

# RAPPORT DU TP1

---

INVERSION DE CONTROLE ET INJECTION DE DEPENDANCES

Abderrahmane ET-Tounani

IIBDCC-2

2022-2023

# Introduction

L'inversion de contrôle (IoC) en informatique est un concept devenu incontournable dans le développement de logiciels et de frameworks modernes, ayant un impact significatif sur la qualité, la flexibilité et la maintenabilité des applications.

L'IoC est un patron d'architecture qui fonctionne selon le principe que le flux d'exécution d'un logiciel n'est plus sous le contrôle direct de l'application elle-même, mais sous celui du Framework ou de la couche logicielle sous-jacente. Cela signifie que le framework prend en charge l'exécution principale du programme et coordonne et contrôle les activités de l'application.

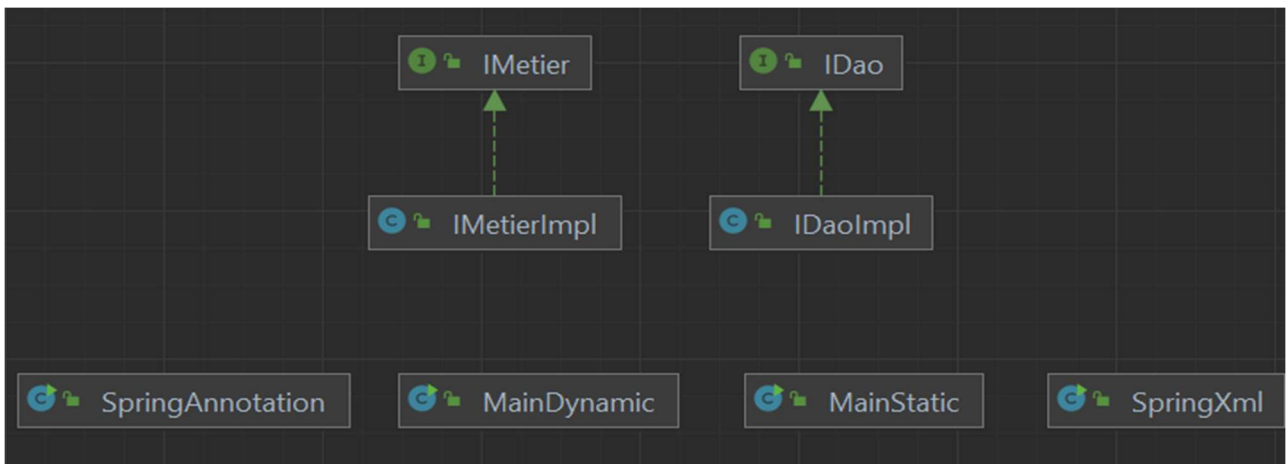
Le mécanisme le plus connu pour implémenter l'IoC est l'injection de dépendances. Il consiste à créer dynamiquement les dépendances entre les différents objets en utilisant une description ou de manière programmatique. Cela permet de découpler les dépendances entre objets et de les déterminer dynamiquement à l'exécution. Le principe d'Hollywood "Ne nous appelez pas, c'est nous qui vous appellerons" illustre bien l'IoC. Selon ce principe, c'est le framework ou la couche logicielle sous-jacente qui gère les appels à l'application et non l'inverse.

En plus d'améliorer la qualité, la flexibilité et la maintenabilité des applications, l'IoC présente également des avantages pour le développement en équipe. En permettant une séparation claire des responsabilités entre le framework et l'application, les développeurs peuvent travailler de manière plus efficace et indépendante. De plus, en utilisant des frameworks fiables et testés, les développeurs peuvent se concentrer sur la résolution des problèmes métiers sans se soucier de la gestion de certaines tâches techniques.

L'IoC est également utile pour faciliter les tests unitaires et les tests d'intégration. En utilisant l'injection de dépendances, les développeurs peuvent facilement remplacer les dépendances par des mock-ups pour les tests, ce qui rend les tests plus fiables et plus faciles à maintenir.

# Partie 1

## Conception



## Implémentation

**l'interface IDao**

```
1 package com.ettounani.Dao;
2 import java.util.Date;
3
4 public interface IDao {
5     Date getDate();
6 }
```

**une implémentation de l'interface IDao**

```
1 package com.ettounani.Dao;
2 import java.util.Date;
3
4 public class IDaoImpl implements IDao{
5     @Override
6     public Date getDate() {
7         return new Date();
8     }
9 }
```



## l'interface IMetier

```
1 package com.ettounani.Metier;
2
3 public interface IMetier {
4     double calcul();
5 }
```



## une implémentation de l'interface IMetie

```
1 package com.ettounani.Metier;
2 import com.ettounani.Dao.IDao;
3 public class IMetierImpl implements IMetier{
4     IDao dao;
5     // constructor
6     public IMetierImpl(IDao dao) {
7         this.dao = dao;
8     }
9     public IMetierImpl() {
10    }
11    @Override
12    public double calcul() {
13        return dao.getDate().getMinutes();
14    }
15    public void setDao(IDao dao) {
16        this.dao = dao;
17    }
18    public IDao getDao() {
19        return dao;
20    }
21 }
```



## Presentation Instanciation Statique

```
1 package com.ettounani.Presentation;
2 import com.ettounani.Dao.IDaoImpl;
3 import com.ettounani.Metier.IMetierImpl;
4 public class MainStatic {
5     public static void main(String[] args) {
6         IDaoImpl dao=new IDaoImpl();
7         /* First way
8         IMetierImpl metier=new IMetierImpl(dao);
9         */
10        IMetierImpl metier =new IMetierImpl();
11        metier.setDao(dao);
12        System.out.println(metier.calcul());
13    }
14 }
```



## Application context

```
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"
4       xsi:schemaLocation="http://www.springframework.org/schema/beans
5       http://www.springframework.org/schema/beans/spring-beans.xsd">
6     <bean id="dao" class="com.ettounani.Dao.IDaoImpl"/>
7     <bean id="metier" class="com.ettounani.Metier.IMetierImpl">
8         <property name="dao" ref="dao"/>
9     </bean>
10 </beans>
```



## Presentation Instanciation Dynamique

```
1 package com.ettouant.Presentation;
2 import com.ettouant.Dao.IDao;
3 import com.ettouant.Metier.IMetier;
4 import java.io.File;
5 import java.io.FileNotFoundException;
6 import java.lang.reflect.InvocationTargetException;
7 import java.lang.reflect.Method;
8 import java.util.Scanner;
9 public class MainDynamic {
10     public static void main(String[] args) {
11         try {
12             Scanner scanner=new Scanner(new File("configs.txt"));
13
14             String clsName1=scanner.nextLine();
15             Class daoClass=Class.forName(clsName1);
16             IDao dao=(IDao) daoClass.newInstance();
17
18             String clsName2=scanner.nextLine();
19             Class metierClass=Class.forName(clsName2);
20             IMetier metier=(IMetier) metierClass.newInstance();
21
22             Method setDao=metierClass.getMethod("setDao", IDao.class);
23             setDao.invoke(metier,dao);
24
25             System.out.println(metier.calcul());
26         } catch (FileNotFoundException | ClassNotFoundException |
27             InstantiationException | IllegalAccessException |
28             NoSuchMethodException | InvocationTargetException e) {
29             throw new RuntimeException(e);
30         }
31     }
```



## Pom dependencies

```
1 <dependencies>
2     <dependency>
3         <groupId>org.springframework</groupId>
4         <artifactId>spring-core</artifactId>
5         <version>5.3.23</version>
6     </dependency>
7     <dependency>
8         <groupId>org.springframework</groupId>
9         <artifactId>spring-context</artifactId>
10        <version>5.3.23</version>
11    </dependency>
12    <dependency>
13        <groupId>org.springframework</groupId>
14        <artifactId>spring-beans</artifactId>
15        <version>5.3.23</version>
16    </dependency>
17 </dependencies>
```



## Spring xml

```
1 package com.ettounani.Presentation;
2
3 import com.ettounani.Metier.IMetier;
4 import org.springframework.context.ApplicationContext;
5 import
6     org.springframework.context.support.ClassPathXmlApplicationContext;
7
8 public class SpringXml {
9     public static void main(String[] args) {
10         ApplicationContext context=new
11             ClassPathXmlApplicationContext("applicationContext.xml");
12         IMetier metier=(IMetier) context.getBean("metier");
13         System.out.println(metier.calcul());
14     }
15 }
```



```
SpringAnnotation

1 ...
2 @Component
3 public class IMetierImpl implements IMetier{
4     @Autowired
5     IDao dao;
6 ...
```

```
SpringAnnotation

1 package com.ettounani.Presentation;
2 import com.ettounani.Metier.IMetier;
3 import org.springframework.context.ApplicationContext;
4 import
    org.springframework.context.annotation.AnnotationConfigApplicationContext;
5 public class SpringAnnotation {
6     public static void main(String[] args) {
7         ApplicationContext context= new
            AnnotationConfigApplicationContext("com");
8         IMetier metier=context.getBean(IMetier.class);
9         System.out.println(metier.calcul());
10    }
11 }
```

Vous trouverez le code source du TP sur GitHub :

Lien : <https://github.com/ET-TOUNANI/TP-Inversion-de-Contr-le>

## Captures d'écran

```
SpringAnnotation x
C:\Java\jdk1.8.0_171\bin\java.exe ...
Result --> 31.0 :)

Process finished with exit code 0
```



## Partie 2

### Implémentation



#### IMetier

```
1 package com.enset.metier;  
2 public interface IMetier {  
3     double RealFois2();  
4 }
```



#### ComplexImpl

```
1 package com.enset.Dao;  
2 public final class ComplexImpl implements IComplex{  
3     final Double re;  
4     final Double im;  
5     public ComplexImpl(Double re, Double im) {  
6         this.re = re;  
7         this.im = im;  
8     }  
9     @Override  
10    public Double imaginPart() {  
11        return im;  
12    }  
13    @Override  
14    public Double realPart() {  
15        return re;  
16    }  
17 }
```

## MetierImpl

```
1 package com.enset.metier;
2 import com.enset.Dao.ComplexImpl;
3 import com.enset.Dao.IComplex;
4 public final class MetierImpl implements IMetier{
5     IComplex complex;
6     public MetierImpl(IComplex complex) {
7         this.complex = complex;
8     }
9     @Override
10    public double RealFois2() {
11        return complex.realPart()*2;
12    }
13    public IComplex getComplex() {
14        return complex;
15    }
16    public void setComplex(IComplex complex) {
17        this.complex = complex;
18    }
19    @Override
20    public String toString() {
21        return "("+complex.imaginPart()+"i"+complex.realPart()+")";
22    }
23 }
```

## IComplex

```
1 package com.enset.Dao;
2 public interface IComplex {
3     Double imaginPart();
4     Double realPart();
5 }
```



## MetierImpl

```
1 ...  
2 @Component  
3 public final class MetierImpl implements IMetier{  
4     @Autowired  
5     IComplex complex;  
6 ...
```



## ComplexImpl

```
1 package com.enset.Dao;  
2 @Component  
3 public final class ComplexImpl implements IComplex{
```



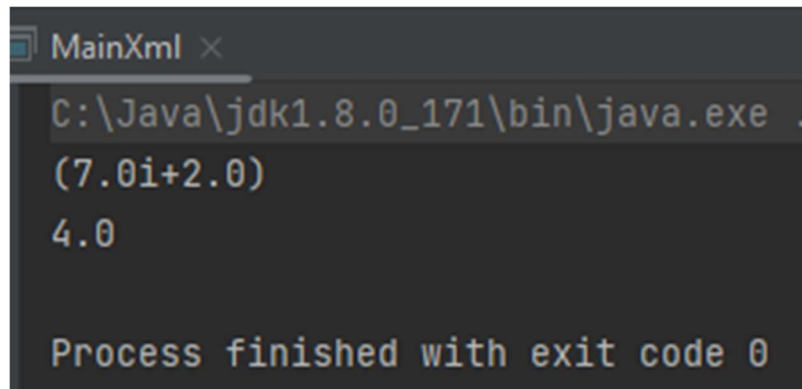
## Applicaion config.xml

```
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"  
4     xsi:schemaLocation="http://www.springframework.org/schema/beans  
5         http://www.springframework.org/schema/beans/spring-beans.xsd">  
6     <bean id="complex" class="com.enset.Dao.ComplexImpl">  
7         <constructor-arg index="0" value="2"/>  
8         <constructor-arg index="1" value="7"/>  
9     </bean>  
10    <bean id="metier" class="com.enset.metier.MetierImpl">  
11        <constructor-arg index="0" ref="complex"/>  
12    </bean>  
13 </beans>
```

Vous trouverez le code source du TP sur GitHub :

Lien : <https://github.com/ET-TOUNANI/TP-Inversion-de-Contr-le>

## Captures d'écran



```
MainXml x
C:\Java\jdk1.8.0_171\bin\java.exe .
(7.0i+2.0)
4.0

Process finished with exit code 0
```

## Conclusion

En conclusion, ce TP sur l'Inversion de contrôle et l'Injection de dépendances a été bénéfique pour comprendre l'importance de ces concepts dans le développement de logiciels et de frameworks modernes. Cela a également permis de découvrir la réflexion API en Java, ce qui peut être utile pour des projets futurs. L'utilisation de ces mécanismes peut améliorer la qualité, la flexibilité et la maintenabilité des applications, ainsi que faciliter le travail en équipe et les tests. Il est évident que ces concepts continueront à jouer un rôle important dans l'évolution de l'informatique et du développement logiciel.