

DE Java

Documents autorisés : tous documents papier.

En programmation objet, un design pattern est une technique de programmation très utilisée et adaptée à une situation donnée.

Le pattern Observer-Observable peut-être utilisé dans le cas où un objet souhaite suivre les évolutions d'un autre objet. Imaginons par exemple une classe appelée TauxCroissance qui représente le taux de croissance d'un pays. Quand ce taux devient inférieur à un certain seuil, on prévient des instances d'autres classes (par exemple du type Investisseur). Ici la classe TauxCroissance est l'observable et la classe Investisseur est l'observer.

Ce pattern est implanté dans le langage Java, mais le problème que l'on vous demande de résoudre ici est d'écrire votre implantation.

Question 1 :

Compléter la définitions de la classe Observable (package monJava.util) suivante :

```
public class Observable{
    /** Adds an observer to the set of observers for this object */
    public void addObserver(Observer o){
        // to do
    }

    /** Deletes an observer from the set of observers of this object. */
    public void deleteObserver(Observer o){
        // to do
    }

    /** If this object has changed, as indicated by the hasChanged method, then notify all of its
    observers */
    public void notifyObservers(){
        // to do
    }

    /** Marks this Observable object as having been changed. The hasChanged method will now
    return true.*/
    protected void setChanged(){
        // to do
    }

    /** Indicates that this object has no longer changed */
    protected void clearChanged(){
        // to do
    }

    /** Tests if this object has changed.*/
```

Handwritten notes:

- A blue checkmark is next to the opening curly brace of the class definition.
- Next to the `addObserver` method, there is a handwritten note: "il est au non?"
- Next to the `deleteObserver` method, there is a handwritten "X" mark.
- Next to the `notifyObservers` method, there is a handwritten checkmark.

```

public boolean hasChanged(){
    // to do
}
}

```

Pour compléter cette classe, ajoutez-y un attribut du type List, ArrayList... ou tout autre conteneur pouvant contenir des objets du type Observer (voir plus bas la définition d'un Observer) , et implanter toutes les méthodes de la classe Observable.

Remarque : si vous ne savez pas utiliser les conteneurs de Java vous pouvez utiliser un simple tableau à la place.

Un observer est définit par son interface :

```

public interface Observer{

    /** This method is called whenever the observed object is changed. An application calls
    an Observable object's notifyObservers method to have all the object's observers notified of the
    change. */

    void update(Observable o) ;
}

```

Question 2 :

Ecrivez la classe TauxCroissance en la faisant hériter de Observable. Munissez cette classe d'un constructeur permettant d'initialiser le taux de croissance, surcharger la méthode toString pour qu'elle affiche la valeur actuelle taux.

Enfin, ajoutez à cette classe une méthode setTaux qui, fixe une nouvelle valeur pour le taux de croissance, et dès que ce taux devient inférieur à un seuil (que vous définirez vous-même), notifie le changement aux Observers.

Question 3 :

Implanter l'interface Observer dans une classe appelée Investisseur. La méthode update se contentera d'afficher un message.

Question 4 :

Remarque : vous pouvez répondre partiellement à cette question même si vous n'avez pas répondu à la question 3.

Ecrivez une classe appelée Test qui possède un programme principal, et ajoutez-y un code qui permet de tester les classes TauxCroissance et Investisseur ?