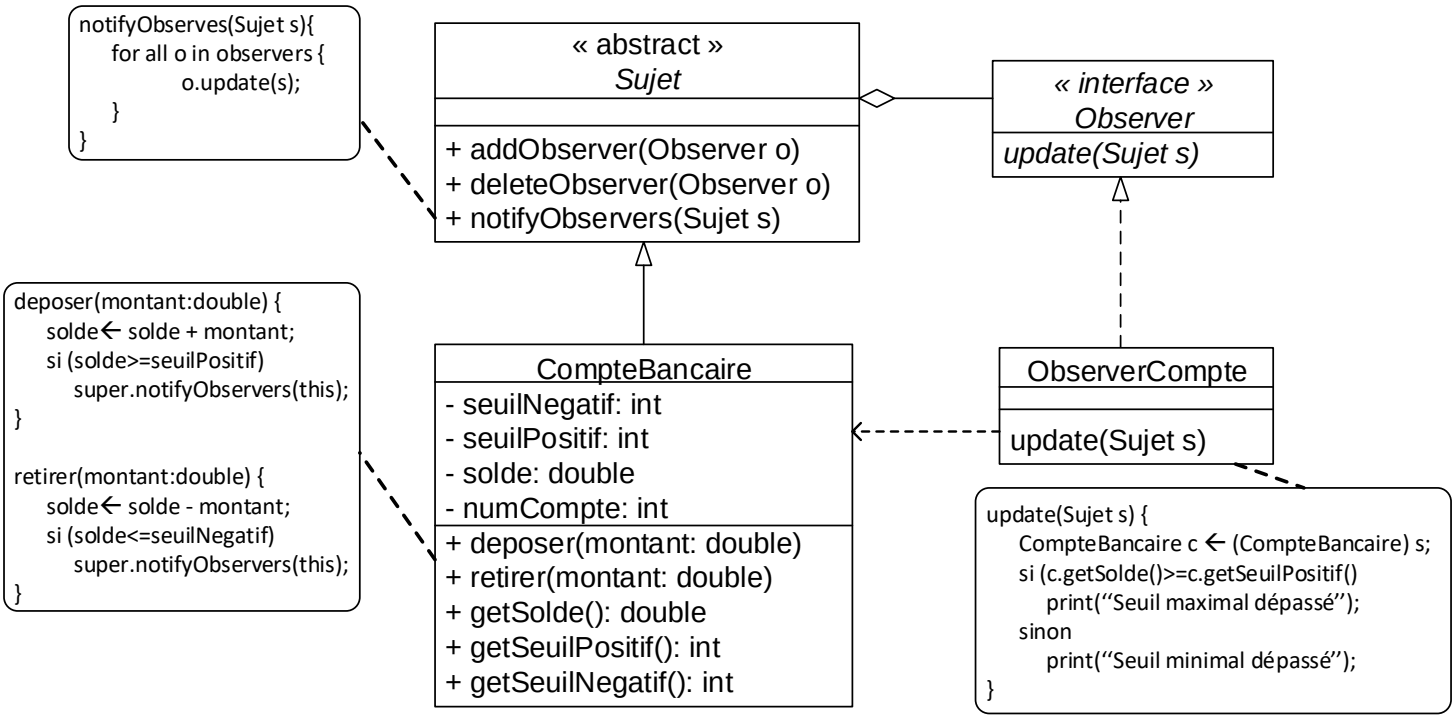
Solution des exercices

Exercice 1

Diagramme de classes



Code source

Le code suivant implémente la solution présentée dans le diagramme précédent mais en utilisant la classe Observable et l'interface Observer fournies dans le package java.util.

La classe CompteBancaire :

```
import java.util.Observable;
import java.util.Observer;

public class CompteBancaire extends Observable{

	private int seuilNegatif = -1000;
	private int seuilPositif = 10000;
	private double solde;
	private int numCompte;


	public CompteBancaire (double soldeInitial, int no) {
		solde = soldeInitial;
```

```
            numCompte = no;
      }

      public void deposer (double montant) {
            solde = solde + montant;
            System.out.println("Le dépôt est de " + montant);
            if ( solde >= seuilPositif) {
                  this.notifyObservers();
            }
      }

      public void retirer (double montant) {
            solde = solde - montant;
            System.out.println("Le retrait est de " + montant);
            if ( solde <= seuilNegatif){
                  this.notifyObservers();
            }
      }

      public void notifyObservers() {
            setChanged();
            super.notifyObservers();
      }

    // d'autres méthodes comme getSolde, getSeuil, setSeuil…
      public double getSolde(){
            return solde;
      }

      public int getSeuilPositif(){
            return seuilPositif;
      }

      public int getSeuilNegatif(){
            return seuilNegatif;
      }

}
```

La classe ObserverCompte :

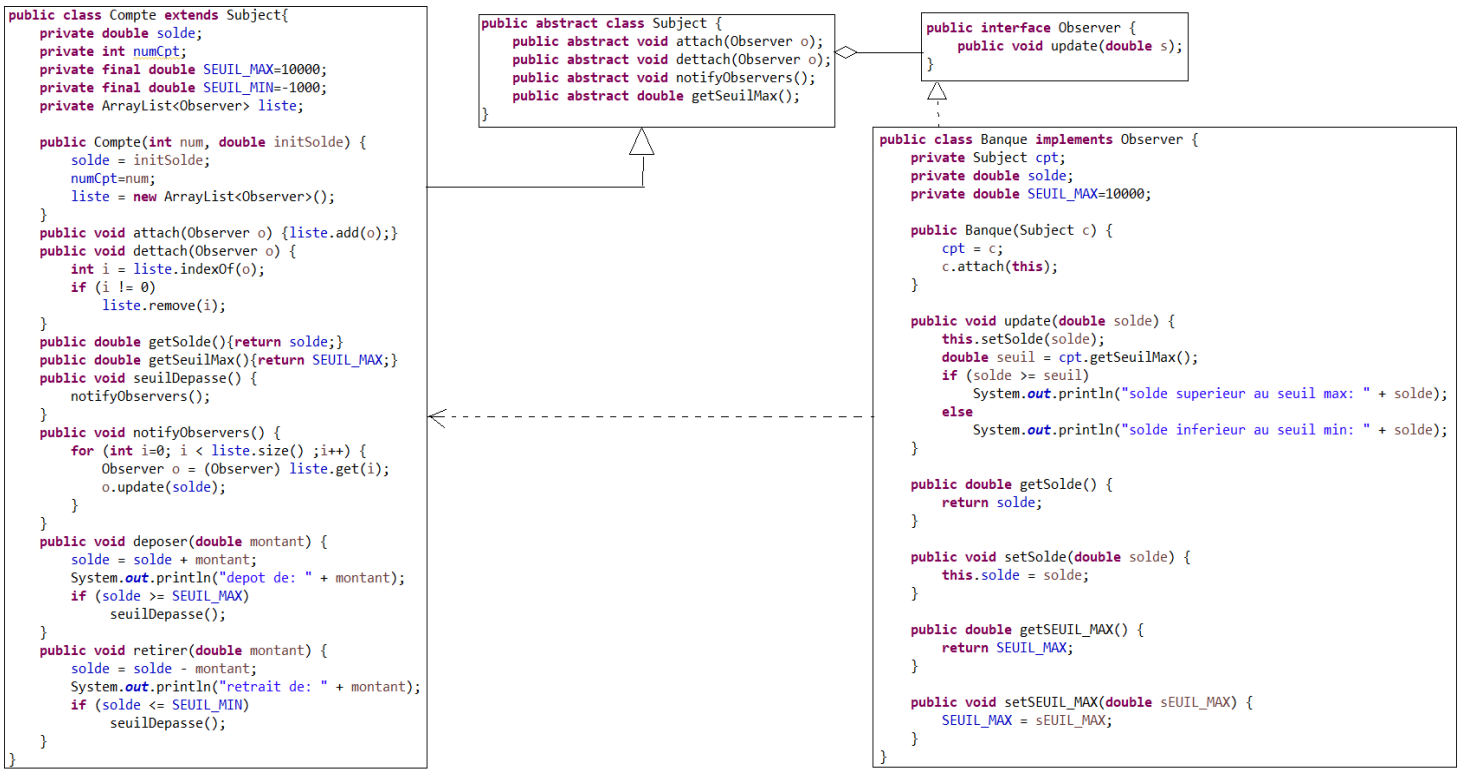
```
import java.util.Observable;
import java.util.Observer;

public class ObserverCompte implements Observer{

      public void update (Observable O, Object arg){
            CompteBancaire compte = (CompteBancaire)O;
            if (compte.getSolde() >= compte.getSeuilPositif()){
                  System.out.println("Seuil maximum dépassé!!!!");
            }

            else if (compte.getSolde() <= compte.getSeuilNegatif()){
                  System.out.println("Seuil négatif dépassé!!!!");
```

```
            }
        }
}
```

Une classe pour tester le tout :

```
import java.util.Observer;

public class CompteTest {

      public static void main (String[] args) {
      CompteBancaire compte = new CompteBancaire(3500, 1);
      System.out.println("Solde initial: "+ compte.getSolde());
      Observer banque = new ObserverCompte();
      compte.addObserver(banque);

      compte.retirer(10000); // solde est de -6500 --> doit déclencher la
notification
      compte.deposer(20000); //solde est de 13500 --> doit déclencher la
notification
      compte.retirer(4000); // rien ne se passe, on est dans la zone!
      }

}
```

Autre solution : sans l'utilisation des outils de java.util (Observer et Observable)



```
public class Compte extends Subject{
    private double solde;
    private int numCpt;
    private final double SEUIL_MAX=10000;
    private final double SEUIL_MIN=-1000;
    private ArrayList<Observer> liste;

    public Compte(int num, double initSolde) {
        solde = initSolde;
        numCpt=num;
        liste = new ArrayList<Observer>();
    }
    public void attach(Observer o) {liste.add(o);}
    public void dettach(Observer o) {
        int i = liste.indexOf(o);
        if (i != 0)
            liste.remove(i);
    }
    public double getSolde(){return solde;}
    public double getSeuilMax(){return SEUIL_MAX;}
    public void seuilDepasse() {
        notifyObservers();
    }
    public void notifyObservers() {
        for (int i=0; i < liste.size() ;i++) {
            Observer o = (Observer) liste.get(i);
            o.update(solde);
        }
    }
    public void deposer(double montant) {
        solde = solde + montant;
        System.out.println("depot de: " + montant);
        if (solde >= SEUIL_MAX)
             seuilDepasse();
    }
    public void retirer(double montant) {
        solde = solde - montant;
        System.out.println("retrait de: " + montant);
        if (solde <= SEUIL_MIN)
             seuilDepasse();
    }
}
```

```
public abstract class Subject {
    public abstract void attach(Observer o);
    public abstract void dettach(Observer o);
    public abstract void notifyObservers();
    public abstract double getSeuilMax();
}
```

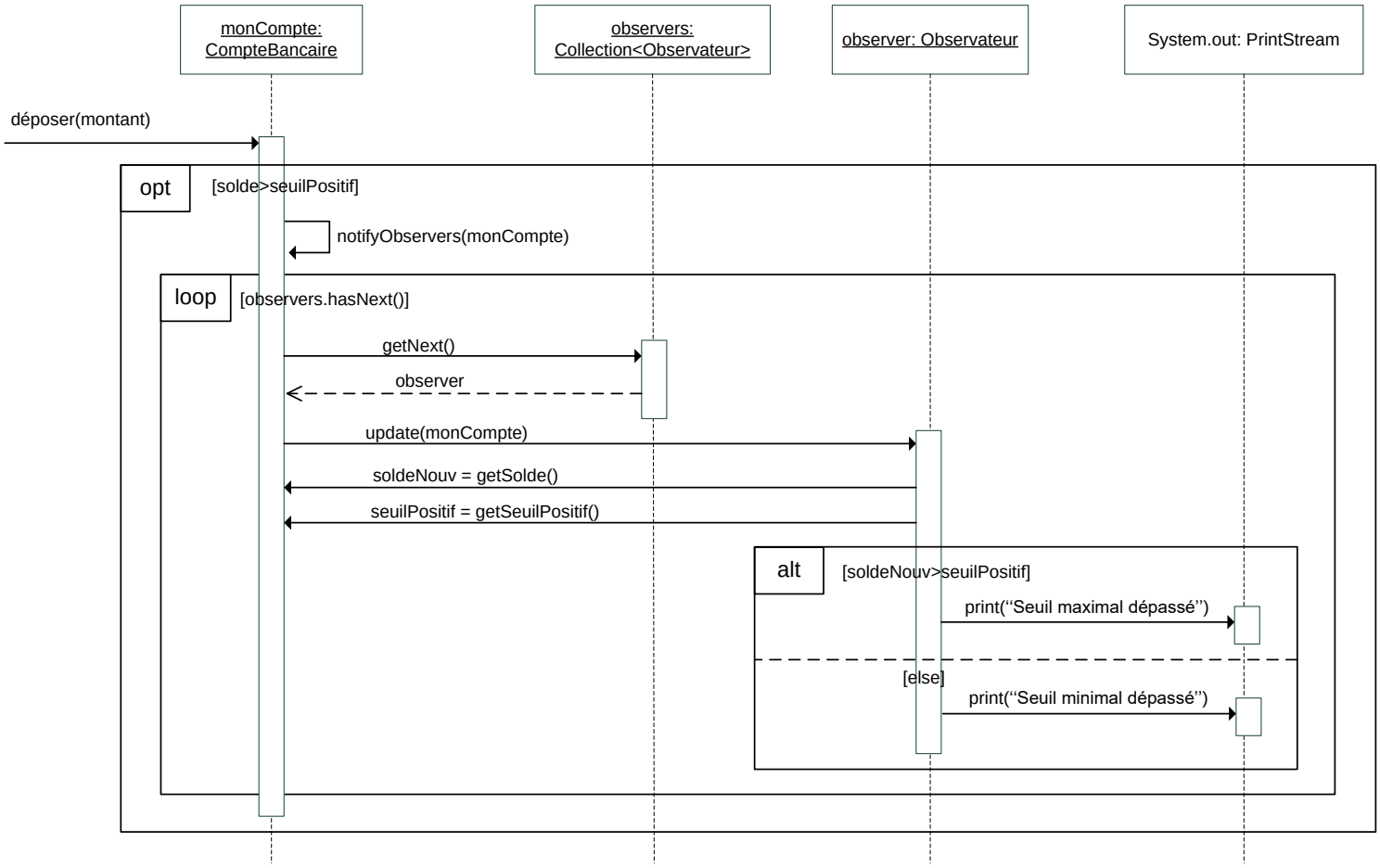
```
public interface Observer {
    public void update(double s);
}
```

```
public class Banque implements Observer {
    private Subject cpt;
    private double solde;
    private double SEUIL_MAX=10000;

    public Banque(Subject c) {
        cpt = c;
        c.attach(this);
    }

    public void update(double solde) {
        this.setSolde(solde);
        double seuil = cpt.getSeuilMax();
        if (solde >= seuil)
            System.out.println("solde superieur au seuil max: " + solde);
        else
            System.out.println("solde inferieur au seuil min: " + solde);
    }

    public double getSolde() {
        return solde;
    }

    public void setSolde(double solde) {
        this.solde = solde;
    }

    public double getSEUIL_MAX() {
        return SEUIL_MAX;
    }

    public void setSEUIL_MAX(double sEUIL_MAX) {
        SEUIL_MAX = sEUIL_MAX;
    }
}
```

Et pour tester le tout :

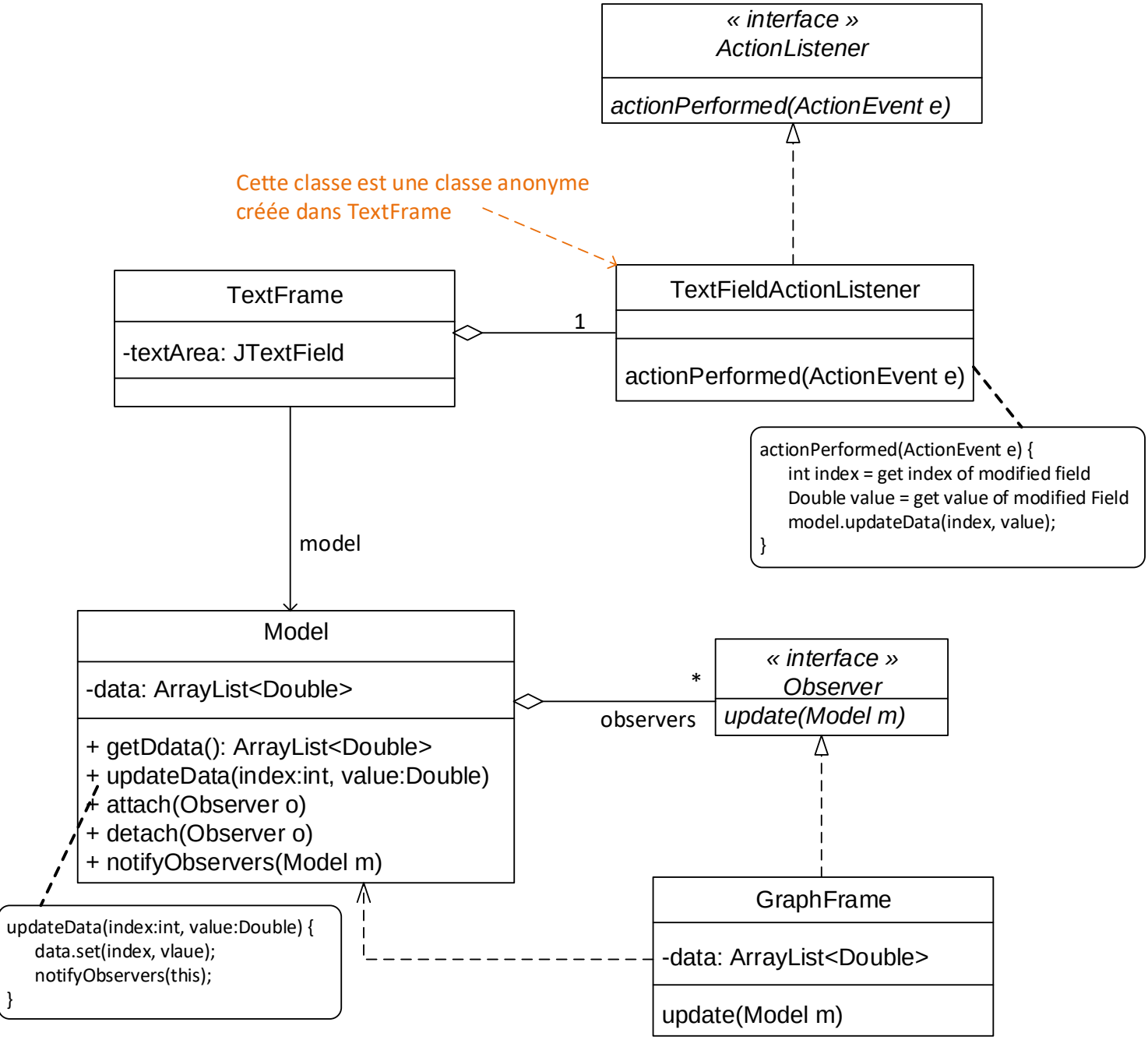
```
public class TestSoldeMain {
    public static void main (String[] args) {
        Compte c = new Compte(1,3500);
        System.out.println("Solde initial: "+ c.getSolde());

        Banque b = new Banque(c);
        c.retirer(10000); // solde est de -6500 --> doit déclencher la notification
        c.deposer(20000); //solde est de 13500 --> doit déclencher la notification
        c.retirer(4000); // rien ne se passe, on est dans la zone!
    }
}
```

Exercice 2



Exercice 3 (Exercice 5.1 du livre de Horstman)

La solution de cet exercice est adaptée du site du livre :
http://www.horstmann.com/oodp2/solutions/solutions.html

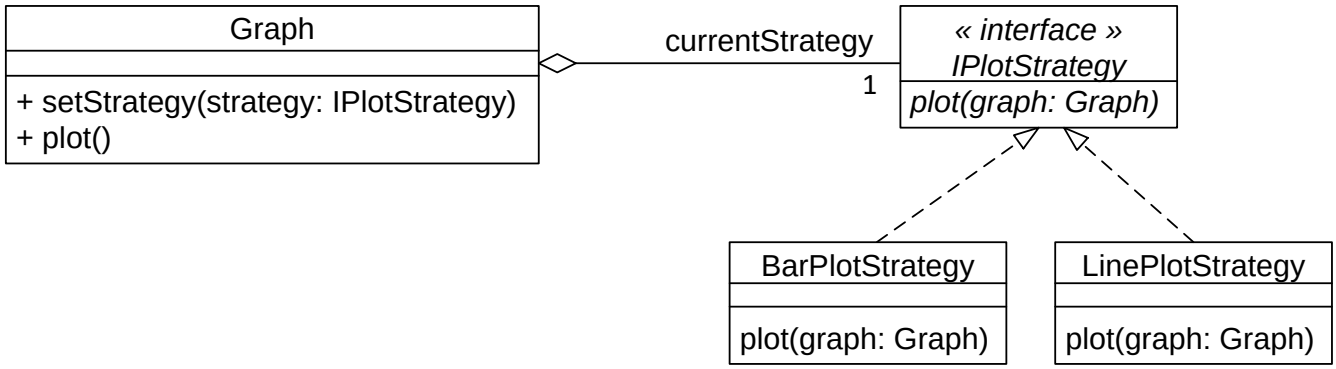


Le code zippé de la solution se trouve sur le site du cours (sous l'onglet « Exercices », le fichier zip nommé « graph »).

Exercice 4

Patron Stratégie

Diagramme de classes



Code source

La classe Graph :

```
public class Graph {

	private IPlotStrategy currentStrategy;

	public Graph() {
	// default strategy
		currentStrategy = new LinePlotStrategy();
	}

	public void setStrategy(IPlotStrategy newStrategy) {
		currentStrategy =  newStrategy;
	}

	public void plot() {
		currentStrategy.plot(this);
	}

}
```

L'interface IPlotStrategy :

```
public interface IPlotStrategy {

	void plot(Graph graph);

}
```

La classe LinePlotStrategy:

```
public class LinePlotStrategy implements IPlotStrategy{

	public void plot(Graph graph){
		System.out.println("Currently, I am ploting lines....");
	}
}
```

La classe BarPlotStrategy:

```
public class BarPlotStrategy implements IPlotStrategy{

	public void plot(Graph graph){
		System.out.println("Currently, I am ploting bars....");
	}
}
```

Une classe pour tester le tout :

```
public class PlotTester {

	public static void main(String[] args) {

		Graph graph1 = new Graph();
		graph1.plot();
		BarPlotStrategy barPlot = new BarPlotStrategy();
		graph1.setStrategy(barPlot);
		graph1.plot();
	}

}
```

Exercice 5

Diagramme de séquences :