C. Recanati
Interfaces Graphiques
INFO2, 2012

Feuille de TP n° 3

Le but de cet exercice est de développer un logiciel de dessin que nous implanterons au fur et à mesure des TP. La classe principale sera intitulée `Sketcher` (pour croquis). On prévoit de définir une interface `Constants` dans un fichier séparé pour la définition des constantes, ainsi que deux autres classes : `SketcherFrame` et `SketcherModel`. La classe `SketcherFrame` définira le cadre de la fenêtre où sera affiché le croquis, et `SketcherModel` contiendra la liste des éléments figurant sur la feuille de dessin (rectangle, cercle, ligne, etc.). Cette classe (le modèle) permettra la gestion des éléments du croquis (ajout, retrait). Un élément du croquis sera défini dans une classe abstraite `Element` qui comportera des sous-classes statiques implémentées. Ces sous-classes d'éléments instanciables (`Ligne`, `Rectangle`, etc.) - seront définies dans la classe abstraite `Element` comme classes internes, et seront donc accessibles dans le code par l'opérateur point « . ». On aura ainsi accès à la classe `Ligne` en écrivant `Element.Ligne`, et à la classe `Rectangle` avec `Element.Rectangle`.

La classe d'application `Sketcher` sera instanciée dans le `main` par un `new`, et « mémorisera » dans deux variables privées une instance de la classe du cadre qui lui sera associée, et une instance de la classe du modèle. Une autre variable privée mémorisera également l'instance de l'application créée.

1) Commencer l'implémentation du logiciel Sketcher en définissant l'interface `Constants`. Cette interface définira simplement des constantes pour les types d'éléments de dessin (`RECTANGLE, LIGNE, CERCLE, COURBE`) et des valeurs initiales pour la création d'un élément (`DEFAULT_ELEMENT_TYPE, DEFAULT_ELEMENT_COLOR`).

2) Dans un premier temps, le cadre ne sera utilisé que pour mémoriser le titre de la fenêtre, et le type de l'élément courant à dessiner. `SketcherFrame` aura donc trois membres (privés) : `title` de type `String`, `elementType` de type `int`, et `elementColor` de type `Color`. Ecrire les différents membres de cette classe, et utiliser l'interface `Constants` pour en initialiser les valeurs dans un constructeur. (Vous pouvez aussi avancer le code du logiciel de dessin en incorporant ici la barre de menu qui a été définie dans l'exercice 2 du TP2).

3) Ecrire le squelette de la classe d'application `Sketcher`.

4) Ecrire le squelette de la classe `SketcherModel`. Le modèle d'un dessin sera par exemple implanté par une liste chaînée de type `LinkedList` intilulée `ListeElements` et aura une méthode `add` et une méthode `remove`. La méthode `add` permettra d'ajouter un élément à la liste, et la méthode `remove` supprimera un élément en retournant un booléen indiquant si l'élément existait dans la liste et s'il a pu en être retiré.

5) On va définir des sous-classes de la classe abstraite `Element` qui seront des classes (statiques) implémentées. Dans un premier temps, définir la sous-classe `Element.Rectangle`, en vous inspirant du code de la classe abstraite `Element`, et de celui de sa classe interne `Element.Ligne` :

```
// Element.java
import java.awt.*;
import java.awt.geom.*;

public abstract class Element {

   protected Color color; // un élément a tjs une couleur

   public Element(Color color) {
      this.color = color;
   }

   public Color getColor() {
      return color;
   }
         // un élément a une forme récupérable par getShape() :
   public abstract Shape getShape();
         // un élément a aussi un plus petit rectangle englobant
         // récupérable par getBounds() :
   public abstract java.awt.Rectangle getBounds();
         // + une méthode modify pour positionner l'élément :
   public abstract void modify(Point debut, Point fin);

         // on définit ici une sous-classe « concrète » de Element
         // la sous-classe Ligne, accessible par Element.Ligne
   public static class Ligne extends Element {

      private Line2D.Double  ligne;     // ça sera sa forme (Shape)

      public Ligne(Point debut, Point fin, Color couleur){
         super(couleur);
         ligne = new Line2D.Double(debut, fin);
      }

      public Shape getShape() {
         return ligne;
      }

      public java.awt.Rectangle getBounds() {
```

```
            return ligne.getBounds();
        }

        public void modify(Point debut, Point fin) {
            ligne.x1 = debut.x;
            ligne.y1 = debut.y;
            ligne.x2 = fin.x;
            ligne.y2 = fin.y;
        }

    }
        //  vient ensuite le code implémentant Element.Rectangle
        // etc. pour tous les types d'élements de dessin prévus
}
```

Les constructeurs des classes d'éléments prennent 3 arguments : un début et une fin (de type `Point`) pour positionner la forme, et un argument couleur (de type `Color`). Ces constructeurs « mémorisent » également la forme créée dans un membre privé qui a un type spécifique lié au type de l'élément (ici `Line2D.Double` pour une `Ligne`). Un objet de type `Shape` peut néanmoins ensuite être retourné par l'accesseur `getShape`(), en *castant* la valeur de ce champ privé (car tous ces types de `awt` peuvent être subsumés par le type `Shape` de Swing).

La classe abstraite `Element` inclut un membre de type `Color` (avec un accesseur `getColor`() et un constructeur qui initialise ce membre) et déclare des méthodes `abstract getShape()` et `abstract getBounds()` retournant respectivement la forme correspondant à l'élément (un rectangle, cercle, etc.) et son rectangle englobant. Le rectangle englobant retourné par `getBounds` servira lors de l'affichage d'un élément, à restreindre la zone impliquée dans l'affichage.