

DEPARTEMENT MATHÉMATIQUES ET INFORMATIQUE

Compte rendu de l'activité pratique 2

**Principe de l'Inversion de Contrôle et Injection des dépendances
avec Spring**

**Filière :
« Génie du Logiciel et des Systèmes Informatiques Distribués »
GLSID2**

Module : Architecture Distribuée et Middlewares

Élaboré par :

ELMAJNI Khaoula

Encadré par :

M. YOUSSEFI Mohammed

Année Universitaire : 2021-2022

Introduction

Dans ce travail pratique on va décortiquer l'injection des dépendances par un framework 'Spring' avec 2 versions : XML et Annotation.

L'injection de dépendances est la base de tout programme moderne.

Il est mieux de faire la liaison des composants du programme dans un framework afin de pouvoir facilement changer ces composants ou leur comportement, et pour ça on a utilisé le Spring IOC.

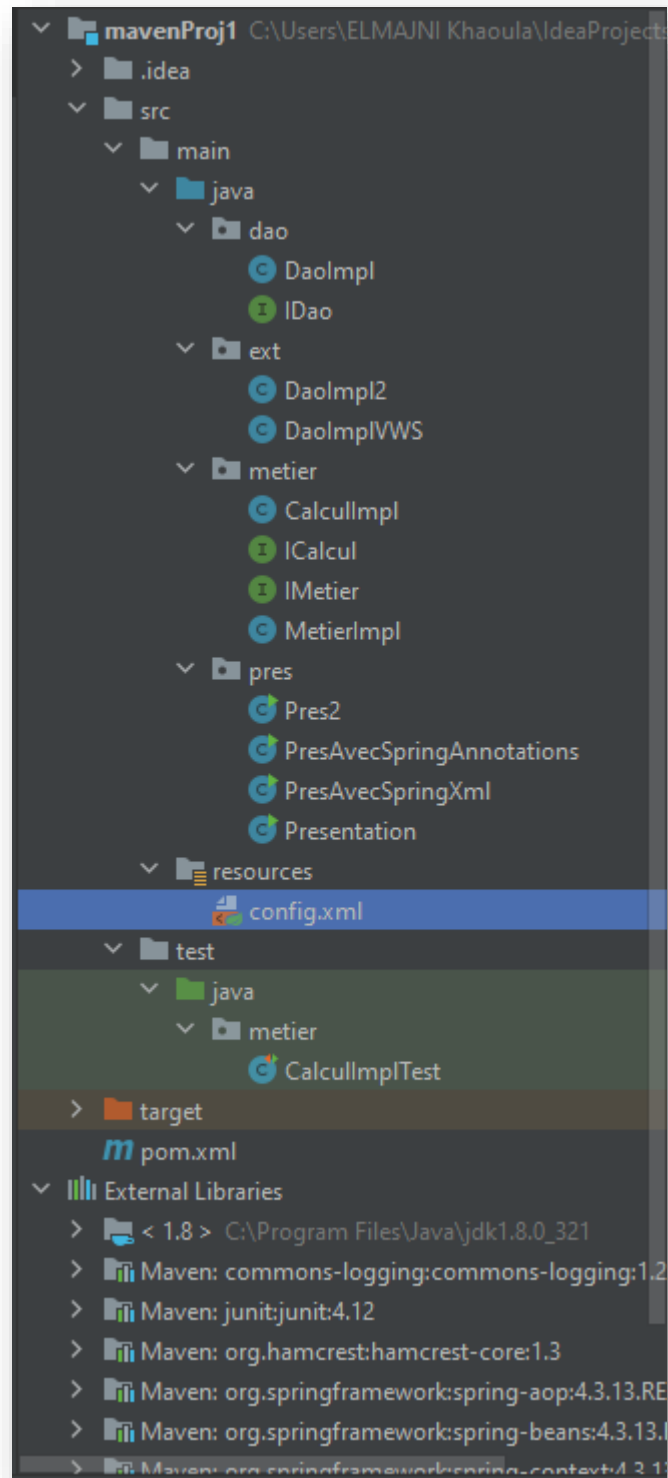
Spring IOC

Spring IOC conciste sur lire un fichier XML qui déclare quelles sont différentes classes à instancier et d'assurer les dépendances entre les différentes instances, et pour faire une nouvelle implémentation dans l'application on la déclare dans le fichier xml de spring.

Pour démarrer Spring il y a 2 versions :

- **Xml** : Spring va lire le fichier xml de configuration spring puis il va s'occuper des injections des dépendances.
- **Annotations** : il suffit d'ajouter des annotations aux classes pour déclarer au Spring qu'il doit les instancier au démarrage de l'application, ainsi qu'auprès des objets qu'on doit attribuer des dépendances aux autres instances des classes qui sont déjà déclarées.

Arcitecture du travail



Dans «External librairies» on a téléchargé 3 jars :

1. Spring core
2. Spring context
3. Spring beans

Ces jars vont être utilisés par Spring.

Le fichier xml de configuration de spring, dans la balise « beans » on déclare les instances qu'on a besoin avec 'id' est le nom de chaque instance, et 'class' le nom de la classe, et pour injecter la dépendance il y a la balise « property » avec un 'name' nom de l'objet et 'ref' la référence vers quelle instance :

```
<?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/sche
ma/beans
http://www.springframework.org/schema/beans/spring-
beans.xsd">

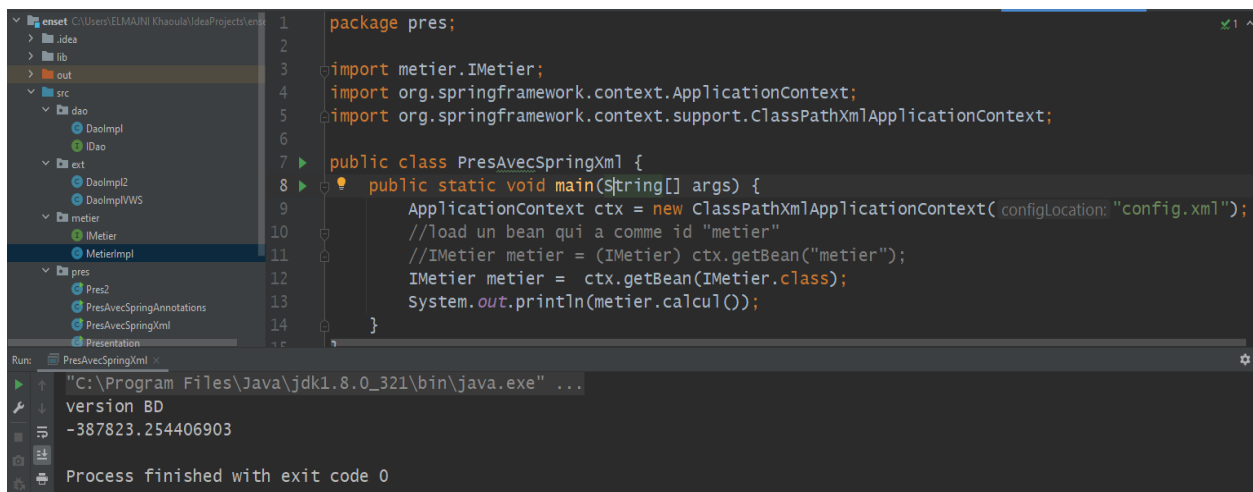
    <bean id="d" class="dao.DaoImpl" bean>
    <bean id="metier" class="metier.MetierImpl">
        <property name="dao" ref="d"></property>
    </bean>
</beans>
```

```

1 package pres;
2
3 import metier.IMetier;
4 import org.springframework.context.ApplicationContext;
5 import org.springframework.context.support.ClassPathXmlApplicationContext;
6
7 public class PresAvecSpringXml {
8     public static void main(String[] args) {
9         ApplicationContext ctx = new ClassPathXmlApplicationContext("config.xml");
10        //load un bean qui a comme id "metier"
11        //IMetier metier = (IMetier) ctx.getBean("metier");
12        IMetier metier = ctx.getBean(IMetier.class);
13        System.out.println(metier.calcul());
14    }
15 }

```

Figure 1 : la présentation de l'application à partir du fichier de configuration xml de spring



```

1 package pres;
2
3 import metier.IMetier;
4 import org.springframework.context.ApplicationContext;
5 import org.springframework.context.support.ClassPathXmlApplicationContext;
6
7 public class PresAvecSpringXml {
8     public static void main(String[] args) {
9         ApplicationContext ctx = new ClassPathXmlApplicationContext("config.xml");
10        //load un bean qui a comme id "metier"
11        //IMetier metier = (IMetier) ctx.getBean("metier");
12        IMetier metier = ctx.getBean(IMetier.class);
13        System.out.println(metier.calcul());
14    }
15 }

```

Run: PresAvecSpringXml x

"C:\Program Files\Java\jdk1.8.0_321\bin\java.exe" ...

version BD

-387823.254406903

Process finished with exit code 0

Figure 2: exécution de la version xml

En ce qui concerne la version annotation :

Au niveau de l'implémentation de DAO :

```

package dao;

import org.springframework.stereotype.Component;
import org.springframework.stereotype.Repository;

@Repository("dao") //==> remplace le <bean id="dao"
class="dao.DaoImpl"></bean> dans la version xml
public class DaoImpl implements IDao{
    @Override

```

```
public double getData() {
    /*
     * se connecter a la BD pour recuperre la
    temperature
     */
    System.out.println("version BD");
    double temp = Math.random()*40;
    return temp;
}

public void init(){
    System.out.println("DaoImpl Instatnciation");
}
}
```

au niveau de l'implémentation du 'IMetier' :

l'annotation @Autowired est pour faire l'injection des dépendances.

```
package metier;

import dao.IDao;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import org.springframework.stereotype.Service;

@Service("metier")
public class MetierImpl implements IMetier{
    //couplage faible

    @Autowired
    private IDao dao=null;

    @Override
    public double calcul() {
        double temp = dao.getData();//n'importe quel
source:classe
        double res=temp*540/Math.cos(temp*Math.PI);
        return res;
    }

    //permet d'injecter dans la variable dao un obj
```

```
d'une classe qui implemente l'interface IDao
    public void setDao(IDao dao) {
        this.dao = dao;
        System.out.println("injections des
dependances");
    }

    public void init(){
        System.out.println("MetierImpl
Initialisation");
    }

    public MetierImpl() {
        System.out.println("MetierImpl Instantiation");
    }
}
```

au niveau de la présentation :

```
package pres;

import metier.IMetier;
import org.springframework.context.ApplicationContext;
import
org.springframework.context.annotation.AnnotationConfig
ApplicationContext;

public class PresAvecSpringAnnotations {
    public static void main(String[] args) {
        //je le donne les packages à scanner ==> ou il
se trouve les annotations @Component
        ApplicationContext ctx = new
AnnotationConfigApplicationContext("ext","metier");

        IMetier metier = ctx.getBean(IMetier.class);
        System.out.println(metier.calcul());
    }
}
```

```

package pres;

import metier.IMetier;
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class PresAvecSpringAnnotations {
    public static void main(String[] args) {
        //je le donne les packages à scanner ==> ou il se trouve les annotations @Component
        ApplicationContext ctx = new AnnotationConfigApplicationContext(...basePackages: "ext","metier");

        IMetier metier = ctx.getBean(IMetier.class);
        System.out.println(metier.calcul());
    }
}

```

Run: PresAvecSpringAnnotations

```

"C:\Program Files\Java\jdk1.8.0_321\bin\java.exe" ...
MetierImpl Instantiation
version : web service
48600.0
Process finished with exit code 0

```

Figure 3 : exécution de la version annotation

Maven project

Maven : est un outil n'est pas d'un framework, qui permet l'automatisation des processus de développement d'un projet java, il utilise un paradigme connu sous le nom de **POM (Project Object Model)**.

Principe : à chaque fois on ajoute une dépendance au fichier xml 'pom.xml' il va chercher dans le 'repository local' s'il en trouve il va les utiliser, sinon il va se connecter à l'internet et il va télécharger les dépendances déclarées.

Users > ELMAJNI Khaoula > .m2 > repository > org > springframework

Name	Date modified	Type	Size
spring-aop	2/21/2022 10:09 AM	File folder	
spring-beans	2/21/2022 10:09 AM	File folder	
spring-context	2/21/2022 10:09 AM	File folder	
spring-core	2/21/2022 10:08 AM	File folder	
spring-expression	2/21/2022 10:09 AM	File folder	
spring-jcl	2/21/2022 10:08 AM	File folder	

Le fichier pom.xml :

```

<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"

```



```
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>ma.enset</groupId>
  <artifactId>enset-ioc-2</artifactId>
  <version>1.0-SNAPSHOT</version>

  <properties>
    <maven.compiler.source>8</maven.compiler.source>
    <maven.compiler.target>8</maven.compiler.target>
  </properties>

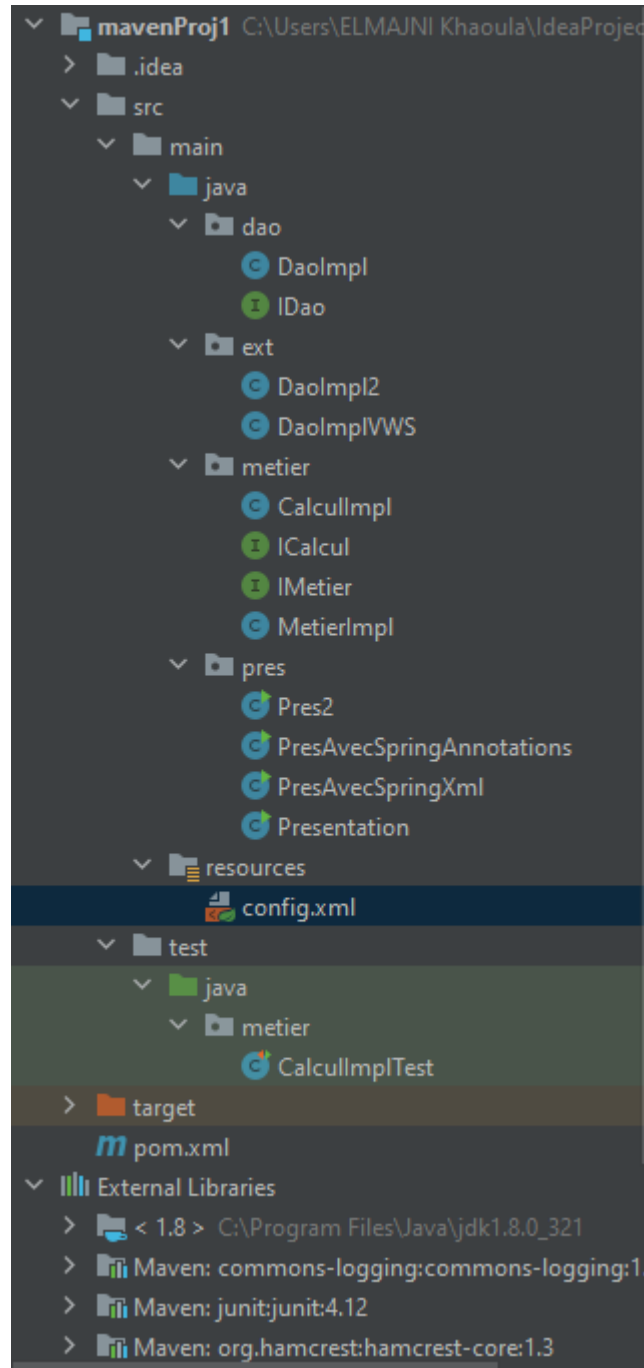
  <dependencies>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-core</artifactId>
      <version>5.3.16</version>
    </dependency>

    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-context</artifactId>
      <version>5.3.16</version>
    </dependency>

    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-beans</artifactId>
      <version>5.3.16</version>
    </dependency>

    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.12</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

Architecture du projet maven :



1->mvn compile : cette commande compile le code source du projet

2->mvn test : avec cette commande maven va parcourir le projet et à chaque fois il trouve un test unitaire il va l'exécuter, puis il montre qui sont les tests réussit et qui ne sont pas.

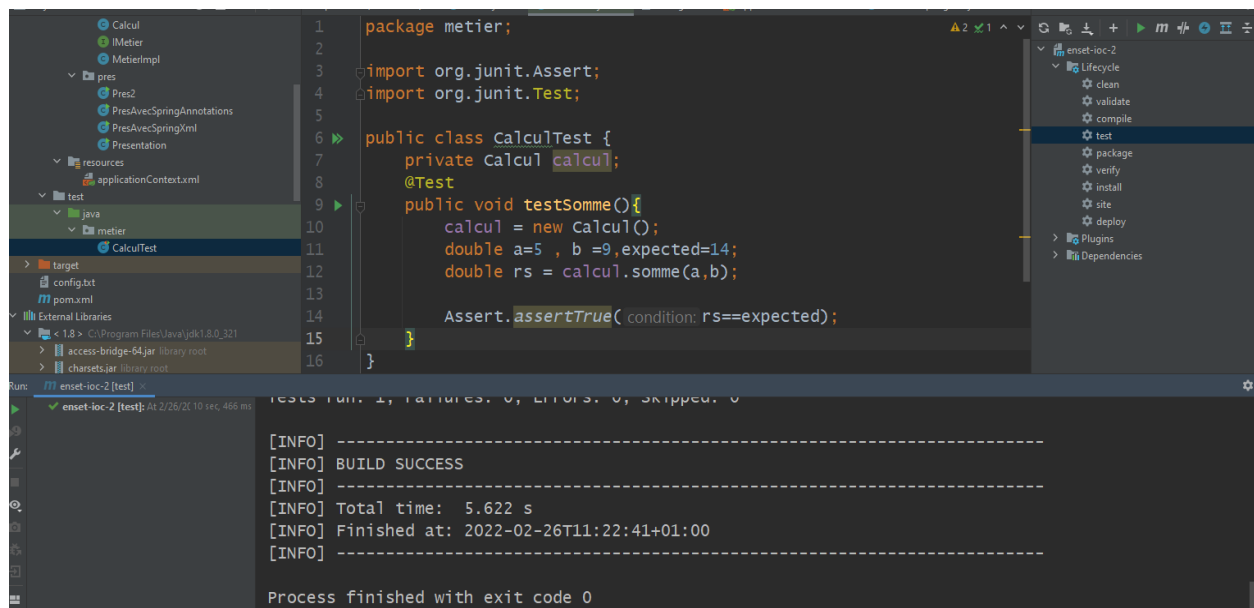
On a utilisé un test unitaire sur une classe 'Calcul' grace à JUnit qui est un outil de test unitaire de Java, comme suit :

```
package metier;

import org.junit.Assert;
import org.junit.Test;

public class CalculTest {
    private Calcul calcul;
    @Test
    public void testSomme(){
        calcul = new Calcul();
        double a=5 , b =9, expected=14;
        double rs = calcul.somme(a,b);

        Assert.assertTrue(rs==expected);
    }
}
```



3->mvn package: cette commande exécute la commande 1 et 2 puis archive le projet maven dans archive (.jar / .war)

4->mvn install : cette commande exécute la commande 1,2 et 3 puis elle installe le projet dans le repository local pour l'utiliser au cas de besoin.

>mvn deploy : cette commande pour déployer un projet vers un serveur

>mvn site : générer un site de documentation

Conclusion

Dans ce travail on a étudié par la pratique l'injection des contrôle (IOC) par le framework Spring, avec ses 2 versions XML et Annotations, et la particularité de chaque version, ainsi qu'on a initié avec l'outil Maven d'automatisation des dépendances des projets java, aussi il permet la gestion de cycle de vie du projet.