Les interfaces Graphiques avec JAVA Swing

TP 02

Principes de base

- Des composants graphiques

  (exemple: JFrame, JButton …)

  - Hiérarchie de classes
- Des événements et les actions à effectuer

  (exemple presser un bouton)
- (Et d'autres choses…)

Principes

- Définir les composants (instance de classes)
- Les placer à la main (layout Manager) dans un JPanel ou un content pane ou en utilisant des outils comme netbeans
- Définir les actions associées aux événements (Listener) et les associer aux composants graphiques

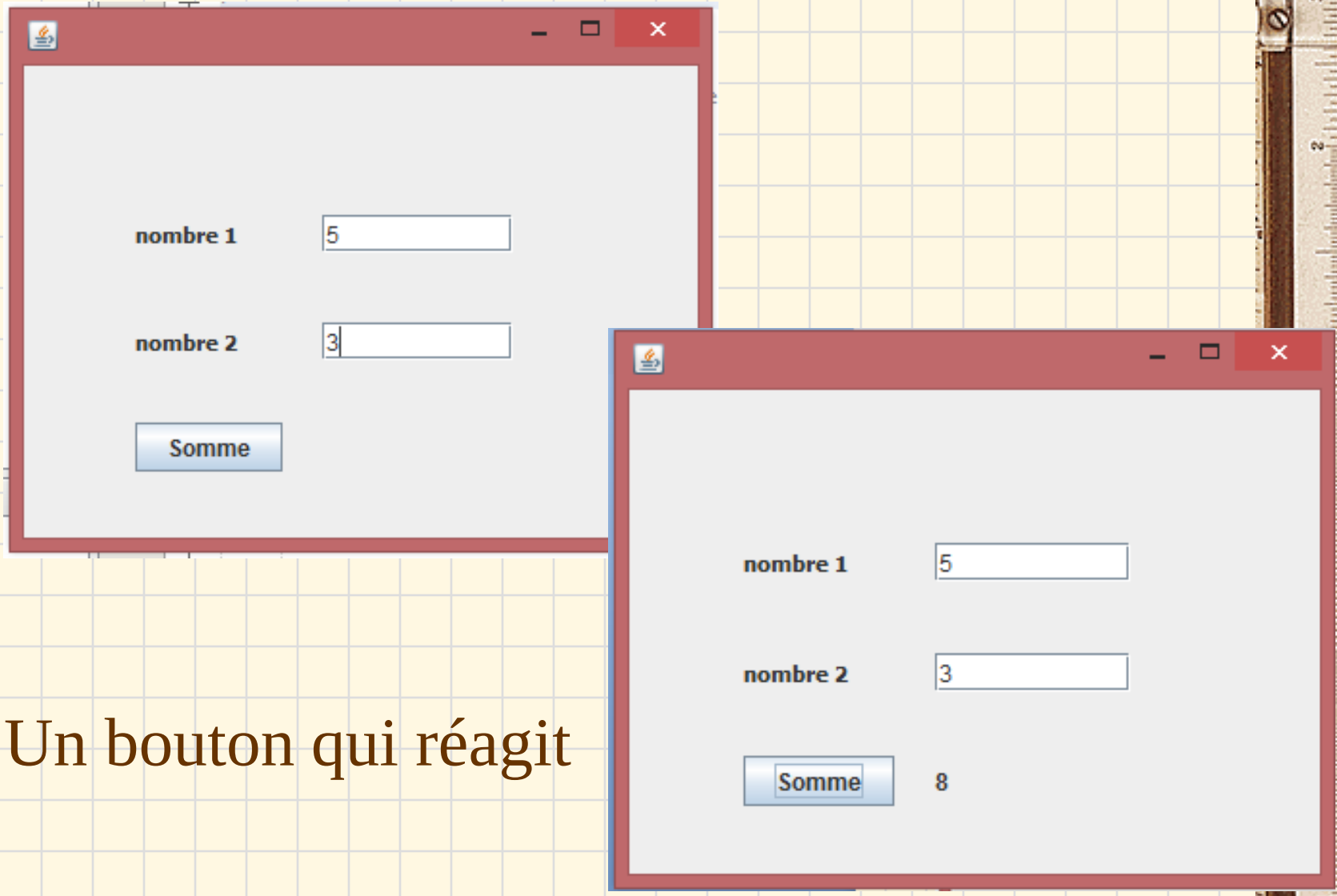
Principes

- Dans une interface graphique, le programme réagit aux interactions avec l'utilisateur
- Les interactions génèrent des événements
- Le programme est dirigé par les événements (event-driven)

Evénements

- Chaque composant génère des événements:
  - Presser un JButton génère un ActionEvent (système d'interface graphique)
    - Cet ActionEvent contient des infos (quel bouton, position de la souris, modificateurs…)
  - Un event listener (implémente ActionListener)
    - définit une méthode actionPerformed
    - S'enregistre auprès du bouton addActionListener
  - Quand le bouton est "clické",l'actionPerformed sera exécuté (avec l'ActionEvent comme paramètre)

Un exemple



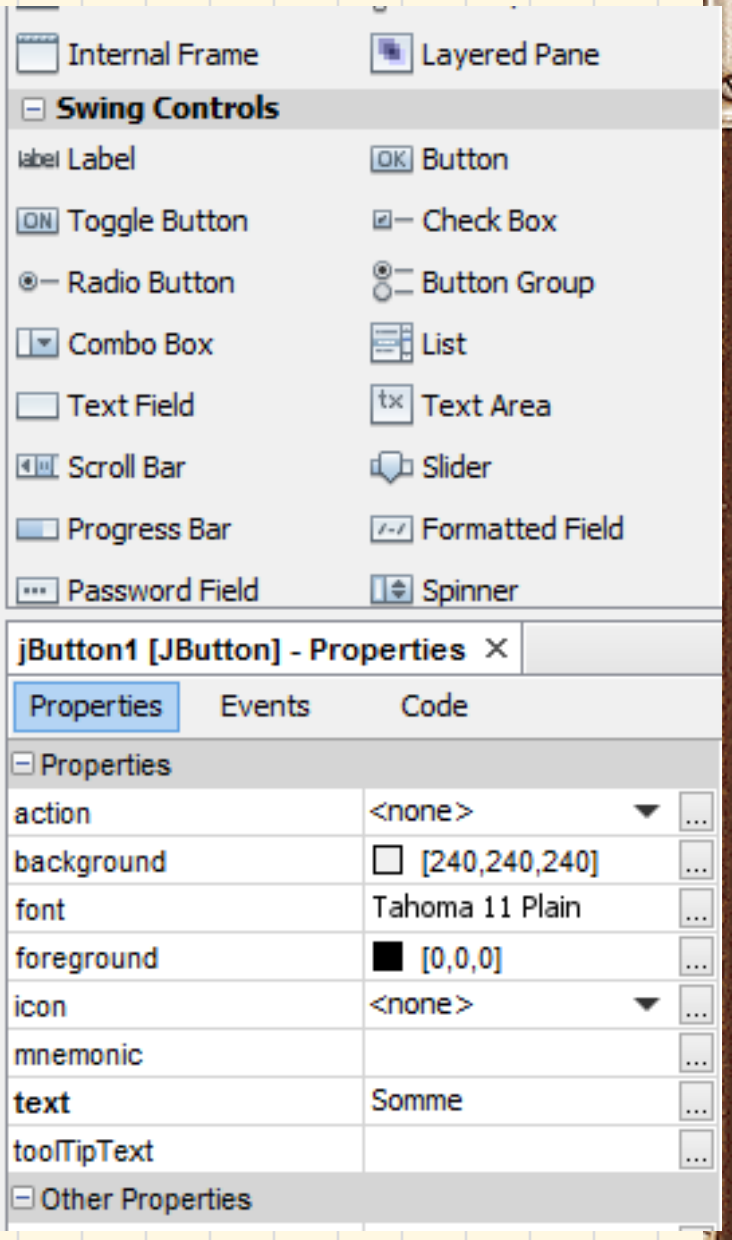
- Un bouton qui réagit

Le code:

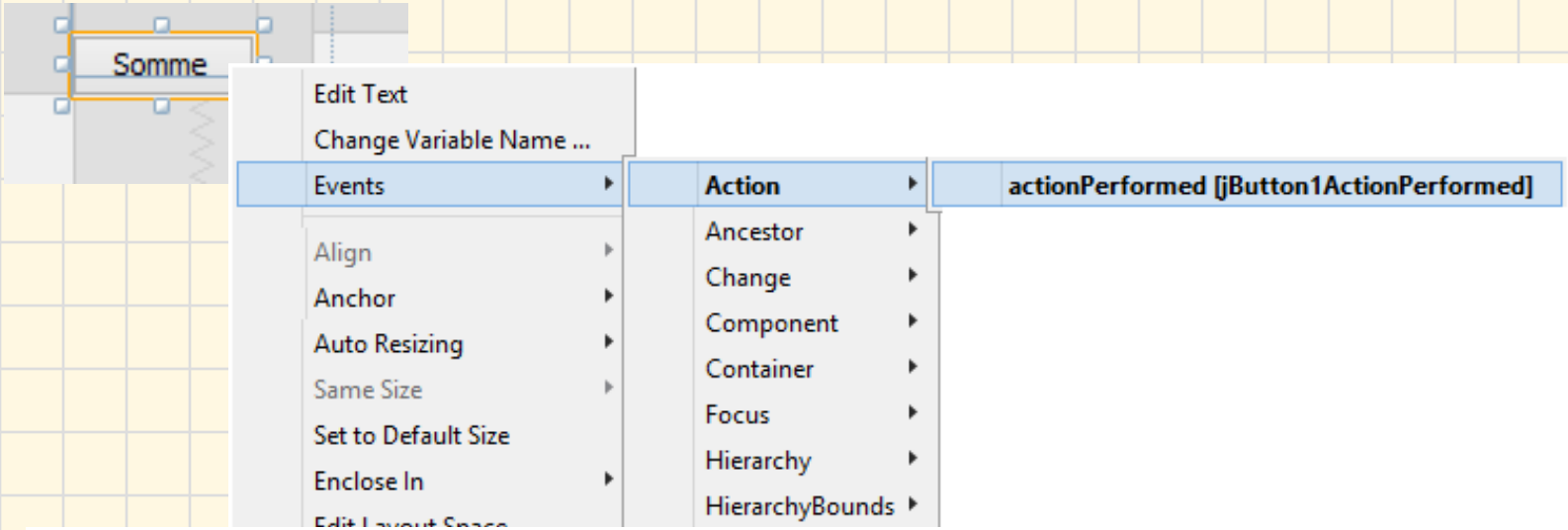
- Un JButton
- deux Jlabel, et 3 TextField
- Implementer ActionListener
  - actionPerfomed définit ce qui se passe quand le bouton est cliqué

Le code:


nombre 1
nombre 2
Somme


Internal Frame
Layered Pane
Swing Controls
Label
Button
Toggle Button
Check Box
Radio Button
Button Group
Combo Box
List
Text Field
Text Area
Scroll Bar
Slider
Progress Bar
Formatted Field
Password Field
Spinner
jButton1 [JButton] - Properties
Properties
Events
Code
Properties
action
<none>
background
[240,240,240]
font
Tahoma 11 Plain
foreground
[0,0,0]
icon
<none>
mnemonic
text
Somme
toolTipText
Other Properties

Le code:



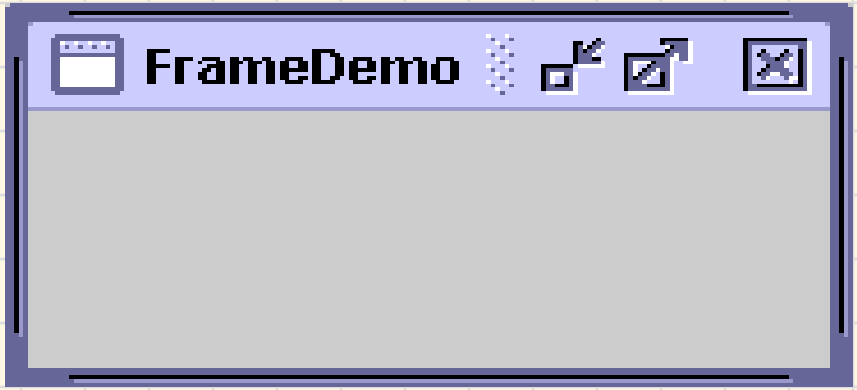

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    int num1= Integer.parseInt(jTextField1.getText());
    int num2= Integer.parseInt(jTextField2.getText());
    int somme= num1 + num2;
    jLabel3.setText(String.valueOf(somme));
}
```

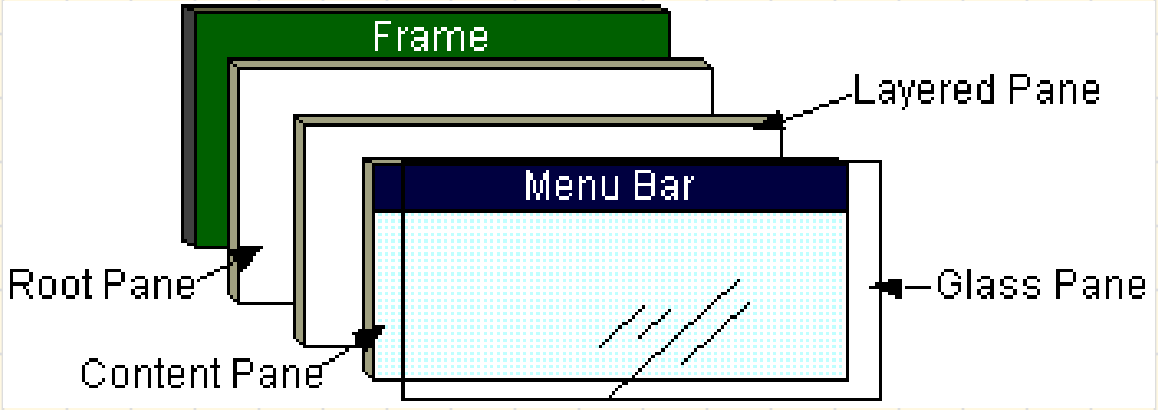
Les objets graphiques

- 3 niveaux
  - Haut niveau
    - Définir une fenêtre
    - *JApplet*, *JDialog*, *JFrame*, *JWindow*
  - Niveau intermédiaire
    - Pour composer la fenêtre
    - *JPanel, JScrollPane, JSplitPane, ...*
  - Niveau inférieur
    - Les éléments de base
    - *JButton, JCheckBox, JTextField, JTextArea, ...*

Insérer des éléments dans la fenêtre

- Composition d'une fenêtre JFrame





Jframe internal structure

Ajouter des composants dans une fenêtre

```
import javax.swing.*;
public class DisplayFrame {
    public static void main (String[] args) {
        JFrame f = new JFrame("FrameDemo");
        JLabel label = new JLabel("Hello World");
        JPanel p = (JPanel)f.getContentPane();
        p.add(label);
        f.setSize(300,200); //alternative: f.pack();
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setVisible(true);
    }
}
```

Haut niveau

Composante de base

Niveau intermédiaire

Composer une fenêtre

- Créer une fenêtre (1)
- Créer un ou des composants intermédiaires (2)
  - Pour *JFrame*, un *JPanel* est associé implicitement (ContentPane)
- Créer des composants de base (3)

- Insérer (3) dans (2)
- Insérer (2) dans (1) (s'ils ne sont pas déjà associés)

- Afficher

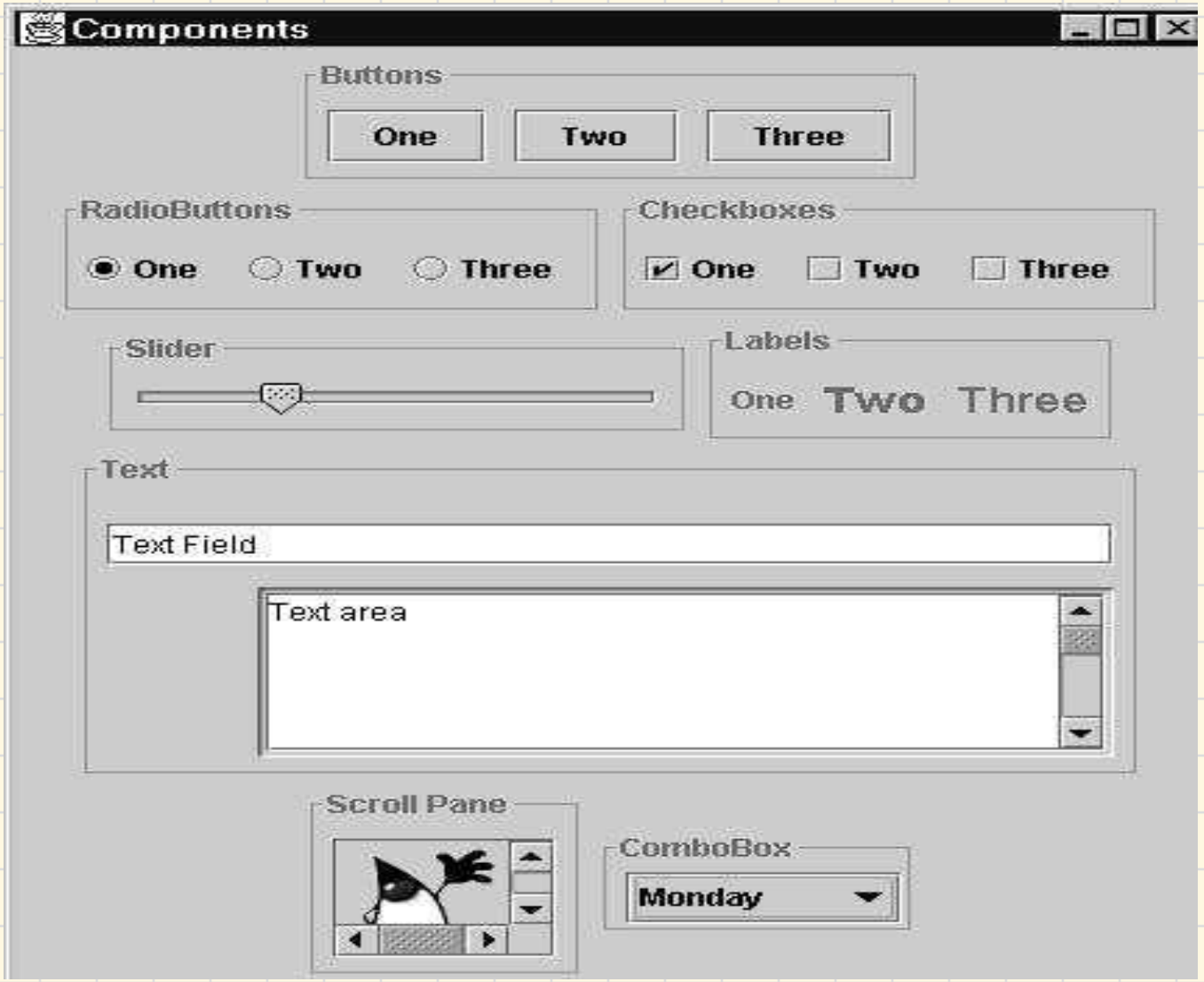
Composant de base (pour obtenir des données)

- *JButton*
- *JCheckBox* a toggled on/off button displaying state to user.
- *JRadioButton* a toggled on/off button displaying its state to user.
- *JComboBox* a drop-down list with optional editable text field. The user can key in a value or select a value from drop-down list.
- *Jlist* allows a user to select one or more items from a list.
- *Jmenu* popup list of items from which the user can select.
- *Jslider* lets user select a value by sliding a knob.
- *JTextField* area for entering a single line of input.

Composants de base pour afficher l'information

- *Jlabel* contains text string, an image, or both.
- *JProgressBar* communicates progress of some work.
- *JToolTip* describes purpose of another component.
- *Jtree* a component that displays hierarchical data in outline form.
- *Jtable* a component user to edit and display data in a two-dimensional grid.
- *JTextArea*, *JTextPane*, *JEditorPane*
  - define multi-line areas for displaying, entering, and editing text.

Swing components: Illustration



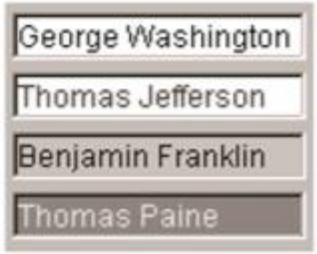
Buttons

Combo box

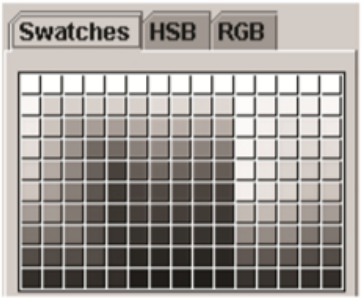
List

Menu

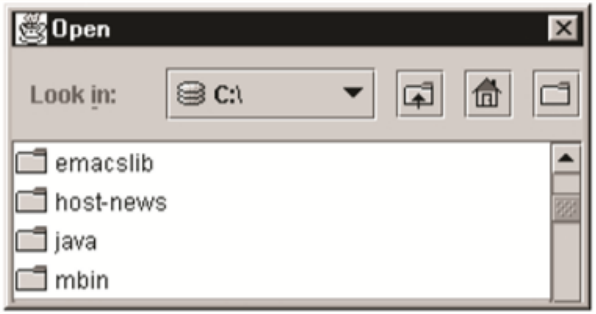
Slider



Text fields



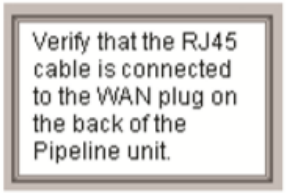
Color chooser



File chooser

First Na...	Last Name
Mark	Andrews
Tom	Ball
Alan	Chung
Jeff	Dinkins

Table



Text

Tree

Label

Progress bar

Tool tip

La classe JButton

Elle permet de définir des boutons sur lesquels on peut cliquer. Ses principales méthodes sont :

JButton(String) construction avec définition du texte contenu dans le bouton

JButton(ImageIcon) construction avec une icone dans le bouton

JButton(String,ImageIcon) construction avec définition du texte et d'une icône dans le bouton

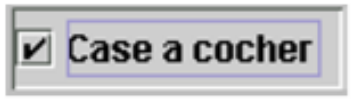
String getText() qui retourne le texte contenu dans le bouton

void setText(String) qui définit le texte contenu dans le bouton

void addActionListener(ActionListener) pour associer l'objet qui traitera les clics sur le bouton

La classe JCheckBox

Elle permet de définir des cases à cocher. Ses principales méthodes sont :



JCheckBox(String) construction avec définition du texte contenu dans la case à cocher

JCheckBox(String,boolean) construction avec en plus définition de l'état initial de la case à cocher

boolean isSelected() qui retourne l'état de la case à cocher (cochée ou non).

void setSelected(boolean) qui définit l'état de la case à cocher (cochée ou non).

String getText() qui retourne le texte contenu dans la case à cocher

void setText(String) qui définit le texte contenu dans la case à cocher

void addActionListener(ActionListener) pour associer l'objet qui traitera les actions sur la case à cocher

La classe JLabel

Elle permet de définir des textes fixes. Ses principales méthodes sont :

JLabel(String) qui construit le titre avec son contenu
void setText(String) qui définit le texte
String getText() qui retourne le texte

La classe JComboBox

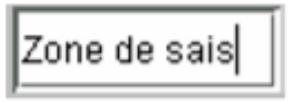
Elle permet de définir des boîtes permettant de choisir une valeur parmi celles proposées. Ses principales méthodes sont :

JComboBox(Object[]) construction avec définition de la liste. On peut utiliser un tableau de chaînes de caractères (**String**) ou de toute autre classe d'objets.
void addItem(Object) qui ajoute une valeur possible de choix
int getSelectedIndex() qui retourne le numéro du choix actuel

Object getSelectedItem() qui retourne l'objet associé au choix actuel. Attention l'objet retourné est de classe **Object**, il faudra utiliser l'opérateur de coercition pour le transformer en sa classe d'origine (**String** par exemple).
void setSelectedIndex(int) qui sélectionne un élément défini par son numéro
void addActionListener(ActionListener) pour associer l'objet qui traitera les choix faits

La classe JTextField

Elle permet de définir des zones de texte sur une seule ligne modifiables Ses principales méthodes sont :



JTextField(String) qui la crée avec un contenu initial

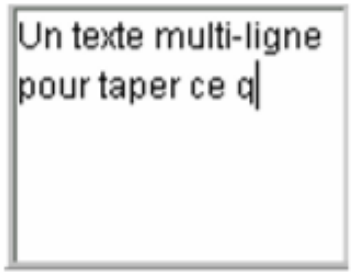
JTextField(String,int) qui la crée avec un contenu initial et définit le nombre de colones

void addActionListener(ActionListener) pour associer l'objet qui traitera les modifications du texte (voir 2.1.6.3)

Remarque : Lorsque l'on saisit du texte dans un tel composant celui-ci adapte sa taille au texte saisi au fur et à mesure. Ce comportement n'est pas toujours souhaitable, on peut l'éviter en lui définissant une taille préférentielle par sa méthode **setPreferredSize** et une taille minimale par sa méthode **SetMinimumSize** (voir 2.1.1). On procédera comme suit : texte.**setPreferedSize**(texte.**getPreferredSize**()) ;
texte.**setMinimumSize**(texte.**getPreferredSize**()) ;

La classe JTextArea

Elle permet de définir des zones de texte sur plusieurs lignes modifiables sans ascenseurs (pour disposer d'ascenseurs il faut faire appel à un contenant en possédant voir 2.1.3.4). Ses principales méthodes sont :



JTextArea(String) qui crée une zone de texte avec un contenu initial

JTextArea(String,int,int) qui crée une zone de texte avec un contenu initial et précise le nombre de lignes et de colonnes de la zone de texte

void append(String) qui ajoute la chaîne à la fin du texte affiché

void insert(String,int) qui insère la chaîne au texte affiché à partir du rang donné

void setTabSize(int) qui définit la distance entre tabulations.

void setLineWrap(boolean) qui détermine si les lignes longues doivent ou non être repliées.

void setWrapStyleWord(boolean) qui détermine si les lignes sont repliées en fin de mot (true) ou pas.

Remarque : Lorsque l'on saisit du texte dans une zone de texte celle-ci adapte sa taille au texte saisi au fur et à mesure. Ce comportement n'est pas toujours souhaitable, on peut l'éviter en mettant ce composant dans un **JScrollPane** pour disposer d'ascenseurs.

LayoutManager

- Chaque container est associé avec un *LayoutManager*.
- Spécifier un LayoutManager pour un *Container* (JPanel est une sous classe de Container):

```
public LayoutManager getLayout();
public void setLayout (LayoutManager manager);
```

e.g.

```
JButton b = new JButton("Press");
JLabel label = new JLabel("Hello World");
JPanel p = new JPanel();
p.setLayout(new BorderLayout());
p.add(label, BorderLayout.NORTH);
p.add(b, BorderLayout.SOUTH);
```

Les Layout

- BorderLayout
- FlowLayout
- GridLayout
- CardLayout
- ….

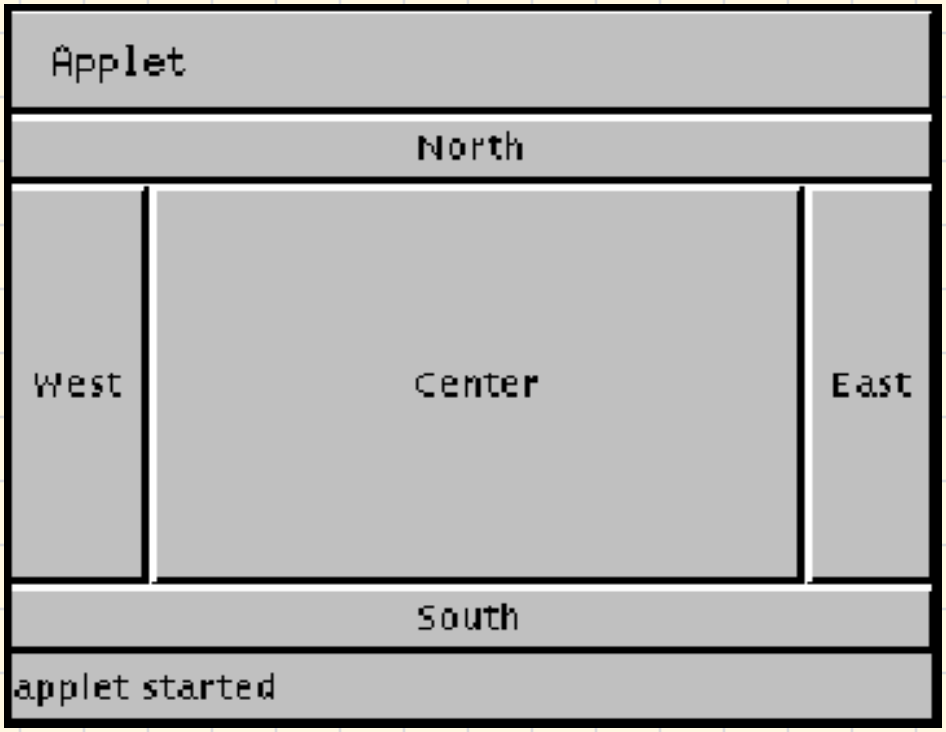
BorderLayout

En 5 zones

BorderLayout .NORTH

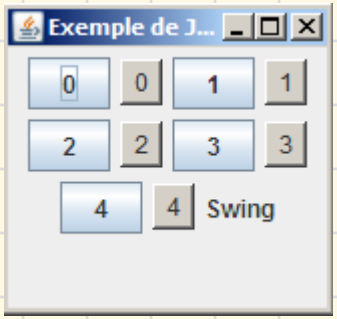
BorderLayout .WEST BorderLayout .CENTER, BorderLayout .EAST

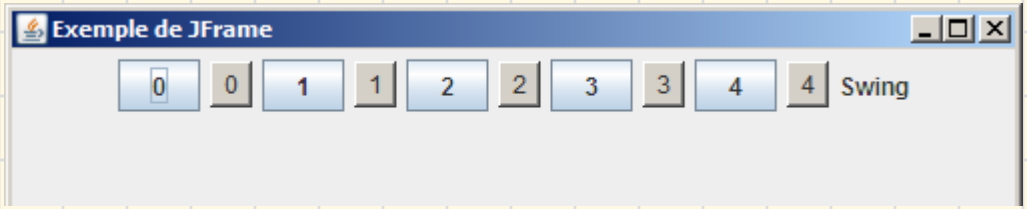
BorderLayout .SOUTH

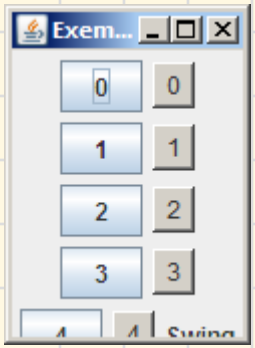


FlowLayout

- comme ils viennent …







Grid Layout

En une table