

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

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

Réalisé par :
EZACCANI salma

Introduction

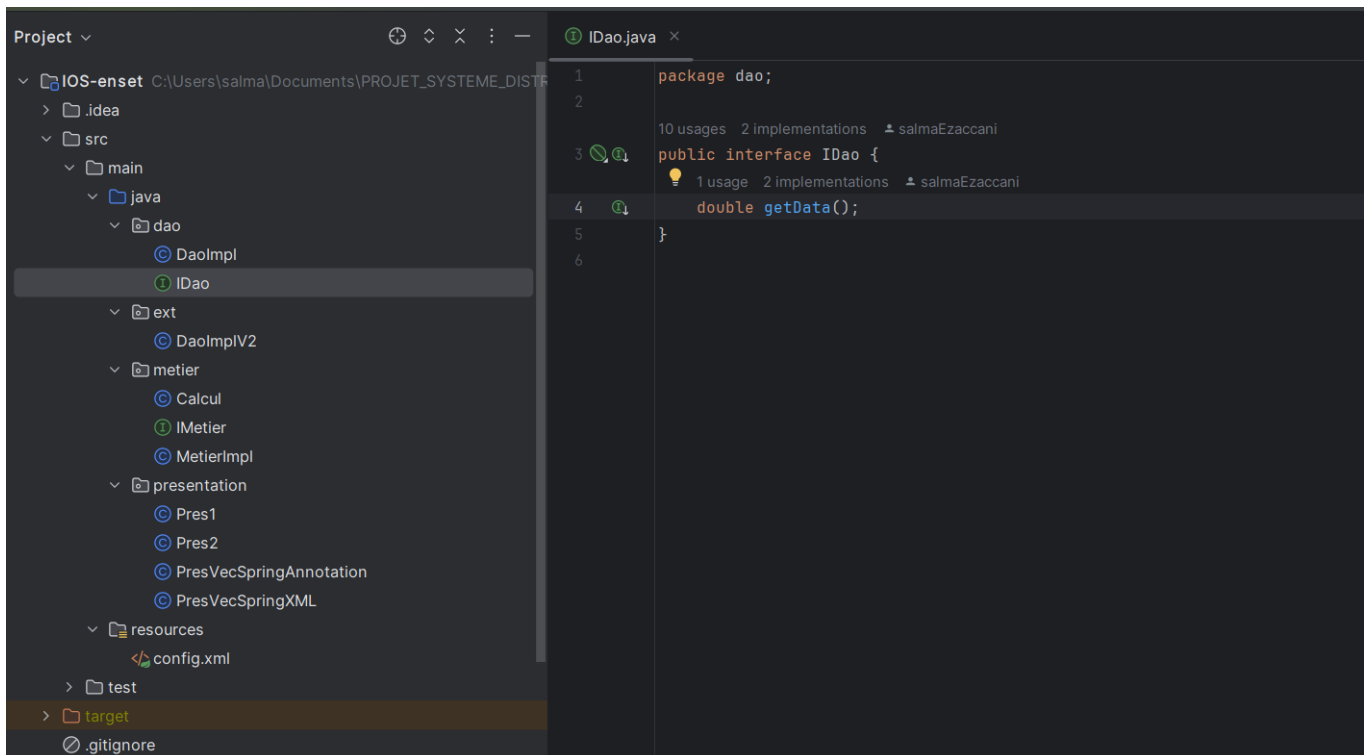
Dans le développement logiciel, la conception d'une architecture robuste et modulaire est cruciale pour assurer la flexibilité, la maintenance et l'évolutivité d'une application. Une approche courante pour atteindre ces objectifs consiste à utiliser des interfaces, des implémentations, et à mettre en 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



ii. Créer l'interface IDao avec une méthode getDate

```
IDao.java      DaoImpl.java x
1 package dao;
2
3
4 import org.springframework.stereotype.Component;
5
6 salmaEzaccani
7 @Component("dao") //créer un objet de cette class au démarrage c une autre annotation de spring ca remplace ficl
8 public class DaoImpl implements IDao{
9
10 1 usage salmaEzaccani
11  @Override
12  public double getData() {
13      System.out.println("version baase de données ");
14      double data=34;
15      return data;
16  }
17 }
```

iii. Créer l'interface IMetier avec une méthode calcul

```
IMetier.java x MetierImpl.java
1 package metier;
2
3 10 usages 1 implementation salmaEzaccani
4 public interface IMetier {
5     3 usages 1 implementation salmaEzaccani
6     double calcul();
7 }
```

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

```
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 @Component // 2eme method de spring annotation
9 public class MetierImpl implements IMetier {
10
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

```
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 a une variable de l'autre objet
7 public class Pres1 {
8
9     public static void main(String[] args) {
10
11         //methode 1 "instantiation statique"
12
13         DaoImpl dao=new DaoImpl(); // lorsque fait new une adrs memoire va se genere et stocker dans var dao ,"instantiation statique"
14         MetierImpl metier=new MetierImpl();//inject via constructor on va mettre ici parameter dao et on va supprimer set
15         metier.setDao(dao);//injection des dependances
16         System.out.println("RESS="+metier.calcul());
17
18         //methode 2 "instantiation dynamique??
19         // IL FAUT CREER UN FICHIER DE CONFIGURATION"
20     }
21 }
22
```

```
Run Pres1 x
"C:\Program Files\Java\jdk-19\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2023.2.2\lib\idea_rt
version baase de données
RESS=374.0
Process finished with exit code 0
```

vi. l'injection des dépendances : Par instantiation dynamique

```
1  IMetier.java  MetierImpl.java  Pres1.java  Pres2.java x
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     public static void main(String[] args) throws Exception {
16
17         // DaoImpl dao=new DaoImpl();
18         Scanner scanner=new Scanner(new File( pathname: "config.txt")); //changer le fichier txt
19         String daoClassName= scanner.nextLine(); //lire la premiere ligne et importer la classe
20         Class cDao=Class.forName(daoClassName); // tester si cette class existe ou pas
21         IDao dao=(IDao) cDao.getConstructor().newInstance(); // creer un objet de cette class qui implemente cette interface => new DaoImpl()
22
23
24         // MetierImpl metier=new MetierImpl();
25         String metierClassName= scanner.nextLine(); // lire 2eme ligne
26         Class cMetier=Class.forName(daoClassName); // 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

```
PresVecSpringAnnotation.java  PresVecSpringXML.java x
1 package presentation;
2
3 import metier.IMetier;
4 import org.springframework.context.ApplicationContext;
5 import org.springframework.context.support.ClassPathXmlApplicationContext;
6
7 public class PresVecSpringXML {
8     public static void main(String[] args) {
9         ApplicationContext springContext=new ClassPathXmlApplicationContext("config.xml");
10
11         IMetier metier=springContext.getBean(IMetier.class);
12         System.out.println("RES="+metier.calcul());
13     }
14 }
15
```

```
PresVecSpringAnnotation.java  PresVecSpringXML.java  config.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 http://www.springframework.org/schema/beans/
5
6       <bean id="dao" class="dao.DaoImpl"></bean>
7       <bean id="metier" class="metier.MetierImpl">
8           <property name="dao" ref="dao"></property>
9       </bean>
10 </beans>
```

viii. En utilisant le Framework Spring : Version annotations

```
PresVecSpringAnnotation.java × PresVecSpringXML.java config.xml config.txt
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("dao","metier"); // scanner les 2 packages chercher sil ya composants il va in
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
```

```
Run PresVecSpringAnnotation ×
"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:\Pro
version baase de données
RES=374.0
Process finished with exit code 0
```

Partie Test :

```
Calcul.java  CalculTest.java x
1  package metier;
2
3  > import ...
4
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     }
22 }
```

```
Run  CalculTest.testSomme x
[Icons]
✓ CalculTest (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.