Exercices sur les patrons Observateur et Stratégie

Exercice 1

Une banque désire être automatiquement avisée par un message lorsque le solde d'un compte se retrouve dans un état particulier : soit solde négatif sous -1000$ ou solde positif supérieur à 10000$. Les opérations pouvant agir sur l'état du solde du compte sont les opérations *déposer* et *retirer*. Quel patron peut aider à implémenter une solution adéquate dans le système de gestion de cette banque. Proposez une conception et décrire vos algorithmes en pseudocode.

Exercice 2

Faîtes le diagramme de séquence pour l'appel à la méthode *déposer(montant : double)* de la classe représentant le compte bancaire dans la conception que vous avez proposée comme solution à l'exercice 1.

Exercice 3 (Exercice 5.1 du livre de Horstman)

Écrire un programme qui contient deux vues (frames). Un frame avec une colonne de texte contenant des nombres et un frame qui dessine un graphe de barres affichant les valeurs de ces nombres. Lorsque l'utilisateur change un nombre, le graphe doit se mettre à jour. Stockez les données dans un modèle et utiliser le patron Observer pour attacher la vue de graphe au modèle. Lorsqu'un nombre est modifié, la vue des nombres doit mettre à jour le modèle et ce dernier doit notifier la vue de graphe qu'il y a eu un changement dans les données. La vue de graphe doit se mettre à jour alors.

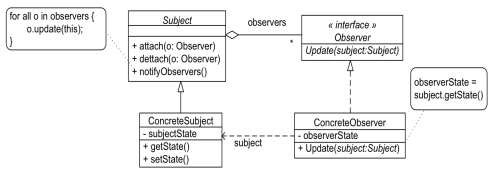
Exercice 4

Nous désirons concevoir une application qui permet de dessiner un graphe sous différents formats (« Line Plot » et « Bar Plot ») et qui facilite l'ajout de nouveaux formats de dessin.

1) Quel patron de conception peut nous aider à concevoir une telle application?

Voilà la classe Graphe avant l'utilisation de ce patron:

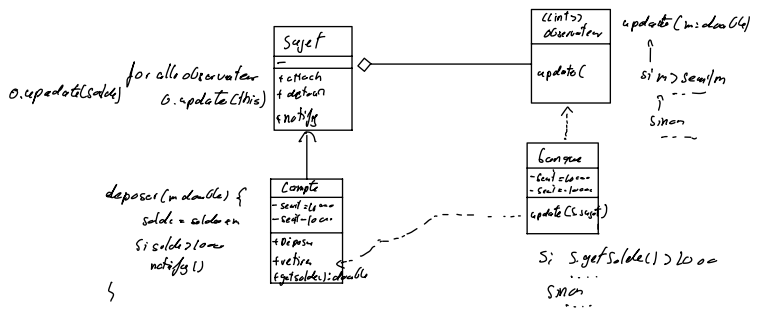
```
public class Graphe {

      private int type_dessin;
      static final int LINE = 0;
      static final int BAR = 1;
```


for all o in observers {
o.update(this);
}
Subject
+ attach(o: Observer)
+ dettach(o: Observer)
+ notifyObservers()
observers
*
« interface »
Observer
Update(subject:Subject)
observerState =
subject.getState()
ConcreteSubject
- subjectState
+ getState()
+ setState()
subject
ConcreteObserver
- observerState
+ Update(subject:Subject)

Ex. 1

Observateur

banque



```
        public Graphe(int entree) {
                type_dessin = entree;
        }

        public void plot() {

              switch (type_dessin) {
                    case LINE: dessinerLine(); break;
                    case BAR: dessinerBar(); break;
              }
        }

        public void dessinerLine() {
              // dessine en ligne
        }

        public void dessinerBar() {
              // dessine en barres
        }

        public int get_type_dessin(){
              return type_dessin;
        }

        public void set_type_dessin(int entree){
                          type_dessin = entree;

        }
}
```

2) Donner la nouvelle conception en utilisant le patron approprié. Considérez Line Plot le format de dessin utilisé par défaut.

3) Faîtes le diagramme de séquence pour l'appel à la méthode plot() de la classe Graphe dans la nouvelle conception que vous avez proposée comme solution.

Ex 4 strategie