Ecole Normale Supérieure de l'Enseignement Technique Mohammedia

Université Hassan II de Casablanca

ENSET

ا لمدرسة العليا لأساتذة التعليم التقني المحمدية

جامعة الحسن الثاني بالدار البيضاء

DÉPARTEMENT MATHÉMATIQUES ET INFORMATIQUE

Filière :

« Ingénierie Informatique : Big Data et Cloud Computing »

II-BDCC

TP 1: Inversion de contrôle et Injection des dépendances

Réalisé Par: Laila Sad Elbouyoud

Classe: ii-BDCC 2

1. Création de l'interface IDao

```java
package dao;

public interface IDao {
    double getData();
}
```

2. Implémentation de l'interface IDao

```java
package dao;

public class DaoImpl implements IDao{
    @Override
    public double getData() {
        double temp = Math.random()*20;
        return temp;
    }
}
```

3. Interface IMetier

```java
package metier;

public interface IMetier {
    double calcul();
}
```

4. Implémentation de l'interface IMetier en utilisant le couplage faible

```java
package metier;

import dao.IDao;

public class MetierImpl implements IMetier{
    private IDao dao;

    @Override
    public double calcul() {
        double temp = dao.getData();
        double r = temp *40/3;
        return r;
    }

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

5. Injection des dépendances :

   a. instanciation statique : **couplage fort**

```java
package pres;

import dao.DaoImpl;
import metier.MetierImpl;

public class pres_statique {

    public static void main(String[] args){

        DaoImpl dao = new DaoImpl();
        MetierImpl metier = new MetierImpl();
        metier.setDao(dao);
        System.out.println("Résultat : " + metier.calcul());
    }
}
```

b. instanciation dynamique : **couplage faible**

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

public class pres_dynamique {

    public static void main(String[] args) throws Exception {

        Scanner scanner = new Scanner(new File( pathname: "config.txt"));

        String daoClassName = scanner.nextLine();
        Class cDao = Class.forName(daoClassName);
        IDao dao = (IDao) cDao.newInstance();

        String metierClassName = scanner.nextLine();
        Class cMetier = Class.forName(metierClassName);
        IMetier metier = (IMetier) cMetier.newInstance();

        Method methode = cMetier.getMethod( name: "setDao", IDao.class);
        methode.invoke(metier, dao);

        System.out.println("Résultat : " + metier.calcul());
    }
}
```

c. Framework Spring :

i. Version XML :

Fichier applicationContext.xml:

```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/sche

    <bean id="dao" class="dao.DaoImpl"></bean>
    <bean id="metier" class="metier.MetierImpl">
        <property name="dao" ref="dao"></property>
    </bean>
</beans>
```

classe présentation:

```java
import metier.IMetier;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class pres_SpringXML {

    public static void main(String[] args){
        ApplicationContext context = new ClassPathXmlApplicationContext( configLocation: "applicationContext.xml
        IMetier metier = (IMetier) context.getBean( s: "metier");
        System.out.println("Résultat : " + metier.calcul());
    }
}
```

ii. Versions annotations :

ajout des composants **@Component(“nom”)**

```java
@Component("dao")
public class DaoImpl implements IDao{
    @Override
    public double getData() {
        double temp = Math.random()*20;
        return temp;
    }
}
```

et **@Autowired** et **@Qualifier(“nom”)** pour faire les injections

```java
@Component("metier")
public class MetierImpl implements IMetier{
    @Autowired
    @Qualifier("dao")
    private IDao dao;

    @Override
    public double calcul() {
        double temp = dao.getData();
        double r = temp *40/3;
        return r;
    }
```

On peut aussi utiliser un constructeur au lieu de @Autowired, c'est une solution plus recommandée :

```java
@Component("metier")
public class MetierImpl implements IMetier{
    //@Autowired
    //@Qualifier("dao")
    private IDao dao;

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

classe présentation :

```java
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class pres_SpringAnnotation {

    public static void main(String[] args){
        ApplicationContext context = new AnnotationConfigApplicationContext( ...basePackages: "dao", "metier");
        IMetier metier = context.getBean(IMetier.class);
        System.out.println("Résultat : " + metier.calcul());
    }
}
```