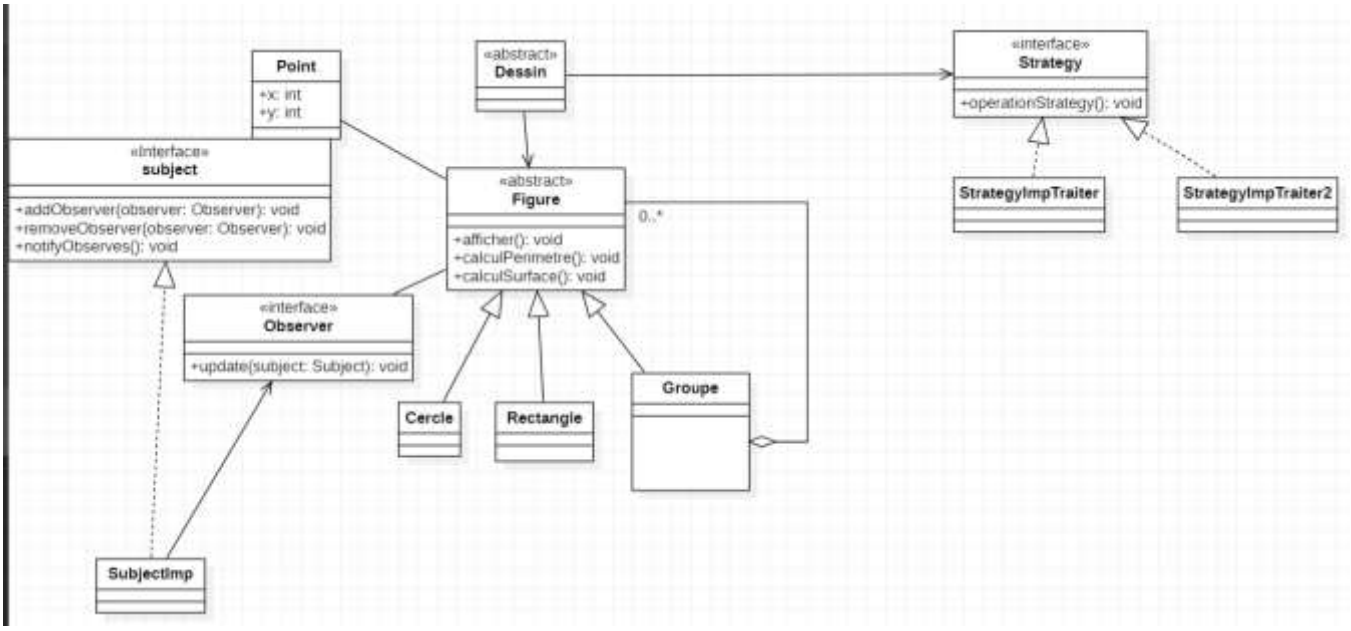
Examen Blanc Design Pattern et Programmation Orientée Aspect

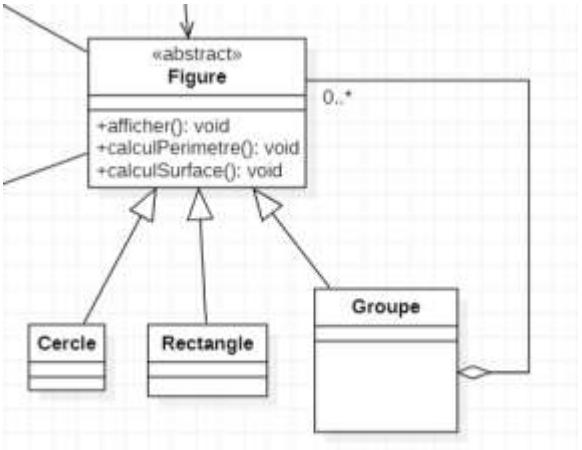
Durée : 3H | Documents Autorisés

1. Etablir un Diagramme de classe du modèle en appliquant les design patterns appropriés en justifiant les designs patterns appliqués.



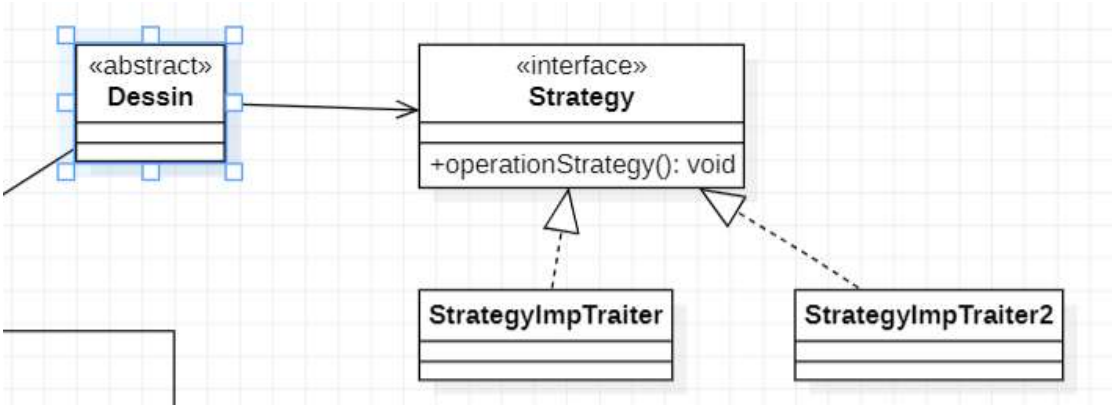
Les designs patterns appliqués :

Pattern Composite : il s'agit d'une arborescence, une figure peut être soit un rectangle, un cercle, ou un groupe de figures



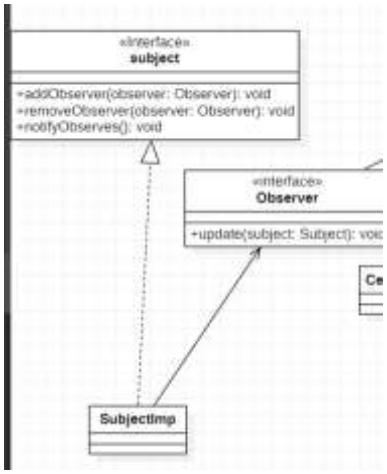
Pattern Strategy :

Nous avons définie t une famille d'algorithmes qui sont interchangeables dynamiquement .



Pattern Observer :

Il s'agit d'une relation de façon que, lorsqu'un objet change d'état, tous ce qui en dépendent en soient informés et soient mis à jour automatiquement



2. Faire une implémentation du modèle en utilisant un projet Maven sans prendre en considération des aspects techniques.

   Class Point

```
public class Point {
    private int x ;
    private int y;

    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public int getX() {
        return x;
    }

    public int getY() {
        return y;
    }

    public void setX(int x) {
```

```
        this.x = x;
    }

    public void setY(int y) {
        this.y = y;
    }
}
```

Class Dessin

```
public abstract class Dessin {
    private List<Figure> figures = new LinkedList<>();
    private Strategy strategy;
    public void traiter(){
        strategy.operationStrategy();
    }
    public void setStrategy(Strategy strategy) {
        this.strategy = strategy;
    }
    public void addFigure(Figure figure){
        this.figures.add(figure);
    }
    public void removeFigure(Figure figure){
        figures.remove(figure);
    }
}
```

Class Figure

```
3. public abstract class Figure extends Dessin implements Observer {

    protected Point point;
    public abstract void afficher();
    public abstract double calculPerimetre();
    public abstract double calculSurface();

    @Override
    public void update(Subject subject) {
        int color = ((SubjectImpl)subject).getColor();
        int epaisseur = ((SubjectImpl)subject).getEpaisseur();
        int remplissage = ((SubjectImpl)subject).getRemplissage();
        System.out.println("color was : "+color+" && epaisseur was :
"+epaisseur+" && remplissage was : "+remplissage);
    }
}
4.
```

interface Observer

```
public interface Observer {
    public void update(Subject subject);
}
```

class SubjectImpl

```
public class SubjectImpl implements Subject {
    private final List<Observer> observers = new ArrayList<>();
    private int color;
    private int remplissage ;
    private int epaisseur ;

    @Override
    public void addObserver(Observer observer) {
        observers.add(observer);
```

```
    }

    @Override
    public void removeObserver(Observer observer) {
        observers.remove(observer);
    }

    @Override
    public void notifyObserves() {
        for(Observer o:observers){
            o.update(this);
        }
    }

    public int getColor() {
        return color;
    }

    public int getRemplissage() {
        return remplissage;
    }

    public int getEpaisseur() {
        return epaisseur;
    }

    public void setColor(int color) {
        this.color = color;
        notifyObserves();
    }

    public void setRemplissage(int remplissage) {
        this.remplissage = remplissage;
        notifyObserves();
    }

    public void setEpaisseur(int epaisseur) {
        this.epaisseur = epaisseur;
        notifyObserves();
    }
}
```

Class Cercle

```
5. public class Cercle extends Figure {

    private int r ;

    public Cercle(int r , int color , int epaisseur, int
remplissage){
        this.r = r;
    }

    @Override
    public void afficher() {
        System.out.println("Dessin cercle");
    }

    @Override
    public double calculPerimetre() {
        return 3.14*2*r;
    }

    @Override
    public double calculSurface() {
```

```
        return 3.14*r*r;
    }
6.
}
7.
```

class Rectangle

```
public class Rectangle extends Figure {

    private int h ;
    private int l ;

    public Rectangle(int  l , int h){
        this.l = l ;
        this.h = h ;
    };

    @Override
    public void afficher() {
        System.out.println("Dessin Rectangle");
    }

    @Override
    public double calculPerimetre() {
        return 2*l*h;
    }

    @Override
    public double calculSurface() {
        return l*h;
    }


}
```

Class group

```
public class Group extends Figure {
    private List<Figure> figures = new LinkedList<>();

    @Override
    public void afficher() {
        for (Figure f : figures){
            System.out.println("Dessin de figure : "+f);
        }
    }

    @Override
    public double calculPerimetre() {
        double p = 0 ;
        for (Figure f : figures){
            p+= f.calculPerimetre();
        }
        return p;
    }

    @Override
    public double calculSurface() {
```

```
        double s = 0 ;
        for (Figure f : figures){
            s+= f.calculSurface();
        }
        return s;
    }
}
```

```
public class StrategyImplTraiter implements Strategy {
    @Override
    public void operationStrategy() {

        System.out.println("Traiter  hajar TAJAYOUTI VERSION 1 ");
    }
}
```

```
public class StrategyImplTraiter2 implements Strategy {
    @Override
    public void operationStrategy() {

        System.out.println("Traiter  hajar TAJAYOUTI VERSION 2 ");
    }
}
```