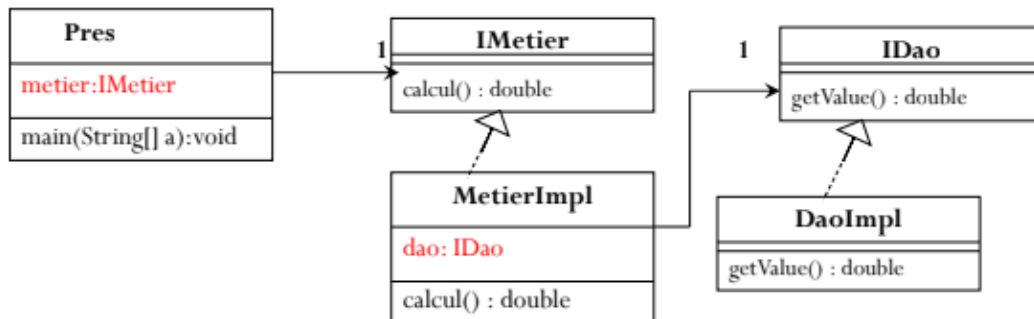


Activité pratique N°1

Inversion de contrôle et Injection des dépendances

Hajar TAJAYOUTI

GLSID2



1. Créer l'interface IDao

Nous avons ajouté dans cette classe une méthode abstraite **getValue()** qui permet de retourner une valeur de type double.

```
package dao;

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

2. Créer une implémentation de cette interface

```
package dao;

public class DaoImpl implements IDao{

    public double getValue() {
        return 5;
    }
}
```

3. Créer l'interface IMetier

Cette interface représente **les besoins fonctionnels**, nous avons ajouté une méthode abstraite **calcul()** qui permet de faire des traitement de calcule et par la suite retourner une valeur de type double.

```
package metier;

public interface IMetier {

    public double calcul();

}
```

4. Créer une implémentation de cette interface en utilisant le couplage faible

Nous avons créé un objet **dao** de type **IDao** pour assurer le **couplage faible**, et afin d'**injecter** dans la variable **dao** un objet d'une classe qui implémente l'Interface **IDao** nous avons créé la méthode **setDao()**.

```
package metier;
import dao.IDao;
public class MetierImpl implements IMetier{

    private IDao dao;

    public double calcul() {

        double nb=dao.getValue();
        return nb*2;
    }

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

5. Faire l'injection des dépendances :

a. Par instantiation statique

```
1 package presentation;
2
3 import ...
4
15
16 public class pres1 {
17
18     public static void main(String[] args) throws Exception {
19         /*injection des dependances par instantiation statique*/
20
21         DaoImpl dao=new DaoImpl();
22         MetierImpl metier=new MetierImpl();
23         metier.setDao(dao); //injection... invoquer la methode set sur l objet metier
24         System.out.println(metier.calcul());
25     }
26 }
```



- IDao** est fermée à la modification puisqu'elle ne dépend pas d'une autre classe, elle est ouverte à l'extension vue qu'il s'agit d'une interface.
- MetierImpl** est fermée à la modification puisqu'elle dépend que d'une interface et elle est ouverte à l'extension vue qu'il s'agit d'une interface.
- pres1** n'est fermée à la modification ! Pour rendre cette dernière fermée à la modification nous devons faire une instantiation

b. Par instantiation dynamique

Nous avons créé un fichier de configuration **conf.text** pour déclarer les noms des classes que on va utiliser .

```
conf.txt x
1 dao.DaoImpl
2 metier.MetierImpl
3
```

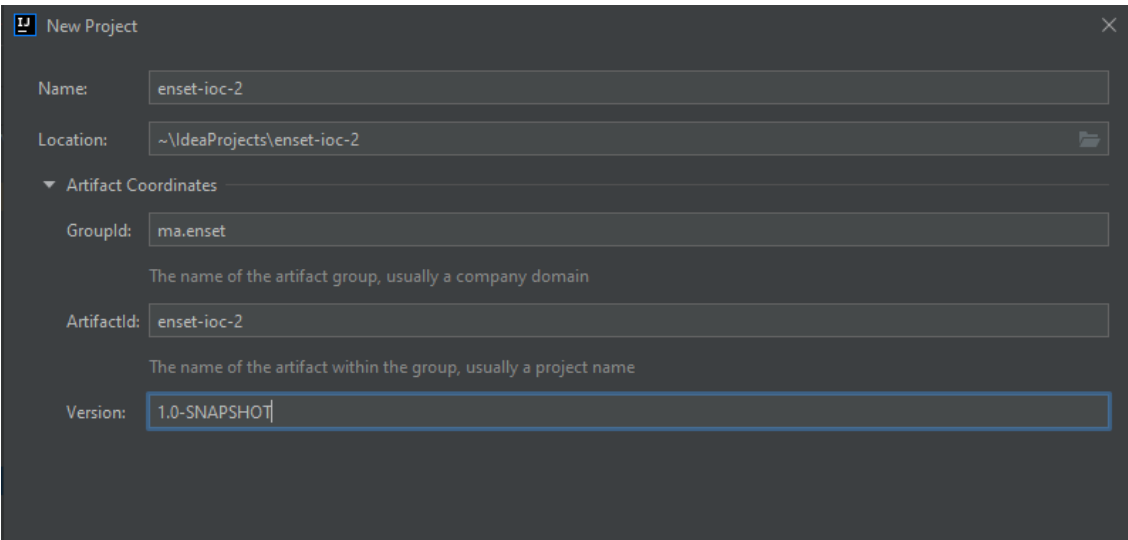
Nous avons instancié la classe **DaoImpl** et **MetierImp** par la methode **newInstance** (instanciation dynamique), par la suite nous avons appelé d'une façon dynamique la fonction **setDao ()**.

```
Scanner scanner=new Scanner(new File( pathname: "conf.txt"));
String daoClassName=scanner.nextLine();
Class cDao=Class.forName(daoClassName);
IDao dao=(IDao) cDao.newInstance();

String metierClassName=scanner.nextLine();
Class cMetier=Class.forName(metierClassName);
IMetier metier=(IMetier) cMetier.newInstance();
//app dynamique d'une methode
Method m=cMetier.getMethod( name: "setDao", IDao.class);
m.invoke(metier,dao);
System.out.println(metier.calcul());
```

c. En utilisant le Framework Spring

Nous avons créé un projet Maven



The screenshot shows the 'New Project' window in an IDE. The fields are filled as follows:

- Name: enset-ioc-2
- Location: ~\IdeaProjects\enset-ioc-2
- Artifact Coordinates:
 - GroupId: ma.enset (with a note: 'The name of the artifact group, usually a company domain')
 - ArtifactId: enset-ioc-2 (with a note: 'The name of the artifact within the group, usually a project name')
 - Version: 1.0-SNAPSHOT

Nous avons inclus le framwork **Spring** dans le fichier **pom.xml** ,pour faire l'injection des dépendances nous avons besoin de 3 modules **Spring core** ,**Spring context** et **Spring beans** .

```

<dependencies>
  <!-- https://mvnrepository.com/artifact/org.springframework/spring-core -->
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>5.3.16</version>
  </dependency>

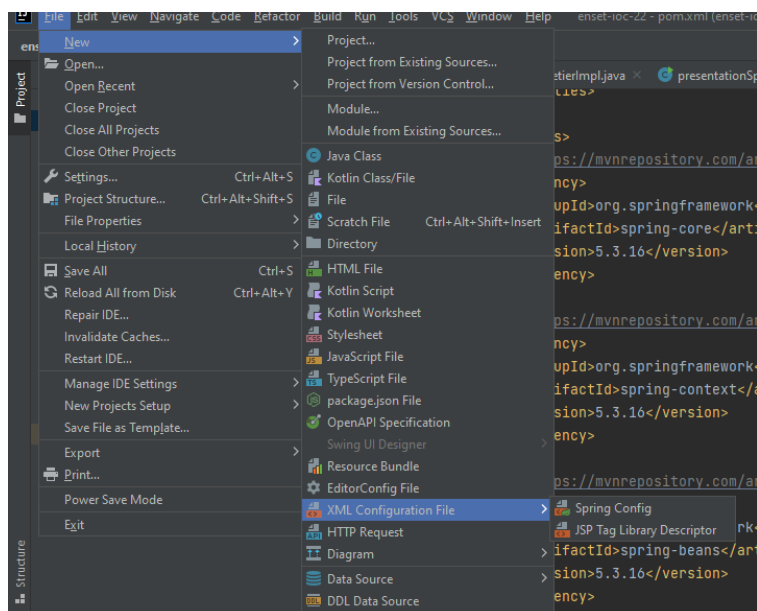
  <!-- https://mvnrepository.com/artifact/org.springframework/spring-context -->
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>5.3.16</version>
  </dependency>

  <!-- https://mvnrepository.com/artifact/org.springframework/spring-beans -->
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-beans</artifactId>
    <version>5.3.16</version>
  </dependency>
</dependencies>

```

⇒ **Version XML**

Nous avons créé un fichier XML « 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
       <bean id="dao" class="dao.DaoImpl"></bean> <!--creation d'un objet de type DaoImpl et lui donner comme nom dao-->
       <bean id="metier" class="metier.MetierImpl">
         <property name="dao" ref="dao"></property> <!--faire l injection des dependences-->
       </bean>
```

```
package presentation;

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

public class presentationSpringXML {
    public static void main(String[] args) {
        ApplicationContext context=new ClassPathXmlApplicationContext( configLocation: "applicationContext.xml");

        IMetier metier=(IMetier)context.getBean( s: "metier");
        System.out.println(metier.calcul());
    }
}
```

⇒ *Version annotations*

à l'aide de l'annotation **@Component** nous pouvons instancier la classe **DaoImpl** et lui donner comme nom « dao ».

```
package dao;

import org.springframework.stereotype.Component;

@Component("dao")

public class DaoImpl implements IDao{

    public double getValue() {

        return 5;
    }
}
```

La même chose pour MetierImpl .

Pour faire l'injection des dépendances nous avons utilisé l'annotation **@Autowired**

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

@Component
public class MetierImpl implements IMetier{
    @Autowired
    private IDao dao;

    public double calcul() {
        double nb=dao.getValue();
        return nb*2;
    }

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

```
package presentation;

import metier.IMetier;
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import java.net.SocketTimeoutException;

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

Injection via le constructeur

```
@Component
public class MetierImpl implements IMetier{

    private IDao dao;

    public MetierImpl(IDao dao)
    {
        this.dao = dao;
    }

    public double calcul() {

        double nb=dao.getValue();
        return nb*2;
    }
}
```

```
<?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.xsd">
    <bean id="dao" class="dao.DaoImpl"></bean> 
```