Cours programmation par objets et Java

3ère partie: Le graphisme et AWT

Jacques Ferber

Version du : 20 octobre 1999

Table des matières

1. Graphisme en Java 2
1.1 Dessin "à la main" 2
2. Le nouveau modèle de gestion des événements 2
2.1 Table des gestionnaires d'événéments dans 1.1 3
3. Les composants d'interfaces 6
3.1 Principes 6
3.2 Hiérarchie des composants d'interface 6
3.3 Component 6
3.4 Button 7
3.5 Checkbox 8
3.6 Choice 8
3.7 Label 9
3.8 List 10
3.9 Panel et Applet 12
3.10 TextArea et TextField 12
3.11 Frame et Dialog 13
3.12 Canvas 16
3.13 Gestionnaires de présentation (layout managers) 16
3.14 Dialogues 17

1. Graphisme en Java

Remarque, sauf mentionné, toutes les techniques graphiques présentées ici fonctionnent pour Java 1.1 et au-dessus. Les techniques de la version 1.0 ne sont pas présentées.

1.1 Dessin "à la main"

Pour dessiner : on utilise des primitives graphiques qui sont des messages envoyés au context graphique passé dans l'applet (ou dans un composant graphique, voir plus loin).

Exemple :

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

public class EssaiGraphics extends Applet
{
    public void init()
    {
       resize(250,250);
      // setBackground(Color.white);
    }
    public void paint( Graphics g)
    {
      Dimension d = size();
                  Color bg = getBackground();

                  //Draw a fancy frame around the applet.
      g.setColor(bg);
      g.draw3DRect(0, 0, d.width - 1, d.height - 1, true);
      g.draw3DRect(3, 3, d.width - 7, d.height - 7, false);
      g.setColor(Color.darkGray);
      g.drawLine(0,0,250,250);
      g.drawLine(0,250,250,0);
      g.setColor(Color.blue);
      g.fillRoundRect(100,100,50,50,25,25);
      g.setColor(Color.magenta);
      g.drawString("salut les mecs",40,50);
      g.fillOval(120,120,60,40);
    }
}
```

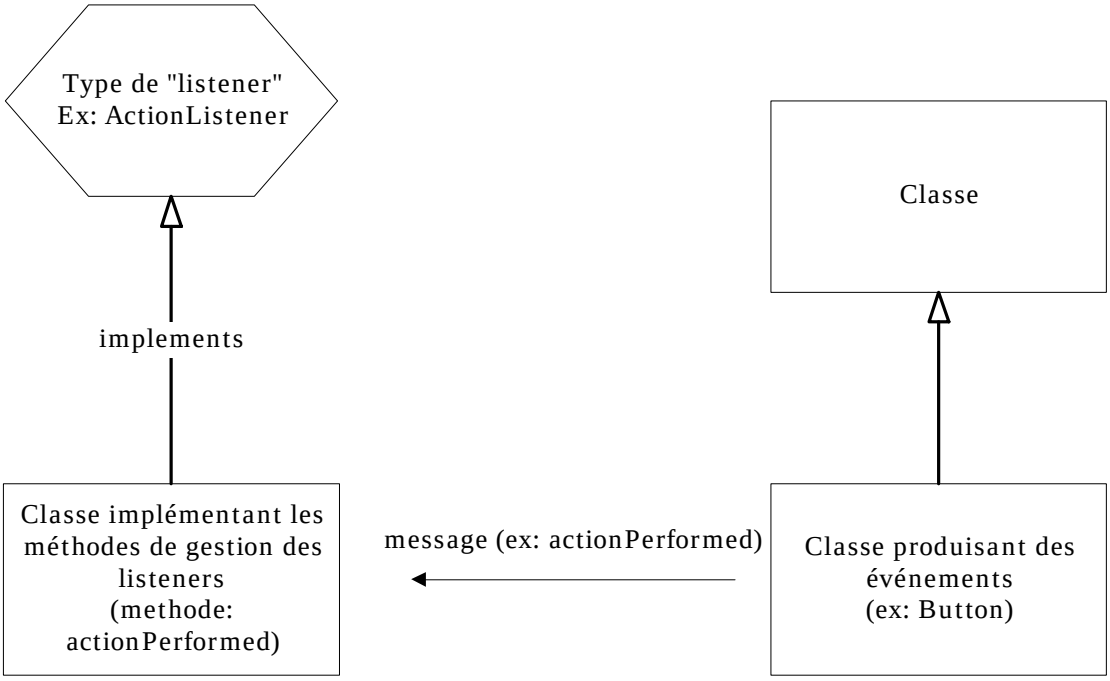
On ne doit pas appeler **paint** soi même. Si on a besoin de redessiner une fenêtre on fait appel à la méthode **repaint()** qui relance la mise à jour du dessin.

2. Le nouveau modèle de gestion des événements

Ce modèle est fondé sur la notion de « producteur d'événements » et « d'écouteurs » (listeners) qui interprètent et réagissent aux événements produits par les producteurs.

Button est un exemple de producteur de l'événement ActionEvent.

Le modèle :



¨Pour ajouter un « écouteur d'action » (**`listener`**) à un composant, il suffit de faire:

```
b.addActionListener(leListener);
```

où b est un composant qui reçoit des événement et **`leListener`** est un "listener" qui implémente une (ou plusieurs) méthodes de gestion d'événement.

2.1 Table des gestionnaires d'événéments dans 1.1

Listener	**InterfaceAdapter**	**ClassMethods**
ActionListener	none	actionPerformed(ActionEvent)
AdjustmentListener	none	adjustmentValueChanged(Adjustment Event)
ComponentListener	ComponentAdapter	componentHidden(ComponentEvent componentMoved(ComponentEvent) componentResized(ComponentEvent) componentShown(ComponentEvent)
MouseListener	MouseAdapter	mouseClicked(MouseEvent) mouseEntered(MouseEvent) mouseExited(MouseEvent) mousePressed(MouseEvent) mouseReleased(MouseEvent)
MouseMotionListener	MouseMotionAdapter	mouseDragged(MouseEvent) mouseMoved(MouseEvent)
TextListener	none	textValueChanged(TextEvent)
ItemListener	none	itemStateChanged(ItemEvent)

FocusListener	FocusAdapter	focusLost(FocusEvent) focusGained(FocusEvent)
KeyListener	KeyAdapter	keyPressed(KeyEvent) keyReleased(KeyEvent) keyTyped(KeyEvent)
WindowListener	WindowAdapter	windowActivated(WindowEvent) windowClosed(WindowEvent) windowClosing(WindowEvent) windowDeactivated(WindowEvent) windowDeiconified(WindowEvent) windowIconified(WindowEvent) windowOpened(WindowEvent)
ContainerListener	ContainerAdapter	componentAdded(ContainerEvent) componentRemoved(ContainerEvent)

Exemple d'utilisation de boutons et de leur gestion.

1er cas : c'est l'Applet qui gère les événements.

```
public class Beeper extends Applet
                    implements ActionListener {
    Button button;

    public void init() {
        setLayout(new BorderLayout());
        button = new Button("Click Me");
        add("Center", button);

        button.addActionListener(this);
    }

    public void actionPerformed(ActionEvent e) {
        Toolkit.getDefaultToolkit().beep();
    }
}
```

Les méthodes :

```
<producteur>.add<type de listener>(<listener>)
```

Exemple :

```
b.addActionListener(this) ; //où this est l'Applet qui hérite de
draw.ActionListener.
```

Exemple : dessiner Scribble en 1.1

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

public class Scribble extends Applet implements MouseListener,
MouseMotionListener {
   private int lastX, lastY,

   public void init() {
      this.addMouseListener(this);
```

```
        this.addMouseMotionListener(this);
    }

    public void mousePressed(MouseEvent e) {
        lastX = e.getX();
        lastY = e.getY();
    }

    public void mouseDragged(MouseEvent e) {
        Graphics g = this.getGraphics();
        int x = e.getX(), y = e.getY();
        g.drawLine(lastX, lastY, x, y);
        lastX = x; lastY = y;
    }

    public void mouseReleased(MouseEvent e) {}
    public void mouseClicked(MouseEvent e) {}
    public void mouseEntered(MouseEvent e) {}
    public void mouseExited(MouseEvent e) {}

    public void mouseMoved(MouseEvent e) {}
}
```

Le même en utilisant des classes internes:

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

public class Scribble2 extends Applet {
    private int lastX, lastY,

    public void init() {
        this.addMouseListener(new MouseAdapter() {
            public void mousePressed(MouseEvent e) {
                lastX = e.getX();
                lastY = e.getY();
            }
        });

        this.addMouseMotionListener(new MouseMotionAdapter(){
            public void mouseDragged(MouseEvent e) {
                Graphics g = this.getGraphics();
                int x = e.getX(), y = e.getY();
                g.drawLine(lastX, lastY, x, y);
                lastX = x; lastY = y;
            }
        });

        // on crée un bouton pour qui efface les dessins
        Button b = new Button("Effacer");

        b.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                Graphics g = getGraphics();
                g.setColor(getBackground());
                g.fillRect(0,0,getSize().width,getSize().height);
            }
        });

    this.add(b); // on ajoute le bouton a l'Applet.
    } // fin de l'init()

}
```

3. Les composants d'interfaces

3.1 Principes

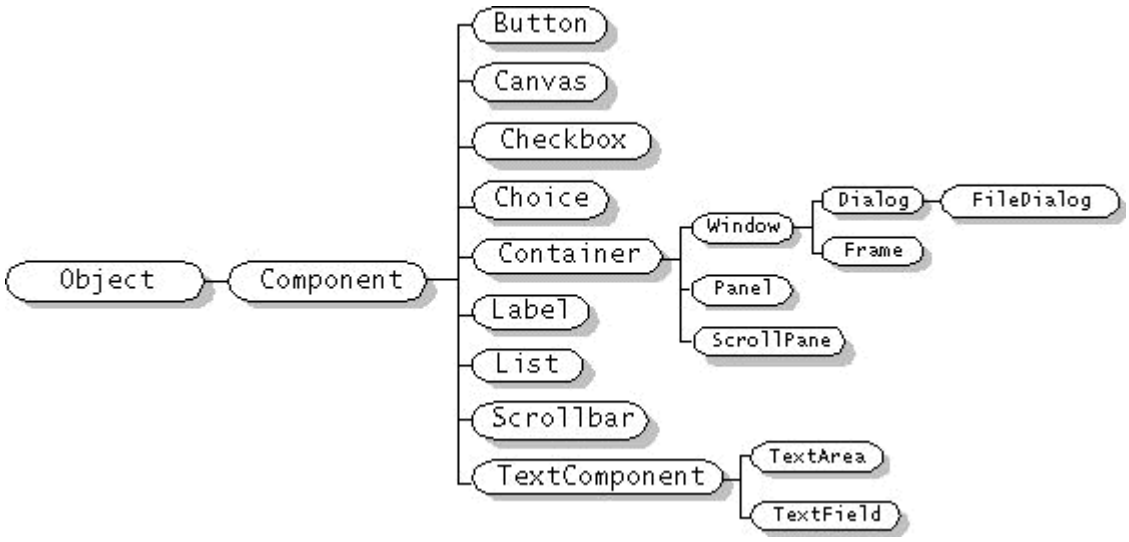
On place des composants dans des containers (**`Panel, Window, Frame,`** etc..). On les déclare dans les containers et on les ajoute avec la méthode **`add(Component)`** qui les place en fonction du **`layoutManager`** (voir plus loin).

Les composants savent s'afficher et recevoir des événements (voir plus loin).

Tous ces composants héritent de **`Component`**.

3.2 Hiérarchie des composants d'interface

Attention, il y a de nouveau composants d'interface, plus complets, les Swing (on les développera ensuite).



3.3 Component

Classe abstraite qui est responsable des comportements génériques de l'ensemble de l'interface. Elle fournit:

Ce qu'apportent les composants (les sous-classes donc de **`Component`**):

- Un support graphique: **`paint(), update(), repaint()`**
- Gestion d'événements (par transfert vers un listener)
- L'apparence: *fontes* (get, set the current font) et *couleur* (**`setForeground(Color)`**, **`getForeground()`**, **`setBackground(Color)`**, and **`getBackground()`**. The foreground color is the color used for all text in the component, as well as for any custom drawing the component performs. The background color is the color behind the text or graphics. For the sake of readability, the background color should contrast with the foreground color.
- Taille et position (géré par l'intermédiaire du Layout, cf. plus loin). **`preferredSize()`** and **`minimumSize()`** methods allow a component to inform layout managers of the component's preferred and minimum sizes.

Component also provides methods that get or set (subject to layout manager oversight) the component's current size and location.

3.4 Button

Bouton traditionnel. Sur lequel on clique normalement. Principe général on utilise le modèle événementiel, en indiquant la commande qui peut être utilisée.

Voici un bout de code qui "enable" un bouton (exemple tiré du tutoriel de Sun):


```
import java.awt.*;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.applet.Applet;

public class ButtonDemo extends Applet
                        implements ActionListener {

Button b1, b2, b3;
static final String DISABLE = "disable";
static final String ENABLE = "enable";

    public void init() {
        b1 = new Button();
        b1.setLabel("Disable middle button");
        b1.setActionCommand("disable");

        b2 = new Button("Middle button");

        b3 = new Button("Enable middle button");
        b3.setEnabled(false);
        b3.setActionCommand("enable");

      //Listen for actions on buttons 1 and 3.
      b1.addActionListener(this);
      b3.addActionListener(this);

        //Add Components to the Applet, using the default FlowLayout.
        add(b1);
        add(b2);
        add(b3);
    }

    public void actionPerformed(ActionEvent e) {
        String command = e.getActionCommand();

        if (command == DISABLE) { //They clicked "Disable middle button"
            b2.setEnabled(false);
            b1.setEnabled(false);
            b3.setEnabled(true);
        } else { //They clicked "Enable middle button"
            b2.setEnabled(true);
            b1.setEnabled(true);
            b3.setEnabled(false);
        }
    }
}
```

3.5 Checkbox

Permet de valider ou non une propriété. Si on veut qu'ils soient mutuellement exclusifs, on les place dans un **checkboxGroup**, et ils on alors l'apparence d'un "bouton radio".

Exemple du tutoriel de Sun:


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

public class CheckboxDemo extends Applet {

    public void init() {
        Panel p1, p2;
        Checkbox cb1, cb2, cb3; //independent checkboxes
        Checkbox cb4, cb5, cb6; //only one of these three can be selected
        CheckboxGroup cbg;

        //Build first panel, which contains independent checkboxes
        cb1 = new Checkbox();
        cb1.setLabel("Checkbox 1");
        cb2 = new Checkbox("Checkbox 2");
        cb3 = new Checkbox("Checkbox 3");
        cb3.setState(true);
        p1 = new Panel();
        p1.setLayout(new FlowLayout());
        //Using a GridLayout didn't work--kept box and text too far
apart.
        p1.add(cb1);
        p1.add(cb2);
        p1.add(cb3);

        //Build second panel, which contains a checkbox group
        cbg = new CheckboxGroup();
        cb4 = new Checkbox("Checkbox 4", cbg, false);
        cb5 = new Checkbox("Checkbox 5", cbg, false);
        cb6 = new Checkbox("Checkbox 6", cbg, false);
        p2 = new Panel();
        p2.setLayout(new FlowLayout());
        p2.add(cb4);
        p2.add(cb5);
        p2.add(cb6);

        //Add panels to the Applet.
        setLayout(new GridLayout(0, 2));
        add(p1);
        add(p2);

        validate();
    }
}
```

3.6 Choice

Fonctionne comme des "comboBox" de Windows.

- Ajout des items (qui doivent être des **String**) avec **addItem(String)**.
- Récupération de l'index sélectionné par **getSelectedIndex()**.

- Récupération de l'item (donc la chaîne sélectionnée) par **`getSelectedItem()`**.
- **`int countItems()`** : retourne le nombre d'items dans le **`Choice`**.
- **`String getItem(int)`** : retourne la chaîne correspondant à l'index spécifié.
- **`void select(int)`** et **`void select(String)`**: sélectionne l'item correspondant à l'index ou à la chaîne de caractère.

Exemple de Sun:


```
import java.awt.*;
import java.awt.event.ItemListener;
import java.awt.event.ItemEvent;
import java.applet.Applet;

public class ChoiceDemo extends Applet
                        implements ItemListener {
    Choice choice; //pop-up list of choices
    Label label;

    public void init() {
        choice = new Choice();
        choice.addItem("ichi");
        choice.addItem("ni");
        choice.addItem("san");
        choice.addItem("yon");

        choice.addItemListener(this);

        label = new Label();
        setLabelText(choice.getSelectedIndex(),
                     choice.getSelectedItem());

        //Add components to the Applet.
        add(choice);
        add(label);
    }

    void setLabelText(int num, String text) {
        label.setText("Item #" + num + " selected. "
                    + "Text = \"" + text + "\".");
    }

    public void itemStateChanged(ItemEvent e) {
        setLabelText(choice.getSelectedIndex(),
                     choice.getSelectedItem());
    }
}
```

3.7 Label

Les "labels" sont de simples chaînes de caractères qui ne peuvent pas être éditées.

Méthodes pratiques:

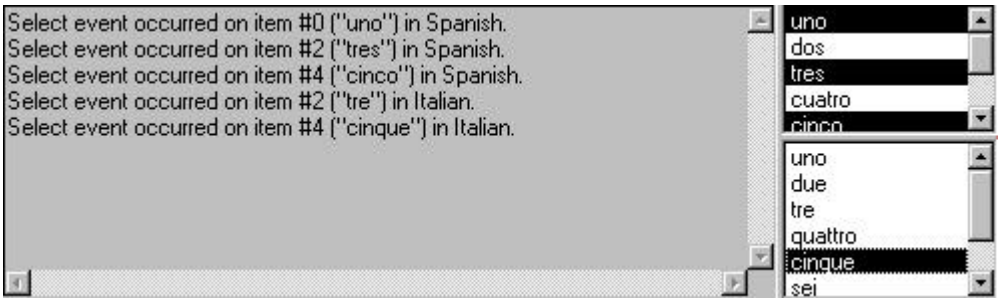
- setText(String) et getText() qui associe une chaîne à un label et qui retourne ce texte.
- getAlignment et setAlignment qui permettent de manipuler l'alignement du texte dans la chaîne (LEFT (par défaut), CENTER, RIGHT).

```
Exemples de création de labels
          Label label1 = new Label();
          label1.setText("Left");
          Label label2 = new Label("Center");
          label2.setAlignment(Label.CENTER);
          Label label3 = new Label("Right", Label.RIGHT);
```

3.8 List

Permet d'afficher une liste d'item à sélectionner. Très pratique pour faire des browser (et autres choses du même genre).

Exemple de Sun:




```
public class ListDemo extends Applet
             implements ActionListener,
                 ItemListener {
    TextArea output;
    List spanish, italian;
    String newline;

    public void init() {
         newline = System.getProperty("line.separator");

        //Build first list, which allows multiple selections.
        spanish = new List(4, true); //prefer 4 items visible
        spanish.add("uno");
        spanish.add("dos");
        spanish.add("tres");
        spanish.add("cuatro");
        spanish.add("cinco");
        spanish.add("seis");
        spanish.add("siete");
   spanish.addActionListener(this);
   spanish.addItemListener(this);

        //Build second list, which allows one selection at a time.
       //Defaults to none visible, only one selectable
        italian = new List();
        italian.add("uno");
        italian.add("due");
        italian.add("tre");
        italian.add("quattro");
        italian.add("cinque");
        italian.add("sei");
        italian.add("sette");
         italian.addActionListener(this);
         italian.addItemListener(this);

        //Add lists to the Applet.
        GridBagLayout gridBag = new GridBagLayout();
        setLayout(gridBag);

        //Can't put text area on right due to GBL bug
        //(can't span rows in any column but the first).
        output = new TextArea(10, 40);
```

```
        output.setEditable(false);
        GridBagConstraints tc = new GridBagConstraints();
        tc.fill = GridBagConstraints.BOTH;
        tc.weightx = 1.0;
        tc.weighty = 1.0;
        tc.gridheight = 2;
        gridBag.setConstraints(output, tc);
        add(output);

        GridBagConstraints lc = new GridBagConstraints();
        lc.fill = GridBagConstraints.VERTICAL;
        lc.gridwidth = GridBagConstraints.REMAINDER; //end row
        gridBag.setConstraints(spanish, lc);
        add(spanish);
        gridBag.setConstraints(italian, lc);
        add(italian);
    }

    public void actionPerformed(ActionEvent e) {
   List list = (List)(e.getSource());
        String language = (list == spanish) ?
                          "Spanish" : "Italian";
        output.append("Action event occurred on \""
                          + list.getSelectedItem()  + "\" in "
                          + language + "." + newline);
    }

    public void itemStateChanged(ItemEvent e) {
        List list = (List)(e.getItemSelectable());
        String language = (list == spanish) ?
                          "Spanish" : "Italian";

   int index = ((Integer)(e.getItem())).intValue();
   if (e.getStateChange() == ItemEvent.SELECTED) {
            output.append("Select event occurred on item #"
                          + index + " (\""
                          + list.getItem(index) + "\") in "
                          + language + "." + newline);
   } else { //the item was deselected
            output.append("Deselect event occurred on item #"
                          + index + " (\""
                          + list.getItem(index) + "\") in "
                          + language + "." + newline);
        }
    }
}
```

Quelques méthodes:

- **`int countItems()`** retourne le nombre d'items dans la liste.
- String getItem(int) retourne la chaîne associée à l'index.
- void addItem(String, int) ajoute un item à l'index spécifié
- void replaceItem(String, int) replace l'item à l'index spécifié.
- void clear(), void delItem(int), void delItems(int, int) supprime des items de la liste.
- int getSelectedIndex() retourne l'index de l'item sélectionné. -1 si aucun n'est sélectionné.
- int[] getSelectedIndexes() retourne le tableau des index des items sélectionnés.
- String getSelectedItem(), String[] getSelectedItems(), getSelectedIndexes(), idem que précédemment, mais sur les items.

- void select(int), void deselect(int) : sélectionne des items

3.9 Panel et Applet

Panel est un container quelconque.

Exemple d'une classe qui crée un cadre en "relief" autour de ses composants:

```
class FramedArea extends Panel {
              public FramedArea(Panel controller) {
                  ...//Set the layout manager.
                     //Add any Components this Panel contains...
                     super();

           //Set layout to one that makes its contents as big as possible.
                     setLayout(new GridLayout(1,0));

                     add(new CoordinateArea(controller));
                     validate();
              }

              //Ensure that no Component is placed on top of the frame.
              //The inset values were determined by trail and error.
              public Insets insets() {
                  return new Insets(4,4,5,5);
              }

              //Draw the frame at this Panel's edges.
              public void paint(Graphics g) {
                  Dimension d = size();
                  Color bg = getBackground();

                  g.setColor(bg);
                  g.draw3DRect(0, 0, d.width - 1, d.height - 1, true);
                  g.draw3DRect(3, 3, d.width - 7, d.height - 7, false);
              }
           }

           FramedArea p1 = new FramedArea();
            p1.add(new Button("Button 1"));
            p1.add(new Button("Button 2"));
            p1.add(new Button("Button 3"));
```

Applet est une sous-classe de Panel qui est contient du code lui permettant de fonctionner dans un browser.

3.10 TextArea et TextField

Sert à afficher et à lire du texte. Sont sous-classes de TextComponent.

TextField n'affiche qu'une ligne. TextArea affiche plusieurs lignes (et contient donc un scrollbar qui fonctionne de manière automatique.

```
public class TextDemo extends Applet implements ActionListener {
    TextField textField;
    TextArea textArea;
    String newline;

    public void init() {
        textField = new TextField(20);
        textArea = new TextArea(5, 20);
        textArea.setEditable(false);

        //Add Components to the Applet.
        GridBagLayout gridBag = new GridBagLayout();
        setLayout(gridBag);
```

```
        GridBagConstraints c = new GridBagConstraints();
        c.gridwidth = GridBagConstraints.REMAINDER;

        c.fill = GridBagConstraints.HORIZONTAL;
        gridBag.setConstraints(textField, c);
        add(textField);

        c.fill = GridBagConstraints.BOTH;
        c.weightx = 1.0;
        c.weighty = 1.0;
        gridBag.setConstraints(textArea, c);
        add(textArea);

    textField.addActionListener(this);

        newline = System.getProperty("line.separator");
    }

    public void actionPerformed(ActionEvent evt) {
        String text = textField.getText();
        textArea.append(text + newline);
        textField.selectAll();
    }
}
```

Quelques méthodes:

- **getText(), setText(String), selectAll(), setEditable(boolean), getSelectedText(), getSelectionStart(), getSelectionEnd()**: héritées de TextComponent.
- dans TextArea: **appendText(String), insertText(String),**

3.11 Frame et Dialog

Fenêtre spécifique détachée de la fenêtre principale. S'utilise en fait comme un Panel détaché. Dialog est aussi une fenêtre détachée mais qui dépend d'une fenêtre mère. Cette fenêtre peut aussi être modale.

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

public class MenuWindow extends Frame
                        implements ActionListener,
                                   ItemListener {
    boolean inAnApplet = true;
    TextArea output;
    PopupMenu popup;
    String newline;

    public MenuWindow() {
        MenuBar mb;
        Menu m1, m2, m3, m4, m4_1, m5;
        MenuItem mi1_1, mi1_2, mi3_1, mi3_2, mi3_3, mi3_4,
                 mi4_1_1, mi5_1, mi5_2,
                 pmi1, pmi2, mi5_1_duplicate;
        CheckboxMenuItem mi2_1;

        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                if (inAnApplet) {
                    dispose();
                } else {
                    System.exit(0);
                }
            }
```

```
});

newline = System.getProperty("line.separator");

//Add regular components to the window.
setLayout(new BorderLayout()); //max space: output
output = new TextArea(5, 30);
output.setEditable(false);
add("Center", output);
Label label = new Label("Try bringing up"
                        + " a popup menu!");
add("North", label);

//Build the menu bar.
mb = new MenuBar();
setMenuBar(mb);

//Build first menu in the menu bar.
//Specifying the second argument as true
//makes this a tear-off menu.
m1 = new Menu("Menu 1", true);
mb.add(m1);
mi1_1 = new MenuItem("Menu item 1_1");
m1.add(mi1_1);
mi1_2 = new MenuItem("Menu item 1_2");
m1.add(mi1_2);

//Build help menu.
m5 = new Menu("Help Menu");
mb.setHelpMenu(m5);
mi5_1 = new MenuItem("Menu item 5_1");
mi5_1.setShortcut(new MenuShortcut(KeyEvent.VK_5));
m5.add(mi5_1);
mi5_2 = new MenuItem("Menu item 5_2");
m5.add(mi5_2);

//Make a popup menu.
popup = new PopupMenu("A Popup Menu");
add(popup);
pmi1 = new MenuItem("A popup menu item");
popup.add(pmi1);
mi5_1_duplicate =
  new MenuItem("Duplicate of menu item 5_1",
                    new MenuShortcut(KeyEvent.VK_5));
popup.add(mi5_1_duplicate);
pmi2 = new MenuItem("An item with a shortcut",
                    new MenuShortcut(KeyEvent.VK_6));
popup.add(pmi2);

//Build second menu in the menu bar.
m2 = new Menu("Menu 2");
mb.add(m2);
mi2_1 = new CheckboxMenuItem("Menu item 2_1");
m2.add(mi2_1);

//Build third menu in the menu bar.
m3 = new Menu("Menu 3");
mb.add(m3);
mi3_1 = new MenuItem("Menu item 3_1");
m3.add(mi3_1);
mi3_2 = new MenuItem("Menu item 3_2");
m3.add(mi3_2);
m3.addSeparator();
mi3_3 = new MenuItem("Menu item 3_3");
m3.add(mi3_3);
mi3_4 = new MenuItem("Menu item 3_4");
mi3_4.setEnabled(false);
m3.add(mi3_4);
```

```
        //Build fourth menu in the menu bar.
        m4 = new Menu("Menu 4");
        mb.add(m4);
        m4_1 = new Menu("Submenu 4_1");
        m4.add(m4_1);
        mi4_1_1 = new MenuItem("Menu item 4_1_1");
        m4_1.add(mi4_1_1);

        //Register as an ActionListener for all menu items.
        m1.addActionListener(this);
        m2.addActionListener(this);
        m3.addActionListener(this);
        m4.addActionListener(this);
        mi4_1_1.addActionListener(this); //m4 can't detect
                       //submenu actions
        m5.addActionListener(this);
        popup.addActionListener(this);

        //Set action commands for a few menu items.
        mi1_1.setActionCommand("1_1");
        mi1_2.setActionCommand("1_2");
        mi5_1.setActionCommand("5_1");
        mi5_2.setActionCommand("5_2");
        pmi1.setActionCommand("popup item #1");
        mi5_1_duplicate.setActionCommand("5_1");
        pmi2.setActionCommand("popup item #2");

        //Register as ItemListener on checkbox menu item.
        mi2_1.addItemListener(this);

        //Listen for when the popup menu should be shown.
        MouseListener listener = new PopupListener();
        addMouseListener(listener);
        output.addMouseListener(listener);
        label.addMouseListener(listener);
    }

    class PopupListener extends MouseAdapter {
        public void mousePressed(MouseEvent e) {
            maybeShowPopup(e);
        }

        public void mouseReleased(MouseEvent e) {
            maybeShowPopup(e);
        }

        private void maybeShowPopup(MouseEvent e) {
            if (e.isPopupTrigger()) {
                popup.show(e.getComponent(),
                           e.getX(), e.getY());
            }
        }
    }

    public void actionPerformed(ActionEvent e) {
        output.append("\"" + e.getActionCommand()
                      + "\" action detected in menu labeled \""
                      + ((MenuItem)(e.getSource())).getLabel()
                      + "\"." + newline);
    }

    public void itemStateChanged(ItemEvent e) {
        output.append("Item state change detected on item \""
                      + e.getItem()
                      + "\" (state is "
                      + ((e.getStateChange() ==
                            ItemEvent.SELECTED)?
```

```
                            "selected)."
                        : "deselected).")
                    + newline);
    }

    public static void main(String[] args) {
        MenuWindow window = new MenuWindow();

        window.inAnApplet = false;
        window.setTitle("MenuWindow Application");
        window.setSize(450, 200);
        window.setVisible(true);
    }
}
```

3.12 Canvas

La classe Canvas existe pour être sous-classée. Elle ne fait rien par elle-même. Elle est donc la classe racine pour les composants "custom".

Les "lightweight" components sont héritées de Component.

3.13 Gestionnaires de présentation (layout managers)

Un objet qui contrôle l'emplacement des objets à l'intérieur d'un container.

Les gestionnaires disponibles:

- FlowLayout: place les objets sur une ligne. C'est aussi le gestionnaire par défaut.
- GridLayout: place les objets sur une grille (gauche-droite et puis bas-haut)
- BorderLayout: place les objets en les reliant aux extrémités du container (fenêtre, applet,...).
- CardLayout: est utilisé lorsqu'on a plusieurs ensembles de boutons différents.
- GridBagLayout: c'est le gestionnaire le plus flexible (et le plus complexe) de tous. Utilise des gestionnaire de "contraintes".

Pour utiliser un gestionnaire, il suffit de placer cette ligne lors de l'utilisation (ou de l'initialisation) du container.

```
aContainer.setLayout(new GridLayout());
```

Exemple:

```
import java.awt.*;

public class GridWindow extends Frame implements ActionListener {
    private boolean inAnApplet = true;

    public GridWindow() {

   //Construct a GridLayout with 2 columns and an unspecified
   //number of rows.
        setLayout(new GridLayout(0,2));
        setFont(new Font("Helvetica", Font.PLAIN, 14));

        Button b1 = new Button("Button 1"));
        Button b2 = new Button("2");
        Button b3 = new Button("Button 3");
        Button b4 = new Button("Long-Named Button 4"));
```

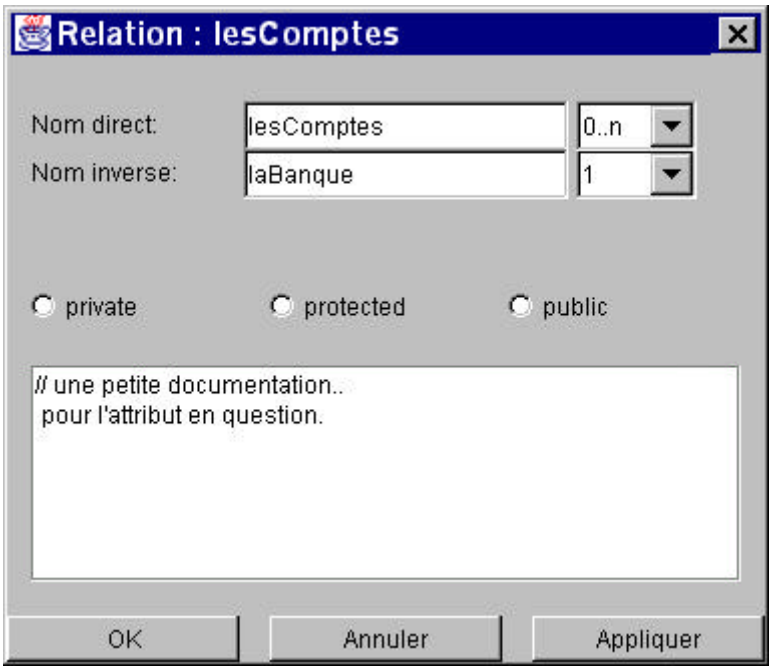
```
            Button b5 = new Button("Button 5"));

            add(b1)
            add(b2)
            add(b3);
            add(b4);
            add(b5);
        }

           b1.addActionListener(this);
           b2.addActionListener(this);
           b3.addActionListener(this);
           b4.addActionListener(this);
           b5.addActionListener(this);

        public void actionPerformed(ActionEvent e) {
            String command = e.getActionCommand();
            System.out.println("commande: " + command);
        }


        public static void main(String args[]) {
            GridWindow window = new GridWindow();

            window.setTitle("GridWindow Application");
            window.pack();
            window.show();
        }
    }
```

Remarques:

- On peut créer ses propres gestionnaires de présentation. Il y en a d'ailleurs pleins sur le Web...
- Si on place les composants "à la main", cela empêche le fonctionnement normal des gestionnaires de présentation.

3.14 Dialogues

Exemple un peu plus complet qui utilise les composants Swing: l'éditeur des relations dans le formalisme Object de SEdit.




```
import com.sun.java.swing.*;
import com.sun.java.swing.border.*;
import com.sun.java.swing.text.*;


class RelationEditor extends JDialog {

   public static String cardinTypes[]={"0..1", "1", "0..n", "1..n"};


   AssociationLink myLink;

   RelationEditor(AssociationLink a) {
      myLink = a;

      setSize(insets().left + insets().right + 380,insets().top +
                                          insets().bottom + 328);
      setTitle("Relation : " + a.getLabel());

      getContentPane().setLayout(new BorderLayout());

      JPanel pCentral = new JPanel();
      pCentral.setLayout(null);

      // les champs editables
      getContentPane().add(pCentral,"Center");
      // pCentral.setBounds(insets().left,insets().top,372,264);

      JLabel labNom = new JLabel("Nom direct:");
      labNom.setBounds(12,24,94,23);
      pCentral.add(labNom);

      JLabel labInverse = new JLabel("Nom inverse:");
      labInverse.setBounds(12,48,94,23);
      pCentral.add(labInverse);

      final JTextField texNom = new JTextField();
      texNom.setBounds(118,24,162,25);
      pCentral.add(texNom);
      texNom.setText(a.getLabel());

      final JTextField texInverse = new JTextField();
```

```
    texInverse.setBounds(118,48,162,25);
    pCentral.add(texInverse);
    texInverse.setText(a.getInverseName());

    final JComboBox cardDirect = new JComboBox();
    cardDirect.setBounds(284,24,60,25);
    cardDirect.setEditable(false);
    for(int i=0; i<cardinTypes.length;i++) {
       cardDirect.addItem(cardinTypes[i]);
    }
    cardDirect.setSelectedItem(a.getDirectCardinName());
    pCentral.add(cardDirect);

    final JComboBox cardInverse = new JComboBox();
    cardInverse.setBounds(284,48,60,25);
    cardInverse.setEditable(false);
    for(int i=0; i<cardinTypes.length;i++) {
       cardInverse.addItem(cardinTypes[i]);
    }
    cardInverse.setSelectedItem(a.getInverseCardinName());
    pCentral.add(cardInverse);


    // les radio buttons de portee
    JPanel rbGroup1 = new JPanel(new GridLayout(1,3,0,0));

    rbGroup1.setBounds(12,108,358,36);
    pCentral.add(rbGroup1);

    ButtonGroup rbg = new ButtonGroup();

    JRadioButton rbPrivate = new JRadioButton("private", false);
    rbPrivate.setBounds(0,0,113,26);
    rbGroup1.add(rbPrivate);
    rbg.add(rbPrivate);

    JRadioButton rbProtec = new JRadioButton("protected", false);
    rbProtec.setBounds(113,0,113,26);
    rbGroup1.add(rbProtec);
    rbg.add(rbProtec);

    JRadioButton rbPublic = new JRadioButton("public", false);
    rbPublic.setBounds(226,0,113,26);
    rbGroup1.add(rbPublic);
    rbg.add(rbPublic);

    /* String v = a.getVisibility();
    if (a.equals("private"))
           rbPrivate.setSelected(true);
      else if (a.equals("public"))
        rbPublic.setSelected(true);
      else
        rbProtec.setSelected(true); */


    // l'editeur de la documentation
/*  // Version avec les composants 1.1
    texAreaDoc = new TextArea();
    texAreaDoc.setBounds(12,156,354,108);
    pCentral.add(texAreaDoc);
    texAreaDoc.setText("HOP HOP."); */

    // version avec les composants Swing
    JTextArea textAreaDoc = new JTextArea("...");
    JScrollPane scroller = new JScrollPane();
    scroller.setBounds(12,156,354,108);
    scroller.getViewport().add(textAreaDoc);
    textAreaDoc.setFont(ClassEditor.defaultFont);
```

```
        pCentral.add(scroller);

        // les boutons de commande
        JPanel pButtons = new JPanel(new GridLayout(1,4,15,5));
        getContentPane().add(pButtons,"South");

        JButton b1 = new JButton("OK");
        pButtons.add(b1);
        JButton b2 = new JButton("Annuler");
        pButtons.add(b2);
        JButton b3 = new JButton("Appliquer");
        pButtons.add(b3);
        JButton b4 = new JButton("Aide");
        pButtons.add(b3);

        // Gestion des actions sur les boutons

        // OK
        b1.addActionListener(new ActionListener() {
                  public void actionPerformed(ActionEvent e){
                            myLink.updateRel(
                                  texNom.getText(),
                                  texInverse.getText(),
                                  (String) cardDirect.getSelectedItem(),
                                  (String) cardInverse.getSelectedItem());
                            myLink.update();
                            close();
                   }});

        // Annuler
        b2.addActionListener(new ActionListener() {
                  public void actionPerformed(ActionEvent e){
                            close();
                   }});

        // Appliquer
        b3.addActionListener(new ActionListener() {
                  public void actionPerformed(ActionEvent e){
                            myLink.updateRel(
                                  texNom.getText(),
                                  texInverse.getText(),
                                  (String) cardDirect.getSelectedItem(),
                                  (String) cardInverse.getSelectedItem());
                            myLink.update();
                   }});


        // pack();
        show();
    }
```