Swings dans JAVA

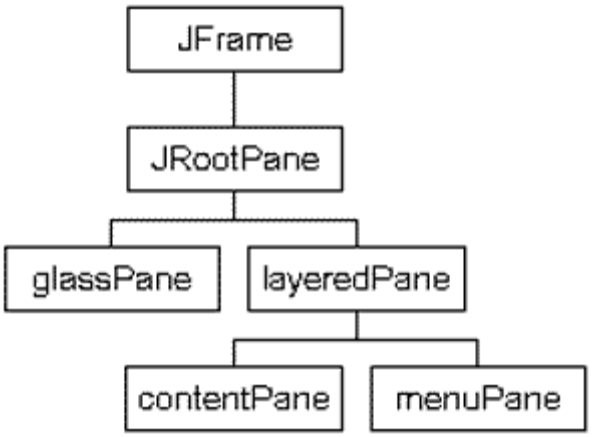
Table des matières

1. La classe JFRAME :

```
import javax.swing.*;

public class TestJFrame1 {

  public static void main(String argv[]) {
    JFrame f = new JFrame("ma fenetre");
    f.setSize(300,100);
    f.setVisible(true);
  }
}
```

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

public class swing2 extends JFrame {

   public swing2() {

     super("titre de l'application");

      WindowListener l = new WindowAdapter() {
         public void windowClosing(WindowEvent e){
           System.exit(0);
         }
      };
     addWindowListener(l);

     JButton bouton = new JButton("Mon bouton");
     JPanel panneau = new JPanel();
     panneau.add(bouton);

     setContentPane(panneau);
     setSize(200,100);
     setVisible(true);
   }

   public static void main(String [] args){
      JFrame frame = new swing2();
   }
}
```

Tous les composants associés à un objet JFrame sont gérés par un objet de la classe JRootPane. Un objet JRootPane contient plusieurs Panes. Tous les composants ajoutés au JFame doivent être ajoutés à un des Pane du JRootPane et non au JFrame directement. C'est aussi à un de ces Panes qu'il faut associer un layout manager si nécessaire.



1.1. Layout

Le Pane le plus utilisé est le ContentPane. Le Layout manager par défaut du contentPane est BorderLayout. Il est possible de le changer :

```
f.getContentPane().setLayout(new FlowLayout());
```

```
import javax.swing.*;

public class TestJFrame2 {

  public static void main(String argv[]) {

    JFrame f = new JFrame("ma fenetre");
    f.setSize(300,100);
    JButton b =new JButton("Mon bouton");
    f.getContentPane().add(b);
    f.setVisible(true);
  }
}
```

Le JRootPane se compose de plusieurs éléments :

- glassPane : un JPanel par défaut
- layeredPane qui se compose du contentPane (un JPanel par défaut) et du menuBar (un objet de type JMenuBar)

Le glassPane est un JPanel transparent qui se situe au-dessus du layeredPane. Le glassPane peut être n'importe quel composant : pour le modifier il faut utiliser la méthode setGlassPane() en fournissant le composant en paramètre.

Le layeredPane regroupe le contentPane et le menuBar.

Le contentPane est par défaut un JPanel opaque dont le gestionnaire de présentation est un BorderLayout. Ce panel peut être remplacé par n'importe quel composant grâce à la méthode setContentPane().

Attention : il ne faut pas utiliser directement la méthode setLayout() d'un objet JFrame sinon une exception est levée.

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

public class TestJFrame7 {

  public static void main(String argv[]) {

    JFrame f = new JFrame("ma fenetre");
    f.setLayout(new FlowLayout());
    f.setSize(300,100);
    f.setVisible(true);
  }
}
```

```
Exception in thread "main" java.lang.Error: Do not use
javax.swing.JFrame.setLay
out() use javax.swing.JFrame.getContentPane().setLayout() instead
        at javax.swing.JFrame.createRootPaneException(Unknown Source)
        at javax.swing.JFrame.setLayout(Unknown Source)
        at TestJFrame7.main(TestJFrame7.java:8)
```

1.2. JMenuBar

Le menuBar permet d'attacher un menu à la JFrame. Par défaut, le menuBar est vide. La méthode setJMenuBar() permet d'affecter un menu à la JFrame.

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

public class TestJFrame6 {

  public static void main(String argv[]) {

    JFrame f = new JFrame("ma fenetre");
    f.setSize(300,100);
    JButton b =new JButton("Mon bouton");
    f.getContentPane().add(b);

    JMenuBar menuBar = new JMenuBar();
    f.setJMenuBar(menuBar);

    JMenu menu = new JMenu("Fichier");
    menu.add(menuItem);
    menuBar.add(menu);

    f.setVisible(true);
  }
}
```

1.3. Le comportement par défaut à la fermeture

Il est possible de préciser comment un objet JFrame, JInternalFrame, ou JDialog réagit à sa fermeture grâce à la méthode setDefaultCloseOperation(). Cette méthode attend en paramètre une valeur qui peut être :

onstante	Rôle
WindowConstants.DISPOSE_ON_CLOSE	détruit la fenêtre
WindowConstants.DO_NOTHING_ON_CLOSE	rend le bouton de fermeture inactif
WindowConstants.HIDE_ON_CLOSE	cache la fenêtr

Cette méthode ne permet pas d'associer d'autres traitements. Dans ce cas, il faut intercepter l'événement et lui associer les traitements.

```
import javax.swing.*;

public class TestJFrame3 {

  public static void main(String argv[]) {

    JFrame f = new JFrame("ma fenetre");
    f.setSize(300,100);
    JButton b =new JButton("Mon bouton");
    f.getContentPane().add(b);

    f.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);

    f.setVisible(true);
  }
}
```

1.4. La personnalisation de l'icône

La méthode setIconImage() permet de modifier l'icône de la JFrame.

```
import javax.swing.*;

public class TestJFrame4 {

  public static void main(String argv[]) {

    JFrame f = new JFrame("ma fenetre");
    f.setSize(300,100);
    JButton b =new JButton("Mon bouton");
    f.getContentPane().add(b);
    f.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);

    ImageIcon image = new ImageIcon("book.gif");
    f.setIconImage(image.getImage());
    f.setVisible(true);

  }
}
```

1.5. Centrer une JFrame à l'écran

Par défaut, une JFrame est affichée dans le coin supérieur gauche de l'écran. Pour la centrer dans l'écran, il faut procéder comme pour une Frame : déterminer la position de la Frame en fonction de sa dimension et de celle de l'écran et utiliser la méthode setLocation() pour affecter cette position.

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

public class TestJFrame5 {

  public static void main(String argv[]) {

    JFrame f = new JFrame("ma fenetre");
    f.setSize(300,100);
    JButton b =new JButton("Mon bouton");
    f.getContentPane().add(b);

    f.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);

    Dimension dim = Toolkit.getDefaultToolkit().getScreenSize();
    f.setLocation(dim.width/2 - f.getWidth()/2, dim.height/2 -
f.getHeight()/2);

    f.setVisible(true);
  }
}
```

1.6. Les événements associées à un JFrame

La gestion des événements associés à un objet JFrame est identique à celle utilisée pour un objet de type Frame de AWT.

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

public class TestJFrame8 {

  public static void main(String argv[]) {

    JFrame f = new JFrame("ma fenetre");
    f.setSize(300,100);
    f.setVisible(true);
    f.addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
          System.exit(0);
        }
      }
    );

  }
}
```

1.7. La classe JLabel

Le composant JLabel propose les mêmes fonctionnalités que les intitulés AWT mais ils peuvent en plus contenir des icônes .

Constructeurs	Rôle
JLabel()	Création d'une instance sans texte ni image
JLabel(Icon)	Création d'une instance en précisant l'image
JLabel(Icon, int)	Création d'une instance en précisant l'image et l'alignement horizontal
JLabel(String)	Création d'une instance en précisant le texte
JLabel(String, Icon, int)	Création d'une instance en précisant le texte, l'image et l'alignement horizontal
JLabel(String, int)	Création d'une instance en précisant le texte et l'alignement horizontal

Le composant JLabel permet d'afficher un texte et/ou une icône en précisant leur alignement. L'icône doit être au format GIF et peut être une animation dans ce format.

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

public class TestJLabel1 {

  public static void main(String argv[]) {

    JFrame f = new JFrame("ma fenetre");
    f.setSize(100,200);

    JPanel pannel = new JPanel();
    JLabel jLabel1 =new JLabel("Mon texte dans JLabel");
    pannel.add(jLabel1);

    ImageIcon icone = new ImageIcon("book.gif");
    JLabel jLabel2 =new JLabel(icone);
    pannel.add(jLabel2);

    JLabel jLabel3 =new JLabel("Mon texte",icone,SwingConstants.LEFT);
    pannel.add(jLabel3);

    f.getContentPane().add(pannel);
    f.setVisible(true);
  }
}
```

La classe JLabel définit plusieurs méthodes pour modifier l'apparence du composant :

Méthodes	Rôle
setText()	Permet d'initialiser ou de modifier le texte affiché
setOpaque()	Indique si le composant est transparent (paramètre false) ou opaque (true)
setBackground()	Indique la couleur de fond du composant (setOpaque doit être à true)
setFont()	Permet de préciser la police du texte
setForeGround()	Permet de préciser la couleur du texte
setHorizontalAlignment()	Permet de modifier l'alignement horizontal du texte et de l'icône
setVerticalAlignment()	Permet de modifier l'alignement vertical du texte et de l'icône
setHorizontalTextAlignment()	Permet de modifier l'alignement horizontal du texte uniquement
setVerticalTextAlignment()	Permet de modifier l'alignement vertical du texte uniquement Exemple : jLabel.setVerticalTextPosition(SwingConstants.TOP);
setIcon()	Permet d'assigner une icône
setDisabledIcon()	Permet de définir l'icône associée au JLabel lorsqu'il est désactivé

L'alignement vertical par défaut d'un JLabel est centré. L'alignement horizontal par défaut est soit à droite s'il ne contient que du texte, soit centré s'il contient une image avec ou sans texte. Pour modifier cet alignement, il suffit d'utiliser les méthodes ci-dessus en utilisant des constantes en paramètres : SwingConstants.LEFT, SwingConstants.CENTER, SwingConstants.RIGHT, SwingConstants.TOP, SwingConstants.BOTTOM

Par défaut, un JLabel est transparent : son fond n'est pas dessiné. Pour le dessiner, il faut utiliser la méthode setOpaque() :

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

public class TestJLabel2 {

  public static void main(String argv[]) {

    JFrame f = new JFrame("ma fenetre");

    f.setSize(100,200);
    JPanel pannel = new JPanel();
```

```
    JLabel jLabel1 =new JLabel("Mon texte dans JLabel 1");
    jLabel1.setBackground(Color.red);
    pannel.add(jLabel1);

    JLabel jLabel2 =new JLabel("Mon texte dans JLabel 2");
    jLabel2.setBackground(Color.red);
    jLabel2.setOpaque(true);
    pannel.add(jLabel2);

    f.getContentPane().add(pannel);
    f.setVisible(true);
  }
}
```

Dans l'exemple, les 2 JLabel ont le fond rouge demandé par la méthode setBackground(). Seul le deuxième affiche un fond rouge car il est rendu opaque avec la méthode setOpaque().

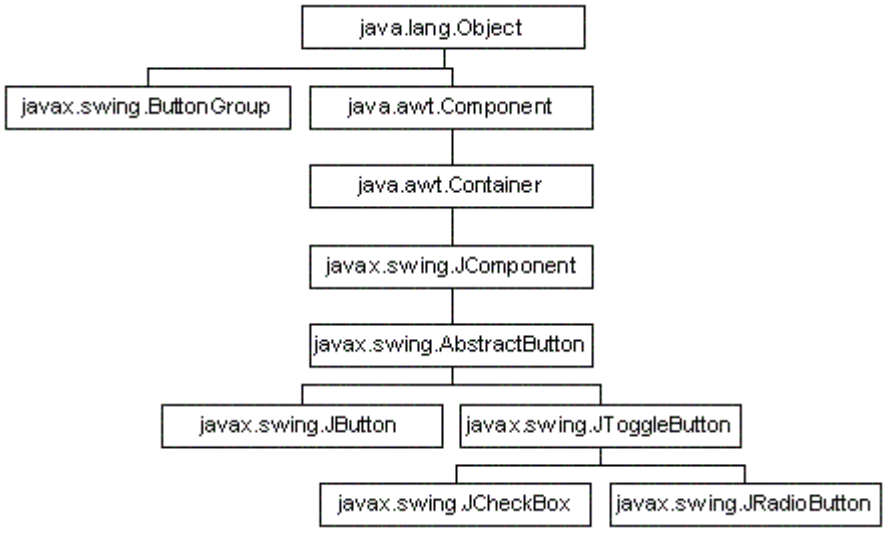
Il est possible d'associer un raccourci clavier au JLabel qui permet de donner le focus à un autre composant. La méthode setDisplayedMnemonic() permet de définir le raccourci clavier. Celui-ci sera activé en utilisant la touche Alt avec le caractère fourni en paramètre. La méthode setLabelFor() permet d'associer le composant fourni en paramètre au raccourci.

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

public class TestJLabel3 {

  public static void main(String argv[]) {

    JFrame f = new JFrame("ma fenetre");
    f.setSize(300,100);
    JPanel pannel = new JPanel();

    JButton bouton = new JButton("saisir");
    pannel.add(bouton);

    JTextField jEdit = new JTextField("votre nom");

    JLabel jLabel1 =new JLabel("Nom : ");
    jLabel1.setBackground(Color.red);
    jLabel1.setDisplayedMnemonic('n');
    jLabel1.setLabelFor(jEdit);
    pannel.add(jLabel1);
    pannel.add(jEdit);

    f.getContentPane().add(pannel);
    f.setVisible(true);
  }
}
```

Dans l'exemple, à l'ouverture de la fenêtre, le focus est sur le bouton. Un appui sur Alt+'n' donne le focus au champ de saisie.

2. Les boutons

Il existe plusieurs boutons définis par Swing.



2.1. La classe AbstractButton

C'est une classe abstraite dont héritent les boutons Swing JButton, JMenuItem et JToggleButton.

Cette classe définit de nombreuses méthodes dont les principales sont :

Méthode	Rôle
AddActionListener	Associer un écouteur sur un événement de type ActionEvent
AddChangeListener	Associer un écouteur sur un événement de type ChangeEvent
AddItemListener	Associer un écouteur sur un événement de type ItemEvent
doClick()	Déclencher un clic par programmation
getText()	Obtenir le texte affiché par le composant
setDisabledIcon()	Associer une icône affichée lorsque le composant a l'état désélectionné
setDisabledSelectedIcon()	Associer une icône affichée lors du passage de la souris sur le composant à l'état désélectionné
setEnabled()	Activer/désactiver le composant
setMnemonic()	Associer un raccourci clavier
setPressedIcon()	Associer une icône affichée lorsque le composant est cliqué
setRolloverIcon()	Associer une icône affichée lors du passage de la souris sur le composant

setRolloverSelectedIcon()	Associer une icône affichée lors du passage de la souris sur le composant à l'état sélectionné
setSelectedIcon()	Associer une icône affichée lorsque le composant a l'état sélectionné
setText()	Mettre à jour le texte du composant
isSelected()	Indiquer si le composant est dans l'état sélectionné
setSelected()	Définir l'état du composant (sélectionné ou non selon la valeur fournie en paramètre

Tous les boutons peuvent afficher du texte et/ou une image.

Il est possible de préciser une image différente lors du passage de la souris sur le composant et lors de l'enfoncement du bouton : dans ce cas, il faut créer trois images pour chacun des états (normal, enfoncé et survolé). L'image normale est associée au bouton grâce au constructeur, l'image enfoncée grâce à la méthode setPressedIcon() et l'image lors d'un survol grâce à la méthode setRolloverIcon(). Il suffit enfin d'appeler la méthode setRolloverEnable() avec en paramètre la valeur true.

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

public class swing4 extends JFrame {

   public swing4() {
     super("titre de l'application");

     WindowListener l = new WindowAdapter() {
       public void windowClosing(WindowEvent e){
         System.exit(0);
       }
     };
     addWindowListener(l);

     ImageIcon imageNormale = new ImageIcon("arrow.gif");
     ImageIcon imagePassage =  new ImageIcon("arrowr.gif");
     ImageIcon imageEnfoncee = new ImageIcon("arrowy.gif");

     JButton bouton = new JButton("Mon bouton",imageNormale);
     bouton.setPressedIcon(imageEnfoncee);
     bouton.setRolloverIcon(imagePassage);
     bouton.setRolloverEnabled(true);
     getContentPane().add(bouton, "Center");

     JPanel panneau = new JPanel();
     panneau.add(bouton);
     setContentPane(panneau);
     setSize(200,100);
     setVisible(true);
   }

   public static void main(String [] args){
      JFrame frame = new swing4();
   }
}
```

Un bouton peut recevoir des événements de type ActionEvents (le bouton a été activé), ChangeEvents, et ItemEvents.

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

public class TestJButton3 {

  public static void main(String argv[]) {

    JFrame f = new JFrame("ma fenetre");
    f.setSize(300,100);
    JPanel pannel = new JPanel();

    JButton bouton1 = new JButton("Bouton1");
    bouton1.addActionListener( new ActionListener() {
       public void actionPerformed(ActionEvent e) {
         System.exit(0);
       }
    }
    );

    pannel.add(bouton1);
    f.getContentPane().add(pannel);
    f.setVisible(true);
  }
}
```

2.2. La classe JButton

JButton est un composant qui représente un bouton : il peut contenir un texte et/ou une icône.

Les constructeurs sont :

Constructeur	Rôle
JButton()	
JButton(String)	préciser le texte du bouton
JButton(Icon)	préciser une icône
JButton(String, Icon)	préciser un texte et une icône

Toutes les indications concernant le contenu du composant JLabel sont valables pour le composant JButton.

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

public class swing3 extends JFrame {

   public swing3() {

      super("titre de l'application");
```

```
      WindowListener l = new WindowAdapter() {
         public void windowClosing(WindowEvent e){
            System.exit(0);
         }
      };
      addWindowListener(l);

      ImageIcon img = new ImageIcon("tips.gif");
      JButton bouton = new JButton("Mon bouton",img);

      JPanel panneau = new JPanel();
      panneau.add(bouton);
      setContentPane(panneau);
      setSize(200,100);
      setVisible(true);
  }

  public static void main(String [] args){
      JFrame frame = new swing3();
  }

}
```

Dans un conteneur de type JRootPane, il est possible de définir un bouton par défaut grâce à sa méthode setDefaultButton().

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

public class TestJButton2 {

  public static void main(String argv[]) {

    JFrame f = new JFrame("ma fenetre");
    f.setSize(300,100);
    JPanel pannel = new JPanel();
    JButton bouton1 = new JButton("Bouton 1");
    pannel.add(bouton1);

    JButton bouton2 = new JButton("Bouton 2");
    pannel.add(bouton2);

    JButton bouton3 = new JButton("Bouton 3");
    pannel.add(bouton3);

    f.getContentPane().add(pannel);
    f.getRootPane().setDefaultButton(bouton3);
    f.setVisible(true);
  }
}
```

Le bouton par défaut est activé par un appui sur la touche Entrée alors que le bouton actif est activé par un appui sur la barre d'espace.

La méthode isDefaultButton() de JButton permet de savoir si le composant est le bouton par défaut.

2.3. La classe JToggleButton

Cette classe définit un bouton à deux états : c'est la classe mère des composants JCheckBox et JRadioButton.

La méthode setSelected() héritée de AbstractButton permet de mettre à jour l'état du bouton. La méthode isSelected() permet de connaître cet état.

2.4. La classe ButtonGroup

La classe ButtonGroup permet de gérer un ensemble de boutons en garantissant qu'un seul bouton du groupe sera sélectionné.

Pour utiliser la classe ButtonGroup, il suffit d'instancier un objet et d'ajouter des boutons (objets héritant de la classe AbstractButton) grâce à la méthode add(). Il est préférable d'utiliser des objets de la classe JToggleButton ou d'une de ses classes filles car elles sont capables de gérer leurs états.

```
import javax.swing.*;

public class TestGroupButton1 {

  public static void main(String argv[]) {

    JFrame f = new JFrame("ma fenetre");
    f.setSize(300,100);
    JPanel pannel = new JPanel();

    ButtonGroup groupe = new ButtonGroup();
    JRadioButton bouton1 = new JRadioButton("Bouton 1");
    groupe.add(bouton1);
    pannel.add(bouton1);
    JRadioButton bouton2 = new JRadioButton("Bouton 2");
    groupe.add(bouton2);
    pannel.add(bouton2);
    JRadioButton bouton3 = new JRadioButton("Bouton 3");
    groupe.add(bouton3);
    pannel.add(bouton3);

    f.getContentPane().add(pannel);
    f.setVisible(true);
  }
}
```

3. La classe JCheckBox

Les constructeurs sont les suivants :

Constructeur	Rôle
JCheckBox(String)	précise l'intitulé
JCheckBox(String, boolean)	précise l'intitulé et l'état

JCheckBox(Icon)	spécifie l'icône utilisée
JCheckBox(Icon, boolean)	précise l'intitulé et l'état du bouton
JCheckBox(String, Icon)	précise l'intitulé et l'icône
JCheckBox(String, Icon, boolean)	précise l'intitulé, une icône et l'état

Un groupe de cases à cocher peut être défini avec la classe ButtonGroup. Dans ce cas, un seul composant du groupe peut être sélectionné. Pour l'utiliser, il faut créer un objet de la classe ButtonGroup et utiliser la méthode add() pour ajouter un composant au groupe.

Exemple (code Java 1.1) :

```
import javax.swing.*;

public class TestJCheckBox1 {

  public static void main(String argv[]) {

    JFrame f = new JFrame("ma fenetre");
    f.setSize(300,100);
    JPanel pannel = new JPanel();

    JCheckBox bouton1 = new JCheckBox("Bouton 1");
    pannel.add(bouton1);
    JCheckBox bouton2 = new JCheckBox("Bouton 2");
    pannel.add(bouton2);
    JCheckBox bouton3 = new JCheckBox("Bouton 3");
    pannel.add(bouton3);

    f.getContentPane().add(pannel);
    f.setVisible(true);
  }
}
```

4. La classe JRadioButton

Un objet de type JRadioButton représente un bouton radio d'un groupe de boutons . A un instant donné, un seul des boutons radio associés à un même groupe peut être sélectionné. La classe JRadioButton hérite de la classe AbstractButton.

Un bouton radio possède un libellé et éventuellement une icône qui peut être précisée, pour chacun des états du bouton, en utilisant les méthodes setIcon(), setSelectedIcon() et setPressedIcon().

Exemple (code Java 1.1) :

```
import javax.swing.*;
```

```
public class TestJRadioButton1 {

  public static void main(String argv[]) {

    JFrame f = new JFrame("ma fenetre");
    f.setSize(300,100);
    JPanel pannel = new JPanel();
    JRadioButton bouton1 = new JRadioButton("Bouton 1");
    pannel.add(bouton1);
    JRadioButton bouton2 = new JRadioButton("Bouton 2");
    pannel.add(bouton2);
    JRadioButton bouton3 = new JRadioButton("Bouton 3");
    pannel.add(bouton3);

    f.getContentPane().add(pannel);
    f.setVisible(true);
  }
}
```

La méthode isSelected() permet de savoir si le bouton est sélectionné ou non.

La classe JRadioButton possède plusieurs constructeurs :

Constructeur	Rôle
JRadioButton()	Créer un bouton non sélectionné sans libellé
JRadioButton(Icon)	Créer un bouton non sélectionné sans libellé avec l'icône fournie en paramètre
JRadioButton(Icon, boolean)	Créer un bouton sans libellé avec l'icône et l'état fournis en paramètres
JRadioButton(String)	Créer un bouton non sélectionné avec le libellé fourni en paramètre
JRadioButton(String, boolean)	Créer un bouton avec le libellé et l'état fournis en paramètres
JRadioButton(String, Icon)	Créer un bouton non sélectionné avec le libellé et l'icône fournis en paramètres
JRadioButton(String, Icon, boolean)	Créer un bouton avec le libellé, l'icône et l'état fournis en paramètres

Un groupe de boutons radio est encapsulé dans un objet de type ButtonGroup.

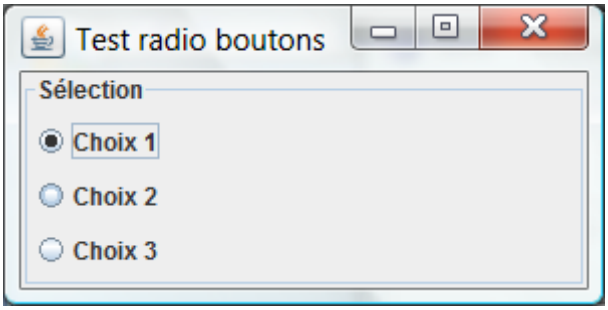
Il faut ajouter tous les JRadioButton du groupe en utilisant la méthode add() de la classe ButtonGroup. Lors de la sélection d'un bouton, c'est l'objet de type ButtonGroup qui se charge de déselectionner le bouton précédemment sélectionné dans le groupe.

Un groupe n'a pas l'obligation d'avoir un bouton sélectionné.

Exemple :

```
import java.awt.BorderLayout;
```

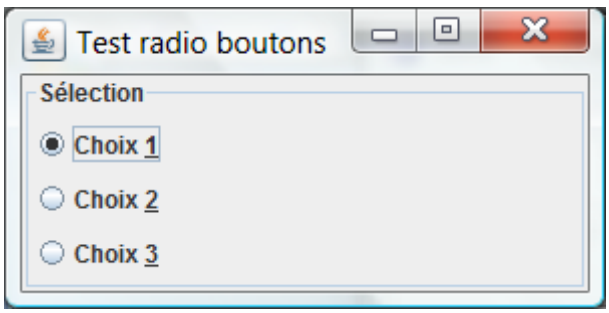
```
import java.awt.Container;
import java.awt.GridLayout;
import javax.swing.BorderFactory;
import javax.swing.ButtonGroup;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JRadioButton;
import javax.swing.border.Border;

public class TestJRadioButton extends JFrame {
  public static void main(String args[]) {
    TestJRadioButton app = new TestJRadioButton();
    app.init();
  }

  public void init() {
    this.setTitle("Test radio boutons");

    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    JPanel panel = new JPanel(new GridLayout(0,1));
    Border border = BorderFactory.createTitledBorder("Sélection");
    panel.setBorder(border);
    ButtonGroup group = new ButtonGroup();
    JRadioButton radio1 = new JRadioButton("Choix 1", true);
    JRadioButton radio2 = new JRadioButton("Choix 2");
    JRadioButton radio3 = new JRadioButton("Choix 3");
    group.add(radio1);
    panel.add(radio1);
    group.add(radio2);
    panel.add(radio2);
    group.add(radio3);
    panel.add(radio3);
    Container contentPane = this.getContentPane();
    contentPane.add(panel, BorderLayout.CENTER);
    this.setSize(300, 150);
    this.setVisible(true);
  }
}
```



Exemple :

```
import java.awt.BorderLayout;
import java.awt.Container;
import java.awt.GridLayout;
import java.awt.event.KeyEvent;
import javax.swing.BorderFactory;
import javax.swing.ButtonGroup;
import javax.swing.JFrame;
import javax.swing.JPanel;
```

```
import javax.swing.JRadioButton;
import javax.swing.border.Border;

public class TestJRadioButton extends JFrame {
  public static void main(String args[]) {
    TestJRadioButton app = new TestJRadioButton();
    app.init();
  }

  public void init() {
    this.setTitle("Test radio boutons");
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    JPanel panel = new JPanel(new GridLayout(0, 1));
    Border border = BorderFactory.createTitledBorder("Sélection");
    panel.setBorder(border);

    ButtonGroup group = new ButtonGroup();
    JRadioButton radio1 = new JRadioButton("Choix 1");
    radio1.setMnemonic(KeyEvent.VK_1);
    radio1.setActionCommand("Choix_1");
    radio1.setSelected(true);

    JRadioButton radio2 = new JRadioButton("Choix 2");
    radio2.setMnemonic(KeyEvent.VK_2);
    radio2.setActionCommand("Choix_2");

    JRadioButton radio3 = new JRadioButton("Choix 3");
    radio3.setMnemonic(KeyEvent.VK_3);
    radio3.setActionCommand("Choix_3");

    group.add(radio1);
    panel.add(radio1);
    group.add(radio2);
    panel.add(radio2);
    group.add(radio3);
    panel.add(radio3);

    Container contentPane = this.getContentPane();
    contentPane.add(panel, BorderLayout.CENTER);
    this.setSize(300, 150);
    this.setVisible(true);
  }
}
```



Lors de la sélection d'un bouton du groupe, il y a plusieurs événements qui peuvent être émis :

- Un événement de type Action
- Un événement de type Item émis par le bouton sélectionné
- Un événement de type Item émis par le bouton désélectionné s'il y en a un

Exemple :

```
import java.awt.BorderLayout;
import java.awt.Container;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import java.awt.event.KeyEvent;
import javax.swing.BorderFactory;
import javax.swing.ButtonGroup;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JRadioButton;
import javax.swing.border.Border;

public class TestJRadioButton extends JFrame implements
ActionListener, ItemListener {
  public static void main(String args[]) {
    TestJRadioButton app = new TestJRadioButton();
    app.init();
  }

  public void init() {
    this.setTitle("Test radio boutons");

    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    JPanel panel = new JPanel(new GridLayout(0, 1));
    Border border = BorderFactory.createTitledBorder("Sélection");
    panel.setBorder(border);
    ButtonGroup group = new ButtonGroup();

    JRadioButton radio1 = new JRadioButton("Choix 1");
    radio1.setMnemonic(KeyEvent.VK_1);
    radio1.setActionCommand("Choix_1");
    radio1.setSelected(true);

    JRadioButton radio2 = new JRadioButton("Choix 2");
    radio2.setMnemonic(KeyEvent.VK_2);
    radio2.setActionCommand("Choix_2");

    JRadioButton radio3 = new JRadioButton("Choix 3");
    radio3.setMnemonic(KeyEvent.VK_3);
    radio3.setActionCommand("Choix_3");

    group.add(radio1);
    panel.add(radio1);
    group.add(radio2);
    panel.add(radio2);
    group.add(radio3);
    panel.add(radio3);

    radio1.addActionListener(this);
    radio2.addActionListener(this);
    radio3.addActionListener(this);
    radio1.addItemListener(this);
    radio2.addItemListener(this);
    radio3.addItemListener(this);
```

```
    Container contentPane = this.getContentPane();
    contentPane.add(panel, BorderLayout.CENTER);
    this.setSize(300, 150);
    this.setVisible(true);
  }

  @Override
  public void actionPerformed(ActionEvent e) {
    System.out.println("Clic sur le bouton : " +
e.getActionCommand());
  }

  @Override
  public void itemStateChanged(ItemEvent e) {
    System.out.print("Bouton " + ((JRadioButton)
e.getItem()).getActionCommand());
    if (e.getStateChange() == ItemEvent.DESELECTED)
      System.out.println(" deselectionne");
    if (e.getStateChange() == ItemEvent.SELECTED)
      System.out.println(" selectionne");
  }
}
```

La méthode getSelection() de la classe ButtonGroup renvoie le modèle du bouton radio sélectionné encapsulé dans un objet de type ButtonModel.

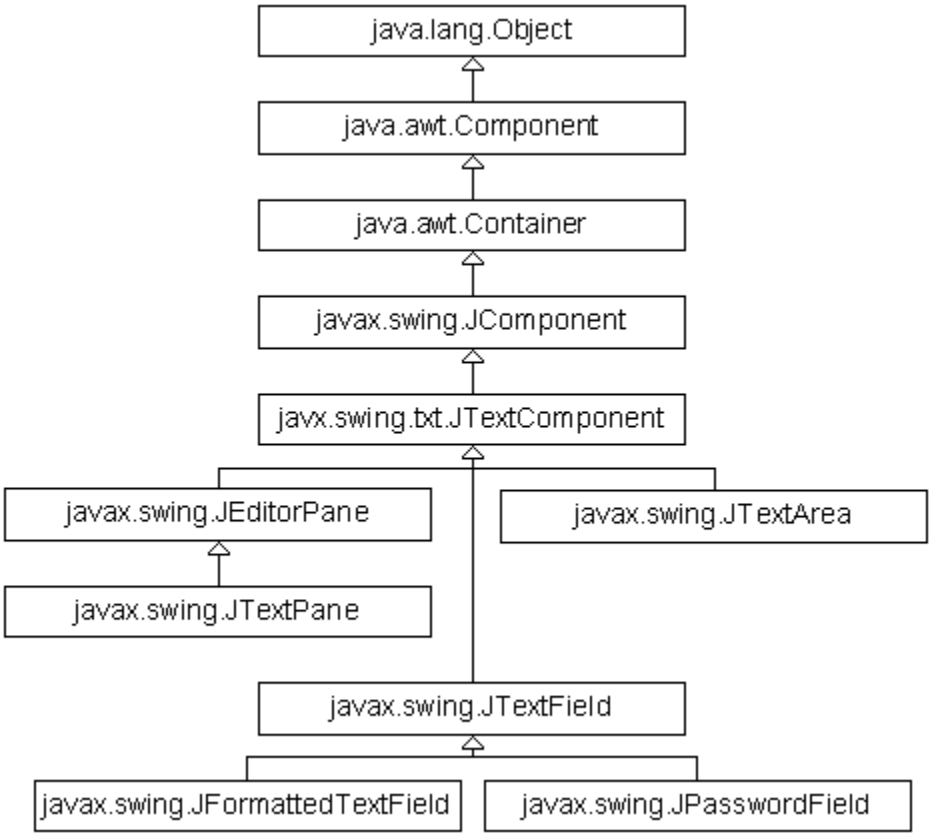
Pour déterminer le bouton sélectionné, il faut parcourir les boutons du groupe et comparer leurs modèles.

Exemple :

```
  public static JRadioButton getBoutonSelectionne(ButtonGroup
group) {
    JRadioButton result = null;
    for (Enumeration e = group.getElements(); e.hasMoreElements();)
{
      JRadioButton bouton = (JRadioButton) e.nextElement();
      if (bouton.getModel() == group.getSelection()) {
        result = bouton;
        break;
      }
    }
    return result;
  }
```

5. Les composants de saisie de texte

Swing possède plusieurs composants pour permettre la saisie de texte.



5.1. La classe JTextComponent

La classe abstraite JTextComponent est la classe mère de tous les composants permettant la saisie de texte.

Les données du composant (le modèle dans le motif de conception MVC) sont encapsulées dans un objet qui implémente l'interface Document. Deux classes implémentant cette interface sont fournies en standard : PlainDocument pour du texte simple et StyledDocument pour du texte riche pouvant contenir entre autres plusieurs polices de caractères, des couleurs, des images, ...

La classe JTextComponent possède de nombreuses méthodes dont les principales sont :

Méthode	Rôle
void copy()	Copier le contenu du texte et le mettre dans le presse papier système
void cut()	Couper le contenu du texte et le mettre dans le presse papier système
Document getDocument()	Renvoyer l'objet de type Document qui encapsule le texte saisi

String getSelectectedText()	Renvoyer le texte sélectionné dans le composant
int getSelectionEnd()	Renvoyer la position de la fin de la sélection
int getSelectionStart()	Renvoyer la position du début de la sélection
String getText()	Renvoyer le texte saisi
String getText(int, int)	Renvoyer une portion du texte débutant à partir de la position donnée par le premier paramètre et la longueur donnée dans le second paramètre
bool isEditable()	Renvoyer un booléen qui précise si le texte est éditable ou non
void paste()	Coller le contenu du presse papier système dans le composant
void select(int,int)	Sélectionner une portion du texte dont les positions de début et de fin sont fournies en paramètres
void setCaretPosition(int)	Déplacer le curseur dans le texte à la position précisé en paramètre
void setEditable(boolean)	Permet de préciser si les données du composant sont éditables ou non
void setSelectionEnd(int)	Modifier la position de la fin de la sélection
void setSelectionStart(int)	Modifier la position du début de la sélection
void setText(String)	Modifier le contenu du texte

Toutes ces méthodes sont donc accessibles grâce à l'héritage pour tous les composants de saisie de texte proposés par Swing.

5.2. La classe JTextField

La classe javax.Swing.JTextField est un composant qui permet la saisie d'une seule ligne de texte simple. Son modèle utilise un objet de type PlainDocument.

Exemple (code Java 1.1) :

```
import javax.swing.*;
```

```
public class JTextField1 {

  public static void main(String argv[]) {

    JFrame f = new JFrame("ma fenetre");
    f.setSize(300, 100);
    JPanel pannel = new JPanel();

    JTextField  testField1 = new JTextField ("mon texte");

    pannel.add(testField1);
    f.getContentPane().add(pannel);
    f.setVisible(true);
  }
}
```

La propriété horizontalAligment permet de préciser l'alignement du texte dans le composant en utilisant les valeurs JTextField.LEFT , JTextField.CENTER ou JTextField.RIGHT.

5.3. La classe JPasswordField

La classe JPasswordField permet la saisie d'un texte dont tous les caractères saisis seront affichés sous la forme d'un caractère particulier ('*' par défaut). Cette classe hérite de la classe JTextField.

Exemple (code Java 1.1) :

```
import java.awt.Dimension;

import javax.swing.*;

public class JPasswordField1 {

  public static void main(String argv[]) {

    JFrame f = new JFrame("ma fenetre");
    f.setSize(300, 100);
    JPanel pannel = new JPanel();

    JPasswordField  passwordField1 = new JPasswordField ("");
    passwordField1.setPreferredSize(new Dimension(100,20 ));

    pannel.add(passwordField1);
    f.getContentPane().add(pannel);
    f.setVisible(true);
  }
}
```

La méthode setEchoChar(char) permet de préciser le caractère qui sera montré lors de la saisie.

Il ne faut pas utiliser la méthode getText() qui est déclarée deprecated mais la méthode getPassword() pour obtenir la valeur du texte saisi.

Exemple (code Java 1.1) :

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

import javax.swing.*;

public class JPasswordField2 implements ActionListener {

  JPasswordField passwordField1 = null;

  public static void main(String argv[]) {
    JPasswordField2 jpf2 = new JPasswordField2();
    jpf2.init();
  }

  public void init() {
    JFrame f = new JFrame("ma fenetre");
    f.setSize(300, 100);
    JPanel pannel = new JPanel();

    passwordField1 = new JPasswordField("");
    passwordField1.setPreferredSize(new Dimension(100, 20));
    pannel.add(passwordField1);

    JButton bouton1 = new JButton("Afficher");
    bouton1.addActionListener(this);

    pannel.add(bouton1);
    f.getContentPane().add(pannel);
    f.setVisible(true);
  }

  public void actionPerformed(ActionEvent e) {

    System.out.println("texte saisie = " +
String.copyValueOf(passwordField1.getPassword()));
  }

}
```

Les méthodes copy() et cut() sont redéfinies pour n'émettre qu'un bip. Elles empêchent l'exportation du contenu du champ.

5.4. La classe JFormattedTextField

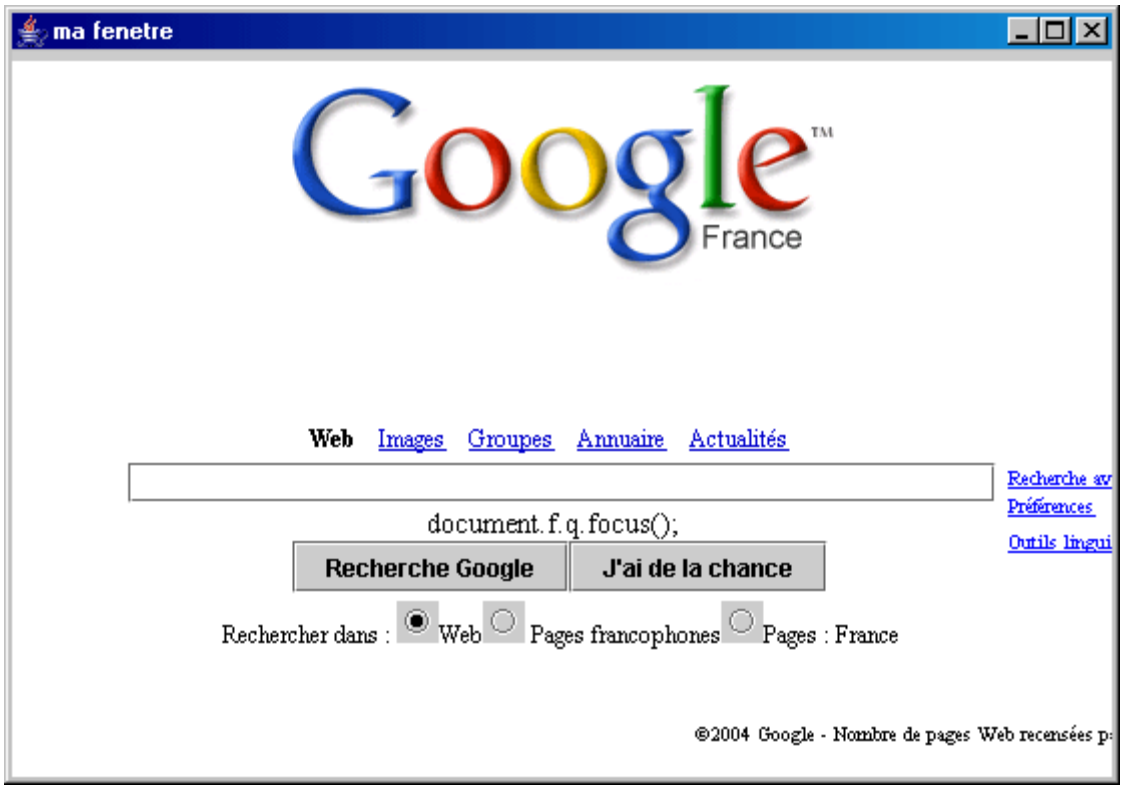
Le JDK 1.4 propose la classe JFormattedTextField pour faciliter la création d'un composant de saisie personnalisé. Cette classe hérite de la classe JTextField.

5.5. La classe JEditorPane

Ce composant permet la saisie de texte riche multilignes. Ce type de texte peut contenir des informations de mise en pages et de formatage. En standard, Swing propose le support des formats RTF et HTML.

Exemple (code Java 1.1) : affichage de la page de Google avec gestion des hyperliens

```
import java.net.URL;
import javax.swing.*;
import javax.swing.event.*;

public class JEditorPane1 {

  public static void main(String[] args) {
    final JEditorPane editeur;
    JPanel pannel = new JPanel();

    try {
      editeur = new JEditorPane(new URL("http://google.fr"));
      editeur.setEditable(false);
      editeur.addHyperlinkListener(new HyperlinkListener() {
        public void hyperlinkUpdate(HyperlinkEvent e) {
          if (e.getEventType() ==
HyperlinkEvent.EventType.ACTIVATED) {
            URL url = e.getURL();
            if (url == null)
              return;
            try {
              editeur.setPage(e.getURL());
            } catch (Exception ex) {
              ex.printStackTrace();
            }
          }
        }
      });

      pannel.add(editeur);
    } catch (Exception e1) {
      e1.printStackTrace();
    }
    JFrame f = new JFrame("ma fenetre");
    f.setSize(500, 300);

    f.getContentPane().add(pannel);
    f.setVisible(true);

  }
}
```



5.6. La classe JTextArea

La classe JTextArea est un composant qui permet la saisie de texte simple en mode multiligne. Le modèle utilisé par ce composant est le PlainDocument : il ne peut donc contenir que du texte brut sans éléments multiples de formatage.

JTexteArea propose plusieurs méthodes pour ajouter du texte dans son modèle :

- soit fournir le texte en paramètre du constructeur utilisé
- soit utiliser la méthode setText() qui permet d'initialiser le texte du composant
- soit utiliser la méthode append() qui permet d'ajouter du texte à la fin de celui contenu dans le composant
- soit utiliser la méthode insert() qui permet d'insérer du texte dans le composant à une position données en caractères

La méthode replaceRange() permet de remplacer la partie de texte occupant les index donnés en paramètres par la chaîne fournie.

La propriété rows permet de définir le nombre de lignes affichées par le composant : cette propriété peut donc être modifiée lors d'un redimensionnement du composant. La propriété lineCount en lecture seule permet de savoir le nombre de lignes qui composent le texte. Il ne faut pas confondre ces deux propriétés.

Exemple (code Java 1.1) :

```
import javax.swing.*;
```

```
public class JTextArea1 {

  public static void main(String argv[]) {

    JFrame f = new JFrame("ma fenetre");
    f.setSize(300, 100);
    JPanel pannel = new JPanel();

    JTextArea  textArea1 = new JTextArea ("mon texte");

    pannel.add(textArea1);
    f.getContentPane().add(pannel);
    f.setVisible(true);
  }
}
```

Par défaut, la taille du composant augmente au fur et à mesure de l'augmentation de la taille du texte qu'il contient. Pour éviter cet effet, il faut encapsuler le JTextArea dans un JScrollPane.

Exemple (code Java 1.1) :

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

public class JTextArea1 {

  public static void main(String argv[]) {

    JFrame f = new JFrame("ma fenetre");
    f.setSize(300, 100);
    JPanel pannel = new JPanel();
    JTextArea  textArea1 = new JTextArea ("mon texte");
    JScrollPane scrollPane = new JScrollPane(textArea1);
    scrollPane.setPreferredSize(new Dimension(200,70));
    pannel.add(scrollPane);
    f.getContentPane().add(pannel);
    f.setVisible(true);
  }
}
```