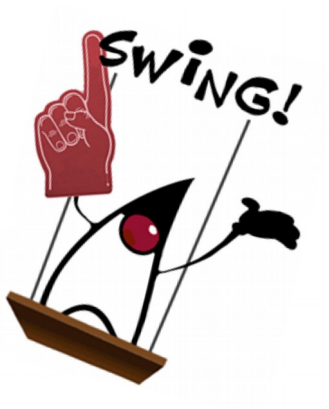
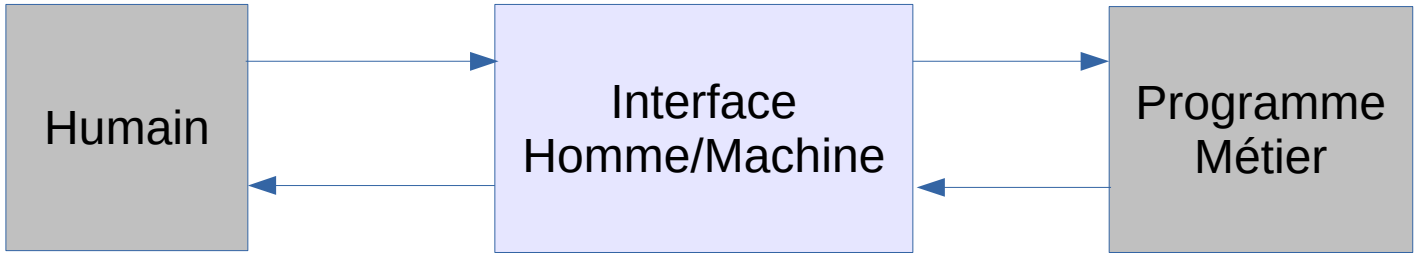
Introduction à



Les Interface graphique en Java

- Interface graphique ou Interface Homme Machine



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

AWT (***A**bstract **W**indowing **T**oolkit*)

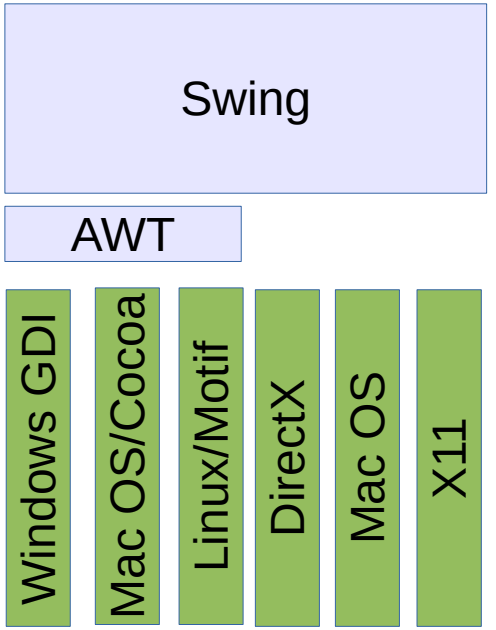
- **AWT**, chaque composant Java possède un « ***peer*** » C++ correspondant au composant de la plateforme

- 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 (**J**ava **F**ondation **C**lass – 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

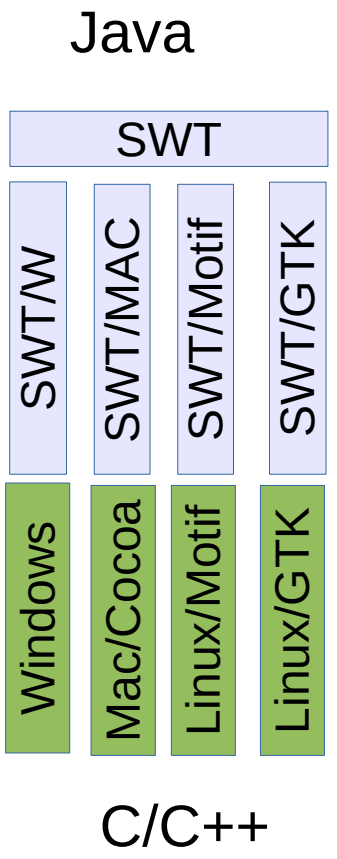


- Problème de vitesse d'affichage avant la version du JDK 1.3
- La version 1.5 utilise la carte 3D !

SWT (**S**tandart **W**idget **T**oolkit)

- 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 (*plugins*) avec Eclipe
  - dans certaines applications écrites en Java et compilées en code machine pour une plateforme (azureus)

Relation entre Swing et AWT

- **Swing** utilise **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`**

  - les ***lightweight*** : composants Java qui effectuent le dessin

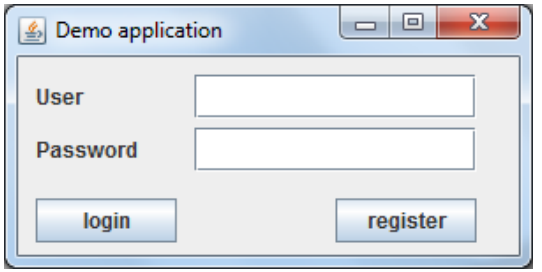
- Tous les composants AWT sont ***heavyweights***

Structure d'un interface SWING

- Création de l'interface graphique passe forcément par une instance de la classe ***Jframe,*** du point de vue du système d'exploitation cette fenêtre représente l'application.

- **La fenêtre est un « conteneur » dans lequel sont disposés les éléments constitutifs (composants) de l'interface graphique (boutons, listes déroulantes, zone de saisie…)**

Les applications graphiques sont comme des poupées russes : on ajoute des composants dans des composants.



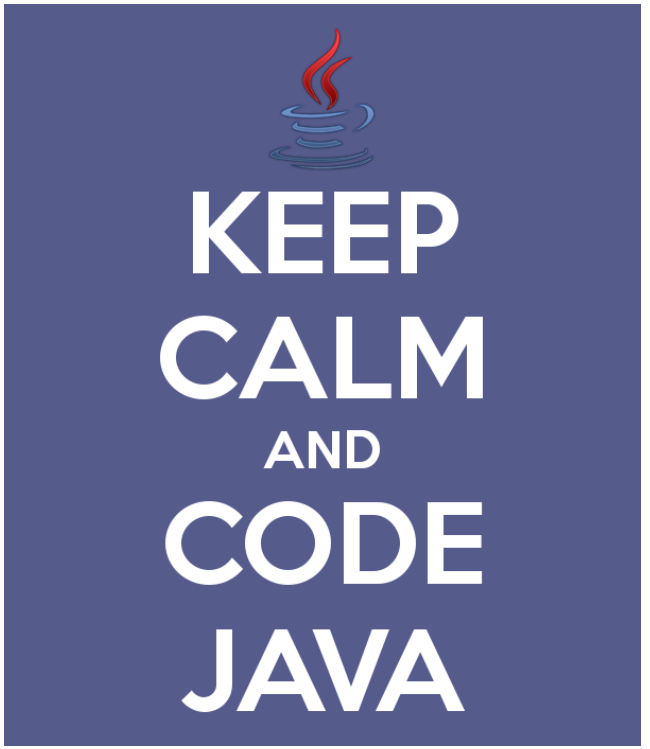
HelloWolrd version swing

```
import javax.swing.*;
public class FirstSwingExample {
public static void main(String[] args) {
JFrame f=new JFrame();//creating instance of JFrame

JButton b=new JButton("click");//creating instance of JButton
b.setBounds(130,100,100, 40);//x axis, y axis, width, height

f.add(b);//adding button in JFrame

f.setSize(400,500);//400 width and 500 height
f.setLayout(null);//using no layout managers
f.setVisible(true);//making the frame visible
}
}
```

Création de la fenêtre

Création d'un bouton

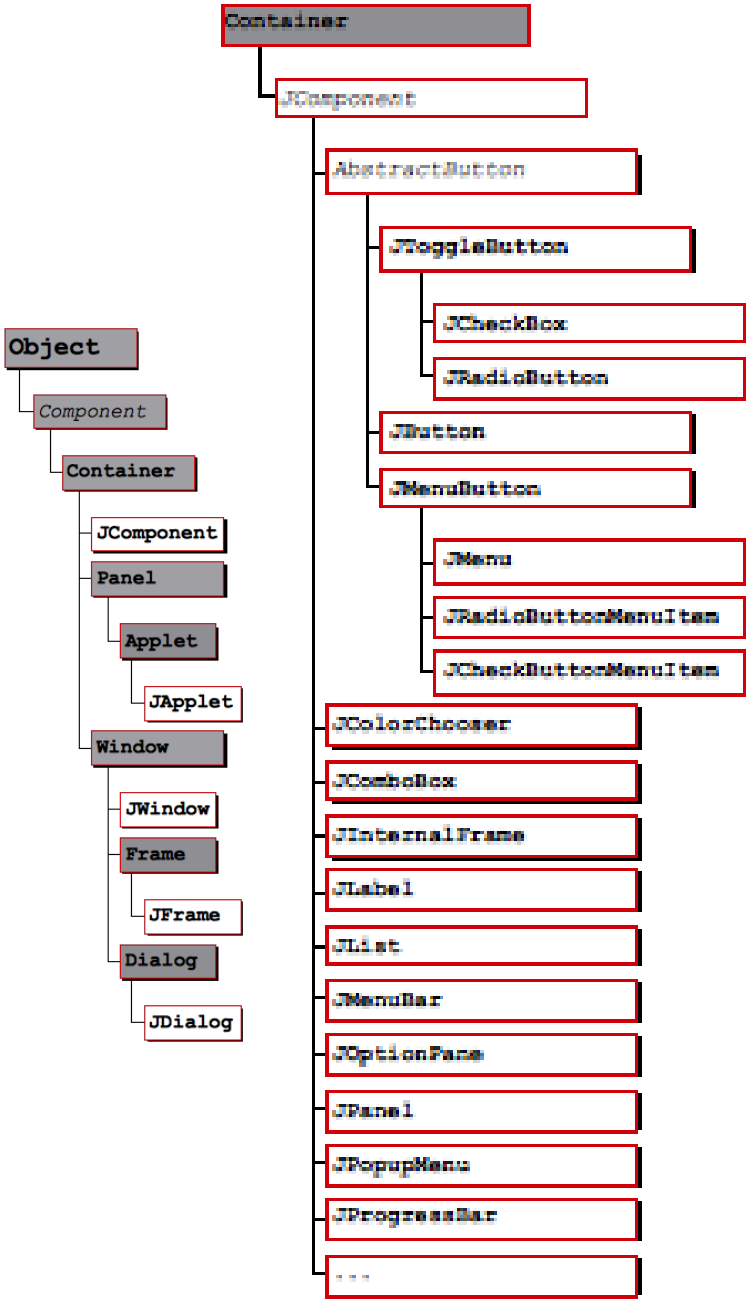
Ajout du bouton dans la fenêtre



Composants de Swing

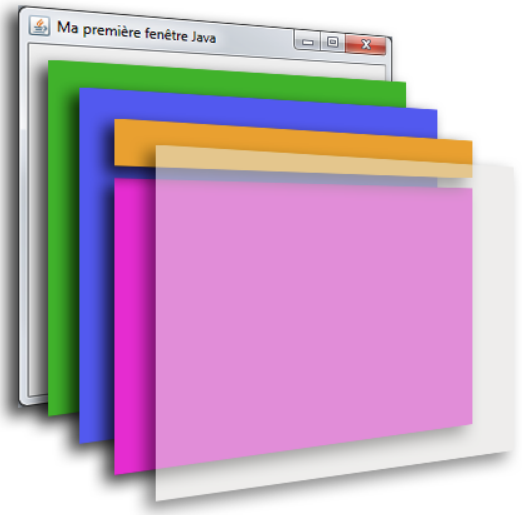
- Les composants de Swing partagent une partie de leur implantation avec ceux de l'AWT

→ Il y a beaucoup plus de composants Swing que de composants AWT



Le composant *JFrame*

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



Le *JRootPane*

- **Le *glasspane* :**
  composant transparent héritant de JComponent placé devant tous les composants du *contentpane*, il permet d'intercepter les évènements souris avant que ceux-ci soit distribués
- **Le *layeredpane* :**
  container sachant gérer la notion de *profondeur*
- **Le *menubar* :**
  barre de menu de l'application dont l'ensemble des menus sera affiché devant le *contentPane*
- **Le *contentPane* :**
  zone permettant d'ajouter de nouveaux composants

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();
  }
}
```

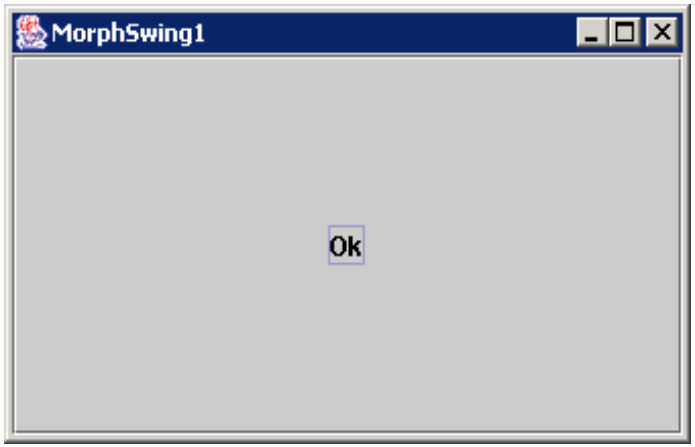
JFrame, contentPane et add

- 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 équivalent à *getContentPane*().*add*()
- Marche également pour les méthodes : **{add,remove,set}Layout**

```
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();
  }
}
```

Fermeture de fenêtre : comportements

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



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

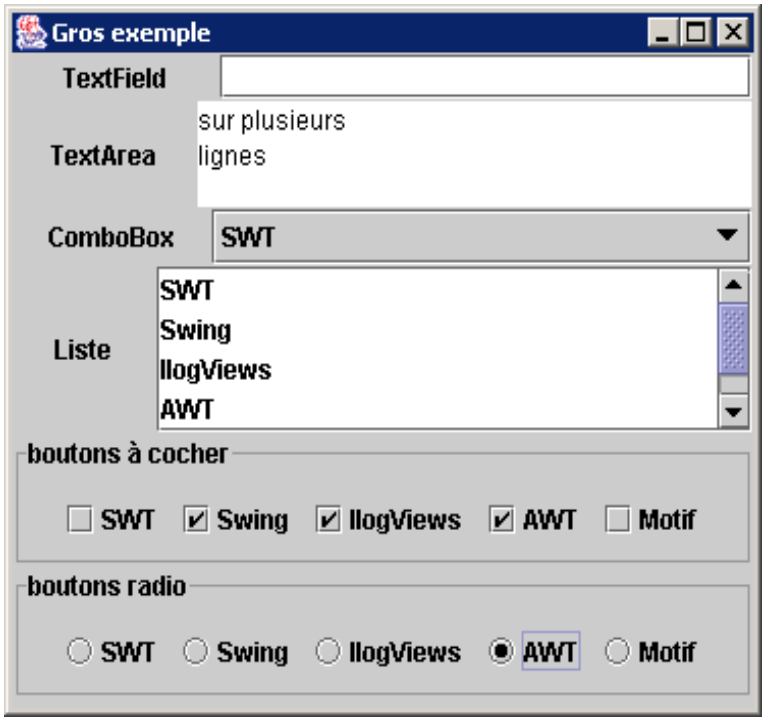
- **Comportements** :
  - DO_NOTHING_ON_CLOSE
  - HIDE_ON_CLOSE (défaut)
  - DISPOSE_ON_CLOSE
  - EXIT_ON_CLOSE

```
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 concret et 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);
  }
}
```



Le placement des composants

- Il est possible d'associer à un « conteneur », un ***LayoutManager*** chargé de disposer les Composants dans le plan.

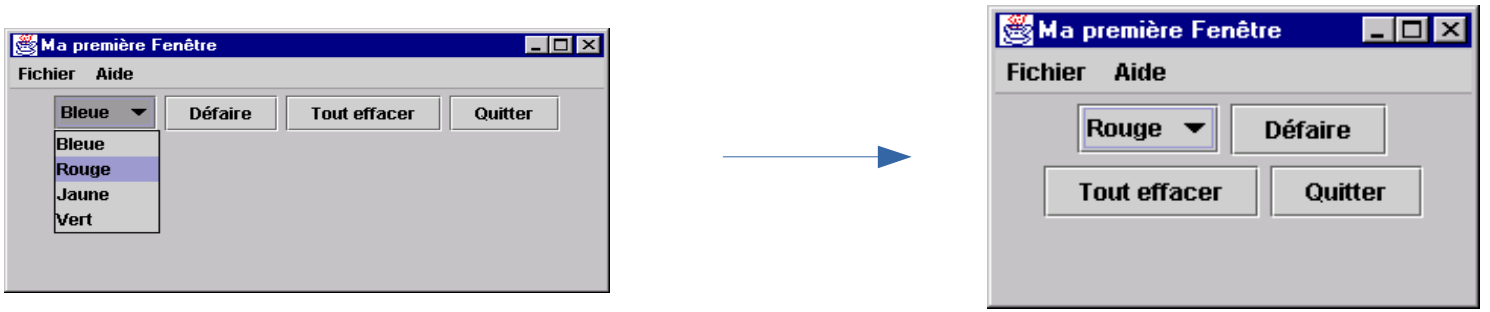
→ ***LayoutManager*** **est un objet associé à un Container :**

- Se charge de gérer la disposition des composant appartenant à celui-ci...

Par exemple le *Layout* par défaut des ***JPanels*** :

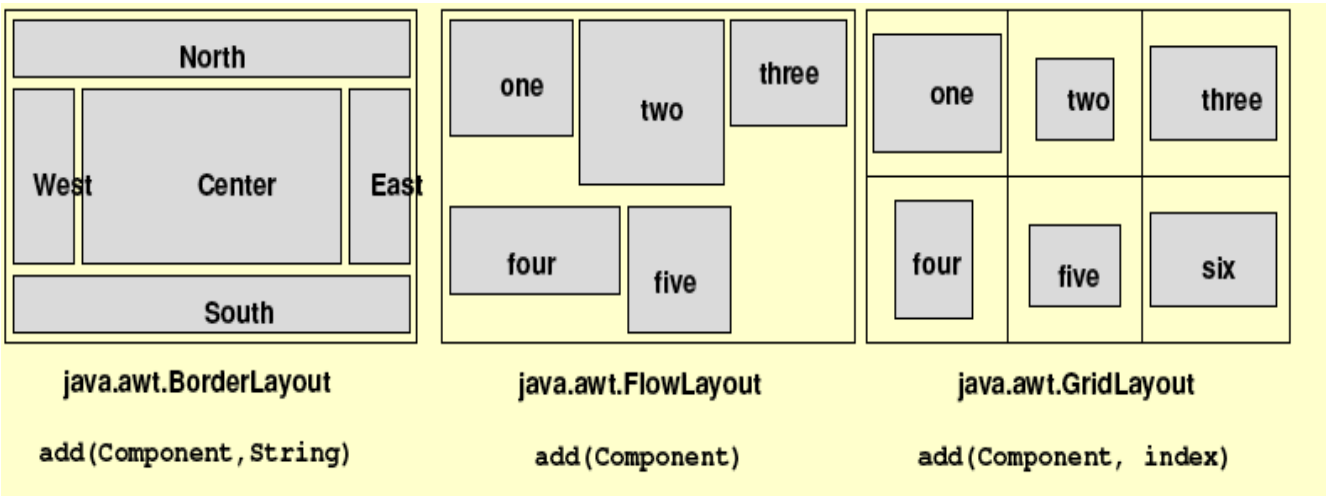
- composants placés les uns après les autres dans leur ordre d'ajout
- le tout doit être centré

- Ordonnancement automatique des composants lorsque la fenêtre est redimensionnée



Les *LayoutManager*

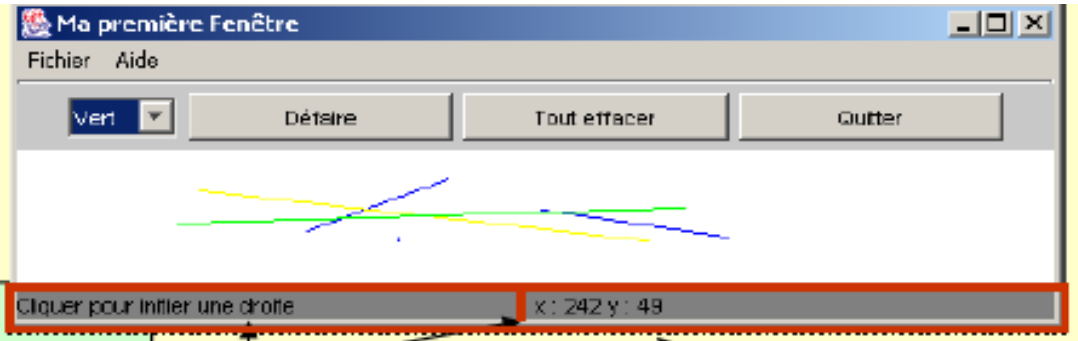
- 5 gestionnaires de mise en forme implémentant l ’interface ***LayoutManager*** sont prédéfinis dans ***AWT*** :
  - BorderLayout
  - FlowLayout
  - GridLayout
  - CardLayout
  - GridBagLayout



- D’autres sont définis dans les swings (BoxLayout, SpringLayout….)
- Il est aussi possible de définir ses propres gestionnaires...

Le GridLayout

- Définition de la barre d'état de l'application




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

public class BarreEtat extends JPanel {
   private JLabel coord, info;

   public BarreEtat() {
      this.setBackground(Color.darkGray);
      this.setLayout(new GridLayout(1,2));
      this.add(info = new JLabel());
      this.add(coord = new JLabel());
    }

    public void afficheCoord(int x, int y)
    {
       coord.setText("x : " + x + " y : " + y);
    }

    public void afficheInfo(String message)
    {
      info.setText(message);
    }
}
```

JLabel

JPanel

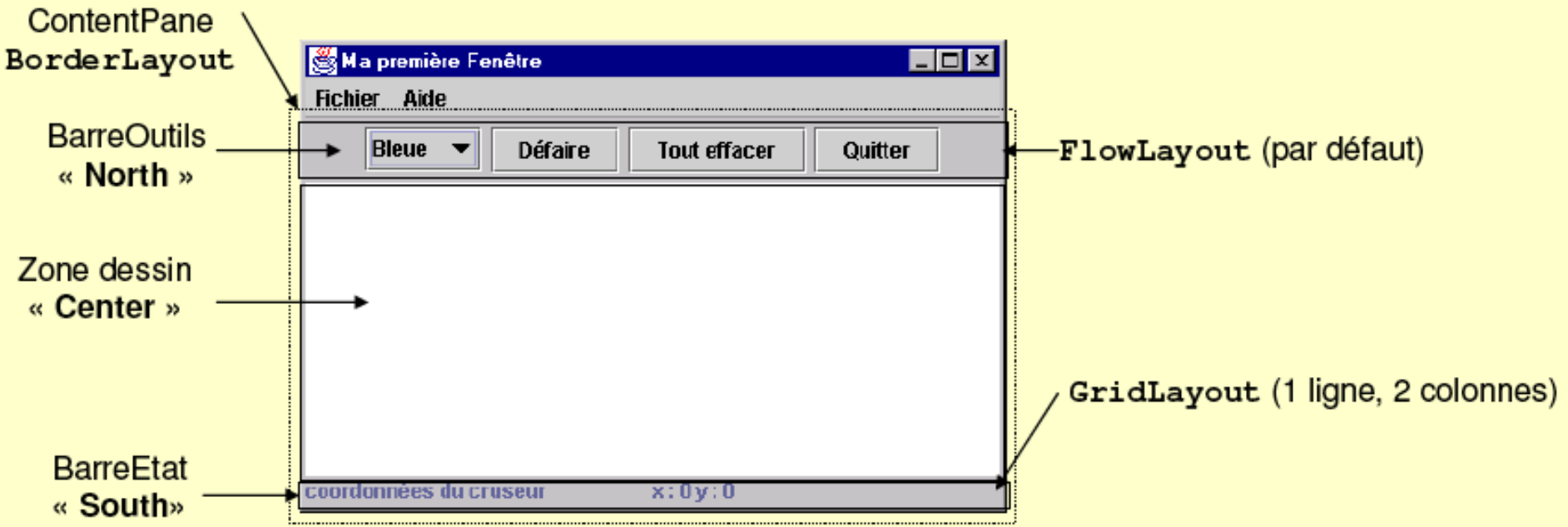
GrigLayout : 1 ligne, 2 colonnes

Associe un GridLayout en spécifiant le nombre de lignes et colonnes

Ajout des composants dans les différentes cellules définies par le layout

Le ***BorderLayout***

- Ajout d'une zone de dessin et d'une barre d'état à la fenêtre de l'application



Associe un BorderLayout en spécifiant espacement entre les composants

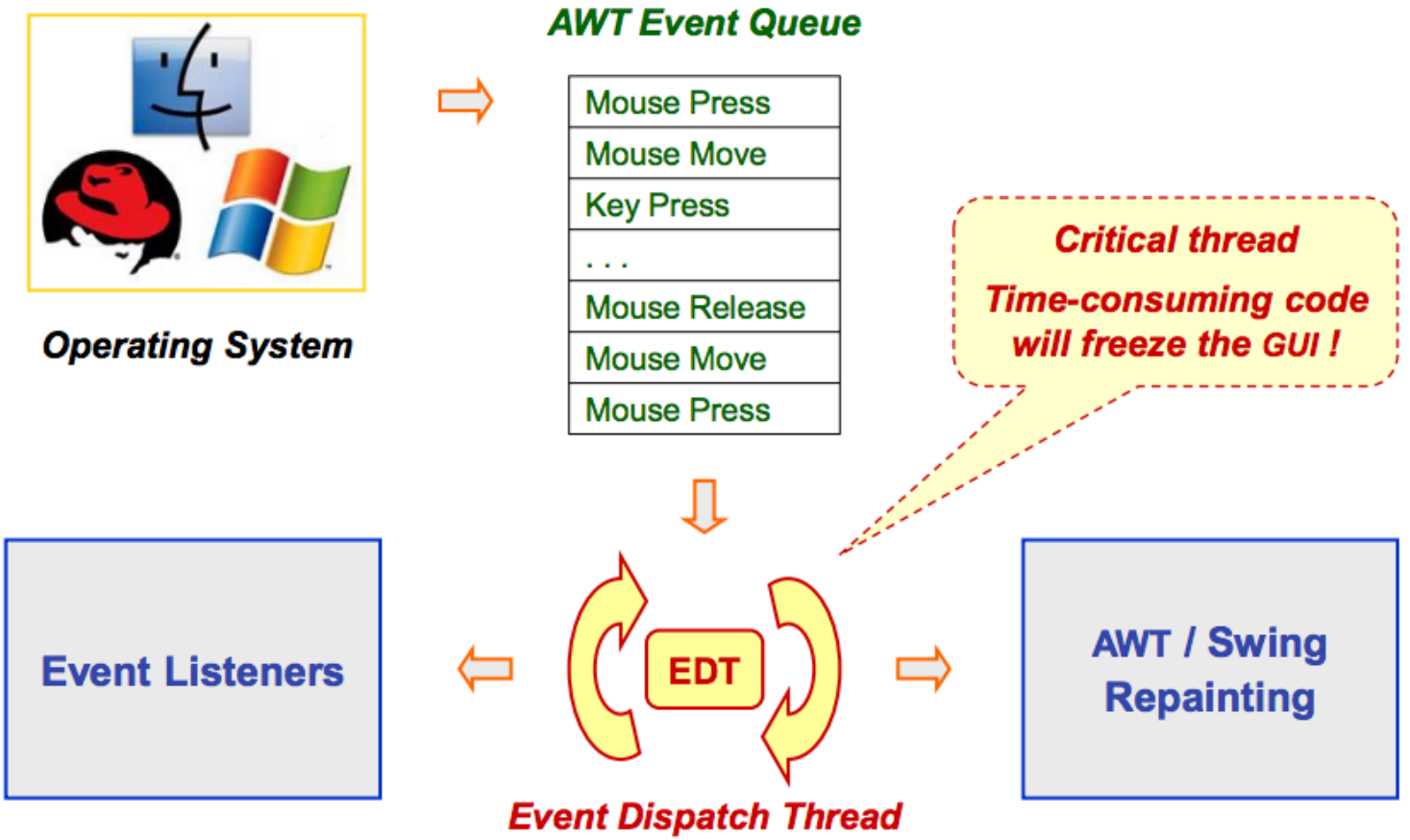
Ajout des composants dans les différentes régions définies par le layout

```
public MyFrame () {
    BarreEtat barreEtat;
    setTitle("Ma première Fenêtre");
    setSize(LARGEUR,HAUTEUR);
    setJMenuBar(new MenuEditeur());
    this.getContentPane().setLayout(new BorderLayout(2,2));
    this.getContentPane().add(new BarreOutils(),"North");
    this.getContentPane().add(new ZoneGraphique(),"Center");
    this.getContentPane().add(barreEtat = new BarreEtat(),"South");
    barreEtat.afficheCoord(0,0);
    barreEtat.afficheInfo("coordonnées du cruseur");

    setVisible(true);
}
```

L’**E**vent **D**ispatch **T**hread

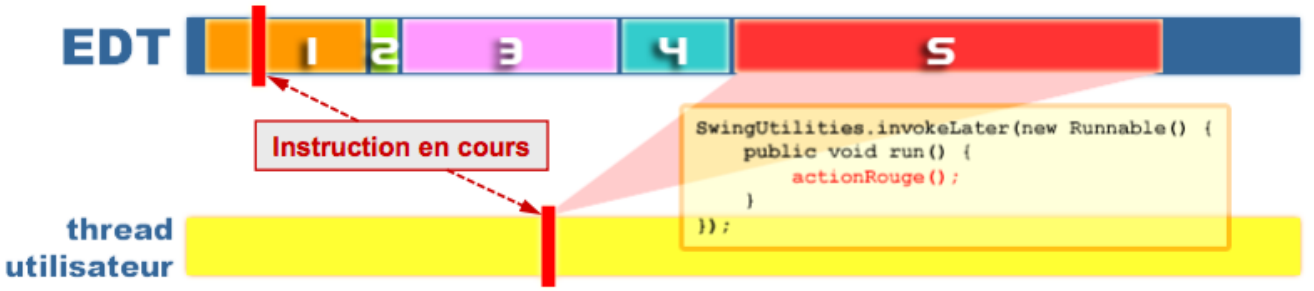
- La conception de la librairie Swing impose que les instructions qui accèdent et manipulent des composants de l'interface graphique soient effectuées dans un unique Thread : l’EDT.
- Autrement dit, la librairie Swing n'est pas **Thread-Safe**
- Ce Event Dispatch Thread est démarré automatiquement lorsque l'on utilise des composants de la librairie Swing.

Swing Event Dispatch Model



Séquencement des Threads

- L'objet Runnable s'exécutera de manière asynchrone par rapport au Thread utilisateur (Main Thread par exemple).
- Séquencement avec la méthode invokeLater() :




```
SwingUtilities.invokeLater(new Runnable() {
    public void run() {
        createAndShowGUI();
    }
});
```

Exemple de problème

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

public class FreezeEDT extends JFrame implements ActionListener {
  public FreezeEDT() {
    super("Freeze");
    JButton freezer = new JButton("Freeze");
    freezer.addActionListener(this);
    add(freezer);
    pack();
  }

  public void actionPerformed(ActionEvent e) {
    try {
      Thread.sleep(4000);
    } catch (InterruptedException ex) {
    }
  }

  public static void main(String[] args) {
    FreezeEDT edt = new FreezeEDT();
    edt.setVisible(true);
  }
}
```

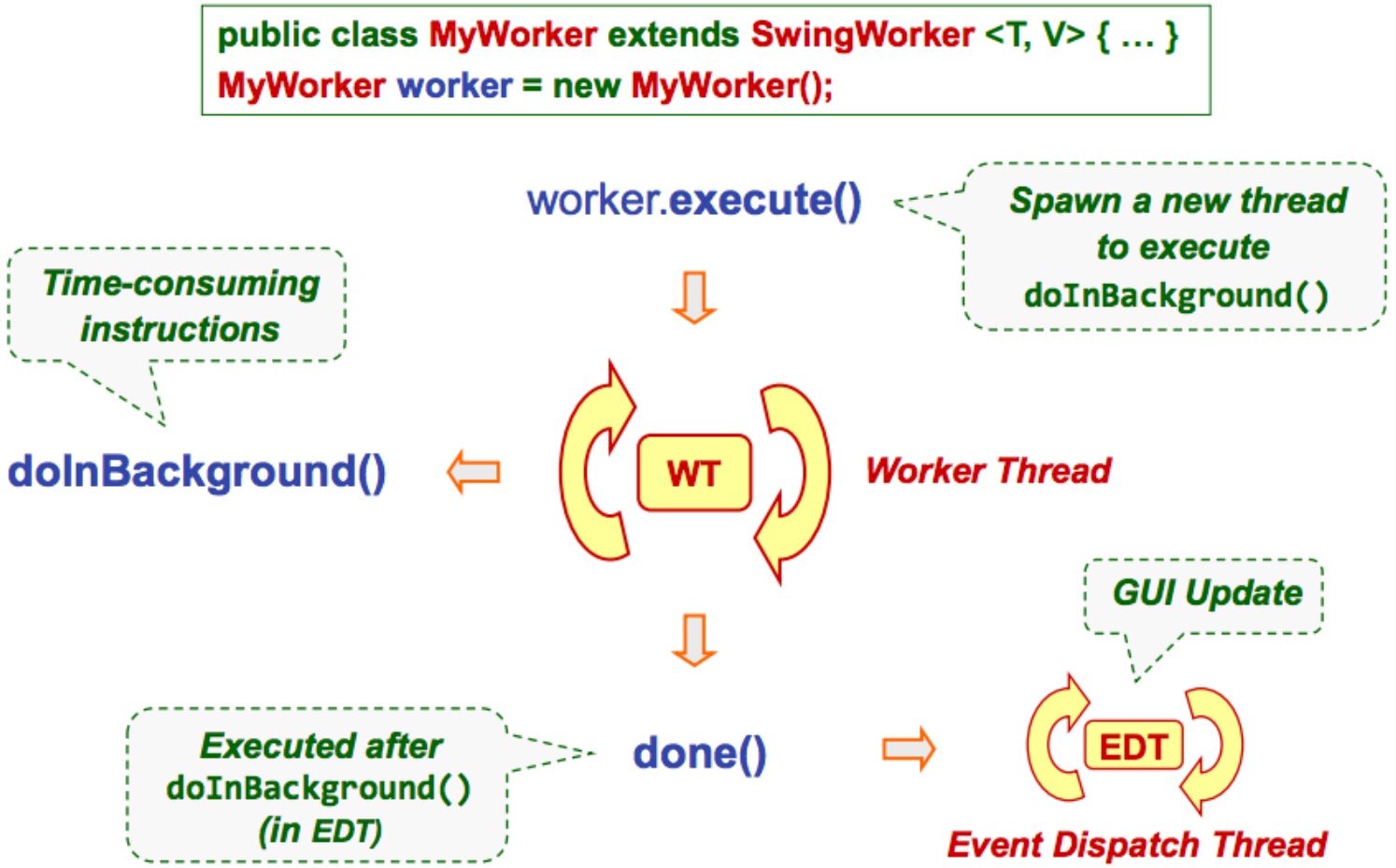
Exemple 2 chargement d'un fichier

```
private Document doc;
...
JButton button = new JButton("Open XML");
button.addActionListener(new ActionListener() {
   public void actionPerformed(ActionEvent e) {
     SwingWorker<Document, Void> worker =
        new SwingWorker<Document, Void>() {

        public Document doInBackground() {
          Document intDoc = loadXML();
          return intDoc;
        }
        public void done() {
          try {
             doc = get();
          } catch (InterruptedException ex) {
             ex.printStackTrace();
          } catch (ExecutionException ex) {
             ex.printStackTrace();
          }
        }
     };
     worker.execute();
   }
});
```

L'ActionListener créée un SwingWorker pour charge un fichier XML.Le chargement (long) du fichier va avoir lieu dans une seconde thread.

http://java.sun.com/developer/technicalArticles/javase/swingworker/

SwingWorker mode d'emploi



Exemple de SwingWorker

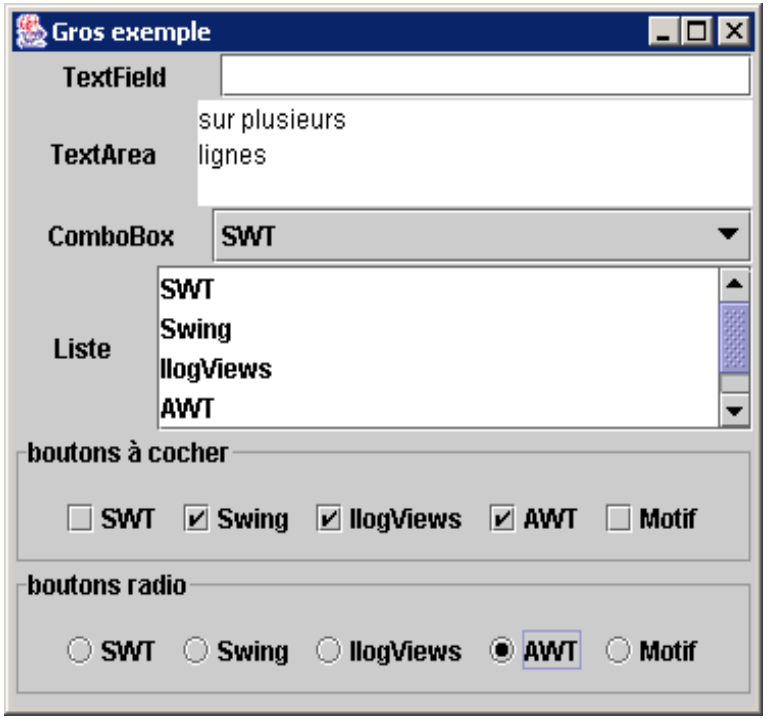
```
SwingWorker worker = new SwingWorker<ImageIcon[], Void>() {
    @Override
    public ImageIcon[] doInBackground() {
        final ImageIcon[] innerImgs = new ImageIcon[nimgs];
        for (int i = 0; i < nimgs; i++) {
            innerImgs[i] = loadImage(i+1);
        }
        return innerImgs;
    }

    @Override
    public void done() {
        //Remove the "Loading images" label.
        animator.removeAll();
        loopslot = -1;
        try {
            imgs = get();
        } catch (InterruptedException ignore) {}
        catch (java.util.concurrent.ExecutionException e) {
            String why = null;
            Throwable cause = e.getCause();
            if (cause != null) {
                why = cause.getMessage();
            } else {
                why = e.getMessage();
            }
            System.err.println("Error retrieving file: " + why);
        }
    }
};
```

Quelques composants basiques

Etiquette, zone / champs texte ...

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

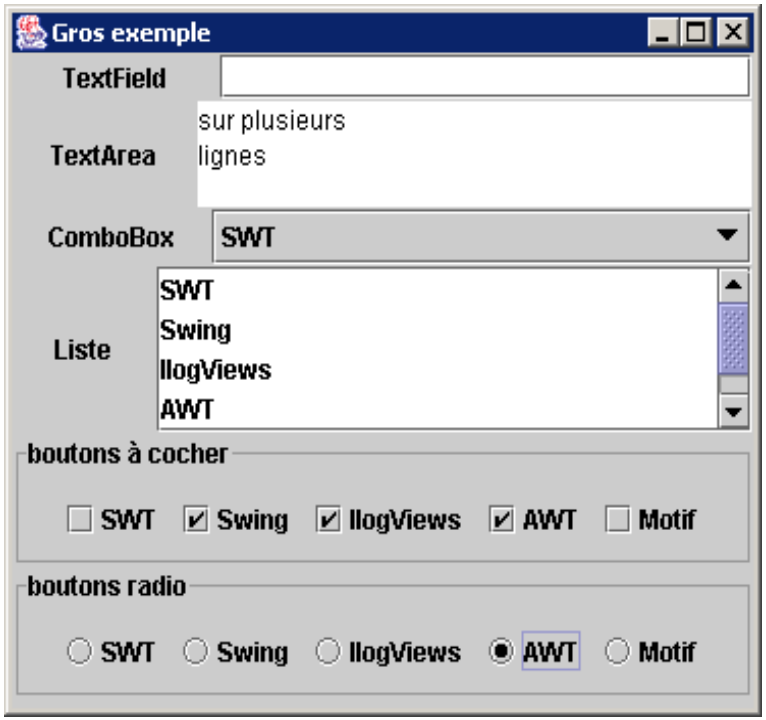
```
private JPanel createForm(String text,JComponent c) {
    JPanel panel=new JPanel(new GridBagLayout());
    GridBagConstraints constraints=new
    GridBagConstraints();
    constraints.weightx=1.0;
    panel.add(new JLabel(text),constraints);
    constraints.weightx=5.0;
    constraints.fill=GridBagConstraints.HORIZONTAL;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

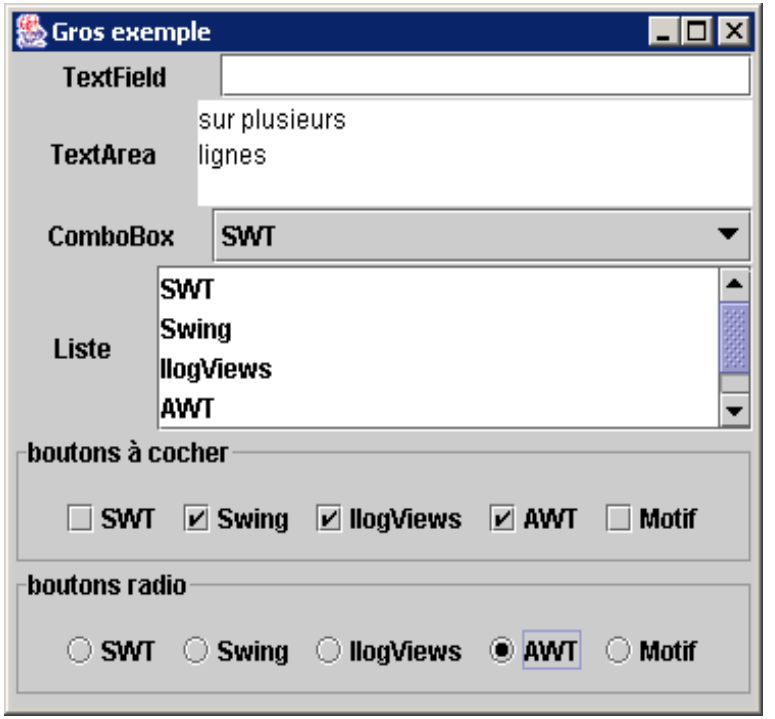
```
public JPanel createComboBoxPanel(Object... list) {
  JComboBox comboBox=new JComboBox(list);
  return createForm("ComboBox",comboBox);
}

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;
  }
```

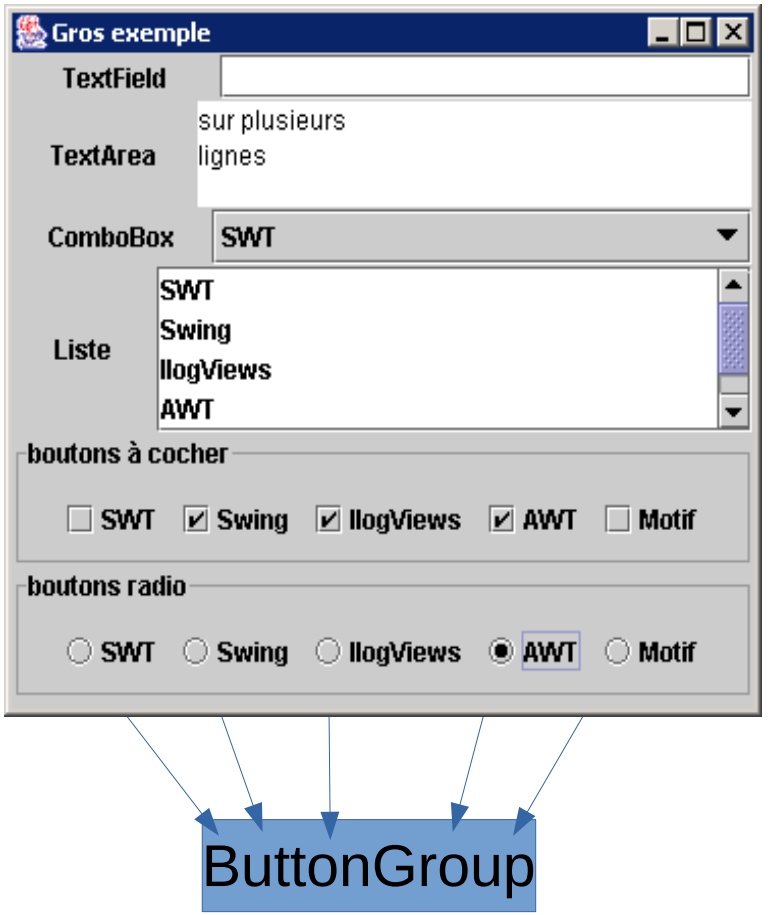


- **JCheckBox**, **JRadioButton** comme **JButton** héritent **d'AbstractButton**

Boutons radio et groupe de boutons

- Pour avoir un seul bouton sélectionné 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