La classe l'objet et le dessin

`http://www.enseignement.polytechnique.fr/profs/informatique/Luc.Maranget/IF/06/`

On peut décrire grossièrement les objets de l'*Abstract Window Toolkit* de Java ainsi.

– Les composants `Component` sont les objets graphiques élémentaires, par exemple un bouton `Button` etc. Les composants possèdent entre autres une méthode `paint` dont la mission est de dessiner le composant. Cette méthode est appelée par le système graphique quand cela lui semble utile. On peut par ailleurs forcer le rafraichissement d'un composant en invoquant sa méthode `repaint`.
– Les conteneurs `Container` sont des regroupements de composants. Ils possèdent entre autre une méthode `add(Component c)` pour enregistrer les nouveaux composants. Par héritage les conteneurs sont des composants.
– Les cadres `Frame` sont des fenêtres affichées à l'écran avec notamment un titre (passé en argument du constructeur). Par héritage, les cadres sont des conteneurs. Les `Frame` possèdent une méthode `show` qui sert à effectivement « montrer » la fenêtre à l'écran, une fois celle ci construite.

On peut noter que les classes `Component` et `Container` sont abstraites.

1 Interprétation

1. Peut-on avoir des conteneurs de conteneurs ?
2. La méthode `paint` prend en argument un contexte graphique `Graphics` dans lequel on peut dessiner par les méthodes idoines. Que représenet ce contexte, et pourquoi `repaint` ne prend-elle pas d'argument ?
3. Que fait probablement la méthode `repaint` d'un conteneur ?
4. Cela a-t-il un sens de mettre plusieurs cadres dans un conteneur ? Discuter typage et comportement à l'exécution.

2 Salade d'étiquettes

Voici le programme graphique minimal :

```
import java.awt.* ;

class Minimal {
  public static void main(String [] arg) {
    Frame f = new Frame(arg[0]) ;
    f.show() ;
  }
}
```

1. Que fait le programme graphique minimal ?
2. Que va-t-il se passer si on fait « `java Minimal` ». Corriger un éventuel problème.
3. Sur ma machine les valeurs par défaut du positionement et de la taille de la fenêtre du programme minimal ne sont pas satisfaisantes. Comment, à votre avis, contrôler la taille et le positionement de la fenêtre du programme minimal ?
4. Le composant graphique tout prêt le plus simple est l'étiquette, `Label`, qui affiche juste la chaîne passée en argument au constructeur. Écrire la classe `Salut` dont les objets sont des fenêtres qui disent bonjour.

5. On peut changer la couleur du fond d'un composant par la méthode `setBackgroup(Color c)`, où les couleurs sont du style `Color.red`, `Color.white` etc. Concevoir, par héritage de la classe `Label` une classe des étiquettes au fond colorié.
6. Tous les conteneurs possèdent un `LayoutManager` qui règle le positionnement relatif de leurs composants. Le conteneur le plus simple est le `Panel` et son `LayoutManager` est par défaut un `FlowLayout` qui affiche ses composants les uns à la suite des autres, d'abord de gauche à droite, puis sur une nouvelle ligne en cas de débordement. Écrire la classe `Simple` qui produit la fenêtre du document joint.

3 Salade de boutons

Le bouton est un composant assez simple, il se présente un peu comme l'étiquette mais peut en plus être solicité à la souris. Le bouton réagit à une solicitation en appelant la méthode `actionPerformed` de ses `ActionListener`. Ceux-ci sont enregistrés au préalable par la méthode `addActionListener` (du bouton). Ainsi le bouton mininal est :

```
import java.awt.* ;
import java.awt.event.* ;

class MinimalButton extends Button {
  MinimalButton () {
    super("Minimal") ;
    addActionListener(new Minimal ()) ;
  }
}

class Minimal implements ActionListener {

  public void actionPerformed (ActionEvent e) {
    System.err.println("Bouton␣solicité") ;
  }
}
```

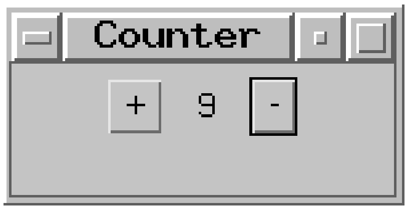
1. Concevoir la classe `QuitButton` des boutons qui servent à quiter une application.
2. On se donne la classe `Counter` des compteurs.

   ```
   class Counter {
     private int n ;

     Counter(int n) { this.n = n ; }
     void incr() { n++ ; }
     void decr() { n-- ; }
     int get() { return n ; }
     public String toString() {return Integer.toString(n) ; }

   }
   ```

   Concevoir et réaliser un composant `GraphicCounter` qui, une fois ajouté à un `Frame` quelconque, donnera ce résultat.



Étant entendu que cliquer sur « + » incrémente le compteur, que cliquer sur « - » le décrémente, et que le contenu courant du compteur est affiché. On procédera par étapes :

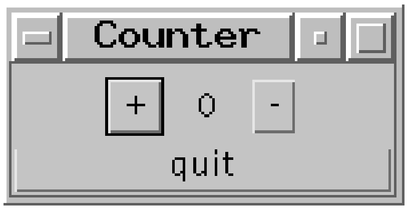
(a) Trouver une classe dont `GraphicCounter` peut être une sous classe.

(b) Concevoir et écrire le constructeur de cette classe.

(c) Se préocuper des `ActionListener`. Un démarche excessivement objet est de concevoir une classe abstraite des espions de nos deux boutons regroupant ce que ces espions ont de commun. On peut changer le texte affiché dans un `Label` par sa méthode `setText(String txt)`.

(d) Donner le code qui ajoute notre compteur graphique à la fenêtre quelconque, comme par exemple, la méthode `main` de la classe `GraphicCounter`.

3. Ajouter un bouton pour quitter. Ce bouton sera ajouté à la fenêtre quelconque. Le `LayoutManager` par défaut des `Frame` gère cinq zones, à savoir quatre bordures désignées par les quatre points cardinaux (`"North"`, etc.), et une zone centrale (`"Center"`). Sa méthode `add` prend une chaîne désignant la zone en premier argument. L'effet recherché est celui-ci :



4 Un composant très graphique

Le composant le plus simple est le canevas `Canvas`. Il correspond à un rectange dans lequel la méthode `paint` dessinera. Voici un canevas dont la méthode `paint` dessine le triangle de Sierpinsky à l'ordre donné par le contenu d'un compteur.

```
import java.awt.* ;

class SiepCanvas extends Canvas {
  private Counter c ;

  SiepCanvas(Counter c) {
    super() ;
    this.c = c;
  }

  public void paint(Graphics g) {
    int w = getSize().width ;
    int h = getSize().height ;

    Point top = new Point(w/2,0) ;
    Point left = new Point(0,h) ;
    Point right = new Point(w,h) ;
```

```
    // effectivement dessiner par une méthode non-donnée.
    paintSiep(g, c.get() , top, left, right) ;
  }
}
```

Ne soyez pas trop curieux, l'essentiel est que la classe `SiepCanvas` dessine bien un triangle de Sierpinsky. La version web du cours contient tous les détails.

1. On suppose que notre `SiepCanvas` est dans un `Frame` quelconque. À votre avis, que se passe-t-il lorsque que je redimensionne la fenêtre à la souris.
2. Écrire une application graphique qui va présenter, de haut en bas, un compteur graphique, un triangle de Sierpinsky, et enfin un bouton pour quitter.
3. Clique sur « `+` » ou « `-` » entraîne-t-il, à votre avis, l'affichage d'un nouveau triangle ? Si non, tenter d'y remédier avec les moyens du bord, mais en écrivant un minimum de code. L'héritage de `Counter` est suggéré.
4. Résoudre le même problème en utilisant le style des `ActionListener`. Du point de vue de la class `GraphicCounter` cela demande d'ajouter une méthode `addActionListener` à la classe `GraphicCounter`, afin d'enregistrer l'espion, et d'invoquer l'espion chaque fois que l'un des boutons du compteur graphique est solicité. Noter que la construction des espions `Plus` et `Moins` par héritage facilite un peu cette dernière modification.

Solutions des exercices

Exercice 1

1. Oui car, par héritage les conteneurs sont des composants.
2. Le contexte graphique représente une zone de l'écran où afficher, cette zone étant propre à chaque composant.

   La méthode `repaint()` peut être invoquée par n'importe qui, à des moments où le contexte graphique d'un composant à repeindre n'est pas nécessairement disponible.

   De toute façon, la méthode `repaint` signale simplement à tout le système graphique Java qu'il faut redessiner, aucun contexte n'est ici utile. C'est le système Java qui invoquera `paint` avec le bon contexte graphique en argument.
3. La méthode `repaint` d'un conteneur invoque probablement les méthodes `repaint` des sous-composants du conteneur.
4. Il est conceptuellement bizarre de composer une fenêtre effective à l'aide de plusieures fenêtres effectives. Les fenêtres du système graphique de la machine sont en effet des *a priori* sans rapports entre elles. Mais le typage ne peut pas détecter cette absurdité. Tenter de commettre l'absurdité déclenche une exception spécifique.

Exercice 2

Par manque de temps, voici seulement la classe `Simple`.

```
import java.awt.* ;

class Simple extends Frame {

  Simple (String title) {
    super(title) ;
    Panel p = new Panel () ;
    p.add(new LabelColor (Color.red, "La␣rouge")) ;
    p.add (new Label("et")) ;
    p.add(new LabelColor (Color.yellow, "la␣jaune")) ;
    add(p) ;
    setSize(100,100) ;
    setLocation(100,100) ;
    show() ;
  }

   public static void main (String[ ] args) {
     Simple f = new Simple("Simple") ;
   }
}

class LabelColor extends Label {
  LabelColor (Color c, String txt) {
    super(txt) ;
    setBackground(c) ;
  }
}
```

Exercice 3

1. Par manque de temps, voici seulement la classe solution. D'abord le bouton pour quitter, classe `Quit`.

   ```
   import java.awt.* ;
   import java.awt.event.* ;

   class QuitButton extends Button {
     QuitButton () {
       super("quit") ;
       addActionListener
         (new ActionListener () {
           public void actionPerformed (ActionEvent e) {
             System.err.println("Adios") ;
             System.exit(0) ;
           } ;
         }) ;
     }
   }

     /*
   class Quit implements ActionListener {

     public void actionPerformed (ActionEvent e) {
       System.exit(0) ;
     }
   }
     */
   ```

   On note l'utlisation d'une *inner class* anonyme à la place d'une classe `Quit`, donnée en commentaire.

2. Voici la code solution, classe `GraphicCounter`.

   ```
   import java.awt.* ;
   import java.awt.event.*;

   class GraphicCounter extends Panel {
     Button plus = new Button("+") ;
     Button moins = new Button("-") ;
     Label lab = new Label("␣", Label.CENTER) ;
     Counter c ;
     GraphicCounter (Counter c) {
       super() ;
       this.c = c ;
       add(plus) ; plus.addActionListener(new Plus(this)) ;
       add(lab) ; lab.setText(c.toString()) ;
       add(moins) ; moins.addActionListener(new Moins(this)) ;
     }

     ActionListener listener ;

     public void addActionListener(ActionListener listener) {
       this.listener = listener ;
     }
   ```

```
    public static void main (String[ ] args) {
      GraphicCounter c = new GraphicCounter(new Counter (0)) ;
      Frame f = new Frame("Counter") ;
      f.add("North", c) ;
      f.add("South", new QuitButton ()) ;
      f.setSize(150,75) ;
      f.setLocation(100,100) ;
      f.show() ;
    }
}

abstract class Op implements ActionListener {
  GraphicCounter gc ;

  Op (GraphicCounter gc) {
    this.gc = gc ;
  }

  abstract void op() ;

  public void actionPerformed (ActionEvent e) {
    op() ;
    gc.lab.setText(gc.c.toString()) ;
    if (gc.listener != null)
      gc.listener.actionPerformed(e) ;
  }

}

class Plus extends Op {

  Plus (GraphicCounter gc) { super(gc) ; }

  void op() { gc.c.incr() ; }
}

class Moins extends Op {
   Moins (GraphicCounter gc) { super(gc) ; }

  void op() { gc.c.decr() ; }
}
```

On note que les constructeurs des espions prennent l'objet `GraphicCounter` en argument, en effet, ces espions ont besoin d'intervenir sur certains champs, essentiellement sur le `Counter c` et sur le `Label` lab. Il faut noter l'usage de **this** dans l'appel de ces constructeurs.

Ne vous occupez pas du champ `listener` et de la méthode `addActionListener`, c'est pour le prochain exercice.

3. (et 4) La méthode `main` de `GraphicCounter` répond aux questions.

Exercice 4

1. La fractale est redessinée dans la fenêtre retaillée. Cela fonctionne parce que les trois points de base du triangle sont recalculés au début de la méthode `paint` en fonction de la taille

courante du composant `SiepCanvas`.

2. On regroupe tout sans se préocuper de relier le `GraphicCounter` et le `SiepCanvas`

```
import java.awt.* ;
import java.awt.event.*;

class Siep extends Frame {
  Siep (String title, int n) {
    super(title) ;

    Counter c = new Counter(n) ;
    GraphicCounter gc = new GraphicCounter(c) ;
    SiepCanvas sc = new SiepCanvas(c) ;
    Button quit = new QuitButton () ;


    add("North", gc) ;
    add("Center",sc) ;
    add("South", quit) ;
    setSize (450, 480);
    setLocation(100,100) ;
  }

   public static void main (String[] args) {
     int n = 3 ;
     if (args.length > 0) n = Integer.parseInt(args[0]) ;
     new Siep("Sierpinsky", n).show() ;
   }
}
```

   Dès lors, il n'y a aucune raison que la fractale se redessine quand la valeur du compteur change.

3. Un idée est de redéfinir les deux méthodes qui changent l'état du compteur, afin qu'elles invoquent la méthode `repaint` du canevas après le changement d'état du compteur.

```
class OtherCounter extends Counter {
  private Siep s ;

  OtherCounter(int n, Siep s) {  super(n) ; this.s = s ; }

  void incr() { super.incr() ; s.sc.repaint() }

  void decr() { super.decr() ; s.sc.repaint() }
}
```

   Il faut ensuite modifier `Siep` ainsi :

```
class Siep extends Frame {
  SiepCanvas sc ;

  Siep (String title, int n) {
    ...
    Counter c = new OtherCounter(n, this) ;
    GraphicCounter gc = new GraphicCounter(c) ;
    sc = new SiepCanvas(c) ;
    ...
```

```
  }
}
```

C'est en fait assez tordu, il n'y a pas de raison que le compteur connaisse quelque chose du graphisme. La technique de la question suivante semble plus cohérente, à condition d'avoir doté les compteurs graphiques d'une méthode `addActionListener` (ce que nous avons fait).

4. Voici la réponse finale, selon le modèle des `ActionListener`. Une classe `Siep`.

```
import java.awt.* ;
import java.awt.event.*;

class Siep extends Frame {
  Siep (String title, int n) {
    super(title) ;

    Counter c = new Counter(n) ;
    GraphicCounter gc = new GraphicCounter(c) ;
    SiepCanvas sc = new SiepCanvas(c) ;
    Button quit = new QuitButton () ;


    add("North", gc) ;
    add("Center",sc) ;
    add("South", quit) ;
    gc.addActionListener (new RepaintListener (sc)) ;

    setSize (450, 480);
    setLocation(100,100) ;
  }

   public static void main (String[] args) {
     int n = 3 ;
     if (args.length > 0) n = Integer.parseInt(args[0]) ;
    new Siep("Sierpinsky", n).show() ;
   }
}

class RepaintListener implements ActionListener {
  Component comp ;

  RepaintListener(Component comp) { this.comp = comp ; }

  public void actionPerformed(ActionEvent e) {
    comp.repaint() ;
  }
}
```

Un aspect notable est que l'espion fonctionne avec n'importe quel composant.

Il est sans doute temps de revenir à la définition de la class `Op` pour voir ce qui est fait du champ `listener` du compteur graphique.