ECOLE MAROCAINE DES SCIENCES DE L'INGENIEUR

Membre de HONORIS UNITED UNIVERSITIES

Rapport de Travaux Pratiques

Réalisé par :

- *Abderrahim Elazhari*

Encadré par :

-Pr.Badri Tijane

Année universitaire 2022/2023

Couplage fort et faible

Un couplage fort signifie que les classes et les objets dépendent les uns des autres. En général, le **couplage fort** n'est pas bon car il réduit la flexibilité et la réutilisation du code, tandis que le **couplage faible** signifie la réduction des dépendances d'une classe qui utilise directement les différentes classes.

Couplage fort:

- Un objet fortement couplé est un objet qui a besoin de connaître les autres objets et est généralement très dépendant les uns des autres.
- La modification d'un objet dans une application fortement couplée nécessite souvent de modifier d'autres objets.

  Exemple de Couplage fort :

```
package org.example.CouplageFort;

2 usages
public class Moteur {
    1 usage
    void demarrer(){

        System.out.println("Demarrer le moteur");
    }
}
```

```java
package org.example.CouplageFort;

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

```java
package org.example.CouplageFort;

no usages
public class Main {
    no usages
    public static void main(String[] args) {
        Voiture v=new Voiture();
        v.bouger();
    }
}
```

Dans cet exemple la classe Voiture dépend fortement de la classe Moteur.

L'implémentation d'un moteur avec une batterie nécessite une ouverture de la classe voiture pour modifier.

Couplage faible:

- Le couplage faible permet de réduire les interdépendances entre les composants d'un système dans le but de réduire le risque que les changements dans un composant nécessitent des changements dans tout autre composant.
- Le couplage faible est un concept destiné à augmenter la flexibilité du système, à le rendre plus maintenable et à rendre l'ensemble du framework plus stable.

Exemple du couplage faible :

```java
package org.example.CouplageFaible;

3 usages   1 implementation
public interface IMoteur {
    1 usage   1 implementation
    void demarrer();
}
```

```java
package org.example.CouplageFaible;

1 usage   1 implementation
public interface IVoiture {
    1 usage   1 implementation
    void rouler();

}
```

```java
package org.example.CouplageFaible;

1 usage
public class Moteur implements IMoteur{

    1 usage
    @Override
    public void demarrer() {
        System.out.println("Demarrer le moteur");
    }
}
```

```java
package org.example.CouplageFaible;

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

    }

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

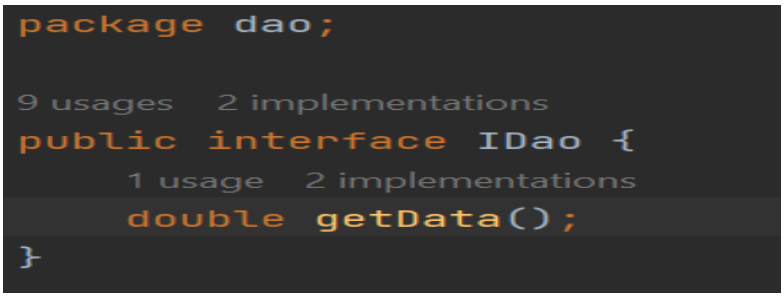
```java
package org.example.CouplageFaible;

no usages
public class Voyage {
    no usages
    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 changer le moteur par batterie il suffit de créer une classe qui implémente l'interface IMoteur.

Injection des dépendances

L'injection de dépendances 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.

-**Instanciation statique**



```java
package dao;

2 usages
public class DaoImpl implements IDao {

    1 usage
    @Override
    public double getData() {
        System.out.println("From SQL DB");
        return 7;
    }
}
```

```java
package metier;

4 usages   1 implementation
public interface IMetier {
    2 usages   1 implementation
    double calcul();
}
```

```java
package metier;

import dao.IDao;

4 usages
public class MetierImpl implements IMetier{
    2 usages
    IDao dao;
    2 usages
    @Override
    public double calcul() {
        double data = dao.getData();
        return data*10;
    }

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

```java
package presentation;

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

no usages
public class presentation {
    no usages
    public static void main(String[] args) {
        MetierImpl metier= new MetierImpl();
        DaoNSQL nosql= new DaoNSQL();
        metier.setDao(nosql);
        double resultat= metier.calcul();
        System.out.println("Resultat est:" + resultat);
    }
}
```

-Instanciation Dynamique

-Fichier de configuration

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

```java
package presentation;

import dao.IDao;
import metier.IMetier;
import java.io.File;
import java.lang.reflect.Method;
import java.util.Scanner;
no usages
public class PresDynamique {
    no usages
    public static void main(String[] args) throws Exception{
        Scanner sc=new Scanner(new File( pathname: "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( name: "setDao",IDao.class);
        method.invoke(objMetier,objDao);

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

Injection des dépendances avec les annotations

```java
package dao;

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

```java
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;
    }
}
```

```java
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());
    }

}
```