Nom : EL garti Fatima

Atelier 1 : Injection de dépendances

I. Introduction :

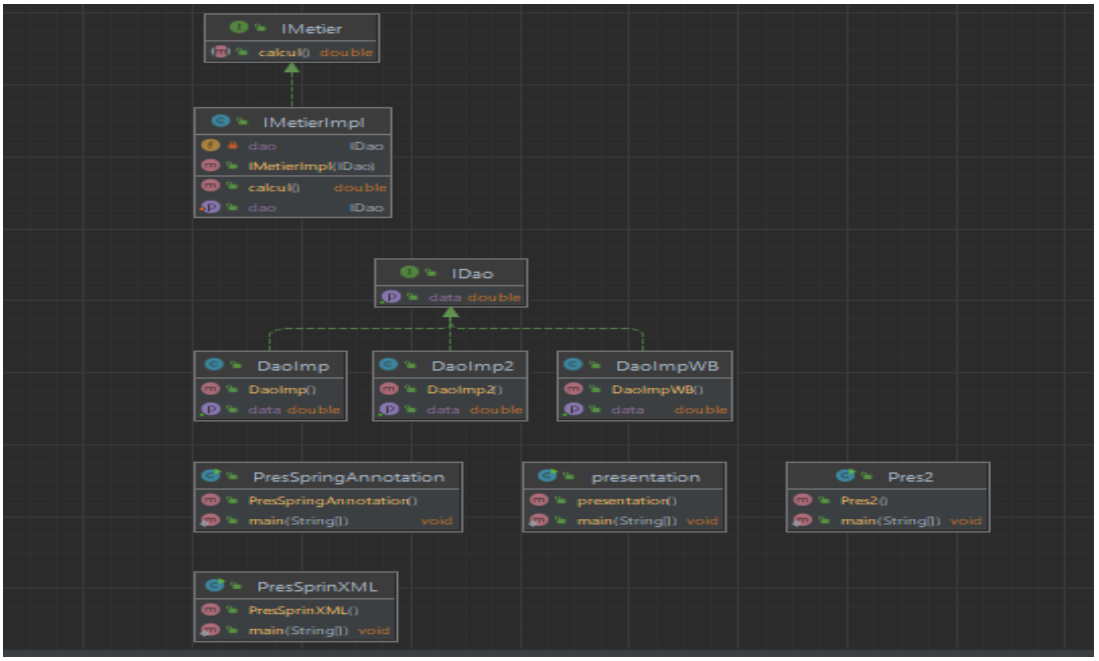
Cet atelier a visé l'injection de dépendances (couplage faible),d'abord sans framework , et ensuite en utilisant le framework Spring . Les notions abordées sont :

- Couplage fort et faible
- Inversion de contrôle
- injection des dépendance
- Code métier et code technique
- programmation orientée aspect et objet

II. Enoncé :

1. Créer l'interface IDao.
2. Créer une implémentation de cette interface.
3. Créer l'interface Métier.
4. Créer une implémentation de cette interface.
5. Créer la couche Présentation en faisant l'injection des dépendances
    a. Par Instanciation statique
    b. Par Instanciation dynamique
    c. En utilisant Spring : - XML - Annotations

III. Conception et architecture :



IV. Code Source :

- **Github Repository :** https://github.com/felgarti/InjectionDependances.git

V. Captures d'écran :

○ Interface Dao :

```
package Dao;

public interface IDao {
    double getData();
}
```

○ Version database : DaoImp :

```
package Dao;

import org.springframework.stereotype.Component;

@Component("dao")
public class DaoImp implements IDao {
    @Override
    public double getData() {
        System.out.println("version database");
        double temp=Math.random()*40 ;
        return temp;
    }
}
```

```
"C:\Program Files\Java\jdk-17.0.1\bin\
version database
Résultat => 289.9507100477041

Process finished with exit code 0
```

- **Version capteurs : DaoImp2**

```java
@Component("dao2")
public class DaoImp2 implements IDao{

    @Override
    public double getData() {
        System.out.println("Version capteurs ");
        return 6000;
    }
}
```

```
"C:\Program Files\Java\jdk-17.0.1\bin\java
Version capteurs
Résultat => 6273.0

Process finished with exit code 0
```

- **Version Web : DaoWB**

```java
@Component("dao3")
public class DaoImpWB implements IDao{
    @Override
    public double getData() {
        System.out.println("Version Web service ");
        return 7000;
    }
}
```

```
"C:\Program Files\Java\jdk-17.0.1\bin
Version Web service
Résultat => 7273.0

Process finished with exit code 0
```

- **Interface metier :**

```java
package Metier;

public interface IMetier {
    double calcul();
}
```

- **IMetierImpl :**

```java
@Component
public class IMetierImpl implements IMetier {
    @Autowired
    @Qualifier(value = "dao2")
    private IDao  dao ;
    public IMetierImpl(IDao dao)
    {
       this.dao=dao ;
    }
    @Override
    public double calcul() {
        double temp=dao.getData() ;
        return temp+273 ;
        //55:24
    }

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

- **Présentation :**
  - **Présentation 1 couplage fort :**

```java
package pres;

import ...

public class presentation {
    public static void main(String[] args) {
        DaoImp dao = new DaoImp() ;
        IMetierImpl metier=new IMetierImpl(dao) ;
        //  metier.setDao(dao);
        System.out.println(metier.calcul());
    }
}
```

  - **Présentation 2 couplage faible :**

```java
public class Pres2 {
    public static void main(String[] args) throws Exception {
        Scanner sc = new Scanner(new File( pathname: "src/config.txt")) ;

        String DaoClassName=sc.nextLine();
        Class cDao=Class.forName(DaoClassName) ;
       IDao dao= (IDao) cDao.getConstructor().newInstance() ;
       //System.out.println(dao.getData());
       //classCastException si la classe de l'objet dao n implemente pas IDao

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

        Method method = cMetier.getMethod( name: "setDao" , IDao.class) ;
        method.invoke(metier, dao) ;

        System.out.println("Résultat => "+metier.calcul());
```

- **Présentation Annotation Spring :**

```java
public class PresSpringAnnotation {

    public static void main(String[] args) {
        ApplicationContext context = new AnnotationConfigApplicationContext( ...basePackages: "Dao","Metier" ) ;
        IMetier metier=context.getBean(IMetier.class) ;
        System.out.println("Result => " + metier.calcul());
    }
}
```

- **Presentation XML :**

```java
public class PresSprinXML {
    public static void main(String[] args) {
        ApplicationContext context = new ClassPathXmlApplicationContext( configLocation: "applicationContext.xml") ;
        IMetier metier = (IMetier) context.getBean( s: "metier") ;
        System.out.println("Result => " + metier.calcul());

    }
}
```

- **ApplicationContext.xml :**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instanc
       xsi:schemaLocation="http://www.springframework.org/
    <bean id="dao" class="Dao.DaoImp"></bean>
    <bean id="metier" class="Metier.IMetierImpl">

        <constructor-arg ref="dao" ></constructor-arg>

    </bean>

</beans>
```

VI. Conclusion :

L'injection des dépendances peut se faire selon différentes manières gérées par le framework spring. Il permet aussi la séparation entre le code métier et le code technique (inversion de contrôle).