ECOLE NORMALE SUPÉRIEURE DE L'ENSEIGNEMENT
TECHNIQUE DE MOHAMMEDIA
UNIVERSITÉ HASSAN II DE CASABLANCA

Département Mathématique informatique

Compte Rendu

filière :

“ Ingénierie Informatique - Big Data & Cloud Computing ”

II-BDCC

Injection Des Dépendances

Réalisation :

Zakaria Mansouri

Année Universitaire : 2022 - 2023

Résumé

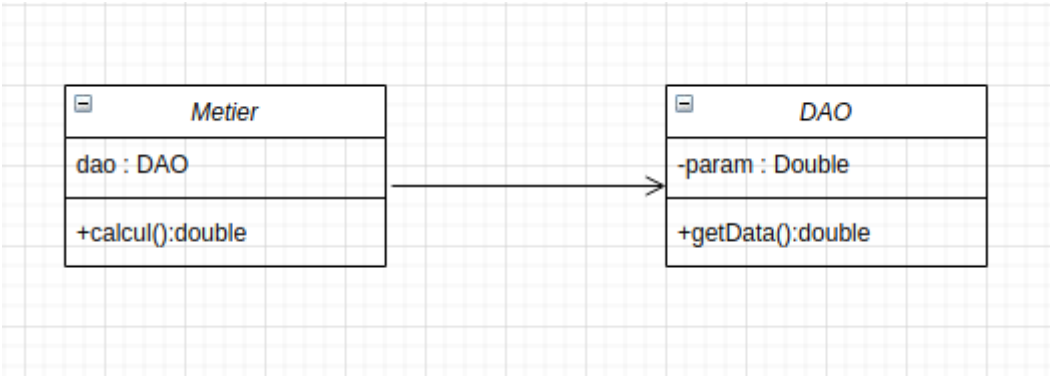
chaque projet informatique à des exigences fonctionnelles c'est à dire les fonctionnalités du système qu'il doit être capable de faire et des exigences techniques tel que : la performance , la sécurité , la persistances des données , la maintenabilité, l'évolutivité

et pour qu'une application soit maintenable ,robuste , et évolutif elle doit respecter la règle d'or :

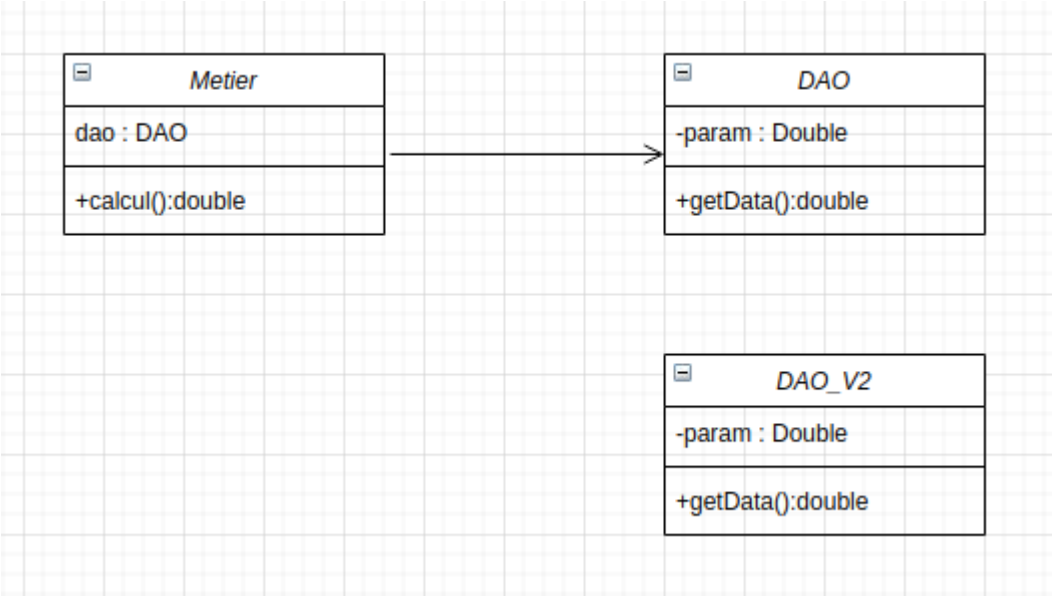
"Une application doit être fermé à la modification et ouverte à l'extension"

La question qui se pose maintenant c'est comment on peut développer une application qui respecte la règle d'or ci-dessus ?

➔ nous allons traiter l'exemple suivant:

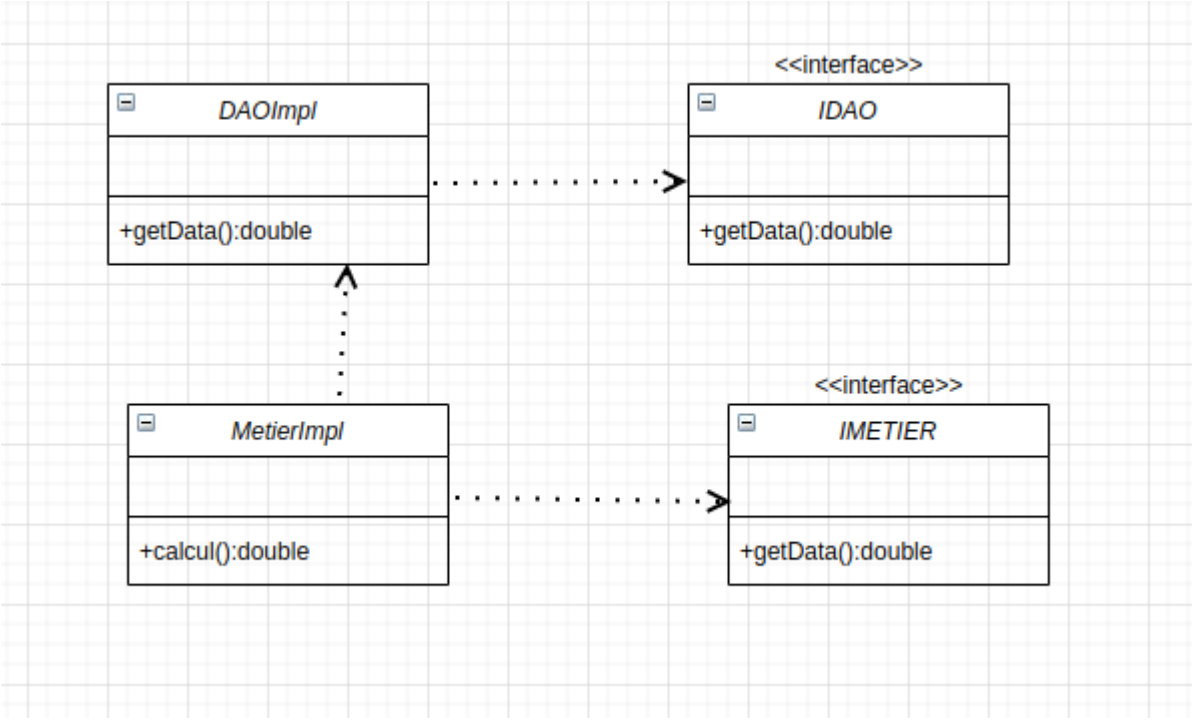


on a la classe Métier dépend directement du classe DAO et donc si on veut évoluer notre application on va briser la règle d'or :



on dit qu'il ya un couplage fort entre la classe Metier et la classe DAO et toujours lors de l'utilisation du couplage fort l'application sera ouvert à la modification

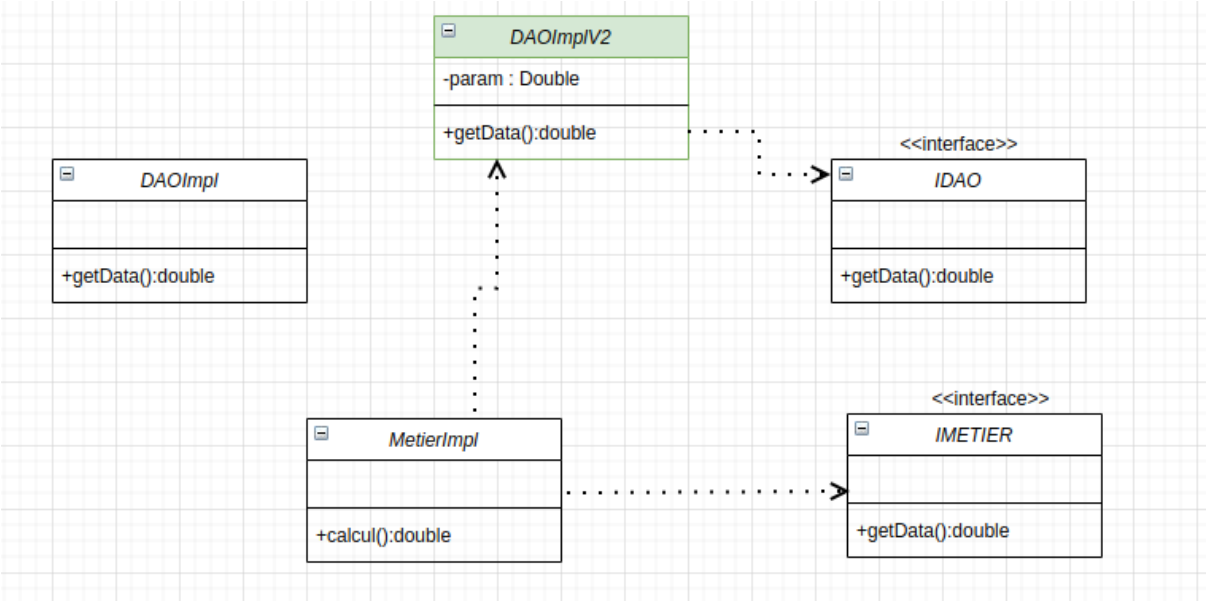
la solution est d'utiliser la couplage faible comme il montre le diagramme suivant :



maintenant on a pas aucune classe qui dépend d'une autre classe mais plutôt tout les classes dépendent des interfaces

"Dépendez les interfaces et ne dépendez pas les classes"

et c'est ça le couplage faible.
maintenant si on a besoin d'évoluer notre application par exemple au lieu d'utiliser la classe DAOImpl on veut utiliser une autre version DAOImplV2 et sans briser la règle d'or bien sûr !!!!!



pour faire échanger une classe avec une autre on doit utiliser la notion de l'injection des dépendances qui est un mécanisme de l'inversion de contrôle .

nous allons traiter l'exemple ci dessus :

➔ Création de l'interface IDAO:

```java
package dao;

public interface IDAO {
    double getdata();
}
```

Création d'une implémentation de l'interface IDAO :

```java
package dao;

public class DaoImpl implements IDAO{
    public double getdata() {
        System.out.println("version base de données");
        return Math.PI*100/Math.cos(10);
    }
}
```

➔ Création de l'interface IMetier:

```java
package metier;

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

➔ Création d'une implémentation de l'interface IMetier :

```java
package metier;
import dao.IDAO;

public class MetierImpl implements IMetier {
    private IDAO dao;

    public double calcul() {
        double res = dao.getdata();
        return Math.random() * 100 + 3 * res;
    }
}
```

➔ on a dao est de type IDAO , càd que l'on peut utiliser n'importe quel objet d'une classe qui implémente l'interface IDAO

➔ maintenant on va passer à la partie injection des dépendances:

➔ il existe plusieurs types de l'injection des dépendances:

- par instanciation statique .
- par instanciation dynamique.
- en utilisant le framework Spring

Par Instanciation Statique:

➔ On va créer une instance de la classe DaoImpl et puis utiliser le setter de la classe MetierImpl pour injecter les dépendances

➔ Création du fichier présentation_statique.java

```java
package presentation;
import dao.DaoImpl;
import metier.MetierImpl;
public class presentation_statique {
    public static void main(String args[])
    {
        DaoImpl dao = new DaoImpl();
        MetierImpl metier = new MetierImpl();
        metier.setDao(dao);
        System.out.println(metier.calcul());
    }
}
```

➔ on a utiliser le setter pour injecter l'objet dao

Par Instanciation Dynamique:

➔ pour l'instanciation dynamique on va utiliser un fichier de configuration qui contient les noms des classes qu'on doit instancier dynamiquement et passer les dépendances via le setter d'une manière dynamique

d'abord on va créer une autre implémentation pour bien tester les choses par exemple DaoImplV2 dans un package appelé ext

```java
package ext;

import dao.IDAO;

public class DaoImplV2 implements IDAO{

    public double getdata() {

        System.out.println("versions capteurs");
        return Math.random() * 1000;
    }
}
```

➔ Création du fichier de configuration:

CONFIG.TXT

```
ext.DaoImplV2
metier.MetierImpl
```

➔ Création du programme pour l'injection des dépendances par instanciation dynamique:

```
public class presenation_dynamique {
    public static void main(String args[]) throws FileNotFoundException, ClassNotFoundException, IllegalAccessException,
        Scanner scanner = new Scanner(new File( pathname: "src/main/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 method = cMetier.getMethod( name: "setDao", IDAO.class);
        method.invoke(metier, dao);

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

➔ le résultat d'exécution :

```
/usr/lib/jvm/java-8-openjdk/bin/java ...
versions capteurs
Résultat = 1115.05524864768

Process finished with exit code 0
```

➔ si on change le fichier de configuration :

CONFIG.TXT | PRESENATION DYNAMIQUE.JAVA

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

➔ et on réexécute le résultat est :

```
/usr/lib/jvm/java-8-openjdk/bin/java ...
version base de données
Résultat = -1062.584367274692

Process finished with exit code 0
```

En utilisant le framework Spring:

➔ la première étape consiste à ajouter les dépendances du Framework Spring dans notre projet , dans le fichier pom.xml:

```xml
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-core</artifactId>
        <version>5.3.16</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>5.3.16</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-beans</artifactId>
        <version>5.3.16</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-aop</artifactId>
        <version>5.3.16</version>
    </dependency>
```

➔ en utilisant spring il ya 2 versions de l'injection des dépendances :

- en utilisant la version xml
- en utilisant la version annotations

En utilisant la version xml

➔ on va créer un fichier xml qui contient la configuration dans lequel spring va l'utiliser pour l'injection :
on crée le fichier appconfig.xml dans le dossier ressources:

```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/

    <!-- bean definitions here -->

    <bean name="dao" class="dao.DaoImpl"/>
    <bean name="metier" class="metier.MetierImpl">
        <constructor-arg ref="dao"/>
    </bean>
</beans>
```

➔ Création du programme présentation de l'injection en utilisant Spring version xml:

```java
package presentation;

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

public class presentation_spring_xml {
    public static void main(String args[])
    {
        ApplicationContext applicationContext = new ClassPathXmlApplicationContext( configLocation: "appconfig.xml");
        IMetier metier = (IMetier) applicationContext.getBean( name: "metier");
        System.out.println(metier.calcul());

    }
}
```

➔ Exécution:

```
/usr/lib/jvm/java-8-openjdk/bin/java ...
version base de données
-1072.757545913389

Process finished with exit code 0
```

En utilisant la version Annotations :

il suffit d'ajouter avant la déclaration des classes qu'on veut instancier l'annotation *@Component*

dans notre cas c'est les classes : MetierImpl,DaoImpl

```java
package dao;

import org.springframework.stereotype.Component;

@Component
public class DaoImpl implements IDAO{
    public double getdata() {
        System.out.println("version base de données");
        return Math.PI*100/Math.cos(10);
    }
}
```

```java
package metier;
import dao.IDAO;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component
public class MetierImpl implements IMetier {

    @Autowired
    private IDAO dao;

    public MetierImpl(IDAO dao) {
        this.dao = dao;

    }
```

pour faire l'injection on ajoute le mot clé *@Autowired* , Spring va chercher un objet déjà instancié et puis il l'injecte sur la classe MetierImpl

➔ Exécution :

```
/usr/lib/jvm/java-8-openjdk/bin/java ...
version base de données
-1106.2705105225364

Process finished with exit code 0
```

➔ Le Code Complet sur Github :

https://github.com/zakariamanssouri/Mansouri-JEE/tree/mastr/InjectionDépendances