Langage C++

- CIR2 -

ISEN ALL IS DIGITAL! NANTES | yncréa

TP10 : Observateur

Ayoub KARINE (ayoub.karine@isen-ouest.yncrea.fr)

Les patrons de conception (design patterns en anglais) sont des solutions à des problèmes récurrents de conception logicielle. L'idée du patron de conception "Observateur" est de permettre à des objets (les observateurs) d'être informés du changement d'état d'autres objets (les sujets ou les observables).
Le but de ce TP est d'implémenter le patron de conception "Observateur" pour les raisons suivantes :

- il permet de mettre en oeuvre un nouvelle fois le polymorphisme, et d'ainsi consolider votre pratique sur ce concept fondamental en POO
- il permet de manipuler deux nouveaux concepts : les smart pointers et l'héritage multiple
- il va faciliter votre compréhension du framework QT

Application 1 : Station de météo

L'objectif de cette application est de notifier les agriculteurs à chaque changement de température. Ainsi l'enchaînement va être comme suit :

Changement de l'état de la température → notifier les agriculteurs → affichage de la nouvelle valeur de température

Exigence : Utiliser que les smart pointer

Questions :

1. Créer une classe abstraite Observateur qui contient :
    a. Une méthode virtuelle pure qui permet de notifier l'agriculteur du changement de température
2. Créer une classe Agriculteur contenant un id. Un Agriculteur est considéré comme Observateur. De ce fait, elle redéfinit la seule méthode de la classe mère (voir affichage de la question 5.c).
3. Créer une classe abstraite Sujet qui contient :
    a. Une méthode virtuelle pure qui permet d'ajouter un observateur
    b. Une méthode virtuelle pure qui permet de notifier les observateurs

c. Un vecteur d'observateur

4. Créer une classe StationMeteo qui est considéré comme le sujet observé par l'Agriculteur. De ce fait, elle redéfinit les deux méthodes de la classe mère. En plus, cette classe contient :

   a. La température
   b. Une méthode de modification de température

5. Programmer le scénario ci-dessous dans la fonction main :

   a. Création d'une station météo
   b. Création d'un agriculteur (observateur)
   c. Ajouter cet observateur à cette station
   d. Changer la valeur de température et vérifier si une notification est affichée. Un exemple de notification est donnée ci-dessous :

      La nouvelle valeur de température est 15

Application 2 : Modification des coordonnées d'un point

L'objectif de cette application est d'implémenter une classe fille de la classe Point qui notifie tous ses observateurs à chaque changement de coordonnées.

Exigence : Utiliser que les smart pointer

Questions :

1. Récupérez votre classe Point de vos TPs précédents.
2. Implémenter les classes suivantes :

   a. Subject, stockant un vecteur de smart pointers sur Observer
   b. Observer
   c. ObservablePoint, héritant de Subject et de Point
   d. PointObserver1, héritant d'Observer, écrivant à chaque modification d'un ObservablePoint :

      ```
      Point {this} is now at ({x}, {y})
      ```

      Les parties entre {} devant être remplacées par leurs valeurs respectives.
   e. PointObserver2, héritant d'Observer, écrivant à chaque modification d'un ObservablePoint :

      ```
      New coordinates of Point {this} : x = {x}, y = {y}
      ```

3. Tester vos classes via le code ci-dessous :

```
void obsersablePointTest() {

  ObservablePoint observablePoint(0,0);
   observablePoint.addObserver(
```

```
            shared_ptr<PointObserver1>(new PointObserver1()) );
  observablePoint.addObserver(
            shared_ptr<PointObserver2>(new PointObserver2()) );
  observablePoint.moveTo(10, 10);
  cout << "---" << endl;
  observablePoint.moveTo(42, 0);
}

// Exemple d'affichage
```

```
Point 0x7fffcab838e0 is now at (10,10)
New coordinates of Point 0x7fffcab838e0: x = 10, y = 10
---
Point 0x7fffcab838e0 is now at (42,0)
New coordinates of Point 0x7fffcab838e0: x = 42, y = 0
```