Introduction

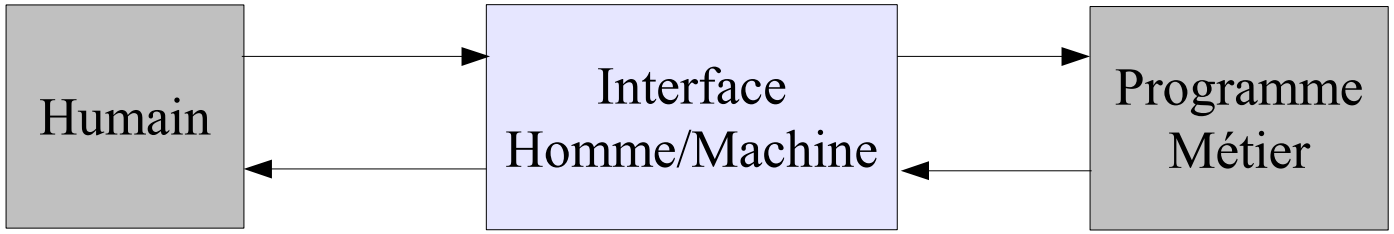
- Interface Graphique en Java
- AWT, Swing et SWT
- Hiérarchie de composants
- Composants Basiques

Livre et ressources

- Ce cours a été écrit à partir du cours de Jean Berstel.
- La plupart des exemples de programmes sont inspirés des exemples figurant dans les livres suivants:
  - M. Robinson, P. Vorobiev, “Swing”, http://manning.spindoczine.com/sbe 2nd edition (February 2003), Manning Publ. Co.
  - J. Knudsen, “Java 2D Graphics”, O’Reilly 1999.
  - C. Haase, R.Guy, “Filthy Rich Clients“, Java Series, 2007
- Pour des compléments Java, voir :
  - Java et Internet - Concepts et programmation 2e édition, Tome 1 - Coté client, Vuibert 2002. Gilles Roussel, Etienne Duris, Nicolas Bedon et Rémi Forax

Interface Graphique en Java

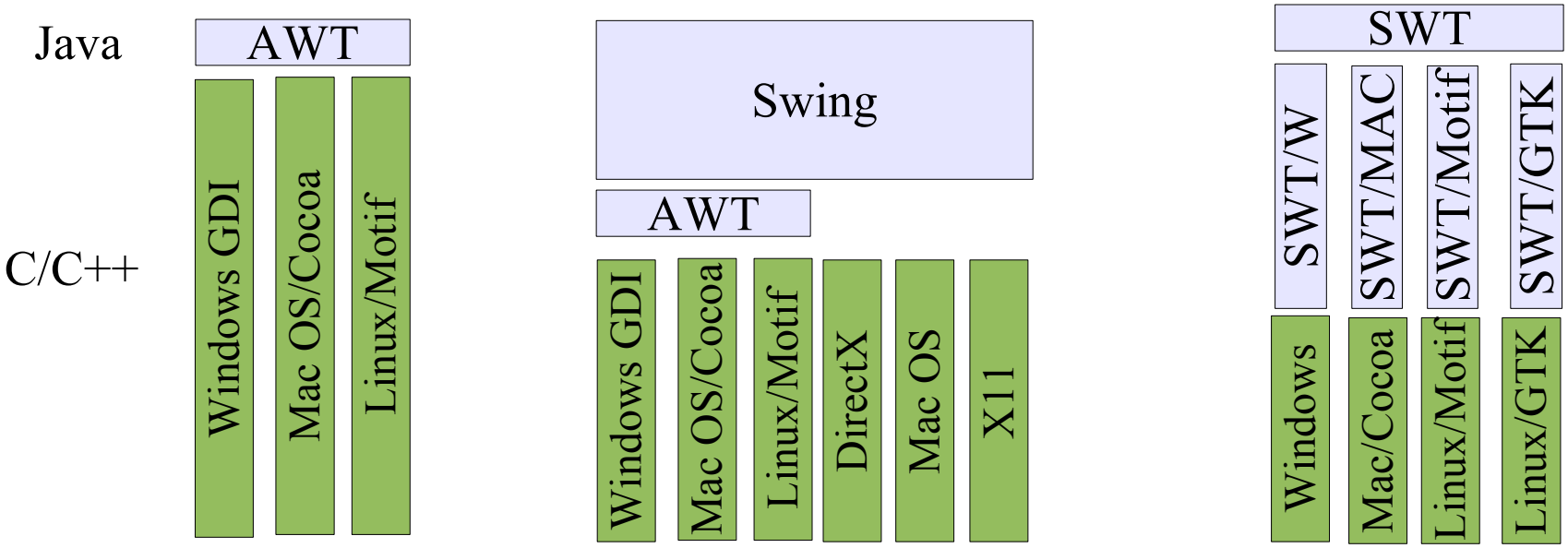
- Interface graphique ou Interface Homme Machine



- Problème spécifique à Java :
  Java est « *cross-platform* », une interface graphique développée sur une platforme doit fonctionner de façon identique sur une autre plateforme.

AWT, Swing et SWT

- Il existe trois toolkits de fenêtrage en Java
  - AWT (SUN, à partir du JDK 1.0)
  - Swing (SUN, à partir du JDK 1.2)
  - SWT (IBM, à partir du JDK 1.2)

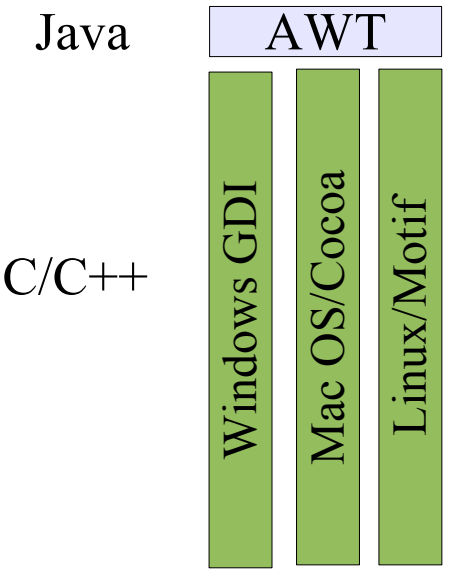


AWT (Abstract Windowing Toolkit)

- AWT, chaque composant Java possède un « peer » C++ correspondant au composant de la plateforme

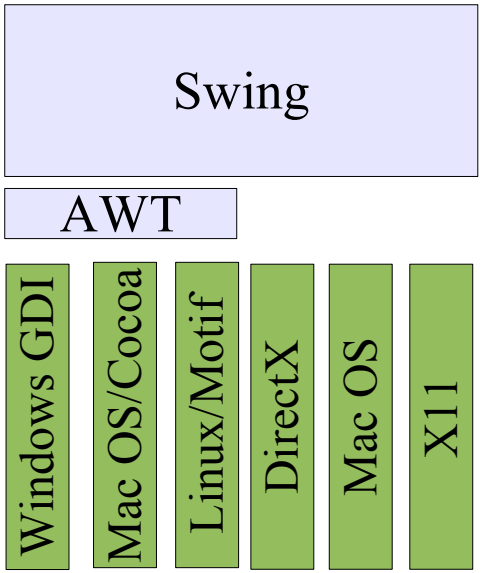
- Problème, AWT est limité par les limitations des composants de la plateforme qui ne marchent pas pareil suivant les plateformes

  Exemple :
  - limitation de la zone de texte
    Windows à 65k
  - impossible d'afficher les caractères kanji avec Motif



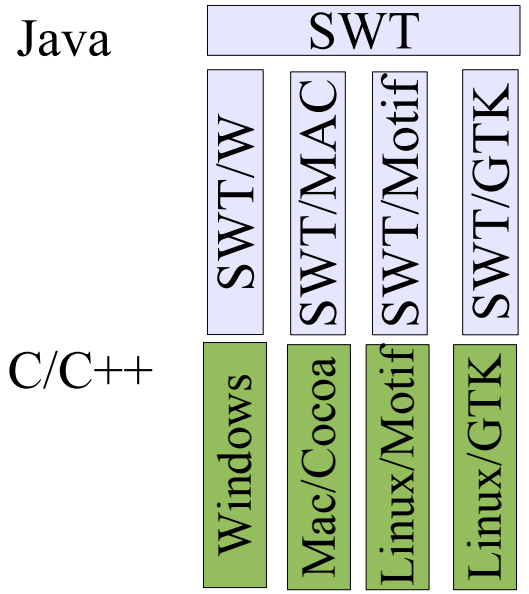
Swing

- Initialement un projet de Netscape donné à SUN (Java Fondation Class, ou JFC)
- Chaque composant est dessiné sur une fenêtre de la plateforme, même *look* quelque soit la plateforme
- Permet de personnaliser le look & feel d'une application
- Problème de vitesse d'affichage avant la version du JDK 1.3
- La version 1.5 utilise la carte 3D !



SWT (Standart Widget Toolkit)

- Créé par IBM pour l'IDE Eclipse
- Comme avec AWT, chaque composant a un Peer mais la gestion est effectuée en Java et pas en C++
- Pas intégré au JDK
- Défaut de jeunesse, API commence à être stable et bugs majeurs corrigés (dans la version 3.0)
- L'utilisateur retrouve le look de la plateforme au maximum



AWT, Swing et SWT (en résumé)

- AWT n'est quasiment plus utilisé
  (sauf pour des applications compatibles JDK 1.1)

- Swing est le plus utilisé actuellement

- SWT est utilisé :
  - pour l'intégration (plugin) avec éclipe
  - dans certaines applications écrites en Java
    et compilées en code machine
    pour une plateforme

Composant et hiérarchie de composants

- AWT, SWT, Swing utilise des classes différentes pour représenter chaque composant graphique.

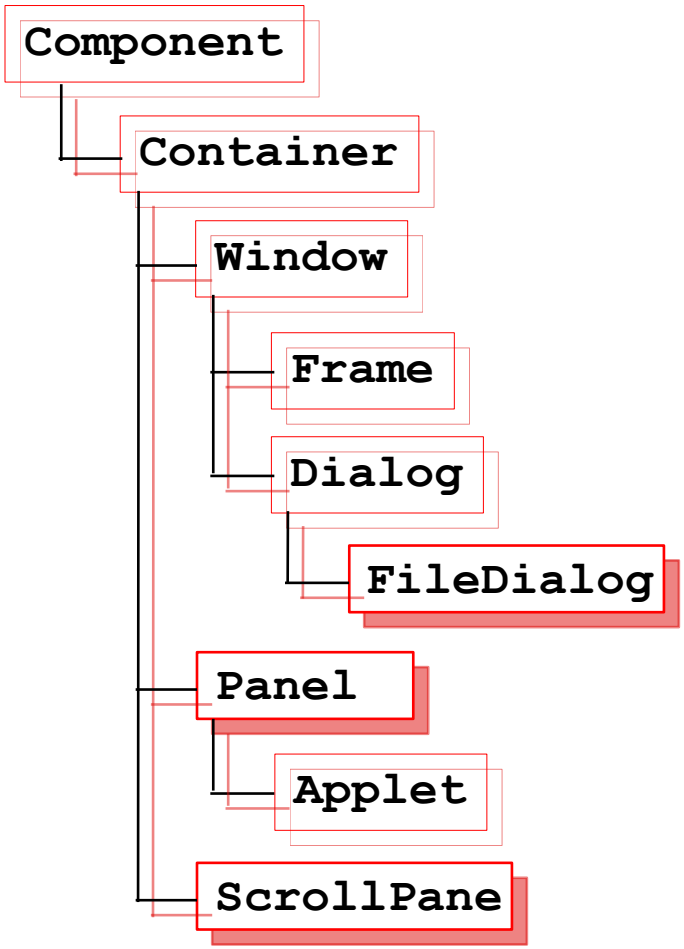
- Tous les composants ont une classe racine commune (cette classe n'est pas abstraite !!)

- La classe représentant les composants est :
  - pour l'AWT : **`Component`**
  - pour Swing : **`JComponent`**
  - pour SWT : **`Widget`**

- Notons que tous les noms des composants graphiques de Swing commence par J...

Composants de l'AWT

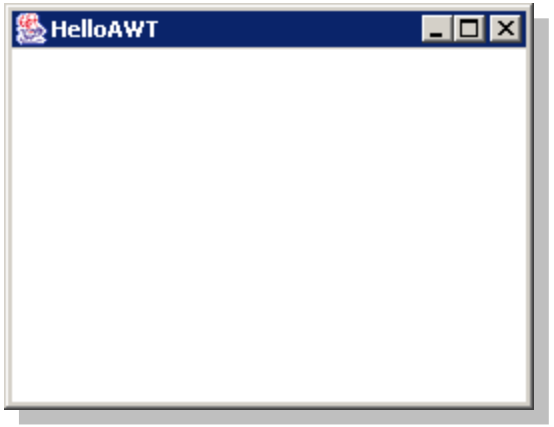
- **Container** composant contenant des composants (*design pattern Composite*)
- **Window** fenêtre sans bordure
- **Frame** fenêtre principale d'application
- **Panel** contient des composants, à l'intérieur d'une fenêtre
- **Applet** ouvre un emplacement dans un browser
- **ScrollPane** conteneur d'ascenseurs



HelloWorld avec AWT

- Le programme affiche une fenêtre ayant pour titre « HelloAWT ».

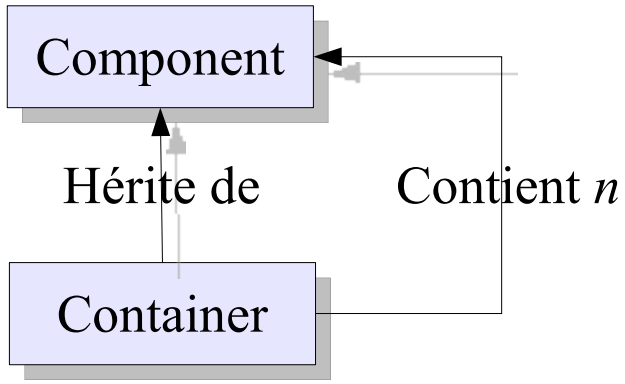
```
import java.awt.Frame;

public class HelloAWT {
  public static void main(String[] args) {
    Frame frame=new Frame();
    frame.setTitle("HelloAWT");
    frame.setSize(400,300);
    frame.setVisible(true);
  }
}
```



- Notons que s'il on clique sur la croix, la fenêtre devient non visible mais l'application continue

Composant et hiérarchie

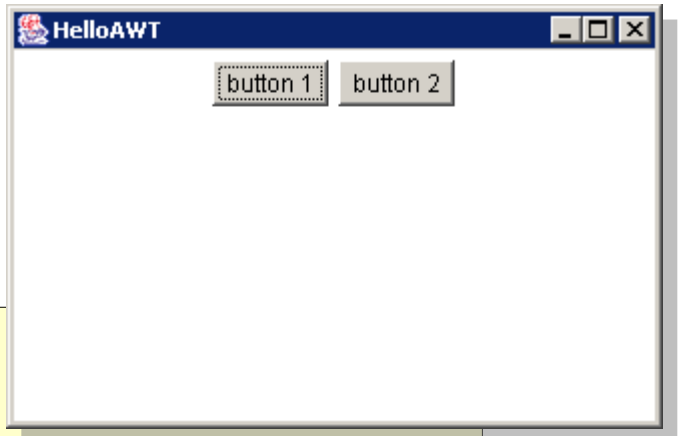
- Un composant possède un container père getParent()



- Un container est un composant qui possède des sous-composants (*design-pattern composite*)
  - **`getComponentCount()`** renvoie le nombre de fils.
  - **`getComponent(int index)`** renvoie un fils.

Hiérarchie de composants

- Les composants sont organisés sous la forme d'un arbre



- La méthode **`add()`** permet d'ajouter un composant à un container

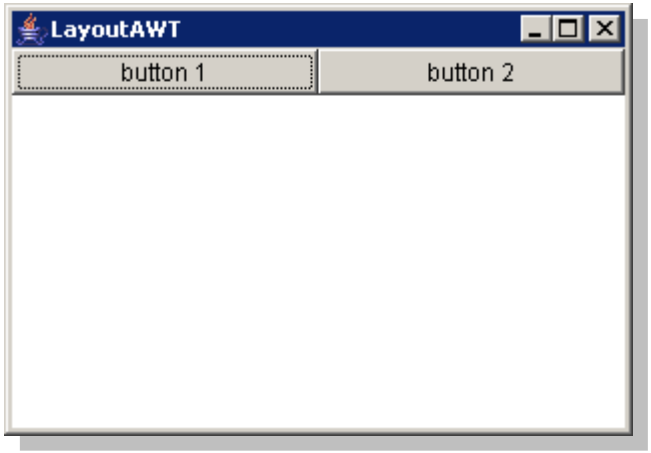
- **`Panel`** est un sous-type de **`Container`**

```
import java.awt.*;

public class Hierarchy {
  public static void main(String[] args) {
    Button button1=new Button("button 1");
    Button button2=new Button("button 2");

    Panel panel=new Panel();
    panel.add(button1);
    panel.add(button2);

    Frame frame=new Frame("HelloAWT");
    frame.add(panel);
    frame.setSize(400,300);
    frame.setVisible(true);
  }
}
```

Placement des composants

- En Java, le placement des composants est de la responsabilité du gestionnaire de géométrie et pas du container
- Permet de ne pas avoir des fenêtres figées !! donc pas de placement absolu (x,y)

```
...
import java.awt.LayoutManager;
import javax.swing.BoxLayout;

public class Hierarchy {
  public static void main(String[] args) {
    Button button1=new Button("button 1");
    Button button2=new Button("button 2");

    Panel panel=new Panel();
    LayoutManager layout=
      new BoxLayout(panel,BoxLayout.X_AXIS));
    panel.setLayout(layout);
    panel.add(button1);
    panel.add(button2);
    ...
```



Affichage et visibilité

- Chaque composant possède un état indiquant s'il est visible ou non (**`is/setVisible()`**)

- Un changement d'état de la visibilité se propage sur l'ensemble des composants fils du composant

- Les composants **`Frame`**, **`Dialog`**, **`Window`**, **`Applet`** possédent aussi la méthode **`setVisible()`**

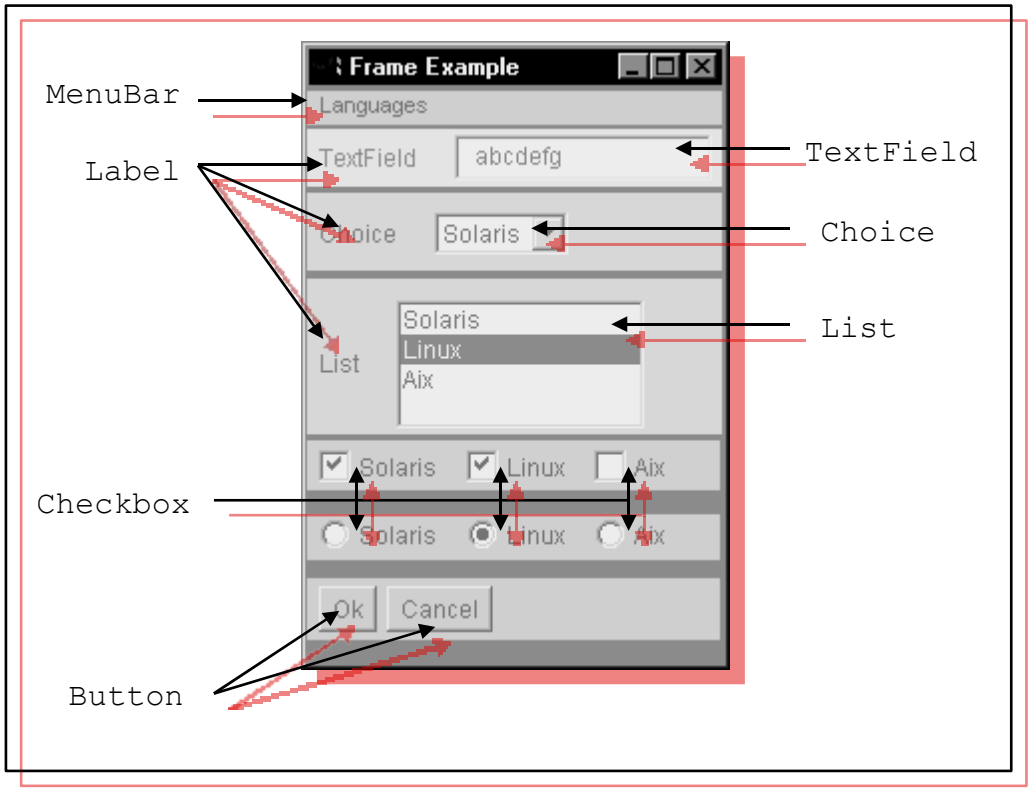
```
import java.awt.*;

public class Hierarchy {
  public static void main(String[] args) {
    Button button1=new Button("button 1");
    Button button2=new Button("button 2");

    Panel panel=new Panel();
    panel.add(button1);
    panel.add(button2);

    Frame frame=new Frame("HelloAWT");
    frame.add(panel);
    frame.setSize(400,300);
    frame.setVisible(true);
  }
}
```

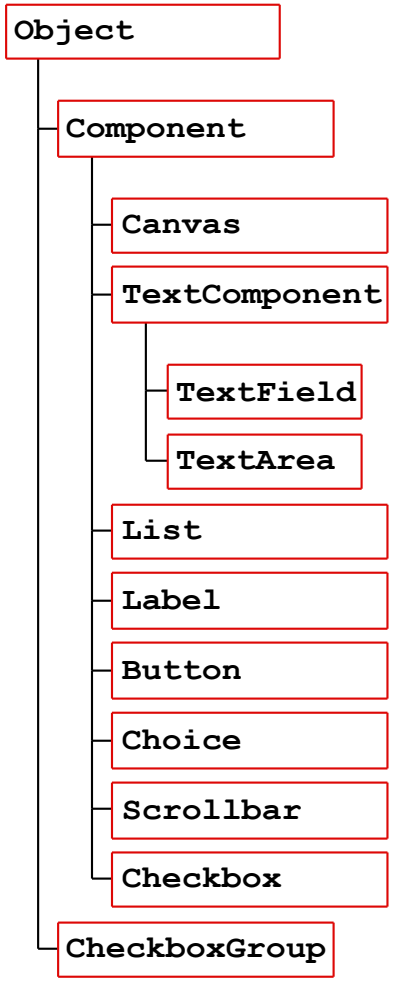
Autres composants de l'AWT

Canvas pour le dessin,

Choices est une sorte de combobox

CheckboxGroup composant logique





Relation entre Swing et AWT

- Swing utilise l'AWT pour ouvrir une fenêtre

- Il y a deux types de composants Swing

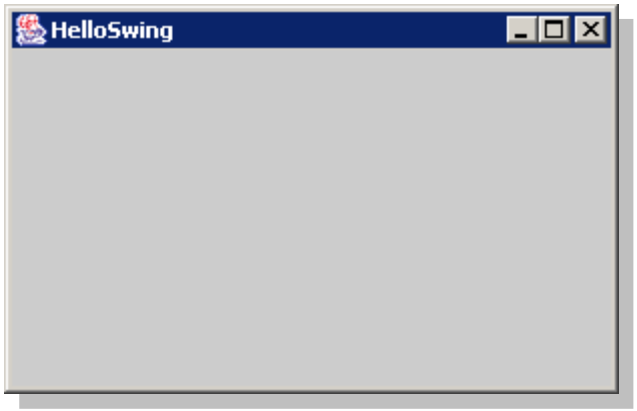
  - les heavyweight : gérés par l'AWT, correspondent à des fenêtres de la plateforme.
    `JFrame`, **`JDialog`**, **`JWindow`**, **`JApplet`**

  - les lightweight : composants Java qui effectuent le dessin

- Tous les composants AWT sont heavyweights

HelloWorld avec Swing

- Le programme affiche une fenêtre ayant pour titre « HelloSwing ».

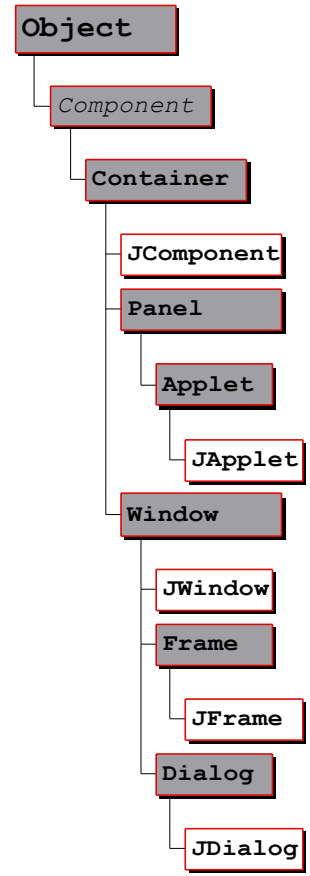


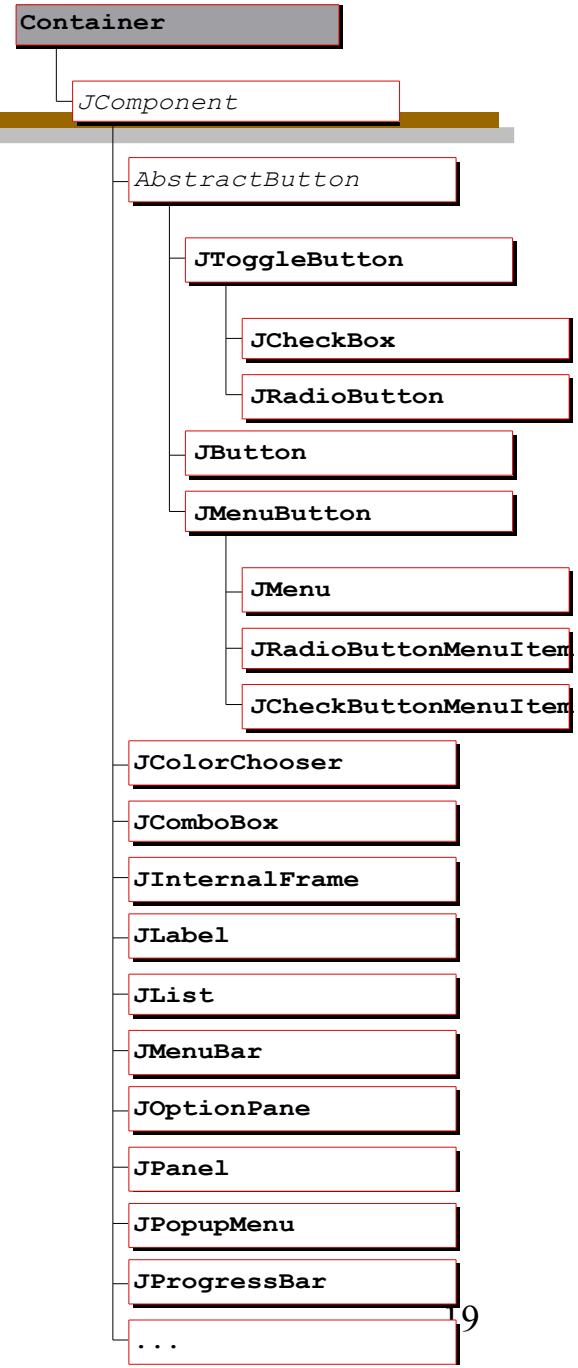

```
import javax.swing.JFrame;

public class HelloSwing {

  public static void main(String[] args) {
    JFrame frame=new JFrame();
    frame.setTitle("HelloSwing");
    frame.setSize(400,300);
    frame.setVisible(true);
  }
}
```

- Notons que s'il on clique sur la croix, l'application ne se ferme toujours pas.

Composants de Swing

- Les composants de Swing partagent une partie de leur implantation avec ceux de l'AWT
- Il y a beaucoup plus de composants Swing que de composants AWT

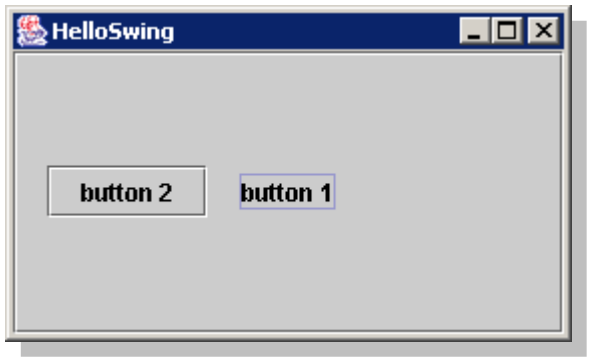




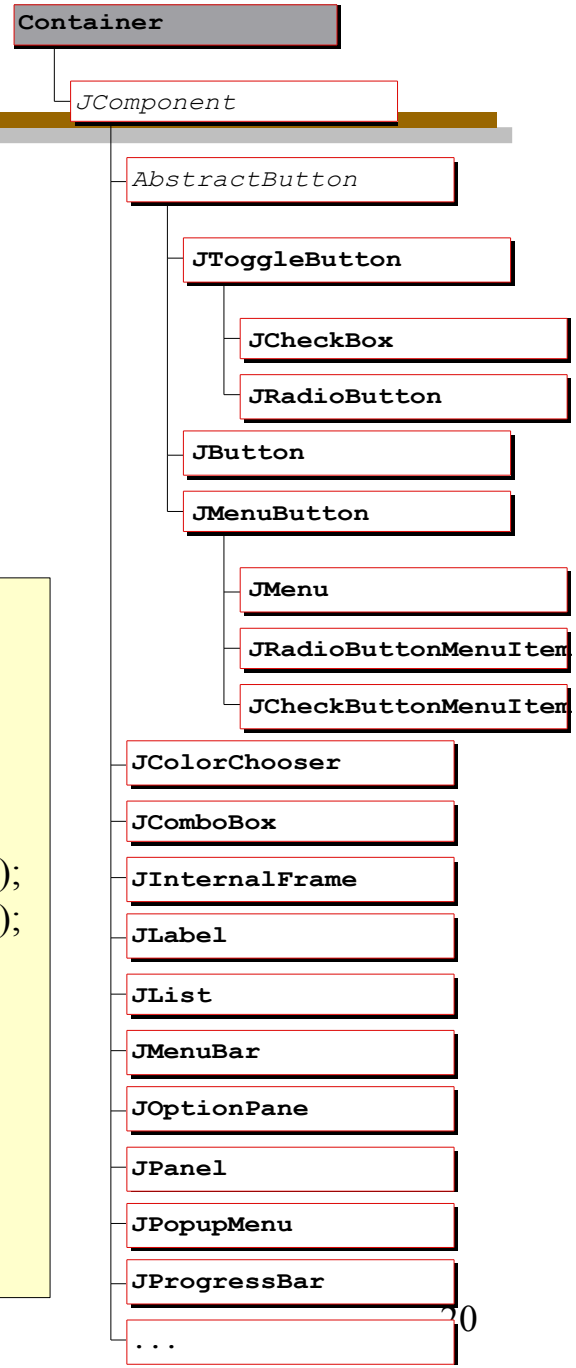
Composants de Swing (2)

- Problème de design
  n'importe quel JComponent
  est un Container

- Le code suivant est donc
  possible

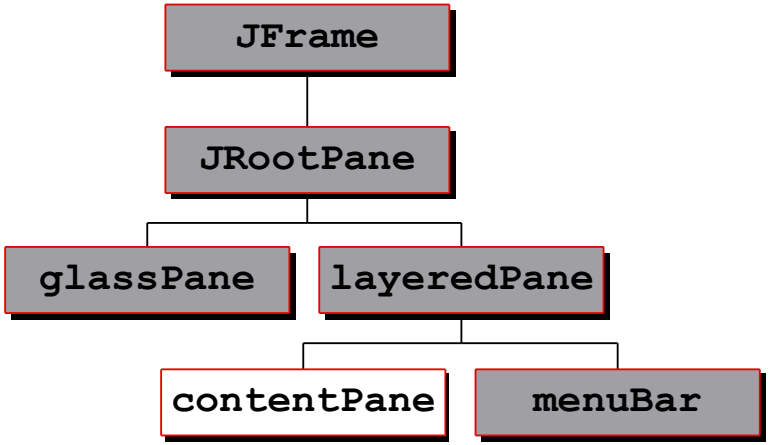



```
import javax.swing.*;

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

  JButton button1=new JButton("button 1");
  JButton button2=new JButton("button 2");
  button1.add(button2);

  frame.setContentPane(button1);
  frame.setTitle("HelloSwing");
  frame.setSize(400,300);
  frame.setVisible(true);
 }
}
```



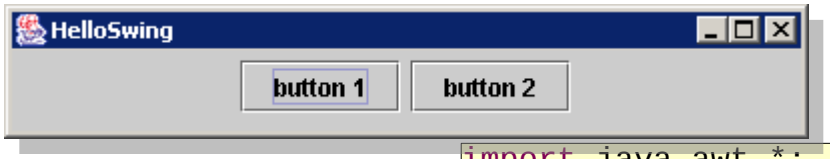
Le composant JFrame

- Une **JFrame** contient une *fille unique*, de la classe **JRootPane**
- Cette fille contient *deux fils*, glassPane (**JPanel**) et layeredPane (**JLayeredPane**)
- La layeredPane a *deux fils*, contentPane (un **Container**) et menuBar (un **JMenuBar**)
- On travaille dans contentPane.
- **JApplet**, **JWindow** et **JDialog** utilisent aussi un **JRootPane**.



Utilisations du contentPane

- Il est possible :
  - de demander le contentPane (**`getContentPane()`**)
  - de changer de contentPane (**`setContentPane()`**)




```
import javax.swing.*;

public class HierarchySwing2 {

  public static void main(String[] args) {
    JButton button1=new JButton("button 1");
    JButton button2=new JButton("button 2");

    JPanel panel=new Jpanel();
    panel.add(button1);
    panel.add(button2);

    JFrame frame=new JFrame("HelloSwing");
    frame.setContentPane(panel);
    frame.setSize(400,300);
    frame.show();
  }
}
```

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

public class HierarchySwing {

  public static void main(String[] args) {

    JFrame frame=new Jframe("HelloSwing");
    Container c=frame.getContentPane();
    c.setLayout(new JFlowLayout());

    JButton button1=new JButton("button 1");
    c.add(button1);
    JButton button2=new JButton("button 2");
    c.add(button2);

    frame.setSize(400,300);
    frame.show();
  }
}
```

JFrame, contentPane et add

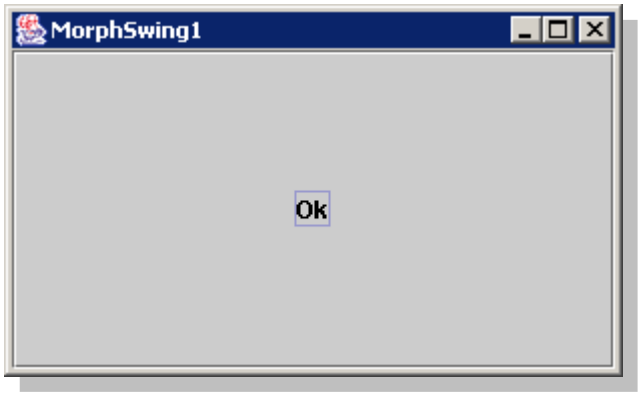
- Normalement, l'ajout de composants se fait sur le contentPane et non sur la JFrame
- Mais, si l'on effectue un add() sur une JFrame :
  - En 1.4 et avant, il y a une erreur à l'exécution
  - En 1.5, est equivalent à getContentPane().add()
- Ce mécanisme marche pour les méthodes : add/remove/setLayout

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

public class HierarchySwing {
  public static void main(String[] args) {

    JFrame frame=new Jframe("HelloSwing");
    frame.setLayout(new JFlowLayout());

    JButton button1=new JButton("button 1");
    frame.add(button1);
    JButton button2=new JButton("button 2");
    frame.add(button2);

    frame.setSize(400,300);
    frame.show();
  }
}
```

Interface graphique et prog. Objet

- Java est un langage Objet, il est donc possible d'utiliser l'héritage

- Doit-on hériter par exemple de la classe **`JFrame`** ?
  NON !




```
import javax.swing.*;

public class MorphSwing1 {
  public static void main(String[] args) {
    JButton button=new JButton("Ok");

    JFrame frame=new JFrame("MorphSwing1");
    frame.setContentPane(button);
    frame.setSize(400,300);
    frame.setVisible(true);
  }
}
```

```
import javax.swing.*;

public class MorphSwing2 extends JFrame {
  public MorphSwing2() {
    super("MorphSwing2");

    JButton button=new JButton("Ok");
    setContentPane(button);
    setSize(400,300);
    setVisible(true);
  }
  public static void main(String[] args) {
    new MorphSwing2();
  }
}
```

Interface graphique et prog. Objet (2)

- On hérite d'une classe si on veut en changer les fonctionnalités
  (i.e. redéfinir une méthode)

- Il est possible de faire des fonctions pour rendre le code plus clair

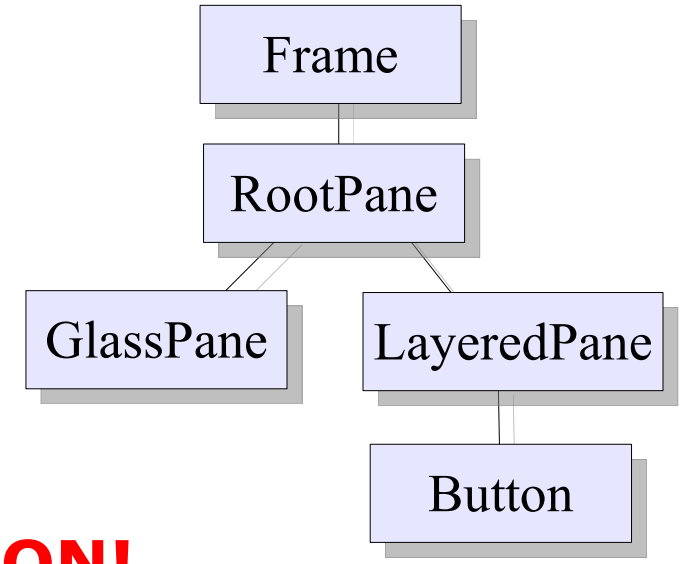
```
import javax.swing.*;

public class MorphSwing1 {
  public static void main(String[] args) {
    JButton button=new JButton("Ok");

    JFrame frame=new JFrame("MorphSwing1");
    frame.setContentPane(button);
    frame.setSize(400,300);
    frame.setVisible(true);
  }
}
```

```
import javax.swing.*;

public class MorphSwing2 extends JFrame {
  public MorphSwing2() {
    super("MorphSwing2");

    JButton button=new JButton("Ok");
    setContentPane(button);
    setSize(400,300);
    setVisible(true);
  }
  public static void main(String[] args) {
    new MorphSwing2();
  }
}
```

```
import javax.swing.*;

public class MorphSwing3 extends JButton {
  public MorphSwing3() {
    super("Ok");
  }
  JFrame create(String title) {
    JFrame frame=new JFrame(title);
    frame.setContentPane(this);
    frame.setSize(400,300);
    frame.setVisible(true);
    return frame;
  }
  public static void main(String[] args) {
    new MorphSwing3().create("MorphSwing3");
  }
}
```

Interface graphique et prog. Objet (3)

- Doit-on stocker les composants en tant qu'attributs de la classe ?



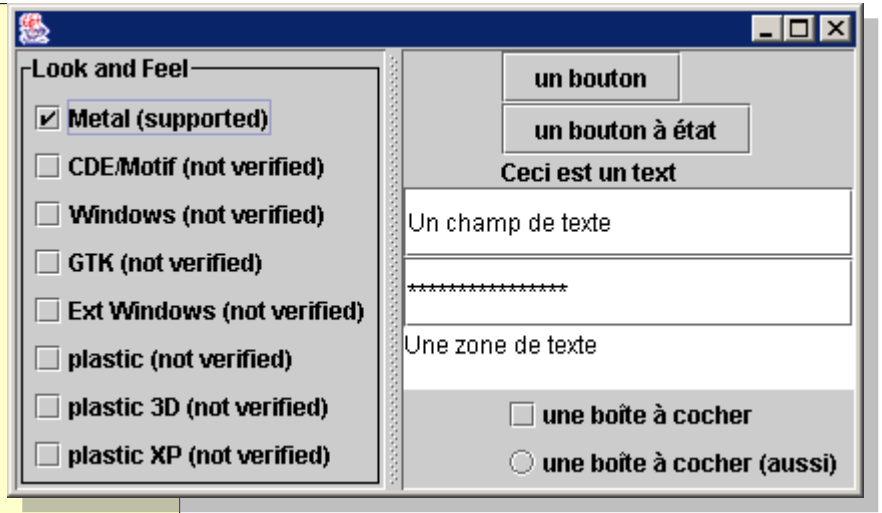

```
import javax.swing.*;

public class MorphSwing4 extends JFrame {
  private final JButton button; // idiot

  public MorphSwing4() {
    super("MorphSwing2");

    button=new JButton("Ok");
    setContentPane(button);
  }
  public static void main(String[] args) {
    JFrame frame=new MorphSwing4();
    frame.setSize(400,300);
    frame.setVisible(true);
  }
}
```

- **NON!**
  Les composants sont déjà stockés dans leurs parents.

Pluggable Look & Feel

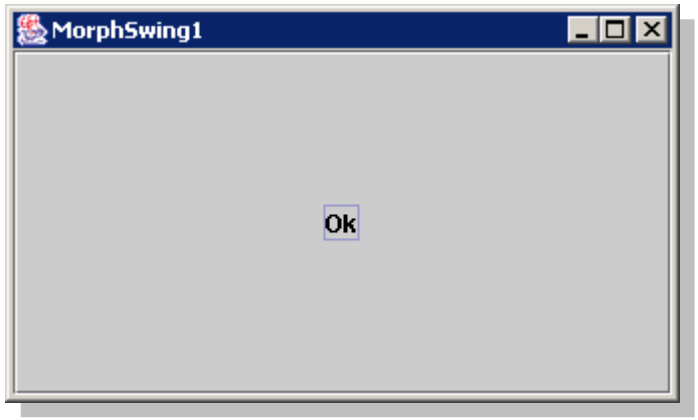
- Swing sépare les composants de leur rendu. Il est ainsi possible de changer le look d'un ensemble de composants
  - **Windows** `Com.sun.java.swing.plaf.windows.WindowsLookAndFeel`
  - **Motif** `com.sun.java.swing.plaf.motif.MotifLookAndFeel`
  - **Metal** `javax.swing.plaf.metal.MetalLookAndFeel`
  - **GTK** `com.sun.java.swing.plaf.gtk.GTKLookAndFeel`

```
try {
  UIManager.setLookAndFeel(className);
} catch (UnsupportedLookAndFeelException e) {
  ...
} catch (Exception e) {
  ...
}

SwingUtilities.updateComponentTreeUI(frame);
frame.pack();
```



Fermeture de fenêtre

- Par défaut, en cliquant sur la croix d'une fenêtre, l'application n'est pas arrêtée.



- **`setDefaultCloseOperation()`** permet de spécifier un comportement

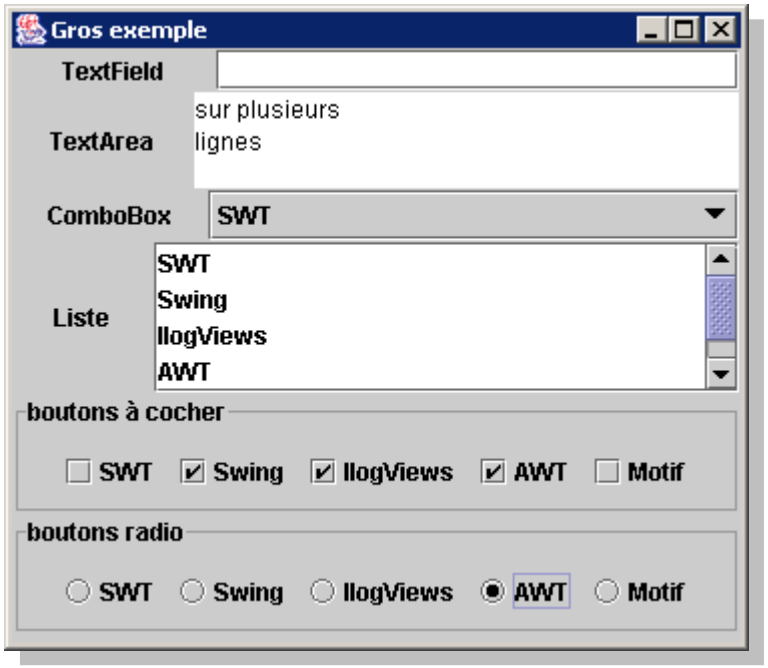
- Comportements :
  - DO_NOTHING_ON_CLOSE
  - HIDE_ON_CLOSE (défaut)
  - DISPOSE_ON_CLOSE
  - EXIT_ON_CLOSE

```
import javax.swing.*;

public class MorphSwing1 {
  public static void main(String[] args) {
    JButton button=new JButton("Ok");

    JFrame frame=new JFrame("MorphSwing1");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setContentPane(button);
    frame.setSize(400,300);
    frame.setVisible(true);
  }
}
```

Un exemple complet

- Voici un exemple utilisant les composants de base de swing

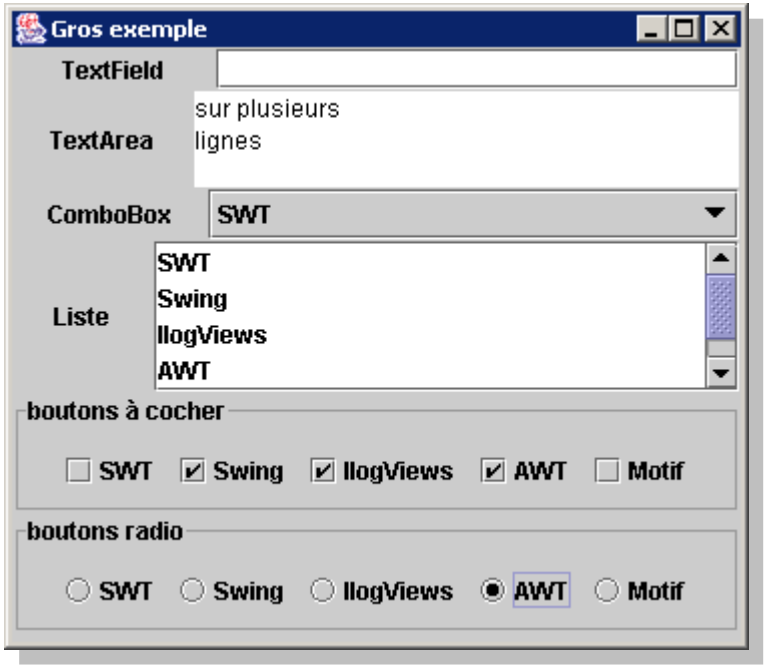
```
Import java.awt.*;
import javax.swing.*;

public class BigExample {
  private JPanel createMainPanel(Object... list) {
    ...
  }
  public static void main(String[] args) {
    JFrame frame=new JFrame("Gros exemple");
    frame.setDefaultCloseOperation(
      JFrame.EXIT_ON_CLOSE);

    BigExample example=new BigExample();
    JPanel panel=example.createMainPanel(
      "SWT","Swing","IlogViews","AWT","Motif"
    );
    frame.setContentPane(panel);
    frame.setSize(400,300);
    frame.setVisible(true);
  }
}
```



Etiquette, champs de texte, zone de texte

- JLabel, JTextField et JTextArea prennent un texte lors de la construction

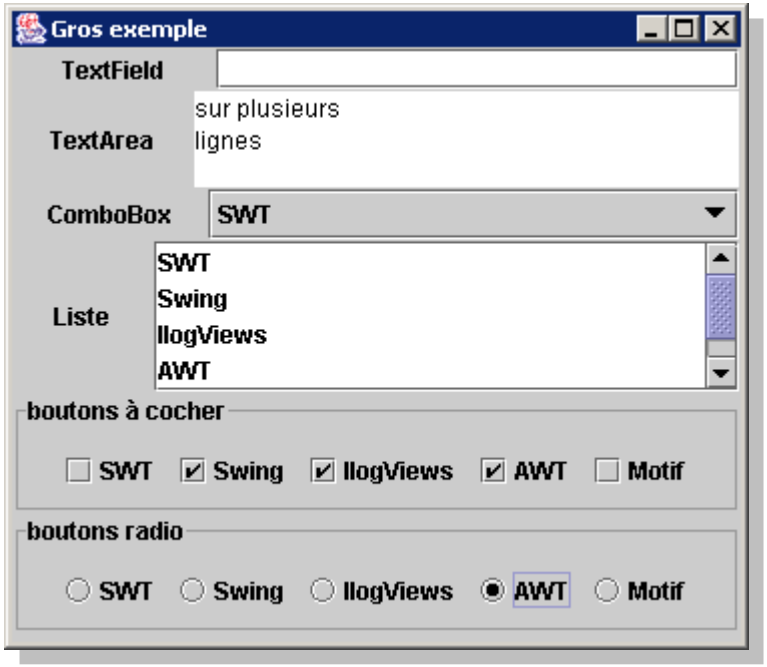
```
private JPanel createForm(String text,JComponent c) {
    JPanel panel=new JPanel(new GridBagLayout());
    GridBagConstraints constraints=new
GridBagConstraints();
    constraints.weightx=1.0;
    panel.add(new JLabel(text),constraints);
    constraints.weightx=5.0;
    constraints.fill=GridBagConstraints.HORIZONTAL;
    constraints.gridwidth=GridBagConstraints.REMAINDER;
    panel.add(c,constraints);
    return panel;
  }

  public JPanel createTextFieldPanel() {
    JTextField textField=new JTextField();
    return createForm("TextField",textField);
  }

  public JPanel createTextAreaPanel() {
    JTextArea textArea=new JTextArea(
      "sur plusieurs\nlignes\n");
    return createForm("TextArea",textArea);
  }
```



Liste et Liste déroulante

- JComboBox et JList peuvent prendre un tableau (Object[]) à la construction

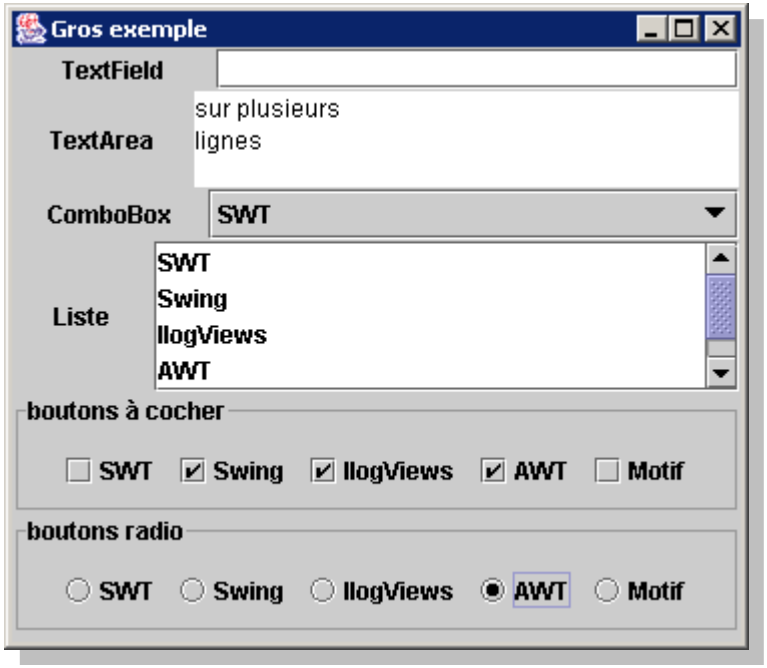
```
public JPanel createComboBoxPanel(Object... list) {
  JComboBox comboBox=new JComboBox(list);
  return createForm("ComboBox",comboBox);
}

public JPanel createListPanel(Object... array) {
  JList list=new JList(array);
  list.setVisibleRowCount(4);
  JScrollPane scrollPane=new JScrollPane(list);
  return createForm("Liste",scrollPane);
}
```



Boutons à cocher

- JCheckBox correspond à une case à cocher
- JRadioButton correspond à un bouton radio

```
private JPanel createCheckBoxes(JPanel panel,
                                String... list) {

    for(String s:list) {
      JCheckBox checkBox=new JCheckBox(s);
      panel.add(checkBox);
    }
    return panel;
  }
```

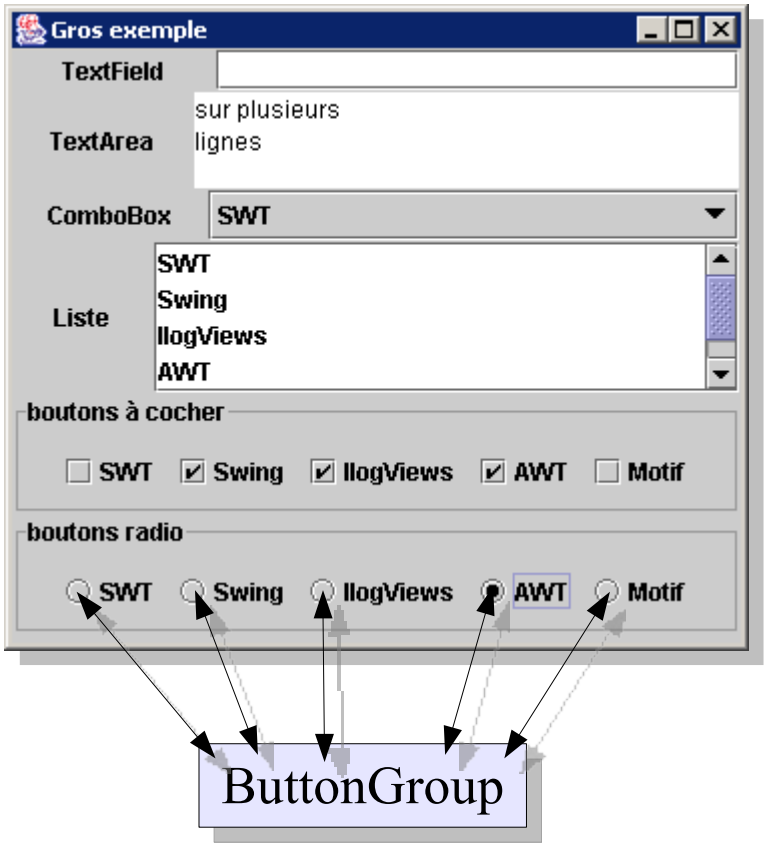


- JCheckBox, JRadioButton comme JButton héritent d'AbstractButton

Boutons radio et groupe de boutons

- Pour avoir un seul bouton séléectionné à la fois, le bouton doit faire partie d'un ButtonGroup

```
private JPanel createRadioButton(JPanel panel,
                                 String... list)
{
  ButtonGroup group=new ButtonGroup();
  for(String s:list) {
    JRadioButton radioButton=
      new JRadioButton(s);
    panel.add(radioButton);
    group.add(radioButton);
  }
  return panel;
}
```

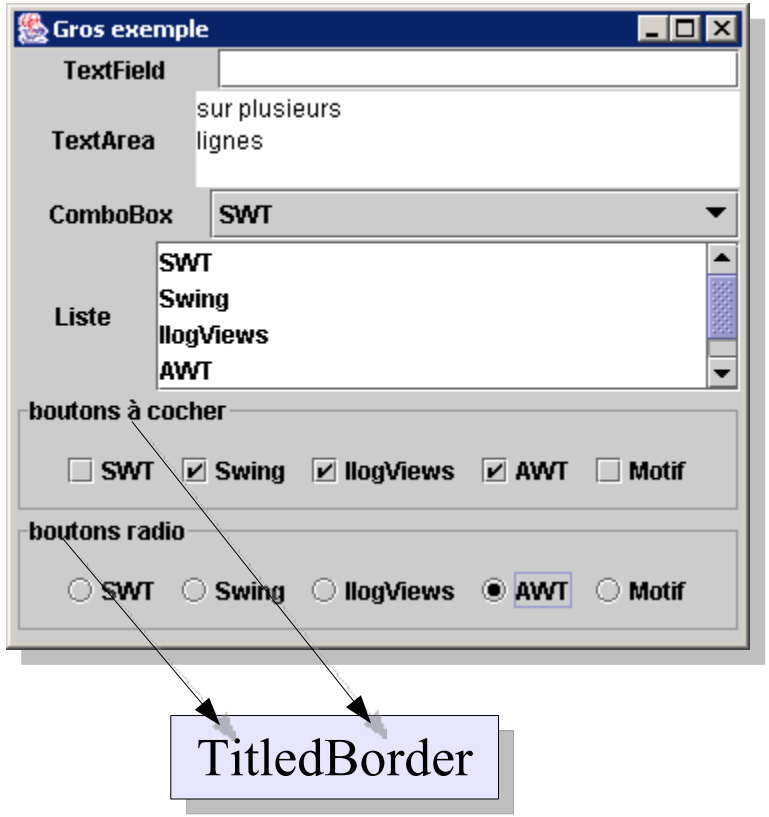


- ButtonGroup est un composant logique pas graphique

Bordure et Assemblage de l'interface

- N'importe quel composant swing peut avoir une bordure

```
private JPanel createMainPanel(Object... list) {
  return createBoxPanel(
    createTextFieldPanel(),
    createTextAreaPanel(),
    createComboBoxPanel(list),
    createListPanel(list),
    createCheckBoxes(
      createTitledPanel("boutons à cocher"),list),
    createRadioButton(
      createTitledPanel("boutons radio"),list)
    );
  }

private JPanel createTitledPanel(String title) {
  JPanel panel=new JPanel();
  Border border= new TitledBorder(title);
  panel.setBorder(border);
  return panel;
}
```




```
private JPanel createBoxPanel(JPanel... panels)
{
  JPanel panel=new JPanel(null);
  panel.setLayout(
    new BoxLayout(panel,BoxLayout.Y_AXIS));
  for(JPanel p:panels)
    panel.add(p);
  return panel;
}
```

La classe JComponent

- Un composant possède 4 tailles :
  - une taille actuelle (**`get/setSize()`**)
  - une taille préférée (**`get/setPreferredSize()`**)
  - une taille minimale (**`get/setMinimumSize()`**)
  - une taille maximale (**`get/setMaximumSize()`**)
- Un emplacement (**`get/setLocation()`**)
- Des états :
  - activé (**`is/setEnabled()`**)
  - visible (**`is/setVisible()`**)
- Une Bordure (**`get/setBorder()`**)
- Des Marges (**`get/setInsets()`**)
- Une Bulle d'aide (**`get/setToolTipText()`**)

Les Marges

- Un composant possède 4 marges représentées par la classe Insets.

- La classe Insets possède 4 champs (top,left,bottom,right) indiquant une taille en pixels.

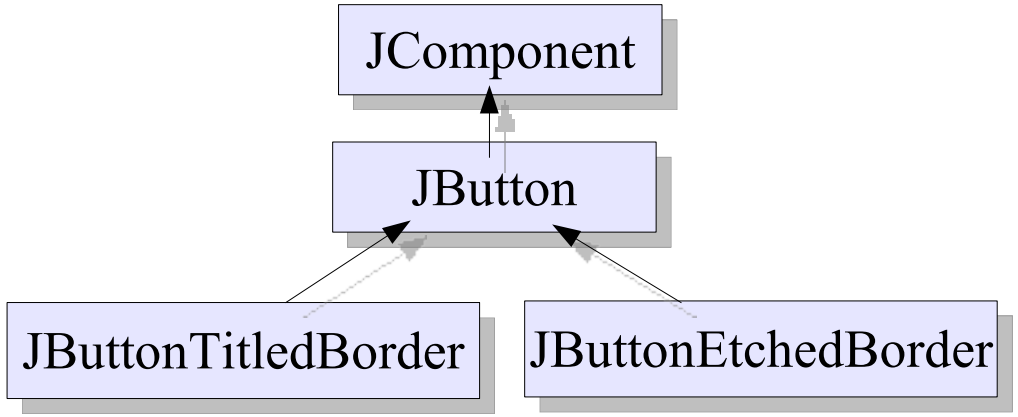
```
Insets(top,left,bottom,right)
```

- La méthode **`getInsets()`** (pas de **`setInsets()`**) permet d'obtenir les marges sur les composants.

- Les marges comprennent le bord, et pour **Frame/JFrame**, aussi la barre de titre !

Les Bordures

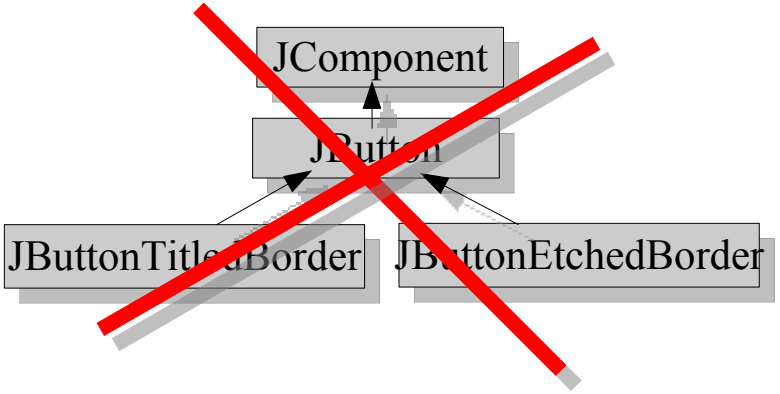
- Il existe différents type de bordures
  - bordure avec titre
  - bordure 3D (EtchedBorder)
  - bordure matérialisé par une ligne
  - bordure composée de plusieurs bordures
  - etc.
- Comment mixé les bordures et les composants ?



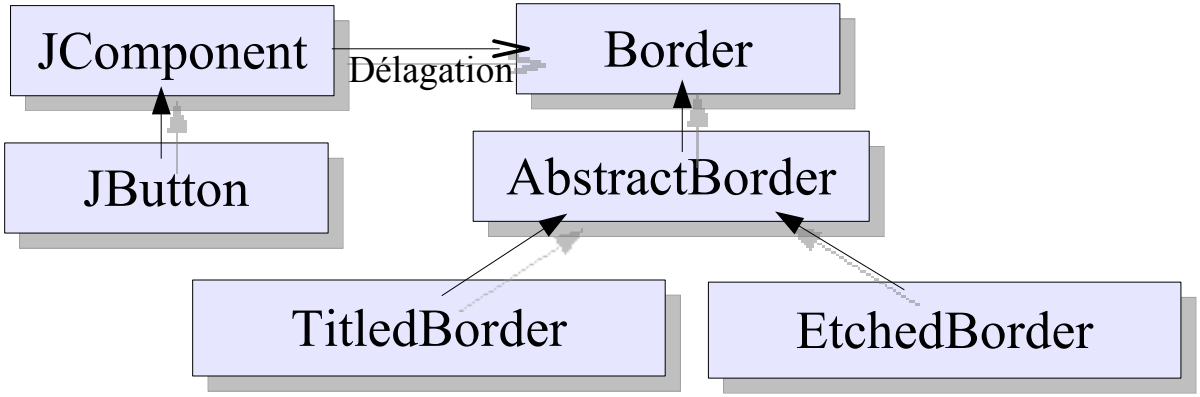
Les Bordures (2)

- Comment mixé les bordures et les composants ?

- Les Bordures utilisent le *design-pattern decorator*

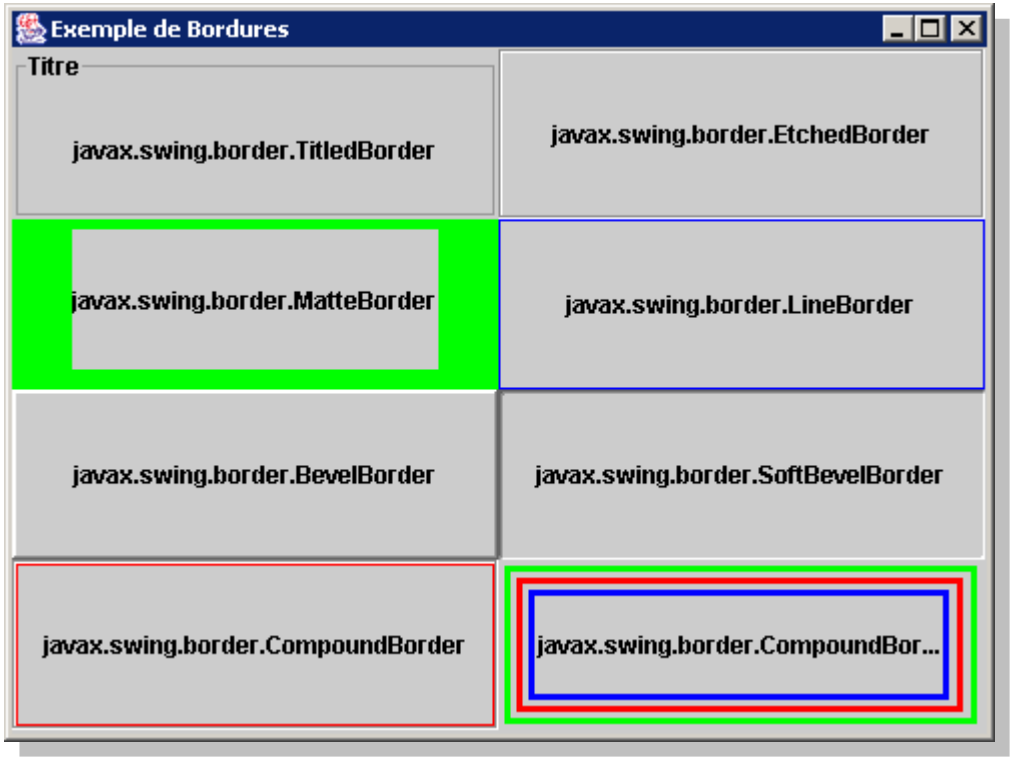



```
JPanel panel=new JPanel();
Border border= new
  TitledBorder(title);
panel.setBorder(border);
```



Les Bordures (3)

- Peut être associé à tout composant dérivant de **`JComponent`**
- Figurent dans **`javax.swing.border`**
- Tout bord étend **`AbstractBorder`** qui implémente Border
- Il est possible d'imbriquer des bordures grâce à la classe **`CompoundBorder`**



Les Bordures (4)

- La classe javax.swing.BorderFactory permet de créer l'ensemble des bordures existantes (cf *design-pattern factory)*

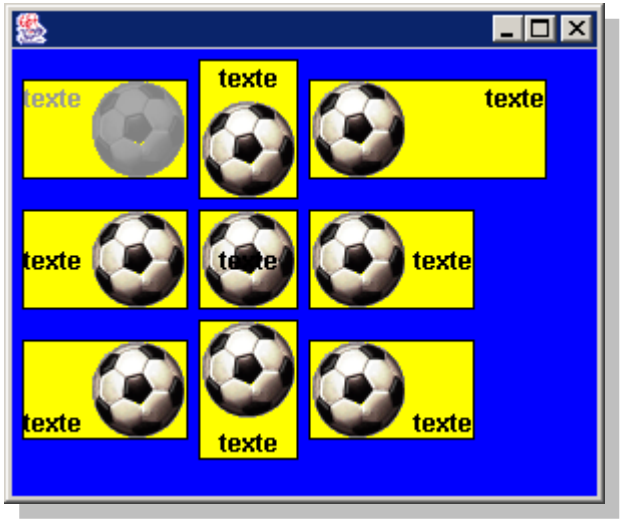
```
public class BorderFactory {
  public static Border createLineBorder(Color color) {
    return new LineBorder(color, 1);
  }

  public static Border createRaisedBevelBorder() {
    return sharedRaisedBevel;
  }

  static final Border sharedRaisedBevel=new
    BevelBorder(BevelBorder.RAISED);
  ...
 }
```

```
JPanel panel=new JPanel();
Border border=
  BorderFactory.createLioneBorder(Color.BLACK);
panel.setBorder(border);
```

Les Icônes

- La classe **ImageIcon** implémente l'interface **Icon**
- Le chargement de l'image de l'icône se fait au moyen d'un **MediaTracker**, mais ceci est transparent pour l'utilisateur.
- Constructeurs

```
ImageIcon(String nomfichier)
ImageIcon(Image image)
ImageIcon(URL url)
```



- Méthodes

```
int getIconHeight()
int getIconWidth()
Image getImage()
```

Les Etiquettes

- Une étiquette peut contenir un texte, une icône, ou les deux.
- Leurs positions dans le conteneur sont fixées par des constantes.
- Constructeurs

```
JLabel(String libellé, Icon image, int alignHorizontal)
```

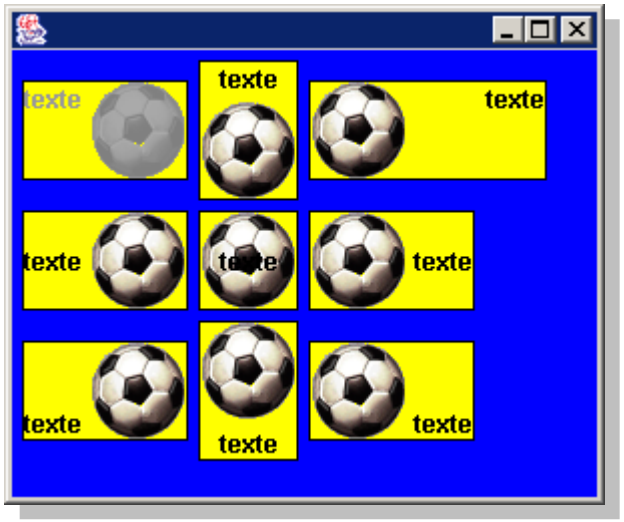
  - les paramètres peuvent être omis

- Méthodes

```
setIcon(Icon icon)
setText(String text)
setFont(Font font)
setVertical/HorizontalTextPosition(int alignement)
```

Les Etiquettes (2)

- Un exemple complet

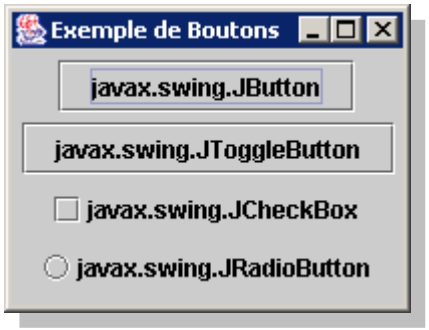



```
Import java.awt.*;
import javax.swing.*;
import javax.swing.border.Border;

public class ImageLabel {
  public static void main(String[] args) {
    Icon icon = new ImageIcon("soccer.gif");
    Border border=
      BorderFactory.createLineBorder(Color.BLACK);
    JLabel[] labels=new JLabel[] {
      createLabel(JLabel.TOP, JLabel.LEFT,icon,border),
      createLabel(JLabel.TOP, JLabel.CENTER,icon,border),
      createLabel(JLabel.TOP, JLabel.RIGHT,icon,border),
      createLabel(JLabel.CENTER, JLabel.LEFT,icon,border),
      createLabel(JLabel.CENTER, JLabel.CENTER,icon,border),
      createLabel(JLabel.CENTER, JLabel.RIGHT,icon,border),
      createLabel(JLabel.BOTTOM, JLabel.LEFT,icon,border),
      createLabel(JLabel.BOTTOM, JLabel.CENTER,icon,border),
      createLabel(JLabel.BOTTOM, JLabel.RIGHT,icon,border)
    };
    labels[0].setEnabled(false);
    labels[2].setIconTextGap(40);
    JPanel panel=new Jpanel(
      new FlowLayout(FlowLayout.LEFT, 5, 5));
    panel.setBackground(Color.BLUE);
    for (JLabel label:labels)
      panel.add(label);
    JFrame frame = new JFrame();
    frame.setContentPane(panel);
    frame.setDefaultCloseOperation(
      JFrame.EXIT_ON_CLOSE);
    frame.setSize(300,250);
    frame.setVisible(true);
  }
}
```

```
private static JLabel createLabel(int vertical, int horizontal,
  Icon icon,Border border) {
  JLabel label = new JLabel("texte", icon, SwingConstants.CENTER);
  label.setVerticalTextPosition(vertical);
  label.setHorizontalTextPosition(horizontal);
  label.setBorder(border);
  label.setBackground(Color.YELLOW);
  label.setOpaque(true);
  return label;
}
```

Les Boutons

- Un bouton comme une étiquette peut contenir un texte, une icône, ou les deux.
- Ils existent plusieurs classes de boutons qui héritent tous de AbstractButton
  - JButton
  - JMenuItem
  - JToggleButton
    - JCheckBox
    - JradioButton



- Constructeurs

```
JButton(String libellé, Icon image)
```

  - les paramètres peuvent être omis

Les zones et champs de texte

- Une zone de texte (**JTextArea**) et un champ de texte (**JTextField**) héritent de la classe **TextComponent**.

- Constructeurs

```
JTextField(String text, int columns)
JTextArea(String text, int rows, int columns)
```

  - Les paramètres peuvent être omis

- Méthodes

```
setEditable(boolean editable)
setText(String text)
setFont(Font font)
```