ECOLE MAROCAINE DES SCIENCES DE L'INGENIEUR
Membre de
HONORIS UNITED UNIVERSITIES

Rapport

De Travaux Pratiques

4IIR-EMSI Centre -G5

Réalisé par :

- FAIDI Zahira

Encadré par :

Pr.BADRI Tijane

Année Universitaire 2022/2023

Couplage fort & faible

Le couplage fort fait référence à la dépendance étroite entre les classes, Plus précisément, il se réfère au fait que deux classes sont étroitement couplées si l'une dépend fortement de l'implémentation interne de l'autre. Alors le principe (opren/closed) des principes SOLID n'est pas respecter.

Exemple de Couplage fort :

```java
package org.example.CouplageFort;

public class Moteur {
    void demarrer(){
        System.out.println("Démerrer le moteur");
    }
}
```

```java
package org.example.CouplageFort;

public class Voiture {
    Moteur m;
    void bouger(){
        m = new Moteur();
        m.demarrer();
        System.out.println("Vitesse 10Km/h");
    }
}
```

```java
package org.example.CouplageFort;

public class Main {
    public static void main(String[] args) {
        Voiture v = new Voiture();
        v.bouger();
    }
}
```

Dans cet exemple la classe Voiture dépende fortement de la classe Moteur si par la suite en veut implémenter une voiture par batterie il faut forcement ouvrier la classe voiture pour effectuer cette modification.

Un objet faiblement couplé est un objet qui interagit avec d'autres objets de manière minimale et dont la modification n'affecte pas les autres objets avec lesquels il interagit. En d'autres termes, un objet faiblement couplé est un objet qui a peu de dépendances avec les autres objets du système et qui est donc plus facile à maintenir et à modifier.

Pour atteindre un couplage faible, les développeurs peuvent utiliser des interfaces ou des abstractions pour limiter les interactions entre les objets. Cela permet de réduire les dépendances et d'isoler les changements, ce qui facilite la maintenance et la réutilisation du code.

Exemple de couplage faible :

```java
package org.example.CouplageFaible;

public interface IMoteur {
    void demarrer();
}
```

```java
package org.example.CouplageFaible;

public interface IVoiture {
    void rouler();
}
```

```java
package org.example.CouplageFaible;

public class Moteur implements IMoteur{
    @Override
    public void demarrer() {
        System.out.println("Démarrer le moteur!");
    }
}
```

```java
package org.example.CouplageFaible;

public class Voiture implements IVoiture{
    private IMoteur moteur;
    @Override
    public void rouler() {
        moteur.demarrer();
        System.out.println("La voiture roule correctement!");
    }

    public void setMoteur(IMoteur moteur) {
        this.moteur = moteur;
    }
}
```

```java
package org.example.CouplageFaible;

public class Voyage {
    public static void main(String[] args) {
        Voiture v = new Voiture();
        v.setMoteur(new Moteur());
        v.rouler();
        System.out.println("Bon voyage!");
    }
}
```

Dans ce cas si on veut par la suite changer le moteur par une batterie il suffit de crée une classe qui implémenté l'interface « IMoteur »

Injection des dépendances

À la base du Spring Framework, on trouve un unique principe de conception : l'inversion de contrôle qui est une façon de concevoir l'architecture d'une application en se basant sur le mécanisme objet de l'injection de dépendance.

L'injection de dépendance est un mécanisme simple à mettre en œuvre dans le cadre de la programmation objet et qui permet de diminuer le couplage entre deux ou plusieurs objets.

1. Instanciation statique

```java
package dao;

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

```java
package dao;
public class DaoImpl implements IDao{
    @Override
    public double getData() {
        System.out.println("From SQL DB");
        return (7);
    }
}
```

```java
package metier;

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

```java
package metier;
import dao.IDao;
public class MetierImpl implements IMetier{
    IDao dao;
     @Override
     public double calcul() {
         double data = dao.getData();
         return data*10;
     }

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

```java
package presentation;

import dao.DaoImpl;
import dao.DaoNSQL;
import metier.MetierImpl;

public class Presentation {
    public static void main(String[] args) {
        MetierImpl metier = new MetierImpl();
        DaoNSQL nosql = new DaoNSQL();

        metier.setDao(nosql);

        double resultat = metier.calcul();

        System.out.println("Résultat est : "+resultat);
    }
}
```

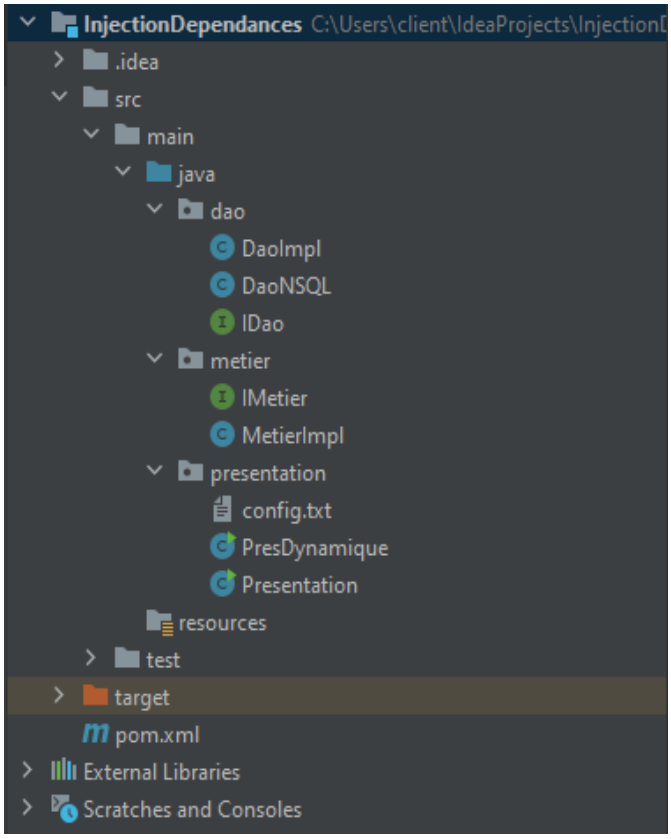
2. Instanciation Dynamique

Fichier de configuration

```
dao.DaoImpl
metier.MetierImpl
```

```java
package presentation;

import dao.IDao;
import metier.IMetier;

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

public class PresDynamique{
    public static void main(String[] args) throws Exception{
        Scanner sc = new Scanner(new
File("src/main/java/presentation/config.txt"));
        String dao = sc.nextLine();
        Class clsDao = Class.forName(dao);
        IDao objDao = (IDao) clsDao.newInstance();

        String metier = sc.nextLine();
        Class clsMetier = Class.forName(metier);
        IMetier objMetier = (IMetier) clsMetier.newInstance();

        Method method = clsMetier.getMethod("setDao",IDao.class);
        method.invoke(objMetier,objDao);

        System.out.println(objMetier.calcul());
    }
}
```

Structure du projet



Injection des dépendances avec les Annotations

```
package dao;

import org.springframework.stereotype.Component;
@Component
public class DaoImpl implements IDao{
    public double getData() {
        System.out.println("From SQL DB");
        return (7);
    }
}
```

```
package metier;

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

@Component
public class MetierV3 implements IMetier{
    @Autowired
    IDao dao;
    @Override
    public double calcul() {
        double d = dao.getData();
        return d*2021;
    }
    public void setDao(IDao dao) {
        this.dao = dao;
    }
}
```

```
package annotation;

import metier.IMetier;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ApplicationContext;
import
org.springframework.context.annotation.AnnotationConfigApplicationContext;

@SpringBootApplication
public class AnnotationApplication {

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

}
```