TD1 – NF28 : Implémentation du modèle MVC

Présentation

Objectif

L'objectif de ce TD est d'implémenter le modèle MVC (modèle, vue, contrôleur), d'illustrer le rôle actif que peuvent avoir certains modèles et plus généralement de mettre en place le mécanisme de dépendance des objets entre eux.
Le principe de base reste qu'une vue est nécessairement adaptée à son modèle et que le modèle ne connaît pas les éventuels objets qui peuvent dépendre de son état (que ceux-ci soient des vues ou des objets quelconques).

Principe de réalisation

On considère une liste d'images et une Console munie d'un timer qui permute à intervalles réguliers l'image qui doit être affichée dans la seconde fenêtre. Pour visualiser cette image on utilise une ImageView pouvant recevoir des notifications. La console notifie tous ses observateurs dès que l'image change. L'ImageView doit être l'un de ces observateurs pour se mettre à jour. La console est pilotée à partir d'une vue très simple (ConsoleView). Elle permet de régler la fréquence, le lancement et l'arrêt du Timer.

Exemple

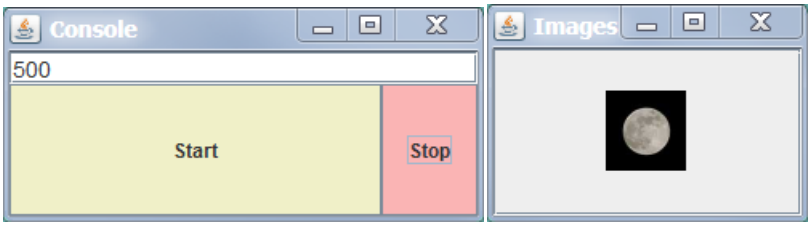

Figure 1 : interface graphique pour la création d'une console

Proposition de phases de réalisation

Phase 1

La classe Console possède un champ de type `PropertyChangeSupport` permettant l'ajout d'observateurs (`PropertyChangeListener`). Prévoir une méthode de la classe Console (ex : `time`) qui sera activée par le timer et qui est chargée d'envoyer une notification aux observateurs comportant la nouvelle Image. Elle doit contenir la ligne suivante :

```
pcs.firePropertyChange("image", null, <une ImageIcon>);
```

Phase 2

Créer une interface graphique permettant de déclencher et d'arrêter le timer. Une console sert de modèle à cette interface graphique.
Créer une classe pour chacune des fenêtres : `ImageView`, et `ConsoleView`.
Pour afficher une image, il suffit de créer un JLabel avec une ImageIcon en paramètre. Celle-ci est créée à partir d'un nom de fichier.
Remarque : c'est bien la console elle-même et non sa vue qui envoie les notifications.

Test

Pour tester l'application, on pourra créer une classe contenant une méthode main ne servant qu'à lancer l'ouverture des fenêtres. Cette classe pourra servir aux autres TD. Pour faire apparaître la fenêtre, il suffit de la créer. Son constructeur doit contenir l'ensemble des paramètres permettant de la rendre visible et de quitter l'application en fermant la fenêtre.

```
public class MainTDnf28 {
  public static void main(String[] args) {
       td1();
  }
  public static void td1() {
     ImageView imageView = new ImageView("Images");
     ConsoleView consoleView = new ConsoleView("Console")
  }
}
```

Phase 3

La classe Console doit posséder un objet de type java.util.Timer. Une classe représentant la tâche déclenchée régulièrement par le timer doit être implémentée (ex : ImageTimerTask). Elle doit hériter de la classe TimerTask. Implémenter la méthode " run " qui appellera la méthode de notification (`time`) prévue dans la classe Console.

Phase 4

Pour déclencher le timer, il suffit qu'une Console exécute les instructions suivantes :

```
ImageTimerTask task = new ImageTimerTask(this);
getTimer().schedule(task,debut,intervalle);
```

Pour arrêter un timer il suffit de l'annuler :

```
getTimer().cancel() ;
```

Il est cependant nécessaire de recréer un objet timer pour relancer le mécanisme.

Phase 5 - Prolongement

Ajouter un listener sur le champ texte pour que la validation du nouvel intervalle change la fréquence du timer.
Ajouter un objet JSlider dans l'interface pour régler l'intervalle.