

Département Mathématique informatique

Filière :

“ Ingénierie Informatique - Big Data & Cloud Computing ”

II-BDCC

Compte rendu de l'activité pratique N 1 JEE

Inversion de contrôle et Injection des dépendances

Réalisé par : Tarik FERTAH

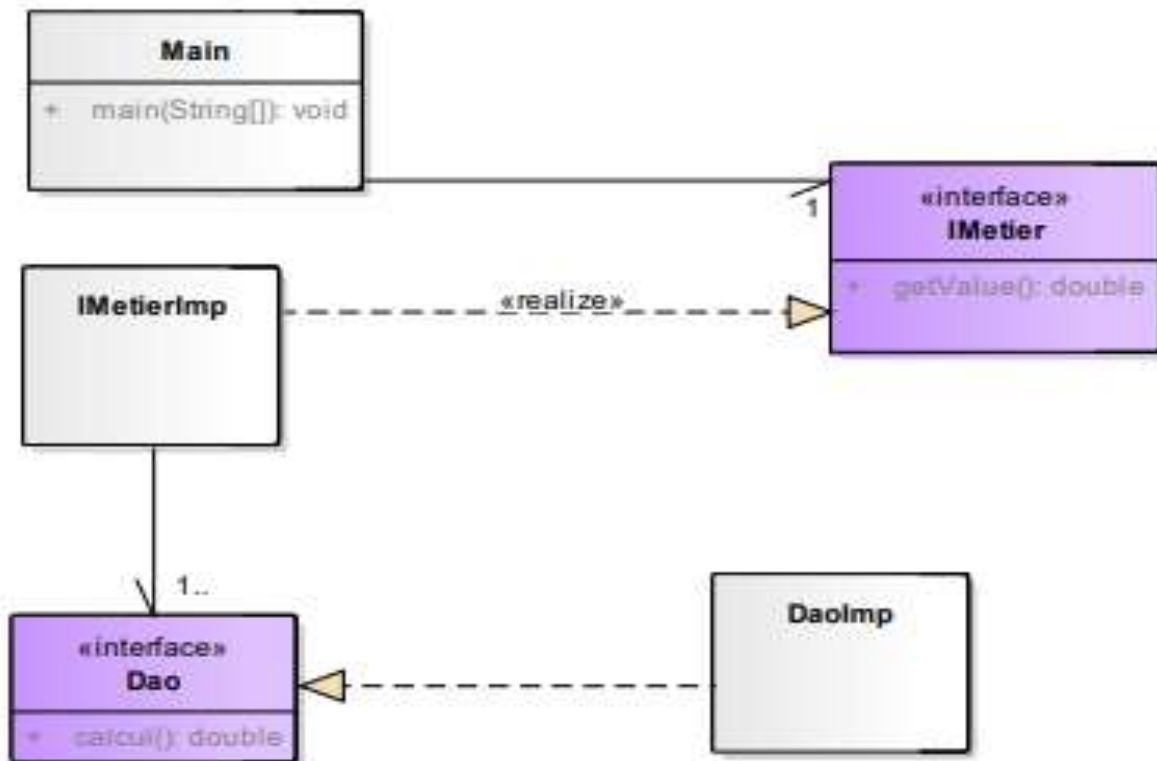
Année Universitaire : 2021 – 2022

Introduction

Inversion de contrôle : est un principe de conception qui préconise l'externalisation des activités de flux de contrôle telles que la découverte, l'instanciation et la destruction des unités dans un cadre indépendant des consommateurs et des fournisseurs. Le principe sous-jacent de l'loc est de découpler les consommateurs et les fournisseurs, ce qui évite aux unités logicielles de s'inquiéter de la découverte, de l'instanciation et du nettoyage de leurs dépendances, et permet aux unités de se concentrer sur leurs propres fonctionnalités. Ce découplage permet de conserver le logiciel extensible et maintenable.

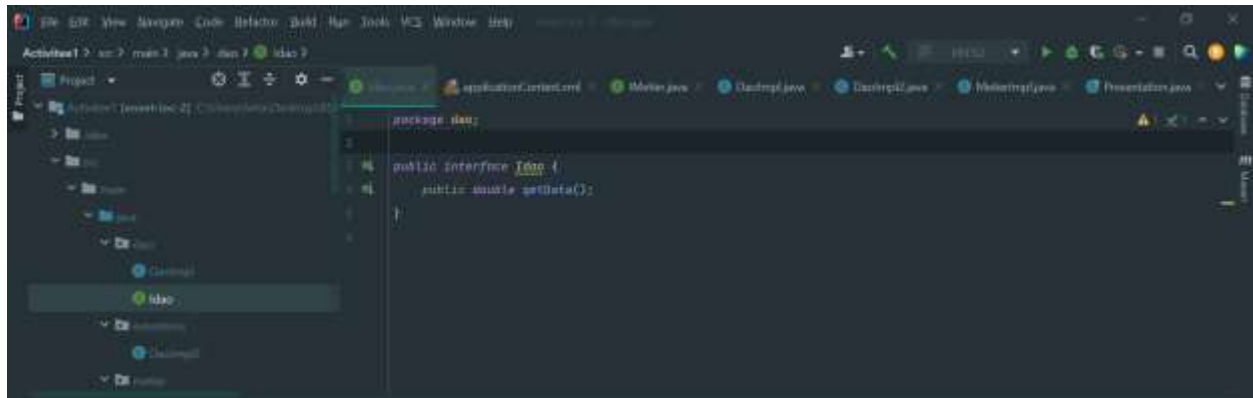
L'injection de dépendance : est l'une des techniques permettant d'implémenter le principe d'inversion de contrôle selon lequel des instances de dépendances (fournisseurs) sont injectées dans une unité logicielle (le consommateur) au lieu que le consommateur les trouve et les instancie.

Conception

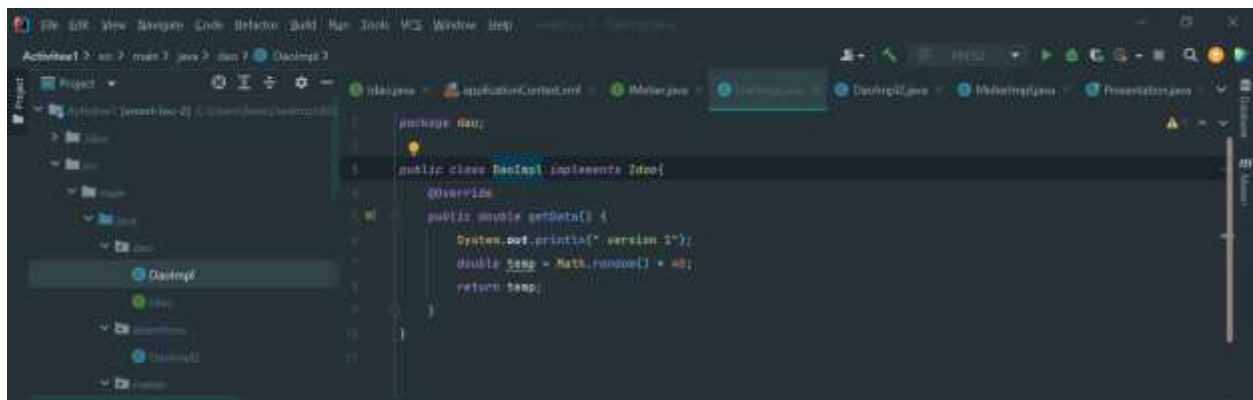


Code Source

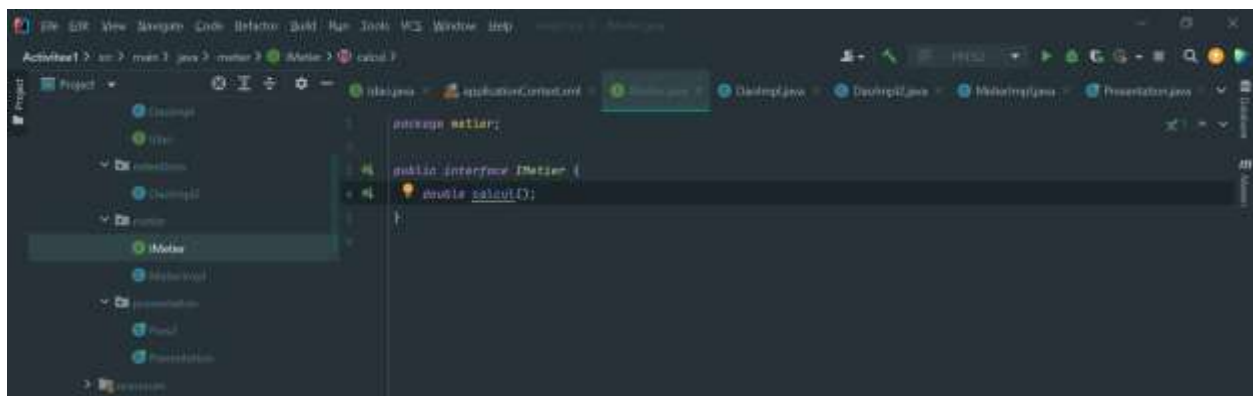
1. Création de l'interface IDao :



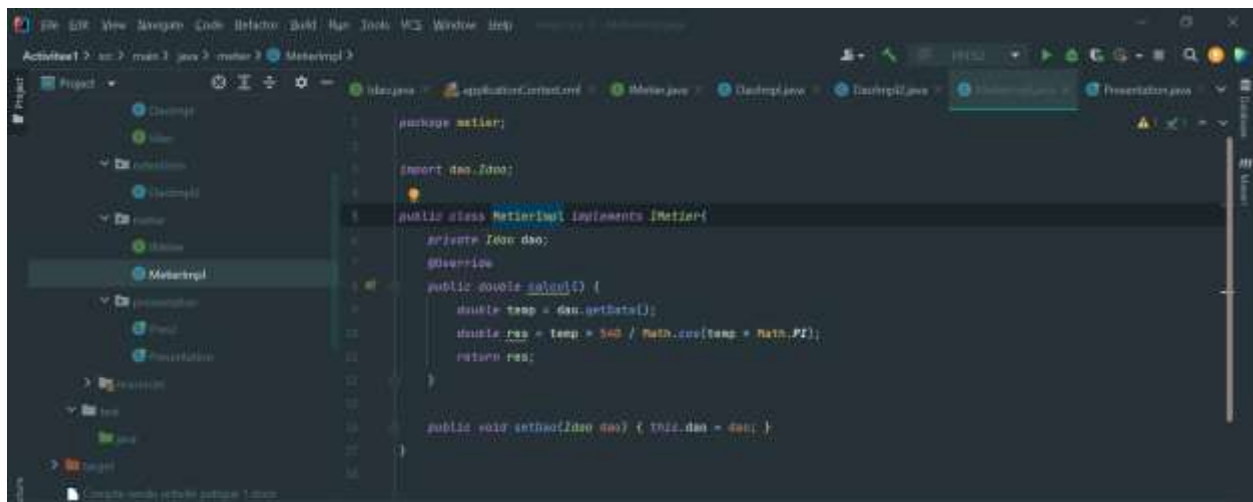
2. Création d'une implémentation de cette interface



3. Création de l'interface IMetier



4. Création d'une implémentation de cette interface en utilisant le couplage faible



```
package metier;

import dao.Dao;

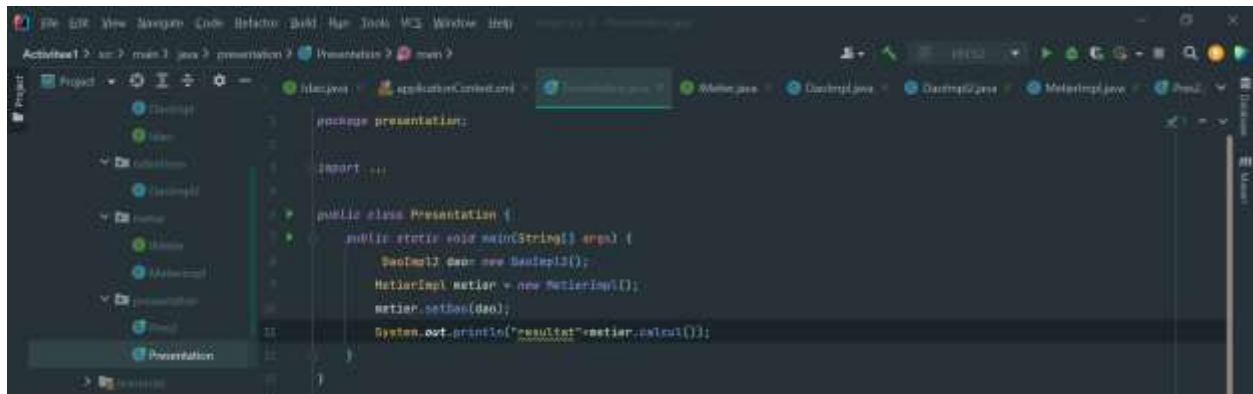
public class MetierImpl implements Metier {
    private Dao dao;

    @Override
    public double calcul() {
        double temp = dao.getStats();
        double res = temp * 540 / Math.cos(temp * Math.PI);
        return res;
    }

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

5. L'injection des dépendances

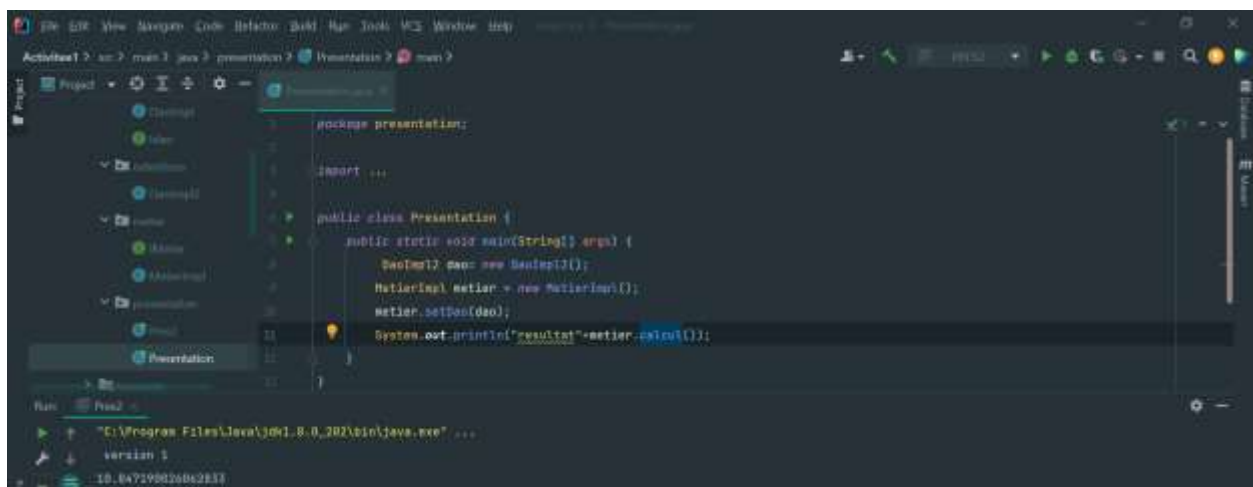
A - Par instanciation statique



```
package presentation;

import ...

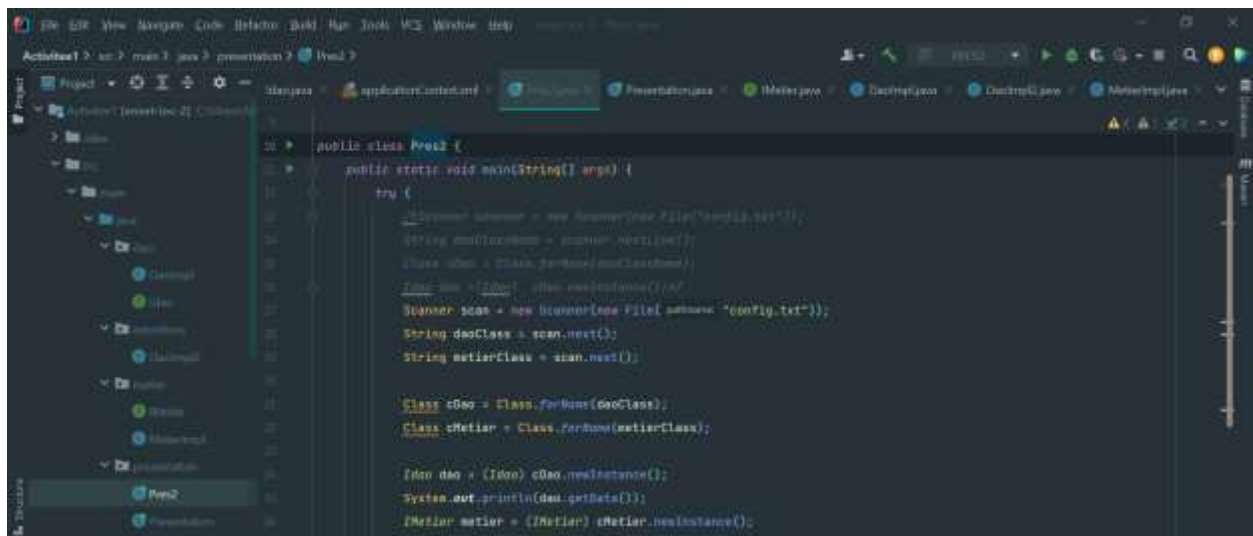
public class Presentation {
    public static void main(String[] args) {
        DaoImpl2 dao = new DaoImpl2();
        MetierImpl metier = new MetierImpl();
        metier.setDao(dao);
        System.out.println("resultat="+metier.calcul());
    }
}
```



```
Run: Press F10

"C:\Program Files\Java\jdk1.8.0_202\bin\java.exe" ...
version 1
10.847199826862813
```

B - Par instantiation dynamique

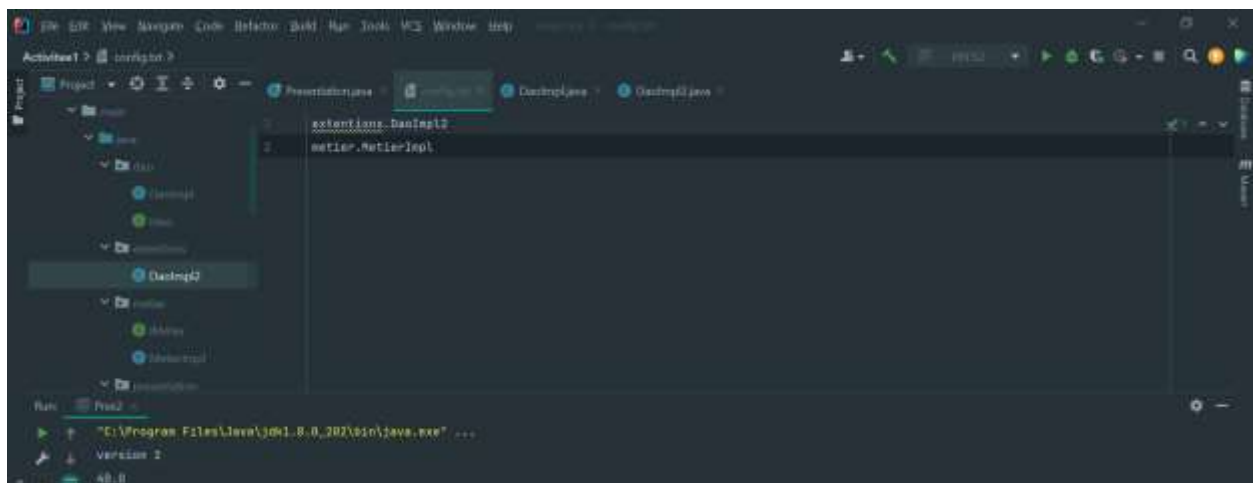


```
public class Pres2 {  
    public static void main(String[] args) {  
        try {  
            Scanner scanner = new Scanner(new File("conf12.txt"));  
            String daoClassName = scanner.nextLine();  
            Class dao = Class.forName(daoClassName);  
            //dao dao = (Dao) dao.newInstance();  
            Scanner scan = new Scanner(new File("conf12.txt"));  
            String daoClass = scan.next();  
            String metierClass = scan.next();  
  
            Class cDao = Class.forName(daoClass);  
            Class cMetier = Class.forName(metierClass);  
  
            Dao dao = (Dao) cDao.newInstance();  
            System.out.println(dao.getInfos());  
            Metier metier = (Metier) cMetier.newInstance();  
        }  
    }  
}
```



```
Method method1 = cMetier.getMethod("setBar", new Class[] { dao.getClass() });  
method1.invoke(metier, dao);  
System.out.println(metier.valin());  
  
} catch (Exception e) {  
    e.printStackTrace();  
}  
}
```

Après le changement dans le fichier config.txt

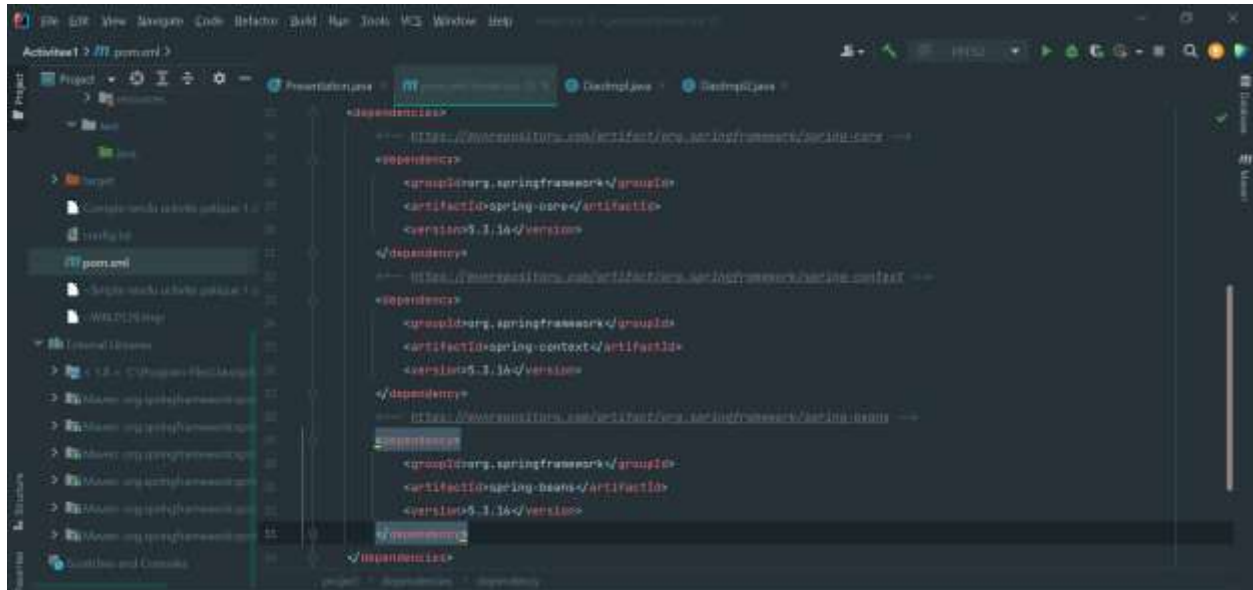


```
extensions.DaoImpl2  
metier.MetierImpl
```

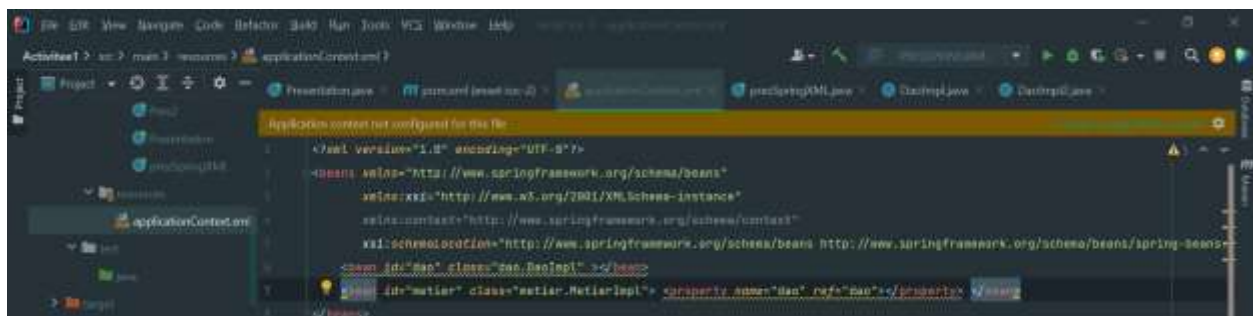
Run: Pres2
"C:\Program Files\Java\jdk1.8.0_202\bin\java.exe" ...
Version 2
48.8

C - Utilisation du Framework Spring (Version XML)

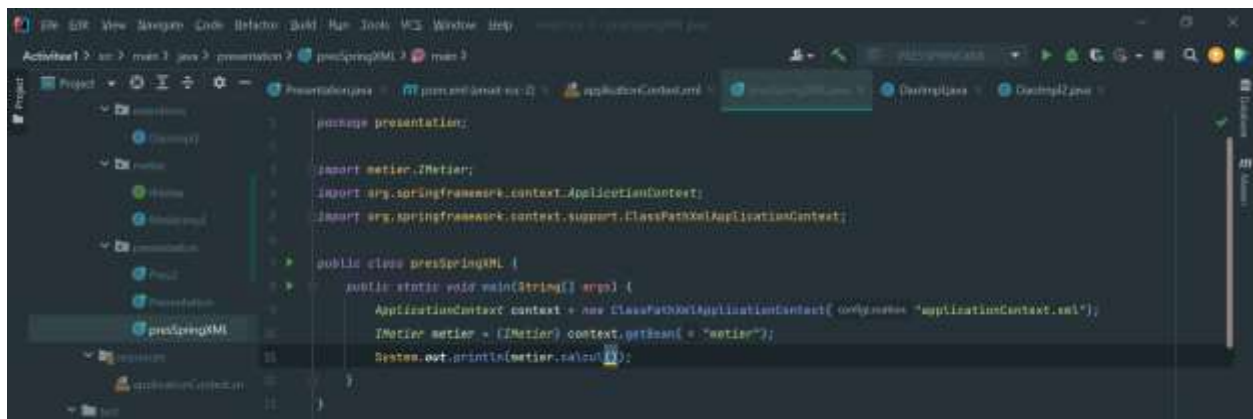
Intégration des dépendances de Spring (module Spring-core, Spring-Context, Spring-beans) dans fichier **pom.xml** :



Après on va créer un fichier des dépendances de Spring dans les ressources



Pour la class PresSpringXML (vesion XML)



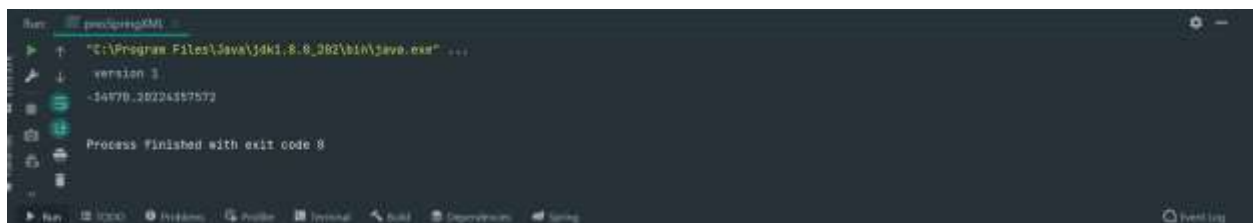
```
package presentation;

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

public class PresSpringXML {

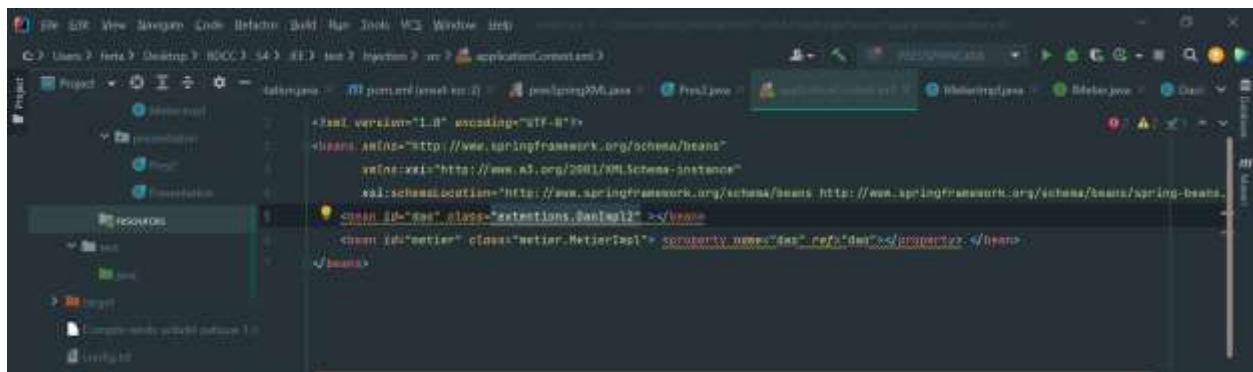
    public static void main(String[] args) {
        ApplicationContext context = new ClassPathXmlApplicationContext("classpath:applicationContext.xml");
        IMetier metier = (IMetier) context.getBean("metier");
        System.out.println(metier.calcul());
    }
}
```

Le résultat de l'exécution :

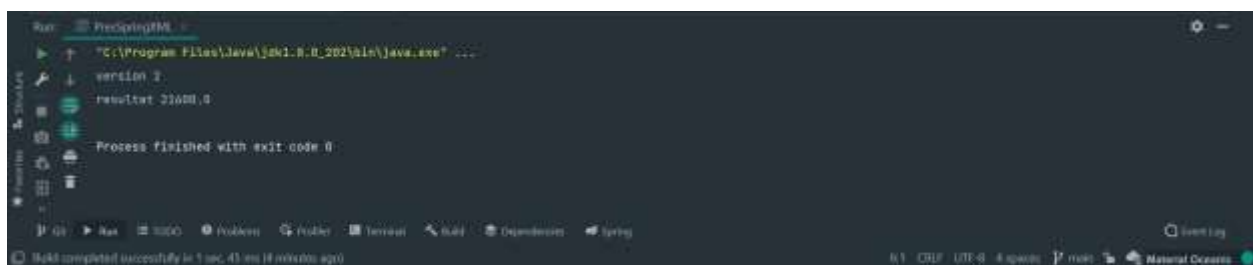


```
Run: PresSpringXML
"C:\Program Files\Java\jdk-8.0.382\bin\java.exe" ...
version 1
-14978.20224357572
Process finished with exit code 0
```

Pour basculer vers une autre extension, il faut changer la class dans le fichier applicationContext.xml

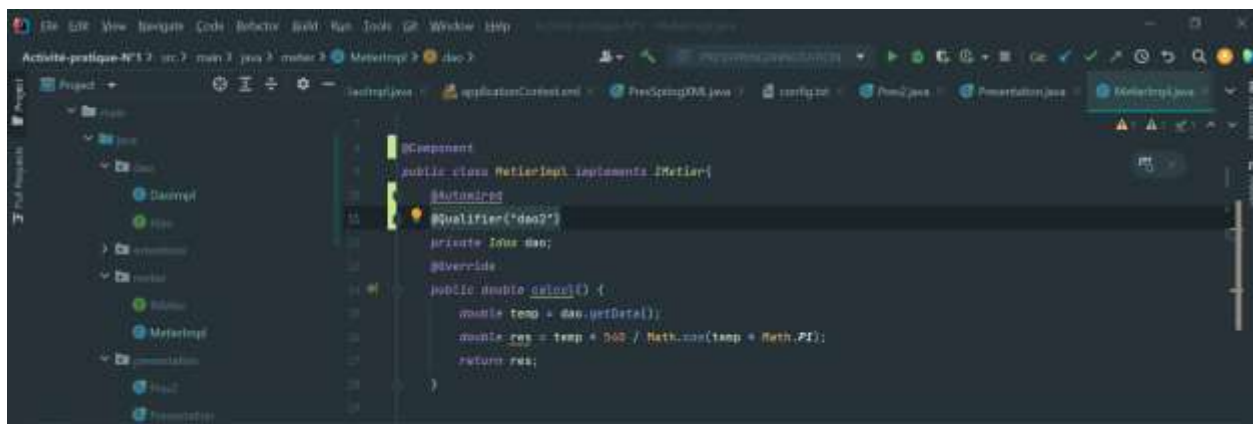
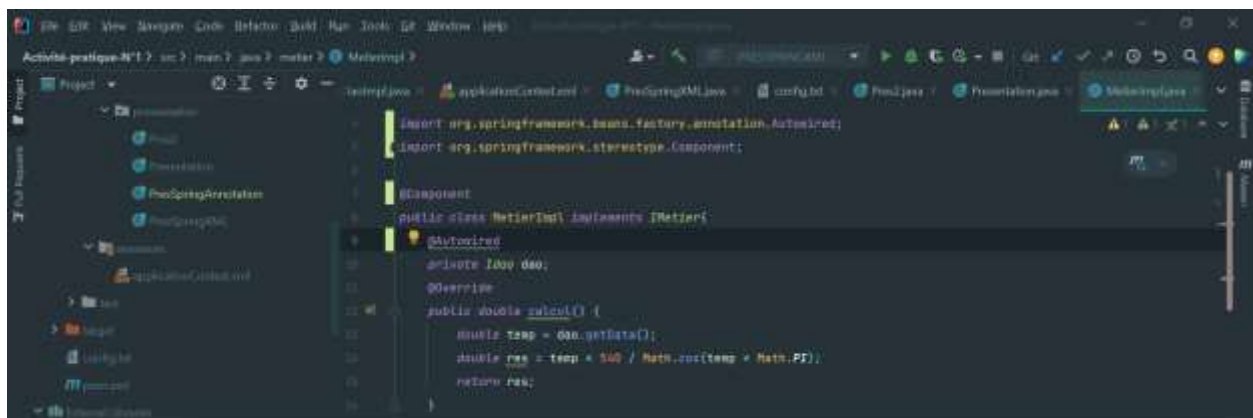
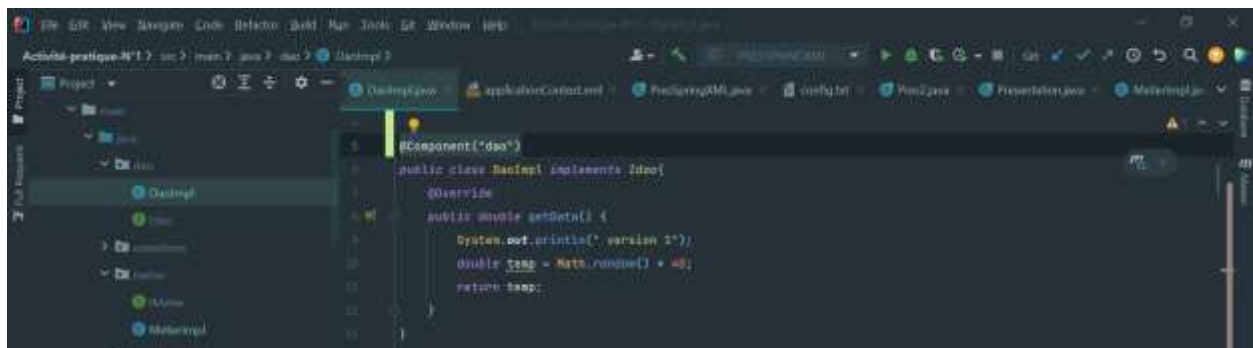


```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">
   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



```
Run: PresSpringXML
"C:\Program Files\Java\jdk-8.0.382\bin\java.exe" ...
version 1
resultat 23680,0
Process finished with exit code 0
```


Utilisation du Framework Spring (Version Annotations)



Concision

Ce TP m'a aidé à bien adopter la notion d'Inversion de contrôle et Injection de Dépendances grâce aux différentes instanciations. J'ai aussi découvert l'importance d'utilisation des Frameworks vu qu'il facilite le travail. Enfin, le plus important dans ce TP est de savoir que notre code doit être fermé à la modification et ouvert à l'extension