Outils de développement,

programmation événementielle et IHM

Chapitre 4 : Java, graphisme de base

Cyrille Bertelle - UFRST Le Havre

0-0

Applets

- Une applet est un programme qui est inclus dans une page HTML et qui va donc être exécuter par le navigateur lisant cette page, à condition qu'il possède les fonctionnalités pour le faire. Il doit donc contenir une machine virtuelle Java compressée
- La classe `java.applet.Applet` doit être la classe mère de toute applet incluse dans un document.

Applets : Un premier exemple

- **HelloWorld.java** définit une applet `HelloWorld` qui utilise un objet `Graphics` (de la bibliothèque `awt`).
- Cette applet utilise la méthode `drawString` sur cet objet, ce qui lui permet d'afficher une chaîne de caractères à une position donnée sur la fenêtre courante du navigateur.

```
import java.applet.Applet ;
import java.awt.Graphics ;
public class HelloWorld extends Applet {
  public void paint(Graphics g) {
    g.drawString("Hello world", 50, 25);
}}
```


Cette applet ne possède pas de point d'entrée `main()` : il est géré implicitement par le navigateur.

Exemple de page HTML appellant l'applet :

```
<html>
  <head>
    <title>  Un exemple d'applet </title>
  </head>
  <body>
    <applet code="HelloWorld.class" width=150
                                    height=25>
    </applet>
  </body>
</html>
```

En appelant cette page HTML dans un navigateur intégrant Java, on verra apparaître dans le navigateur le message “Hello world”.

Passage de paramètres

La méthode suivante permet de récupérer des paramètres passés à une applet dans une page HTML :

```
public String getParameter(String name)
```

Cette méthode ne peut être appelée que dans les méthodes `init()` ou `start()` d’une applet (ces méthodes sont décrites plus loin).

Voici donc une nouvelle version `HelloWorld2.java`

```
import java.applet.Applet;
import java.awt.Graphics;

public class HelloWorld2 extends Applet {
  String e;
  public void init()
      { e=getParameter("message"); }
  public void paint(Graphics g)
      { g.drawString(e, 50, 25); }
}
```


Voici un exemple de page HTML qui appelle l'applet :

```
<html>
  <head>
    <title>
        Un exemple d'applet avec paramètres
    </title>
  </head>
  <body>
    <applet code="HelloWord2.class" width=150
                                    height=25>
    <param name="message" value="Bonjour">
    </applet>
  </body>
</html>
```

Cycle de vie

Le navigateur utilisé pour exécuter l'applet contrôle la vie et l'activation de l'applet grâce aux quatres méthodes suivantes :

- `public void init()` est appelée après le chargement de l'applet dans la page html ;
- `public void stop()` est appelée chaque fois que le navigateur arrêt l'exécution de l'applet, soit parce que l'utilisateur change de page web, soit parce qu'il iconifie le navigateur ;

- `public void start()` est appelée chaque fois que l'applet doit démarrer, après `init()` ou apres un `stop()` lorsque l'utilisateur revient sur la page web contenant l'applet ou lorsqu'il désiconifie le navigateur ;
- `public void destroy()` est appelée à la fermeture du navigateur. Elle détruit les ressources allouées pour l'applet.

Compléments

Nous donnons brièvement et de manière non exhaustive quelques points d'entrée pour les lecteurs désirant utiliser les aspects *multimédia* de Java mais que nous n'utiliserons pas dans le cadre des applications présentées dans cet ouvrage.

Des méthodes sont disponibles pour récupérer des images et des sons :

- `public Image getImage(URL url);`
- `public Image getImage(URL url, String name);`
- `public AudioClip getAudioClip(URL url);`

- `public AudioClip getAudioClip(URL url, String name);`

Des méthodes de la classe `java.applet.AudioClip` permettent de manipuler les sons ainsi récupérés :

- `public abstract void play();`
- `public abstract void loop();`
- `public abstract void stop()`.

Des méthodes sont également disponibles pour jouer directement les sons :

- `public void play(URL url);`
- `public void play(URL url, String name)`.

Gestion de fenêtres avec AWT

Tracés dans des applets

Nous décrivons comment effectuer des tracés géométriques dans la zone graphique d'une applet. Nous utilisons pour cela des méthodes de la classe `java.awt.Graphics`. Elles sont résumées ci-dessous. (une description plus précise sera trouvé dans la documentation des API).

- `public void draw3DRect(int x, int y, int width, int height, boolean raised)` trace un rectangle en relief ;

- `public abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)` trace un arc de cercle ;
- `public abstract void drawLine(int x1, int y1, int x2, int y2)` trace un segment de droite ;
- `public abstract void drawOval(int x, int y, int width, int heignt)` trace une ellipse ;

- `public abstract void drawPolygon(int xPoints[], int yPoints[], int nPoints)` trace un polygone ;
- `public void drawRect(int x, int y, int width, int height)` trace un rectangle vide ;
- `public abstract void drawRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)` trace un rectangle vide à bords arrondis ;

- `public void fill3Drect(int x, int y, int width, int height, boolean raised)` trace un rectangle plein en relief ;
- `public abstract void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle)` trace un arc de cercle plein ;
- `public abstract void fillOval(int x, int y, int width, int heignt)` trace une ellipse pleine ;

- `public abstract void fillPolygon(int xPoints[], int yPoints[], int nPoints)` trace un polygone plein ;
- `public abstract void fillRect(int x, int y, int width, int height)` trace un rectangle plein ;
- `public abstract void fillRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)` trace un rectangle plein à bords arrondis.

Voici un exemple d'applet contenant des graphiques :

```
import java.awt.*;

public class Test extends java.applet.Applet {
  public void paint(Graphics g) {
    int i;
    g.setColor(Color.yellow);
    for (i= 0; i<14; i++)
      g.drawLine(10, 10+16*i, 10+16*i, 218);
    for (i= 0; i<14; i++)
      g.drawLine(10+ 16*i, 10, 218,  10+16*i);
    for (i= 0; i<14; i++)
      g.drawLine(10, 218-16*i, 10+16*i, 10);
    for (i= 0; i<14; i++)
```

```
      g.drawLine(10+16*i, 218, 218, 218-16*i);
  }
}
```

Dans la suite, on présente de manière privilégiée des applications graphiques autonomes et non sous forme d'applet.

Construire des interfaces fenêtrées

Une première fenêtre

- Dans l'exemple qui suit, on construit une simple fenêtre d'une dimension initiale, avec un titre.
- La fenêtre construite dérive de la classe `Frame` qui permet de définir des fenêtres avec barre de titre, menu, bords, etc ...
- On utilise `WindowListener` pour pouvoir gérer la fermeture de la fenêtre qui s'effectue lorsque l'utilisateur clique sur l'icône supérieure droite du bandeau de fenêtre :

  - il faut alors décrire qui gère des évènements de ce type.
  - Cette gestion se fait par des ”écouteurs” d’évènements (`listener`).
  - Ici, pour un objet héritant de `WindowListener`, on se mettra ”à l’écoute des évènements” grâce à la méthode `addWindowListener`.
- `WindowListener` est une interface. Elle contient un certain nombre de méthodes abstraites qu’il faut redéfinir. Ici, on redéfinit non trivialement la seule méthode `windowClosing` qui permettra de gérer la fermeture de la fenêtre.


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

public class Fenetre
        extends Frame
        implements WindowListener {
  public Fenetre() { setSize(400, 300); }
  public void windowClosing(WindowEvent event) {
    System.exit(0);
  }
  public void windowClosed
                (WindowEvent event) {}
  public void windowDeiconified
                (WindowEvent event) {}
```

```
  public void windowIconified
                (WindowEvent event) {}
  public void windowActivated
                (WindowEvent event) {}
  public void windowDeactivated
                (WindowEvent event) {}
  public void windowOpened
                (WindowEvent event) {}

  public static void main(String arg[]) {
    Fenetre Test = new Fenetre();
    Test.setTitle("ma fenetre JAVA");
    Test.show();
    Test.addWindowListener(Test);
```



```
  }
}
```

Dans la suite, nous donnerons une alternative plus allégée à cette gestion d'évènements attachés à des fenêtres, en s'appuyant sur un exemple.

Les différentes parties d'une fenêtre

Une fenêtre gérée par l'AWT peut comorter plusieurs éléments caractéristiques :

- Une page de fond ou canevas, dans laquelle on pourra tracer des figures géométriques comme celles qui ont été décrites au paragraphe 12 ;
- Une étiquette (`Label`) qui permet d'afficher un texte ;
- Une zone de texte (`TextArea`) pour un affichage pouvant être modifié ;

- Une liste déroulante permettant de faire un choix (`List`) ;
- Une zone de saisie de texte (`TextField`) ;
- Un bouton (`Button`) ;
- Une case d'option (`Checkbox`) ;
- Un menu déroulant (`Menu` et `MenuBar`).

Un exemple : un compteur

Dans l'exemple commenté suivant, on met en œuvre une petite interface graphique incluant différents boutons et gérant des évènements.

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

class FenetreCompteur extends Frame {
  int compteur ;

  // définition de boutons
  // en paramètre : le texte des boutons
  Button boutonIncr = new Button("+");
```



```
  Button boutonDecr = new Button("-");
  Button boutonQuit = new Button("quit");

  // un champ affichant la valeur du compteur
  // en paramètre : la taille des caractères
  TextField affichageCompteur = new TextField(7);

  //gestion des évènements provoqués
  //par les clics sur les boutons
  class ActionIncr implements ActionListener {
    public synchronized void actionPerformed
                                (ActionEvent e)
    {  compteur ++; afficherCompteur(); }
  };
```

```
class ActionDecr implements ActionListener {
  public synchronized void actionPerformed
                                (ActionEvent e)
  {  compteur --; afficherCompteur(); }
};

class ActionQuit implements ActionListener {
  public synchronized void actionPerformed
                                (ActionEvent e)
  {  System.exit(0); }
};

void afficherCompteur()
{  affichageCompteur.setText(
```



```
                  String.valueOf(compteur)); }

// constructeur
public FenetreCompteur(String nom) {
  super(nom);
  compteur = 0;
  setSize(240, 80);
  setLayout(new FlowLayout());
  add(boutonIncr);
  add(boutonDecr);
  add(boutonQuit);
  add(affichageCompteur);
  boutonIncr.addActionListener
                (new ActionIncr());
```

```
    boutonDecr.addActionListener
                    (new ActionDecr());
    boutonQuit.addActionListener
                    (new ActionQuit());
  }
}

public class TestAWT {
  static public void main(String argv[]) {
    FenetreCompteur x =
              new FenetreCompteur("compteur");
    x.show();
 }
}
```


Figure 1: Fenêtre générée par le programme TestAWT

Gestion des évènements

Sur l'exemple précédent, on a vu comment utiliser des évènements de type `ActionEvent` à partir des composants `Button` et `TextField`.

D'autres types d'évènements existent :

- `MouseEvent` pour mouvements et clice de souris ;
- `FocusEvent` pour savoir si la souris est au-dessus de la zone considérée ;
- `KeyEvent` pour l'enfoncement d'une touche ;
- `TextEvent` pour la modification de texte pour un composant intégrant une zone de texte.

Il faut alors créer, relativement à un événement de type `xxxEvent`, une classe qui implémente `xxxListener` où l'on définira le traitement à faire lorsque l'événement a lieu (méthode `actionPerformed`). Dans la classe fenêtre, on lancera l'écoute des évènements avec `addListener`.

Placement des composants

Les différents composants d'une fenêtre peuvent être placés de plusieurs manières avec un gestionnaire de placements (`layout manager`):

- `FlowLayout` range les composants ligne par ligne et de gauche à droite ;
- `BorderLayout` place les composants dans 5 zones : le centre et les 4 côtés ;

- `GridLayout` place les composants sur une grille 2D ;
- `GridBagLayout` place les composants sur une grille 2D, avec des coordonnées. Les zones n'ont pas toutes nécessairement la même dimension.