



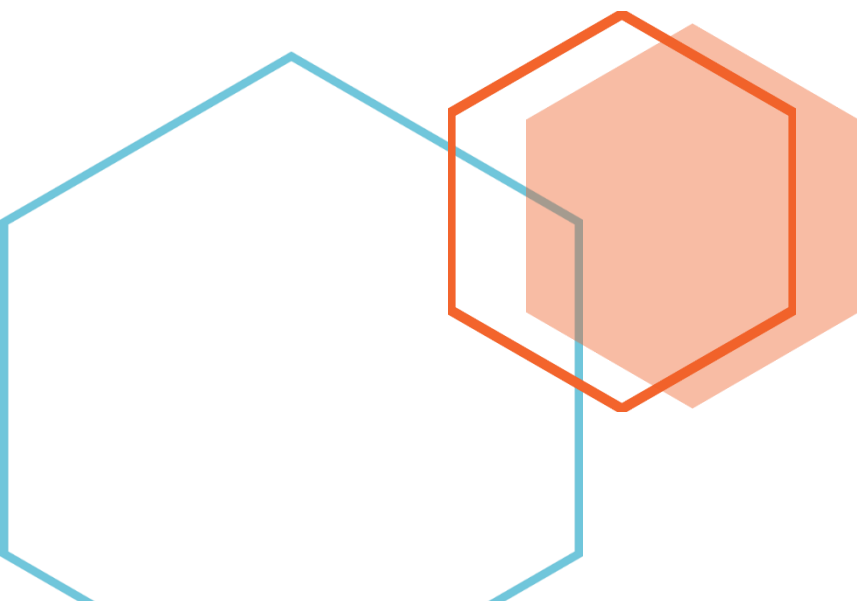
Amina Dahmouni

Classe : II-BDCC2

[Compte Rendu]

Mini Projet Framework d'Injection des
dépendances

Architecture JEE et Middlewares
L'Ecole Normale Supérieure de l'Enseignement
Technique de Mohammedia (ENSET)



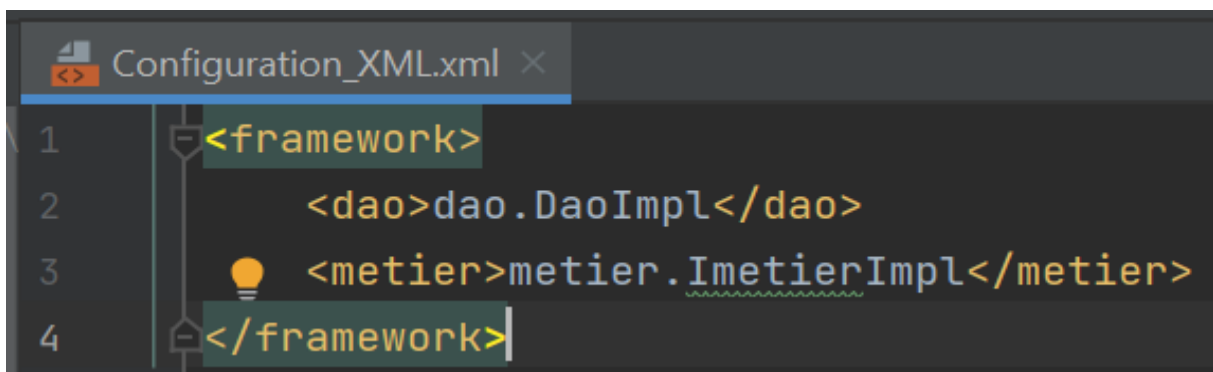
Concevoir et créer un mini Framework d'injection des dépendances similaire à Spring IOC

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)

1. A travers un fichier XML de configuration en utilisant Jax Binding (OXM : Mapping Objet XML)

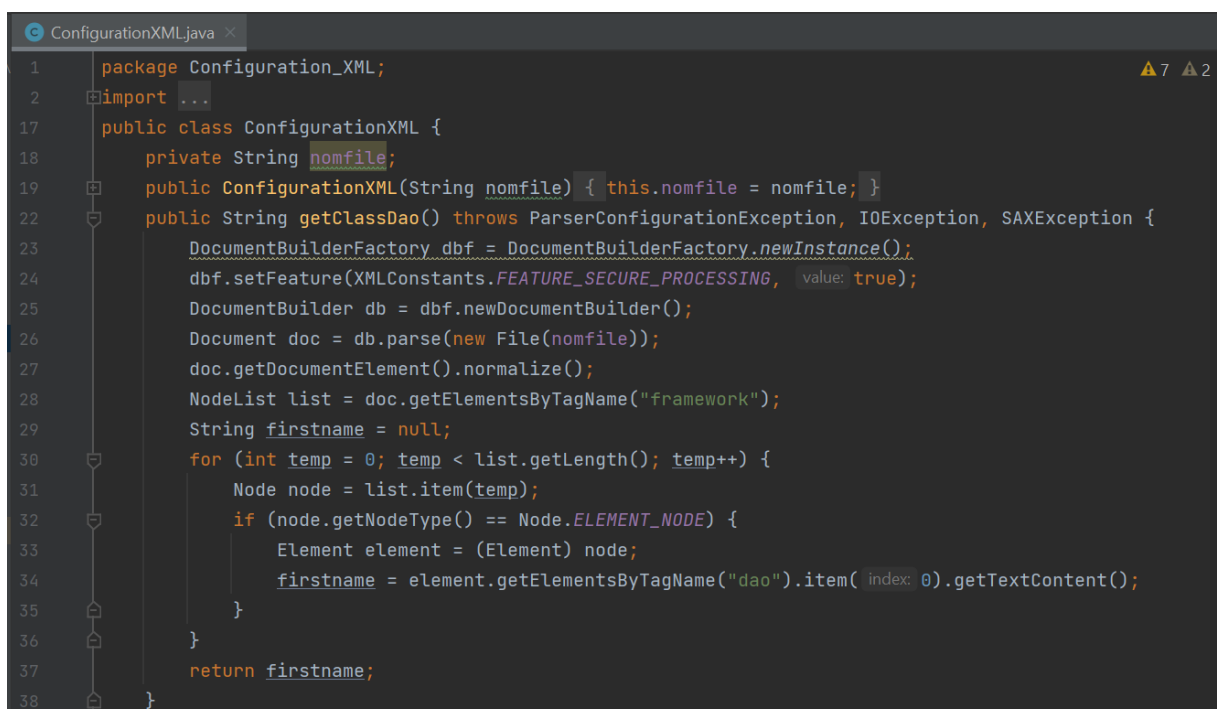
- Le fichier XML :



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

- La classe de la configuration :

Pour retourner le nom de la classe dao



```
package Configuration_XML;
import ...
public class ConfigurationXML {
    private String nomfile;
    public ConfigurationXML(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("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( index: 0).getTextContent();
            }
        }
        return firstname;
    }
}
```

Pour retourner le nom de la classe ImetierImpl

```
40 public String getClassMetier() throws ParserConfigurationException, IOException, SAXException {
41     DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
42     dbf.setFeature(XMLConstants.FEATURE_SECURE_PROCESSING, value: true);
43     DocumentBuilder db = dbf.newDocumentBuilder();
44     Document doc = db.parse(new File(nomfile));
45     doc.getDocumentElement().normalize();
46     NodeList list = doc.getElementsByTagName("framework");
47     String metier = null;
48     for (int temp = 0; temp < list.getLength(); temp++) {
49         Node node = list.item(temp);
50         if (node.getNodeType() == Node.ELEMENT_NODE) {
51             Element element = (Element) node;
52             metier = element.getElementsByTagName("metier").item(0).getTextContent();
53         }
54     }
55     return metier;
56 }
```

Charger la classe daoImpl dans la mémoire et faire l'injection des dépendances d'une manière dynamique.

```
57 public Imetier getClasse() throws InstantiationException, IllegalAccessException, ParserConfigurationException,
58     /**Charger la classe dans la mémoire*/
59     Class cDao=Class.forName(getClassDao());
60     /**Instancier la classe fait appel au constructeur par défaut
61     * newInstance() retourne une classe de type Object*/
62     IDao dao=(IDao) cDao.newInstance();
63     Class cmetier=Class.forName(getClassMetier());
64     Imetier metier= (Imetier) cmetier.newInstance();
65     Method method=cmetier.getMethod( name: "setDao", IDao.class);
66     /** Injection des dépendances d'une manière dynamique
67     * Executer la méthode*/
68     method.invoke(metier,dao);
69     return metier;
70 }
71 }
```

- La class DaoImpl:

```
1 package dao;
2
3 import InjectionAnotation.AmComponent;
4 @AmComponent
5 public class DaoImpl implements IDao {
6     @Override
7     public double getData() {
8         System.out.println("DaoImpl Version 1");
9         double temp=Math.random()*40;
10        return temp;
11    }
12 }
```

- La class MetierImpl :

```

1  package metier;
2  import ...
5  @Annotation_Component
6  public class ImetierImpl implements Imetier {
7      /**@Annotation_Autowired pour faire l'injection des dépendances
8       * on demande au framework au moment qu'on il instancie la classe
9       * MetierImpl de chercher parmi les objets qu'il a crée un objet
10      * de type IDao et l'injecter dans la variable dao
11      */
12      @Annotation_Autowired
13      private IDao dao;
14      @Override
15      public double calcule() {
16          double tmp= dao.getData();
17          double res=tmp*540/Math.cos(tmp*Math.PI);
18          return res;
19      }
20      //affecter une valeur à dao
21      /** Injecter dans la variable dao un objet d'une
22       * classe qu'implemente l'interface IDao
23       */
24      public void setDao(IDao dao) { this.dao = dao; }
27 }

```

- Présentation :

```

1  package Configuration_XML;
2
3  import ...
8
9  public class presentation {
10     public static void main(String[] args) throws ParserConfigurationException, IOException,
11         ConfigurationXML classInst=new ConfigurationXML( nomfile: "Configuration_XML.xml");
12         System.out.println(classInst.getClasse().calcule());
13     }
14 }
15

```

- Résultats :

```
presentation x
"C:\Program Files\Java\jdk1.8.0_321\bin\java.exe" ...
DaoImpl Version 1
-9184.201175120164
Process finished with exit code 0
```

2. En utilisant les annotations

- L'annotation Autowired pour l'injection des dépendances

```
@ Annotation_Autowired.java x
1 package Annotation;
2 import ...
6 /**
7  * @Target Pour créer une annotation personnalisée
8  * L'annotation peut être appliquée pour une méthode, constructeur et champ
9  */
11 @Target({ElementType.METHOD, ElementType.CONSTRUCTOR, ElementType.FIELD})
12 /**
13  * RetentionPolicy.RUNTIME : les annotations annotées à l'aide de la
14  * politique de rétention RUNTIME sont conservées pendant l'exécution et
15  * sont accessibles dans notre programme pendant l'exécution.
16  */
17 @Retention(RetentionPolicy.RUNTIME)
18 public @interface Annotation_Autowired {
19 }
```

- L'annotation Component :

```
@ Annotation_Component.java x
1 package Annotation;
2 import ...
6 /**
7  * @Component est une annotation au niveau de la classe.
8  * Il est utilisé pour désigner une classe en tant que composant.
9  * L'annotation peut être appliquée pour Class, interface or enumeration
10 */
11 @Target(ElementType.TYPE)
12 @Retention(RetentionPolicy.RUNTIME)
13 public @interface Annotation_Component {
14 }
```

- La classe de configuration :

```

Annotation_Configuration.java x Presentation.java x
3  import ...
15
16  public class Annotation_Configuration {
17      HashMap<Class, Object> instances=new HashMap<>();
18  @
19  public void getClasses(String... packages) throws InstantiationException, IllegalAccessException,
20      ArrayList<Class> classes=new ArrayList<>();
21      Set<Class<?>> subTypesOf=null;
22      for(String packageName : packages) {
23          Reflections reflections = new Reflections(new ConfigurationBuilder()
24              .setScanners(new SubTypesScanner(false /* don't exclude Object.class */), new Res
25              .addUrls(ClasspathHelper.forJavaClassPath())
26              .filterInputsBy(new FilterBuilder()
27                  .include(FilterBuilder.prefix(packageName))));
28
29          subTypesOf = reflections.getSubTypesOf(Object.class);
30          for( Class c :subTypesOf) {
31              if(c.toString().contains("class")) {
32                  Object o = c.newInstance();
33                  instances.put(c.getInterfaces()[0], o);
34                  classes.add(c);
35              }
36          }
37          for(Class c : classes) {
38              if( c.getAnnotations()[0].toString().contains("Annotation_Component") && c.getDeclaredFields().length>0
39              Field[] fields =c.getDeclaredFields();
40              for(Field f : fields) {
41                  if(f.getAnnotations()[0].toString().contains("Annotation_Autowired"))
42                  {
43                      Method method=c.getMethod( name: "setDao",f.getType());
44                      method.invoke(instances.get(c.getInterfaces()[0]), instances.get(f.getType()));
45                  }
46              }
47          }
48      }
49  }
50  }
51  public HashMap<Class, Object> getInstances(){ return instances;}
52  }

```

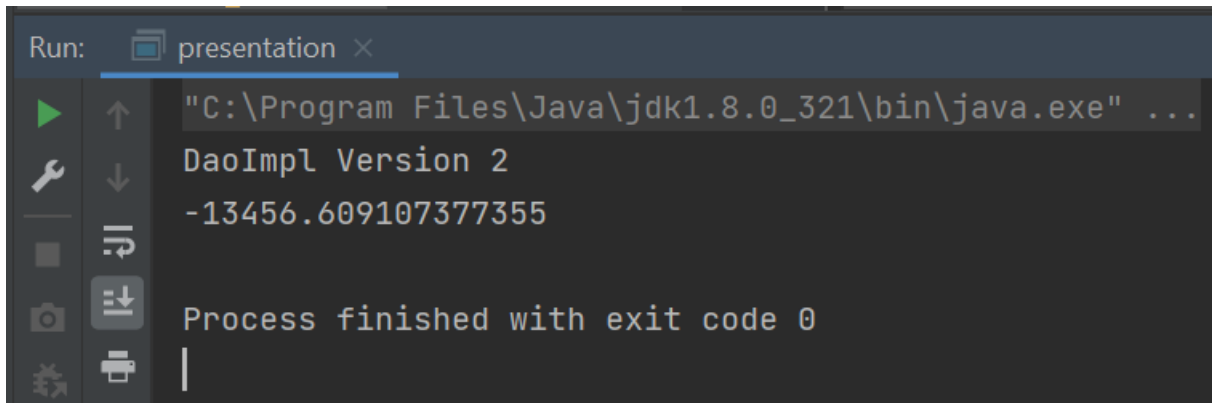
- Présentation :

```

Presentation.java x
1  package Annotation;
2
3  import ...
7  public class Presentation {
8  public static void main(String[] args) throws InvocationTargetException,
9      //Scanner les packages dao & metier
10     ConfigurationAnotation context=new ConfigurationAnotation();
11     context.getClasses( ...packages: "dao", "metier");
12     Imetier imetier= (Imetier) context.getInstances().get(Imetier.class);
13
14     System.out.println(imetier.calculer());
15 }

```

- Résultat :

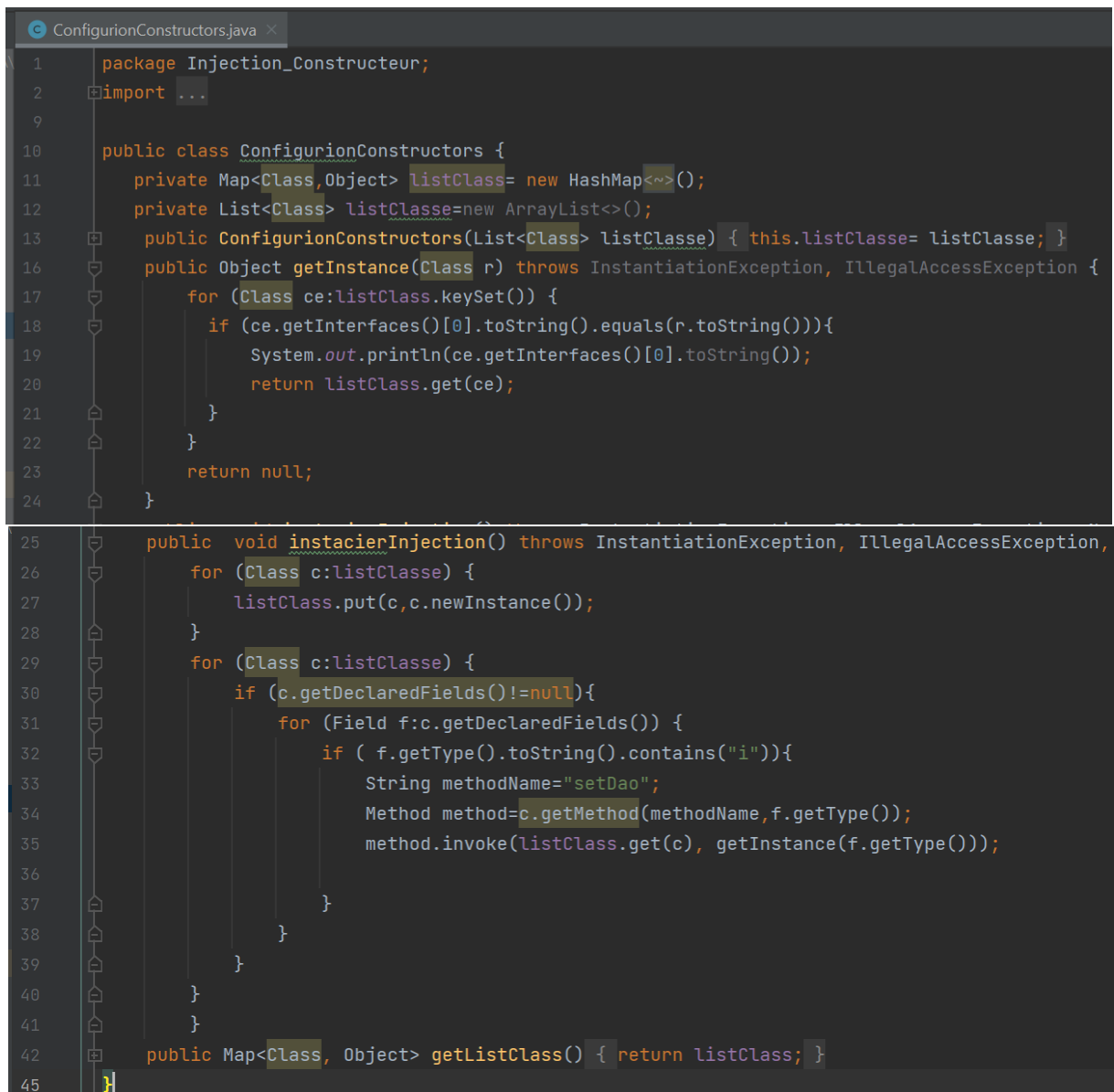


```
Run: presentation x
"C:\Program Files\Java\jdk1.8.0_321\bin\java.exe" ...
DaoImpl Version 2
-13456.609107377355
Process finished with exit code 0
```

3- Possibilité d'injection via :

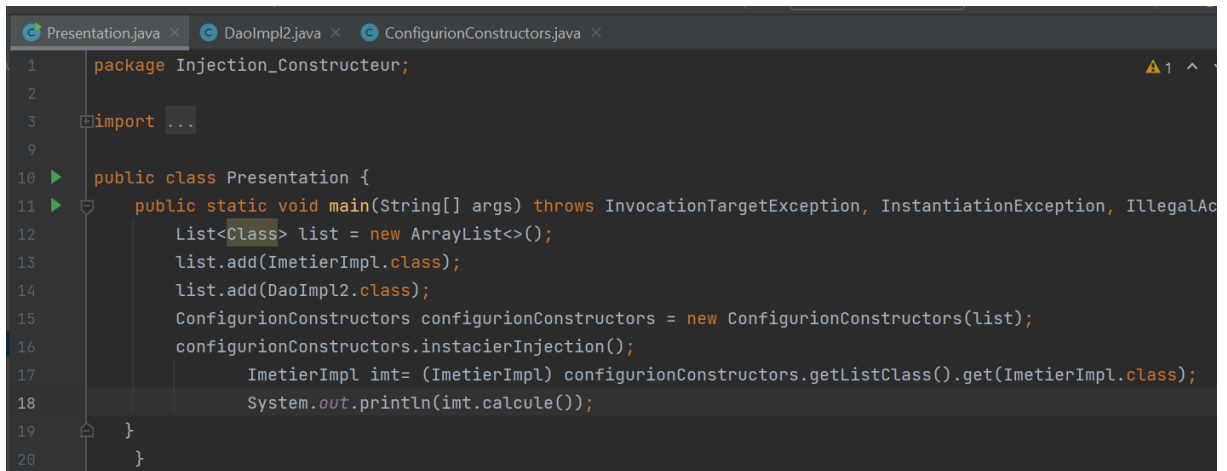
a- Le constructeur

- La classe de configuration :



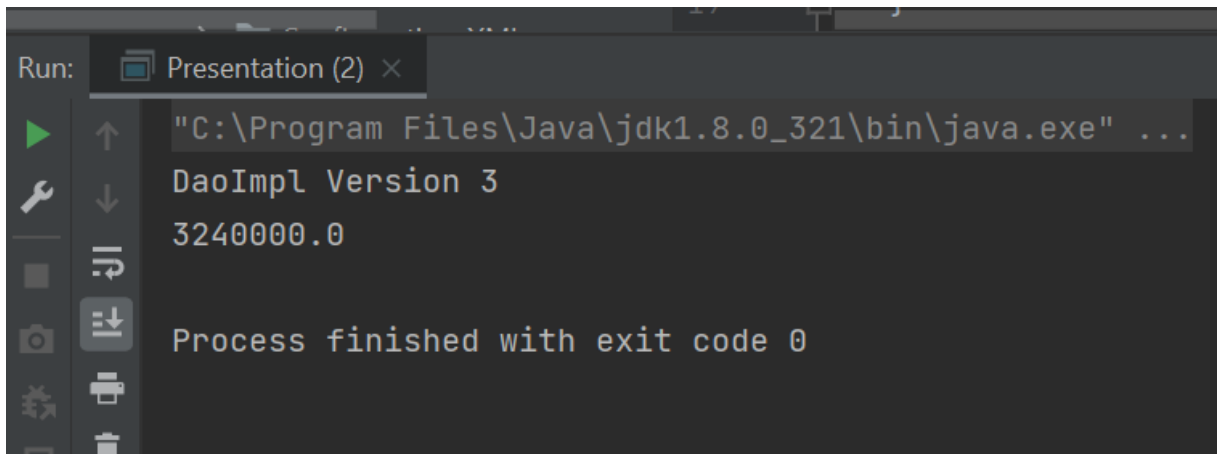
```
ConfigurationConstructeurs.java x
1 package Injection_Constructeur;
2 import ...
9
10 public class ConfigurationConstructeurs {
11     private Map<Class, Object> listClass= new HashMap<>();
12     private List<Class> listClasse=new ArrayList<>();
13     public ConfigurationConstructeurs(List<Class> listClasse) { this.listClasse= listClasse; }
16     public Object getInstance(Class r) throws InstantiationException, IllegalAccessException {
17         for (Class ce:listClass.keySet()) {
18             if (ce.getInterfaces()[0].toString().equals(r.toString())){
19                 System.out.println(ce.getInterfaces()[0].toString());
20                 return listClass.get(ce);
21             }
22         }
23         return null;
24     }
25     public void instancierInjection() throws InstantiationException, IllegalAccessException,
26         for (Class c:listClasse) {
27             listClass.put(c,c.newInstance());
28         }
29         for (Class c:listClasse) {
30             if (c.getDeclaredFields()!=null){
31                 for (Field f:c.getDeclaredFields()) {
32                     if ( f.getType().toString().contains("i")){
33                         String methodName="setDao";
34                         Method method=c.getMethod(methodName,f.getType());
35                         method.invoke(listClass.get(c), getInstance(f.getType()));
36                     }
37                 }
38             }
39         }
40     }
41     }
42     public Map<Class, Object> getListClass() { return listClass; }
45 }
```

- Présentation :



```
1 package Injection_Constructeur;
2
3 import ...
4
5
6
7
8
9
10 public class Presentation {
11     public static void main(String[] args) throws InvocationTargetException, InstantiationException, IllegalAccessException {
12         List<Class> list = new ArrayList<>();
13         list.add(ImetierImpl.class);
14         list.add(DaoImpl2.class);
15         ConfigurionConstructors configurionConstructors = new ConfigurionConstructors(list);
16         configurionConstructors.instacierInjection();
17         ImetierImpl imt= (ImetierImpl) configurionConstructors.getListClass().get(ImetierImpl.class);
18         System.out.println(imt.calculer());
19     }
20 }
```

- Résultat :



```
Run: Presentation (2) x
"C:\Program Files\Java\jdk1.8.0_321\bin\java.exe" ...
DaoImpl Version 3
3240000.0
Process finished with exit code 0
```