Chapitre 10

# Introduction aux interfaces graphiques en Java

# Interfaces graphiques en Java

- AWT (Abstract Windowing Toolkit)

- JFC (Java Foundation Classes) propose :
  - Composants Swing pour GUI
  - Pluggable Look-and-Feel
  - API d'accessibilité
  - API Java 2D
  - Drag-and-Drop entre app. Java et app. natives
  - Internationalisation

On s'intéresse dans la suite à Swing

Principales différences entre AWT et Swing
- Swing propose de nombreux nouveaux composants
- composants « légers » vs « lourds »
- Structure des composants s'inspire du modèle Modèle-Vue-Contrôleur

Les composants Swing remplacent souvent avantageusement leurs homologues AWT tant que possible.

Remarque : Swing utilise le modèle d'évènements de AWT.

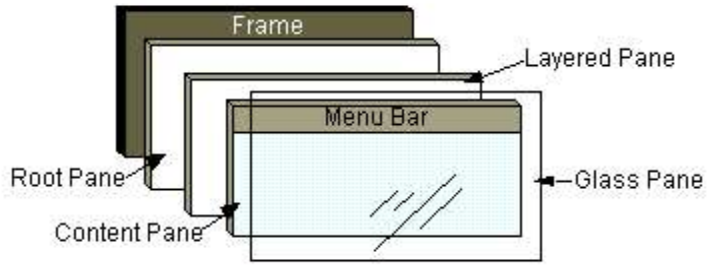
Paquetages : java.awt, javax.swing

→ Objectif : conception de GUI via une IDE.

Structure d’une interface graphique :

- conteneurs: Japplet,JFrame,JPanel...
- composants « atomiques »: JButton, Jlist, JPopupMenu
- gestionnaire de disposition
- interaction avec l’utilisateur : gestionnaire d’évènements

Structure d'un Jframe:




contentPane: panneau « contenu », accessible via getContentPane() de JFrame

# Premier exemple

```
import javax.swing.*;

class SimpleFrame extends JFrame
{

  public static final int WIDTH = 300;
  public static final int HEIGHT = 200;

  public SimpleFrame()
  {
    setSize(WIDTH, HEIGHT);
  }

  public static void main(String[] args)
  {
    SimpleFrame frame = new SimpleFrame();
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.show();
  }
}
```

# Deuxième exemple: Hello World

On écrit dans le JFrame via son contentPane (conteneur)

```
import javax.swing.*;

public class HelloWorldSwing {
        private static void void main(String[] args) {
        JFrame frame = new JFrame("HelloWorldSwing");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JLabel label = new JLabel("Hello World");
        frame.getContentPane().add(label);
        frame.show();
    }


}
```

# Gestionnaire de disposition

- Gestion de disposition « absolue » ?
- Java : disposition dépend du choix d'un gestionnaire de disposition (layout manager)

Interface: java.awt.LayoutManager

BorderLayout : gestionnaire par défaut du panneau de contenu contentpane d'un JFrame.

```
// TestBorderLayout.java

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

public class TestBorderLayout {
     public static void main(String[] args) {
             JFrame frame = new JFrame("TestBorderLayout");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
          JButton b1 = new JButton("bouton b1");
          JButton b2 = new JButton("bouton b2");
          JButton b3 = new JButton("bouton b3");
          JButton b4 = new JButton("bouton b4");
          JButton b5 = new JButton("bouton b5");
        JPanel p=(JPanel)frame.getContentPane();
        p.add(b1,"North");
        p.add(b2,"East");
        p.add(b3,"South");
        p.add(b4,"West");
        p.add(b5,"Center");
        frame.pack();
        frame.show();

    }
 }
```

**Implémentations de java.awt.LayoutManager :**

AWT et Swing:

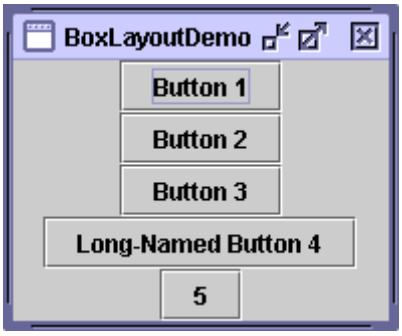
- BorderLayout (par défaut pour un content pane)
- *BoxLayout*
- CardLayout
- FlowLayout (par défaut pour un JPanel)
- GridBagLayout
- GridLayout
- *SpringLayout*

Choisir une mise en forme :

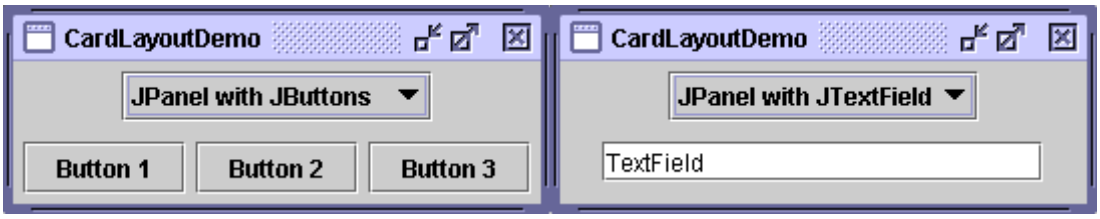
```
...
JPanel panel = new JPanel(new BorderLayout());
...
Container contentPane = frame.getContentPane();
contentPane.setLayout(new FlowLayout());
...
```

On peut définir son propre gestionnaire... ou ne pas en utiliser du tout. Dans ce dernier cas : positionnement « absolu », avec les problèmes associés (conteneur qui change de taille, fontes, …) .

## BoxLayout

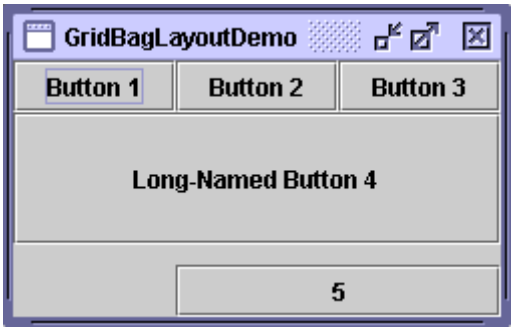



## CardLayout

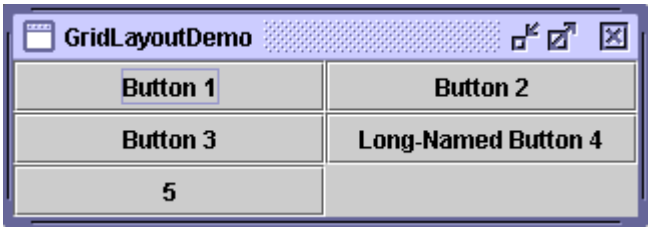



## FlowLayout

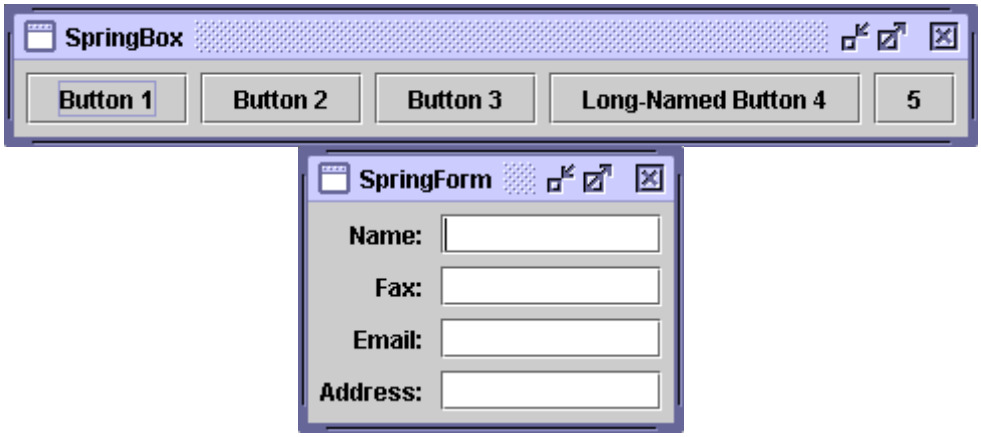

## GridBagLayout



## GridLayout




## SpringLayout



# Gestion des évènements

Evènement peut être généré par une action de l'utilisateur (pression d'une touche, clic sur un JButton) ou par le programme (ex : horloge, compteur).

Gestion des évènements avec AWT : modèle Emetteur/Récepteur

Un composant déclenche un évènement, envoyé sous forme d'un objet *e* à un ou plusieurs objet(s) auditeur (s)/écouteur(s) (*listeners*) qui se sont auparavant « enregistrés » auprès du composant. Ces écouteurs réagissent alors à l'évènement.

Classes d'évènements
Aux différents types d'évènements sont associés différentes classes Java.

Classe de base : java.util.EventObject

AWT et Swing définissent des sous-classes java.awt.AWTEvent et javax.swing.Event

- Méthode dans java.util.EventObject : getSource()
- On trouve dans les sous-classes des méthodes adaptées au type d'évènement,
  Exemple : dans MouseEvent : getX(), getY()

Ecouteurs d'évènements

Tous les composants AWT et Swing sont des sources d'évènements possibles. Chaque type de composant propose des méthodes permettant à un objet écouteur de s'enregistrer (ou de se désenregistrer) auprès de lui.
Forme générale :

- add???Listener()
- remove???Listener()

Exemple, pour la classe JButton:
addActionListener()
removeActionListener().

Un type d'évènement → Un type d'écouteur.
Exemple : ActionEvent → ActionListener

ActionListener : interface qui étend java.util.EventListener (marquage), et qui possède une méthode : actionPerformed() prenant en argument un ActionEvent. Cette méthode détermine la réponse de l'écouteur à l'évènement reçu.

Exemple : gestion d'un clic sur un bouton

- b un JButton
- l1,l2,..,ln des objets écouteurs, instances de classes qui implémentent ActionListener. On suppose que l1,…,ln se sont enregistrés comme écouteurs auprès de b.

Clic sur le bouton b

→ b transmet un objet ActionEvent e qui correspond à cet évènement

→ e est passé à actionPerformed() pour l1,…, ln

Remarques :

- un objet peut écouter plusieurs composants
- un composant peut s'écouter lui-même

on utilise souvent des classes internes/anonymes pour les écouteurs.

# Quelques classes d'écouteurs (extrait du Swing tutorial)

| Some Events and Their Associated Event Listeners | |
|---|---|
| **Act that Results in the Event** | **Listener Type** |
| User clicks a button, presses Enter while typing in a text field, or chooses a menu item | `ActionListener` |
| User closes a frame (main window) | `WindowListener` |
| User presses a mouse button while the cursor is over a component | `MouseListener` |
| User moves the mouse over a component | `MouseMotionListener` |
| Component becomes visible | `ComponentListener` |
| Component gets the keyboard focus | `FocusListener` |
| Table or list selection changes | `ListSelectionListener` |
| Any property in a component changes such as the text on a label | `PropertyChangeListener` |

# Exemple 1: compter le nombre de clics sur un bouton


## Première solution :

```
// CompteClic1.java

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

class CompteClic1 extends JFrame {
     public CompteClic1(){
          super("CompteClic1");
          setSize(200,200);
          setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
          ListenerLabel e=new ListenerLabel();
          JButton b=new JButton("cliquer!");
          b.addActionListener(e);
          JPanel p=new JPanel(new BorderLayout());
          p.add(b,"North");
          p.add(e,"Center");
          setContentPane(p);
     }
     public static void main(String[] args) {
          Frame f=new CompteClic1();
          f.pack();
          f.show();
     }
}

class ListenerLabel extends JLabel implements ActionListener
{
     int nbclics=0;
     ListenerLabel(){
          super("nombre de clics: 0",JLabel.CENTER);
     }
     public void actionPerformed(ActionEvent e){
         nbclics++;
         setText("nombre de clics:"+Integer.toString
(nbclics));
     }
}
```

# 2ème solution: avec une classe interne

```
// CompteClic2.java

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

class CompteClic2 extends JFrame {
     private int nbclics=0;
     private JLabel l=new JLabel("nombre de clics: "+nbclics );

     public CompteClic2(){
          super("CompteClic2");
          setSize(200,200);
          setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
          EcouteurInterne ecouteur=new EcouteurInterne();
          JButton b=new JButton("cliquer!");
          b.addActionListener(ecouteur);
          JPanel p=new JPanel(new BorderLayout());
          p.add(b,"North");
          p.add(l,"Center");
          setContentPane(p);
     }

     class EcouteurInterne implements ActionListener{
          public void actionPerformed(ActionEvent e){
               nbclics++;
               l.setText("nombre de clics:"
               +Integer.toString(nbclics));
          }
     }

     public static void main(String[] args) {
          Frame f=new CompteClic2();
          f.pack();
          f.show();
     }
}
```

# 3ème solution: avec une classe anonyme

```
// CompteClic3.java

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

class CompteClic3 extends JFrame {

     private int nbclics=0;
     private JLabel l=new JLabel("nombre de clics: "+nbclics );

     public CompteClic3(){
          super("CompteClic3");
          setSize(200,200);
          setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
          JButton b=new JButton("cliquer!");
          b.addActionListener(new ActionListener(){
               public void actionPerformed(ActionEvent e){
                    nbclics++;
                    l.setText("nombre de clics:
                    "+Integer.toString(nbclics));
               }
          });
          JPanel p=new JPanel(new BorderLayout());
          p.add(b,"North");
          p.add(l,"Center");
          setContentPane(p);
     }

     public static void main(String[] args) {
          Frame f=new CompteClic3();
          f.pack();
          f.show();
     }
}
```

# 4ème solution :

```
// CompteClic4.java

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

class CompteClic4 extends JFrame implements ActionListener{

     private int nbclics=0;
     private JLabel l=new JLabel("nombre de clics: "+nbclics );

     public CompteClic4(){
          super("CompteClic4");
          setSize(200,200);
          setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
          JButton b=new JButton("cliquer!");
          b.addActionListener(this);
          JPanel p=new JPanel(new BorderLayout());
          p.add(b,"North");
          p.add(l,"Center");
          setContentPane(p);
     }

     public void actionPerformed(ActionEvent e){
          nbclics++;
          l.setText("nombre de clics: "+Integer.toString
          (nbclics));
     }

     public static void main(String[] args) {
          Frame f=new CompteClic4();
          f.pack();
          f.show();
     }
}
```

# Exemple 2 : Un objet écouteur enregistré auprès de deux boutons

```
// PlusOuMoins.java

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

class PlusOuMoins extends JFrame {
    final int width  = 300;
    final int height = 300;
    public PlusOuMoins() {
     super("Plus ou moins");
     setSize(width, height);
     setDefaultCloseOperation(EXIT_ON_CLOSE);
     ListenerLabel label = new ListenerLabel();
     JButton plusButton = new IncrButton("+", +1);
     plusButton.addActionListener(label);
     JButton moinsButton = new IncrButton("-", -1);
     moinsButton.addActionListener(label);
     JPanel mainPanel = new JPanel(new BorderLayout());
     mainPanel.add(plusButton, "East");
     mainPanel.add(label, "Center");
     mainPanel.add(moinsButton, "West");
     setContentPane(mainPanel);
    }
    public static void main(String [] args){
     Frame f= new PlusOuMoins();
     f.setVisible(true);
    }
}

class IncrButton extends JButton {
    private int incr;
    IncrButton(String title, int incr) {
          super(title);
          this.incr = incr;
    }
    int getIncr() { return incr; }
}

class ListenerLabel extends JLabel implements ActionListener {
    int value = 0;
    ListenerLabel () {
          super("0", JLabel.CENTER);
    }
    public void actionPerformed(ActionEvent event) {
          IncrButton button = (IncrButton) event.getSource();
          value += button.getIncr();
          setText(Integer.toString(value));
    }
}
```

# Adaptateurs

Interface MouseListener :

```
void mouseClicked(MouseEvent e)
void mouseEntered(MouseEvent e)
void mouseExited(MouseEvent e)
void mousePressed(MouseEvent e)
void mouseReleased(MouseEvent e)
```

Si on veut écouter que le clic de la souris, il nous faut quand même implémenter toutes les méthodes de MouseListener.

Classe MouseAdapter: classe (abstraite) qui implémente toutes les méthodes de MouseListener avec un corps vide.

Utilisation : dériver la classe de MouseAdapter et redéfinir la méthode mouseClicked

```
class MonEcouteur extends MouseAdapter{
        void mouseClicked(MouseEvent e){
                ...
        }
}
```

# Le modèle Modèle-Vue-Contrôleur (MVC)

Modèle de conception (design pattern) permettant de séparer

- les données utiles à l'application (modèle)
- la/les présentation(s) que l'on en fait (vue(s))
- les mécanismes d'interaction de l'application avec l'utilisateur (contrôleur(s))

Communication entre les trois composants:

- Le contrôleur est activé par une action qui se produit dans la vue.
- Il transmet au modèle la requête associée à cette action (interrogation, mise à jour).
- Il reçoit la « réponse » du modèle, qu'il transmet à la vue.

Intérêt de la séparation modèle/vue : on peut proposer facilement plusieurs présentations (éventuellement simultanées) d'un même modèle.

Exemple : cf projet…

Remarque : rien n'oblige à se conformer au modèle MVC coûte que coûte !

# Modèle MVC et composants Swing

L'architecture des composants Swing s'inspire du modèle MVC.

Structure d'un composant :
- un modèle
- un *délégué UI* qui combine vue et contrôleur.

Les différents types de composants utilisent différentes classes pour leur modèle :

- JButton : ButtonModel (interface)
- JList : ListModel pour les données, ListSelectionModel pour la sélection des données
- JTree : TreeModel, TreeSelectionModel
- JTextComponent : Document

Exemple: si b est un JButton, b.getModel() renvoie un objet de la classe DefaultButtonModel, classe qui implémente l'interface ButtonModel.

# Interface ButtonModel

**All Known Implementing Classes:**
DefaultButtonModel

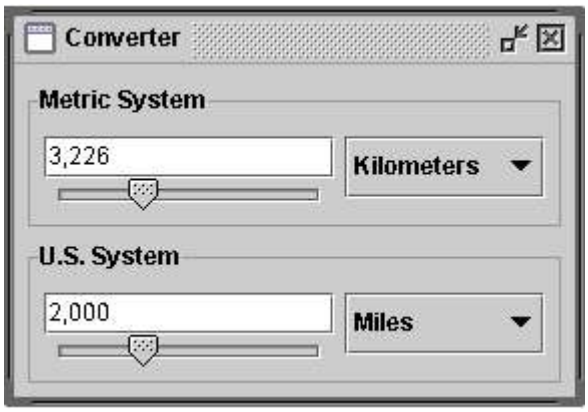
## Method Summary [extrait]

| | |
|---|---|
| void | **addActionListener**(ActionListener l)<br>Adds an ActionListener to the button. |
| void | **addChangeListener**(ChangeListener l)<br>Adds a ChangeListener to the button. |
| void | **addItemListener**(ItemListener l)<br>Adds an ItemListener to the button. |
| String | **getActionCommand**()<br>Returns the action command for this button. |
| boolean | **isArmed**()<br>Indicates partial commitment towards pressing the button. |
| boolean | **isEnabled**()<br>Indicates if the button can be selected or pressed by an input device (such as a mouse pointer). |
| boolean | **isPressed**()<br>Indicates if button has been pressed. |
| boolean | **isRollover**()<br>Indicates that the mouse is over the button. |
| boolean | **isSelected**()<br>Indicates if the button has been selected. |
| void | **setArmed**(boolean b)<br>Marks the button as "armed". |
| void | **setEnabled**(boolean b)<br>Enables or disables the button. |
| void | **setGroup**(ButtonGroup group)<br>Identifies the group this button belongs to -- needed for radio buttons, which are mutually exclusive within their group. |
| void | **setPressed**(boolean b)<br>Sets the button to pressed or unpressed. |
| void | **setRollover**(boolean b)<br>Sets or clears the button's rollover state |
| void | **setSelected**(boolean b)<br>Selects or deselects the button. |

On manipule très souvent les composants sans se préoccuper de leur modèle.

Mais il se peut que l'implémentation par défaut de l'interface ne convienne pas. On peut dans ce cas utiliser une nouvelle classe qui implémente cette interface.

Exemple (cf tutorial Swing) : convertisseur




Le modèle d'un composant « slider » doit implémenter l'interface BoundedRangeModel. Classe utilisée par défaut : DefaultBoundedRangeModel. Or celle-ci manipule des données entières.

Mise en place (très simplifiée) :

```
Class ConverterRangeModel implements BoundedRangeModel{
        …
}
…
JSlider s=new JSlider(new ConverterRangeModel());
...
```