

DEPARTEMENT MATHÉMATIQUES ET INFORMATIQUE

Compte rendu

Filière :

« Génie du Logiciel et des Systèmes Informatiques Distribués »

GLSID

Principe de l'Inversion de Contrôle et Injection des dépendances

Réalisé par :

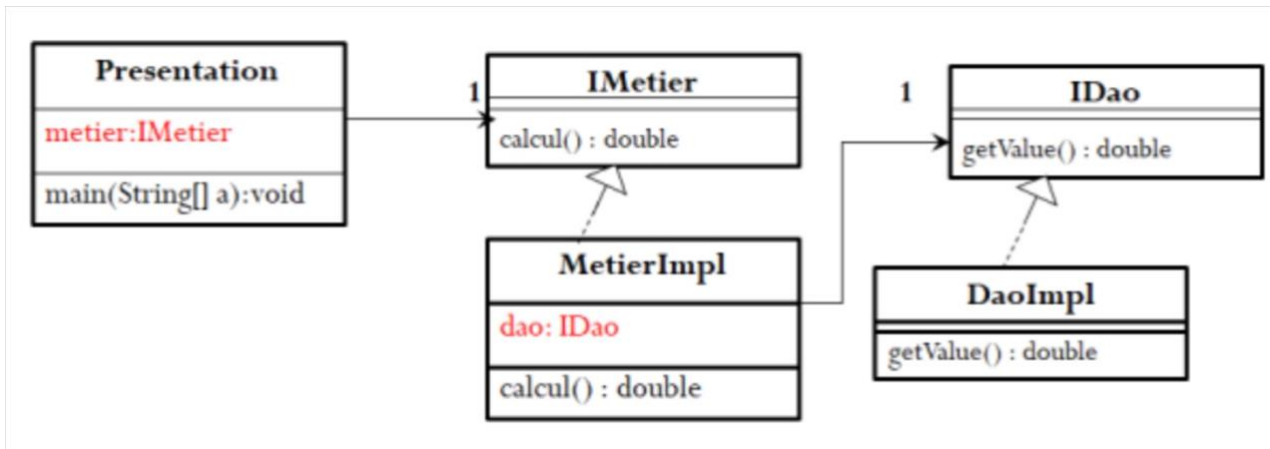
Hassan ELMAKHLOUFI

Encadré par :

Pr. Mohamed Youssfi

Année Universitaire : 2021-2022

Dans ce TP je vais implémenter le diagramme de classe suivant qui utilise le couplage faible avec deux version (version instanciation dynamique et version instanciation statique :



Interface IDao :

```
> .idea 2
> out 3
> src 4
  > dao 5
    DaoImp 6
    IDao 7
  > ext
```

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

Une Implémentation de l'interface IDao :

```
> .idea 2
> out 3
> src 4
  > dao 5
    DaoImp 6
    IDao 7
  > ext 8
  > metier 9
  > Presentation 10
  config.txt
```

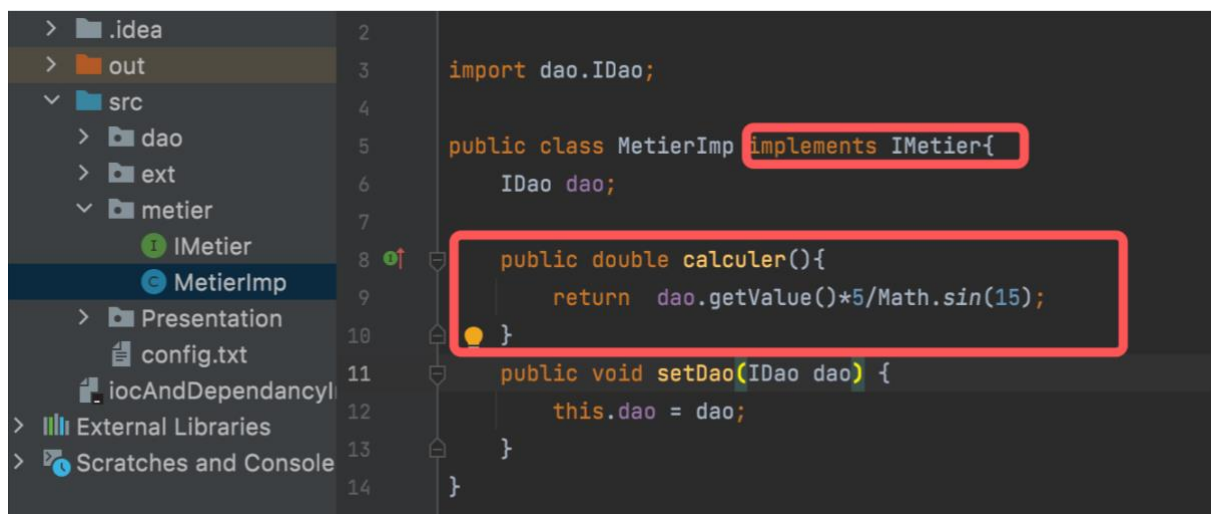
```
public class DaoImp implements IDao {

    @Override
    public double getValue() {
        System.out.println("version soa");
        return 10.1;
    }
}
```

Interface metier

```
IMetier.java x
1 package metier;
2
3 public interface IMetier {
4     double calculer ();
5 }
6
```

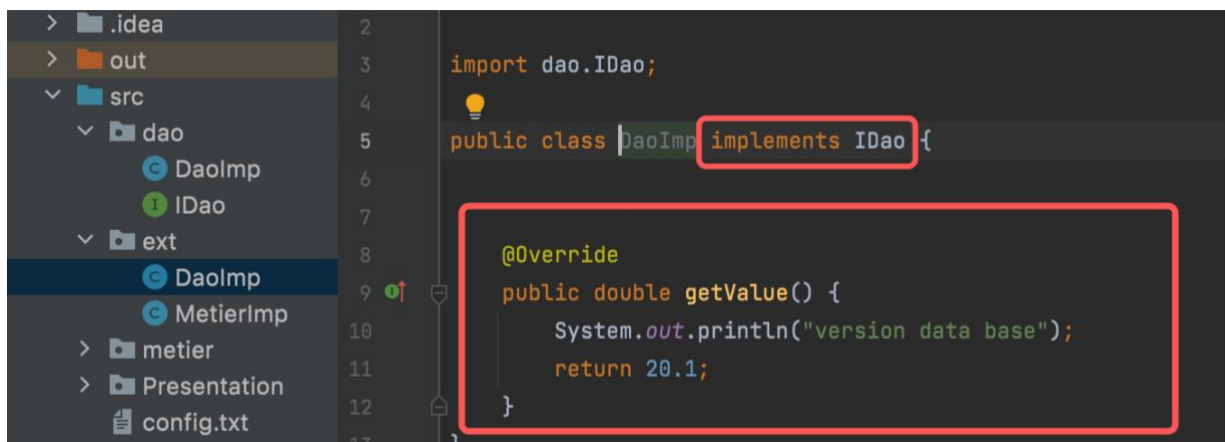
La première Implémentation de l'interface metier :



The screenshot shows the project structure on the left with 'src' expanded, showing 'dao' and 'metier' folders. The 'metier' folder contains 'IMetier' and 'MetierImp'. The 'MetierImp' class is selected. The code in the editor is as follows:

```
2
3 import dao.IDao;
4
5 public class MetierImp implements IMetier{
6     IDao dao;
7
8     public double calculer(){
9         return dao.getValue()*5/Math.sin(15);
10    }
11
12    public void setDao(IDao dao) {
13        this.dao = dao;
14    }
15 }
```

Une deuxième implémentation des interfaces :
Extension de DaoImpl :



The screenshot shows the project structure on the left with 'src' expanded, showing 'dao' and 'ext' folders. The 'ext' folder contains 'DaoImpl' and 'MetierImp'. The 'DaoImpl' class is selected. The code in the editor is as follows:

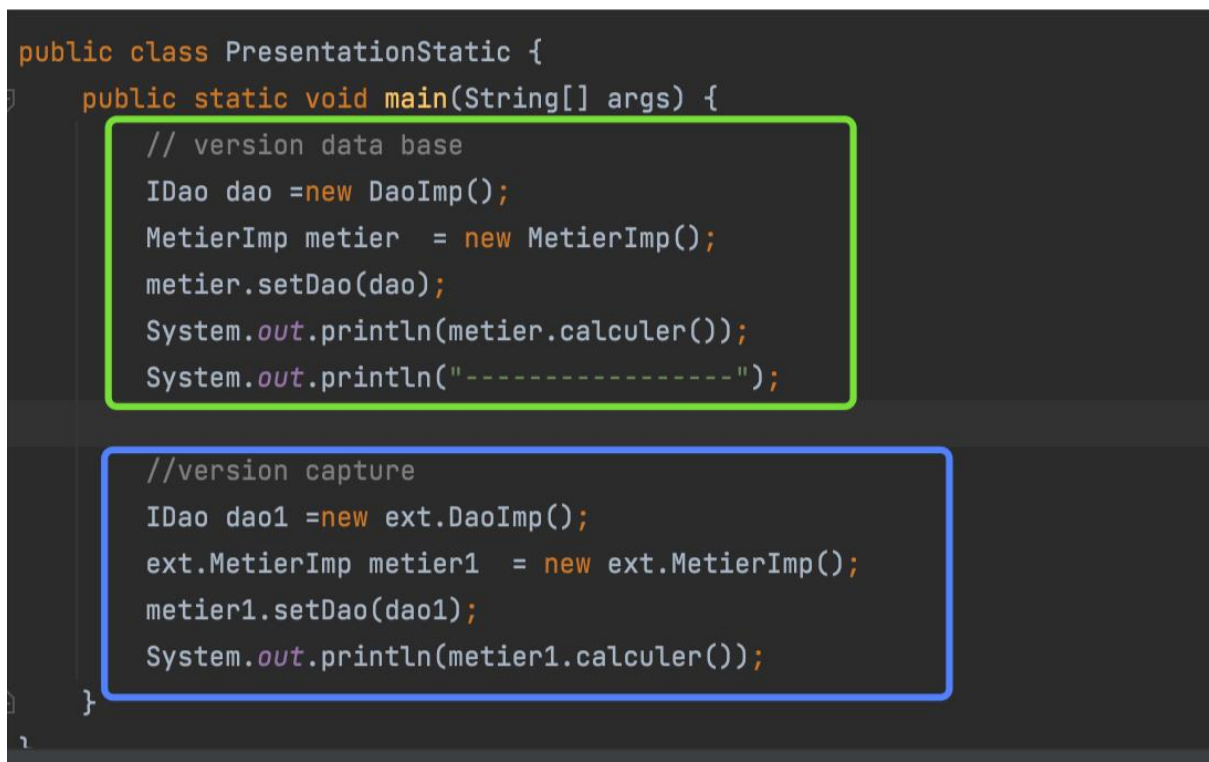
```
2
3 import dao.IDao;
4
5 public class DaoImpl implements IDao {
6
7
8     @Override
9     public double getValue() {
10         System.out.println("version data base");
11         return 20.1;
12     }
13 }
```

Extension du classe MetierImp :



```
2
3 import ...
4
5 public class MetierImp implements IMetier {
6
7     private IDao dao;
8
9     public double calculer(){
10         return dao.getValue()*5/Math.sin(15);
11     }
12
13     public void setDao(IDao dao) {
14
```

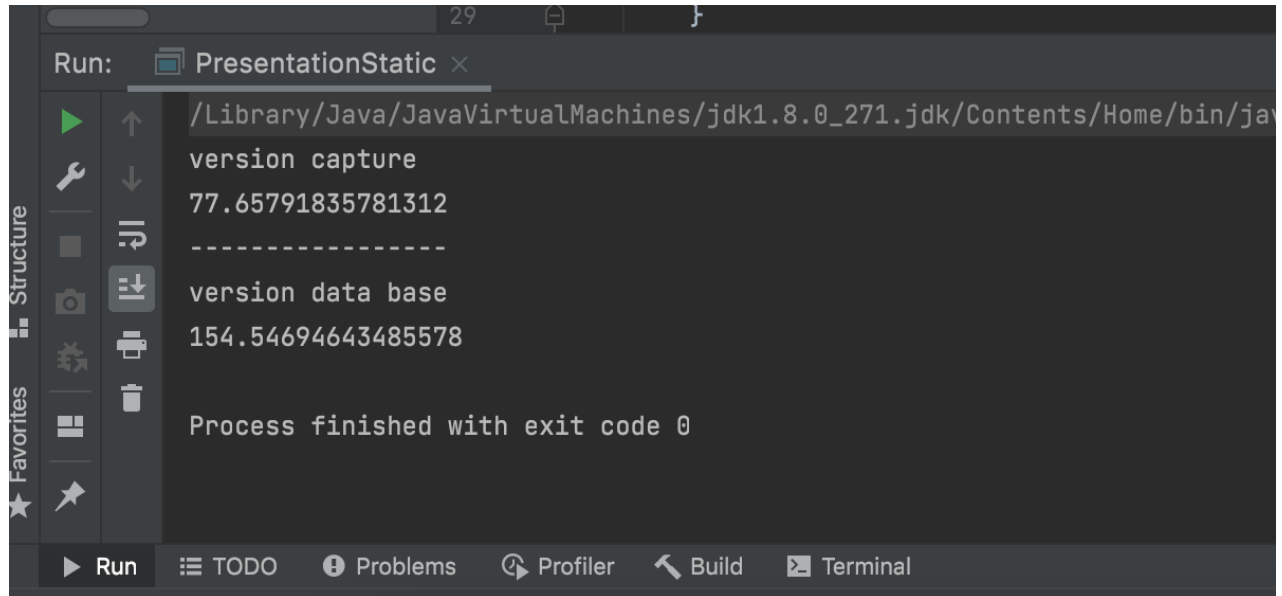
Injection des dépendances avec l'instanciation statique :



```
public class PresentationStatic {
    public static void main(String[] args) {
        // version data base
        IDao dao = new DaoImp();
        MetierImp metier = new MetierImp();
        metier.setDao(dao);
        System.out.println(metier.calculer());
        System.out.println("-----");

        //version capture
        IDao dao1 = new ext.DaoImp();
        ext.MetierImp metier1 = new ext.MetierImp();
        metier1.setDao(dao1);
        System.out.println(metier1.calculer());
    }
}
```

Le résultat du programme :



```
Run: PresentationStatic x
/Library/Java/JavaVirtualMachines/jdk1.8.0_271.jdk/Contents/Home/bin/java
version capture
77.65791835781312
-----
version data base
154.54694643485578

Process finished with exit code 0
```

Injection des dépendances avec l'instanciation dynamique :

- 1- Ouvrir le fichier config.txt afin de lire le contenu de fichier de configuration
- 2- Extraire le nom de la classe qui implémente de l'interface IDao et qui nous voulons utiliser, la charger dans la mémoire puis l'instancier
- 3- Extraire le nom de la classe qui implémente de l'interface IMetier et qui nous voulons utiliser, la charger dans la mémoire puis l'instancier
- 4- Injecter l'objet dao dans le dao du métier .

```

public class PresentationDynamic {
    public static void main(String[] args) {
        IMetier metier ;
        IDao dao ;
        try {
            Scanner sc = new Scanner(new File( pathname: "src/config.txt"));

            String daoClassName = sc.next();
            Class cdao = Class.forName(daoClassName);
            dao = (IDao) cdao.newInstance();

            String metierClassName = sc.next();
            Class cmetier = Class.forName(metierClassName);
            metier = (IMetier) cmetier.newInstance();

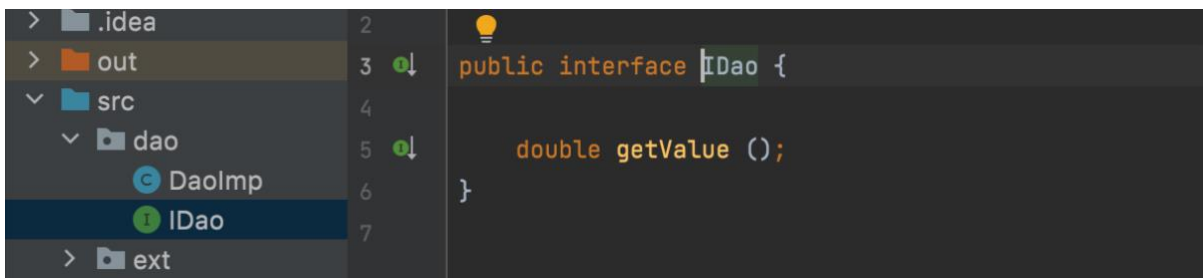
            Method mth= cmetier.getMethod( name: "setDao", IDao.class);
            mth.invoke(metier,dao);

            System.out.println(metier.calculer());
        }catch (Exception e){
            e.printStackTrace();
        }
    }
}

```

Partie Spring :

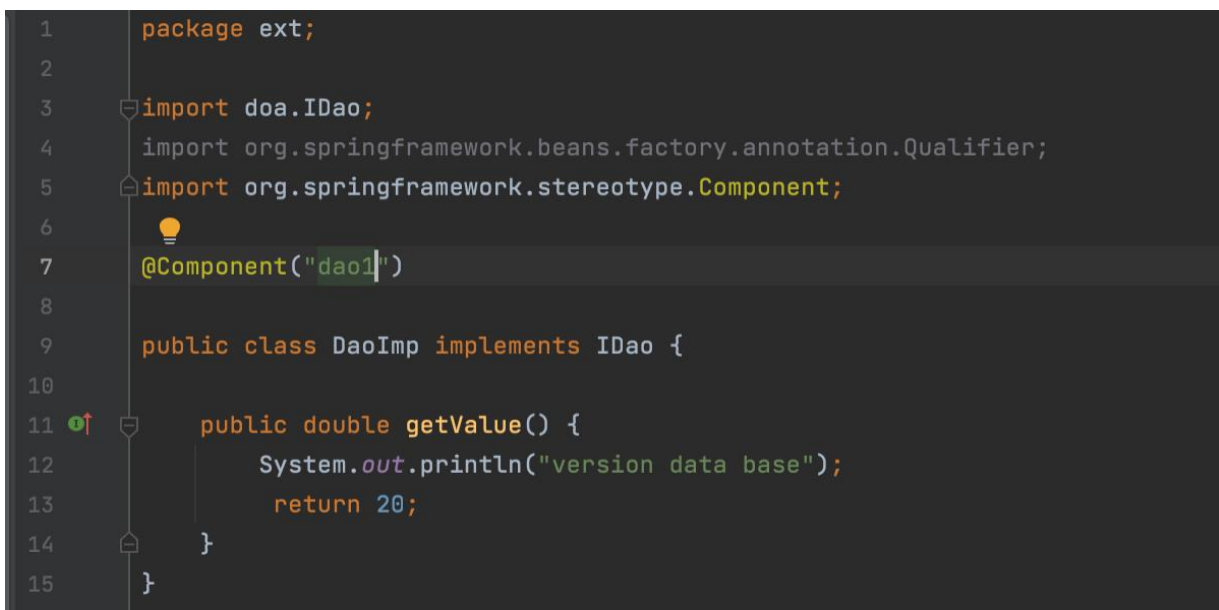
Interface IDao :



The screenshot shows an IDE with a project structure on the left and a code editor on the right. The project structure includes a 'src' directory with a 'dao' subdirectory containing 'DaoImp' and 'IDao'. The code editor displays the definition of the 'IDao' interface:

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

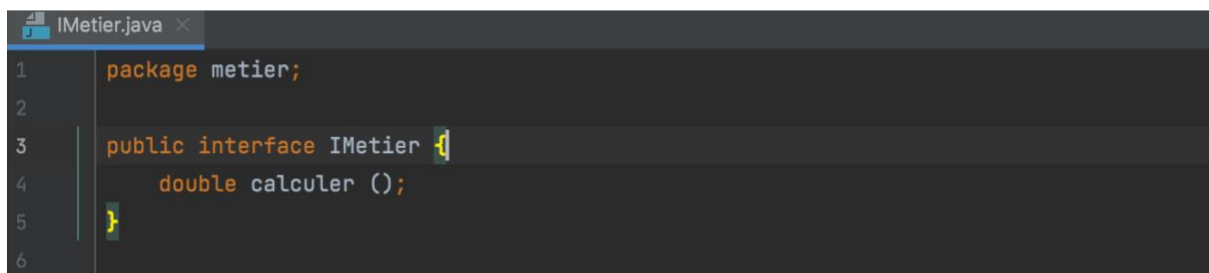
Une Implémentation de l'interface IDao :



The screenshot shows an IDE with a code editor displaying the implementation of the 'IDao' interface in the 'DaoImp' class. The code includes package declarations, imports for 'IDao', 'Qualifier', and 'Component', and the implementation of the 'getValue()' method:

```
package ext;  
  
import doa.IDao;  
import org.springframework.beans.factory.annotation.Qualifier;  
import org.springframework.stereotype.Component;  
  
@Component("dao1")  
  
public class DaoImp implements IDao {  
  
    public double getValue() {  
        System.out.println("version data base");  
        return 20;  
    }  
  
}
```

Interface metier



The screenshot shows an IDE with a code editor displaying the definition of the 'IMetier' interface in a file named 'IMetier.java':

```
package metier;  
  
public interface IMetier {  
  
    double calculer ();  
  
}
```

La première Implémentation de l'interface metier :

```
@Component
public class MetierImp implements IMetier{
    @Autowired
    @Qualifier("dao")
    IDao dao;

    //    public MetierImp(IDao dao) {
    //        this.dao = dao;
    //    }

    public MetierImp(){}

    public double calculer() { return dao.getValue()*5/Math.sin(15); }
    public void setDao(IDao dao) { this.dao = dao; }
}
```

Une deuxième implémentation des interfaces :
Extension de DaoImpl :

```
1 package ext;
2
3 import doa.IDao;
4 import org.springframework.stereotype.Component;
5
6 @Component("dao1")
7
8 public class DaoImp implements IDao {
9
10     public double getValue() {
11         System.out.println("version data base");
12         return 20;
13     }
14 }
15
```


Extension du classe MetierImp :

```
1 package ext;
2
3 import doa.IDao;
4 import metier.IMetier;
5
6 public class MetierImp implements IMetier {
7     private IDao dao;
8
9     public double calculer(){
10         return dao.getValue()*5/Math.sin(15);
11     }
12     public void setDao(IDao dao) { this.dao = dao; }
13 }
14
```

Injection des dépendances avec Spring Version XML.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:util="http://www.springframework.org/schema
4       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/
5       <bean id="d" class="doa.DaoImp"></bean>
6       <!--<bean id="metier" class="metier.MetierImp">-->
7       <!-- <constructor-arg name="dao" ref="d" />-->
8       <!--</bean>-->
9       <bean id="metier" class="metier.MetierImp">
10         <property name="dao" ref="d"/>
11       </bean>
12 </beans>
```

```

package Presentation;

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

public class PresentationVersionXml {
    public static void main(String[] args) {
        ApplicationContext context = new ClassPathXmlApplicationContext( "config.xml");
        IMetier metier = (IMetier) context.getBean( s: "metier");
        System.out.println(metier.calculer());
    }
}

```

- Après le démarrage de l'application Spring deux Objets seront créés : un objet de type dao.DaoImp avec un id « d » et un autre objet metier.MetierImp avec un id metier. La spécification de l'id va nous servir lorsqu'on veut invoquer l'objet.
- Pour récupérer les objets créés on appelle la méthode getBean() de l'objet ApplicationContext.

Injection des dépendances avec Spring Version Annotation :

Classe dao.Dao

```

import org.springframework.stereotype.Component;

@Component("dao")
public class DaoImp implements IDao{
    @Override
    public double getValue() {
        System.out.println("version capture");
        return 10;
    }
}

```

Classe ext.DaoImp

```
import doa.IDao;
import org.springframework.stereotype.Component;

@Component("dao1")

public class DaoImp implements IDao {

    public double getValue() {
        System.out.println("version data base");
        return 20;
    }
}
```

Classe metier.Metier

```
@Component
public class MetierImp implements IMetier{

    @Autowired
    @Qualifier("dao")
    IDao dao;

    // public MetierImp(IDao dao) {
    //     this.dao = dao;
    // }

    public double calculer() { return dao.getValue()*5/Math.sin(15); }
    public void setDao(IDao dao) { this.dao = dao; }
}
```

Classe presentation :

```
import org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class PresentationAnnotation {

    public static void main(String[] args) {
        ApplicationContext ctx = new AnnotationConfigApplicationContext( ...basePackages: "doa","metier","ext");
        IMetier metier =ctx.getBean(IMetier.class);
        System.out.println(metier.calculer());
    }
}
```

