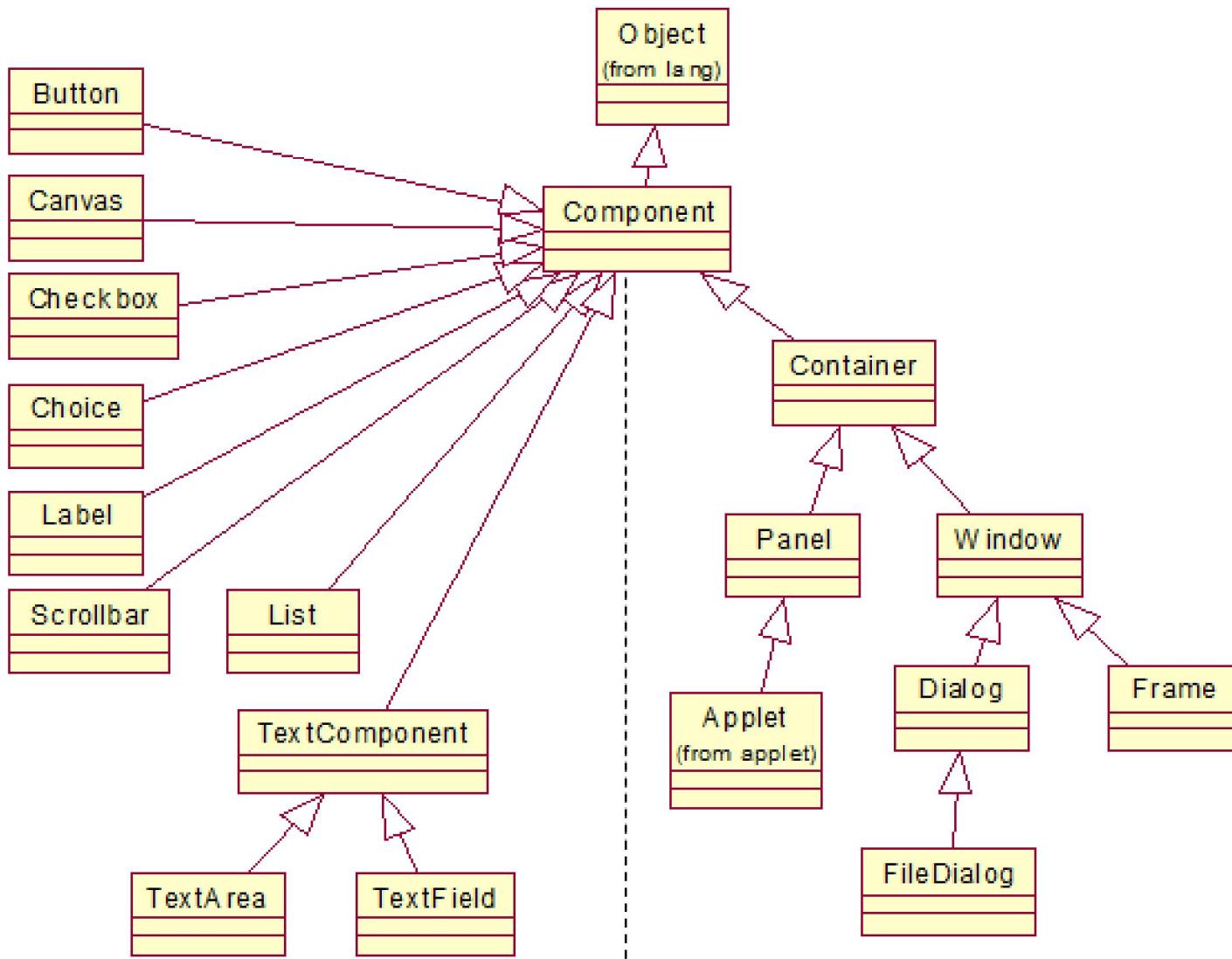


JAVA
Boîte à Outils: Java Swing
EVENEMENTS

Conteneurs et composants

Java: Hiérarchie d'héritage des principaux composants de l'interface

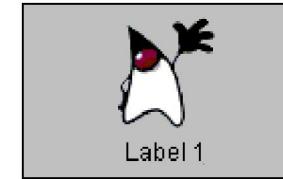


JAVA SWING

- ▶ **Les « Components »:** Constituent différents éléments de l'affichage (boutons, barres de menus, etc.). *Ex: JButton, JLabel, JTextArea, JCheckbox, etc..*
- ▶ **Les « Containers » :** destinés à accueillir des composants, gèrent l'affichage des composants (*Ex: JFrame, JPanel, JScrollPane*)
- ▶ **Les «LayoutManagers»:** Gèrent la disposition des composants au sein d'un conteneur (*BorderLayout, FlowLayout, GridLayout, ...*)



JLabel



- ▶ Javax.swing.JLabel
- ▶ descriptif : texte statique + image
- ▶ ex: devant un champ de saisie

JLabel jl = new JLabel("Nom:");

ou jl.setText("Nom:"); // -> .getText()

jl.setIcon(new ImageIcon("image.gif"));

jl.setVerticalTextPosition(SwingConstants.BOTTOM)

jl.setHorizontalTextPosition(SwingConstants.CENTER)

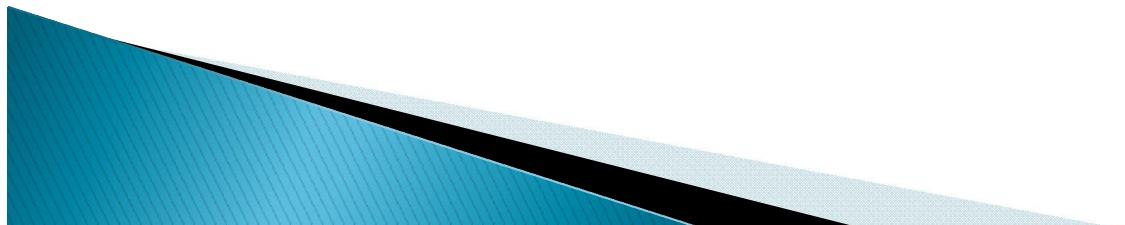
;

JTextField

George Washington
Thomas Jefferson
Benjamin Franklin
Thomas Paine

- ▶ Javax.swing.JTextField
- ▶ saisie de texte (non typé)

```
JTextField jt = new JTextField("Mohamed");
String nom = new String("Mohamed");
jt.setText(nom);
jt.setColumns(nom.length());
jt.copy();    jt.cut();    jt.paste();
```

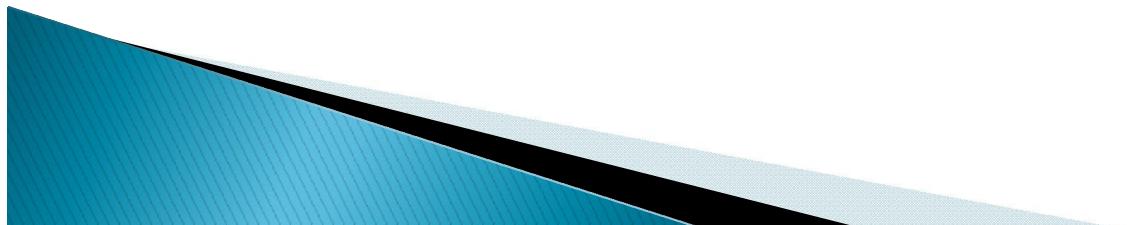


JButton



- ▶ Bouton simple à états

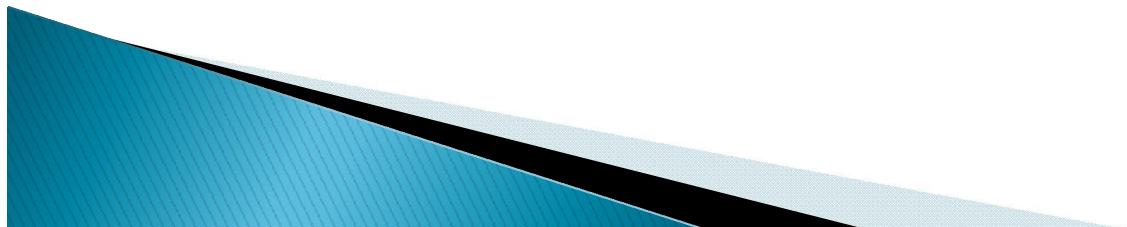
```
JButton jb= new JButton("OK", new ImageIcon("img.gif"));
jb.setRolloverIcon(new ImageIcon("img1.gif"));
jb.setPressedIcon(new ImageIcon("img2.gif"));
jb.setDisabledIcon(new ImageIcon("img3.gif"));
jb.setMnemonic('o'); // ALT + o
jb.setBorderPainted(false);
jb.setFocusPainted(false);
jb.doClick();
```



JmachinButton



- ▶ **JToggleButton**
 - deux états (setIcon et setSelectedIcon)
- ▶ **JCheckBox**
 - cases à cocher
- ▶ **JRadioButton**
 - dans un groupe de boutons “radio”



Exemple de Radio

```
ButtonGroup grp = new ButtonGroup();
```

```
JRadioButton r1 = new JRadioButton("it1");  
JRadioButton r2 = new JRadioButton("it2");  
r2.setSelected(true);
```

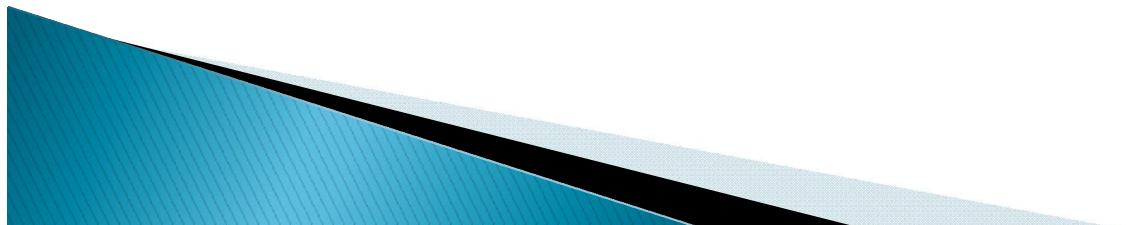
```
grp.add(r1);  
grp.add(r2);
```

JComboBox



- ▶ Liste déroulante (ouverte ou fermée)
- ▶ vector ou tableau d'objets passés en paramètres

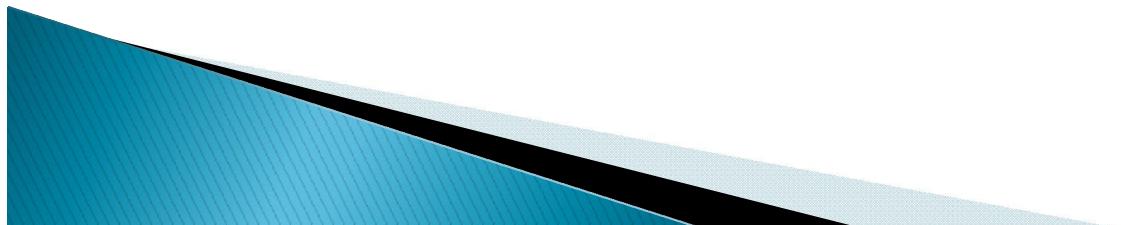
```
JComboBox cb = new JComboBox( items);
cb.setMaximumRowCount(4);
cb.setEditable(true); // JTextField
```



JMenu



- ▶ Une instance de JMenuBar par Jframe
`setJMenuBar(JMenuBar mb);`
- ▶ Plusieurs Jmenu par JMenuBar
`add(JMenu jm);`
- ▶ Plusieurs JMenuItem/JCheckboxMenuItem par Jmenu
`add(JMenuItem mi);`
`addSeparator();`



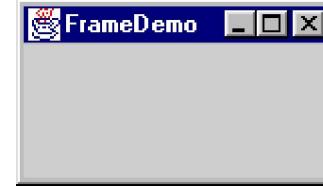
Les composants de SWING



Dialog



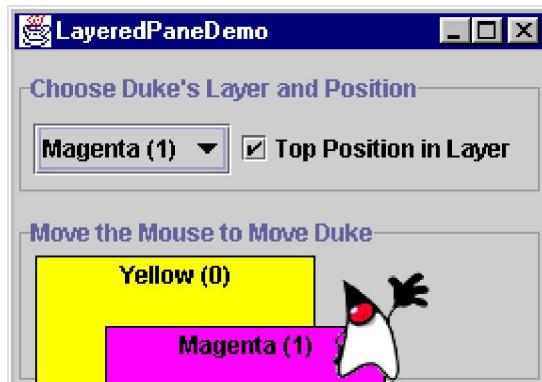
Tabbed Pane



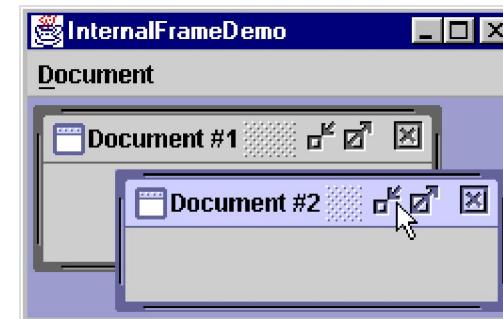
Frame



Split pane



Layered pane

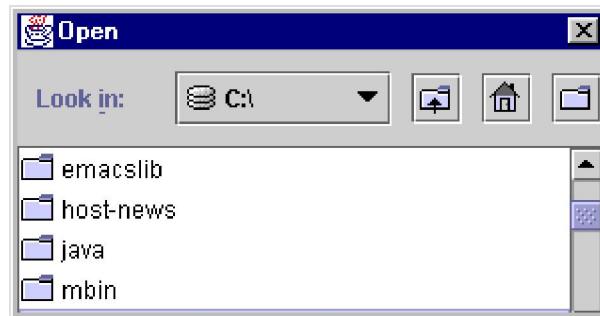


Internal frame



Tool bar

Les composants SWING



File chooser



List



Color chooser



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

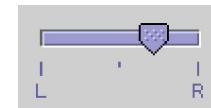
Table

Verify that the RJ45 cable is connected to the WAN plug on the back of the Pipeline unit.

Text



Tree

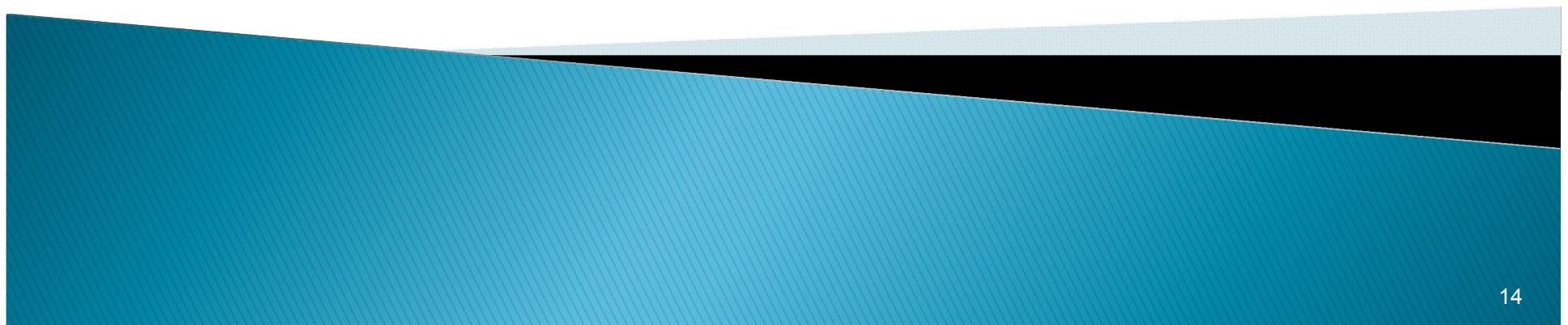


Slider

Capacités communes des composants

- ▶ (dés)activation
 - isEnabled()** **setEnabled(...)**
- ▶ (in)visible
 - setVisible(...)** **isVisible()**
 - module le coût de l'instanciation d'un container !
- ▶ tailles réelle et souhaitable
 - Dimension **getSize()** ou **getSize(Dimension r)** ,
setSize(...)
 - Dimension **getPreferredSize()** ;
setPreferredSize(Dimension r);

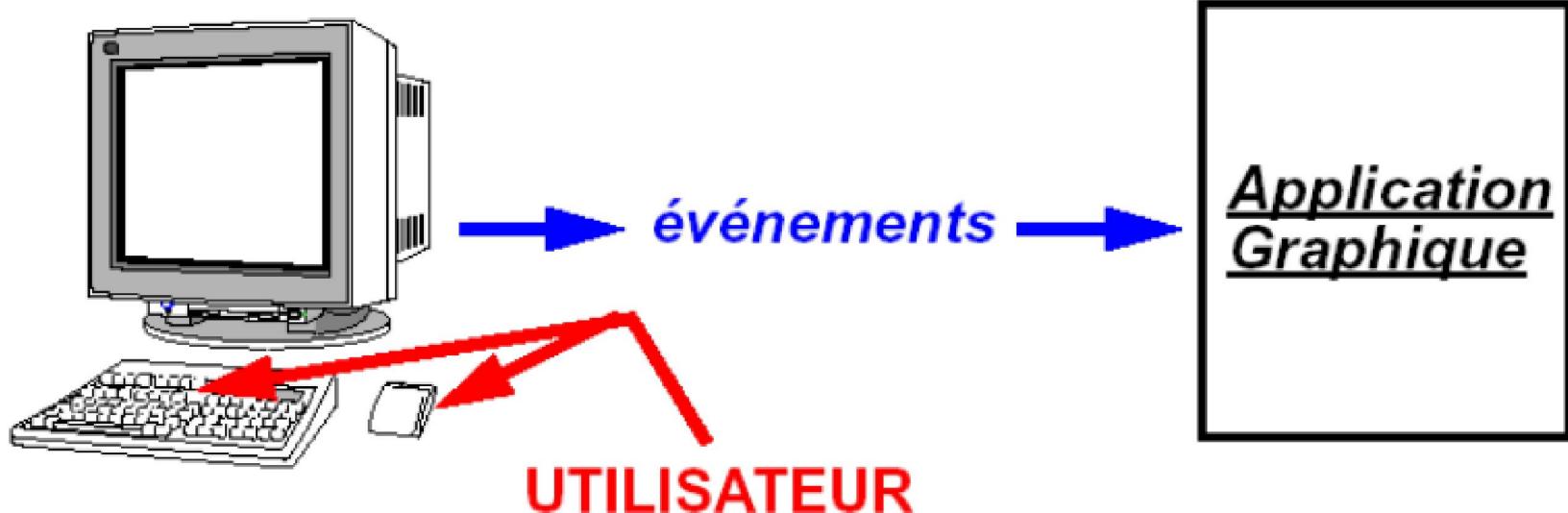
EVENEMENTS



SWING: Evénements

Evénements

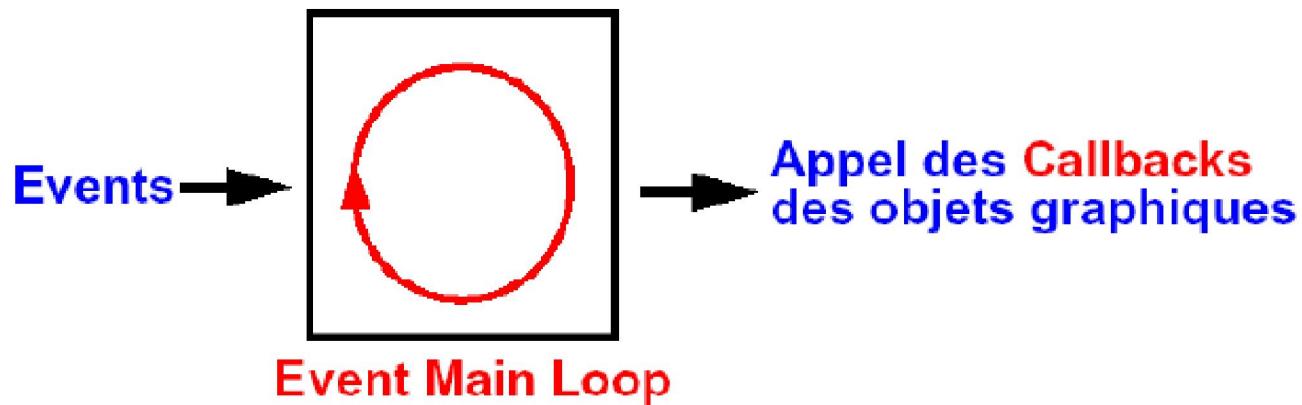
- envoyés à l'application ciblée
- à chaque action élémentaire de l'utilisateur



Boucle de Gestion des Événements

Boucle infinie qui

- récupère les événements
- appelle les fonctions de « callback » des composants graphiques



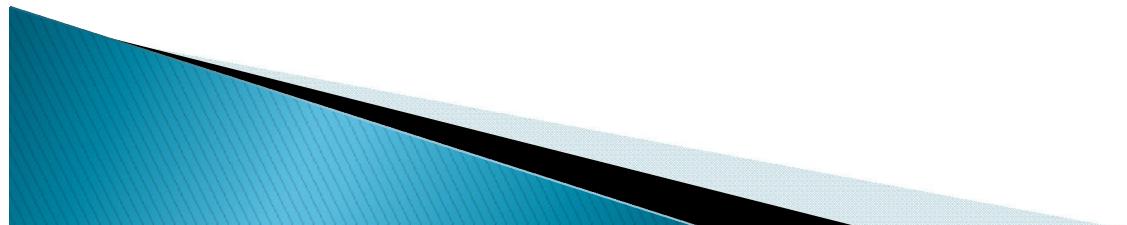
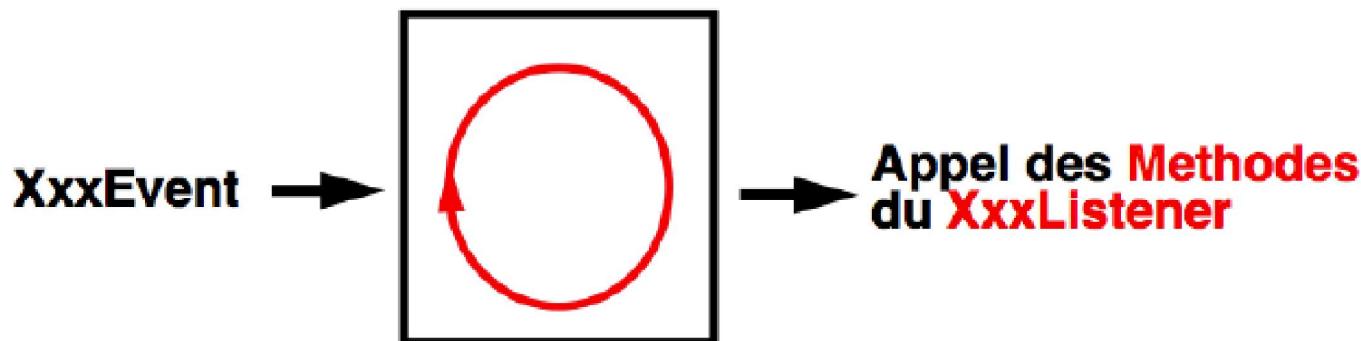
Lancée automatiquement

- à la fin de la fonction `main()` dans le cas de Java

Gestion des Evénements

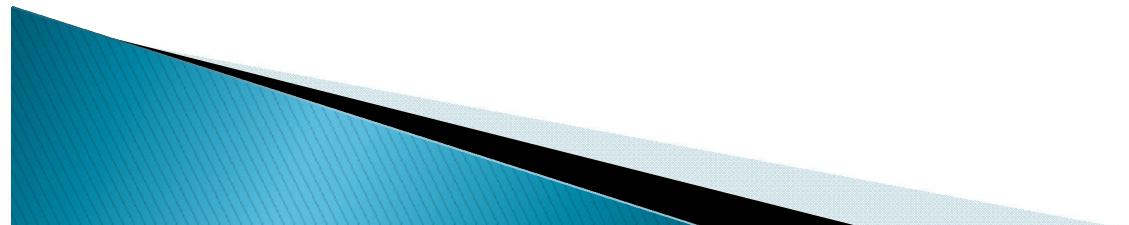
Les observateurs détectent les événements

- ce sont des **fonctions de callback** (en C, C++...)
- ou des objets comme les **Listeners** de Java
 - leurs méthodes servent alors de fonctions de callback



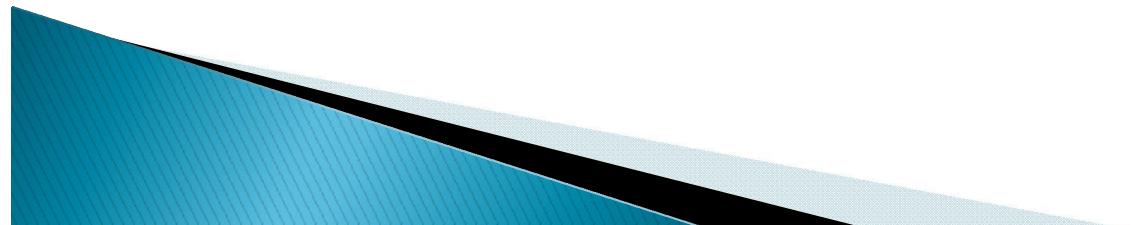
Evénements graphiques

- L'utilisateur effectue
 - ◆ une action au niveau de l'interface utilisateur (clic souris, sélection d'un item, etc)
 - ◆ alors un événement graphique est émis.
- Lorsqu'un événement se produit
 - ◆ il est reçu par le composant avec lequel l'utilisateur interagit (ex: bouton, curseur, champ de texte ...)
 - ◆ Ce composant transmet cet événement à un autre objet, un écouteur qui possède une méthode pour traiter l'événement (interaction avec l'utilisateur)



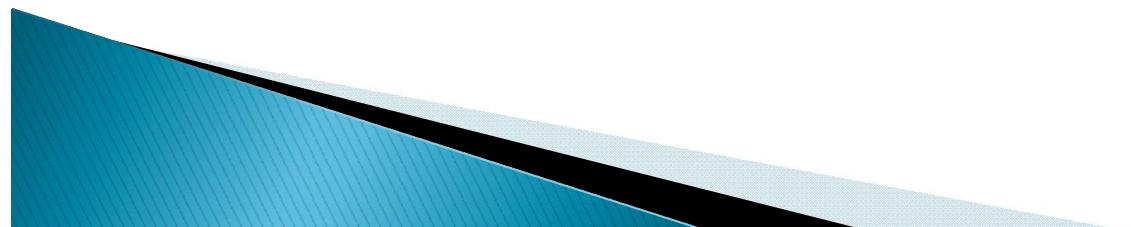
Événements graphiques

- La gestion des événements passe par l'utilisation d'objets "écouteurs d'événements" (*Listener*) et d'objets sources d'événements:
 - ◆ Objet écouteur: instance d'une classe implémentant l'interface *EventListener* (ou interface "fille")
 - ◆ Source d'événements: objet pouvant recenser des objets écouteurs et leur envoyer des objets événements.



Evénements graphiques

- Lorsqu'un événement se produit:
 - ◆ La source d'événements envoie un objet événement correspondant à tous ses écouteurs
 - ◆ Les objets écouteurs utilisent l'information dans l'objet événement pour déterminer leur réponse



Evénements: Exemple 1

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

class MonAction implements ActionListener {
    public void actionPerformed (ActionEvent e) {
        System.out.println ("Une action a eu lieu") ;
    }
}

public class TestBouton {
    public TestBouton(){
        Frame f = new Frame ("TestBouton");
        Button b = new Button ("Cliquer ici");
        f.add (b) ;
        f.pack ();  f.setVisible (true) ;
        b.addActionListener (new MonAction ());
    }

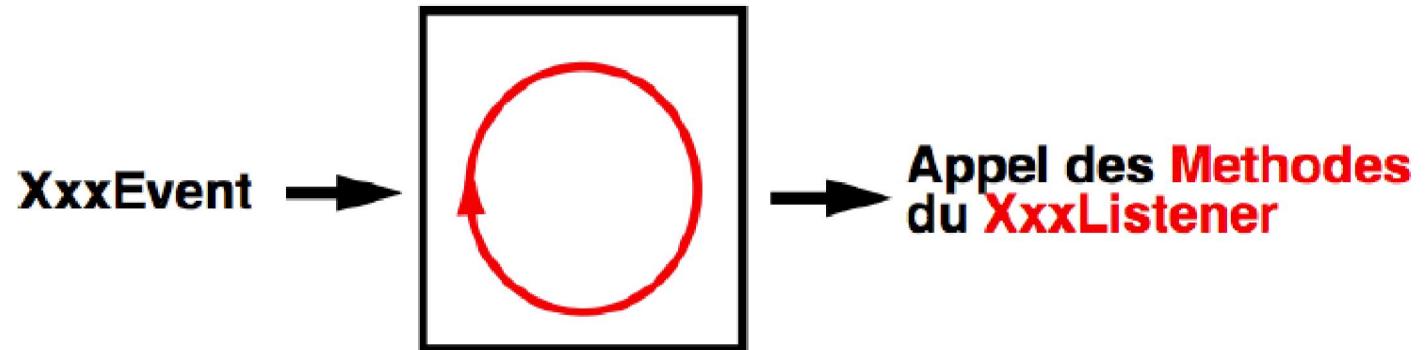
    public static void main(String args[ ]) {
        TestBouton test = new TestBouton();} }
```

Problème: communication entre objets séparés

Evénements Graphiques

Event Listeners

- à chaque sous-classe d'**AWTEvent** correspond un **EventListener** (en général)



Exemple : ActionEvent

- événement : **ActionEvent**
- listener : **ActionListener**
- méthode : **actionPerformed(ActionEvent)**

validation bouton:
– clic ou space



Evénements: Exemple 2

```
import java.awt.*;
import java.awt.event.*;
public class EssaiActionEvent1 extends Frame
    implements ActionListener
{
    public static void main (String args[])
    {EssaiActionEvent1 f= new EssaiActionEvent1();}
    public EssaiActionEvent1()
    {
        super("Utilisation d'un ActionEvent");
        Button b = new Button("action");
        b.addActionListener(this);
        add(BorderLayout.CENTER,b);pack();show();
    }
    public void actionPerformed( ActionEvent e )
    {
        setTitle("bouton cliqué !");
    }
}
```



Implémentation de l'interface ActionListener

On enregistre l'écouteur d'evt action auprès de l'objet source "b"

L'appui sur le bouton permet de changer le titre de la fenêtre



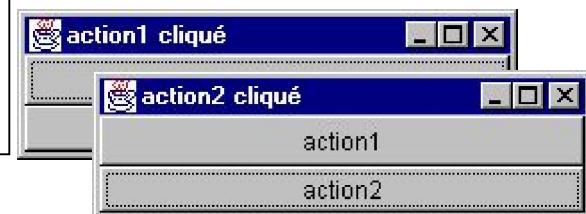
Evénem. avec 2 Boutons: Exemple 3

```
public class EssaiActionEvent2 extends Frame  
    implements ActionListener  
{ private Button b1,b2;  
    public static void main(String args[])  
    {EssaiActionEvent2 f= new EssaiActionEvent2();}  
    public EssaiActionEvent2() {  
        super("Utilisation d'un ActionEvent");  
        b1 = new Button("action1");  
        b2 = new Button("action2");  
        b1.addActionListener(this);  
        b2.addActionListener(this);  
        add(BorderLayout.CENTER,b1);  
        add(BorderLayout.SOUTH,b2);  
        pack();show(); }  
        public void actionPerformed( ActionEvent e ) {  
            if (e.getSource() == b1) setTitle("action1 cliqué");  
            if (e.getSource() == b2) setTitle("action2 cliqué");  
        } }
```



Les 2 boutons ont le même écouteur (la fenêtre)

e.getSource()
renvoie l'objet source de l'événement. On effectue un test sur les boutons (on compare les références)



Evénement avec classe anonyme (ex.4)

```
import java.awt.event.*;
import javax.swing.*;
public class Example extends JFrame {
    public Example() {    initUI();    }
    public final void initUI() {
        JPanel panel = new JPanel();
        getContentPane().add(panel);
        panel.setLayout(null);

        JButton quitButton = new JButton("Quit");
        quitButton.setBounds(50, 60, 80, 30);
        quitButton.addActionListener( new ActionListener() {
            public void actionPerformed(ActionEvent event) {
                System.exit(0);            }
        });
        panel.add(quitButton);
        setTitle("Quit button");
        setSize(300, 200);    setLocationRelativeTo(null);
        setDefaultCloseOperation(EXIT_ON_CLOSE);    }
    public static void main (String[] args) {
        Example ex = new Example();
        ex.setVisible(true);    } }
```

Utilisation de classe anonyme pour chaque bouton ou composant

Classes locales et anonymes

Il y a deux types de classes internes (inner classes):

- On peut déclarer une classe interne à l'intérieur du corps d'une méthode. Ce genre est connu comme une **classe locale**
- On peut déclarer une classe interne à l'intérieur du corps d'une méthode sans donner de nom à la classe. Ce genre est appelé **classe anonyme**.

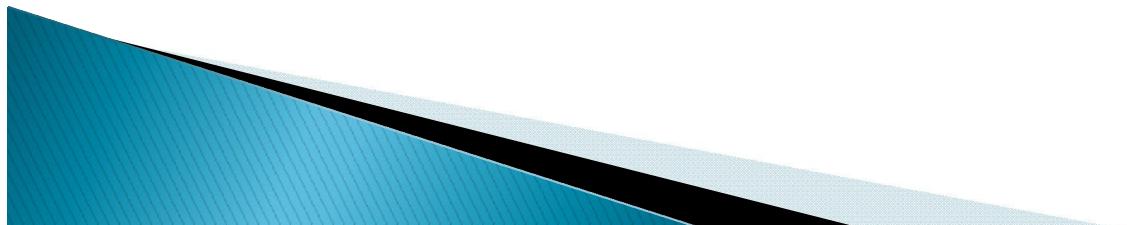
Evénement avec classe anonyme (exe.5)

```
public class Example extends JFrame {  
    public Example() {  
        initUI();  
    }  
    public final void initUI() {  
        JPanel panel = new JPanel();  
        getContentPane().add(panel);  
        panel.setLayout(null);  
        JButton bouton1 = new JButton("Couleur Fond");  
        JButton bouton2 = new JButton("Fermer");  
        bouton1.setBounds(20, 40, 40, 30);  
        bouton1.setBounds(70, 40, 40, 30);  
  
        bouton1.addActionListener( new ActionListener() {  
            public void actionPerformed(ActionEvent event) {  
                this.setBackground(...)  
            } );  
        bouton2.addActionListener( new ActionListener() {  
            public void actionPerformed(ActionEvent event) {  
                System.exit(0);  
            } );  
        panel.add(bouton1);    panel.add(bouton2);  
        setSize(300, 200);    setLocationRelativeTo(null);  
        setDefaultCloseOperation(EXIT_ON_CLOSE);    }  
}
```

Utilisation de classe anonyme pour chaque bouton ou composant

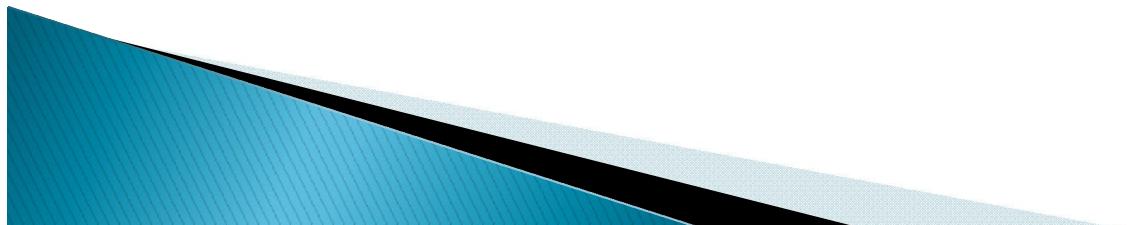
Types d'événements et écouteur

- ▶ **ActionEvent, ActionListener:**
 - Button, List, TextField, MenuItem, JButton, ...
 - public void actionPerformed(ActionEvent)
- ▶ **AdjustmentEvent, AdjustmentListener**
 - Scrollbar, ScrollPane, ...
 - public void adjustmentValueChanged(AdjustmentEvent)
- ▶ **ItemEvent, ItemListener**
 - Checkbox, CheckboxMenuItem, Choice, List
 - public void itemStateChanged(ItemEvent)



Types d'événements et écouteur

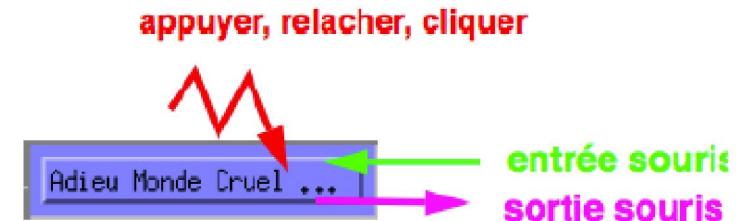
- ▶ **MouseEvent**
 - Souris
 - **MouseListener**
 - public void mouseDragged(MouseEvent)
 - public void mouseMoved(MouseEvent)
 - **MouseMotionListener**
 - public void mousePressed(MouseEvent)
 - public void mouseReleased(MouseEvent)
 - public void mouseEntered(MouseEvent)
 - public void mouseExited(MouseEvent)
 - public void mouseClicked(MouseEvent)
- ▶ **TextEvent, TextListener**
 - TextComponent et ses sous-classes
 - public void textValueChanged(TextEvent)



Evénement Souris

Exemple : MouseEvent

- Evénement : **MouseEvent**
- Listener : **MouseListener**
- Méthodes :
 - mouseClicked(MouseEvent)
 - mouseEntered(MouseEvent)
 - mouseExited(MouseEvent)
 - mousePressed(MouseEvent)
 - mouseReleased(MouseEvent)



- Listener : **MouseMotionListener**
- Méthodes :
 - mouseDragged(MouseEvent)
 - mouseMoved(MouseEvent)

Remarque

- toutes les méthodes **doivent** être implémentées
- car les **Listeners** sont des **interfaces** (au sens du langage Java)