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 Travail Dirigé 1

Design Pattern

Filière :
« Génie du Logiciel et des Systèmes Informatiques Distribués »
GLSID3

Module : Design Pattern & Architecture distribuée .Net

Élaboré par :

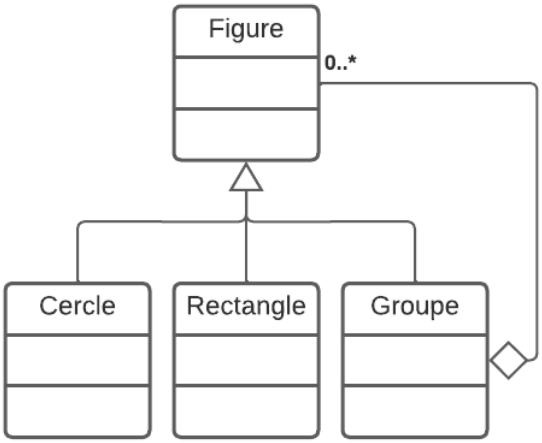
ELMAJNI Khaoula

Encadré par :

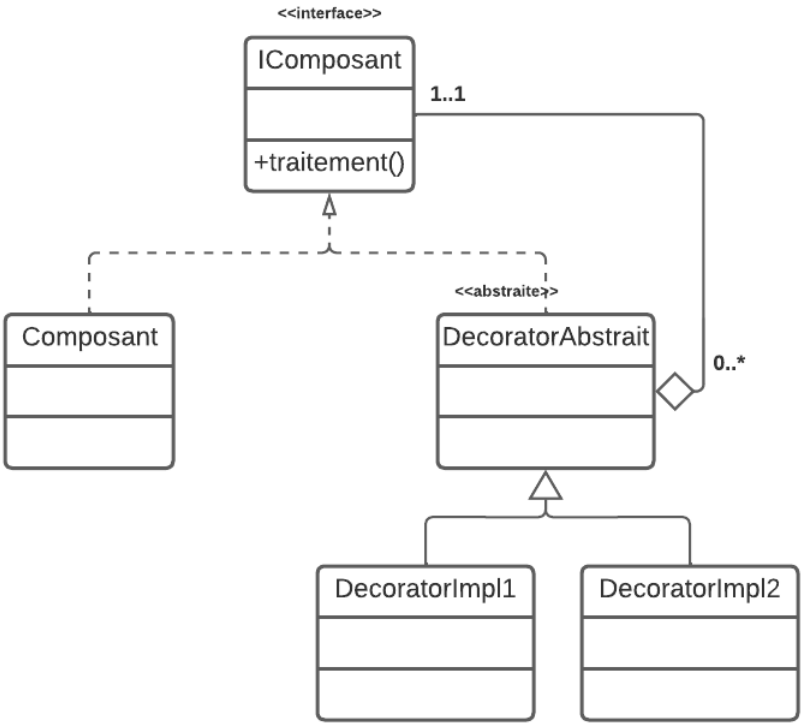
M. ELYOUSFI Mohammed

Année Universitaire : 2022-2023

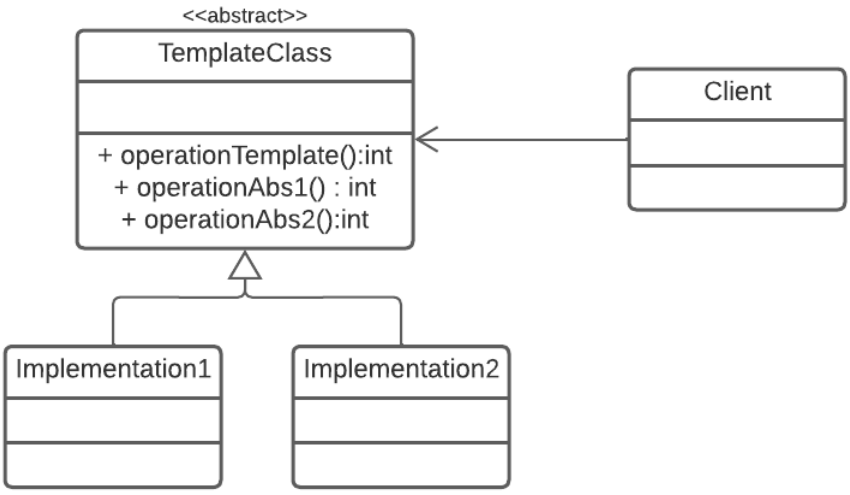
1- Une figure peut être soit un cercle, un rectangle ou un groupe de figures.
Réponse : le Design pattern adéquate à cette situation est **composite**.



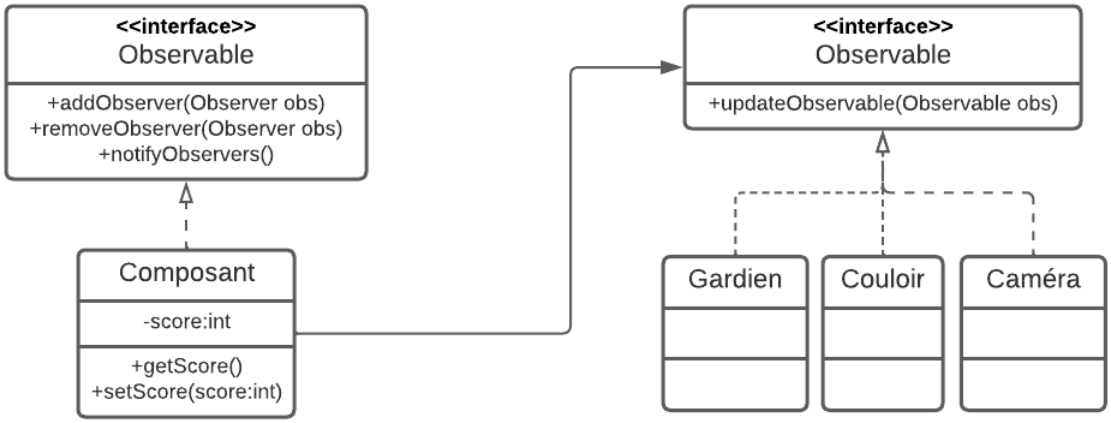
2- Un plugin contient une opération implémentant le squelette d'un algorithme dont deux parties (partie1 et partie2) sont variables. On voudrait laisser le développeur la possibilité d'implémenter les deux parties manquantes de cet algorithme et on voudrait aussi que l'application cliente puisse instancier une implémentation concrète du plugin sans connaitre sa classe d'implémentation.
Réponse : le Design pattern adéquate à cette situation est **Décorateur**.



3- On dispose d'un composant implémentant une interface qui définir une opération « traitement() ». On voudrait rattacher à ce composant des responsabilités supplémentaires sans modifier son code source. C'est-à-dire envelopper l'exécution de la méthode traitement par d'autres traitements avant et après son exécution.
Réponse : le Design pattern adéquate à cette situation est **Template Method**.



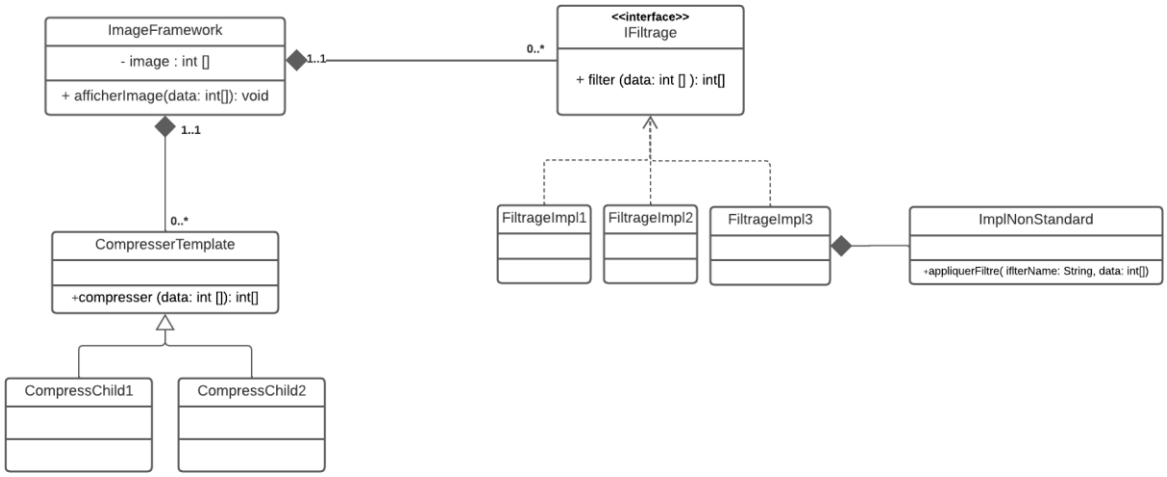
4- On désire créer une classe Joueur ayant un état représenté par une variable score de type int. On voudrait que les objets de l'environnement du jeu (Couloir, Caméra et Gardien) soient informés à chaque fois que le score du joueur change tout en gardant un couplage faible entre la classe Joueur et les autres classes.
Réponse : le Design pattern adéquate à cette situation est **Observer.**



Exercice 2 :

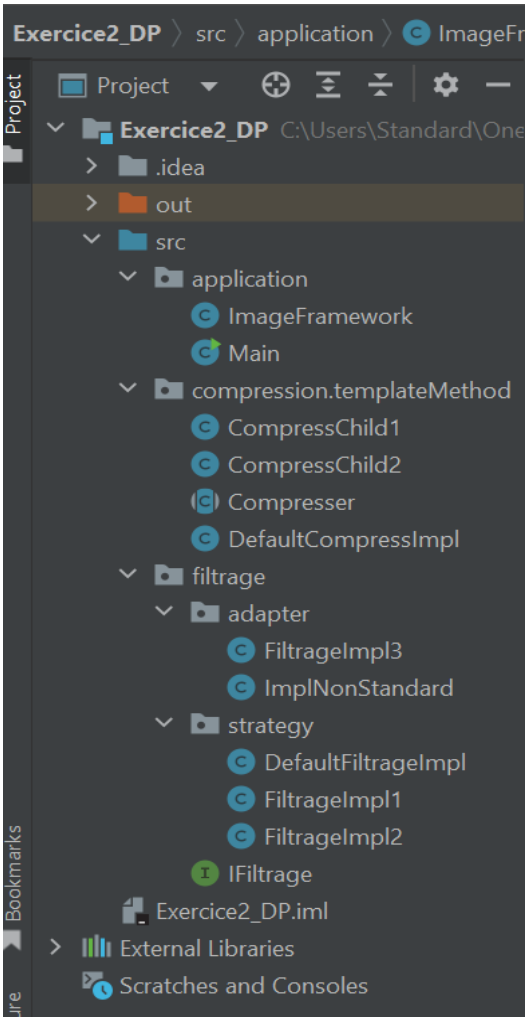
1- Le diagramme de classe proposée est comme suit :

J'ai utilisé dans ce cas 4 designs patterns : Singleton, Strategy, Adapter, Template Method



2- Le code source est rendu avec ce document.

Architecture du projet :



Interface IFiltrage :

```
package filtrage;

public interface IFiltrage {
     int[] filtrer (int [] data);
}
```

Implémentations :

```
public class FiltrageImpl1 implements IFiltrage {

    @Override
    public int[] filtrer(int[] data) {
        int[] imageFiltre = new int[data.length];

        for (int i = 0; i <data.length ; i++) {
            imageFiltre[i] = data[i]+2;
        }

        System.out.println("_____________________Application du filtre
sur l'image en utilisant l'implémentation 1_____________________");

        return  imageFiltre;
    }
}
```

```
public class FiltrageImpl2 implements IFiltrage {
    @Override
    public int[] filtrer(int[] data) {

        int[] imageFiltre =data;
        for (int i = 0; i <data.length ; i++) {
            imageFiltre[i] = data[i]+4;
        }
        System.out.println("_____________________Application du filtre
sur l'image en utilisant l'implémentation 2_____________________");

        return  imageFiltre;
    }
}
```

```
public class DefaultFiltrageImpl implements IFiltrage {

    @Override
    public int[] filtrer(int[] data) {

        int[] imageFiltre =data;
        for (int i = 0; i <data.length ; i++) {
            imageFiltre[i] = data[i]+6;
        }
        System.out.println("_____________________Application du filtre
sur l'image en utilisant le filtre par défaut_____________________");
```

```
        return  imageFiltre;
    }
}
```

```
public class FiltrageImpl3 implements IFiltrage {

    private ImplNonStandard implNonStandard = new ImplNonStandard();

    @Override
    public int[] filtrer(int[] data) {
        return implNonStandard.appliquerFiltre("FiltrageImpl3",data);
    }
}
```

```
public class ImplNonStandard {
    public int[] appliquerFiltre(String filterName, int[] data){
        int[] imageFiltre =data;
        for (int i = 0; i <data.length ; i++) {
            imageFiltre[i] = data[i]+8;
        }
        System.out.println("_____________________Application du filtre
sur l'image en utilisant l'ImplNonStandard_____________________");

        return  imageFiltre;
    }
}
```

Interface de compression:

```
public abstract class Compresser {
     public abstract int[] compresser (int [] data);
}
```

implémtations:

```
public class DefaultCompressImpl extends Compresser {
    @Override
    public int[] compresser(int[] data) {
        int[] imageCompresse = new int [data.length/2];

        for (int i = 0; i < imageCompresse.length; i++) {
            imageCompresse[i] = data[i];
        }

        System.out.println("_____________________________compression d'image
en utilisant la compression par défaut_____________________________");

        return imageCompresse;
    }
}
```

```
public class CompressChild1 extends Compresser {
    @Override
```

```
    public int[] compresser(int[] data) {


        int[] imageCompresse = new int [data.length/2];

        for (int i = 0; i < imageCompresse.length; i++) {
            imageCompresse[i] = data[i];
        }

        System.out.println("_____________________________compression d'image
en utilisant child 1_____________________________");

        return imageCompresse;
    }
}
```

```
public class CompressChild2 extends Compresser {
    @Override
    public int[] compresser(int[] data) {

        int[] imageCompresse = new int [(data.length/2)+(data.length/3)];
        for (int i = 0; i < imageCompresse.length; i++) {
            imageCompresse[i] = data[i];
        }

        System.out.println("_____________________________compression d'image
en utilisant child 2_____________________________");

        return imageCompresse;
    }
}
```

le framework "ImageFramework":

```
public class ImageFramework {
        private IFiltrage filtre;
        private Compresser compresser;

        public ImageFramework() {
            filtre =  new DefaultFiltrageImpl();
            compresser = new DefaultCompressImpl();
        }

        public void setFiltre(IFiltrage filtre) {
            this.filtre = filtre;
        }

        public IFiltrage getFiltre() {
            return filtre;
        }

        public Compresser getCompress() {
            return compresser;
        }

        public void setCompress(Compresser compress) {
```

```
            this.compresser = compress;
        }
    }
```