Programmation évènementielle

nfp121

Les transparents qui suivent doivent beaucoup aux collègues qui mettent leurs cours à disposition sur Internet.

Merci à eux

JavaBeans et Listeners

Bean Pattern

```
public class MonBean {
    String prop1 ;
    public String getProp1() { return prop1; }
    public void setProp1(String s) { prop1 = s; }
    int prop2;
    public int getProp2() { return prop2; }
    public void setProp2(int i) { prop2 = i; }
    byte[] prop3;
    public byte[] getProp3() { return prop3; }
    public void setProp3(byte[] bytes) { prop3 = bytes; }
}
// si une propriété est read-only , pas de setXXX()
```

Principales caractéristiques des JavaBean

- Ensemble de propriétés exposées
  - attributs nommés (variables d'instance)
- Ensemble de méthodes que les autres composants sont autorisés à invoquer
  - par défault toutes les méthodes publiques
- Ensemble d'événements déclenchés
  - une façon d'informer les autres composants que quelque chose d'intéressant est survenu

Evénements

Les événements permettent de propager et de notifier les changements d'états

entre

un objet *source*

et

un ou plusieurs objets *listener*.

Implémentation

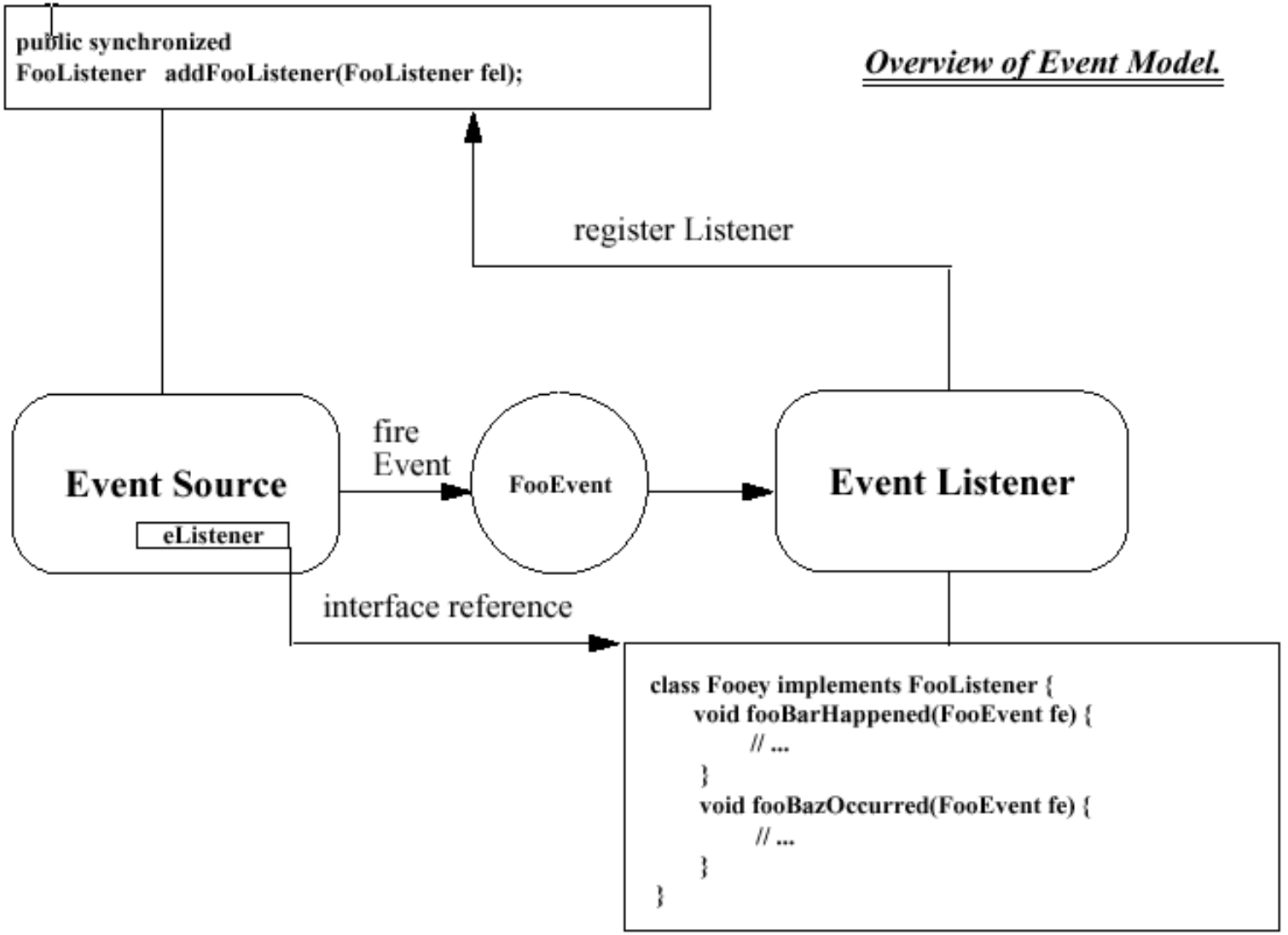
- Notifications d'événements
  - propagées des sources vers les listeners par invocation de méthodes Java

- *java.util.EventListener*
  - Groupe d'événements de la même catégorie
  - A chaque type d'événement correspond une méthode distincte pour la notification.

- Classe de listeners d'événements
  - implémente un ensemble donné d'interfaces *EventListener*.

- *java.util.EventObject*
  - l'état associé avec la notification d'un événement

Notification d'événements



Conventions

- Par convention, les classes qui décrivent l'état d'un événement ont des noms qui se terminent en "Event".

- Par convention, le nom de ces interfaces se termine en "Listener".

Enregistrement d'un Event Listener

- **Les classes qui déclenchent des événements doivent fournir des méthodes d'enregistrement et de "dés-enregistrement" des listeners.**

Design pattern :

```
synchronized
   public void
       add< ListenerType>(< ListenerType> listener);
synchronized
   public void
       remove< ListenerType>(< ListenerType> listener);
```

Exemple

```
public interface ModelChangedListener extends java.util.EventListener {
     void modelChanged(EventObject e);
}

public abstract class Model {
     private Vector listeners = new Vector(); // list of Listeners

     public synchronized void
     addModelChangedListener(ModelChangedListener mcl) {
          listeners.addElement(mcl);
     }

     public synchronized void
     removeModelChangedListener(ModelChangedListener mcl) {
          listeners.removeElement(mcl);
     }
```

Changement d'une Propriété

- *PropertyChangeListener* event listener
  - pour informer des mises-à-jour des propriétés.

```
public void addPropertyChangeListener
                    (PropertyChangeListener x);
public void removePropertyChangeListener
                    (PropertyChangeListener x);
```

  - **invocation de la méthode suivante pour chacun des listeners**

```
aListener.propertyChange(PropertyChangeEvent evt)
```

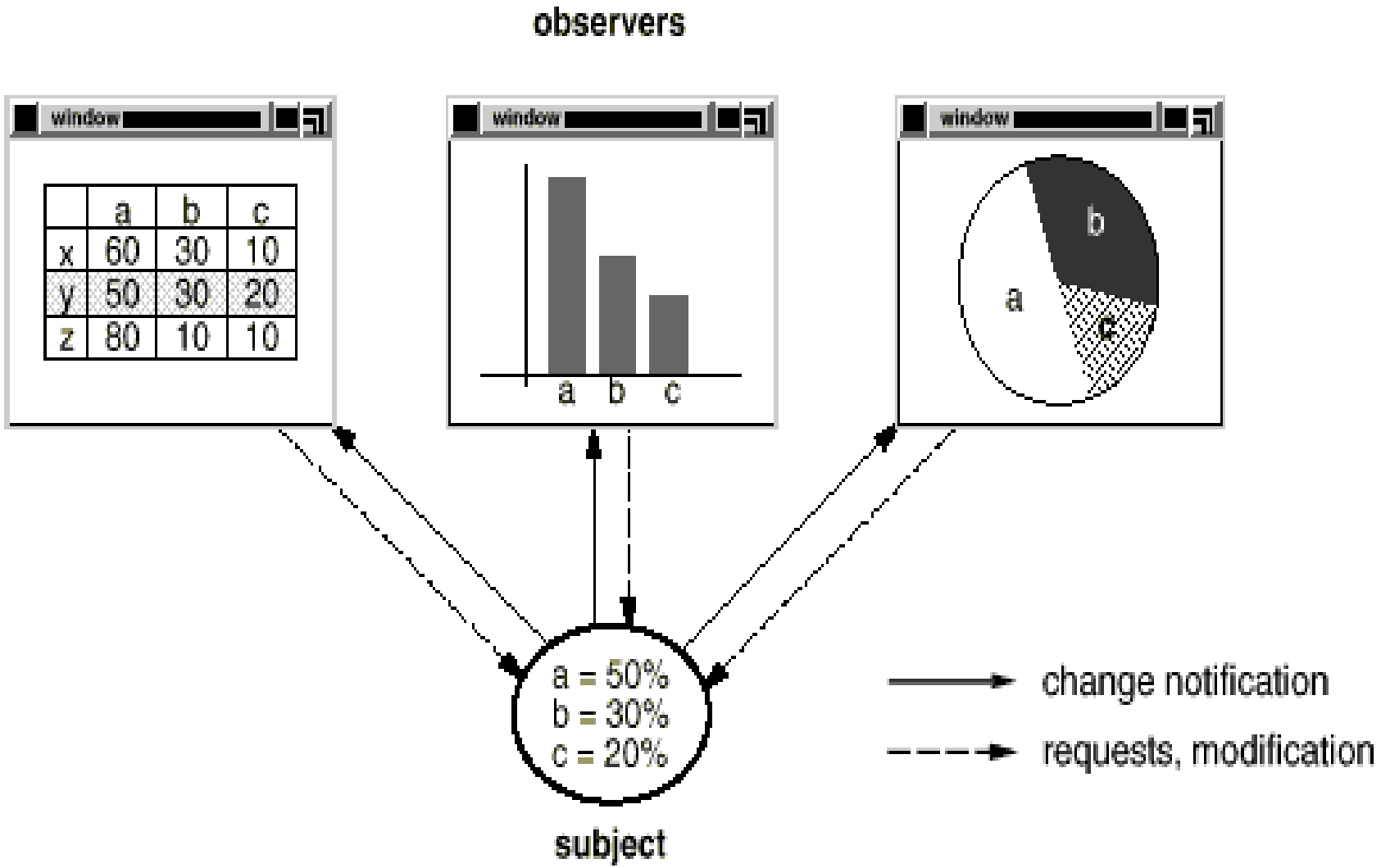
Déclencher l'évènement

```
int myProperty;

public int getMyProperty() { return myProperty; }

public void setMyProperty(int newValue) {
        int oldValue = myProperty;
        myProperty = newValue;
        // on suppose connaître le listener à prévenir
            listener.propertyChange(
                    new PropertyChangeEvent(this, "myProperty",
                                                  new Integer(oldValue),
                                                  new Integer(newValue))
                    );

}
```

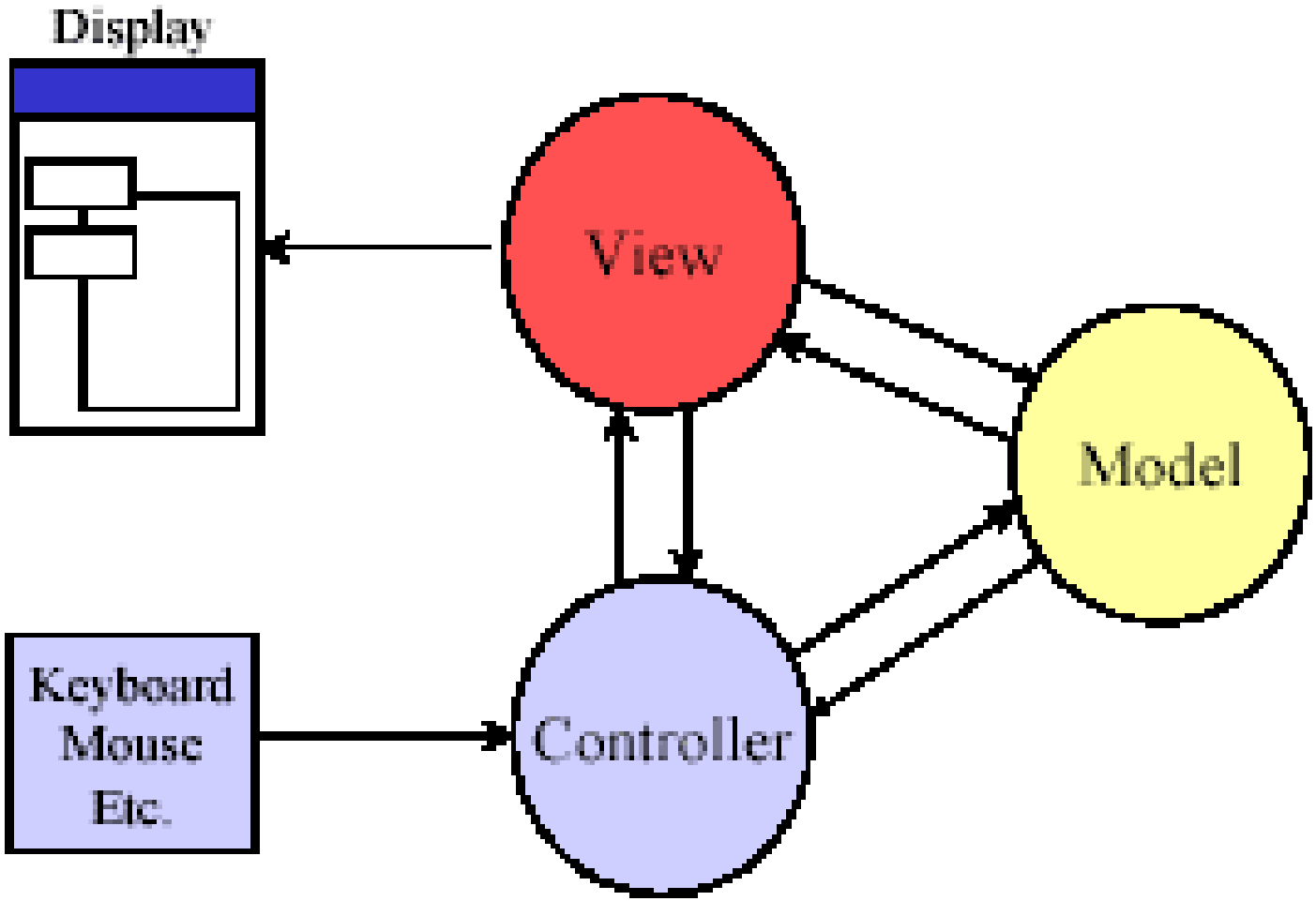
Enregistrer le listener

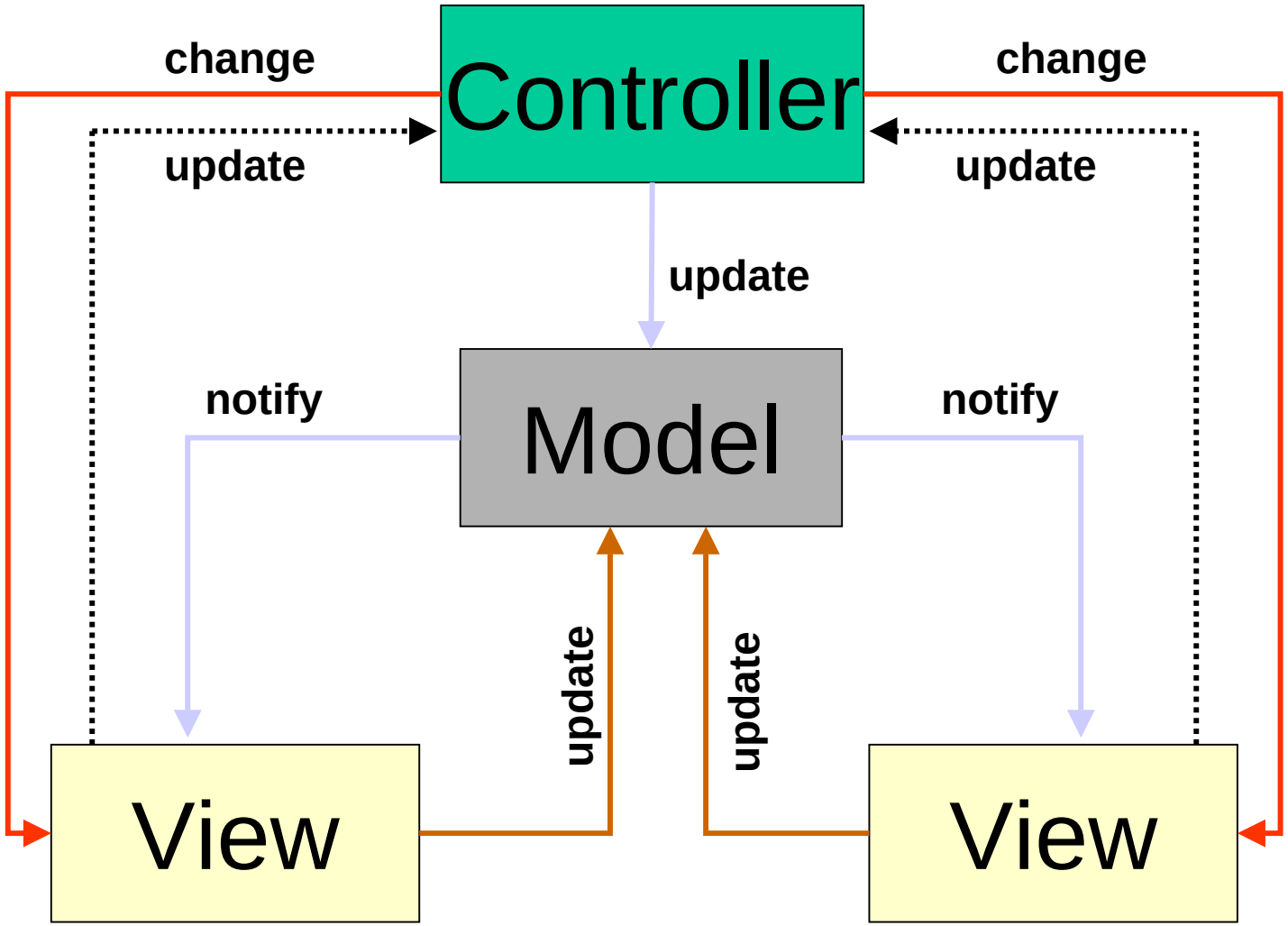
```
bean.addPropertyChangeListener(new MyPropertyChangeListener());


class MyPropertyChangeListener implements PropertyChangeListener {

        public void propertyChange(PropertyChangeEvent evt) {

                Object oldValue = evt.getOldValue();
                Object newValue = evt.getNewValue();
    }
}
```

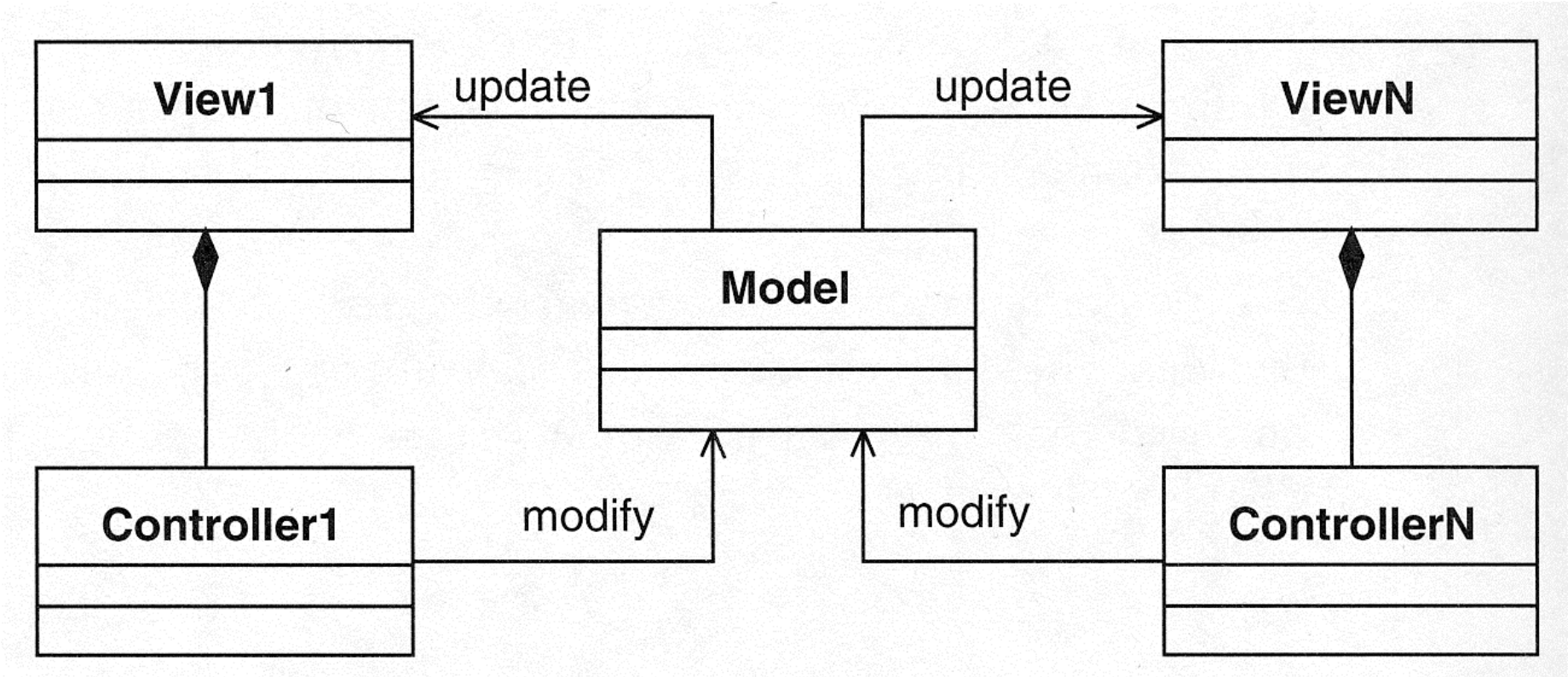
MVC: Observateurs et Modèles



MVC




change
Controller
change
update
update
update
notify
Model
notify
update
update
View
View


View1
update
update
ViewN
Model
Controller1
modify
modify
ControllerN

MVC

- Modèle: objet applicatif
- Vue: objets graphiques observant le modèle (output)
- Contrôleur: objets scannant les « inputs » et provoquant les changements du modèle…

Observer Class Diagram

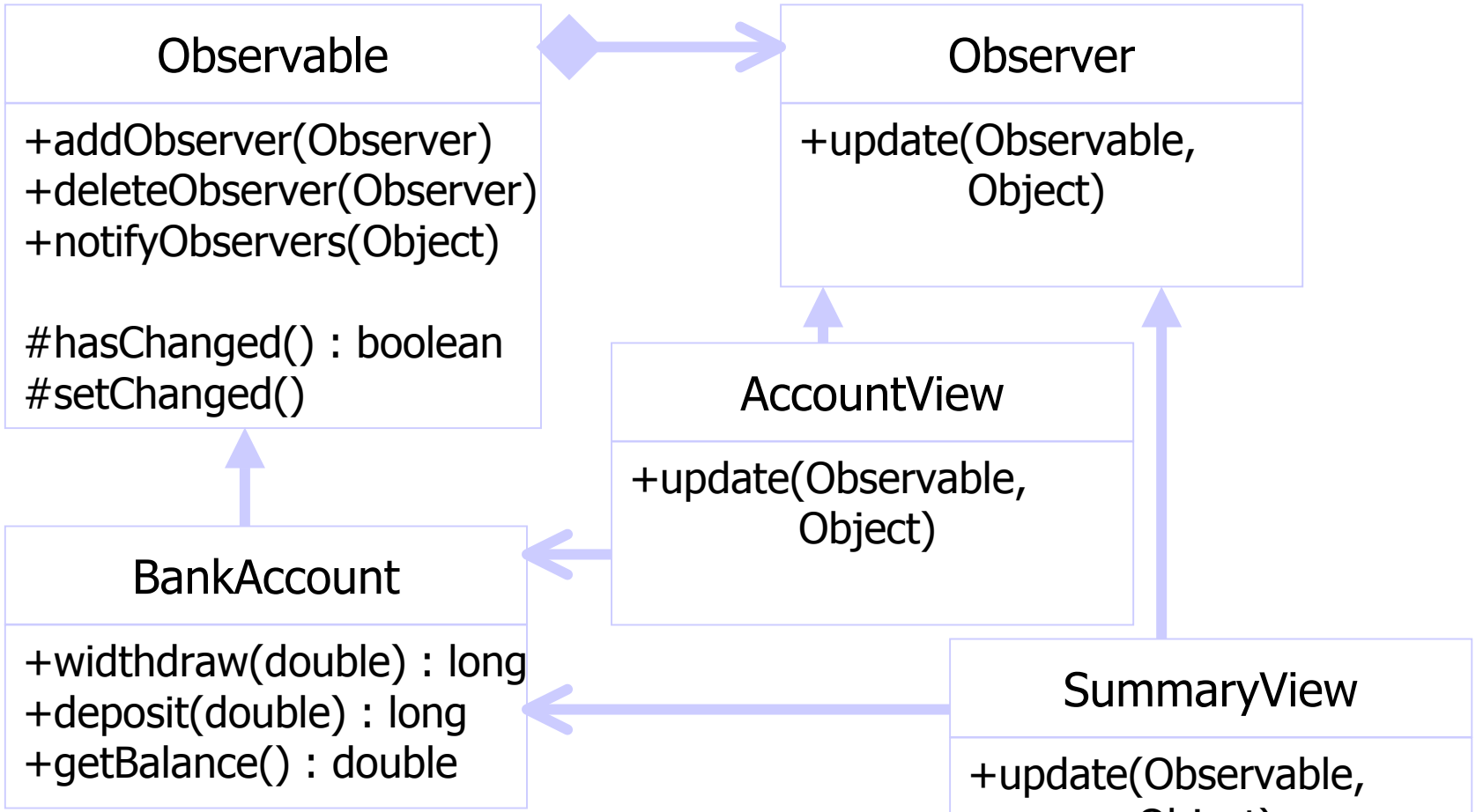
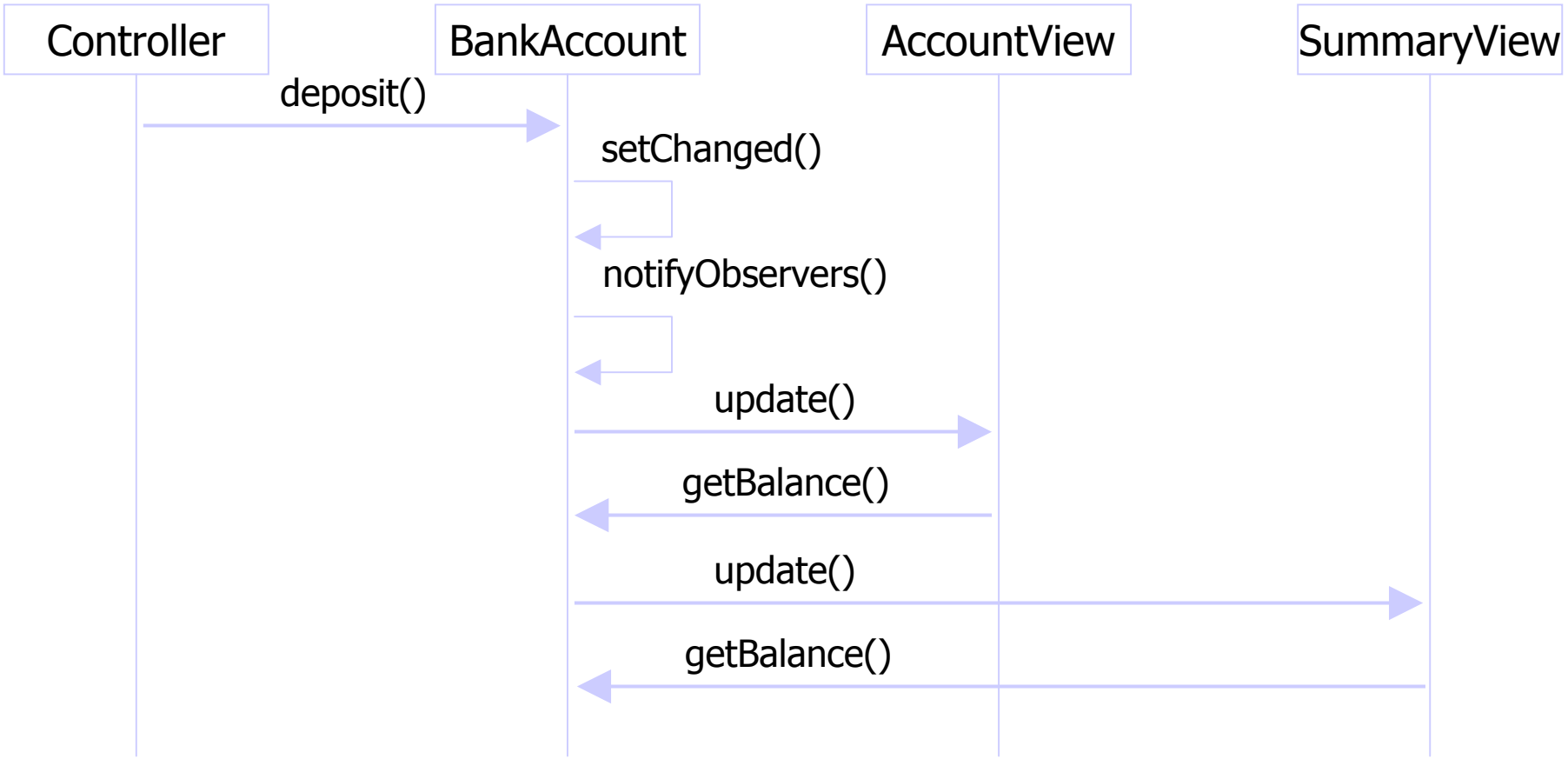
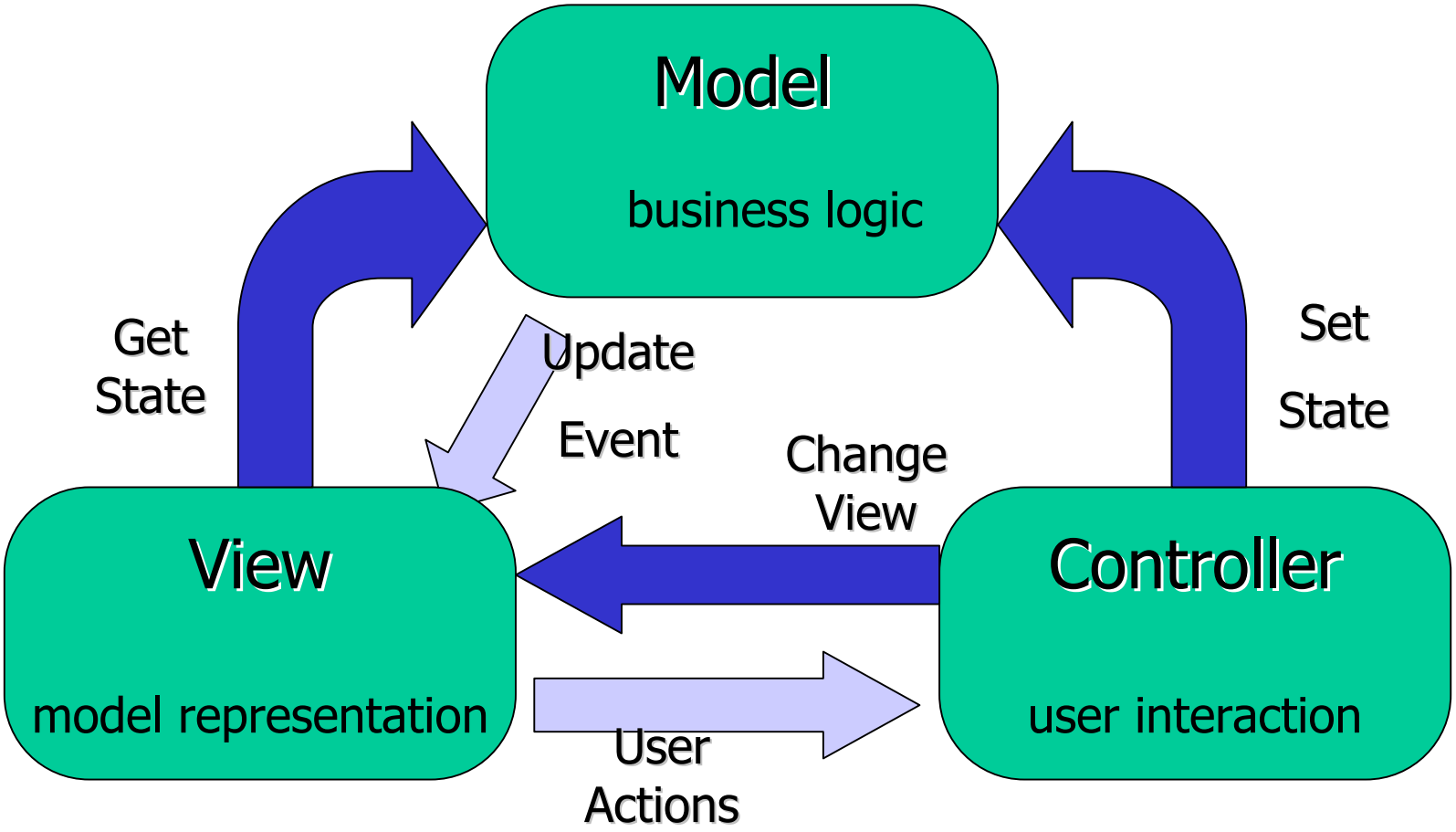


Diagramme de séquence



Architecture Diagram



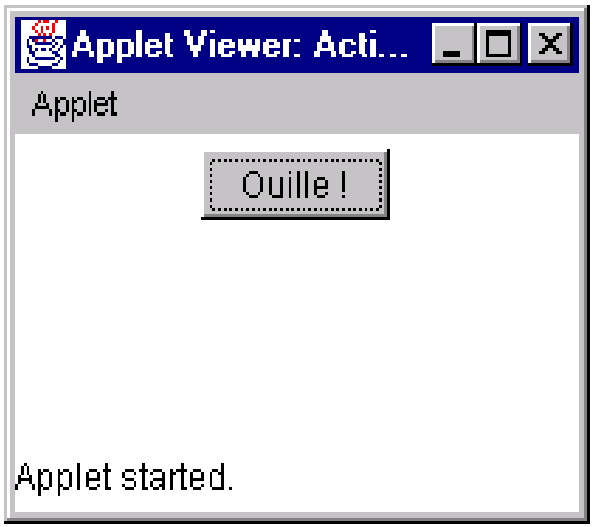
AWT

- Les interfaces homme machine comme exemple de programmation évènementielle

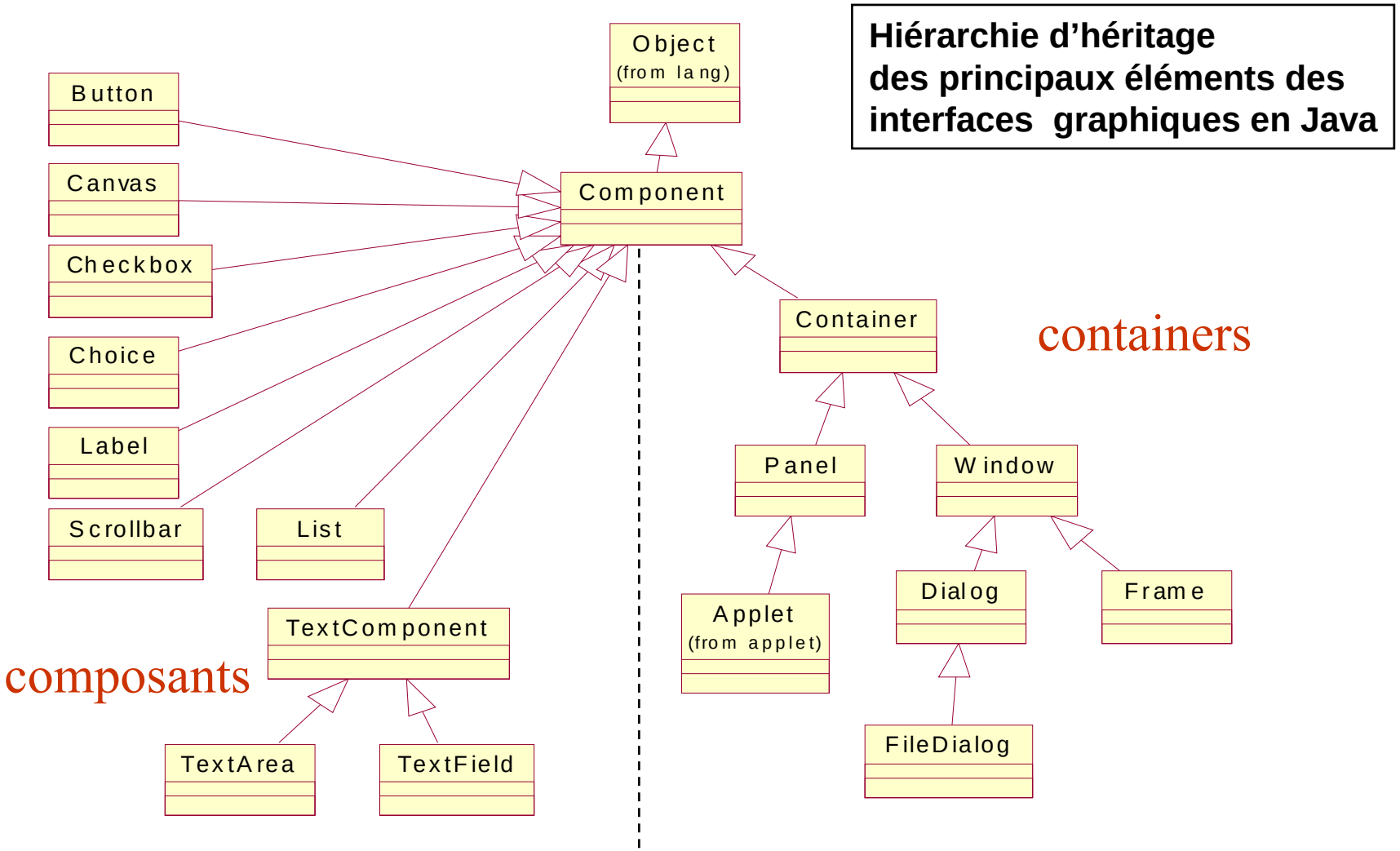
Exemple


ActionExemple.bat

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

public class ActionExemple extends Applet
   implements ActionListener
{
   Button b;
   public void init() {
      b = new Button("En avant !");
      b.addActionListener(this);
      add(b);
   }
   public void actionPerformed(ActionEvent e) {
      if (b.getLabel().equals("En avant !"))
         b.setLabel("Ouille !");
      else
         b.setLabel("En avant !");
   }
}
```



Conteneurs et composants



Conteneurs et composants

```
import java.awt.*;

public class EssaiFenetre1
{
  public static void main(String[] args)
  {
    Frame f =new Frame("Ma première fenêtre");
    Button b= new Button("coucou");
    f.add(b);
    f.pack();
    f.show();
  }
}
```

Création d'une fenêtre (un objet de la classe Frame) avec un titre

Création du bouton ayant pour label « coucou »

Ajout du bouton dans la fenêtre

On demande à la fenêtre de choisir la taille minimum avec pack() et de se rendre visible avec show()

Les événements graphiques

```
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();}
}
```

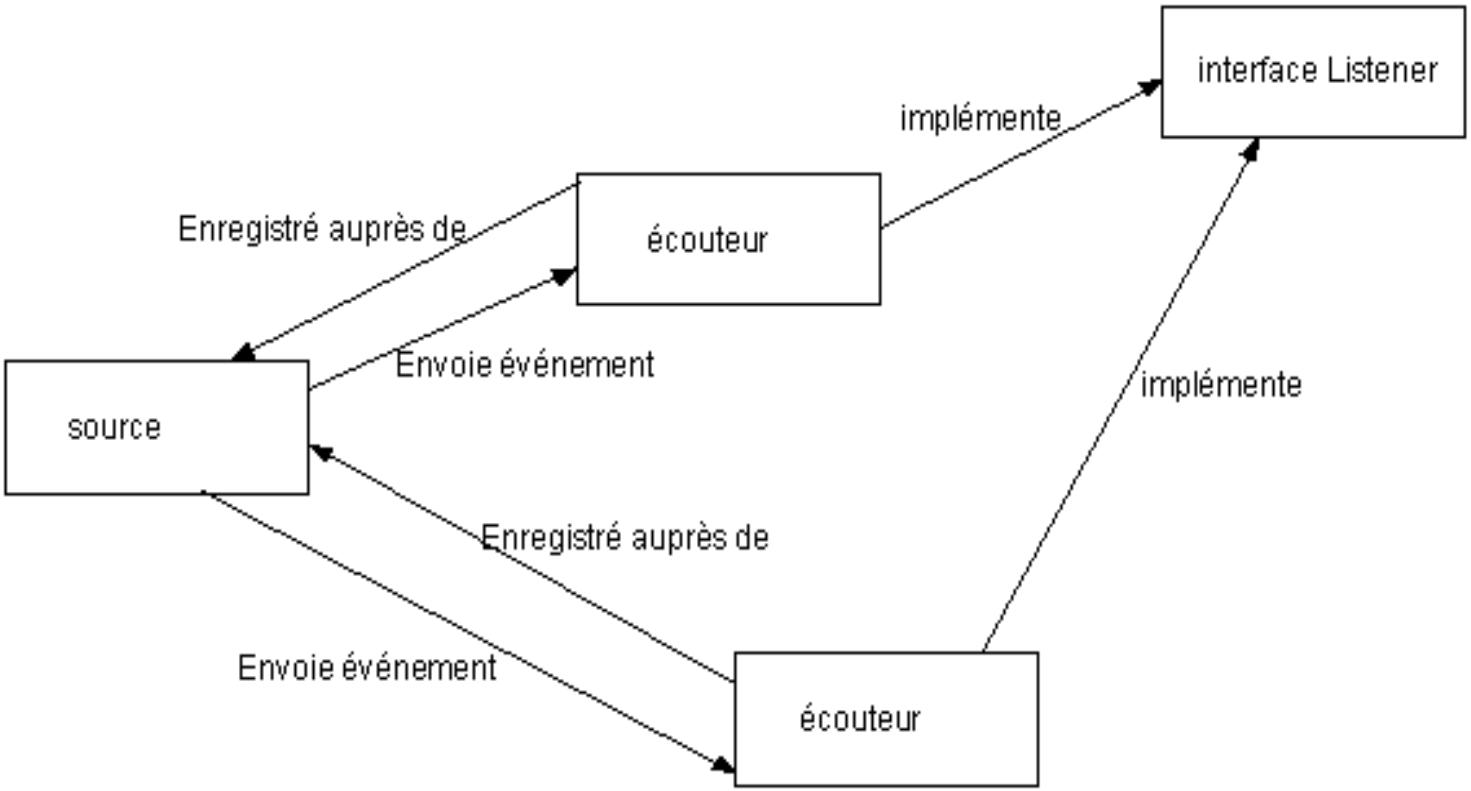
Les événements graphiques

Chaque composant de l'AWT est conçu pour être la source d'un ou plusieurs types d'événements particuliers.

- Cela se voit notamment grâce à la présence dans la classe de composant d'une méthode nommée add**XXX**Listener().

- L'objet événement envoyé aux écouteurs et passé en paramètres des fonctions correspondantes peut contenir des paramètres intéressants pour l'application.
  - Par exemple, getX() et getY() sur un MouseEvent retournent les coordonnées de la position du pointeur de la souris.

Les événements graphiques



Catégories d'événements graphiques

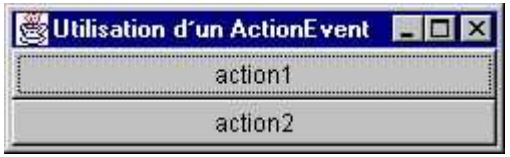
Catégorie	Nom de l'interface	Méthodes
Action	ActionListener	actionPerformed (ActionEvent)
Item	ItemListener	itemStateChanged (ItemEvent)
Mouse	MouseMotionListener	mouseDragged (MouseEvent) mouseMoved (MouseEvent)
Mouse	MouseListener	mousePressed (MouseEvent) mouseReleased (MouseEvent) mouseEntered (MouseEvent) (MouseEvent) mouseExited mouseClicked
Key	KeyListener	keyPressed (KeyEvent) keyReleased (KeyEvent) keyTyped (KeyEvent)
Focus	FocusListener	focusGained (FocusEvent) focusLost (FocusEvent)

Catégories d'événements graphiques

Adjustment	AdjustmentListener	adjustmentValueChanged (AdjustmentEvent)
Component	ComponentListener	componentMoved (ComponentEvent) componentHidden (ComponentEvent) componentResized (ComponentEvent) componentShown (ComponentEvent)
Window	WindowListener	windowClosing (WindowEvent) windowOpened (WindowEvent) windowIconified (WindowEvent) windowDeiconified (WindowEvent) windowClosed (WindowEvent) windowActivated (WindowEvent) windowDeactivated (WindowEvent)
Container	ContainerListener	componentAdded (ContainerEvent) componentRemoved (ContainerEvent)
Text	TextListener	textValueChanged (TextEvent)

Catégories d'événements graphiques

```
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)

Catégories d'événements graphiques

```
import java.awt.*; import java.awt.event.*;
public class WinEvt extends Frame
      implements WindowListener
{
 public static void main(String[] args) {
  WinEvt f= new WinEvt();}

  public WinEvt() {
    super("Cette fenêtre se ferme");
    addWindowListener(this);
    pack();show();}

   public void windowOpened(WindowEvent e){}
   public void windowClosing(WindowEvent e)
   {System.exit(0);}
   public void windowClosed(WindowEvent e){}
   public void windowIconified(WindowEvent e){}
   public void windowDeiconified(WindowEvent e){}
   public void windowActivated(WindowEvent e){}
   public void windowDeactivated(WindowEvent e){} }
```

Implémenter cette interface impose l'implémentation de bcp de méthodes

La fenêtre est son propre écouteur

WindowClosing() est appelé lorsque l'on clique sur la croix de la fenêtre

"System.exit(0)" permet de quitter une application java

Les adapteurs

Solution en implémentant l'interface

```
class Terminator implements WindowListener
{
   public void windowClosing (WindowEvent e) {System.exit(0);}
   public void windowClosed (WindowEvent e) {}
   public void windowIconified (WindowEvent e) {}
   public void windowOpened (WindowEvent e) {}
   public void windowDeiconified (WindowEvent e) {}
   public void windowActivated (WindowEvent e) {}
   public void windowDeactivated (WindowEvent e) {}
}
```

Solution en utilisant un WindowAdapter

```
class Terminator extends WindowAdapter
{
  public void windowClosing (WindowEvent e) {System.exit(0);}
}
```

Les adapteurs

- Il existe 7 classes d'adapteurs (autant que d'interfaces d'écouteurs possédant plus d'une méthode) :
  - **ComponentAdapter**
  - **ContainerAdapter**
  - **FocusAdapter**
  - **KeyAdapter**
  - **MouseAdapter**
  - **MouseMotionAdapter**
  - **WindowAdapter**

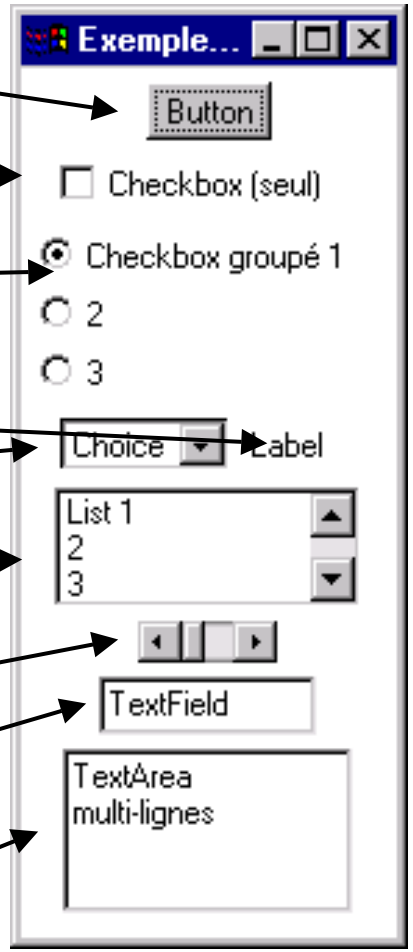
Classe anonyme

- En pratique, et notamment avec la classe WindowAdapter, on utilise très souvent une classe anonyme

```
Frame f = new Frame("Machin")
f.addWindowListener(new WindowAdapter()
                    {
                        public void windowClosing(WindowEvent e)
                        {
                            System.exit(0);
                        }
                    });
```

Les composants graphiques AWT

- Button
- Canvas (zone de dessin)
- Checkbox (case à cocher)
- CheckboxGroup
- Label
- Choice (Sélecteur)
- List
- Scrollbar (barre de défilement)
- TextField (zone de saisie d'1 ligne)
- TextArea (zone de saisie multilignes)



Création de composants

```
Choice c = new Choice();
c.addItem("First");
c.addItem("Second");
...
c.addItemListener (...);
```

```
Label l = new Label ("Bonjour !");
add(l);
```

```
List l =new List (4, false);
l.add("item1");
```

nombre d'items visibles (ici 4 éléments seront visible en même temps)

sélections multiples possibles ou non. Ici, avec la valeur false, non possible

Composants texte

```
TextField f = new TextField ("Une ligne seulement ...", 30);
add(f);
```

Texte par défaut mis dans le TextField

Nombre de caractères visibles dans le TextField

```
TextArea t = new TextArea ("Hello !", 4, 30,TextArea.SCROLLBARS_BOTH);
add(t);
```

Texte par défaut mis dans le TextArea

Nombre de lignes

Nombre de colonnes (en nbre de caractères)

Valeur constante précisant la présence ou l'absence de « scrollbar »

Gestionnaire de présentation

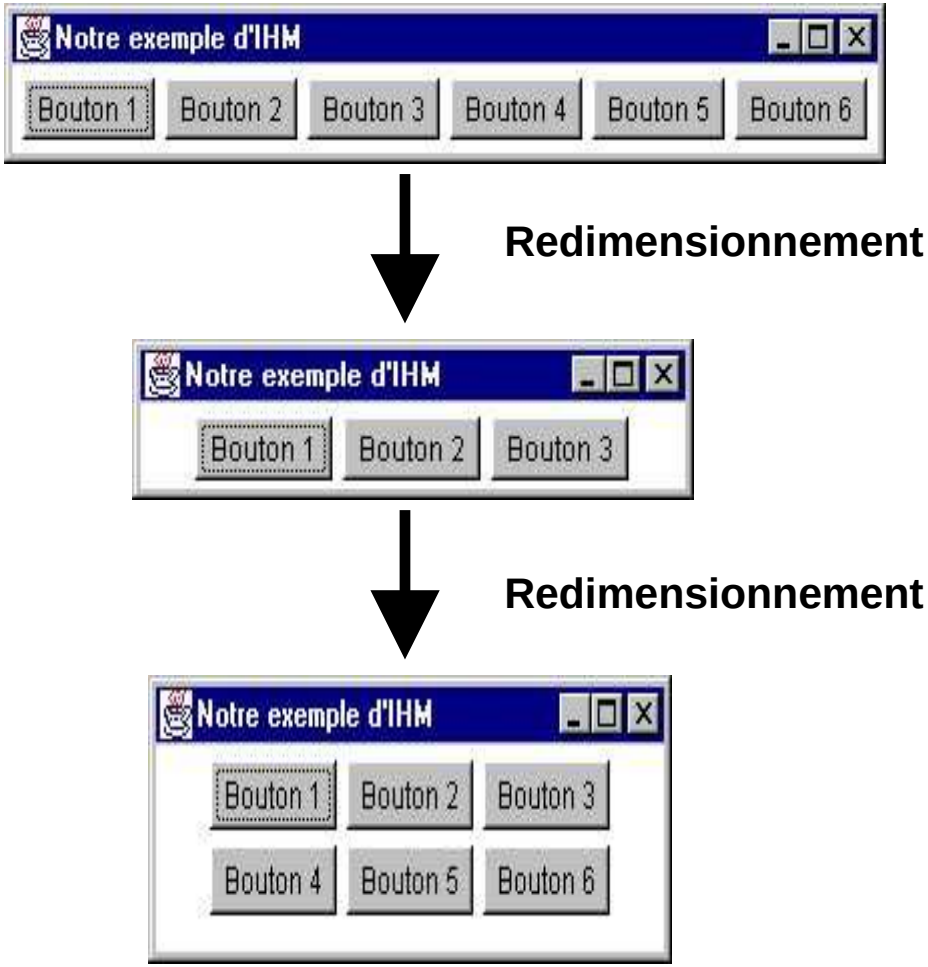
- A chaque conteneur est associé un gestionnaire de présentation (*layout manager*)

- Les principaux gestionnaire de présentation de l'AWT sont : **FlowLayout**, **BorderLayout, GridLayout, CardLayout, GridBagLayout**

- Tout conteneur possède un gestionnaire de présentation par défaut.
  - Tout instance de ***Container*** référence une instance de ***LayoutManager***.
  - Il est possible d'en changer grâce à la méthode **setLayout()**.

- Les gestionnaires de présentation par défaut sont :
  - Le **BorderLayout** pour **Window** et ses descendants (**Frame**, **Dialog**, …)
  - Le **FlowLayout** pour **Panel** et ses descendants (**Applet**, etc.)

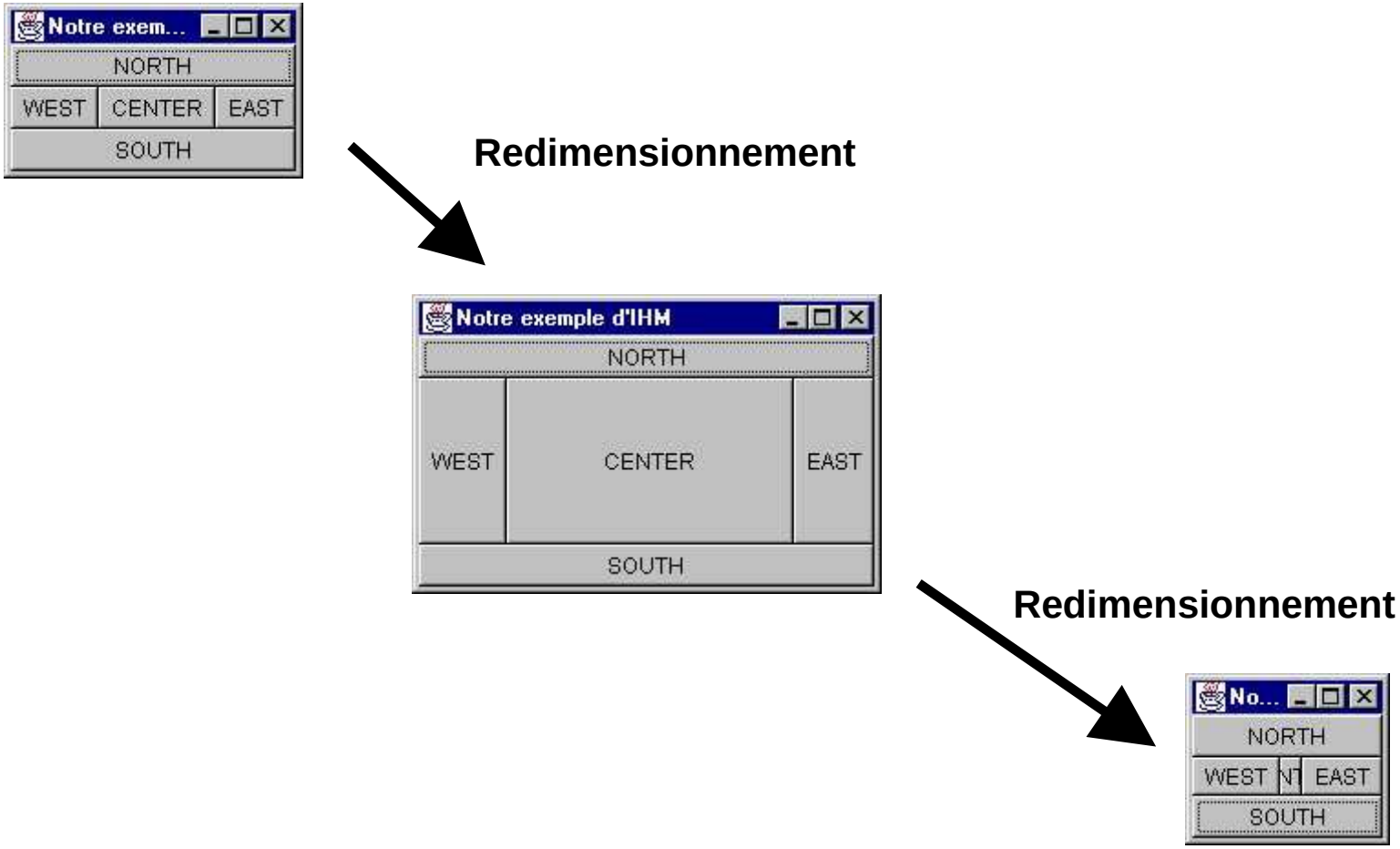
FlowLayout


Notre exemple d'IHM
Bouton 1
Bouton 2
Bouton 3
Bouton 4
Bouton 5
Bouton 6
Redimensionnement
Notre exemple d'IHM
Bouton 1
Bouton 2
Bouton 3
Redimensionnement
Notre exemple d'IHM
Bouton 1
Bouton 2
Bouton 3
Bouton 4
Bouton 5
Bouton 6

BorderLayout

```
import java.awt.*;

public class EssaiBorderLayout extends Frame
{
  private Button b1,b2,b3,b4, b5;
   public EssaiBorderLayout() {
    setLayout(new BorderLayout());
    b1 = new Button ("Nord"); b2 = new Button ("Sud");
    b3 = new Button ("Est"); b4 = new Button ("Ouest");
    b5 = new Button ("Centre");
    this.add(b1, BorderLayout.NORTH);
    this.add(b2 , BorderLayout.SOUTH);
    this.add(b3, BorderLayout.EAST);
    this.add(b4, BorderLayout.WEST);
    this.add(b5, BorderLayout.CENTER);
  }
public static void main (String args []) {
    EssaiBorderLayout essai = new EssaiBorderLayout();
    essai.pack (); essai.setVisible(true) ;
  }}
```

BorderLayout



Swing?

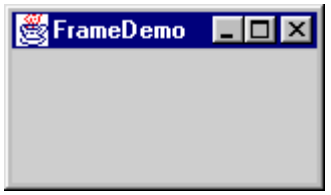
- # Abstract Window Toolkit (AWT)

- 100% Pur Java
  - JDK 1.1.2 au moins

Conteneurs de premier niveau

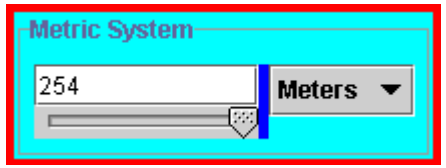
JApplet

JDialog



JFrame

Conteneurs généraux

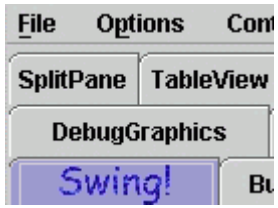


JPanel

JSplitPane

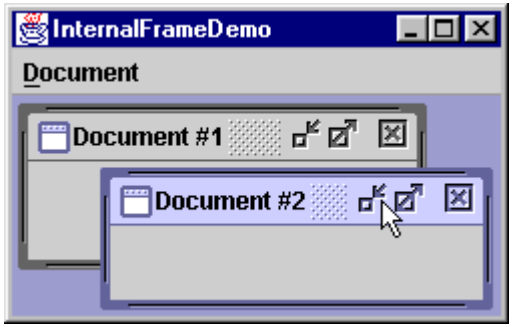
JScrollPane

JToolBar

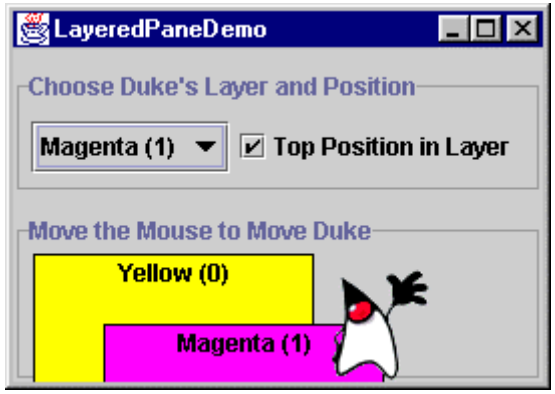


JTabbedPane

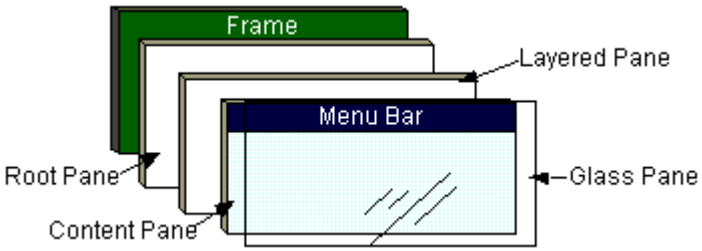
Conteneurs spécialisés



JInternalFrame

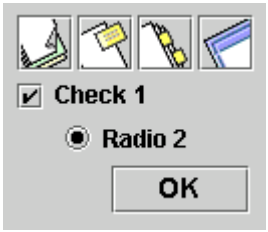


JLayeredPane

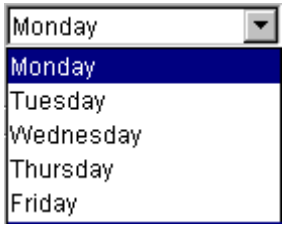


JRootPane

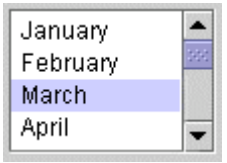
Contrôles de base



JButton

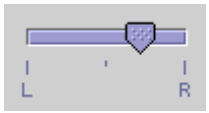


JComboBox



JList

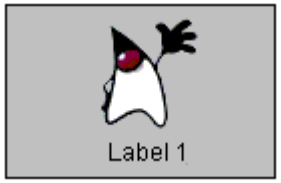
JMenu



JSlider

JTextfield

*Affiche*urs non éditables

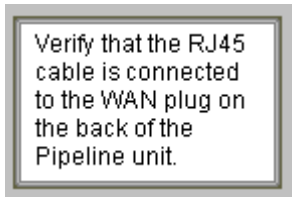


JLabel

JProgressBar

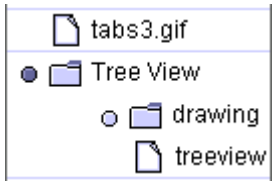
JToolTip

Editeurs d'informations structurées ou formatées

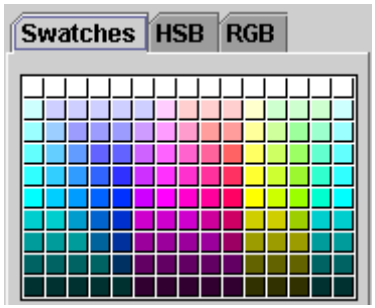


JTextArea

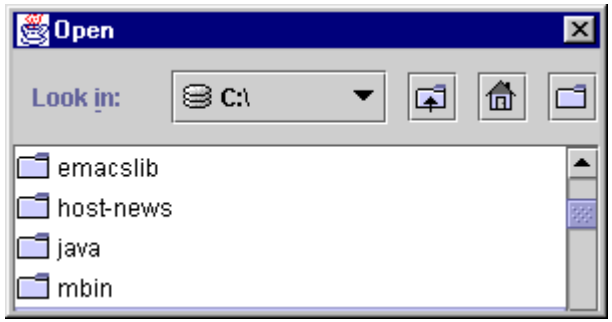
JTable



JTree

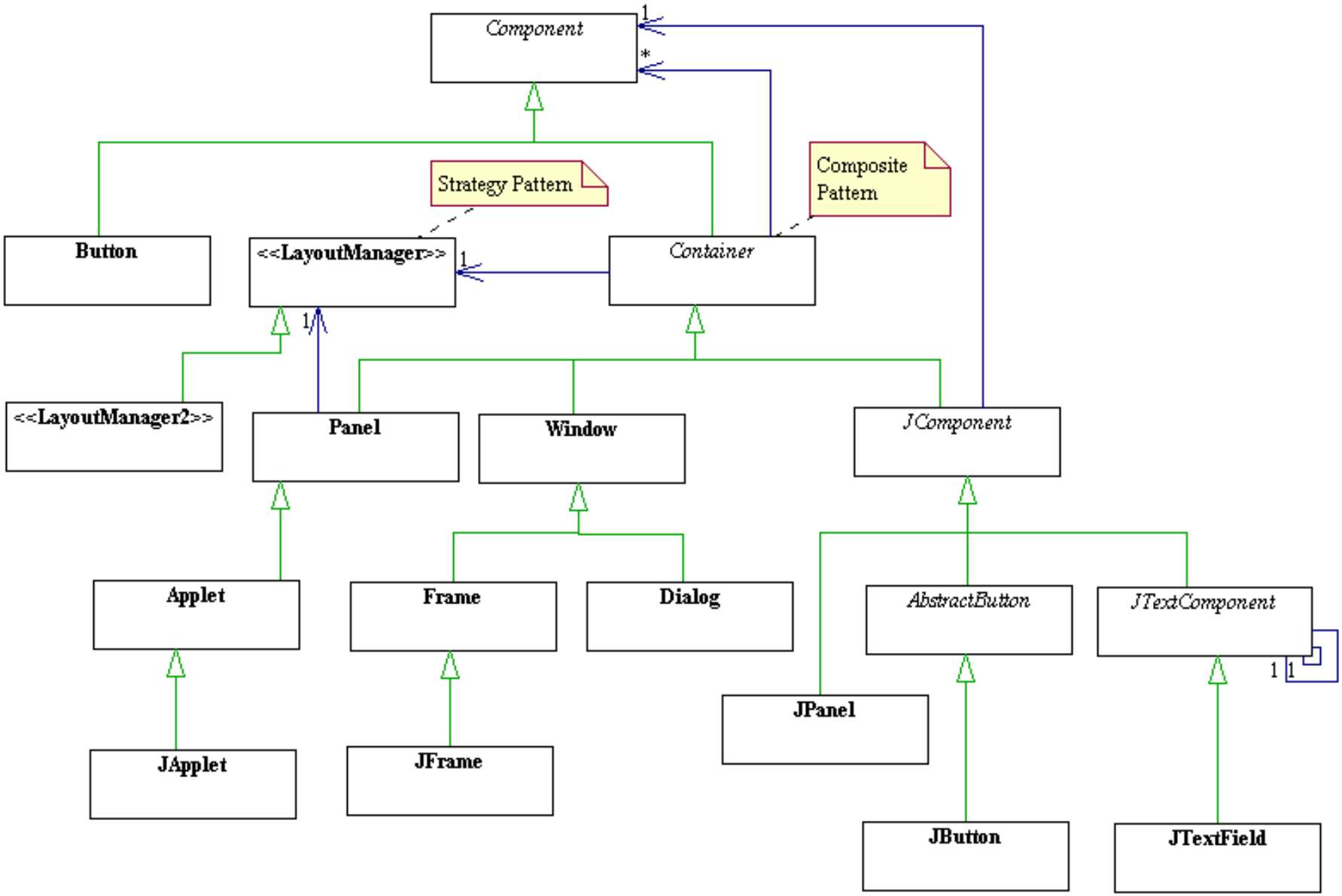


JColorChooser



JFileChooser

Classes Swing



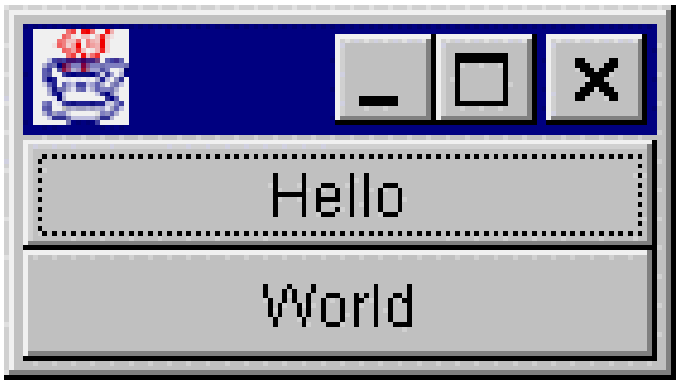
Swing par rapport à AWT

de AWT à Swing

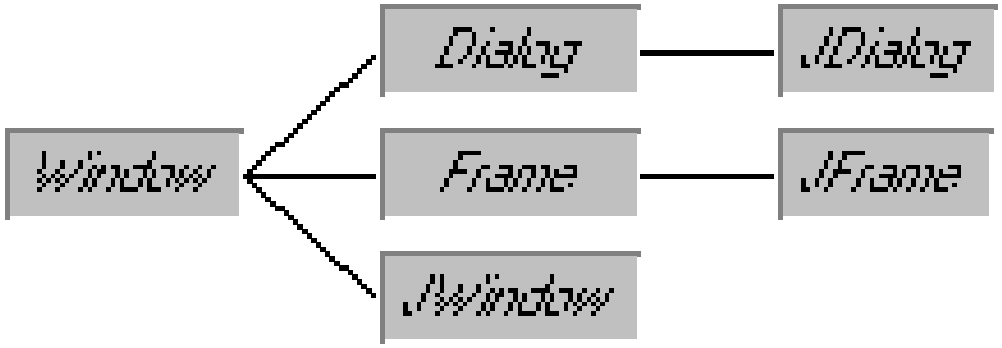
- Enn général ajouter J devant le nom
  - ***Button*** → ***JButton***, ***Applet*** → ***JApplet***, …
- Les composants swing sont des Containers
- Java 1.1 Event Model Only

Swing vs. AWT 1.1

```
class MyActionListener implements ActionListener {
  public void actionPerformed (ActionEvent e) {
    System.out.println (e.getActionCommand());
  }
}
...
ActionListener al = new MyActionListener();
Button b1 = new Button ("Hello");
b1.addActionListener (al);
add (b1, BorderLayout.NORTH);

JButton b2 = new JButton ("World");
b2.addActionListener (al);
add (b2, BorderLayout.SOUTH);
```



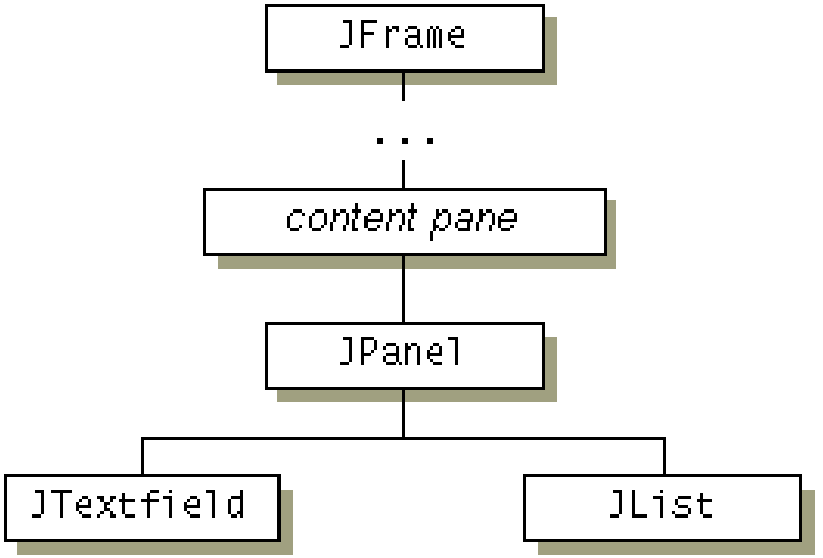
Top Level Components



- Sont sous-classes de *Window*, pas de *JComponent*
- Ne sont pas eux des composants légers
- Les composants s'ajoutent au 'content pane'

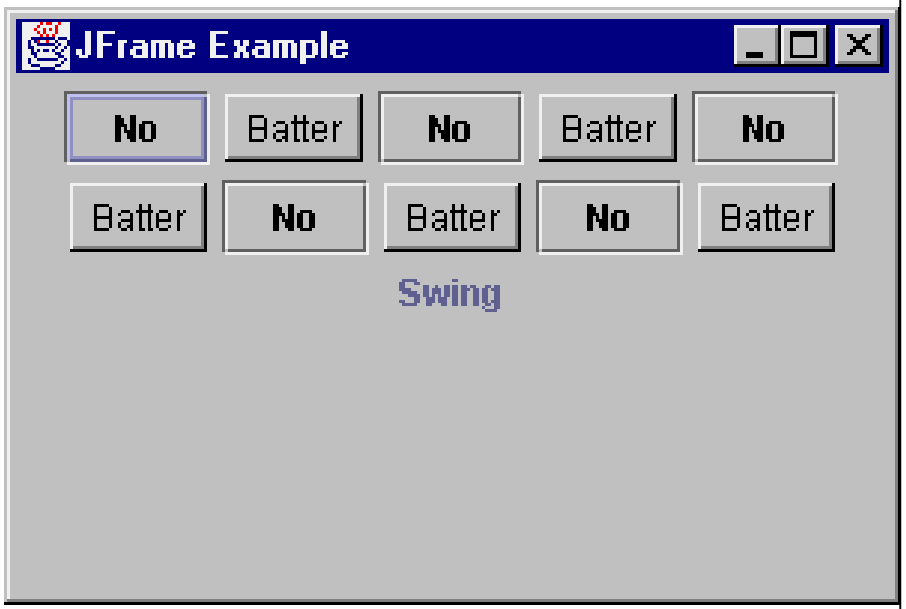
aJFrame.getContentPane()

- On ajoute plus les composants au composant
  - *aFrame.add (new Button (“Help”));*
- mais à son ‘contentPane’
  - ***aJFrame.getContentPane().add (…);***
  - Layout manager too - default ***BorderLayout***



JFrame Example

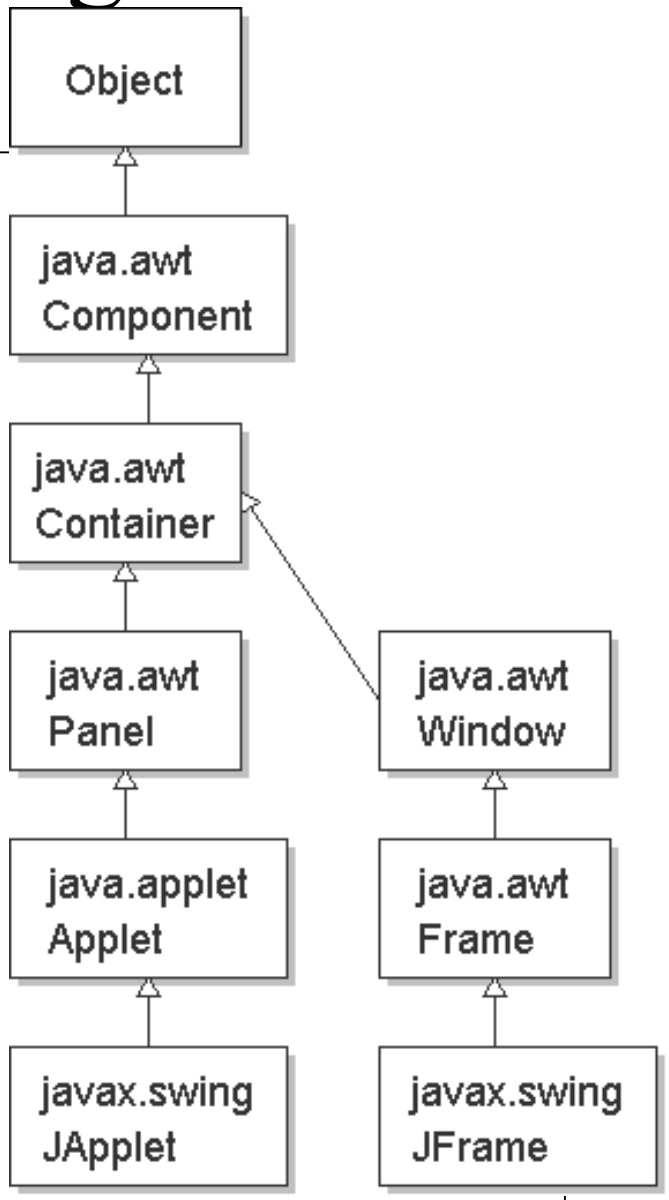
```
public class FrameTester {
  public static void main (String args[]) {
    JFrame f = new JFrame ("JFrame Example");
    Container c = f.getContentPane();
    c.setLayout (new FlowLayout());
    for (int i = 0; i < 5; i++) {
      c.add (new JButton ("No"));
      c.add (new Button ("Batter"));
    }
    c.add (new JLabel ("Swing"));
    f.setSize (300, 200);
    f.show();
  }
}
```



Applet et Swing

- implementation
  - Container de haut niveau
    - comme `JFrame`
  - Plutôt comme un JPanel
  - class `javax.swing.JApplet`

`java.lang.Object`
- `java.awt.Component`
  - `java.awt.Container`
    - `java.awt.Panel`
      - `java.applet.Applet`
        - » `javax.swing.`**`JApplet`**

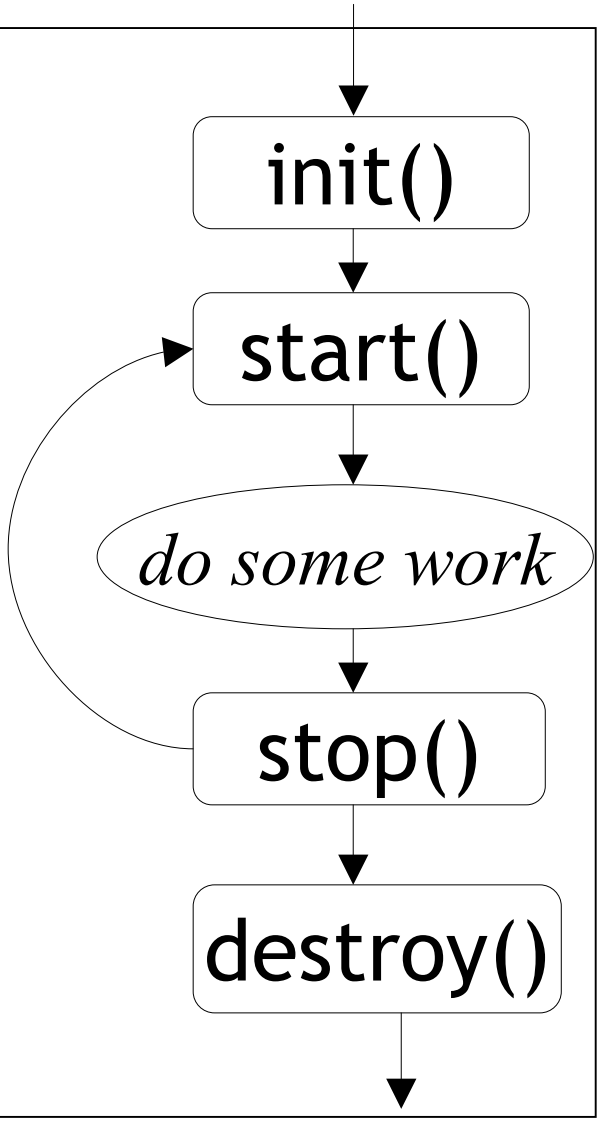


Cycle de vie d'une Applet

- Le navigateur rencontre une balise Applet
  - Il crée l'objet correspondant
  - Il appelle 'init'
  - puis 'start'
- Le navigateur quitte cette page
  - Il appelle 'stop'
- Il revient sur cette page
  - Il ré-appelle 'start'

...

- Le navigateur est fermé
  - Appel de destroy



javax.swing.JApplet

Differences entre `JApplet` et `JFrame`:

- Pas de method `main`; `init`/`start` sont appelés par le navigateur
- Pas d'appel de `setVisible(true)` (fait implicitement)
- Pasde `setSize(...)` / `pack()`; la taille est définie par la page web
- Pas de `setTitle(String)`; le titre est celui de la page web

Un exemple simple

- Une applet peut être vue comme une surface de dessin ou comme un container pour d'autres composants

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

public class HelloWorldApplet extends JApplet {
    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.drawString("Hello World!",
    }
}
```

page Web avec une applet

```
<HTML>
<HEAD>
<TITLE>My Applet Page</TITLE>
</HEAD>

<BODY>
<APPLET code="mypackage/MyApplet.class"
  width=400 height=300> </APPLET>
</BODY>
</HTML>
```

restrictions `JApplet`

- Pas d'accès au disque client
- Pas d'autres accès distants qu'au serveur web
- Pas possible d'exécuter d'autres programmes
- Très peu d'accès aux infos sur le système client
- Les fenêtres crées par l'applet sont marquées d'un warning