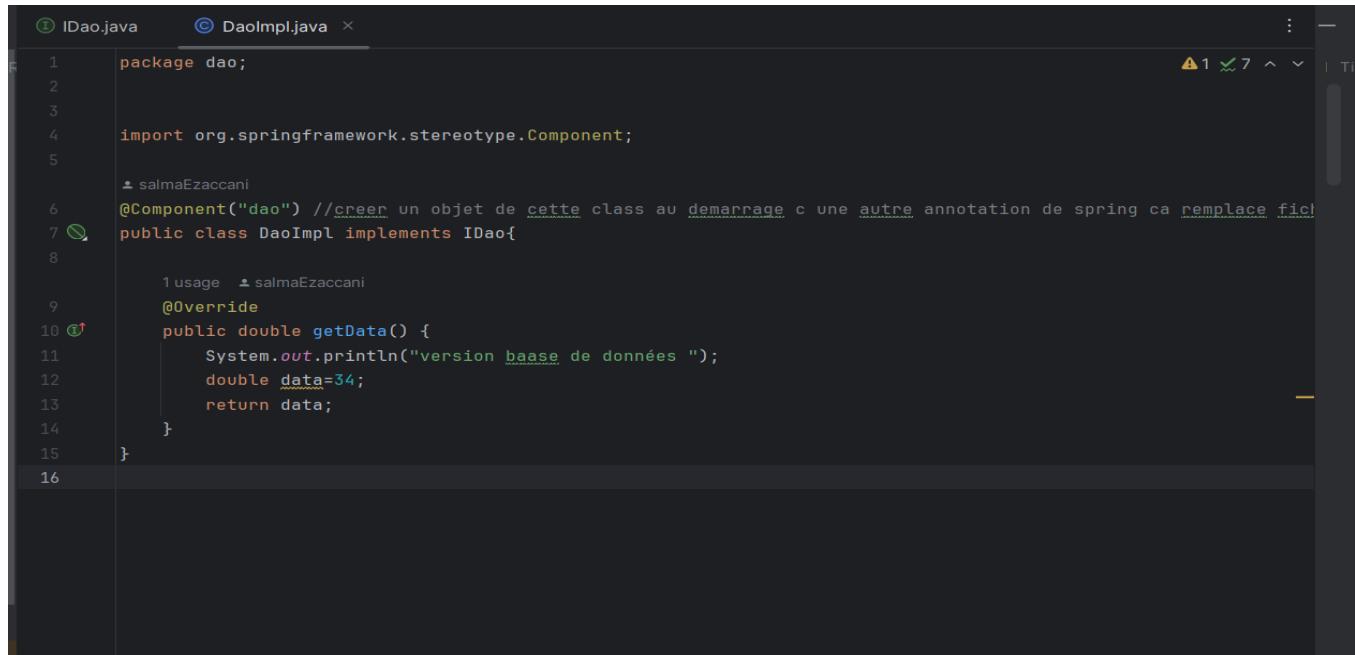
Réalisation

i. Créer l'interface IDao avec une méthode getDate

The screenshot shows the IntelliJ IDEA interface. On the left, the Project tool window displays the project structure under 'IOS-enset'. The 'src' directory contains 'main', 'java', 'ext', 'metier', 'presentation', 'resources', 'test', and 'target'. The 'java' directory contains 'dao' which has 'Daolmpl' and 'IDao'. The 'ext' directory contains 'DaolmplV2'. The 'metier' directory contains 'Calcul', 'IMetier', and 'MetierImpl'. The 'presentation' directory contains 'Pres1', 'Pres2', 'PresVecSpringAnnotation', and 'PresVecSpringXML'. The 'resources' directory contains 'config.xml'. The 'target' directory is highlighted. On the right, the code editor shows the 'IDao.java' file:

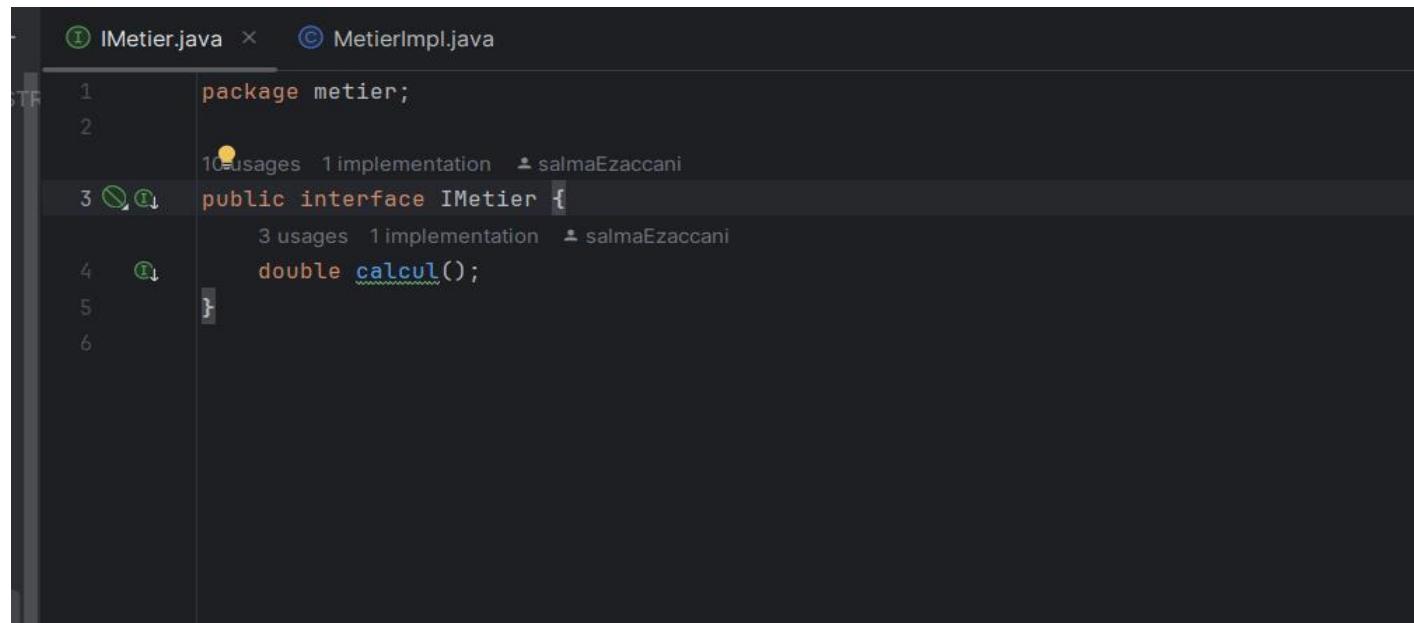
```
1 package dao;  
2  
3 public interface IDao {  
4     double getData();  
5 }
```

ii. Créer l'interface IDao avec une méthode getDate



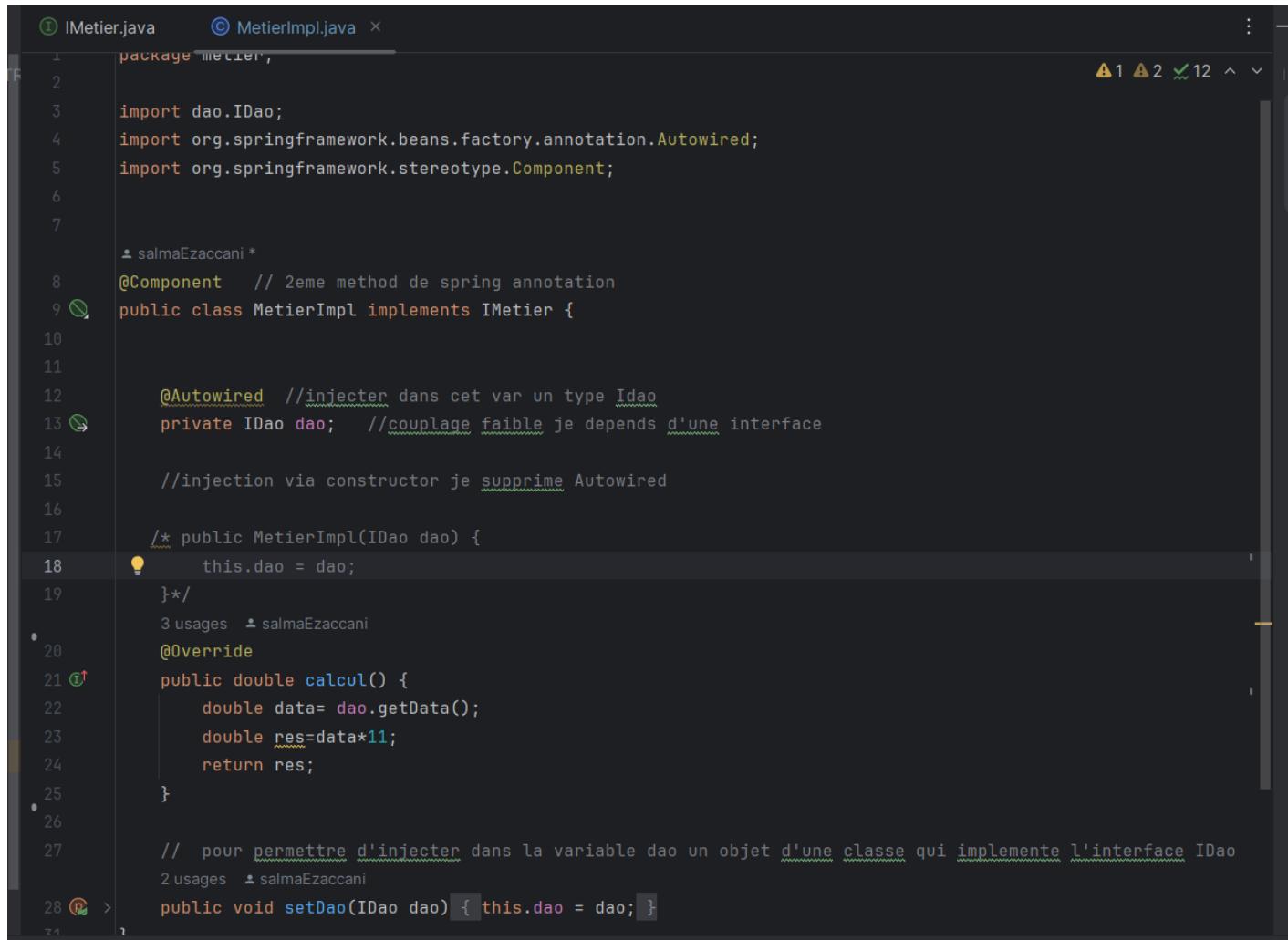
```
1 package dao;
2
3
4 import org.springframework.stereotype.Component;
5
6
7 public class DaoImpl implements IDao{
8
9     @Override
10    public double getData() {
11        System.out.println("version base de données ");
12        double data=34;
13        return data;
14    }
15}
```

iii. Créer l'interface IMetier avec une méthode calcul



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

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



```
IMetier.java  MetierImpl.java x

1 package metier;
2
3 import dao.IDao;
4 import org.springframework.beans.factory.annotation.Autowired;
5 import org.springframework.stereotype.Component;
6
7
8 ▲ salmaEzaccani *
9 @Component // 2eme method de spring annotation
10 public class MetierImpl implements IMetier {
11
12     @Autowired //injecter dans cet var un type IDao
13     private IDao dao; //couplage faible je depends d'une interface
14
15     //injection via constructor je supprime Autowired
16
17     /* public MetierImpl(IDao dao) {
18         this.dao = dao;
19     }*/
20     3 usages ▲ salmaEzaccani
21     @Override
22     public double calcul() {
23         double data= dao.getData();
24         double res=data*11;
25         return res;
26     }
27
28     // pour permettre d'injecter dans la variable dao un objet d'une classe qui implemente l'interface IDao
29     2 usages ▲ salmaEzaccani
30     public void setDao(IDao dao) {this.dao = dao; }
```

v. L'injection des dépendances : Par instantiation statique

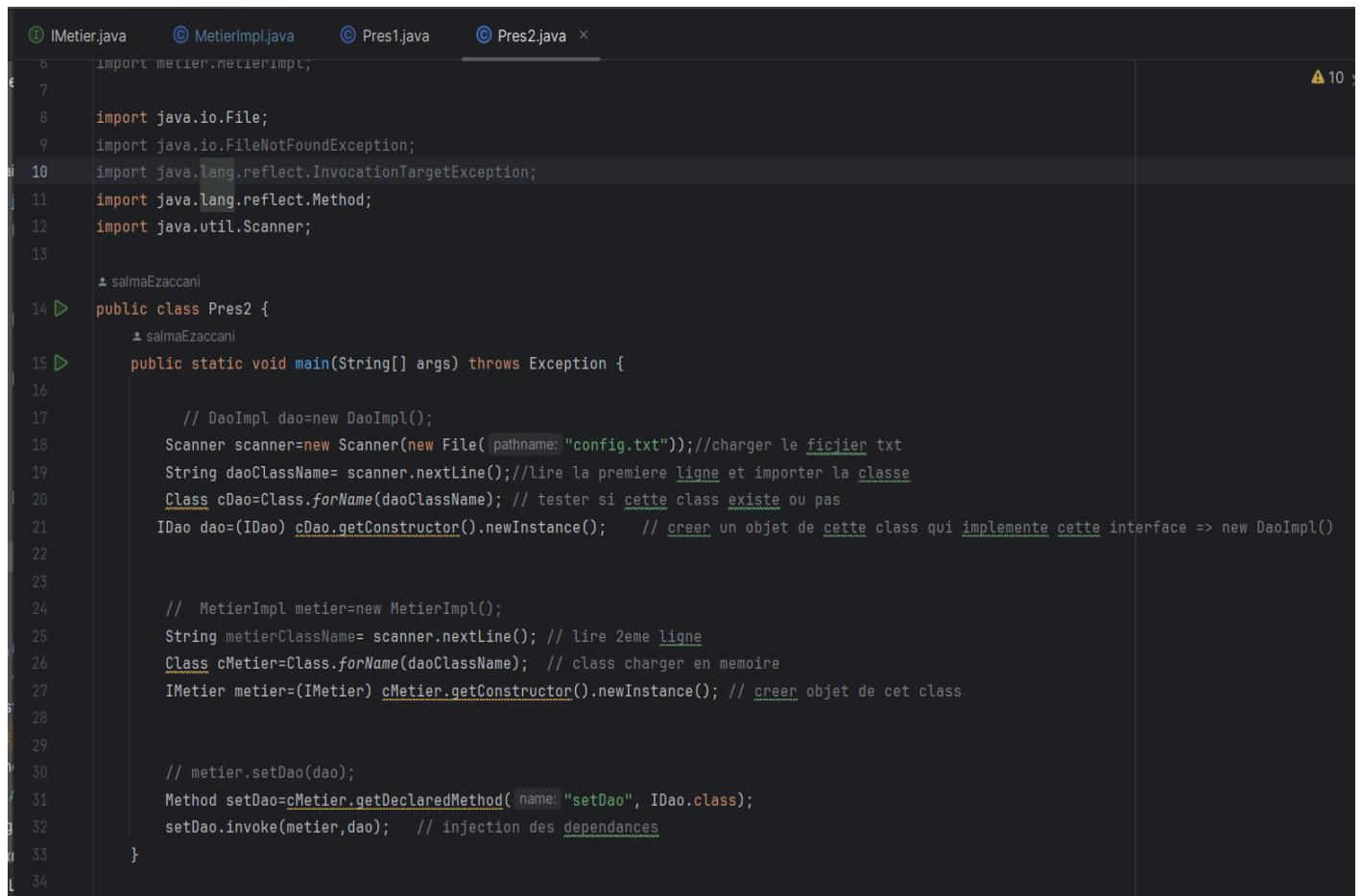
The screenshot shows an IDE interface with three tabs: IMetier.java, MetierImpl.java, and Pres1.java. The Pres1.java tab is active, displaying the following code:

```
1 package presentation;
2
3 import dao.DaoImpl;
4 import metier.MetierImpl;
5
6 //INJECTION DES DEPENDANCES: prendre adrs memoire d'un objet et le met dans une variable de l'autre objet
7 // salmaEzaccani
8 public class Pres1 {
9
10     // salmaEzaccani
11     public static void main(String[] args) {
12
13         //methode 1 "instanciation statique"
14
15         DaoImpl dao=new DaoImpl(); // lorsque fait new une adrs memoire va se generer et stocker dans var dao , "instanciation statique"
16         MetierImpl metier=new MetierImpl(); // inject via constructor on va mettre ici parameter dao et on va supprimer set
17         metier.setDao(dao); // injection des dependances
18         System.out.println("RESS="+metier.calcul());
19
20         //methode 2 "instanciation dynamique??
21         // IL FAUT CREER UN FICHIER DE CONFIGURATION"
22     }
23 }
```

The screenshot shows the IntelliJ IDEA Run tab for the Pres1 project. The output window displays the following text:

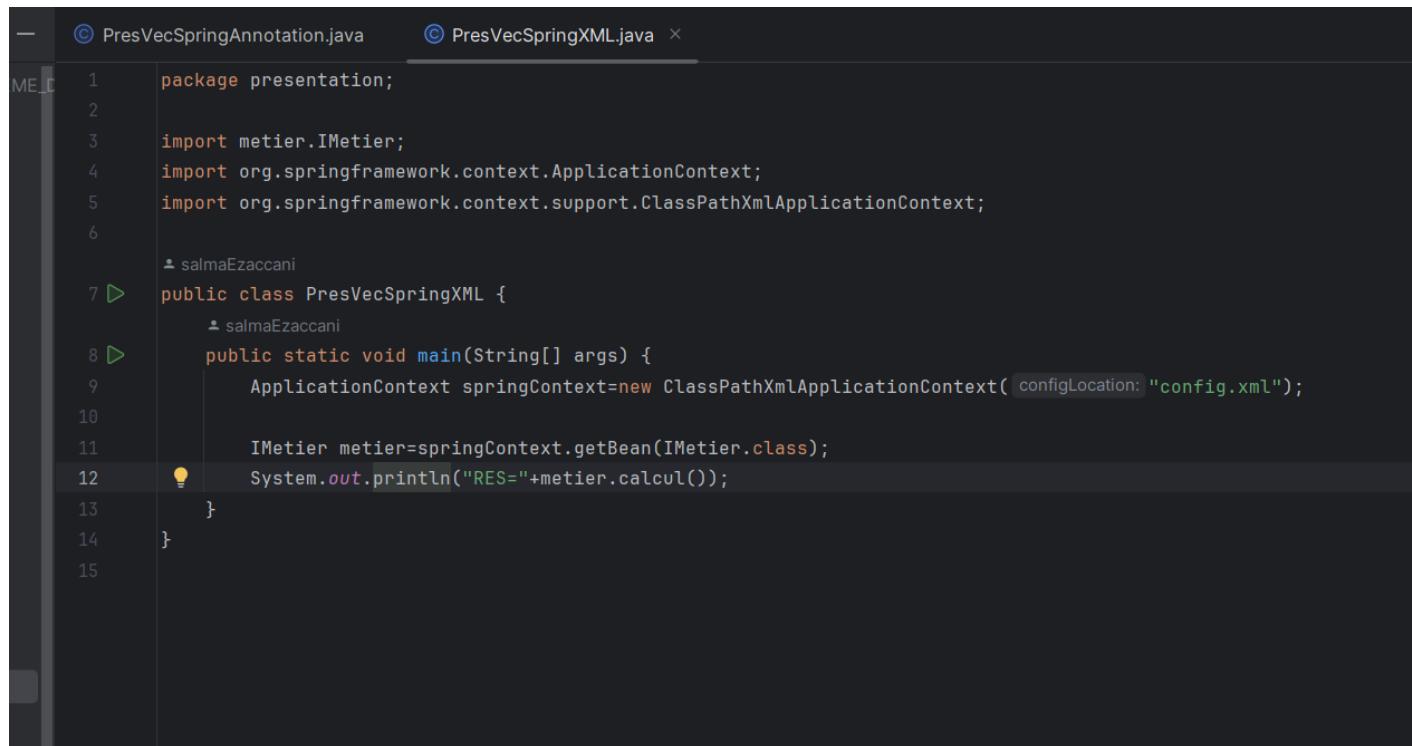
```
"C:\Program Files\Java\jdk-19\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2023.2.2\lib\idea_rt.jar=5000,127.0.0.1:6300" -Dfile.encoding=UTF-8 version base de données
RESS=374.0
Process finished with exit code 0
```

vi. l'injection des dépendances : Par instantiation dynamique



```
① IMetier.java      ② MetierImpl.java    ③ Pres1.java     ④ Pres2.java ×
 6 import metier.metierimpl;
 7
 8 import java.io.File;
 9 import java.io.FileNotFoundException;
10 import java.lang.reflect.InvocationTargetException;
11 import java.lang.reflect.Method;
12 import java.util.Scanner;
13
14 ▶ public class Pres2 {
15     ▲ salmaEzaccani
16     public static void main(String[] args) throws Exception {
17
18         // DaoImpl dao=new DaoImpl();
19         Scanner scanner=new Scanner(new File( pathname: "config.txt")); //charger le fichier txt
20         String daoClassName= scanner.nextLine(); //lire la premiere ligne et importer la classe
21         Class cDao=Class.forName(daoClassName); // tester si cette class existe ou pas
22         IDao dao=(IDao) cDao.getConstructor().newInstance(); // creer un objet de cette class qui implemente cette interface => new DaoImpl()
23
24         // MetierImpl metier=new MetierImpl();
25         String metierClassName= scanner.nextLine(); // lire 2eme ligne
26         Class cMetier=Class.forName(metierClassName); // class charger en memoire
27         IMetier metier=(IMetier) cMetier.getConstructor().newInstance(); // creer objet de cet class
28
29
30         // metier.setDao(dao);
31         Method setDao=cMetier.getDeclaredMethod( name: "setDao", IDao.class);
32         setDao.invoke(metier,dao); // injection des dependances
33     }
34 }
```

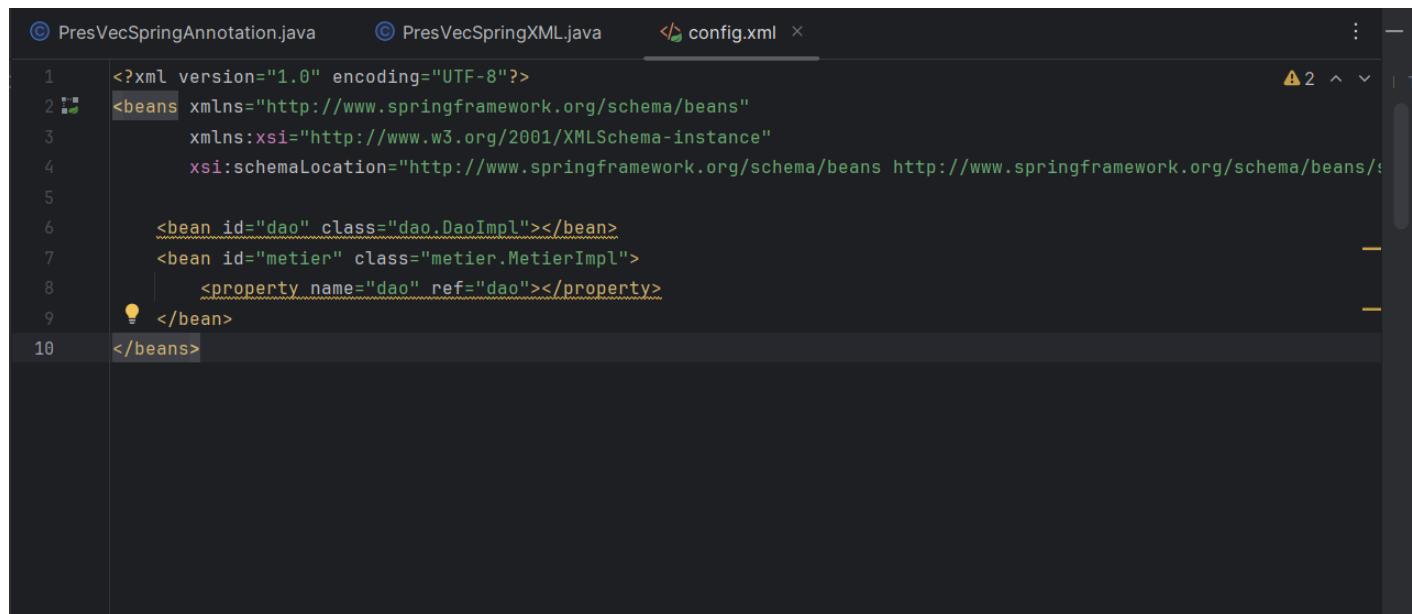
vii. En utilisant le Framework Spring : Version XML



```
package presentation;

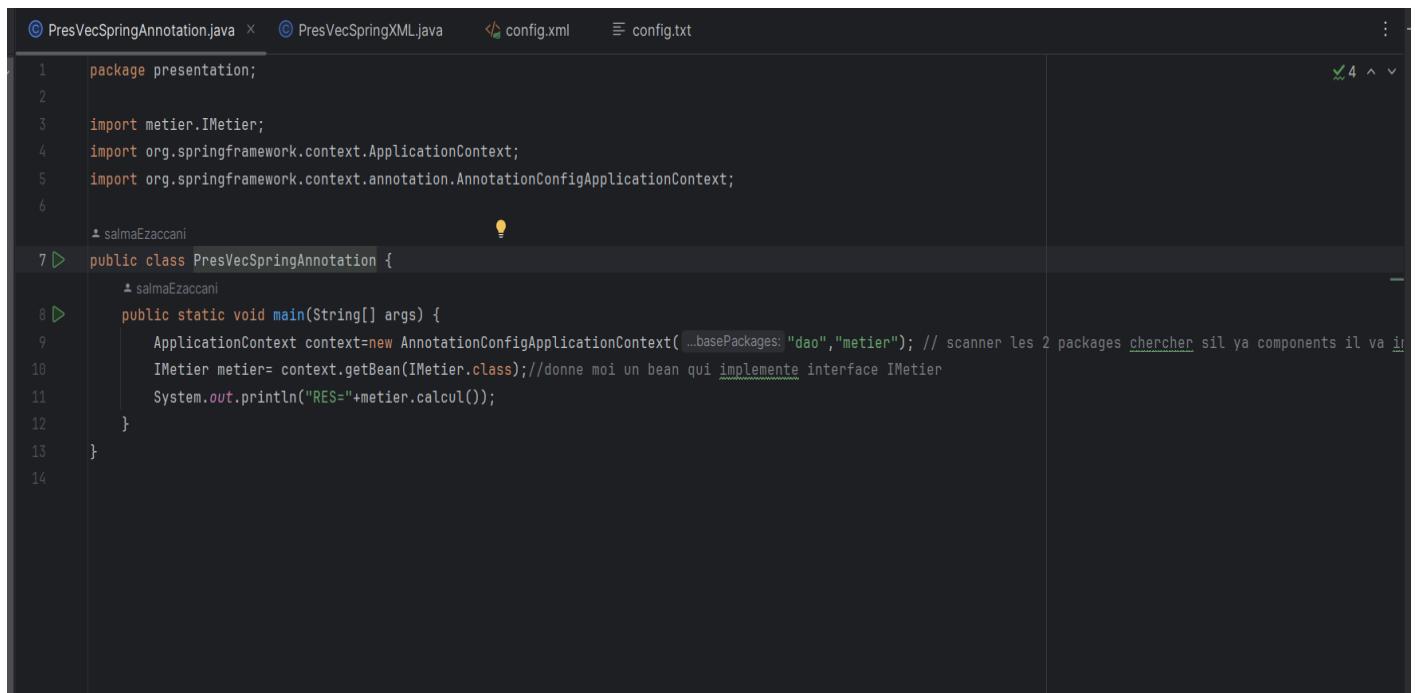
import metier.IMetier;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class PresVecSpringXML {
    public static void main(String[] args) {
        ApplicationContext springContext=new ClassPathXmlApplicationContext("config.xml");
        IMetier metier=springContext.getBean(IMetier.class);
        System.out.println("RES="+metier.calcul());
    }
}
```



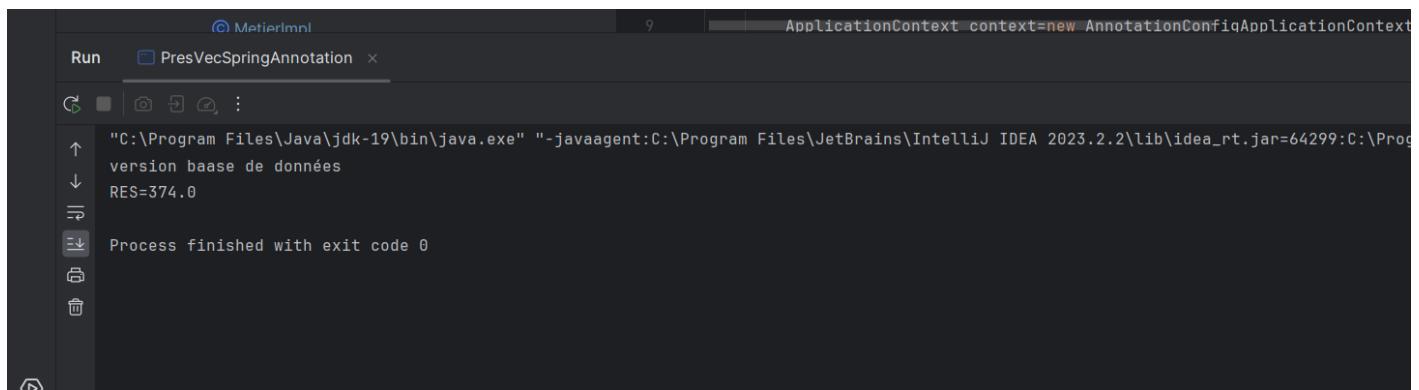
```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">
    <bean id="dao" class="dao.DaoImpl"/>
    <bean id="metier" class="metier.MetierImpl">
        <property name="dao" ref="dao"/>
    </bean>
</beans>
```

viii. En utilisant le Framework Spring : Version annotations



The screenshot shows an IDE interface with several tabs at the top: PresVecSpringAnnotation.java, PresVecSpringXML.java, config.xml, and config.txt. The main editor area displays the following Java code:

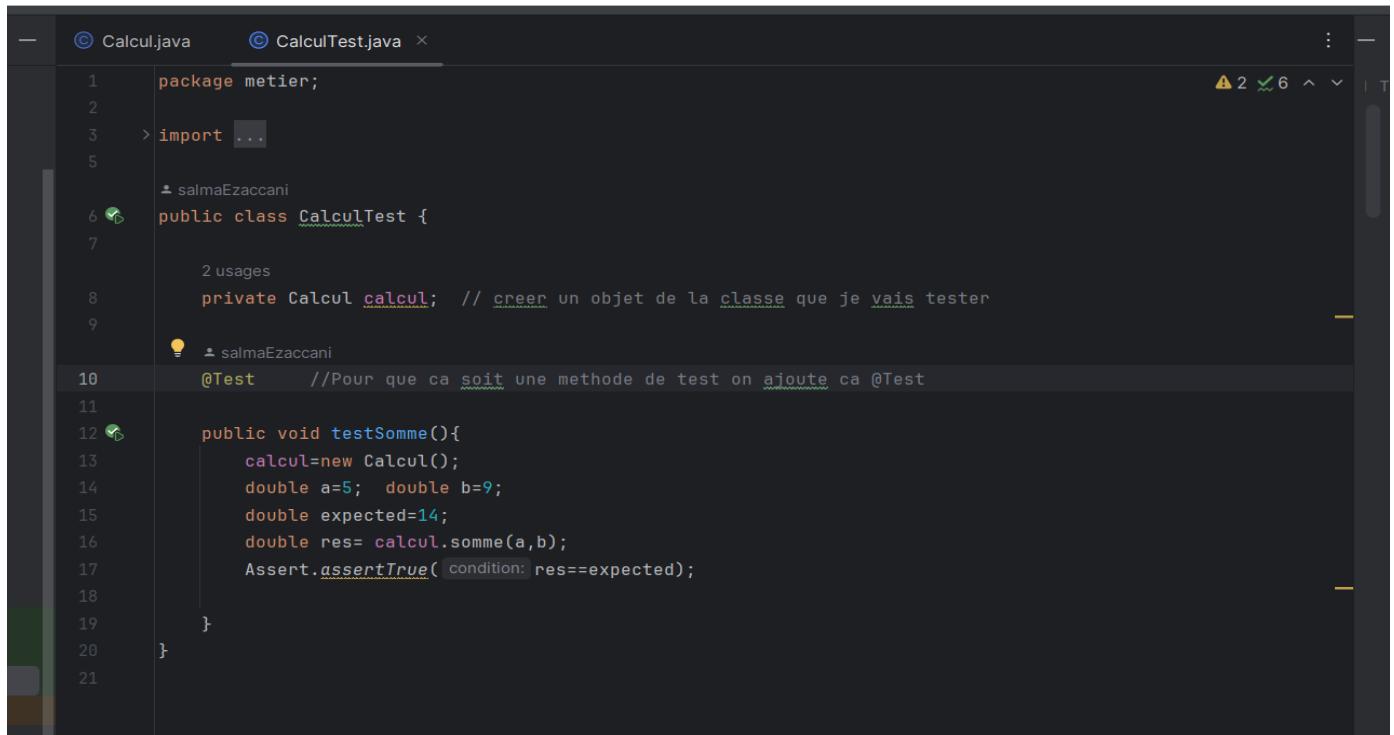
```
1 package presentation;
2
3 import metier.IMetier;
4 import org.springframework.context.ApplicationContext;
5 import org.springframework.context.annotation.AnnotationConfigApplicationContext;
6
7 public class PresVecSpringAnnotation {
8     public static void main(String[] args) {
9         ApplicationContext context=new AnnotationConfigApplicationContext(..basePackages: "dao","metier"); // scanner les 2 packages chercher sil ya components il va implementer l'interface
10        IMetier metier= context.getBean(IMetier.class); //donne moi un bean qui implemente interface IMetier
11        System.out.println("RES="+metier.calcul());
12    }
13 }
14
```



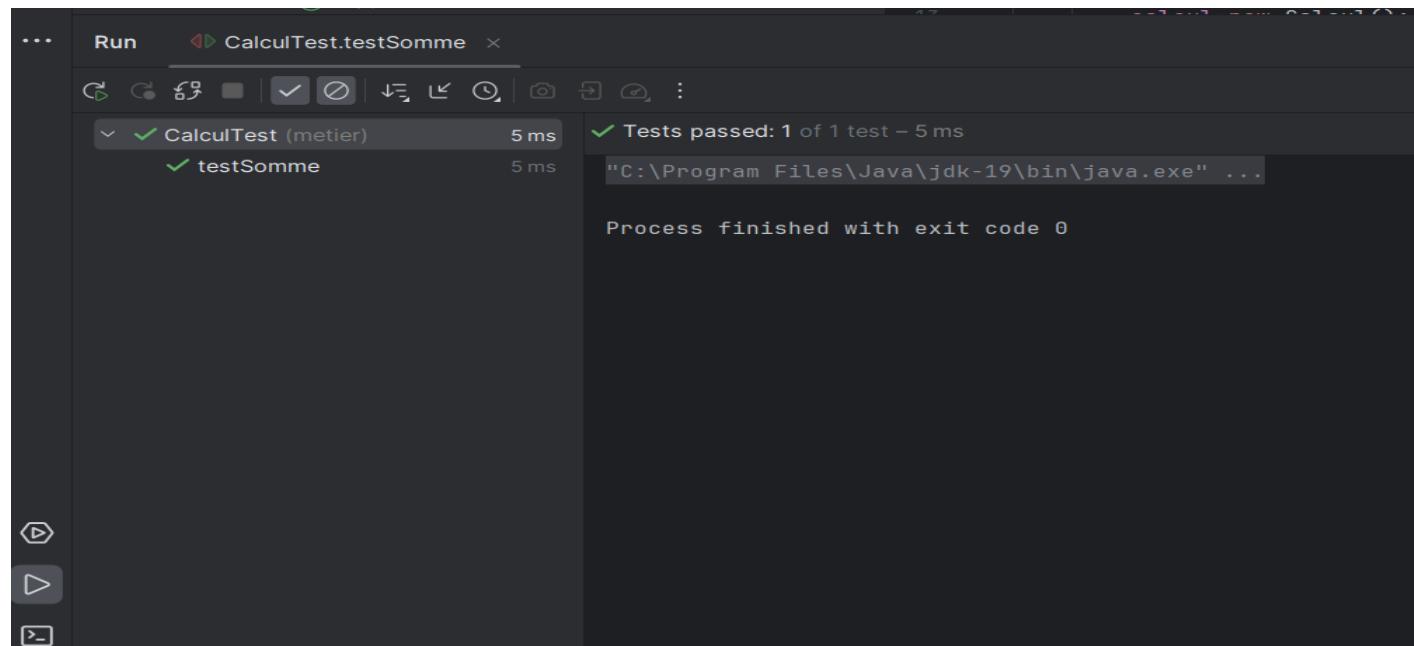
The screenshot shows the run output window of an IDE. It displays the command run and the standard output of the application. The output shows the application is running and printing the result of the calculation.

```
Run PresVecSpringAnnotation x
C:\Program Files\Java\jdk-19\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2023.2.2\lib\idea_rt.jar=64299:C:\Program Files\JetBrains\IntelliJ IDEA 2023.2.2\bin" -Dfile.encoding=UTF-8 version baase de données
RES=374.0
Process finished with exit code 0
```

Partie Test :



```
1 package metier;
2
3 > import ...
5
6 salmaEzaccani
7 public class CalculTest {
8
9     2 usages
10    private Calcul calcul; // creer un objet de la classe que je vais tester
11
12 salmaEzaccani
13 @Test //Pour que ca soit une methode de test on ajoute ca @Test
14
15     public void testSomme(){
16         calcul=new Calcul();
17         double a=5; double b=9;
18         double expected=14;
19         double res= calcul.somme(a,b);
20         Assert.assertTrue( condition: res==expected);
21     }
}
```



Run CalculTest.testSomme

Calculated Test	Time
CalculatedTest (metier)	5 ms
testSomme	5 ms

Tests passed: 1 of 1 test – 5 ms
C:\Program Files\Java\jdk-19\bin\java.exe ...
Process finished with exit code 0

Conclusion

En résumé, notre exploration a permis de créer une architecture logicielle modulaire en utilisant des interfaces, des implémentations, et l'injection de dépendances. Les interfaces **IDao** et **IMetier** définissent des contrats clairs, tandis que les implémentations concrètes illustrent la mise en œuvre. L'injection de dépendances, qu'elle soit statique, dynamique ou via Spring, offre différentes approches pour gérer les relations entre les composants. Ces concepts fondamentaux favorisent une structure logicielle flexible et extensible, éléments essentiels pour le développement d'applications robustes.