

DEPARTEMENT MATHEMATIQUES ET INFORMATIQUE

Compte rendu du Inversion de contrôle et Injection des dépendances

**Filière :
« Génie du Logiciel et des Systèmes Informatiques Distribués »
GLSID**

Le calcul de l'âge à partir d'année

Réalisé par :

Najwa ZRAIDI

Professeur :

M. Mohamed YOUSSEFI

Année Universitaire : 2022-2023

Introduction

L'inversion de contrôle et l'injection de dépendances sont deux concepts clés en programmation orientée objet et en développement logiciel en général.

L'inversion de contrôle consiste à déléguer la gestion des flux de contrôle et des appels de méthode à un conteneur d'inversion de contrôle, plutôt que de les gérer directement dans le code de l'application. Cela permet de rendre le code plus modulaire, facile à tester et à maintenir.

L'injection de dépendances est une technique de programmation qui consiste à fournir les dépendances d'un objet à l'aide d'un conteneur d'injection de dépendances. En utilisant cette technique, les dépendances d'un objet sont injectées par le conteneur au moment de sa création, plutôt que d'être créées dans le code de l'application. Cela permet de rendre le code plus flexible, facile à tester et à maintenir.

En résumé, l'inversion de contrôle et l'injection de dépendances sont deux concepts clés qui permettent de rendre le code plus modulaire, flexible, facile à tester et à maintenir.

1. Créer l'interface IDao avec une méthode getDate

```
package ma.enset.dao;  
  
public interface IDao {  
  
    String getDate();  
  
}
```

2. Créer une implémentation de cette interface

⇒ Implémentation 1 :

```
package ma.enset.dao;  
  
import org.springframework.stereotype.Component;  
//spring annotation  
@Component("dao")  
public class DaoImpl implements IDao{  
  
    @Override  
    public String getDate() {  
        System.out.println("Version 1 Base de données : ");  
  
        /*  
        Se connecter à la base de données  
        pour récupérer la date  
        */  
        String date="2001-06-18";  
        return date;  
    }  
}
```

⇒ Implémentation 2 :

```
package ma.enset.ext;

import ma.enset.dao.IDao;
import org.springframework.stereotype.Component;

@Component("dao2")
public class DaoImpl2 implements IDao {
    @Override
    public String getDate() {
        System.out.println("Version 2 Base de données : ");
        String date="1975-04-12";
        return date;
    }
}
```

⇒ Implémentation 3 :

```
⇒ package ma.enset.ext;

import ma.enset.dao.IDao;

public class DaoImplVWB implements IDao {
    @Override
    public String getDate() {
        System.out.println("Version Web Service : ");
        String date="1975-04-12";
        return date;
    }
}
```

3. Créer l'interface IMetier avec une méthode calcul

```
package ma.enset.metier;

public interface IMetier {
    //methode optional
    int calcul();
}
```

4. Créer une implémentation de cette interface en utilisant le couplage faible

```
package ma.enset.metier;

import ma.enset.dao.IDao;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Component;

import java.time.LocalDate;
import java.time.Period;

@Component("metier")
public class MetierImpl implements IMetier {
    //injection des dependance spring
    //principe de couplage faible
}
```

```

@Autowired
@Qualifier("dao")//inject moi l'instance qui s'appelle dao
private IDao dao;

public MetierImpl(IDao dao) {
    this.dao = dao;
}
public MetierImpl() {}

@Override
public int calcul() {
    String date=dao.getDate();
    LocalDate localDate = LocalDate.parse(date);
    //System.out.println(localDate);
    LocalDate curDate = LocalDate.now();
    //System.out.println(Period.between(localDate,
curDate).getYears() );
    //curDate.getYear()-localDate.getYear()
    int age=Period.between(localDate, curDate).getYears();
    return age;
}

public void setDao(IDao dao) {
    this.dao = dao;
}
}

```

5. Faire l'injection des dépendances :

a. Par instantiation statique

```

package ma.enset.presentation;

import ma.enset.dao.DaoImpl;
import ma.enset.metier.MetierImpl;
import ma.enset.ext.DaoImpl2;
import ma.enset.ext.DaoImplVWB;

public class Application1 {
    public static void main(String[] args) {
        /*
        injection des dépendances
        par instantiation statique
        => new -> couplage forte
        */
        DaoImpl dao=new DaoImpl();
        DaoImpl2 daoImpl2=new DaoImpl2();
        DaoImplVWB daoImplVWB=new DaoImplVWB();
        //injection via constructeur
        MetierImpl metier=new MetierImpl(dao);
        //metier.setDao(dao);
        System.out.println("L\'age est : " +metier.calcul()+" ans
");
        metier.setDao(daoImpl2);
        System.out.println("L\'age est : " +metier.calcul()+" ans
");
        metier.setDao(daoImplVWB);
        System.out.println("L\'age est : " +metier.calcul()+" ans
");
    }
}

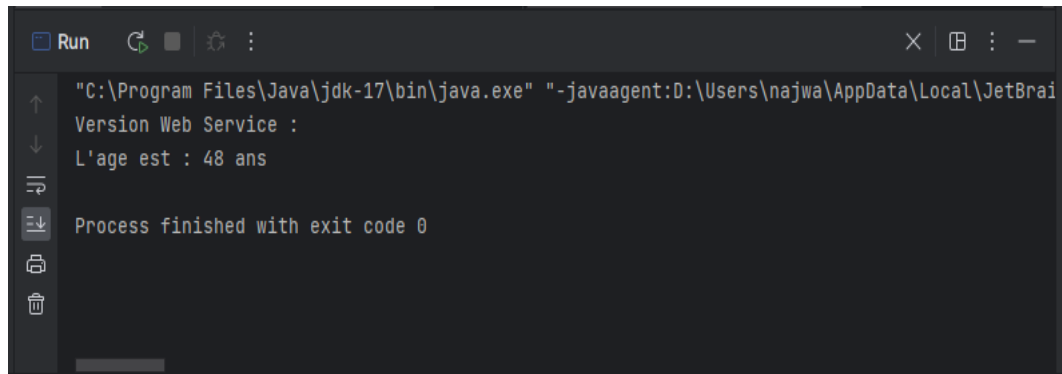
```

```

    }
}

```

⇒ Exécution :



```

Run
"C:\Program Files\Java\jdk-17\bin\java.exe" "-javaagent:D:\Users\najwa\AppData\Local\JetBrai
Version Web Service :
L'age est : 48 ans
Process finished with exit code 0

```

b. Par instanciation dynamique

```

package ma.enset.presentation;

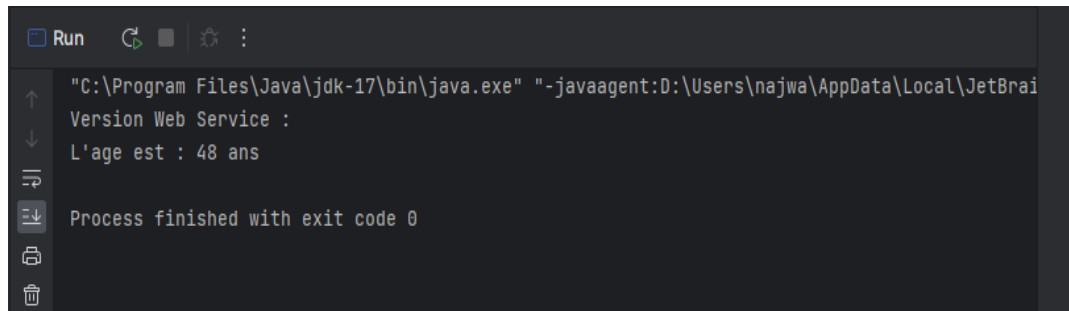
import ma.enset.dao.IDao;
import ma.enset.metier.IMetier;

import java.io.File;
import java.lang.reflect.Method;
import java.util.Scanner;

public class Application2 {
    public static void main(String[] args) throws Exception {
        //la lecteur de fichier text
        Scanner scanner=new Scanner(new
File("C:\\Users\\najwa\\OneDrive\\Desktop\\cours GLSID2
S4\\JEE\\TPs_JEE\\Devoir1\\config.txt"));
        // lire la premire fichier text pour recuperer la lere
classe
        String daoClassName=scanner.nextLine();
        //System.out.println(daoClassName);
        //instanciation dynamique
        Class cDao=Class.forName(daoClassName);
        //System.out.println(cDao);
        //instanciation de class
        IDao Dao = (IDao) cDao.newInstance();
        //System.out.println(Dao.getDate());
        //=> DaoImpl dao=new DaoImpl();
        String metierClassName=scanner.nextLine();
        Class cMetier=Class.forName(metierClassName);
        IMetier metier =(IMetier) cMetier.newInstance();
        //=> MetierImpl dao=new MetierImpl();
        //methode setDao maniere dynamique contient un seul
parametre de type IDao
        Method method=cMetier.getMethod("setDao",IDao.class);
        //execution de methode
        //metier.setDao invoke methode sur l'objet metier et
transmettre par dao
        method.invoke(metier,Dao);
        System.out.println("L'age est : " +metier.calcul()+" ans
");}}

```

⇒ Exécution :



```
Run
"C:\Program Files\Java\jdk-17\bin\java.exe" "-javaagent:D:\Users\najwa\AppData\Local\JetBrai
Version Web Service :
L'age est : 48 ans
Process finished with exit code 0
```

c. En utilisant le Framework Spring - Version XML

```
d. package ma.enset.presentation;

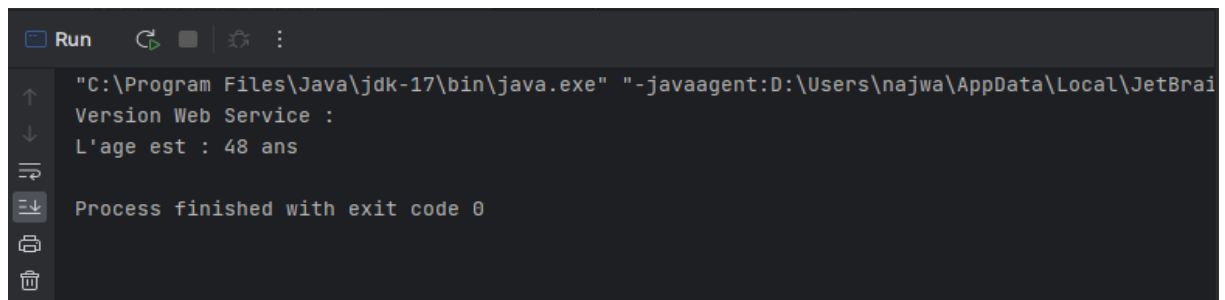
import ma.enset.metier.IMetier;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationCon
text;

public class Application3_spring_xml {
    public static void main(String[] args) {
        ApplicationContext context =new
        ClassPathXmlApplicationContext("ApplicationContexte.xml");
        IMetier metier = (IMetier) context.getBean("metier");
        System.out.println("L'age est : " +metier.calcul()+"
        ans ");
    }
}
```

⇒ Fichier XML :

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
    <bean id="dao" class="ma.enset.ext.DaoImplVWB"></bean>
    <bean id="metier" class="ma.enset.metier.MetierImpl">
        <!-- <property name="dao" ref="dao"></property> pas de
constructeur-->
        <constructor-arg ref="dao"></constructor-arg>
    </bean>
</beans>
```

⇒ Exécution :



```
"C:\Program Files\Java\jdk-17\bin\java.exe" "-javaagent:D:\Users\najwa\AppData\Local\JetBrai
Version Web Service :
L'age est : 48 ans
Process finished with exit code 0
```

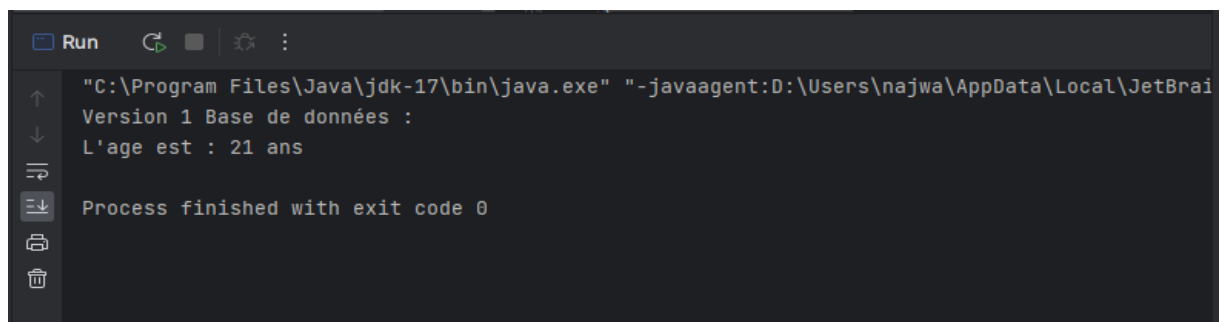
- Version annotations

```
package ma.enset.presentation;

import ma.enset.metier.IMetier;
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class Application4_spring_annotation {

    public static void main(String[] args) {
        //scan de package ma equivalent scan de tous les classes : base
package //ApplicationContext context=new
AnnotationConfigApplicationContext("ma.enset.ao","ma.enset.Metier");
        ApplicationContext context=new
AnnotationConfigApplicationContext("ma");
        IMetier metier = context.getBean(IMetier.class);
        // IMetier metier = (IMetier) context.getBean("metier");
        System.out.println("L'age est : " +metier.calcul()+" ans ");
    }
}
```



```
"C:\Program Files\Java\jdk-17\bin\java.exe" "-javaagent:D:\Users\najwa\AppData\Local\JetBrai
Version 1 Base de données :
L'age est : 21 ans
Process finished with exit code 0
```

Conclusion :

En conclusion, l'inversion de contrôle et l'injection de dépendances sont deux concepts fondamentaux en programmation orientée objet et en développement logiciel en général. En utilisant ces techniques, les développeurs peuvent rendre leur code plus modulaire, flexible, facile à tester et à maintenir.