TP 2 : programme de dessin

Dans ce TP nous allons continuer à travailler sur la petite interface graphique écrite au TP précèdent. Il est donc vivement conseillé d'avoir fini le TP précèdent. Ici, nous allons ajouter la possibilité de dessiner quelques objets.

1 Création et affichage de quelques cercles

Dans les questions qui suivent, on pourra utiliser, si on le souhaite, la classe `Point` décrite dans a documentation de l'API Java (`java.awt.Point`).

- Créer une classe `Cercle`, contenant (au moins) le rayon et le centre du cercle
- Ajouter un constructeur à la classe `Cercle`, permettant de l'initialiser facilement.
- Ajouter une méthode `void dessiner(Graphics g)` à la classe `Cercle`. Cette méthode sera appelé à partir de la méthode `paint` de votre applet (question suivante). Dans la méthode `dessiner` de la classe `Cercle`, on utilisera la méthode `drawOval` de la classe `Graphics` pour dessiner le cercle (n'oubliez pas de consulter la documentation de cette classe).
- Dans votre classe principale (celle qui dérive de `JApplet`), ajoutez trois membres de type `Cercle`. Ajouter un appel à la méthode `dessiner` de chacun de ces trois cercles dans `paint`.

2 Ajout d'autres objets

Suivant le modèle de la classe `Cercle`, créez une classe `Carre` et une classe `Triangle`. Ajoutez quelques triangles et quelques carrés dans votre classe principale.

3 Programme interactif

Les programmes que vous avez l'habitude d'écrire sont des programmes séquentiels : on commence dans une fonction `main` puis on exécute une série d'instructions.

Lorsqu'on programme des interfaces graphiques interactives, l'approche séquentielle n'est pas bien adapté. En effet, l'exécution du programme va dépendre étroitement des actions d'un utilisateur. Le programme devra alors réagir à des événements (clic de souris, entrée au clavier, changement de taille de fenêtre). L'approche événementielle est alors adoptée.

En programmation événementielle, vous ne contrôlez pas l'exécution du programme. En fait, vous attendez que l'utilisateur fasse une action qui provoquera automatiquement l'appel d'une méthode que vous avez écrit. Par exemple, dans le TP précèdent, vous avez écrit la méthode `paint` qui est appelé automatiquement lorsque qu'il y a besoin de redessiner la fenêtre (lorsqu'on redimensionne la fenêtre par exemple).

Dans le fichier `Dessin.java` suivant, nous vous proposons un squelette qui vous aidera à écrire un premier petit programme interactif.

```
import javax.swing.JApplet;
import java.awt.Graphics;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;

// On déclare que la classe Dessin "est un" JApplet. En plus, on
// déclare que notre classe Dessin implémente toutes les
// fonctionnalités d'un "MouseListener". C'est a dire, qu'elle
// implémente toutes les méthodes nécessaires pour gérer les clics de
// la souris. Regardez la documentation de "MouseListener" dans la
```

```
// doc Java.
public class Dessin extends JApplet implements MouseListener
{

    // Cette  méthode "init" est  toujours appelé  automatiquement par
    // JApplet après le démarrage du programme.  On pourra y mettre le
    // code dont  on a besoin  pour initialiser notre  programme. Dans
    // notre cas on voudra juste initialiser la gestion de la souris.
    public void init()
    {
        System.out.println("Bonjour, je m'initialise");

        // Ici on  dit à l'appelet qu'on veut  recevoir les évènements
        // souris. A partir de maintenant JApplet sait que s'il y a un
        // évènement  de souris, il  peut appeler  des méthodes  de la
        // classe Dessin associés à l'objet courant (this).
        addMouseListener(this);
    }

    // Toutes  les méthodes  qui suivent  sont associés  à l'interface
    // MouseListener.  Les  4 premières  ne nous intéressent  pas pour
    // l'instant (mais elles doivent obligatoirement être présentes ..
    // sinon, on ne serait  pas conforme à l'interface MouseListener).
    // Toutes  ces  méthodes  vont  être appelés  automatiquement  par
    // JApplet si les évènements correspondants surviennent.
    public void mouseEntered (MouseEvent e) {;}
    public void mouseExited  (MouseEvent e) {;}
    public void mousePressed (MouseEvent e) {;}
    public void mouseReleased(MouseEvent e) {;}
    public void mouseClicked (MouseEvent e)
    {
        System.out.println("Position:" + e.getX() + "," + e.getY());
    }

    public void paint (Graphics g)
    {
        // effacer avant de tout redessiner
        g.clearRect(0,0,getWidth(),getHeight());
        g.drawString ("Bonjour", 50, 25);
    }
}
```

Lisez-le attentivement et consultez la documentation de l'API Java pour toutes les classes utilisés (en particulier MouseListener, MouseEvent, JApplet). Si vous ne comprenez pas tout, encore ... c'est normal. Nous aurons l'occasion de revenir sur différents aspects par la suite.

- Exécutez le programme d'exemple proposé.
- Fusionnez le programme écrit précédemment avec celui-ci.
- Dans la méthode `mouseClicked`, changez la position d'un des cercles pour que celui-ci se trouve à l'endroit où on a cliqué. Pour ceci, ajoutez l'instruction `repaint();` juste avant la fin de la méthode `mouseClicked`. Cet appel dit à l'applet de redessiner la fenêtre.
- Créez un tableau de cercles et faites en sorte que chaque fois qu'on clique sur la fenêtre, un nouveau cercle soit ajouté au tableau.

Si vous avez fini, essayez, par exemple, d'ajouter un attribut couleur aux différents objets. Référez-vous à la documentation de la classe `Graphics`.