Design pattern

FlyWeight

MEMBRES

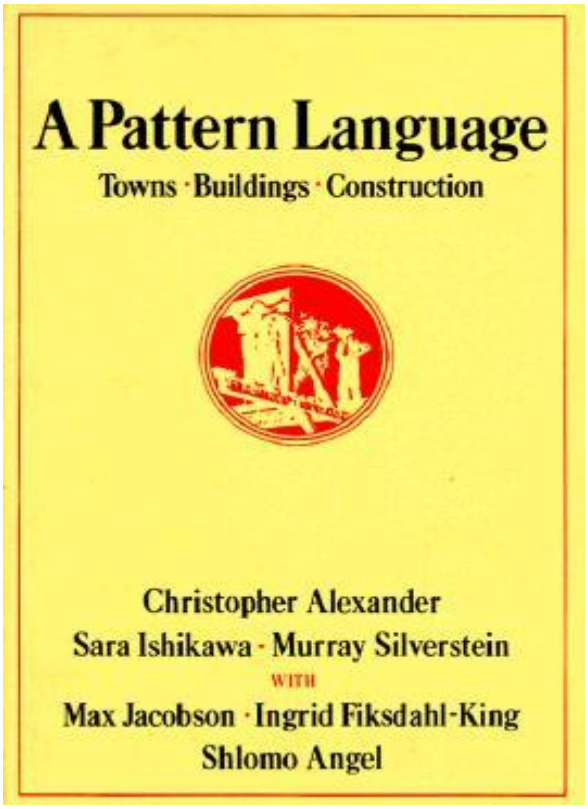
- MAMADOU ALPHA DIALLO
- SIRA ABDOULAYE DRAME
- HADJA AISSATOU BAH

PLAN

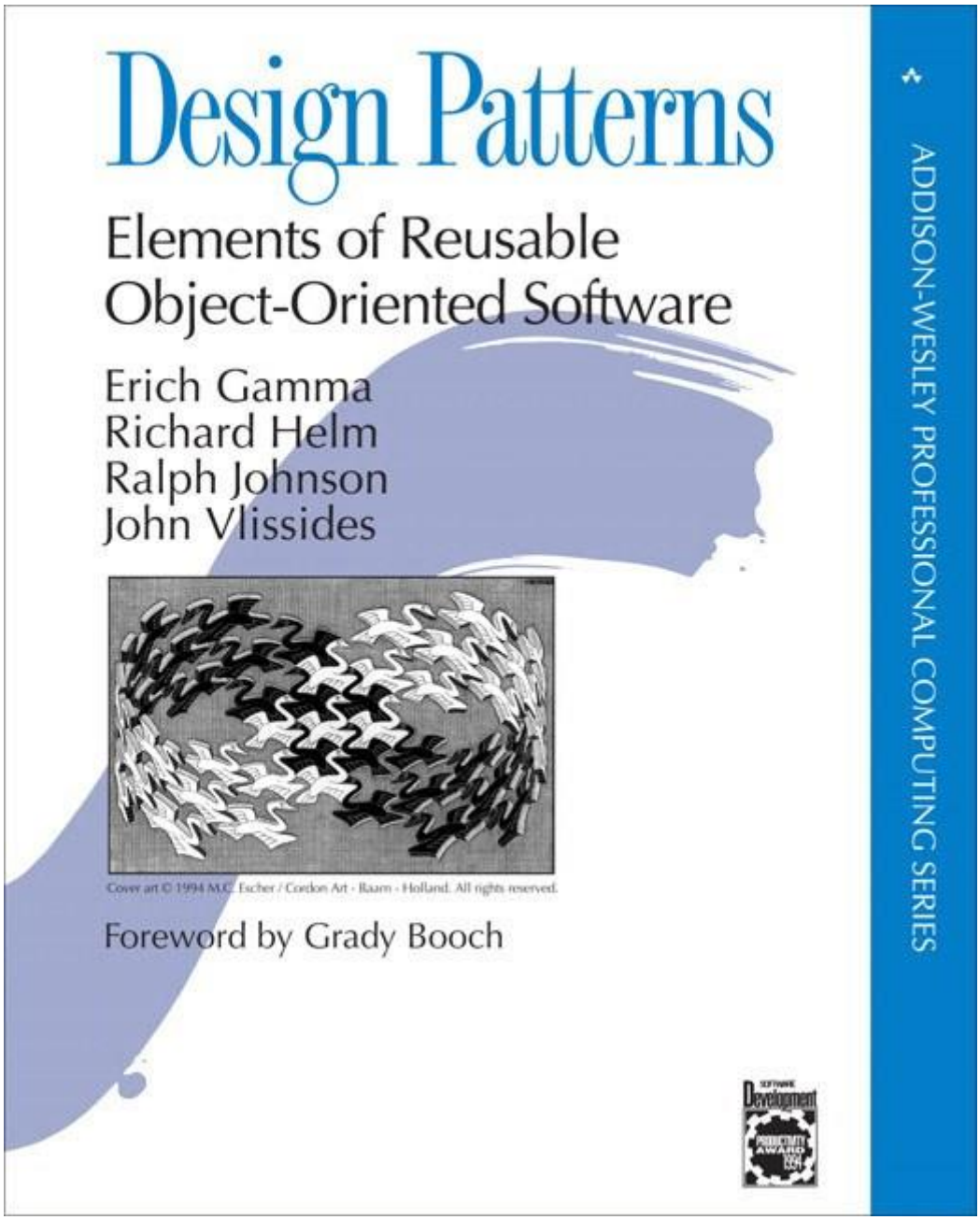
I. INTRODUCTION
II. DEFINITION
III. EXEMPLE DE FlyWeight
IV. CONCLUSION

I. INTRODUCTION

Un design pattern est un ensemble de bonnes pratiques ayant pour but de rendre le code plus propre, optimisé, robuste, maintenable et évolutif afin de répondre aux problèmes récurrents rencontré par les développeurs.edztgerhb La première apparition publique des design pattern vient d'un livre publié en 1977 par l'ingénieur Christopher Alexander nommé « A Pattern Language ».



En 1995 le Gang of four a présenté les 23 design pattern qui font aujourd'hui office de référence dans le domaine de l'informatique.



Il existe 3 types différents de design Pattern :

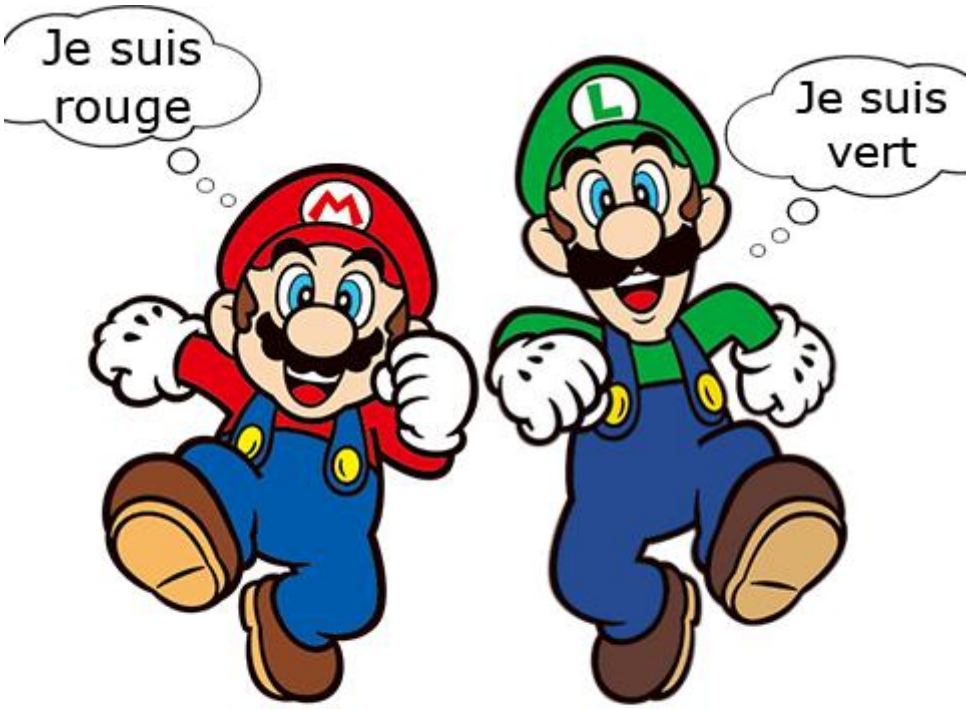
- Les Patterns de créations
- Les Patterns de structurations
- Les Patterns comportementaux

Dans notre cas le design pattern FlyWeight appartient au pattern de structuration.

II. DEFINITION

Souvent lors de l'utilisation d'un programme on se retrouve à instancier de nombreux objets, chacun prenant une certaine quantité de mémoire on peut donc se demander comment réduire la place des objets instanciés, on va donc utiliser le design pattern FlyWeight qui va en plus accélérer la vitesse d'exécution du programme.

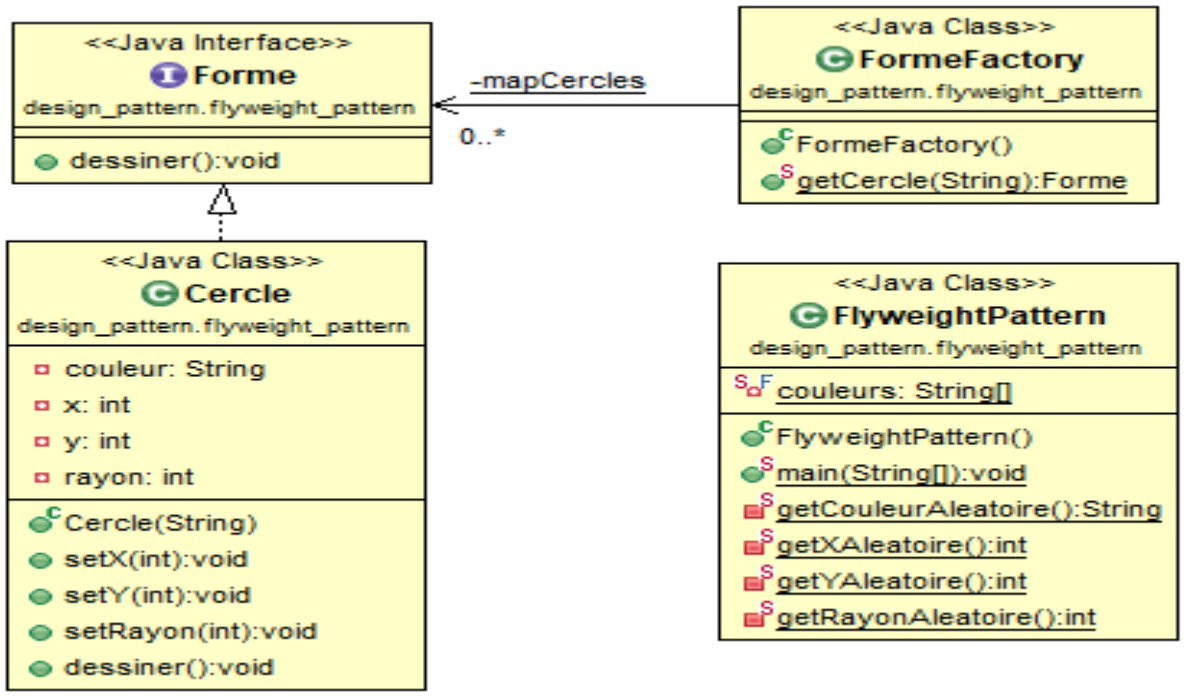
En effet, ce pattern, à l'aide d'une factory si deux objets différents ont un paramètre en commun (exemple deux cercle d'une même couleur mais d'une taille différente) on va utiliser ce paramètre déjà construit dans le premier afin de réaliser le deuxième ainsi la place en mémoire sera celle d'un seul cercle pour en créer deux, pour cela on va utiliser les setters de la classe cercle en résumé on va utiliser un type objet pour représenter une gamme de petits objets tous différents.



III. EXEMPLE

Diagramme de classe

Voici le diagramme de classe que nous allons implémenter en quatre étapes :



Première étape

Premièrement on crée la classe abstraite forme.

Deuxième étape

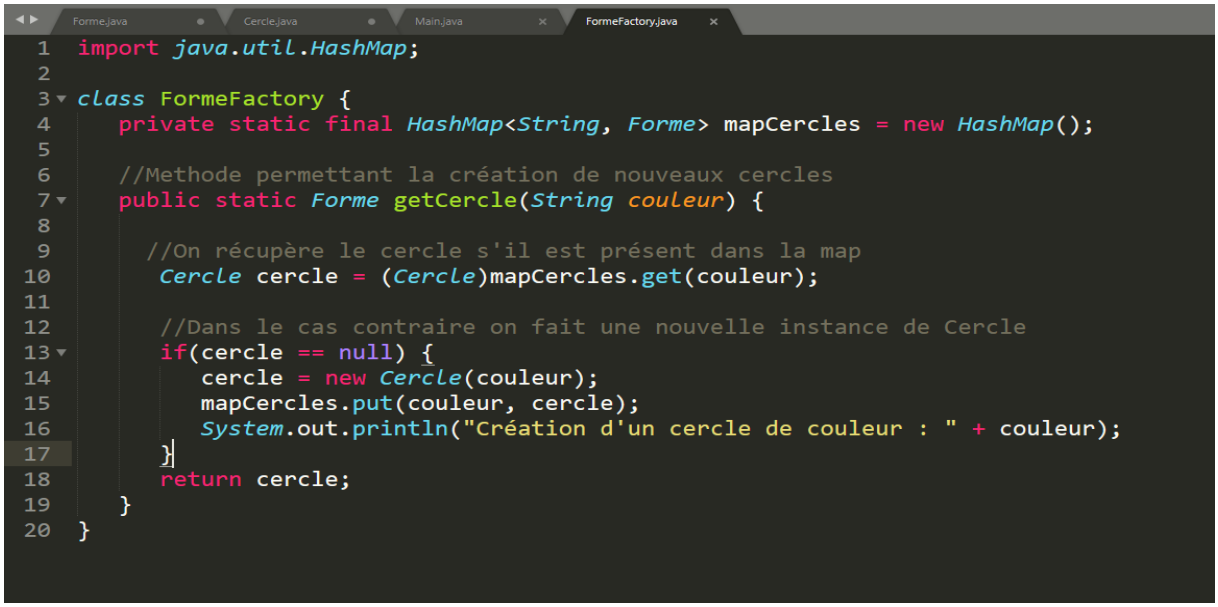
On crée ensuite la classe cercle où on définit les paramètres ainsi que les setters :

Forme.java | Cercle.java | Main.java | FormeFactory.java

```java
class Cercle implements Forme {
        private String couleur;
        private int x;
        private int y;
        private int rayon;

        //Constructeur du cercle
        public Cercle(String couleur){
            this.couleur = couleur;
        }

        // X et Y sont les coordonnées du centre du cercle

        //Setter de X
        public void setX(int x) {
            this.x = x;
        }

        //Setter de Y
        public void setY(int y) {
            this.y = y;
        }

        //Setter du rayon
        public void setRayon(int rayon) {
            this.rayon = rayon;
        }

        //Methode de l'affichage du cercle
        public void dessiner() {
            System.out.println("Cercle: Dessiner() [Couleur : " + this.couleur + ", x : "
              + this.x + ", y :" + this.y + ", Rayon :" + this.rayon);

        }

    }
```

Troisième étape

On crée ensuite la class FormeFactory qui permet de stocker dans une map les cercles crées et de récupérer de cette même map un cercle déjà existant :



```java
import java.util.HashMap;

class FormeFactory {
    private static final HashMap<String, Forme> mapCercles = new HashMap();

    //Methode permettant la création de nouveaux cercles
    public static Forme getCercle(String couleur) {

      //On récupère le cercle s'il est présent dans la map
       Cercle cercle = (Cercle)mapCercles.get(couleur);

       //Dans le cas contraire on fait une nouvelle instance de Cercle
       if(cercle == null) {
          cercle = new Cercle(couleur);
          mapCercles.put(couleur, cercle);
          System.out.println("Création d'un cercle de couleur : " + couleur);
       }
       return cercle;
    }
}
```

Quatrième étape

On implémente ensuite le main afin de créer 20 cercles de couleurs et dimensions aléatoires :

```java
public class Main
{
      // Méthode principale.
      public static void main(String[] args)
      {
         String couleurs[] = { "Rouge", "Vert", "Bleu", "Blanc", "Noir" };
         //Création de 20 Cercles
         for (int i = 0; i < 20; ++i) {
             Cercle cercle = ( Cercle ) FormeFactory.getCercle(couleurs[(int)(Math.random()
             cercle.setX((int)(Math.random() * 100));
             cercle.setY((int)(Math.random() * 100));
             cercle.setRayon((int)((Math.random() * 99)+1));
             cercle.dessiner();
         }
      }
}
```

Apres exécution du code

Création d'un cercle de couleur : Blanc
Cercle: Dessiner() [Couleur : Blanc, x : 94, y :7, Rayon :45
Création d'un cercle de couleur : Bleu
Cercle: Dessiner() [Couleur : Bleu, x : 92, y :92, Rayon :92
Cercle: Dessiner() [Couleur : Bleu, x : 62, y :58, Rayon :86
Cercle: Dessiner() [Couleur : Blanc, x : 12, y :49, Rayon :34
Cercle: Dessiner() [Couleur : Blanc, x : 84, y :57, Rayon :63
Création d'un cercle de couleur : Rouge
Cercle: Dessiner() [Couleur : Rouge, x : 43, y :28, Rayon :18
Cercle: Dessiner() [Couleur : Bleu, x : 20, y :84, Rayon :22
Cercle: Dessiner() [Couleur : Rouge, x : 97, y :26, Rayon :82
Création d'un cercle de couleur : Noir
Cercle: Dessiner() [Couleur : Noir, x : 79, y :70, Rayon :9
Cercle: Dessiner() [Couleur : Rouge, x : 68, y :62, Rayon :55
Cercle: Dessiner() [Couleur : Bleu, x : 31, y :47, Rayon :79
Cercle: Dessiner() [Couleur : Bleu, x : 29, y :41, Rayon :12
Cercle: Dessiner() [Couleur : Rouge, x : 31, y :99, Rayon :92
Cercle: Dessiner() [Couleur : Bleu, x : 5, y :76, Rayon :30
Cercle: Dessiner() [Couleur : Bleu, x : 70, y :52, Rayon :65
Création d'un cercle de couleur : Vert
Cercle: Dessiner() [Couleur : Vert, x : 17, y :54, Rayon :34
Cercle: Dessiner() [Couleur : Vert, x : 6, y :52, Rayon :60
Cercle: Dessiner() [Couleur : Rouge, x : 3, y :74, Rayon :3
Cercle: Dessiner() [Couleur : Noir, x : 66, y :9, Rayon :12
Cercle: Dessiner() [Couleur : Vert, x : 66, y :81, Rayon :78

IV. CONCLUSION

L’utilisation principale du design pattern FlyWeight est d’optimiser la place mémoire occupée par les objets afin d’accélérer l’exécution du programme plusieurs objets sont instanciés grâce à la même méthode est ainsi la place en mémoire est réduite par rapport à une implémentation plus classique.