Compte Mini Projet Framework :
D'Injection des dépendances

Réalisé Par : AYOUJJIL Soukayna
Département : Mathématiques et Informatique
Filière : II-BDCC
Professeur : M. Mohamed YOUSSFI

Ennoncé

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)

Couche Dao :

```
package dao;

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

```
package dao;

import Annotation.AnComponent;
@AnComponent
public class DaoImpl implements IDao {
    @Override
    public double getData() {
        System.out.println("version base de données");
        double temp=Math.random()*40;
        return temp;
    }
}
```

Couche Metier :

```
package metier;

public interface Imetier {
     double calcule();
}
```

```java
package metier;
import Annotation.AnAutowired;
import Annotation.AnComponent;
import dao.IDao;
@AnComponent
public class ImetierImpl implements Imetier {
    @AnAutowired
    private IDao dao;
    @Override
    public double calcule() {
        double tmp= dao.getData();
        double res=tmp*540/Math.cos(tmp*Math.PI);
        return res;
    }
    public void setDao(IDao dao) {
        this.dao = dao;
    }
}
```

1- Injection des dépendances via XML :

- configurationXML.xml

```xml
<framework>
    <dao>dao.DaoImpl</dao>
    <metier>metier.ImetierImpl</metier>
</framework>
```

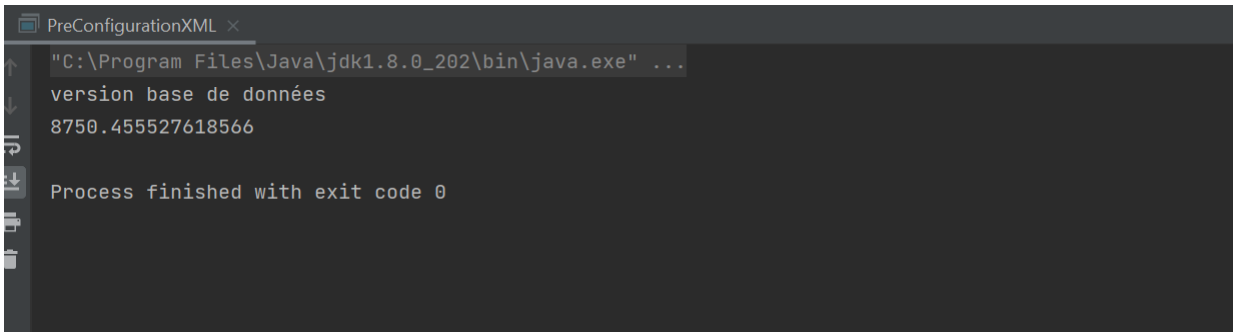
- ConfigurationXML.java

```java
package configurationXML;
import dao.IDao;
import metier.Imetier;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;
import javax.xml.XMLConstants;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import java.io.File;
import java.io.IOException;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;
public class ConfigurationXML {
    private String fileName;
    public ConfigurationXML(String nomfile) {
        this.fileName = nomfile;
    }
    public String getClassDao() throws ParserConfigurationException,
IOException, SAXException {
        DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
        dbf.setFeature(XMLConstants.FEATURE_SECURE_PROCESSING, true);
        DocumentBuilder db = dbf.newDocumentBuilder();
        Document doc = db.parse(new File(fileName));
        doc.getDocumentElement().normalize();
```

```
        NodeList list = doc.getElementsByTagName("framework");
        String firstname = null;
        for (int temp = 0; temp < list.getLength(); temp++) {
            Node node = list.item(temp);
            if (node.getNodeType() == Node.ELEMENT_NODE) {
                Element element = (Element) node;
                firstname =
element.getElementsByTagName("dao").item(0).getTextContent();
            }
        }
        return firstname;
    }

    public String getClassMetier() throws ParserConfigurationException,
IOException, SAXException {
        DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
        dbf.setFeature(XMLConstants.FEATURE_SECURE_PROCESSING, true);
        DocumentBuilder db = dbf.newDocumentBuilder();
        Document doc = db.parse(new File(fileName));
        doc.getDocumentElement().normalize();
        NodeList list = doc.getElementsByTagName("framework");
        String metier = null;
        for (int temp = 0; temp < list.getLength(); temp++) {
            Node node = list.item(temp);
            if (node.getNodeType() == Node.ELEMENT_NODE) {
                Element element = (Element) node;
                metier =
element.getElementsByTagName("metier").item(0).getTextContent();
            }
        }
        return metier;
    }
public Imetier getClasse() throws InstantiationException,
IllegalAccessException, ParserConfigurationException, IOException,
SAXException, ClassNotFoundException, NoSuchMethodException,
InvocationTargetException {
    Class cDao=Class.forName(getClassDao());
    IDao dao=(IDao) cDao.newInstance();
    Class cmetier=Class.forName(getClassMetier());
    Imetier metier= (Imetier) cmetier.newInstance();
    Method method=cmetier.getMethod("setDao",IDao.class);
    method.invoke(metier,dao);
    return  metier;
}
}
```

- PreConfigurationXML.java

```
public class PreConfigurationXML {
    public static void main(String[] args) throws ParserConfigurationException, IOException, SAXException,
        ConfigurationXML classInst=new ConfigurationXML( nomfile: "configurationXML.xml");
        System.out.println(classInst.getClasse().calcule());
    }
}
```



2- Injection des dépendances en utilisant les annotations :

- Annotation : AnAutowired

```java
import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;
@Target({ElementType.METHOD, ElementType.CONSTRUCTOR, ElementType.FIELD})
@Retention(RetentionPolicy.RUNTIME)
public @interface AnAutowired {
}
```

- Annotation : AnComponent

```java
import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
public @interface AnComponent {
}
```

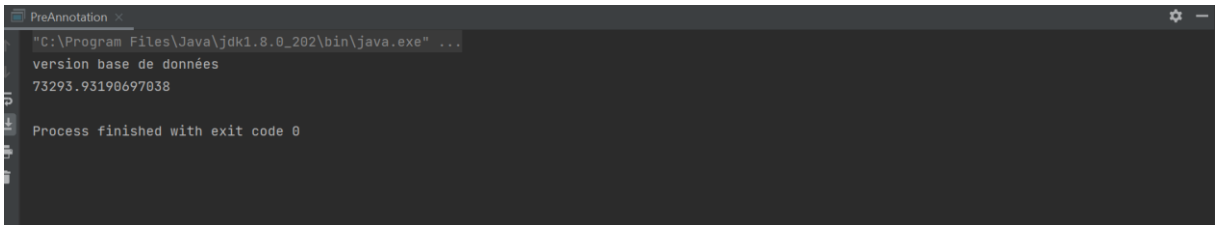
- Classe : ConfigurationAnnotation

```java
package annotation;

import org.reflections.Reflections;
import org.reflections.scanners.ResourcesScanner;
import org.reflections.scanners.SubTypesScanner;
import org.reflections.util.ClasspathHelper;
import org.reflections.util.ConfigurationBuilder;
import org.reflections.util.FilterBuilder;
import java.lang.reflect.Field;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Set;

public class ConfigurationAnnotation {
    HashMap<Class, Object> instances=new HashMap<Class, Object>();
    public void getClasses(String... packages) throws
```

```
InstantiationException, IllegalAccessException, NoSuchMethodException,
SecurityException, IllegalArgumentException, InvocationTargetException {
        ArrayList<Class> classes=new ArrayList<Class>();
        Set<Class<?>> subTypesOf=null;
        for(String packageName : packages) {
            Reflections reflections = new Reflections(new
ConfigurationBuilder()
                    .setScanners(new SubTypesScanner(false /* don't exclude
Object.class */), new ResourcesScanner())
                    .addUrls(ClasspathHelper.forJavaClassPath())
                    .filterInputsBy(new FilterBuilder()
                            .include(FilterBuilder.prefix(packageName))));

            subTypesOf = reflections.getSubTypesOf(Object.class);
            for( Class c :subTypesOf) {
                if(c.toString().contains("class")) {
                    Object o = c.newInstance();
                    instances.put(c.getInterfaces()[0], o);
                    classes.add(c);
                }
            }
        }
        for(Class c : classes) {
            if( c.getAnnotations()[0].toString().contains("AnComponent") &&
c.getDeclaredFields().length>0 ) {
                Field[] fields =c.getDeclaredFields();
                for(Field f : fields) {

if(f.getAnnotations()[0].toString().contains("AnAutowired"))
                    {
                        Method method=c.getMethod("setDao",f.getType());
                        method.invoke(instances.get(c.getInterfaces()[0]),
instances.get(f.getType()));
                    }
                }
            }
        }

    }
    public HashMap<Class, Object> getInstances(){
        return instances;
    }
}
```

- Classe : PreAnnotation

```
package annotation;

import metier.Imetier;

import java.lang.reflect.InvocationTargetException;
public class PreAnnotation {
    public static void main(String[] args) throws InvocationTargetException, InstantiationException, Illega
    ConfigurationAnnotation context=new ConfigurationAnnotation();
    context.getClasses( ...packages: "dao", "metier");
    Imetier imetier= (Imetier) context.getInstances().get(Imetier.class);
    System.out.println(imetier.calcule());
}}
```

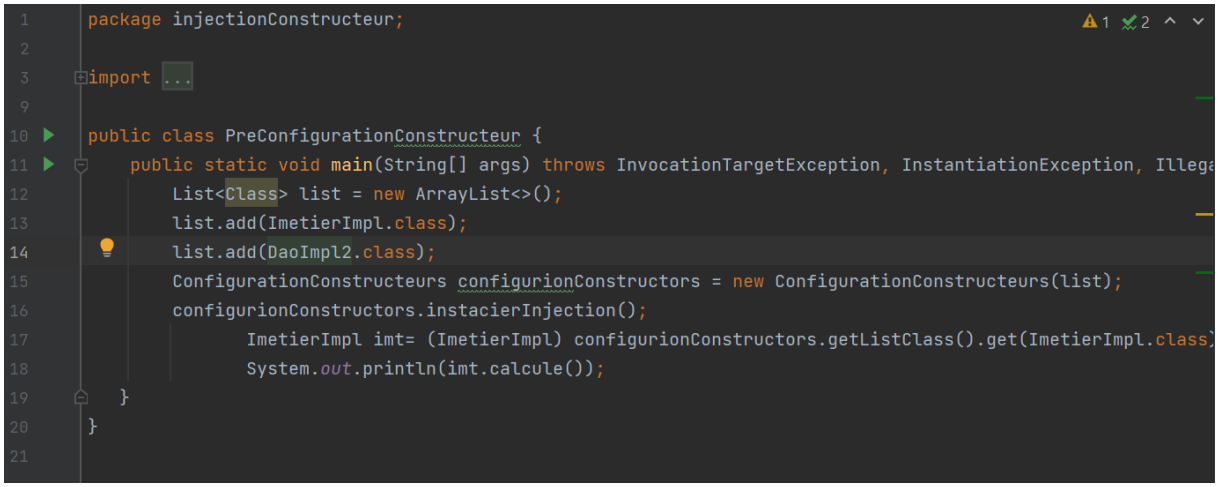


3- Injection des dépendances via constructeur:

- Classe : ConfigurationConstructeurs

```
   package injectionConstructeur;
import java.lang.reflect.Field;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class ConfigurationConstructeurs {
   private Map<Class,Object> listClass= new HashMap<Class,Object>();
   private List<Class> listClasse=new ArrayList<>();
    public ConfigurationConstructeurs(List<Class> listClasse) {
        this.listClasse= listClasse;
    }
    public Object getInstance(Class r) throws InstantiationException,
IllegalAccessException {
        for (Class ce:listClass.keySet()) {
          if (ce.getInterfaces()[0].toString().equals(r.toString())){
              return listClass.get(ce);
          }
        }
        return null;
    }
    public  void instacierInjection() throws InstantiationException,
IllegalAccessException, NoSuchMethodException, InvocationTargetException {
        for (Class c:listClasse) {
            listClass.put(c,c.newInstance());
        }
        for (Class c:listClasse) {
            if (c.getDeclaredFields()!=null){
                for (Field f:c.getDeclaredFields()) {
                    if ( f.getType().toString().contains("i")){
                        String methodName="setDao";
                        Method method=c.getMethod(methodName,f.getType());
                        method.invoke(listClass.get(c),
getInstance(f.getType()));

                    }
                }
            }
        }
        }
    public Map<Class, Object> getListClass() {
        return listClass;
    }
}
```

- Classe : PreConfigurationConstructeur



```
package injectionConstructeur;

import ...

public class PreConfigurationConstructeur {
    public static void main(String[] args) throws InvocationTargetException, InstantiationException, Illega
        List<Class> list = new ArrayList<>();
        list.add(ImetierImpl.class);
        list.add(DaoImpl2.class);
        ConfigurationConstructeurs configurionConstructors = new ConfigurationConstructeurs(list);
        configurionConstructors.instacierInjection();
              ImetierImpl imt= (ImetierImpl) configurionConstructors.getListClass().get(ImetierImpl.class)
              System.out.println(imt.calcule());
    }
}
```