ECOLE MAROCAINE DES SCIENCES DE L'INGENIEUR
Membre de HONORIS UNITED UNIVERSITIES

Compte Rendu

Fait par KNAR Imane

Introduction :

Pour créer une application facile à maintenir, elle doit être fermée à la modification, ouverte à l'extension.

Pour créer un projet qui respecte ces critères, on doit utiliser le principe de l'inversion de contrôle.

L'inversion de contrôle veut dire que le développeur va s'occuper des exigences fonctionnels *(code métier),* et le Framework va s'occuper des exigences techniques *(code technique).*

L'inversion de contrôle est rendu possible grâce à la programmation orienté aspect, qui permet de séparer les différents aspects d'une application : code métier dans une classe, code de la sécurité dans une autre classe, l'accès aux données dans une autre classe…càd on ne mélange jamais le code métier et le code technique.

Une application doit être bâtit sur une architecture d'entreprise, parmi les architectures qu'on peut utiliser on a l'architecture JEE .

Le Framework qu'on va utiliser doit permettre de faire l'inversion de contrôle, parmi ces Framework on va travailler avec Spring.

Atelier 1 : Couplage faible

Dans le couplage faible, on doit avoir une classe qui dépend d'une interface, et non pas une classe qui dépend d'une autre classe.

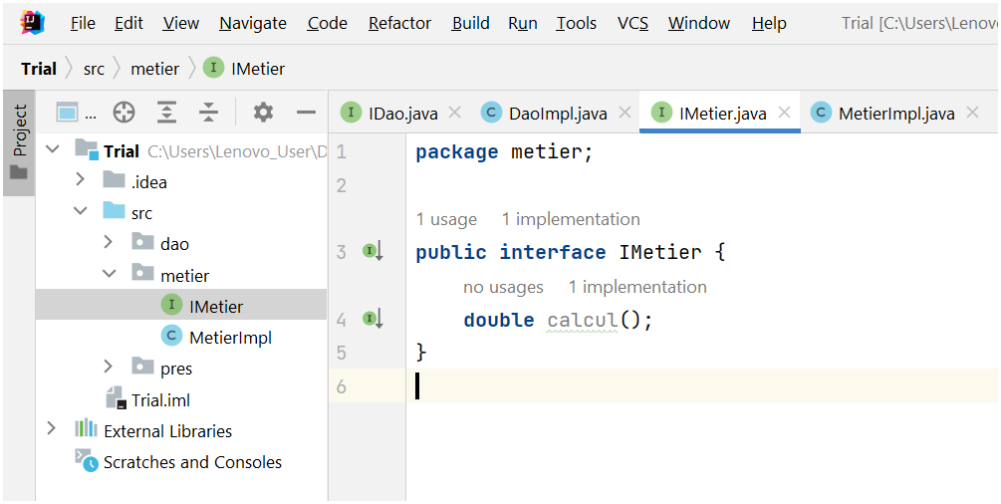
On va commencer par la couche DAO *(Couche d'accès aux données).*

Dans un premier temps, on va créer une interface IDao :

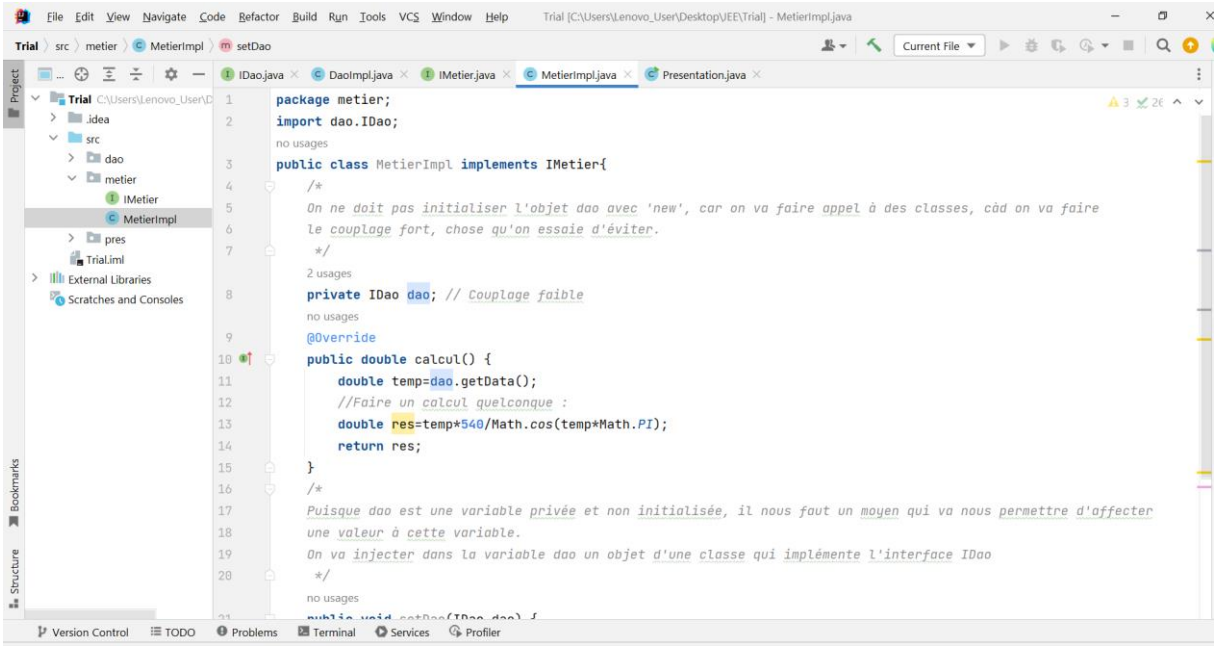
On va créer une implémentation de l'interface IDao :

On va passer à la couche métier , qui va s'occuper des besoins fonctionnels.

On va créer une interface IMetier :



On va créer une implémentation de l'interface IMetier :




```
    /*
    Puisque dao est une variable privée et non initialisée, il nous faut un moyen qui va nous permettre d'affecter
    une valeur à cette variable.
    On va injecter dans la variable dao un objet d'une classe qui implémente l'interface IDao
     */
    no usages
    public void setDao(IDao dao) {
        this.dao = dao;
    }
}
```

Atelier 2 : Injection des dépendances en utilisant les annotations

On doit d'abord ajouter les dépendances de Spring core, Spring context et Spring beans dans le fichier pom.xml :

```xml
</properties>
<dependencies>
<!-- https://mvnrepository.com/artifact/org.springframework/spring-core -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>5.3.23</version>
</dependency>
    <!-- https://mvnrepository.com/artifact/org.springframework/spring-context -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>5.3.23</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.springframework/spring-beans -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-beans</artifactId>
        <version>5.3.23</version>
    </dependency>

</dependencies>
```

Après, on doit ajouter l'annotation @Component sur chaque classe de notre projet

Et on ajoute l'annotation @Autowired sur l'objet, qui va permettre d'activer l'injection des dépendances d'une manière automatique

```java
package dao;

import org.springframework.stereotype.Component;

no usages
@Component
public class DaoImpl implements IDao{
    1 usage
    @Override
    public double getData() {
        /*
        Se connecter à la BD pour récupérer la température
         */
        double temp=Math.random()*40;
        return temp;
    }
}
```

```java
package metier;
import dao.IDao;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

3 usages
@Component
public class MetierImpl implements IMetier{
    /*
    On ne doit pas initialiser l'objet dao avec 'new', car on va faire appel à des classes, càd on va faire
    le couplage fort, chose qu'on essaie d'éviter.
     */
    2 usages
    @Autowired
    private IDao dao; // Couplage faible
    1 usage
    @Override
    public double calcul() {
        double temp=dao.getData();
        //Faire un calcul quelconque :
        double res=temp*540/Math.cos(temp*Math.PI);
        return res;
    }
```

```java
package pres;

import metier.IMetier;
import metier.MetierImpl;
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import org.springframework.stereotype.Component;

no usages
@Component
public class Presentation {
    no usages
    public static void main(String[] args) {
        ApplicationContext context = new AnnotationConfigApplicationContext( ...basePackages: "dao","metier");
        IMetier metier = context.getBean(IMetier.class);
        System.out.println("Température : "+ metier.calcul());
    }
}
```