DEPARTEMENT MATHEMATIQUES ET INFORMATIQUE

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

Compte Rendu

Examen Blanc Design Pattern et Programmation Orientée Aspect

Réalisé par :

Mohcine AHADJANE

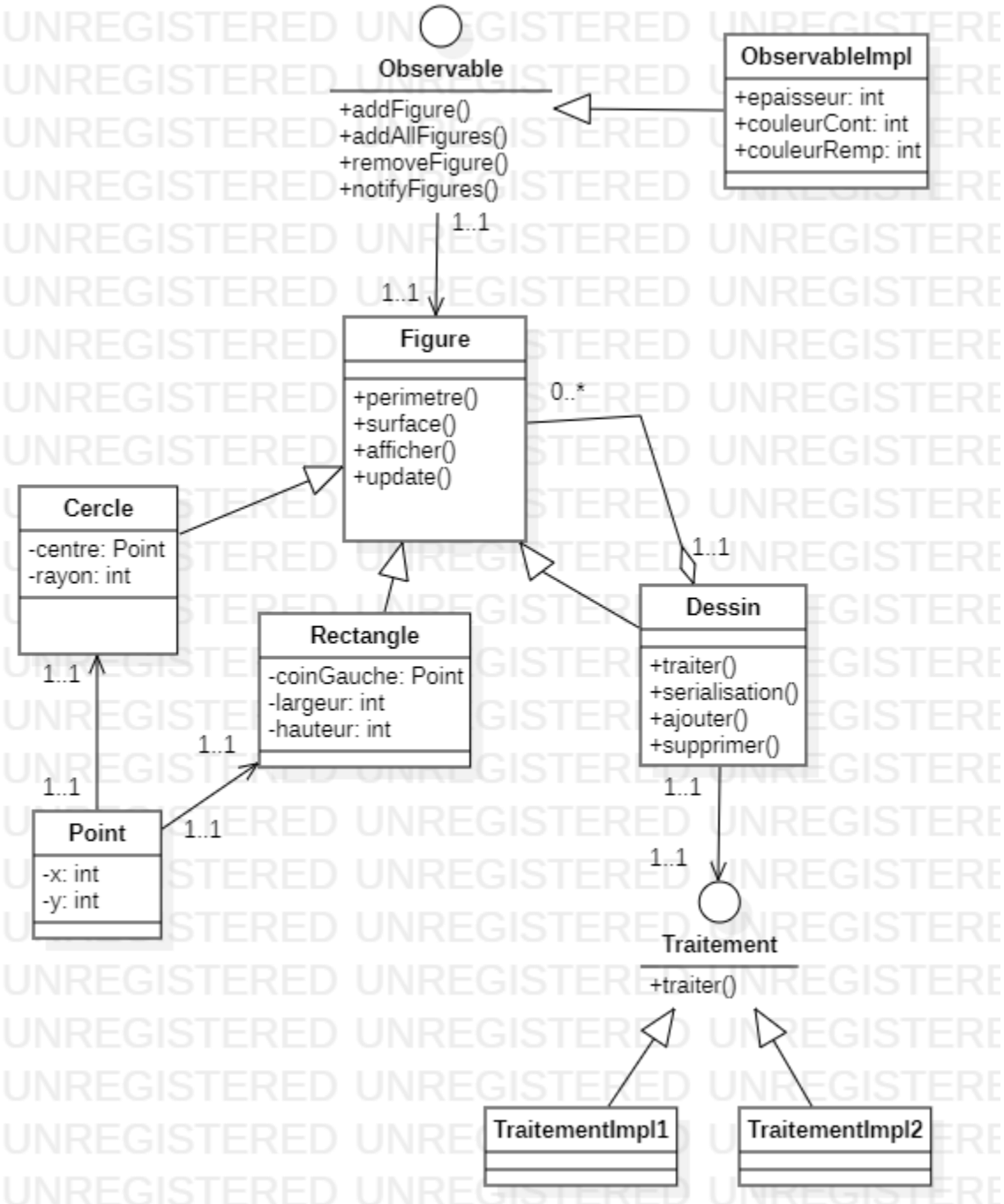
Encadré par :

Mohamed YOUSSFI

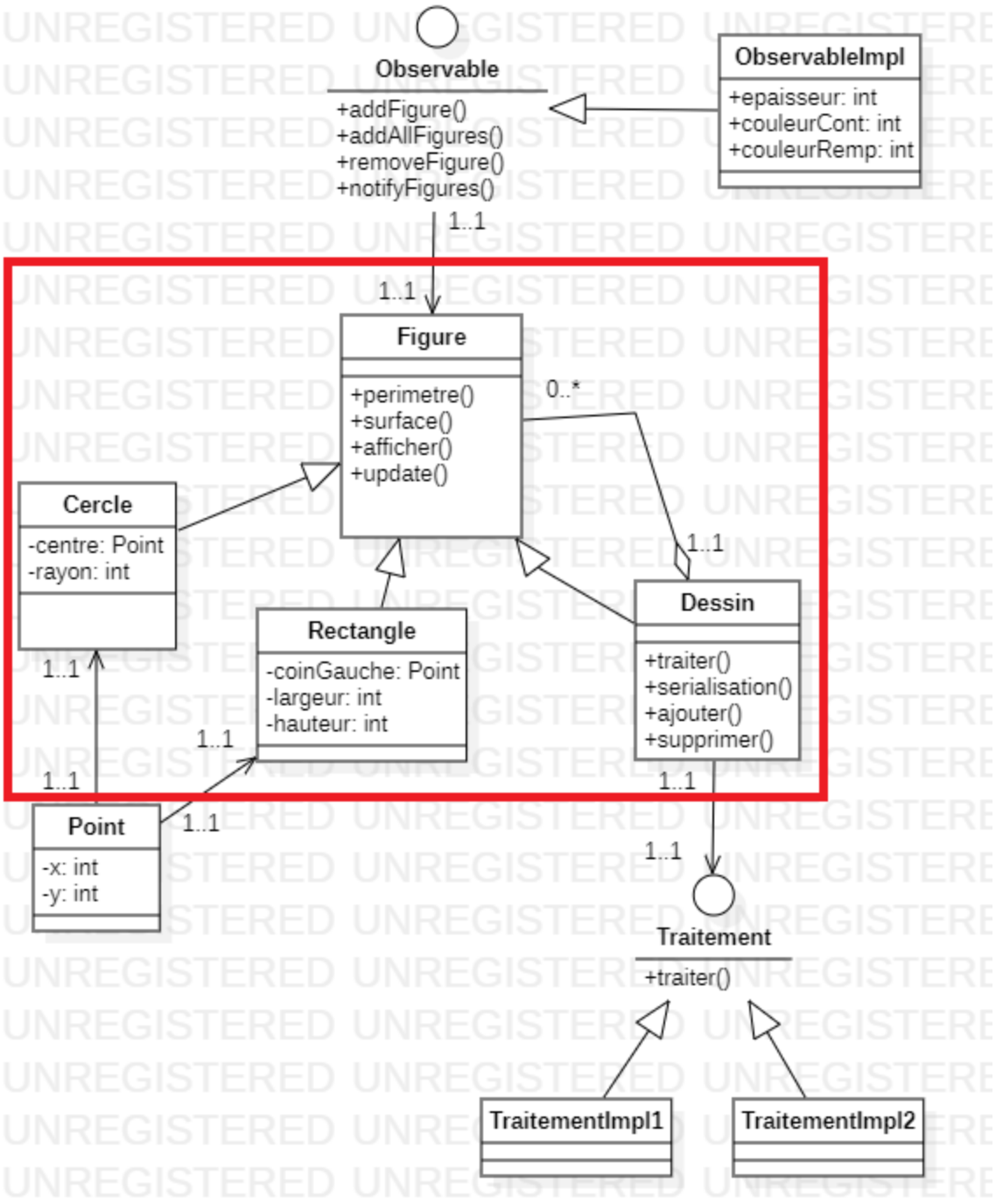
Année Universitaire : 2022-2023

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

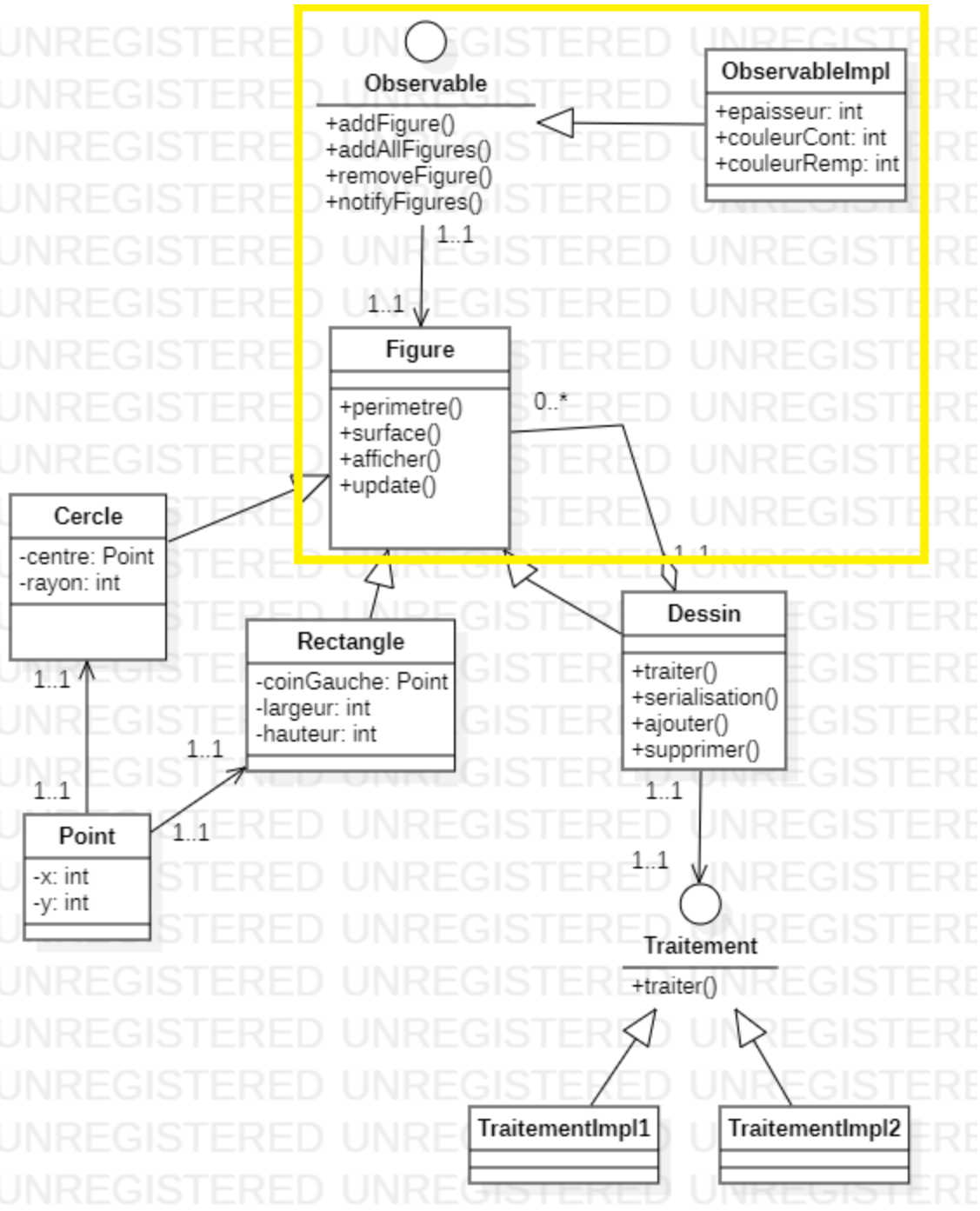
Diagramme de classe



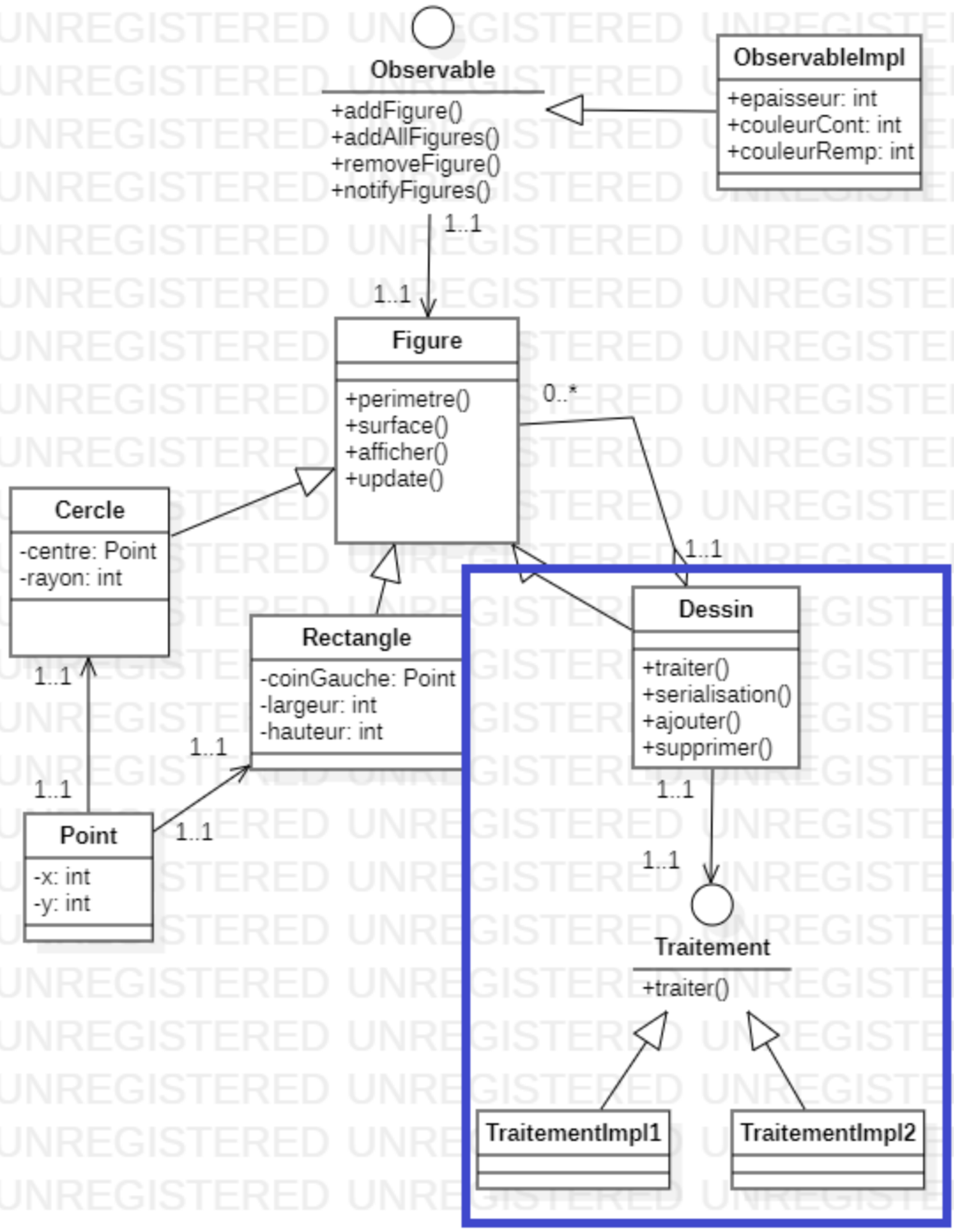
Pour la figure rectangle, cercle et dessin j'ai utilisé le pattern **Composite**



Pour l'objet de paramétrage, j'ai utilisé le pattern **Observer**

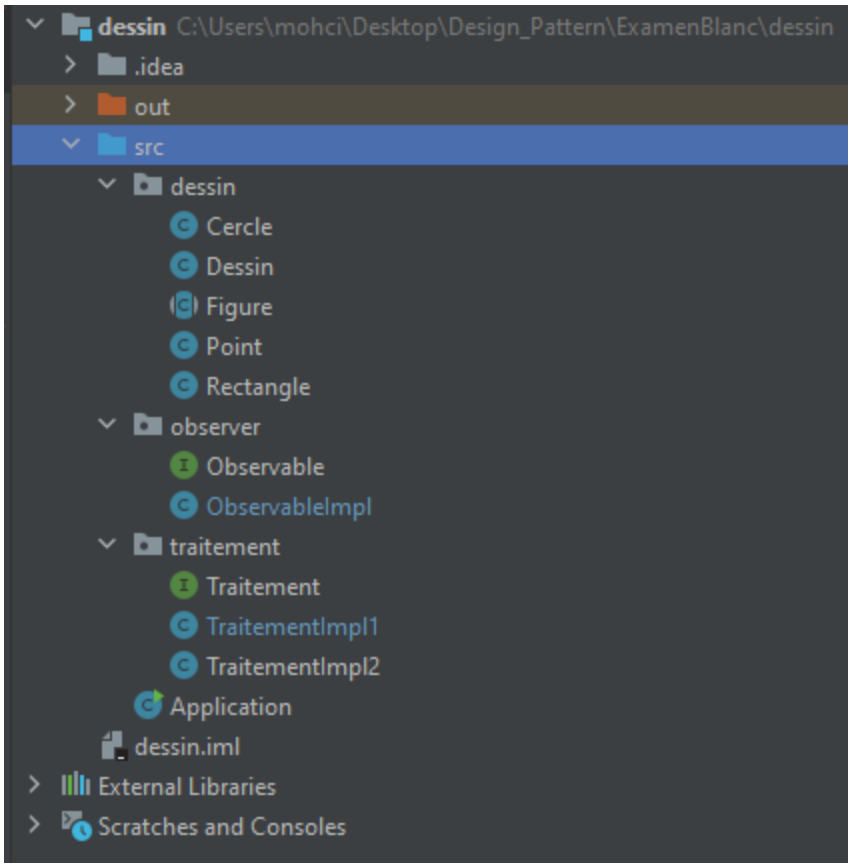


Pour la méthode traitement, j'ai utilisé le pattern **Strategy**



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

Structure du projet



Cercle

```java
package dessin;

import traitement.Traitement;

public class Cercle extends Figure{
    private Point centre =new Point();
    private double rayon ;
    private Traitement traitement ;

    public Cercle(int x, int y, double rayon) {
        this.centre = new Point(x,y);
        this.rayon = rayon;
    }
    public Cercle() {

    }
    public double perimetre() {
        return rayon*3.14;
    }

    @Override
    public double surface() {
        return 2*3.14*rayon;
```

```
    }

    @Override
    public void dessigner() {

    }
    @Override
    public void afficher() {
        System.out.println("************************");
        System.out.println("La figure est une circle de centre (" +centre.x+
","+centre.y+") et de rayon "+rayon);
        System.out.println("le perimetre est : "+ perimetre());
        System.out.println("la surface  est : "+ surface());
    }

}
```

Dessin

```
package dessin;

import traitement.Traitement;

import java.util.ArrayList;
import java.util.List;

public class Dessin extends Figure{

    List<Figure> figures = new ArrayList<>();
    private Traitement traitement ;

    public Dessin(List<Figure> figures) {
        this.figures = figures;
    }
    public Dessin() {

    }
    public List<Figure> getFigures() {
        return figures;
    }

    public void setFigures(List<Figure> figures) {
        this.figures = figures;
    }



    @Override
    public double perimetre() {
        return 0;
    }

    @Override
    public double surface() {
        return 0;
    }
```

```
    @Override
    public void dessigner() {

    }

    public Dessin(List<Figure> figures, Traitement traitement) {
        this.figures = figures;
        this.traitement = traitement;
    }

    public Traitement getTraitement() {
        return traitement;
    }

    public void setTraitement(Traitement traitement) {
        this.traitement = traitement;
    }

    public void ajouterDessin(Figure figure){
        figures.add(figure);
    }
    public void SupprimerDessin(Figure figure){
        figures.remove(figure);
    }
    public void serialisation () {
        System.out.println("serialisation du dessin");
    }
    public void traiter() {
        traitement.traiter();
    }

    @Override
    public void afficher() {
        System.out.println("************* Le dessin ********************");
        for (int i = 0; i < figures.size(); i++) {
            figures.get(i).afficher();
        }
    }
}
```

Figure

```
package dessin;

import observer.Observable;
import observer.ObservableImpl;

public abstract class Figure
{
    int epaisseur;
    int couleurCont ;
    int couleurRemp;
    Observable observable ;
```

```
    public  void update (Observable observable){
        couleurCont =((ObservableImpl) observable).getCouleurCout();
        epaisseur =((ObservableImpl) observable).getEpaisseur(); ;
        couleurRemp = ((ObservableImpl) observable).getCouleurRemp();
    }
    public  abstract double perimetre();
    public  abstract double surface();
    public  abstract void dessigner();

    public abstract  void afficher() ;
}
```

Point

```
package dessin;

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

    public Point() {

    }
}
```

Rectangle

```
package dessin;

public class Rectangle extends Figure{

    private Point coinGauche =new Point();
    private double largeur ;
    private double hauteur ;

    public Rectangle(Point coinGauche, double l, double h) {
        this.coinGauche = coinGauche;
        largeur = l;
        hauteur = h;
    }
    public Rectangle() {

    }

    public Point getCoinGauche() {
        return coinGauche;
    }

    public double getLargeur() {
        return largeur;
```

```
    }

    public double getHauteur() {
        return hauteur;
    }

    public void setCoinGauche(Point coinGauche) {
        this.coinGauche = coinGauche;
    }

    public void setLargeur(double largeur) {
        this.largeur = largeur;
    }

    public void setHauteur(double hauteur) {
        this.hauteur = hauteur;
    }

    @Override
    public double perimetre() {
        return largeur*hauteur;
    }

    @Override
    public double surface() {
        return 2*(largeur+hauteur);
    }

    @Override
    public void dessigner() {

    }


    @Override
    public void afficher() {
        System.out.println("-------------------");
        System.out.println("La figure est un rectangle de Largeur "+largeur+"
et d'hauteur "+hauteur);
        System.out.println("Le perimetre est :"+ perimetre());
        System.out.println("La surface  est :"+ surface());

    }


}
```

Observable

```
package observer;

import dessin.Figure;

import java.util.List;

public interface Observable {
```

```
    public void addFigure(Figure figure) ;
    public void addAllFigures(List<Figure> figures);
    public void removeFigure(Figure figure) ;
    public void notifyFigures() ;

}
```

ObservableImpl

```
package observer;

import dessin.Figure;

import java.util.ArrayList;
import java.util.List;

public class ObservableImpl implements Observable {

    private  int epaisseur;
    private int couleurCont;
    private int couleurRemp;
    List<Figure> figures = new ArrayList<>();

    @Override
    public void addFigure(Figure figure) {
        figures.add(figure);
    }

    @Override
    public void addAllFigures(List<Figure> figures) {
        this.figures.addAll(figures);
    }

    @Override
    public void removeFigure(Figure figure) {
        figures.remove(figure);
    }

    @Override
    public void notifyFigures() {
        for (Figure o : figures) {
            o.update(this);
        }
    }
    public int getEpaisseur() {
        return epaisseur;

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

    public int getCouleurCout() {
        return couleurCont;
```

```
    }

    public void setCouleurCout(int couleurCout) {
        this.couleurCont = couleurCout;
        notifyFigures();
    }

    public int getCouleurRemp() {
        return couleurRemp;
    }

    public void setCouleurRemp(int couleurRemp) {
        this.couleurRemp = couleurRemp;
        notifyFigures();
    }

    public List<Figure> getFigures() {
        return figures;
    }

    public void setFigures(List<Figure> figures) {
        this.figures = figures;
    }
}
```

Traitement

```
package traitement;

public interface Traitement {
    void traiter();
}
```

TraitementImpl1

```
package traitement;

public class TraitementImpl1 implements Traitement{
    @Override
    public void traiter() {
        System.out.println("Traitement version 1");
    }
}
```

TraitementImpl2

```
package traitement;

public class TraitementImpl2 implements Traitement {
    @Override
    public void traiter() {
        System.out.println("Traitement de version 2");
    }
}
```

3. Effectuer des Tests du modèle

```
import dessin.Cercle;
import dessin.Dessin;
import dessin.Point;
import dessin.Rectangle;
import observer.ObservableImpl;
import traitement.Traitement;
import traitement.TraitementImpl1;

public class Application {
    public static void main(String[] args) throws
ClassNotFoundException, InstantiationException, IllegalAccessException
{
        Dessin dessin = new Dessin();

        Cercle c = new Cercle(0,0,3);
        Cercle c1 = new Cercle(0,1,5);
        Rectangle rect = new Rectangle(new Point(6,1),3,2);
        dessin.ajouterDessin(c);
        dessin.ajouterDessin(c1);
        dessin.ajouterDessin(rect);
        dessin.afficher();

        ObservableImpl objetParametrage = new ObservableImpl() ;
        objetParametrage.addAllFigures(dessin.getFigures());
        objetParametrage.setCouleurRemp(25);

        Dessin dessin1 = new Dessin();
        dessin1.ajouterDessin(new Cercle(3,4,5));
        dessin1.ajouterDessin(new Rectangle(new Point(1,1),3,4));

        dessin.ajouterDessin(dessin1);

        Traitement traitement = new TraitementImpl1();
        dessin.setTraitement(traitement);
        dessin.traiter();
        dessin.afficher();

    }
}
```