Outils de développement,

programmation événementielle et IHM

Chapitre 6
Interfaces fenêtrées avec SWING

Cyrille Bertelle - UFRST Le Havre

0-0

Swing : Introduction

- Kit de développement d'interface graphique "léger" du JFC (Java Foundation Class), invoqué par le package `javax.swing`
- Simplifie l'utilisation de l'AWT tout en donnant des fonctionnalités avancées pour la création d'interfaces utilisateurs (IHM)
- Portage indépendant du système de fenêtrage - Possibilité de spécifier le *"look and feel"* qui rapproche le résultat des systèmes fenêtrés connus (Windows, Motif + Métal (Sun) ... limitations dues aux copyrights)

Swing et AWT

- Les classes de Swing sont souvent des classes dérivées de l'AWT (préfixées par J : JFrame, JButton, ...) avec des fonctionnalitées d'utilisation simplifiées.
- `javax.swing.JApplet` classe d'applets d'utilisation similaire à `java.applet.Applet` mais qui doit être utilisée si on intègre des composants Swing.

Swing et AWT - exemple de JApplet

On reprend l'applet "`HelloWorld`" du § 7.1

```
import java.awt.*;

public class JappletHello
  extends javax.swing.JApplet {
    public void paint (Graphics g) {
        g.drawString("Hello World", 50, 25);
    }
}
```

Utilisation de cadres JFrame

Un JFrame est structuré en plusieurs couches superposées :

1. JRoot : pas abordé pour l'instant (gère le *"look and feel"*, p.e.)
2. Gestionnaire de mise en page : permet de choisir des modèles de placement de conteneurs de composants (à gérer de manière facultative)
3. Gestionnaire de barres de menus : pour placer des menus déroulants (à gérer de manière facultative)

4. Couche de contenu / container : tout composant ne peut être placé que dans un container (différence avec AWT), il faut le gérer obligatoirement
5. Couche des composants à placer dans le container

JFrame - un exemple (1)

Le programme qui va suivre gère un JFrame :

- On le dimensionne et on lui donne un titre.
- On gère sa fermeture d'une manière simplifiée (par rapport à 8.2.1) : on n'implémente pas l'interface `WindowListener` mais on invoque la méthode `addWindowListener` avec un paramètre de type `WindowAdapter` sur lequel on peut ne redéfinir que l'opérateur de fermeture (`windowClosing()`).
- On récupère le container du `JFrame` sur lequel on place un composant *volet* de type `JPanel` qui va permettre l'affichage d'une chaîne de caractères.

- Pour faire cela, on construit une classe dérivant de `JPanel` qui implémente la méthode `paintComponent` remplaçant la méthode `paint` de l'AWT (à ne plus utiliser directement).

JFrame - un exemple (2)

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

class HelloPanel extends JPanel {
  public void paintComponent(Graphics g) {
    super.paintComponent(g);
    g.drawString("Bonjour", 75, 100);
} }
class Terminator extends WindowAdapter {
    public void windowClosing(WindowEvent e) {
        System.exit(0);
}   }
```



```
class HelloFrame extends JFrame {
  public HelloFrame() {
    setTitle("HelloInFrame"); setSize(300, 200);
    addWindowListener(new Terminator());
    Container HelloContainer = getContentPane();
    HelloContainer.add(new HelloPanel());
} }
public class HelloInFrame {
  public static void main(String[] args) {
    JFrame frame = new HelloFrame();
    frame.show();
} }
```

JFrame - un exemple (3)

Remarque et exercice : On peut adapter le programme précédent pour lui faire afficher le dessin de l'exemple du § 8.1. La méthode `paint` de cet exemple sera renommer `paintComponent` et viendra remplacer celle qui est définie dans l'exemple précédent.

9.3 Affichage d'images

- Java intègre des classes permettant d'afficher des images (format GIF et JPEG) sur des objets de type `Graphics`
- Pour lire le fichier image, on utilise un objet `Toolkit` créé par l'appel de la méthode statique `getDefaultToolkit`. En invoquant la méthode `getImage(String name)` sur cet objet `Toolkit`, on contruit un objet `Image`.

Exemple :

```
String name="monalisa.jpg";
Image vue=
   Toolkit.getDefaultToolkit().getImage(name);
```

On peut aussi aller chercher le fichier image directement sur un site Internet grâce à la classe `URL`

```
URL name =
  new URL("http://www.leonard.com/mona.jpg");
```

Exemple d'utilisation de l'objet image dans une méthode `drawImage`

```
public void paintComponent(Graphics g) {
   ...
   g.drawImage(vue, x, y, null)
   ...
}
```


Affichage d'images - remarques

Le chargement de l'image en mémoire ou à partir d'Internet peut être long. Java met en œuvre un système de multi-processing permettant la suite du déroulement du programme parallèlement au chargement et affichage de l'image.

On utilise des processus "légés" partageant le même espace mémoire : ce sont des *threads*. Le multithreading existe déjà en langage C sur les systèmes multi-tâches comme Unix, mais ils sont lourds à mettre en œuvre : Java simplifie leur utilisation.

Des composants graphiques

Java dispose de nombreuses classes pour gérer des composants IHM évolués (bouton, menus déroulants, champs texte de saisie, ...) dont l'utilisation est devenue aujourd'hui intuitive mais qui nécessite une gestion complexe au niveau du programme.

Ce sont des composants actifs qui réagissent à des évènements pouvant se produire ou non à des moments quelconques en fonction du souhait des utilisateurs.

Un tel composant actif doit

- être représenté graphiquement
- être associé à des informations spécifiques
- pouvoir provoquer un action spécifique en fonction d'évènements aléatoires (clic de souris, p.e.)

Il faut donc traiter des 2 aspects complémentaires

- la définition et le placement graphique des composants (ce §)
- la gestion des évènements associés au composant (§ suivant)

Bouton

`JButton` correspond à une classe dessinant un bouton qui contient un libellé et qui est capable de détecter les clics de souris.

Dans l'exemple qui suit, on déclare un champs objet JButton, initialisé par son libellé, dans un objet JFrame. Le bouton est ajouté à un volet (JPanel) lui-même ajouté à un container.

Bouton - exemple

```
class BoutonFrame extends JFrame {
  JButton monBouton = new JButton("Mon Bouton");
  public BoutonFrame() {
    setTitle("BoutonFrame"); setSize(300, 200);
    Container BoutonContainer =
                          getContentPane();
    JPanel p = new JPanel();
    p.add(monBouton);
    BoutonContainer.add(p);
  }
}
```

Libellés, champs et zones de texte

- Les étiquettes `JLabel` ne servent qu'à placer une ligne de texte dans l'interface, sans réaction possible.
- Un composant `JTextField` correspond à un champs de saisie d'une seule ligne, alors qu'un composant `JTextArea` correspond à une zone de saise sur plusieurs lignes.

Ces composants s'intègrent comme les boutons. Nous donnerons des exemples ultérieurement pour expliquer la gestion évènementielle des textes saisis.

Cases à cocher, Boutons radio et Listes

- Le composant `JCheckBox` correspond au placement d'un libellé associé à une case située à côté et pouvant être cochée par l'utilisateur.
- Le composant `ButtonGroup` est un objet auquel on peut adjoindre, par la méthode `JRadioButton` des boutons *radio*. L'ensemble constitue des libellés avec des cases à cocher, sachant qu'un seul choix n'est possible sur l'ensemble.

- Le composant `Jlist` se construit avec un tableau de chaînes de caractères et permet l'affichage d'une liste sélectionnable.
- Le composant `JComboBox` permet de placer une liste déroulante. Les différents item de la liste se font en appelant la méthode `addItem` sur un tel objet.
- La liste de composants présentés ici n'est pas exhaustives : voir la documentation de l'API concernant `javax.swing`.

Exemple (sans la gestion évènementielle des composants actifs qui sera vue plus loin) :


```
class ListeFrame extends JFrame {
  JComboBox liste = new JComboBox();
  public ListeFrame() {
    setTitle("ListeFrame"); setSize(150, 150);
    addWindowListener(new Terminator());
    Container ListeContainer = getContentPane();
    liste.addItem("Hello");
    liste.addItem("Bonjour");
    liste.addItem("Bye");
    JPanel p = new JPanel();
    p.add(liste);
    ListeContainer.add(p);
} }
```

Gestionnaires de mise en page

Java permet de définir des *modèles* de mise en page des composants avec la classe `java.awt.LayoutManager`. L'association du gestionnaire de mise en page avec le container se fait par la méthode `setLayout` de l'objet container. Les différents modèles de mise en page sont :

- `FlowLayout` : modèle par défaut qui positionnne les composants sur une même ligne ou sur la ligne suivante si la ligne courante est remplie.
- `GridLayout(int nl, int nc)` : est un modèle de placement de composants dans un container découpé régulièrement en `nl` lignes et `nc` colonnes.

- `BorderLayout` : modèle de placement des composants en 4 bords (North, South, East, West) et un centre (Center). Le placement des composants se fera en ajoutant une de ces situations en premier paramètre de la méthode `add`. Par exemple :
  `bouton.add("West","mon bouton")`

Voici un exemple de programme que l'on pourra modifier pour tester tous les types de gestionnaires de page.

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

```
class Terminator extends WindowAdapter {
  public void windowClosing(WindowEvent e) {
    System.exit(0);
  }
}
class LayoutFrame extends JFrame {
  public Container layoutContainer;
  int i=1;
  public LayoutFrame(LayoutManager lm) {
    setTitle("LayoutFrame"); setSize(200, 200);
    addWindowListener(new Terminator());
    layoutContainer = getContentPane();
    layoutContainer.setLayout(lm);
  }
```



```
  public void newBouton() {
    layoutContainer.add(
            new Button("Bouton "+i++));
} }
public class TestLayout {
  public static void main(String[] args) {
    LayoutFrame frame =
      new LayoutFrame(new FlowLayout());
    frame.newBouton();frame.newBouton();
    frame.newBouton();frame.newBouton();
    frame.newBouton();
    frame.show();
} }
```

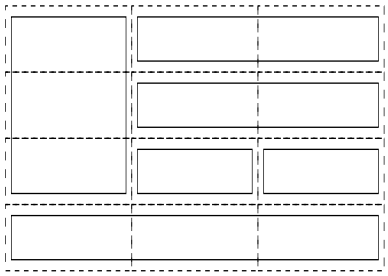
Gestionnaire de mise en page - conclusion

Les gestionnaires précédents sont les principaux, il en existe d'autres :

- `CardLayout` permet l'affichage de composants empilés ; on dispose de méthodes de type `next()`, `previous()` pour les parcourir ;
- `GridBagLayout` est le gestionnaire le plus riche : il étend `GridLayout` en permettant des placements de composants sur des cellules fusionnées (voir figure). On dispose d'objets de contrôle de placement `GridBagConstraints` qui permettent de définir des contraintes de placements pour chaque composant

(nombre de cellules occupées, positionnement du coin supérieur gauche, ...)

- ...



Exercice d'application : réalisation d'un formulaire.

Gestion des évènements

Modèle observateur-observé mis en place depuis l'API 1.1.
Deux types d'objets interviennent

- sources : composants graphiques réactifs à certains types d'évènements ;
- écouteurs objets qui implémentent une interface dérivée de `java.util.EventListener`.

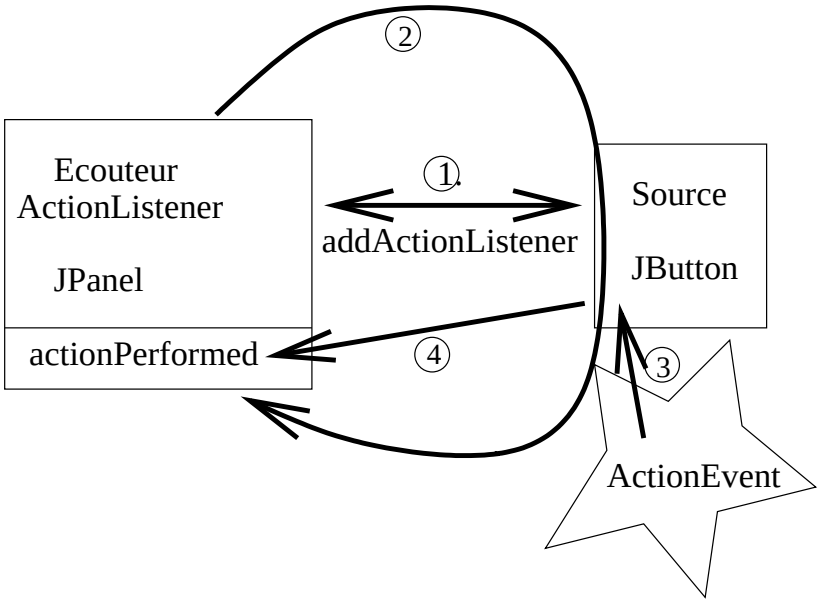
Par exemple, un bouton (`JButton`) peut avoir pour écouteur le volet (`JPanel`) dans lequel il est placé.

Il existe plusieurs types d'écouteurs : `ActionListener`, `FocusListener`, `KeyListener`, `MouseListener`, ...

Voici la suite de processus engendrée par une action de type `ActionEvent` avec un écouteur de type `ActionListener` :

- La source se lie à un ou des écouteurs (avec la méthode `addActionListener`)
- La source possède, par nature, des méthodes capables de reconnaître certains évènements (clic de souris, p.e.). Lorsque cet évènement (objet de type `ActionEvent`) est détecté, la source envoie l'évènement à l'écouteur qui appelle la méthode spécifique de gestion de l'évènement (`actionPerformed`) et le traite suivant sa nature.

Gestion des évènements - schéma



Exemple de gestion de boutons actifs

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

class CompteurFrame extends JFrame
                    implements ActionListener {
  int compteur=0;
  JButton boutonIncr = new JButton("+");
  JButton boutonDecr = new JButton("-");
  JButton boutonQuit = new JButton("Quit");
  JTextField
      affichageCompteur = new JTextField(7);
```

```
void afficherCompteur() {
  affichageCompteur.setText(
                String.valueOf(compteur));
}


public void actionPerformed(ActionEvent evt) {
  Object source = evt.getSource();
  if (source == boutonIncr) compteur++;
  if (source == boutonDecr) compteur--;
  if (source == boutonQuit) System.exit(0);
  afficherCompteur();
}
```



```
public CompteurFrame() {
  setTitle("CompteurFrame"); setSize(150, 150);
  Container compteurContainer =
                          getContentPane();
  JPanel p = new JPanel();
  p.add(boutonIncr); p.add(boutonDecr);
  p.add(boutonQuit);
  p.add(affichageCompteur);
  boutonIncr.addActionListener(this);
  boutonDecr.addActionListener(this);
  boutonQuit.addActionListener(this);
  compteurContainer.add(p);
  afficherCompteur();
}
```

```
}

public class Compteur {
  public static void main(String[] args) {
    JFrame frame = new CompteurFrame();
    frame.show();
  }
}
```


Exemple de gestion de menu

Construction de menus

Pour construire des menus, on doit définir et placer plusieurs types d'objets :

1. On construit une barre de menus

   `JMenuBar menuBar = new JMenuBar();`

   On la place en haut de la fenêtre `frame` avec la méthode

   `frame.setJMenuBar(menuBar);`

2. On construit le menu en lui-même

   `JMenu menu = new JMenu("Fichier");`

3. On construit les items à rattacher au menu

```
JMenuItem unItem = new JMenuItem("Ouvrir");
menu.add(unItem);
```

ou encore, sous forme condensée :

```
menu.add("Ouvrir")
```

qui renvoie en élément de type `JMenuItem`.
On peut aussi créer des sous-menus de la manière suivante

```
JMenu sousMenu = new JMenu("Options");
menu.add(sousMenu);
```

avec possibilité de placer des items dans ce sous-menu en invoquant `add` sur celui-ci comme cela est fait pour le menu principal.

4. On rattache le menu à la barre de menu

```
menuBar.add(menu);
```

Exercice d'application : ...

Exemple de gestion de menu (2)

Menus réactifs

- On ajoute un `ActionListener` sur chaque item réactif à la sélection avec la méthode `addActionListener` ;
- On intercepte les évènements de sélection des menus avec la méthode `actionPerformed` ;
- Dans la méthode `actionPerformed(Event evt)`, on récupère l'objet déclancheur avec la méthode `getSource()` et on regarde la nature de l'objet avec l'opérateur `instanceof` ;
- Si il s'agit d'un item de menu alors on peut récupérer la

chaîne de caractères correspondante avec la méthode `getActionCommand()`.

Le début de la méthode actionPerformed est du type :

```
public void actionPerformed(Event evt) {
  if (evt.getSource() instanceof JMenuItem) {
    String choix = evt.getActionCommand();
    if (choix.equals("Quitter"))
      ...
```

Exercice d'application : ...