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

DEPARTEMENT MATHEMATIQUES ET INFORMATIQUE

Compte rendu de l'activité pratique 1

Principe de l'Inversion de Contrôle et Injection des dépendances

Filière :
« Génie du Logiciel et des Systèmes Informatiques Distribués »
GLSID2

Module : Architecture Distribuée et Middlewares

Élaboré par :

ELMAJNI Khaoula

Encadré par :

M. YOUSSFI Mohammed

Année Universitaire : 2021-2022

Architecture des applications

Il existe 2 types d'architectures des applications :

- **Monolithique :** ce type conciste sur une seule technologie pour une application, et l'application est développée dans un seule bloc.
- **Micro-services :** découper l'application en plusieurs modules, dont chacun résous un petit problème, et développé avec une technologie différente qui peut répondre plus justement à ce problème, et chaque module peut être déployer sur un serveur différent.

Exigences d'un projet informatique

Chaque projet informatique a 2 types des exigences :

Exigence fonctionnelle : ce sont les besoins fonctionnels de l'application (les besoins métiers de l'entreprise)

Exigences techniques : ce sont des besoins techniques qu'ils peuvent être résumer comme suit :

La performance : c'est-à-dire tous ce qui concerne le temps de réponse de l'application, le problème de montée en charge, qu'on peut le résoudre par la scalabilité qui peut peut prendre 2 formes :

- ***Horizontale :*** qui se caractérise par l'équilibrage de charge et la tolérance aux pannes, cette scalabilité conciste sur le démarrage de l'application en plusieurs instances dans différentes machines pour palier au problème de montée en charge, avec un serveur Load Balencer pour l'équilibrage de charge des requete reçu de chez les clients.
- ***Verticale :*** cette scalabilité conciste sur le démarrage de l'application en une seule l'augmentation des capacités du serveur (CPU/RAM/DD), et l'application va créer un thread pour chaque requete d'un client.

La maintenance : car une application doit être évolutive dans le temps, mais en prenant en compte la règle d'or : « une application doit être fermée à la modification et ouverte à l'extension » parce que les modifications peuvent générer les problèmes de régression (un problème peut générer plusieurs autres problèmes)

La sécurité : les failles de sécurité sont un problème critique au développement informatique, sur lequel on doit faire attention.

La persistance des données et la gestion des transactions

Les versions : l'une des suivante (web/mobile/desktop)

Inversion de controle

C'est un principe qui conciste sur la séparation de tous ce qui métier à tous ce qui est technique, il permet aux développeurs de s'occuper uniquement du code métier alors que le code technique va être occuper par un framework, en utilisant le paradigme AOP (Aspect Oriented Programming).

Injection des dépendances

En injection des dépendances il y a 2 types :

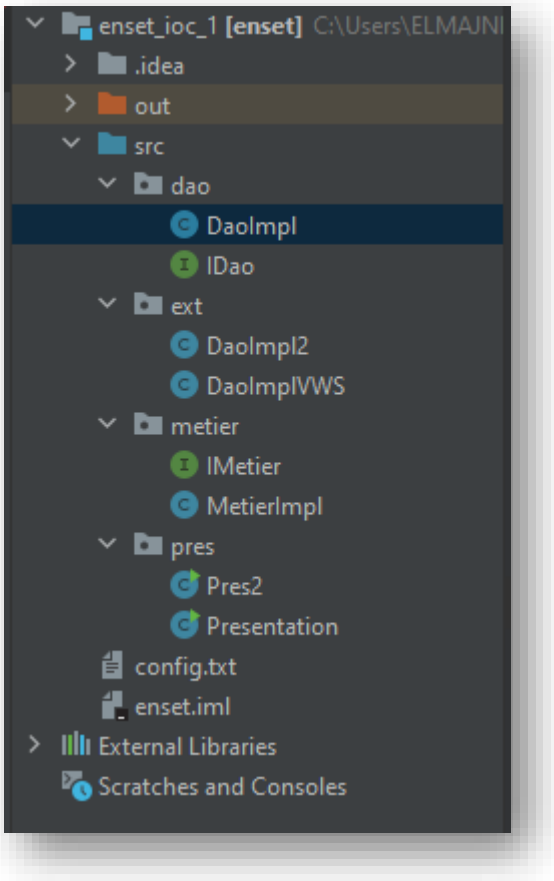
Couplage fort : lorsqu'on a des classes dépendent des autres classes, la chose qui va faire difficile de faire une modification parce qu'on doit faire à chaque fois une modification au code source.

Couplage faible : lorsqu'on a des classes dépendent des interfaces et pas des classes la chose qui va faciliter l'attribution d'une modification.

Travail pratique

Ce travail est pour pratiquer le concept d'inversion de contrôle et d'injection des dépendances.

Voici l'architecture du travail :



Le dossier « dao » contient la couche DAO qui interroge la BDD.

Le dossier « ext » contient la partie d'extension de l'application.

Le dossier « metier » contient la couche métier (les besoins fonctionnels) de l'application.

Le dossier « pres » contient tous ce qui concerne la partie présentation UI (user interface).

Le fichier text « config.txt » de configuration pour la l'injection des dépendances dynamiquement

Interface qui déclare la méthode qu'on aura besoin dans la couche DAO :

```
package dao;

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

Cette classe représente une implémentation de l'interface IDao qui redéfini la méthode 'getData' :

```
package dao;

public class DaoImpl implements IDao{
    @Override
    public double getData() {
        /*
        * se connecter à la BD pour recupérer la
temperature
        * */
        System.out.println("version BD");
        double temp = Math.random()*40;
        return temp;
    }
}
```

à la maintenance on a respecté la régle d'or "l'application doit etre fermée à la modification et ouverte à l'extension" et on a ajouté un autre dossier pour déclarer des nouvelles implémentations de l'interface IDao:

```
package ext;

import dao.IDao;

public class DaoImpl2 implements IDao {
    @Override
    public double getData() {
        System.out.println("versions capteurs");
        double temp = 80;

        return temp;
```

```
    }
}
```

une autre implémentation de l'interface IDao dans la partie d'extension :

```
package ext;

import dao.IDao;

public class DaoImplVWS implements IDao {
    @Override
    public double getData() {
        System.out.println("version : web service");
        return 90;
    }
}
```

dans la couche métier on a déclarer une interface 'IMetier' avec le signature d'une méthode (calcul) pour faire des traitements (besoin fonctionnel) :

```
package metier;

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

la création d'une implémentation de l'interface 'IMetier' avec la déclaration d'un objet de type d'interface 'IDao', ça va nous permet d'utiliser n'importe quelle implémentation de cette interface-là :

```
package metier;

import dao.IDao;

public class MetierImpl implements IMetier{
    //couplage faible
    private IDao dao;

    @Override
    public double calcul() {
        double temp = dao.getData();//n'importe quel
```

```
source:classe
        double res=temp*540/Math.cos(temp*Math.PI);
        return res;
    }

    //permet d'injecter dans la variable dao un obj
d'une classe qui implemente l'interface IDao
    public void setDao(IDao dao) {
        this.dao = dao;
    }
}
```

la couche présentation, ici la déclaration peut se faire en 2 versions : soit statique avec le «new» et directement passer à l'implémentation ciblé, comme suit :

```
package pres;

import dao.DaoImpl;
import ext.DaoImpl2;
import metier.MetierImpl;

public class Presentation {
    public static void main(String[] args) {
        // v1 ==> DaoImpl dao = new DaoImpl();

        /*
        l'instanciation se fait en 2 :
                                        1- statique
                                        2- dynamique


        * injection des dependances par instanciation
statique => new
        */
        DaoImpl2 dao = new DaoImpl2();

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

une autre présentation mais avec la version dynamique, on a déclaré un fichier texte qui contient les noms des différentes implémentations déjà déclarés qu'on va l'utiliser pour l'injection des dépendances d'une façon dynamique sans passer par l'instanciation des objets par 'new' :

```
1  dao.DaoImpl
2  metier.MetierImpl
3
```

```
package pres;

import dao.IDao;
import metier.IMetier;

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

public class Pres2 {
    public static void main(String[] args) throws
Exception{
        /*
        l'instanciation se fait dynamiquement



        Scanner reader = new Scanner(new
File("config.txt"));
        String daoClassName = reader.next();
        Class cDao = Class.forName(daoClassName);
        IDao dao = (IDao) cDao.newInstance();

        String metierClassName = reader.next();
        Class cMetier = Class.forName(metierClassName);
```

```
        IMetier metier = (IMetier)
cMetier.newInstance();

        Method method =
cMetier.getMethod("setDao",IDao.class);
        //metier.setDao(dao);
        method.invoke(metier,dao);



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