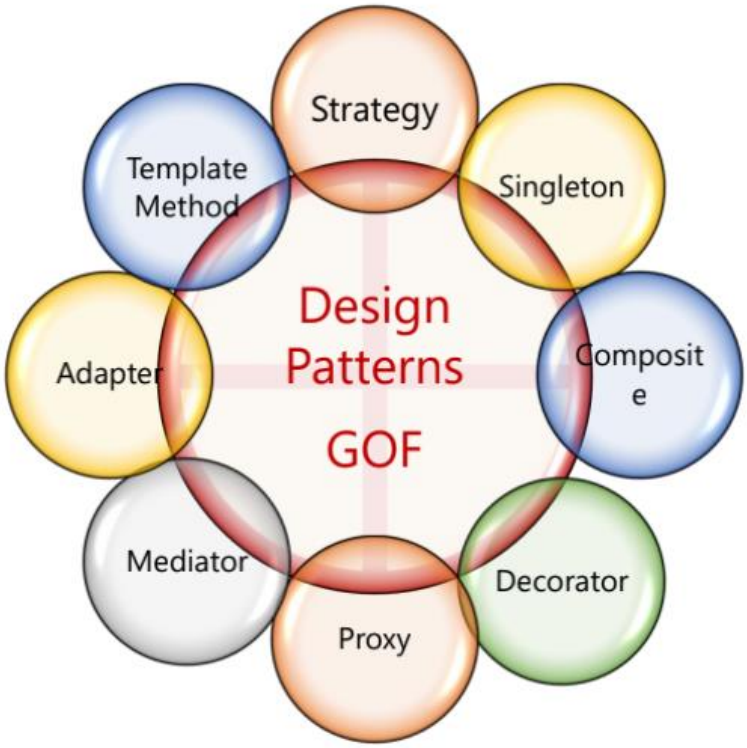
جـامعـة الحـسن الثـاني بالدار البيـضـاء
UNIVERSITÉ HASSAN II DE CASABLANCA

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

Strategy Pattern



Réalisé par : REGOUG Zakia

GLSID3

Année universitaire : 2023-2024

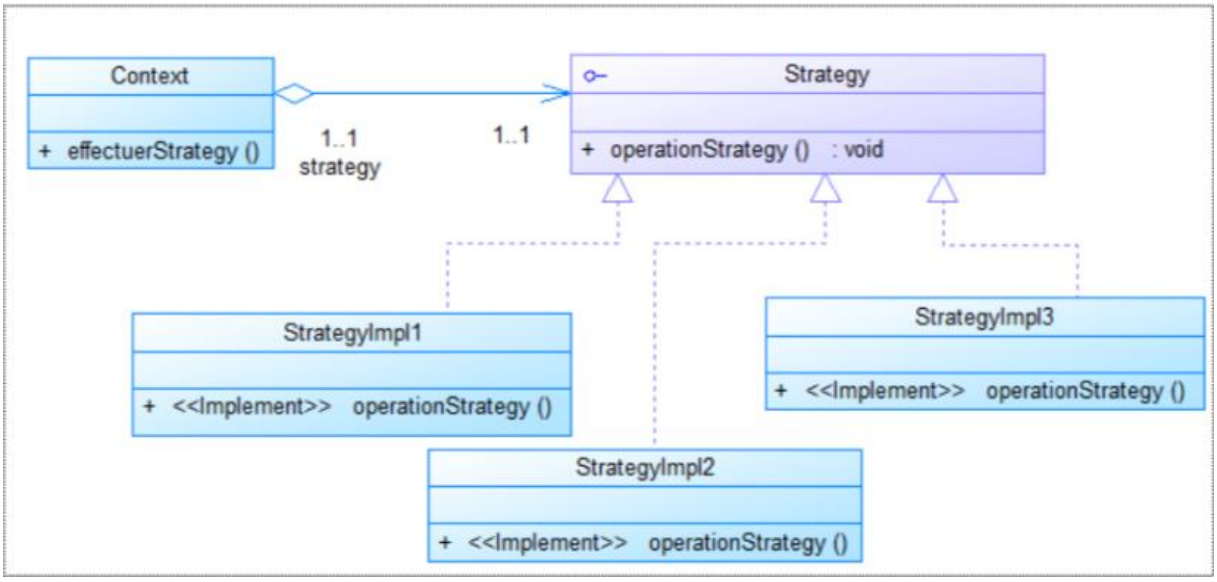
Catégorie :

Comportement

Objectif :

Définir une famille d'algorithmes, et encapsuler chacun et les rendre interchangeables tout en assurant que chaque algorithme puisse évoluer indépendamment des clients qui l'utilisent

Diagramme de Classe :



Note : le pattern strategy doit avoir une seul méthode

Exemple d'application :

Classe context :

```
public class Context {
    2 usages
    private Strategy strategy=new DefaultStrategyImpl();
    1 usage
    public void effectuerStrategy(){
        System.out.println("******************");
        strategy.operation();
        System.out.println(".....................");
    }

    1 usage
    public void setStrategy(Strategy strategy) { this.strategy = strategy; }
}
```

Interface Strategy :

```
8 usages  4 implementations
public interface Strategy {
    1 usage  4 implementations
    void operation();
}
```

Classe StrategyImpl1 :

```
public class StrategyImpl1 implements Strategy {
    1 usage
    @Override
    public void operation() {
        System.out.println("------ Startegy 1 ------");
    }
}
```

Main :

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) throws Exception {
        Context context=new Context();
        Scanner scanner = new Scanner(System.in);
        while(true){
            System.out.println("Preciser strategy");
            String strategyClass=scanner.nextLine();
            Strategy strategy=(Strategy) Class.forName( className: "StrategyImpl"+strategyClass).getConstructor().newInstance();
            context.setStrategy(strategy);
            context.effectuerStrategy();
        }

    }
}
```

Test :

```
Main
C:\Users\Zakia\.jdks\corretto-17.0.6\bin\java.exe "
Preciser strategy
1
********************
------ Startegy 1 ------
......................
Preciser strategy
2
********************
------ Startegy 2 ------
......................
Preciser strategy
3
********************
------ Startegy 3 ------
......................
```

Inconvénient :

Il génère la création de plusieurs objets dans la mémoire

Pour résoudre ce problème dans l'exemple précèdent il peut utiliser le cache Map :

```
public class Main {
    public static void main(String[] args) throws Exception {
        Context context=new Context();
        Scanner scanner = new Scanner(System.in);
        //declarer le cache map
        Map<String,Strategy> strategyMap=new HashMap<>();
        Strategy strategy;
        while(true){
            System.out.println("Precise strategy : ");
            String str=scanner.nextLine();
            strategy=strategyMap.get(str);
            //si la strategie n'existe pas dans la memoire
            if(strategy==null){
                //l'instancier
                System.out.println("Creation de strategy");
                strategy=(Strategy) Class.forName( className: "StrategyImpl"+str).getConstructor().newInstance();
                //ajouter dans le cache
                strategyMap.put(str,strategy);
            }
            context.setStrategy(strategy);
            context.effectuerStrategy();
        }
```

Comme vous voyez le programme crée la stratégie seulement s'il n'existe pas dans le cache

```
Main
C:\Users\Zakia\.jdks\corretto-17.0.6\bin\java.exe "-
Precise strategy :
1
Creation de strategy
******************
------ Startegy 1 ------
.......................
Precise strategy :
1
******************
------ Startegy 1 ------
.......................
Precise strategy :
2
Creation de strategy
******************
------ Startegy 2 ------
.......................
Precise strategy :
```

Exercice d'application :

On considère la classe Employe (voir annexe) qui est définie par :

- deux variables d'instance cin et salaireBrutMensuel,
- deux constructeurs, les getters et setters
- et une méthode calculerIGR qui retourne l'impôt général sur les revenus salariaux.
- La méthode getSalaireNetMensuel retourne le salaire net mensuel.

Supposant que la formule de calcul de l'IGR diffère d'un pays à l'autre.

Au Maroc, par exemple le calcul s'effectue selon les cas suivant :

- Si le salaire annuel est inférieur à 40000, le taux de l'IGR est : 5%
- Si le salaire annuel est supérieur à 40000 et inférieur à 120000, le taux de l'IGR est : 20%
- Si le salaire annuel est supérieur à 120000 le taux de l'IGR est : 42%

En Algérie, le calcul s'effectue en utilisant un taux unique de 35%.

Comme cette classe est destinée à être utilisée dans différent type de pays inconnus au moment du développement de cette classe,

1. Identifier les méthodes qui vont subir des changements chez le client.
2. En appliquant le pattern strategie, essayer de rendre cette classe fermée à la modification et ouverte à l'extension.
3. Créer une application de test.
4. Proposer une solution pour choisir dynamiquement l'implémentation de calcul de l'IGR.

1- Changement de la méthode

La méthode qui va subir des changements est la méthode calculer IGR parce qu'il change lorsque le taux est changé.

2- Application de pattern Strategy :

Employe :

```
public class Employe {
    private String cin;
    private float salaireBrutMensuel;

    //Ajouter attribut taux
    private Taux taux;
    public Employe(String cin, float salaireBrutMensuel) {

        this.cin = cin;
        this.salaireBrutMensuel = salaireBrutMensuel;

    }
    public Employe() {
    }

    public float calculerIGR(){

        float salaireBrutAnuel=salaireBrutMensuel*12;
        float tx=taux.calulerTaux(salaireBrutMensuel);
        return salaireBrutAnuel*tx/100;

    }
    public float getSalaireNetMensuel(){
        float igr=calculerIGR();
        float salaireNetAnuel=salaireBrutMensuel*12-igr;
        return salaireNetAnuel/12;
```

```
    }

    public String getCin() {
        return cin;
    }

    public void setCin(String cin) {
        this.cin = cin;
    }

    public float getSalaireBrutMensuel() {
        return salaireBrutMensuel;
    }

    public void setSalaireBrutMensuel(float salaireBrutMensuel) {
        this.salaireBrutMensuel = salaireBrutMensuel;
    }

    public void setTaux(Taux taux) {
        this.taux = taux;
    }
}
```

Taux

```
public interface Taux {
    float calulerTaux(float salaireBrutMensuel);
}
```

Taux Maroc :

```
public class TauxMaroc implements Taux {

    @Override
    public float calulerTaux(float salaireBrutMensuel) {
        float salaireBrutAnuel=salaireBrutMensuel*12;
        if(salaireBrutAnuel<40000) return 5;
        else if (salaireBrutAnuel>40000 && salaireBrutAnuel<12000) {
            return 20;
        }
        else return 42;
    }
}
```

Taux algerie

```
public class TauxAlgerie implements Taux {

    @Override
    public float calulerTaux(float salaireBrutMensuel) {
        return 35;
    }
}
```

3- Test :

```
4- import java.util.Scanner;

   public class Main {
       public static void main(String[] args) throws Exception {
           Scanner scanner=new Scanner(System.in);
           Employe employe=new Employe();
           System.out.println("Entrez cin :");
           employe.setCin(scanner.nextLine());
           System.out.println("Entrez Salaire mensuel :");
           employe.setSalaireBrutMensuel(scanner.nextFloat());
           Taux taux;
           while(true){
               System.out.println("Entrez Pays :");
               scanner.nextLine();
               String pays=scanner.nextLine();
               System.out.println(pays);
               taux=(Taux)
   Class.forName("Taux"+pays).getConstructor().newInstance();
               employe.setTaux(taux);
               System.out.println("Le salaire net mensuel est :
   "+employe.getSalaireNetMensuel());
           }
       }
   }
```

```
C:\Users\Zakia\.jdks\corretto-17.0.6\bi
Entrez cin :
f67780
Entrez Salaire mensuel :
45000
Entrez Pays :
Maroc
Maroc

Le salaire net mensuel est : 26100.0
Entrez Pays :
Algerie
Algerie
Le salaire net mensuel est : 29250.0
```