

# Mini Projet Framework d'Injection des dépendances



Réalisé par :

Mohamed amine

KHAMMOUR

Professeur :

Mohamed

ELYOUSSFI

## Enonce du travaille



# Mini Projet Framework d'Injection des dépendances

Mohamed YOUSSEFI • 26 févr.

100 points

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)

### Question 1 : injection des dépendances via XML

Dans cette partie j'ai travaillé avec DOM (Document Object Model)

Voila le fichier XML

```
<fremwoerk>
  <dao>Ddao.DaoImpl</dao>
  <metier>metier.ImetierImpl</metier>
</fremwoerk>
```

La classe de la configuration :

```
public class ConfigurationXML {
    private String nomfile;
    public ConfigurationXML(String nomfile) {
        this.nomfile = nomfile;
    }
    public String getNomfile() { return nomfile; }
    public void setNomfile(String nomfile) { this.nomfile = nomfile; }
    public String getClassDao() throws ParserConfigurationException, IOException, SAXException {
        DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();

        dbf.setFeature(XMLConstants.FEATURE_SECURE_PROCESSING, value: true);

        DocumentBuilder db = dbf.newDocumentBuilder();

        Document doc = db.parse(new File(nomfile));
        doc.getDocumentElement().normalize();
        NodeList list = doc.getElementsByTagName("fremwoerk");
        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( index: 0).getTextContent();
    }
}
return firstname;
}

public String gestClassMetier() throws ParserConfigurationException, IOException, SAXException {
    DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();

    dbf.setFeature(XMLConstants.FEATURE_SECURE_PROCESSING, value: true);

    DocumentBuilder db = dbf.newDocumentBuilder();

    Document doc = db.parse(new File(nomfile));

    doc.getDocumentElement().normalize();
    NodeList list = doc.getElementsByTagName("fremwoerk");
    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( index: 0).getTextContent();
        }
    }
    return metier;
}

public Imetier getClasse() throws InstantiationException, IllegalAccessException, ParserConfigurationException, IOE
    Class cDao=Class.forName(gestClassDao());
    IDao dao=(IDao) cDao.newInstance();
    Class cmetier=Class.forName(gestClassMetier());
    Imetier metier= (Imetier) cmetier.newInstance();
    Method method=cmetier.getMethod( name: "setDao",IDao.class);
    method.invoke(metier,dao);
    return metier;
}
}

```

La class présentation :

```

package InjectionXML;

import ...

public class presentation {
    public static void main(String[] args) throws ParserConfigurationException, IOException, SAXException, ClassNotFoundException {
        ConfigurationXML fremworkClass=new ConfigurationXML( nomfile: "config.xml");
        System.out.println(fremworkClass.getClasse().calculer());
    }
}

```

La class DaoImp qui implémente la class l'interface IDao :

```

package Ddao;

import InjectionAnotation.AmComponent;

@AmComponent
public class DaoImpl implements IDao {
    @Override
    public double getData() {
        //connexion base de donne pour recupirer la temperature
        System.out.println("version base de donnees");
        return Math.random()*40;
    }
}

```

La MetierImp qui implémente l'interface Imetier :

```

package metier;

import Ddao.IDao;
import InjectionAnotation.AmAutowired;
import InjectionAnotation.AmComponent;

@AmComponent
public class ImetierImpl implements Imetier {
    @Autowired
    private IDao dao; //couplage faible
    @Override
    public double calculer() { return dao.getData()*540/Math.cos(Math.PI*40); }
    //injecter dans la variable dao un objet d'une clas qui implement l'interface dao
    public void setDao(IDao dao) { this.dao = dao; }
}

```

Résultat :

```
presentation x
E:\Java\bin\java.exe ...
version base de donnees
7880.50412731606
Process finished with exit code 0
|
```

L'injection via Annotation :

L'annotation AmAutowired :

```
package InjectionAnotation;

import ...

@Target({ElementType.METHOD, |
        ElementType.CONSTRUCTOR,
        ElementType.FIELD})
@Retention(RetentionPolicy.RUNTIME)
public @interface AmAutowired {

}
```

L'annotation AmComponent :

```
package InjectionAnotation;

import ...

@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
public @interface AmComponent {

}
```

La class ConfigurationAnotation :

```

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Set;

public class ConfigurationAnotation {
    HashMap<Class, Object> instances=new HashMap<Class, Object>();
    public void getClasses(String... packages) throws InstantiationException, IllegalAccessException, NoSuchMe
        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 ResourcesScal
                .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) {
        }
    }

    for(Class c : classes) {
        if( c.getAnnotations()[0].toString().contains("AmComponent") && c.getDeclaredFields().length>0 ) {
            Field[] fields =c.getDeclaredFields();
            for(Field f : fields) {
                if(f.getAnnotations()[0].toString().contains("AmAutowired"))
                {
                    Method method=c.getMethod( name: "setDao",f.getType());
                    method.invoke(instances.get(c.getInterfaces()[0]), instances.get(f.getType()));
                }
            }
        }
    }
}

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

```

La class présentation de cette partie :

```

package InjectionAnotation;

import metier.Imetier;

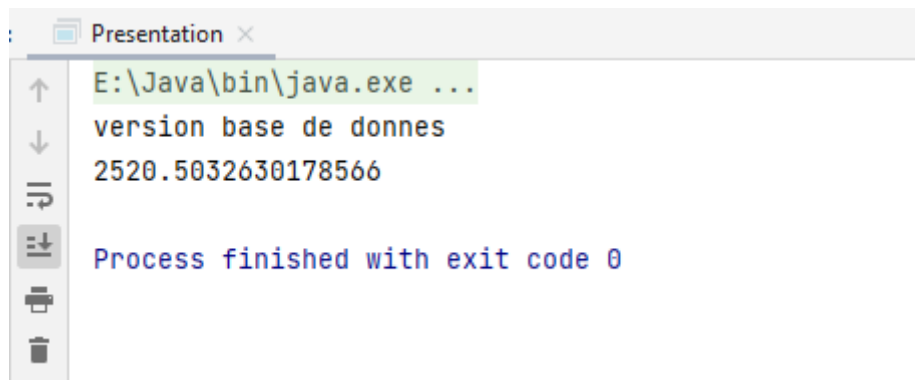
import java.lang.reflect.InvocationTargetException;

public class Presentation {
    public static void main(String[] args) throws InvocationTargetException, 1
        ConfigurationAnotattion contx=new ConfigurationAnotattion();
        contx.getClasses( ...packages: "Ddao", "metier");
        Imetier imetier= (Imetier) contx.getInstances().get(Imetier.class);

        System.out.println(imetier.calcule());
    }}

```

Résultat :



```

Presentation x
E:\Java\bin\java.exe ...
version base de donnes
2520.5032630178566
Process finished with exit code 0

```

Injection via Constructor :

La classe ConfigurionConstructors :

```

public class ConfigurionConstructors {
    private Map<Class, Object> listClass= new HashMap<Class, Object>();
    private List<Class> listClasse=new ArrayList<>();

    public ConfigurionConstructors(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())){
                System.out.println(ce.getInterfaces()[0].toString());
                return listClass.get(ce);
            }
        }
        return null;
    }
}

```

```

public void instancierInjection() throws InstantiationException, IllegalAccessException, NoSuchMethodException,
    for (Class c: listClasse) {
        listClass.put(c, c.newInstance());
    }
    for (Class c: listClasse) {
        if (c.getDeclaredFields() != null) {
            for (Field f: c.getDeclaredFields()) {
                System.out.println(f.getType().toString());
                if (f.getType().toString().contains("i")) {
                    String methodName = "setDao";
                    System.out.println(methodName);
                    System.out.println(f.getClass());
                    Method method = c.getMethod(methodName, f.getType());
                    method.invoke(listClass.get(c), getInstance(f.getType()));
                }
            }
        }
    }
}

```

La class présentation :

```

public class Presentation {
    public static void main(String[] args) throws InvocationTargetException, InstantiationException, IllegalAccessException,
        List<Class> list = new ArrayList<>();
        list.add(ImetierImpl.class);
        list.add(DaoImpl2.class);
        ConfigurionConstructors configurionConstructors = new ConfigurionConstructors(list);
        configurionConstructors.instancierInjection();
        ImetierImpl imt = (ImetierImpl) configurionConstructors.getListClass().get(ImetierImpl.class);
        System.out.println(imt.calculer());
    }
}

```

Le résultat :

```

Presentation x
E:\Java\bin\java.exe ...
interface Ddao.IDao
version capture
2160000.0

Process finished with exit code 0

```