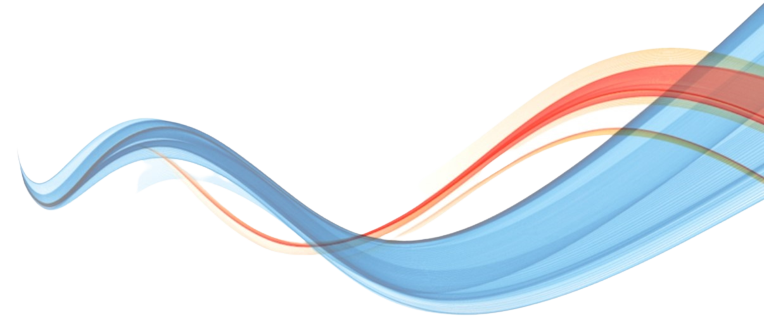
Compte Mini Projet Framework :

D’Injection des dépendances



**Année Universitaire** **: 2021/2022**

**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

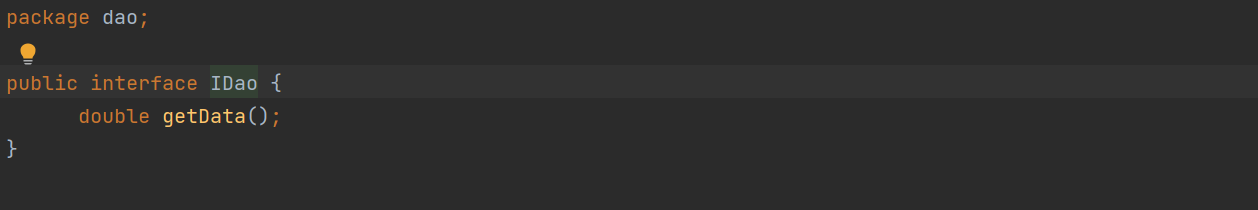
3- Possibilité d'injection via :

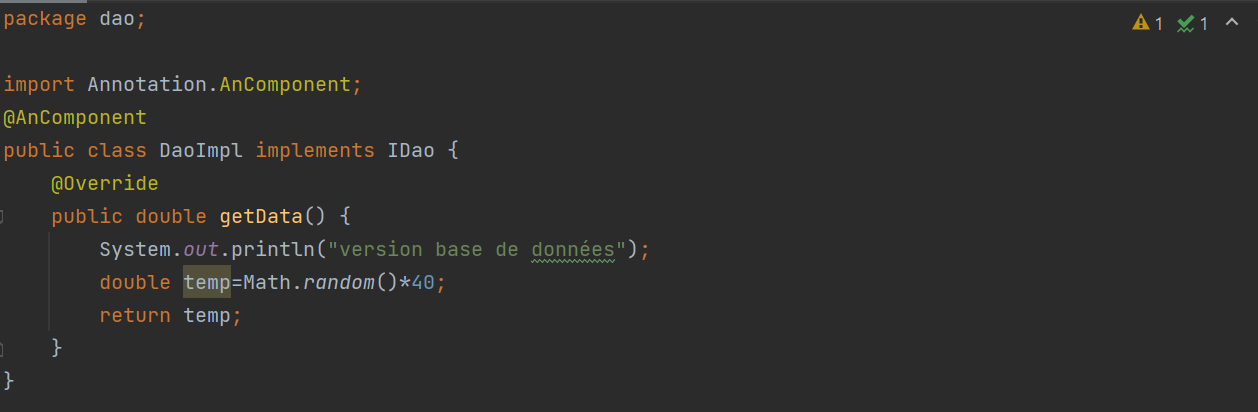
a- Le constructeur

b- Le Setter

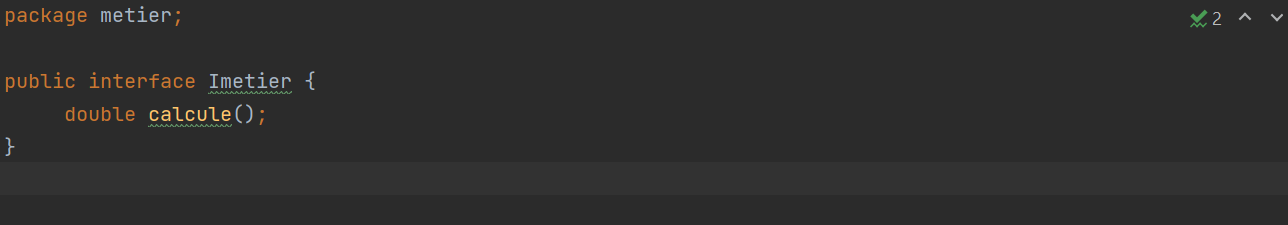
c- Attribut (accès direct à l'attribut : Field)

Couche Dao :



****

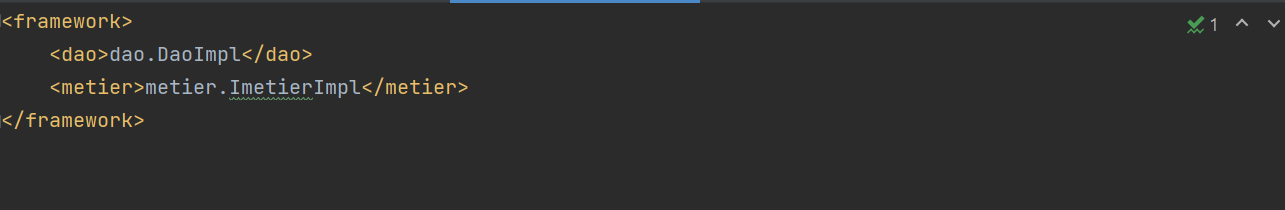
**Couche Metier :**



****

1. **Injection des dépendances via XML :**

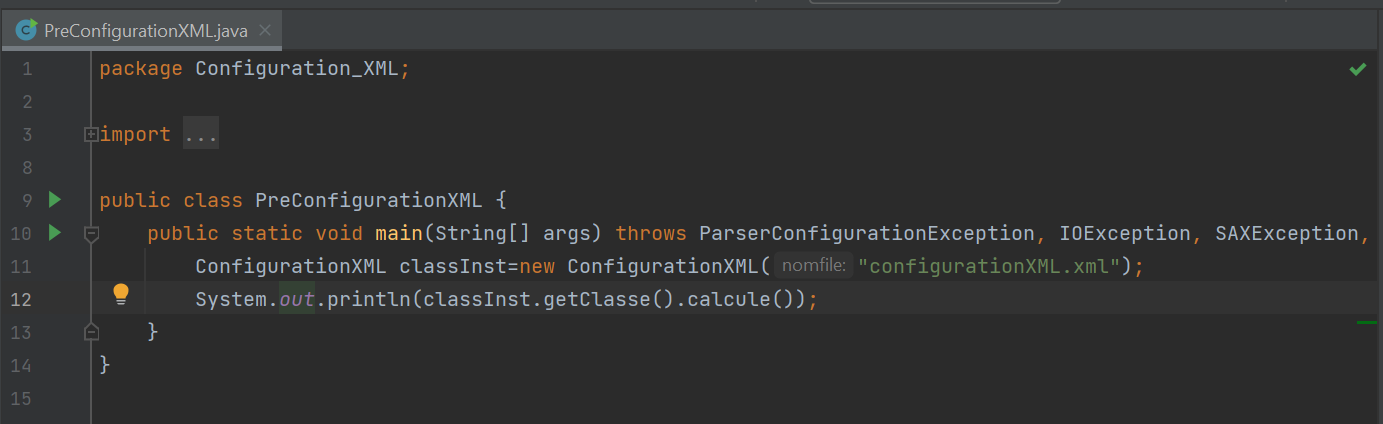
* configurationXML.xml

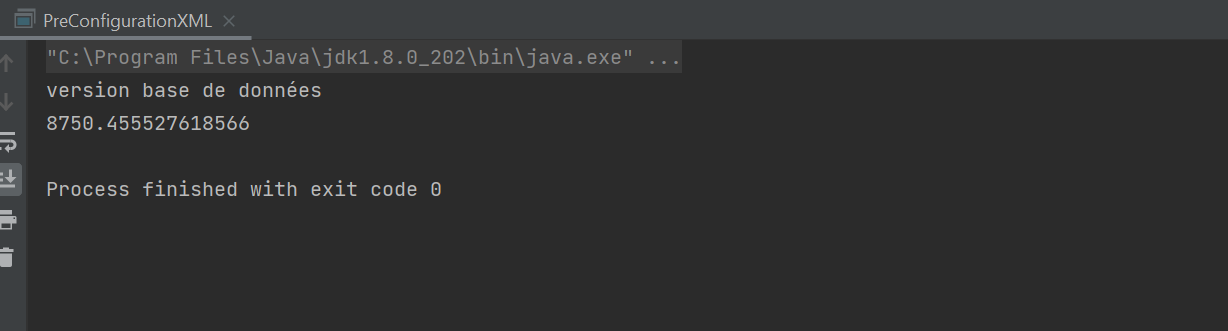
****

* ConfigurationXML.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;  
}  
}

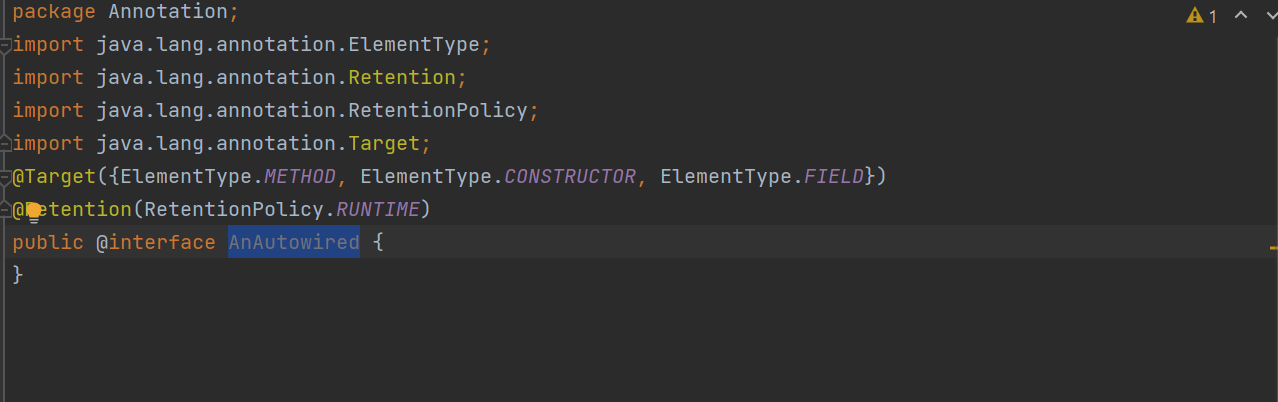
* PreConfigurationXML.java



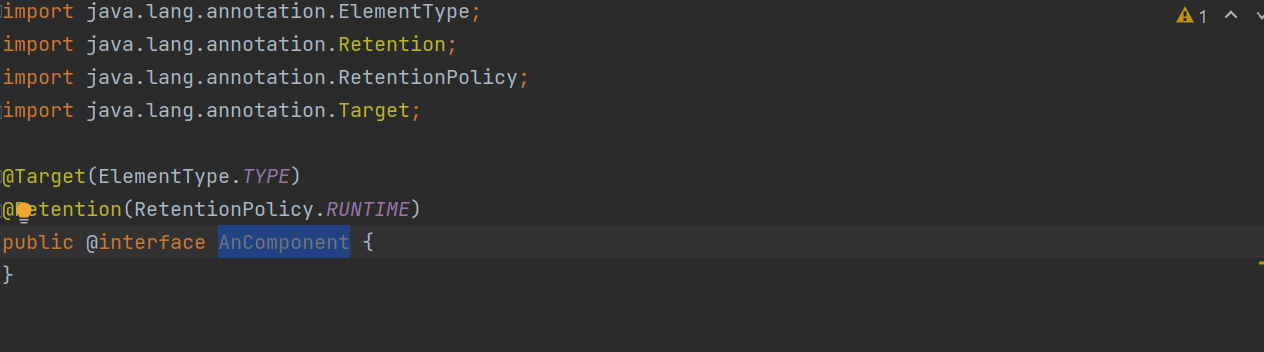


1. **Injection des dépendances en utilisant les annotations :**

* Annotation : AnAutowired



* Annotation : AnComponent

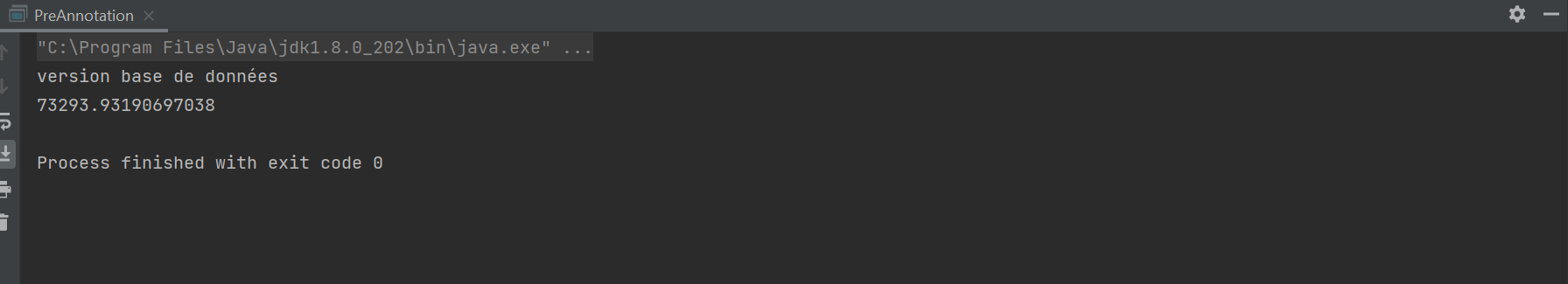


* Classe : ConfigurationAnnotation

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





1. **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

