Activité Pratique N° 1 - Inversion de contrôle et Injection des dépendances

Rendre un compte rendu en reprenant l'exemple traité dans les vidéos des deux dernières séances

Partie 1 : (Voir support et vidéo)

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

// Pour le moment à ignorer

Partie 2 : Mini Projet (Framework Injection des dépendance)

Développer un mini Framework qui permet de faire l'injection des dépendances avec ses deux version XML et Annotations

Concevoir et créer un mini Framework d'injection des dépendances similaire à Spring IOC

Le Framework doit permettre à un programmeur de faire l'injection des dépendances entre les différents composant de son application respectant les possibilités suivantes :

1- A travers un fichier XML de configuration en utilisant Jax Binding (OXM : Mapping Objet XML)

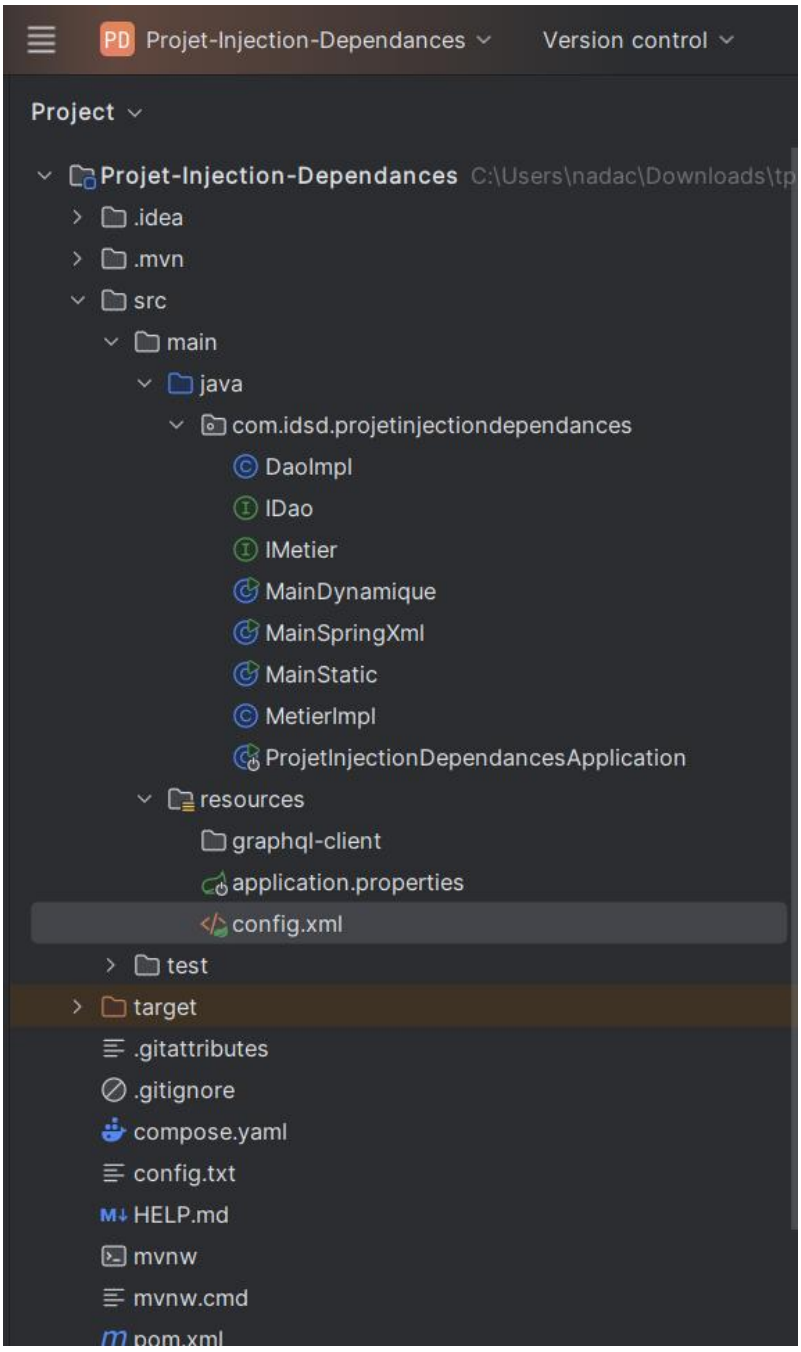
2- En utilisant les annotations

3- Possibilité d'injection via :

a- Le constructeur

b- Le Setter

c- Attribut (accès direct à l'attribut : Field)



```java
package com.idsd.projetinjectiondependances;
public class DaoImpl implements IDao {  4 usages
    @Override  1 usage
    public double getData() {
        System.out.println("[DAO] Accès à la base de données...");
        return 42; // Une valeur simulée
    }
}
```

```java
package com.idsd.projetinjectiondependances;

public interface IDao {  5 usages  1 implementation
    double getData();  1 usage  1 implementation
}
```

```java
package com.idsd.projetinjectiondependances;

public interface IMetier {  3 usages  1 implementation
    double calcul();  3 usages  1 implementation
}
```

```java
package com.idsd.projetinjectiondependances;
import java.io.File;
import java.util.Scanner;

public class MainDynamique {
    public static void main(String[] args) throws Exception {
        Scanner scanner = new Scanner(new File( pathname: "config.txt"));
        String daoClassName = scanner.nextLine();
        String metierClassName = scanner.nextLine();

        Class cDao = Class.forName(daoClassName);
        IDao dao = (IDao) cDao.getConstructor().newInstance();

        Class cMetier = Class.forName(metierClassName);
        MetierImpl metier = (MetierImpl) cMetier.getConstructor().newInstance();

        metier.setDao(dao); // injection dynamique
        System.out.println("Résultat Dynamique = " + metier.calcul());
    }
}
```

```java
package com.idsd.projetinjectiondependances;

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

public class MainSpringXml {
    public static void main(String[] args) {
        ApplicationContext context = new ClassPathXmlApplicationContext( configLocation: "config.xml");
        IMetier metier = (IMetier) context.getBean( name: "metier");
        System.out.println("Résultat Spring XML = " + metier.calcul());
    }
}
```

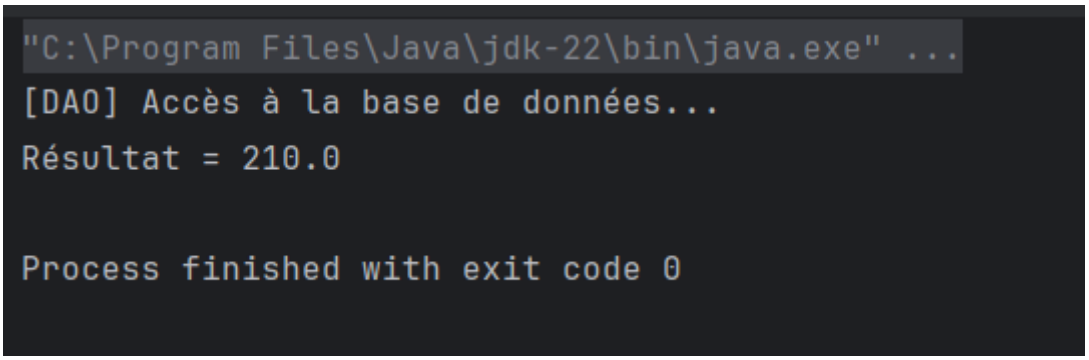
```java
package com.idsd.projetinjectiondependances;

public class MainStatic {
    public static void main(String[] args) {
        DaoImpl dao = new DaoImpl(); // instanciation statique
        MetierImpl metier = new MetierImpl();
        metier.setDao(dao); // injection via setter
        System.out.println("Résultat = " + metier.calcul());
    }
}
```

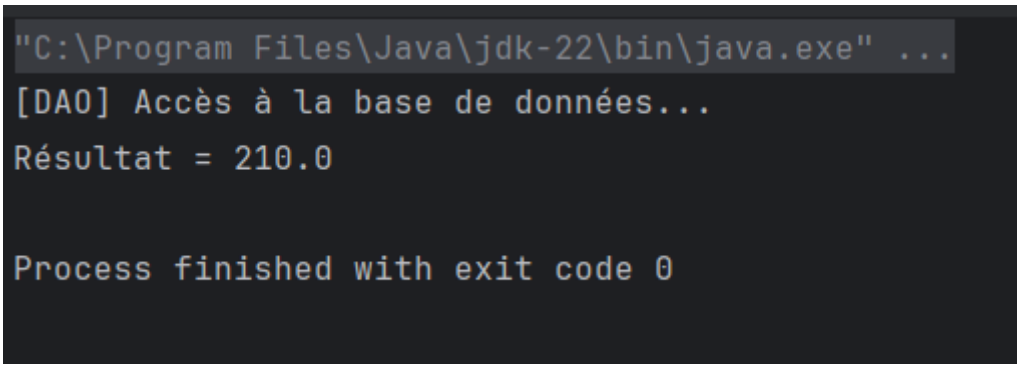
```java
package com.idsd.projetinjectiondependances;

public class MetierImpl implements IMetier {  6 usages
    private IDao dao; // Couplage faible via interface  2 usages

    // Injection via Setter
    public void setDao(IDao dao) {
        this.dao = dao;
    }

    @Override  3 usages
    public double calcul() {
        double data = dao.getData();
        double resultat = data * 5; // logique métier
        return resultat;
    }
}
```

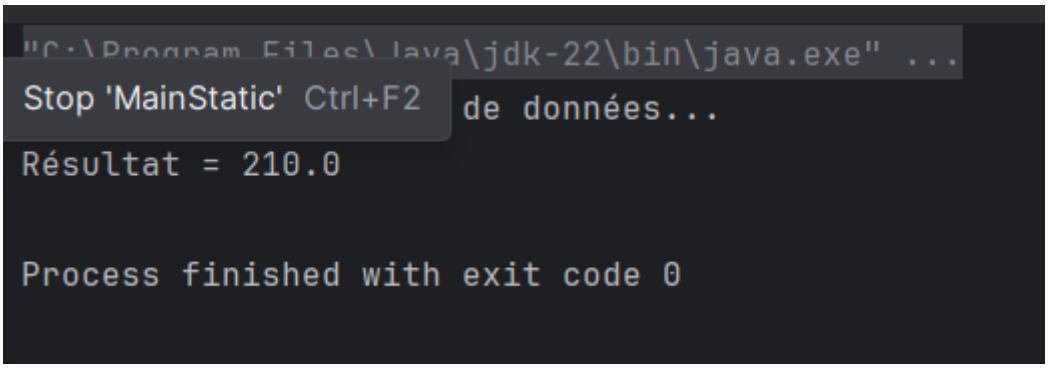
```java
package com.idsd.projetinjectiondependances;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class ProjetInjectionDependancesApplication {

    public static void main(String[] args) { SpringApplication.run(ProjetInjectionDependancesApplication.class, args

}
```

```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="dao" class="com.idsd.projetinjectiondependances.DaoImpl"/>
    <bean id="metier" class="com.idsd.projetinjectiondependances.MetierImpl">
        <property name="dao" ref="dao"/>
    </bean>

</beans>
```

Tests :


"C:\Program Files\Java\jdk-22\bin\java.exe" ...
[DAO] Accès à la base de données...
Résultat = 210.0
Process finished with exit code 0


"C:\Program Files\Java\jdk-22\bin\java.exe" ...
[DAO] Accès à la base de données...
Résultat = 210.0
Process finished with exit code 0


Stop 'MainStatic' Ctrl+F2
de données...
Résultat = 210.0
Process finished with exit code 0