NFP121, Cnam/Paris
Cours 4-2
MVC et swing
programmation événementielle, suite

jean-michel Douin, douin au cnam point fr

version : 8 Octobre 2015

Notes de cours

Sommaire

- **L'API Swing**
  - Un descriptif
  - Exemples
    - Démonstration
  - Usage des patrons
    - Composite, stratégie, fabrique …
  - Évènements, MVC
    - Usage des patrons
      - Observateur, Commande
    - Mise en œuvre du Modèle, Vue, Contrôleur
    - Exemples
      - Démonstration
    - Le modèle devient un java*Bean*
    - MVC d'un composant swing

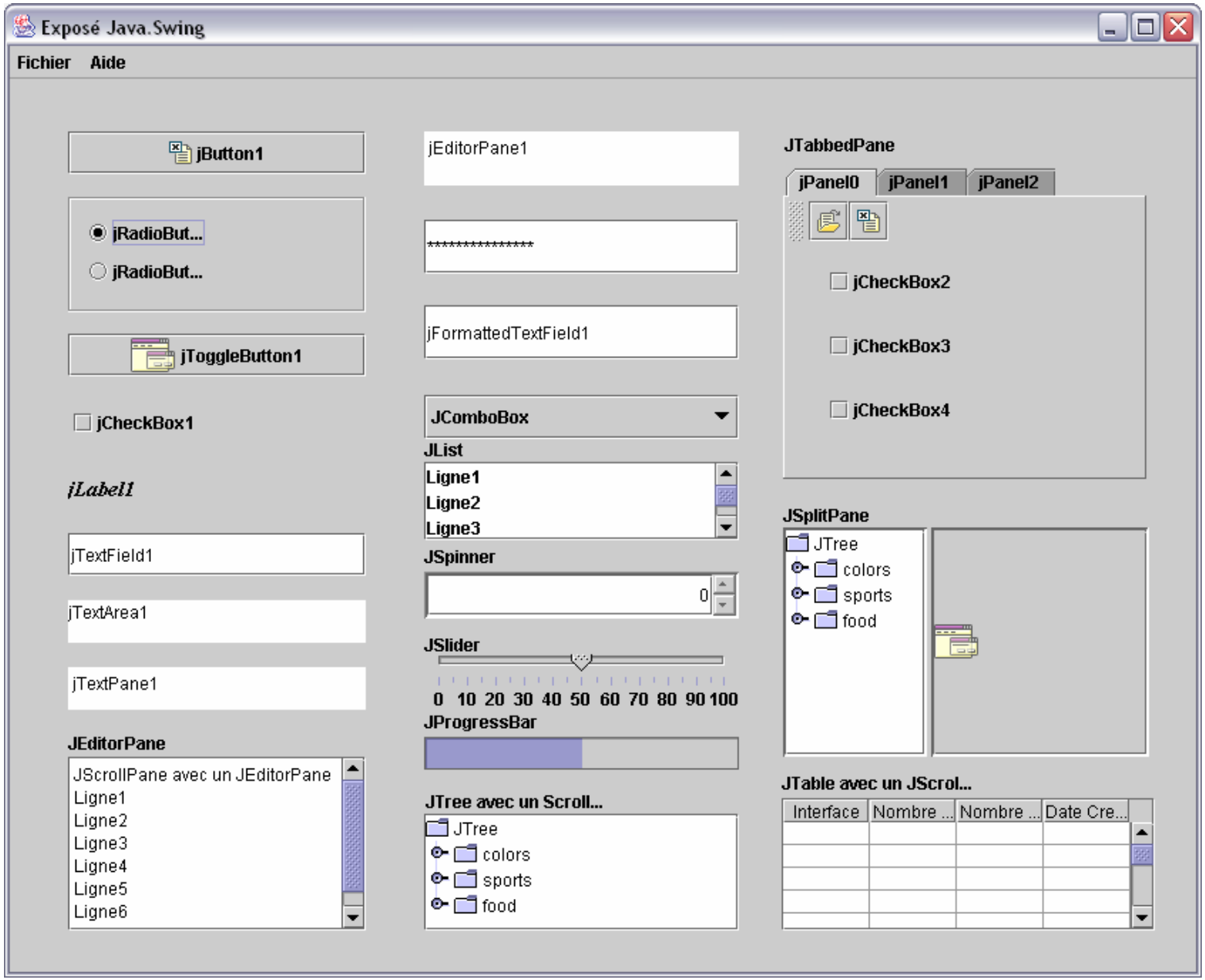
Principale bibliographie utilisée

- Swing, recherche google
  - http://notes.corewebprogramming.com/student/Basic-Swing.pdf
  - mathinfo.ens.univ-reims.fr/.../**ppt/Swing**_et_les_Applications_Graphiques.**ppt**

L'API Swing

- **Présentation « déclarative »**
- **Le package javax.swing**
- **Les composants graphiques**
- **Quelques exemples en « direct »**

Swing en images, javax.swing



Le paquetage swing bien nommé

- javax.swing
  - **Composants élémentaires**
  - **Menus, Barre d'outils et ToolTips**
  - **Containers**
- javax.swing.beaninfo
- javax.swing.beaninfo.images
- javax.swing.border
- javax.swing.colorchooser
- javax.swing.event
- javax.swing.filechooser
- javax.swing.plaf
- javax.swing.plaf.basic
- javax.swing.plaf.metal
- javax.swing.plaf.multi
- javax.swing.table
- javax.swing.text
- javax.swing.text.html
- javax.swing.tree
- javax.swing.undo

Hiérarchie de classes 1/2, tout est JComponent

- **Container**
  - **JComponent**
    - **AbstractButton**
      - **JButton**
      - **JMenuItem**
        - » JCheckBoxMenuItem
        - » JMenu
        - » JRadioButtonMenuItem
      - **JToggleButton**
        - » JCheckBox
        - » JRadioButton

Hiérarchie 2/2, JComponent suite

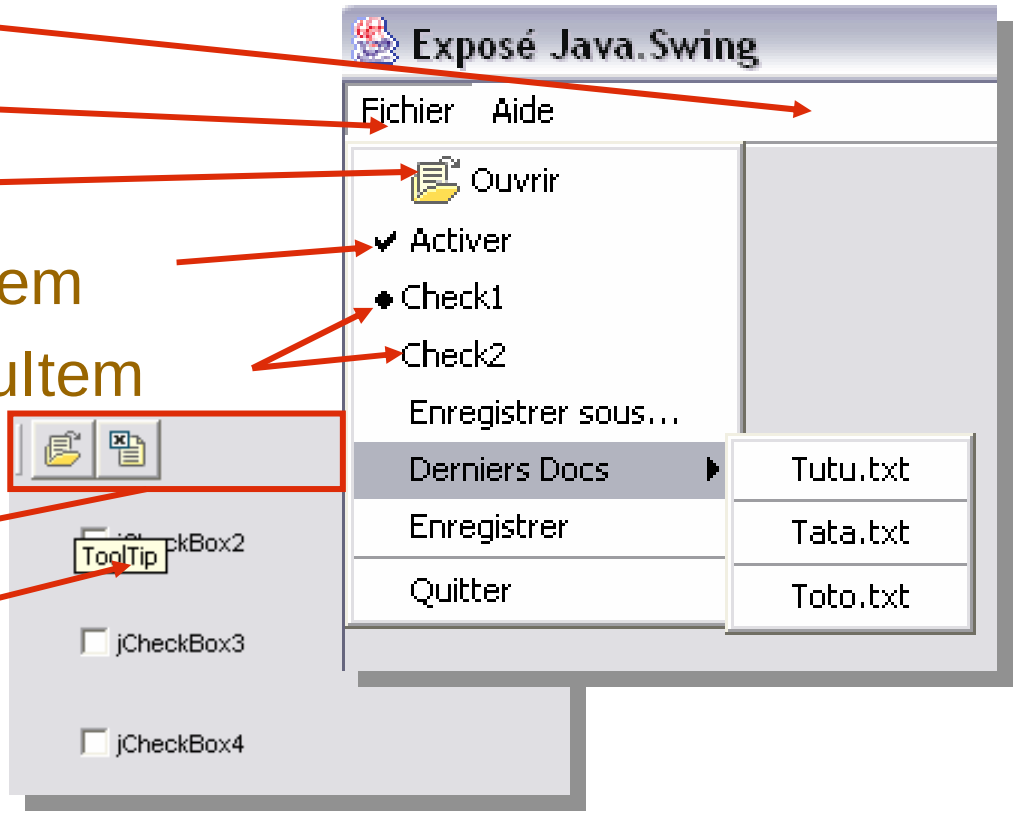
- **JComponent**
  - **JComboBox**
  - **JLabel**
  - **JList**
  - **JMenuBar**
  - **JPanel**
  - **JPopupMenu**
  - **JScrollBar**
  - **JScrollPane**
  - **JTextComponent**
    - **JTextArea**
    - **JTextField**
      - **JPasswordField**
    - **JTextPane**
      - **JHTMLPane**

Autres Composants, toujours des JComponent

- FontChooser
- JColorChooser
- JDesktopIcon
- JDirectoryPane
  - JFileChooser
- JImagePreviewer
- JInternalFrame
- JLayeredPane
  - JDesktopPane
- JOptionPane
- JProgressBar
- JRootPane
- JSeparator
- JSlider
- JSplitPane
- JTabbedPane
- JTable
- JToolBar
- JToolTip
- JTree
- JViewport

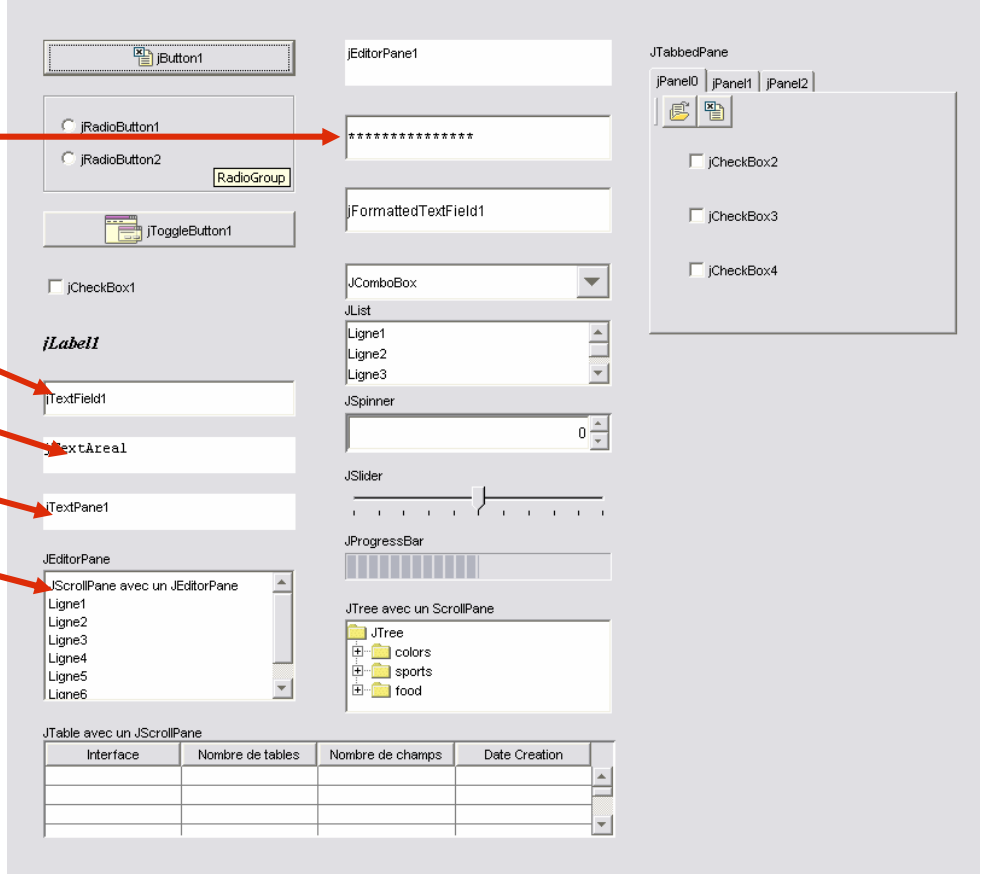
Exemples 1/3, en images

- **Menus, Barre d'outils et ToolTips**
  - JMenuBar
  - JMenu
  - JMenuItem
  - JCheckBoxMenuItem
  - JRadioButtonMenuItem
  - JPopupMenu
  - JToolBar
  - JToolTip



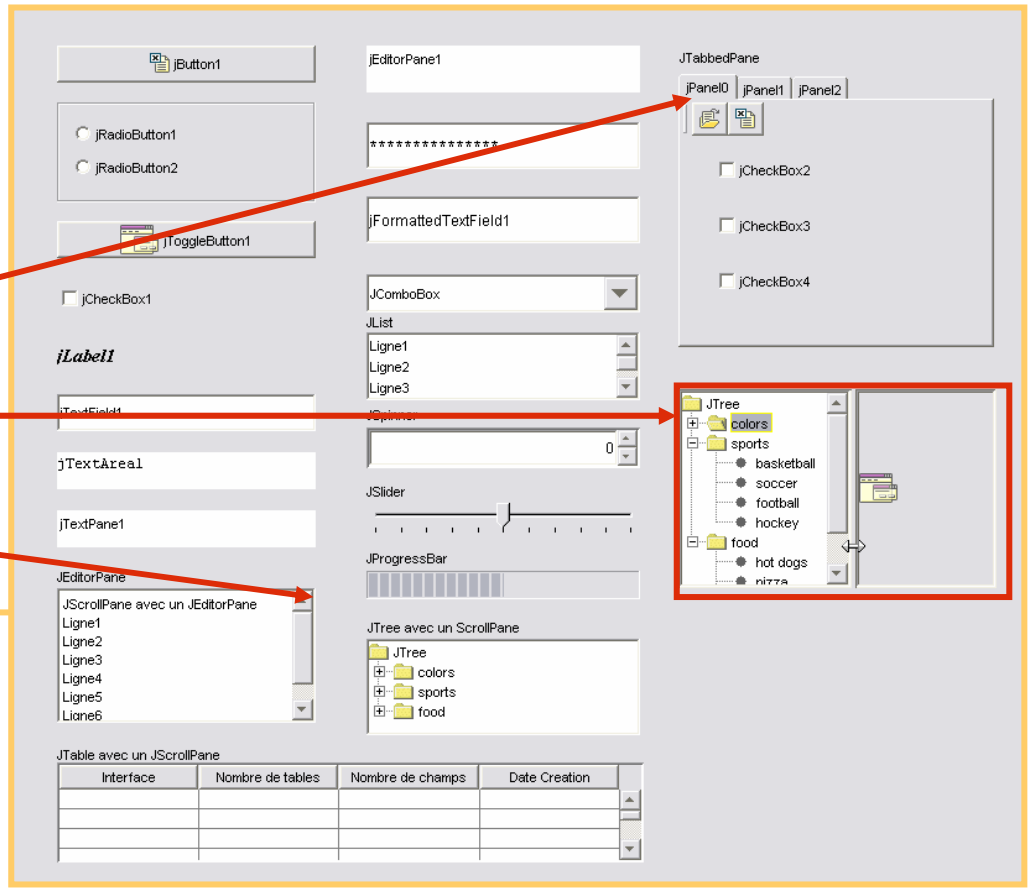
Exemples 2/3

- **Composants Textes**
  - JPasswordField
  - JTextField
  - JTextArea
  - JEditorPane



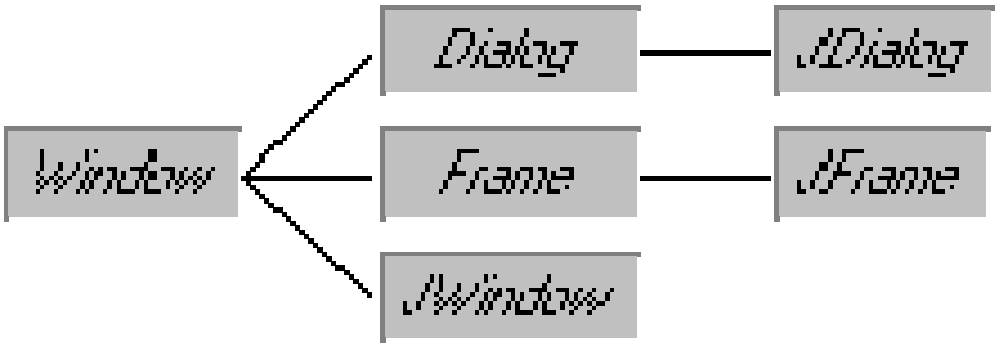
Exemples 3/3

- **Containers**
  - JOptionPane
  - JDialog
  - JTabbedPane
  - JSplitPane
  - JScrollPane
  - JFrame
  - JInternalFrame
  - JDesktopPane
  - JWindow



Hiérarchie, suite, top-level et les autres

- **Les “Top Level containers” et les autres**
  - ***JDialog*, *JFrame*, *JWindow*, *JApplet*, *JInternalFrame***
    - **héritent de *Window***



- **Les autres sont des *JComponent***
  - **Ils sont ajoutés au “*content pane*” d’un “*top level container* “ …**

RootPaneContainer

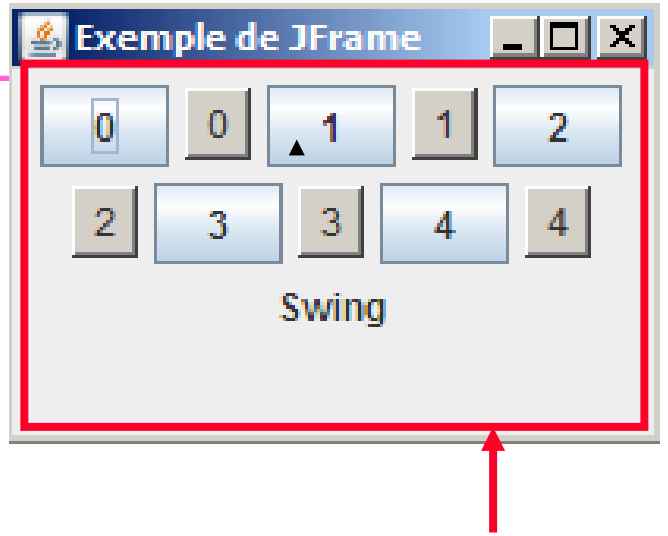
- **Pas d'ajout direct au *container*** *(top level)*
  - ***aJFrame.add (new Button ("Help"));*** ***← non***

- **Ajout au "*content pane*"**
  - ***aJFrame.getContentPane().add (new Button ("Help"));*** ***← oui***

  - *RootPaneContainer* définit la méthode *getContentPane*
    - Implétentée par
      - public Container getContentPane() { return getRootPane().getContentPane(); }

- **Les *top-level***
  - ***JDialog*, *JFrame*, *JWindow*, *JApplet*, *JInternalFrame***

Un exemple

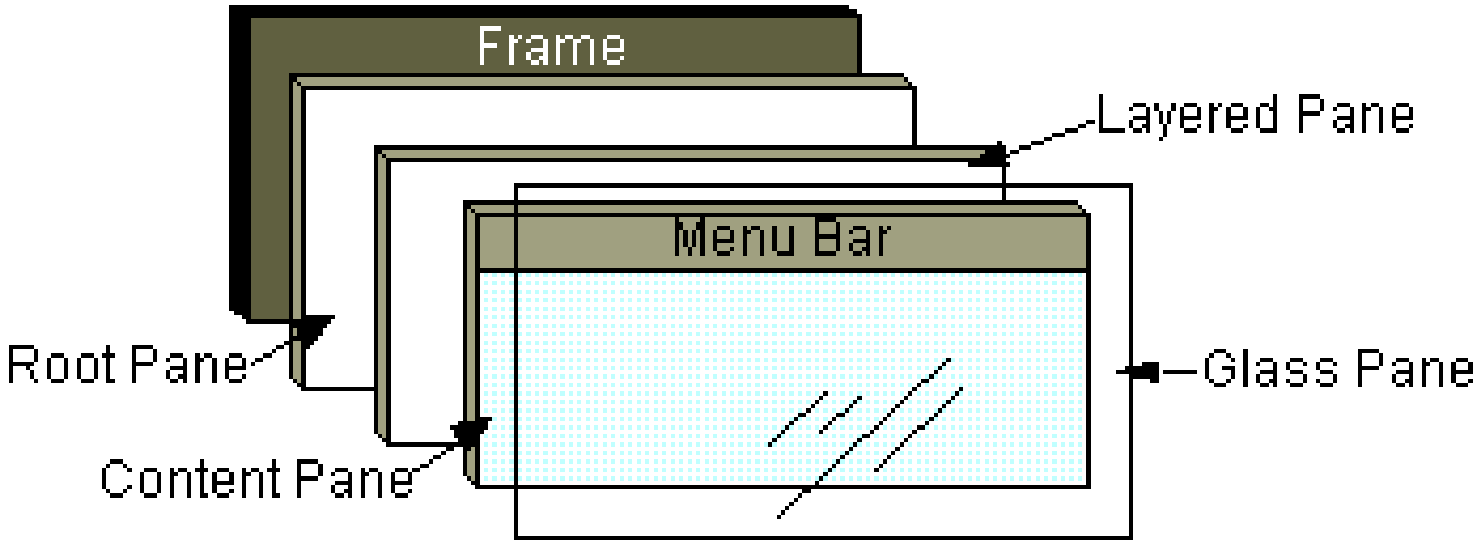


f.getContentPane();

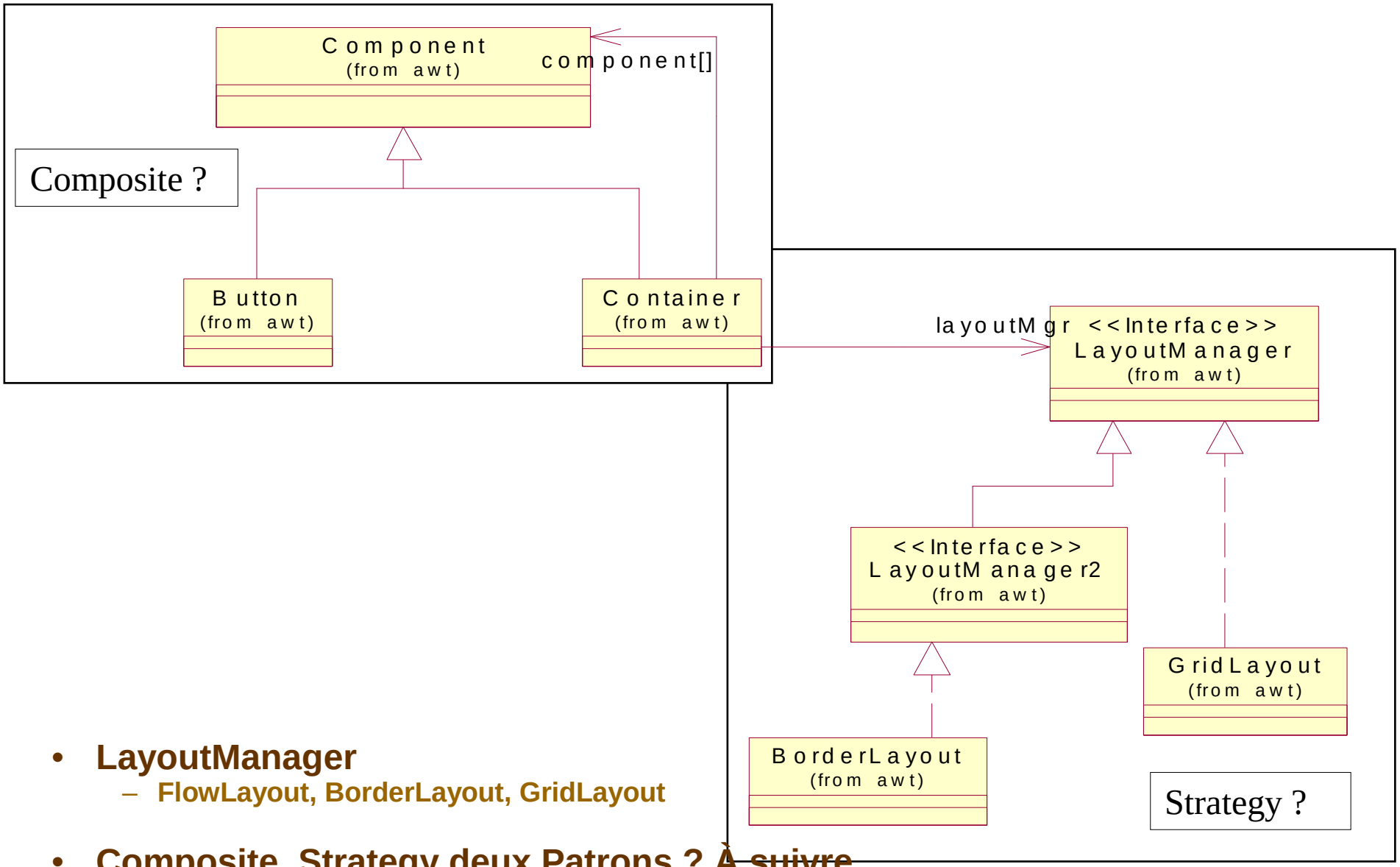
```
import javax.swing.JFrame;
import javax.swing.JButton;
import javax.swing.JLabel;

import java.awt.Container;
import java.awt.FlowLayout;
import java.awt.Button;

public class FrameTester {
  public static void main (String args[]) {
    JFrame f = new JFrame ("Exemple de JFrame");
    Container c = f.getContentPane();
    c.setLayout (new FlowLayout()); // placement des objets (à suivre)
    for (int i = 0; i < 5; i++) {
      c.add (new JButton (Integer.toString(i))); // swing
      c.add (new Button (Integer.toString(i)));  // awt
    }
    c.add (new JLabel ("Swing"));
    f.setSize (300, 200);
    f.show();
  }}
```

ContentPane …

- **Le contenu**



Disposition des objets dans un « container »



- **LayoutManager**
  - **FlowLayout, BorderLayout, GridLayout**
- **Composite, Strategy deux Patrons ? À suivre …**

Les Layout

- **BorderLayout**
- **FlowLayout**
- **GridLayout**
- **CardLayout**
- **….**

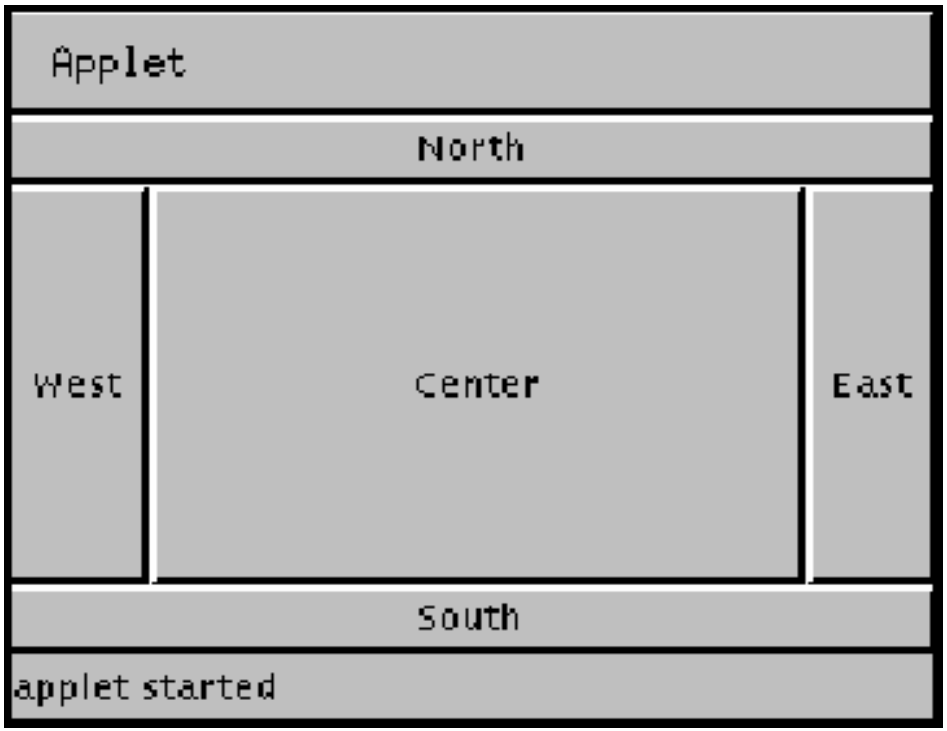
BorderLayout

En 5 zones

BorderLayout .NORTH

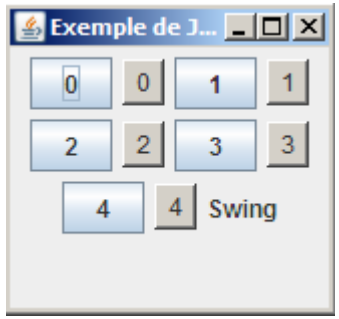
BorderLayout .WEST BorderLayout .CENTER, BorderLayout .EAST

BorderLayout .SOUTH



FlowLayout

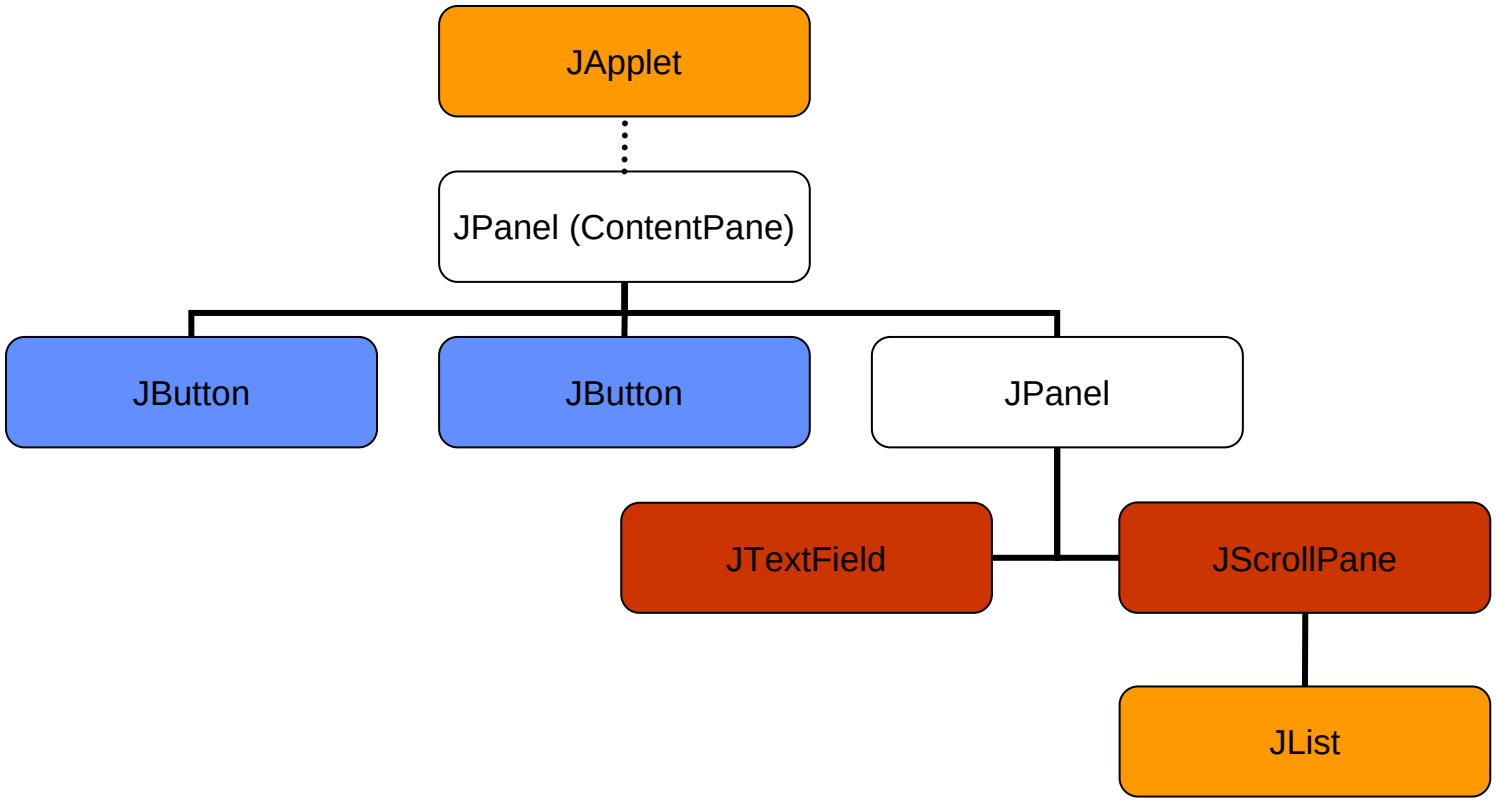
- **comme ils viennent …**



Grid Layout

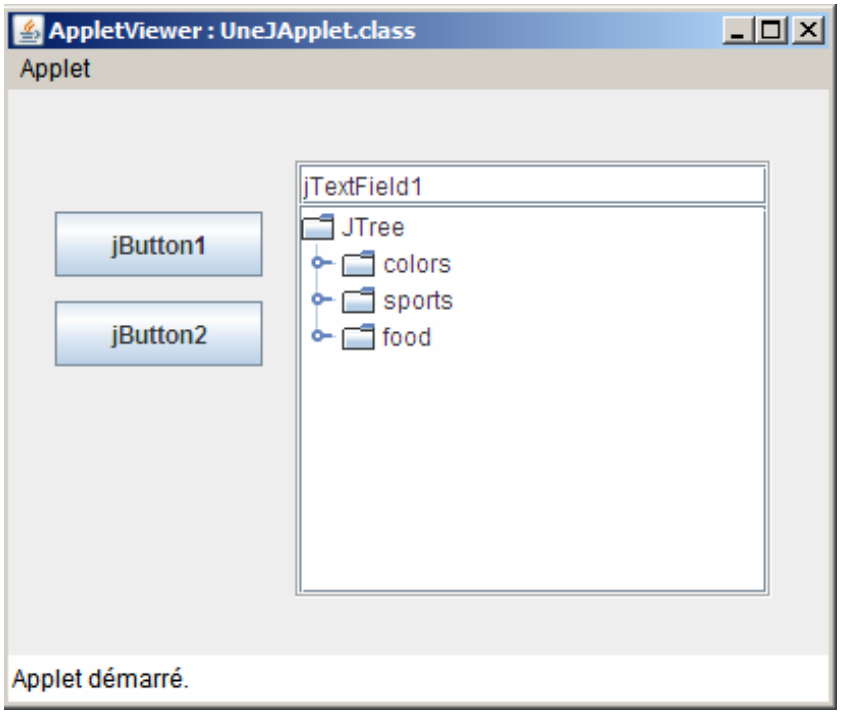
En une table

Un exemple d'IHM : une JApplet



- **Deux boutons, un texte, une liste**

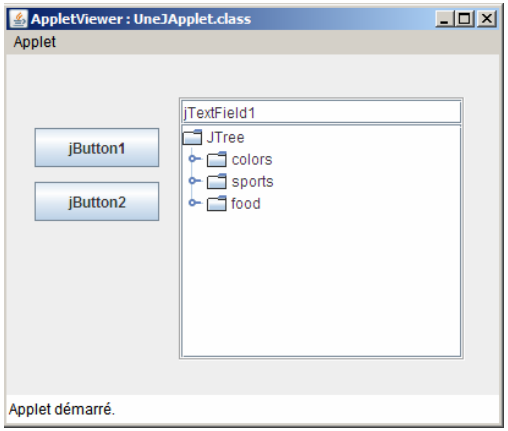
Exemple, Deux boutons, un texte, une liste




```
// quelques déclarations
JPanel contentPane;
JButton jButton1 = new JButton();
JButton jButton2 = new JButton();
JTree jTree1 = new JTree();
JTextField jTextField1 = new JTextField();

JPanel jPanel1 = new JPanel();
JScrollPane jScrollPane1 = new JScrollPane();
BorderLayout borderLayout1 = new BorderLayout();
```

Construction de l'arbre …

contentPane = this.getContentPane(); *// this est une JApplet*



contentPane.add(jButton1, null);
contentPane.add(jButton2, null);
contentPane.add(jPanel1, null);

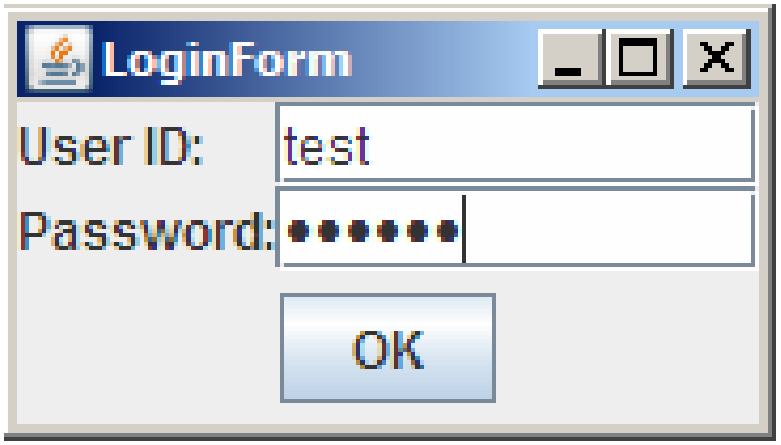
jPanel1.add(jScrollPane1, BorderLayout.CENTER);
jPanel1.add(jTextField1, BorderLayout.NORTH);
jScrollPane1.getViewport().add(jTree1, null);

Divers : de JApplet en JFrame

```
static void run(JApplet applet, int width, int height) {

    JFrame frame = new JFrame();
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    frame.getContentPane().add(applet);
    frame.setSize(width, height);

    applet.init();
    applet.start();
    frame.setVisible(true);

}
```

L'arbre change de racine …

Un autre exemple : Un formulaire



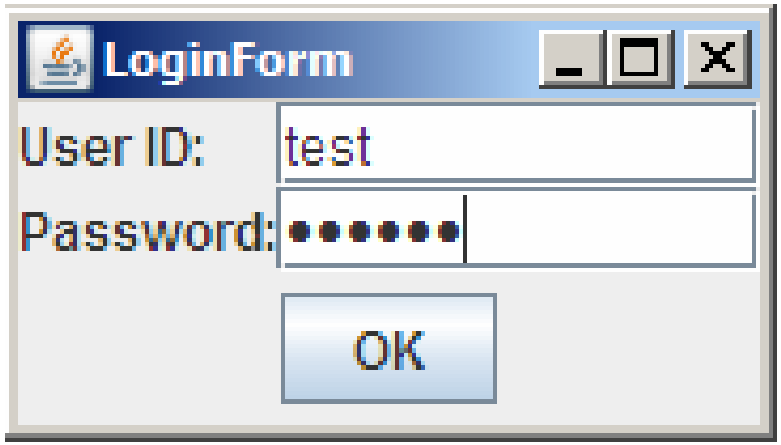

```
public class LoginForm extends JFrame {

  public LoginForm() {
    super("LoginForm");
    Container contents = getContentPane();
   // contents.setLayout(new BorderLayout()); // par défaut
    contents.add( getLoginPanel(), BorderLayout.CENTER);
    contents.add( getButtonPanel(), BorderLayout.SOUTH);
    pack(); setVisible(true);
   }
```

Le dessin suite …

```
…
  public JPanel getLoginPanel() {
     JPanel panel = new JPanel(new BorderLayout());
     JPanel labelPanel = new JPanel(new GridLayout(2,0));
     JLabel userIdLabel = new JLabel("User ID:");
     labelPanel.add(userIdLabel);
     JLabel passwordLabel = new JLabel("Password:");
     labelPanel.add(passwordLabel);
     panel.add(labelPanel, BorderLayout.WEST);
     JPanel fieldPanel = new JPanel(new GridLayout(2,0));
     JTextField userIdField = new JTextField(10);
     fieldPanel.add(userIdField);
     JPasswordField passwordField = new JPasswordField(10);
     fieldPanel.add(passwordField);
     panel.add(fieldPanel, BorderLayout.CENTER);
     return panel;
   }


 public JPanel getButtonPanel() {
   JPanel panel = new JPanel(new FlowLayout());
   JButton okButton = new JButton("OK");
   panel.add(okButton);
   return panel;
}
```

Démonstration

- **Un Arbre ?**



Autres composants

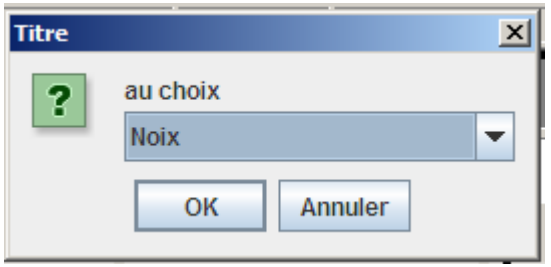
- **À suivre**

- **Fenêtres de dialogues**
  - **JOptionPane**

- **Apparence des objets graphiques**
  - **Apparence « windows », Metal,**
  - ***UIManager.setLookAndFeel***

Une autre famille : JOptionPane

Un exemple




```
import javax.swing.JOptionPane;
public class JOption{

  public static void main(String[] args) {
    Object response = JOptionPane.showInputDialog(null,
      "au choix",
      "Titre",
      JOptionPane.QUESTION_MESSAGE,
      null,
      new Object[] { "Amandes", "Noisettes", "Noix"}, "Noix");

    // response = ( "Amandes" | "Noisettes" | "Noix" | null)
  }
}
```

Au choix

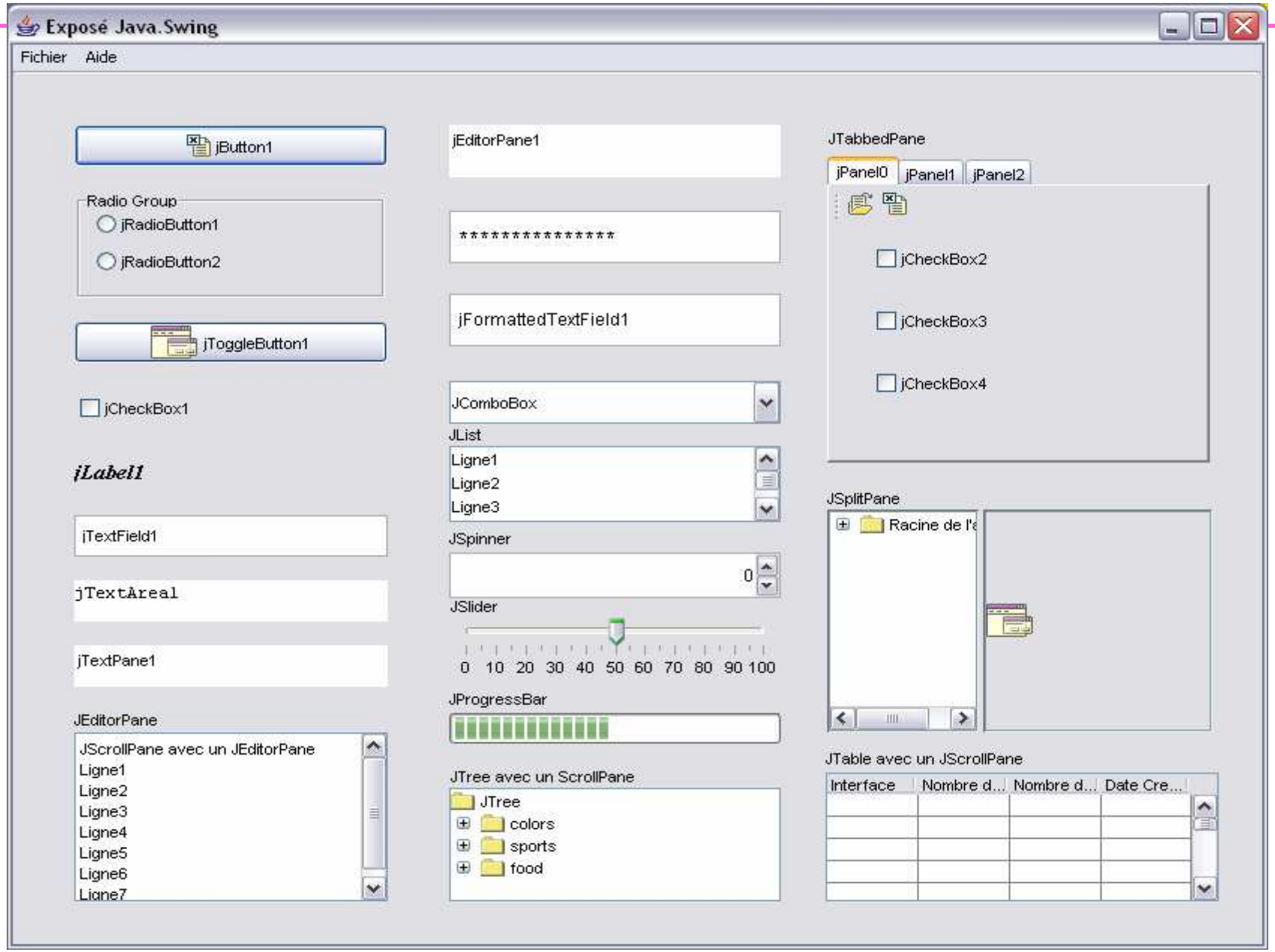
Méthode	Description
showConfirmDialog	Pose une question oui/non/annuler.
showInputDialog	Demande un choix (c.f. exemple précédent).
showMessageDialog	Une indication.
showOptionDialog	Unification des 3 précédents.

- **Lecture bloquante … jusqu'à ce que l'utilisateur**

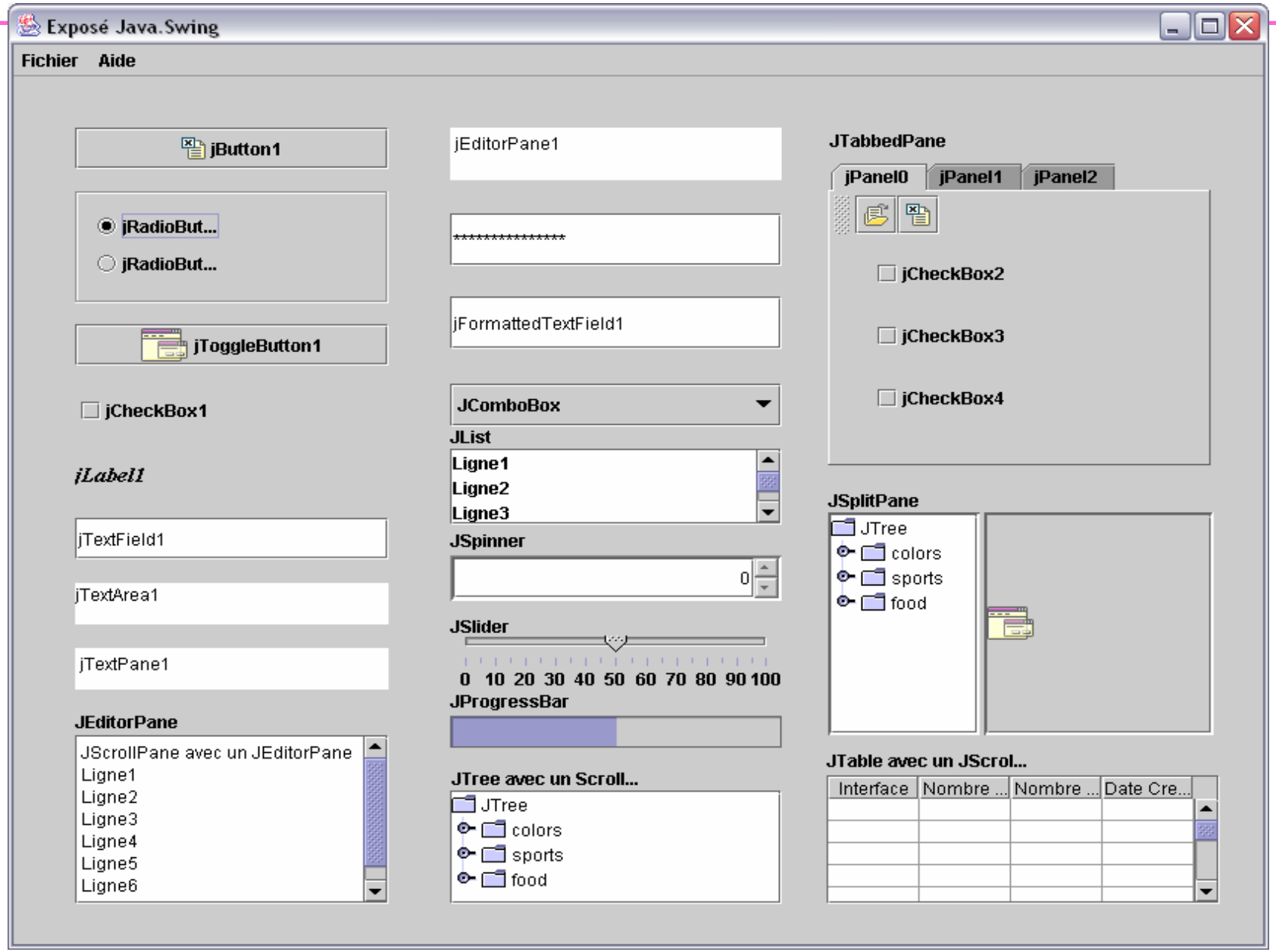
Les Apparences LookAndFeel

- **Choix de l'apparence.**
  - **Metal (par défaut)**
    - ***UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");***
  - **Windows**
    - ***UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");***
  - **Apparence Motif**
    - ***UIManager.setLookAndFeel("com.sun.java.swing.plaf.motif.MotifLookAndFeel");***

Apparence Windows

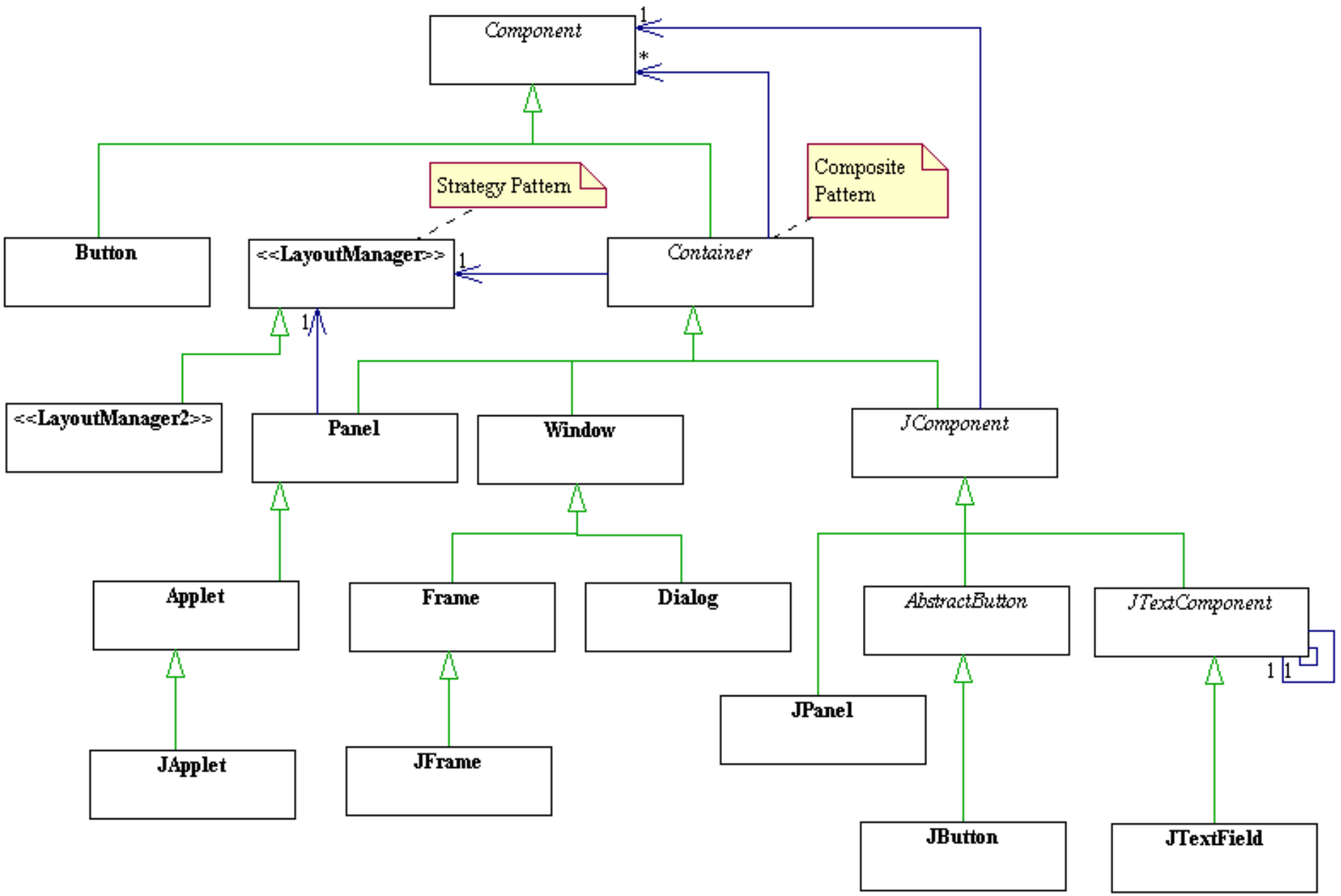


Apparence Métal (Par Défaut)

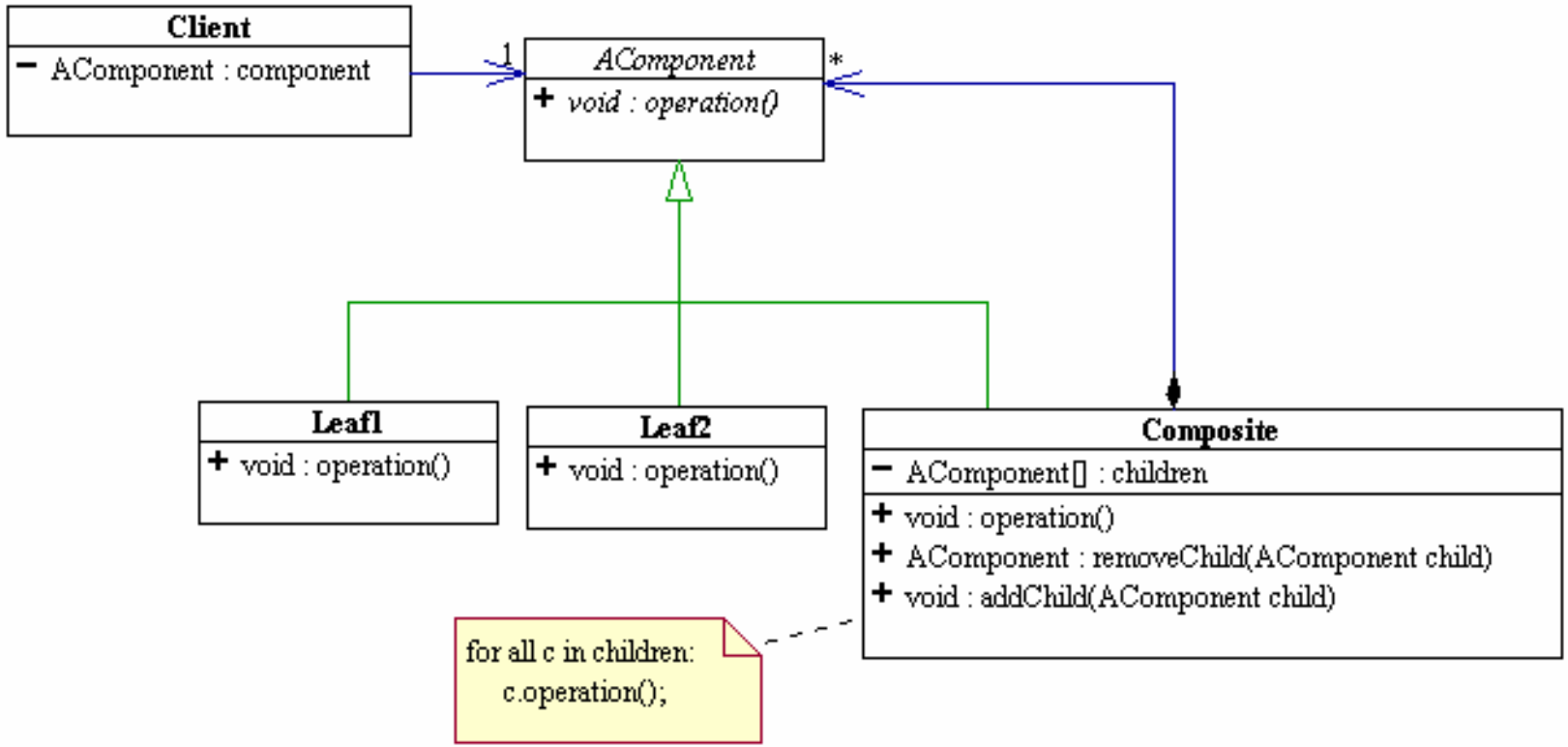


Swing : où sont les patrons ?

- **Patron Composite**
  - **Construction d'une IHM**
- **Patron Stratégie**
  - **Disposition des objets graphiques**

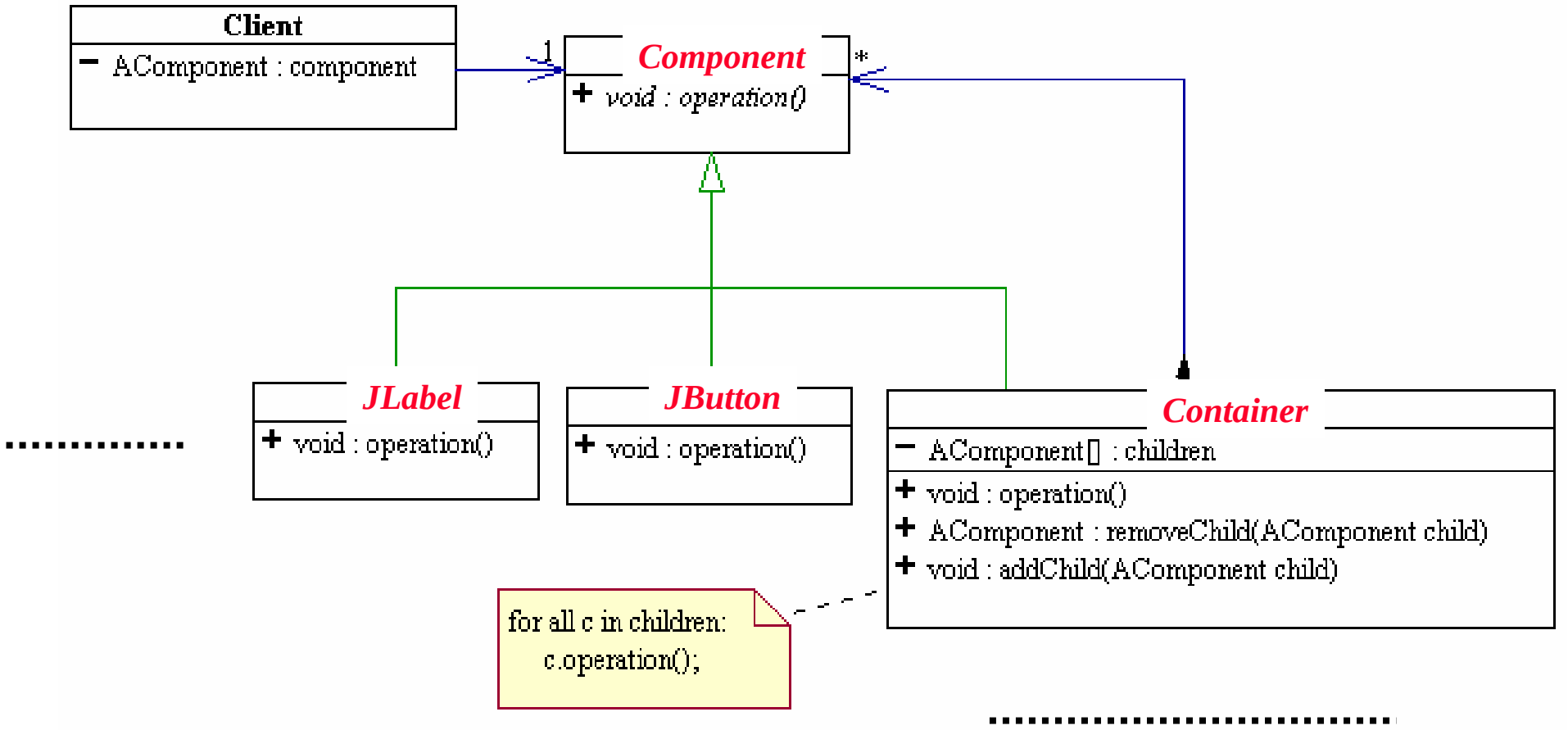

Component
1
*
Strategy Pattern
Composite Pattern
Button
<<LayoutManager>>
1
Container
1
1
<<LayoutManager2>>
Panel
Window
JComponent
Applet
Frame
Dialog
AbstractButton
JTextComponent
1
1
JPanel
JApplet
JFrame
JButton
JTextField

Le patron Composite, rappel



- Structures de données récursives
  - Un Composant est une feuille ou un composite
  - Un Composite est un Composant

Le patron Composite et l'API *graphique*



Le formulaire est une instance du composite !

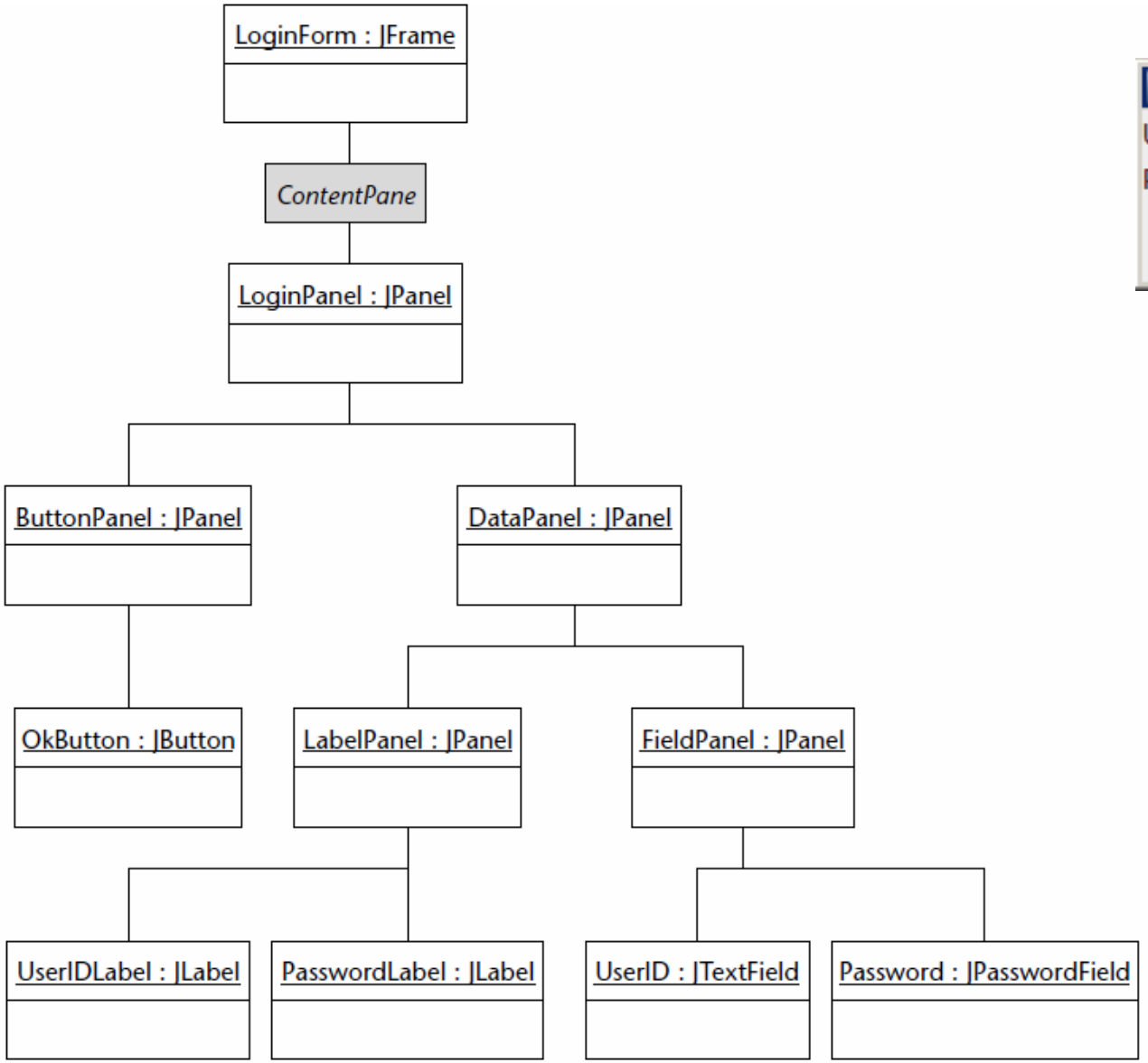
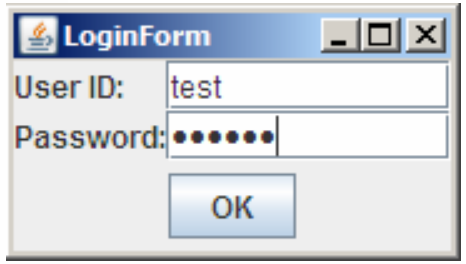


Figure 1.9 Swing login containment hierarchy.



- **src : Mastering JSF page 31**

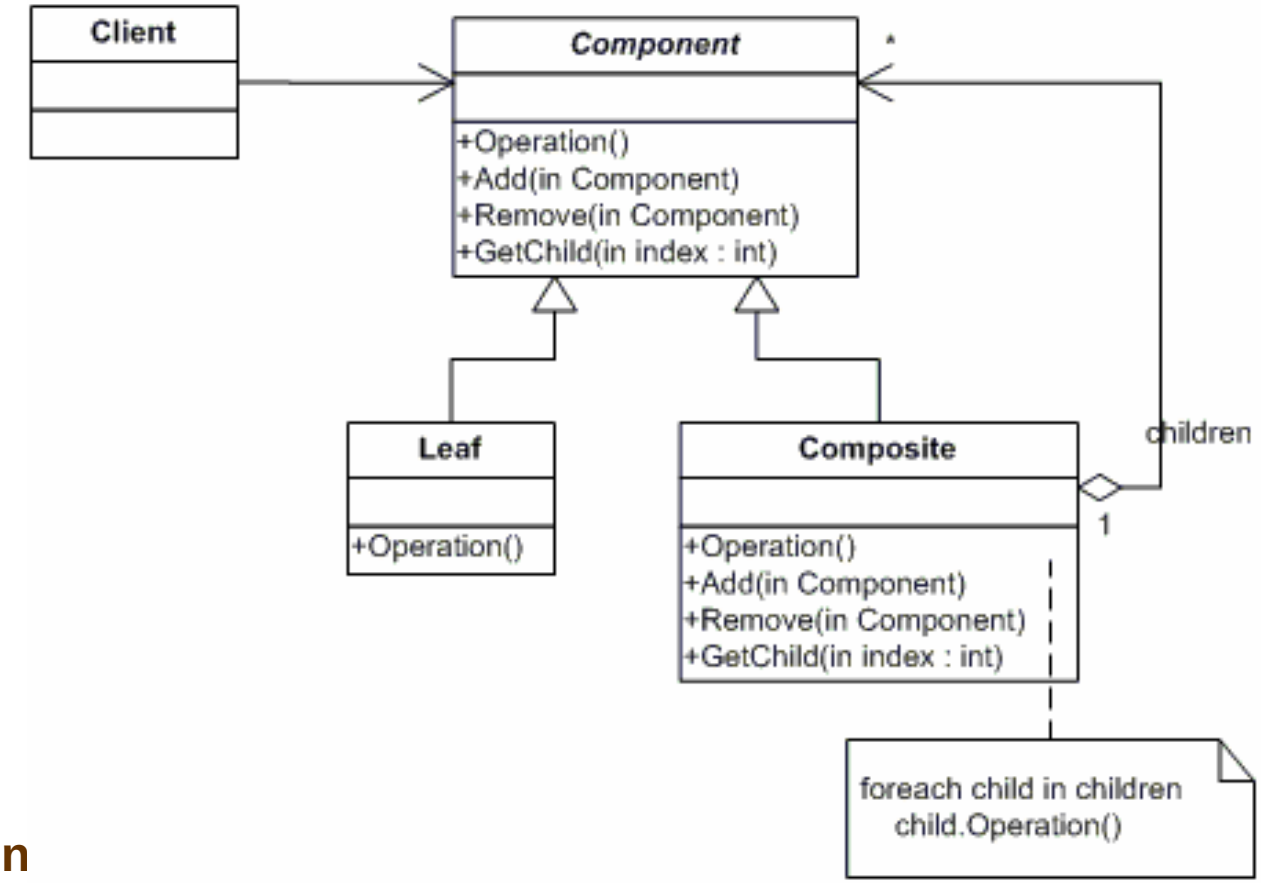
list() comme operation()

- **Méthode de la classe Component quelque soit l'arbre**

- **Component loginForm = new LoginForm();**
- **loginForm.list();**

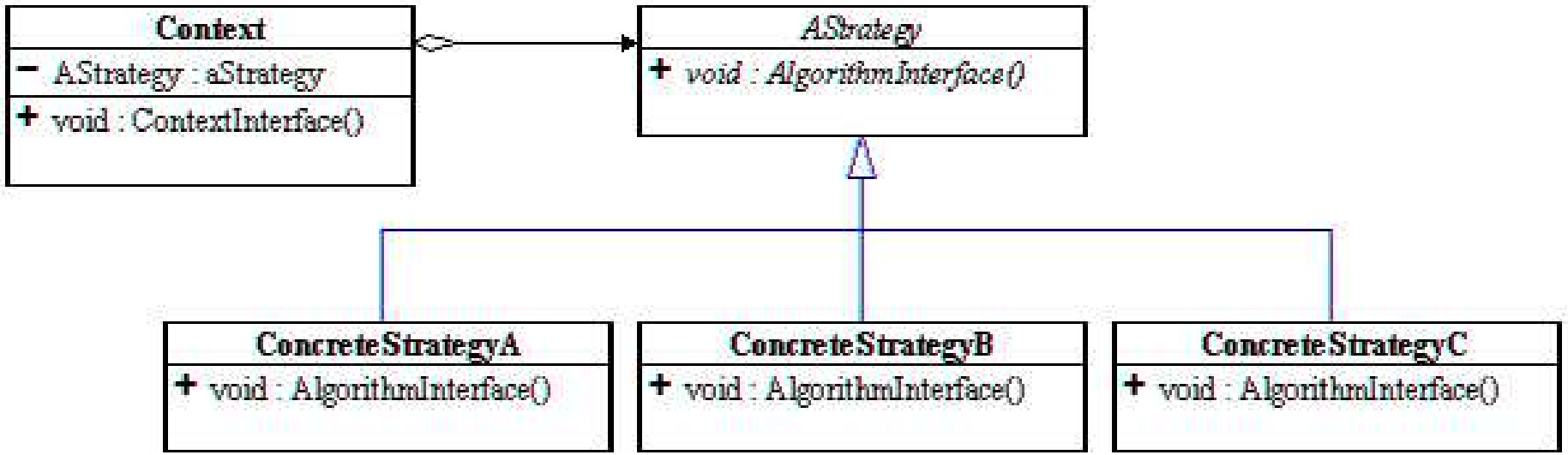
```
LoginForm[frame0,0,0,183x103,layout=java.awt.BorderLayout,title=LoginForm,resizable,normal,defaultCloseOperation=HIDE_ON_CLOSE,rootPane=javax.swing.JRootPane[,4,23,175x76,layout=javax.swing.JRootPane$RootLayout,alignmentX=0.0,alignmentY=0.0,border=,flags=16777673,maximumSize=,minimumSize=,preferredSize=],rootPaneCheckingEnabled=true]

javax.swing.JRootPane[,4,23,175x76,layout=javax.swing.JRootPane$RootLayout,alignmentX=0.0,alignmentY=0.0,border=,flags=16777673,maximumSize=,minimumSize=,preferredSize=]

javax.swing.JPanel[null.glassPane,0,0,175x76,hidden,layout=java.awt.FlowLayout,alignmentX=0.0,alignmentY=0.0,border=,flags=16777217,maximumSize=,minimumSize=,preferredSize=]

javax.swing.JLayeredPane[null.layeredPane,0,0,175x76,alignmentX=0.0,alignmentY=0.0,border=,flags=0,maximumSize=,minimumSize=,preferredSize=,optimizedDrawingPossible=true]
……
……
……
```

Retour sur L'original



- **Méthode Operation**
  - **Component c = … une feuille ou un composite**
  - **C.Operation();**
- **Notons l'accès aux enfants depuis un nœud composite,** *classe Component*
  - **Méthode getChild**

Le patron Strategy

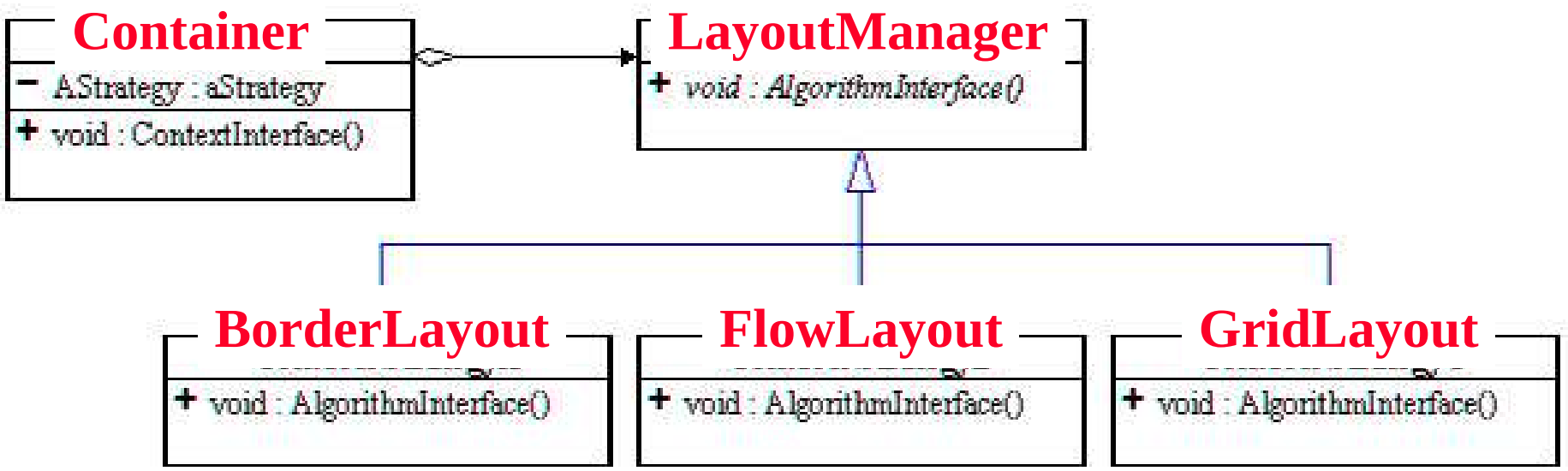


- **Context ctxt = new Context(new ConcreteStrategyB());**
  - **Le champ d'instance aStrategy est affecté par l'argument du constructeur**

Puis

- **ctxt.contextInterface();**
  - **Retournera un AlgorithmInterface() concret**

Le patron Strategy et les « layout »

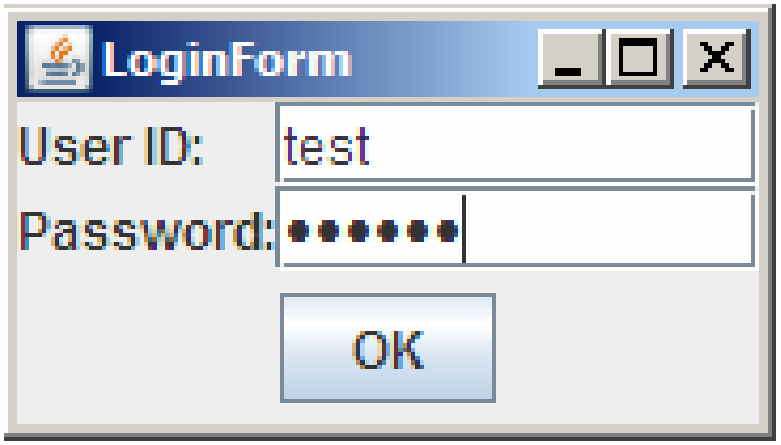


- **Exemple :**
  - **Container content = getContentPane();**
  - **content.setLayout(new BorderLayout());**

Réagir aux clics …

- **Observateur / observé**
  - **JComponent / Listener**
  -

Réagir aux clics : le formulaire déjà vu

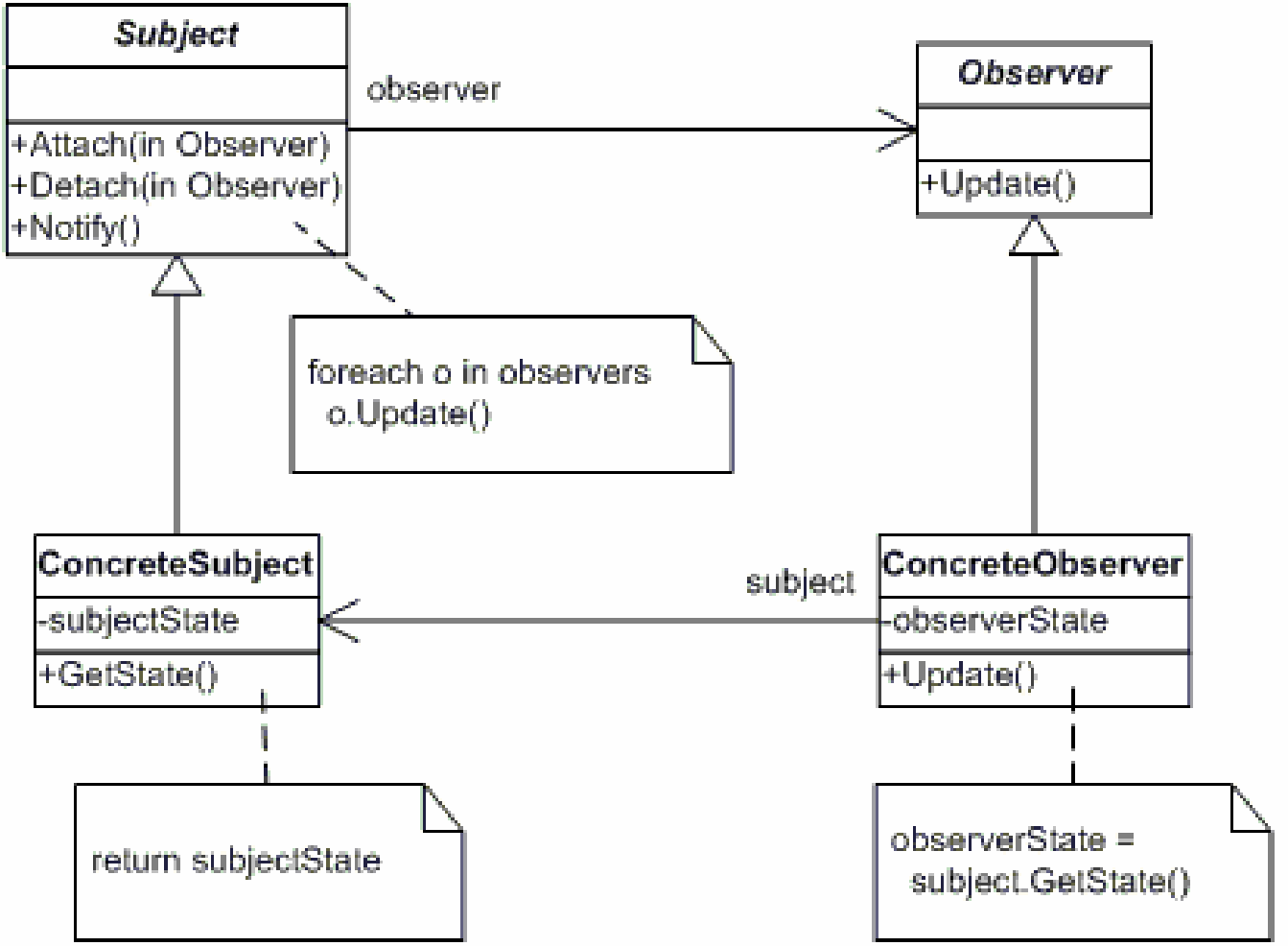


- **Associer une action au clic sur OK**
  - **légitime**
- **Patron observateur/observé**
  - **Les « listeners »**

Patron observé/observateur-écouteur(ou listener)

- **1) enregistrement auprès du bouton d'un écouteur**
  - **bouton.addActionListener(ActionListener al)**
    - **Plusieurs écouteurs sont possibles**
- **2) A chaque clic les écouteurs sont notifiés**
  - **al.actionPerformed(ActionEvent ae)**
  - **L'exemple du formulaire suit …**

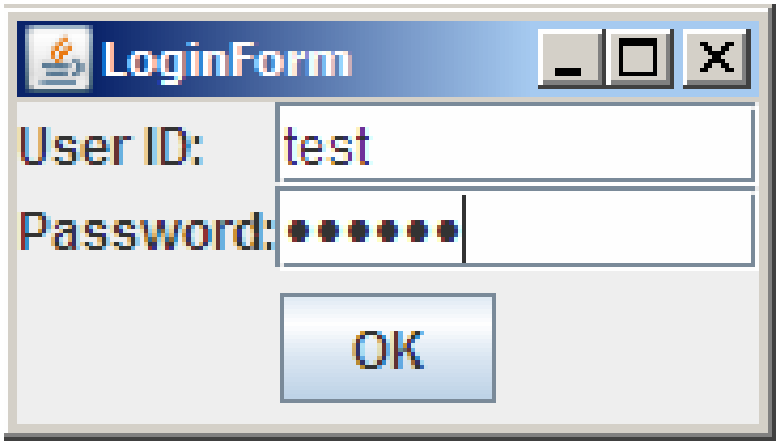
le patron Observateur, l'original



- http://www.codeproject.com/gen/design/applyingpatterns/observer.gif

Ajout d'un observateur du bouton

```
public JPanel getButtonPanel() {
  JPanel panel = new JPanel(new FlowLayout());
  JButton okButton = new JButton("OK");
  panel.add(okButton);

  okButton.addActionListener(new OkButtonAction());

  return panel;
 }

}
```



OkButton action !!!

```
class OkButtonAction implements ActionListener{

  public void actionPerformed(ActionEvent ae){
    // Vérification du nom et mot de passe
    // par exemple …
  }

}
```

Différentes formes syntaxiques possibles
Classe interne et statique
Classe interne et membre
Classe anonyme

Action, Listeners et les autres, par convention

- **Un composant swing enregistre ses écouteurs**
  - **addXXXXListener**
- **Les écouteurs implémentent**
  - **L'interface XXXXListener et « sont des » java.awt.event.EventListener**
  - **Cette interface décrit une méthode acceptant en paramètre un XXXXEvent**
  - **Chaque écouteur s'enregistre auprès du composant swing**
- **A chaque changement d'état du composant swing**
  - **Les méthodes des écouteurs pré-enregistrés sont exécutées**
    - **Le paramètre XXXXEvent contient au moins la source de la notification**

EventListener et ses descendants

java.util

Interface EventListener

All Known Subinterfaces:

Action, ActionListener, AdjustmentListener, AncestorListener, AWTEventListener, BeanContextMembershipListener, BeanContextServiceRevokedListener, BeanContextServices, BeanContextServicesListener, CaretListener, CellEditorListener, ChangeListener, ComponentListener, ConnectionEventListener, ContainerListener, ControllerEventListener, DocumentListener, DragGestureListener, DragSourceListener, DragSourceMotionListener, DropTargetListener, FlavorListener, FocusListener, HandshakeCompletedListener, HierarchyBoundsListener, HierarchyListener, HyperlinkListener, IIOReadProgressListener, IIOReadUpdateListener, IIOReadWarningListener, IIOWriteProgressListener, IIOWriteWarningListener, InputMethodListener, InternalFrameListener, ItemListener, KeyListener, LineListener, ListDataListener, ListSelectionListener, MenuDragMouseListener, MenuKeyListener, MenuListener, MetaEventListener, MouseInputListener, MouseListener, MouseMotionListener, MouseWheelListener, NamespaceChangeListener, NamingListener, NodeChangeListener, NotificationListener, ObjectChangeListener, PopupMenuListener, PreferenceChangeListener, PropertyChangeListener, RowSetListener, RowSorterListener, SSLSessionBindingListener, StatementEventListener, TableColumnModelListener, TableModelListener, TextListener, TreeExpansionListener, TreeModelListener, TreeSelectionListener, TreeWillExpandListener, UndoableEditListener, UnsolicitedNotificationListener, VetoableChangeListener, WindowFocusListener, WindowListener, WindowStateListener

- **Héritage au sens des interfaces**

Des Adaptateurs à adapter

- **Implémenter un XXXXListener peut être fastidieux**
  - **Surtout si une seule méthode de l'interface est concernée …**
  - **Exemple : l'interface WindowListener**
    - public void **windowActivated**(WindowEvent e)
    - public void **windowClosed**(WindowEvent e)
    - public void **windowClosing**(WindowEvent e)
    - public void **windowDeactivated**(WindowEvent e)
    - public void **windowDeiconified**(WindowEvent e)
    - public void **windowIconified**(WindowEvent e)
    - public void **windowOpened**(WindowEvent e)
  - **Au total 7 méthodes à implémenter …**
- **Utile : il existe une classe XXXXAdapter qui implémente XXXXListener**
  - **Exemple la classe WindowAdapter**

XXXXAdapter à adapter

- **Exemple suite : WindowAdapter**
  - **Au clic dans la case de fermeture alors arrêt brutal du programme**
  - **Pas d'action pour les 6 autres méthodes….**

```
Window w = …

w.addWindowListener(
   new WindowAdapter(){
      public void windowClosing(WindowEvent event) {
          System.exit(0);
      }
   });
```

Swing, un constat

C’est une première présentation

- Des composants graphiques
- Une gestion des évènements
- Un *look and feel* indépendant

- IHM alors MVC
  - Modèle Vue Contrôleur
  - Comment ?, quel déploiement ?, pourquoi faire ?

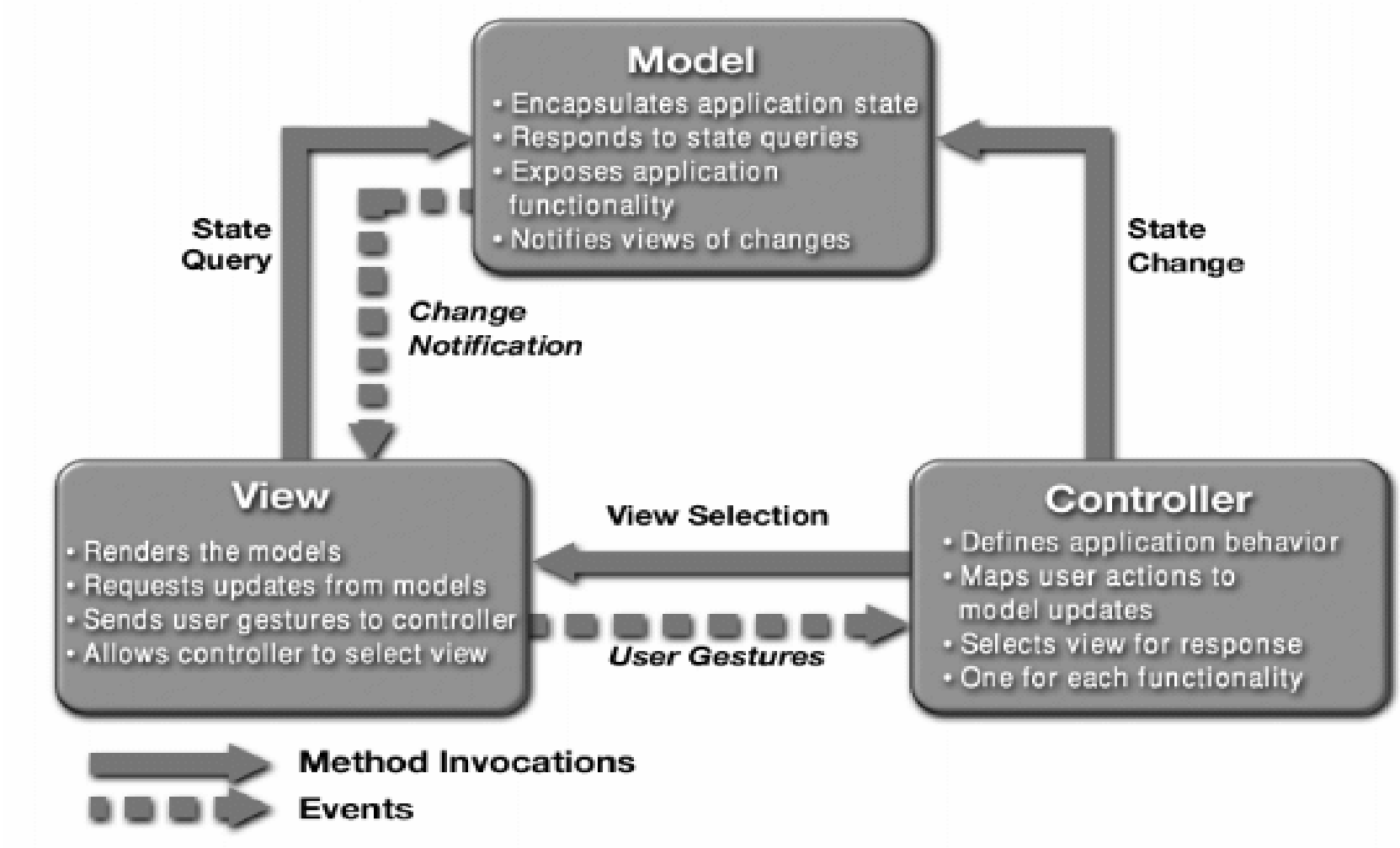
Interactivité, MVC

- **L'interface graphique**
  - **La vue**
- **Interactions avec l'utilisateur**
  - **Le contrôle**
- **Les données de l'application**
  - **Le modèle**
- **A la recherche**

d'une adéquation des données de l'application et des vues

  - **Paradigme MVC**
    - **Modèle Vue Contrôleur**

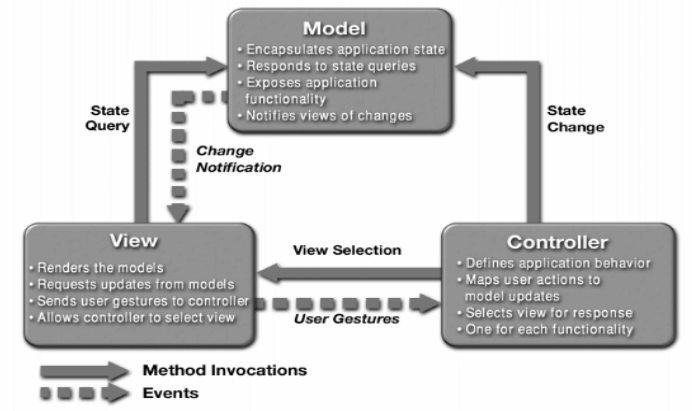
MVC, doc de Sun

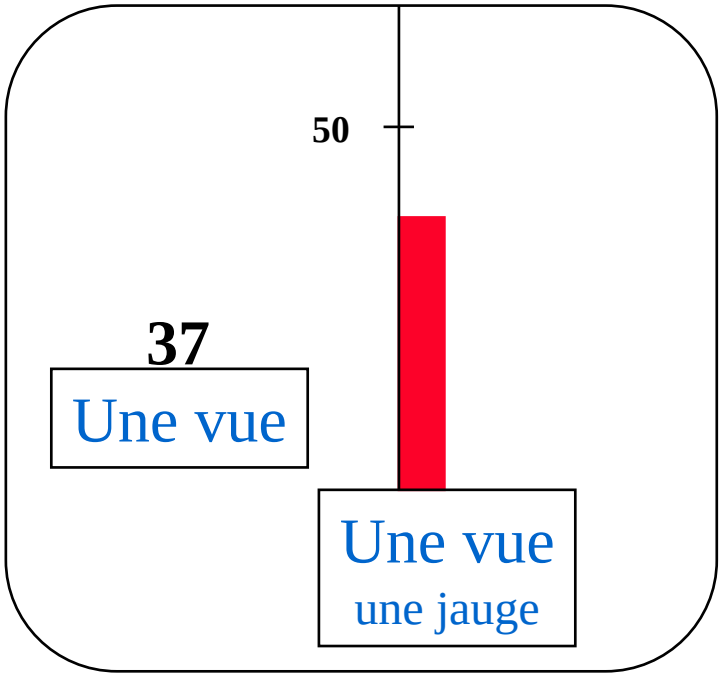


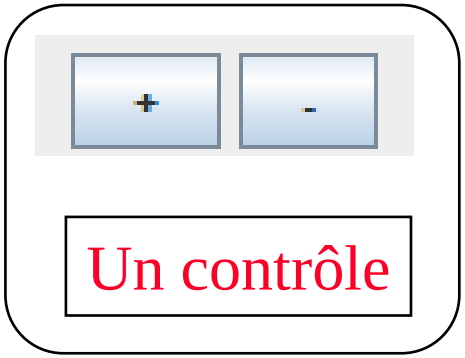
- http://java.sun.com/blueprints/patterns/MVC-detailed.html
- **Un exemple …**

MVC un exemple : un nombre !

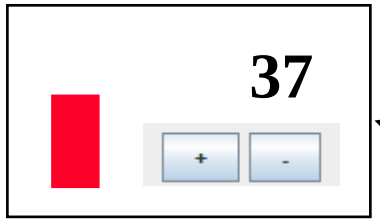
le modèle : un nombre





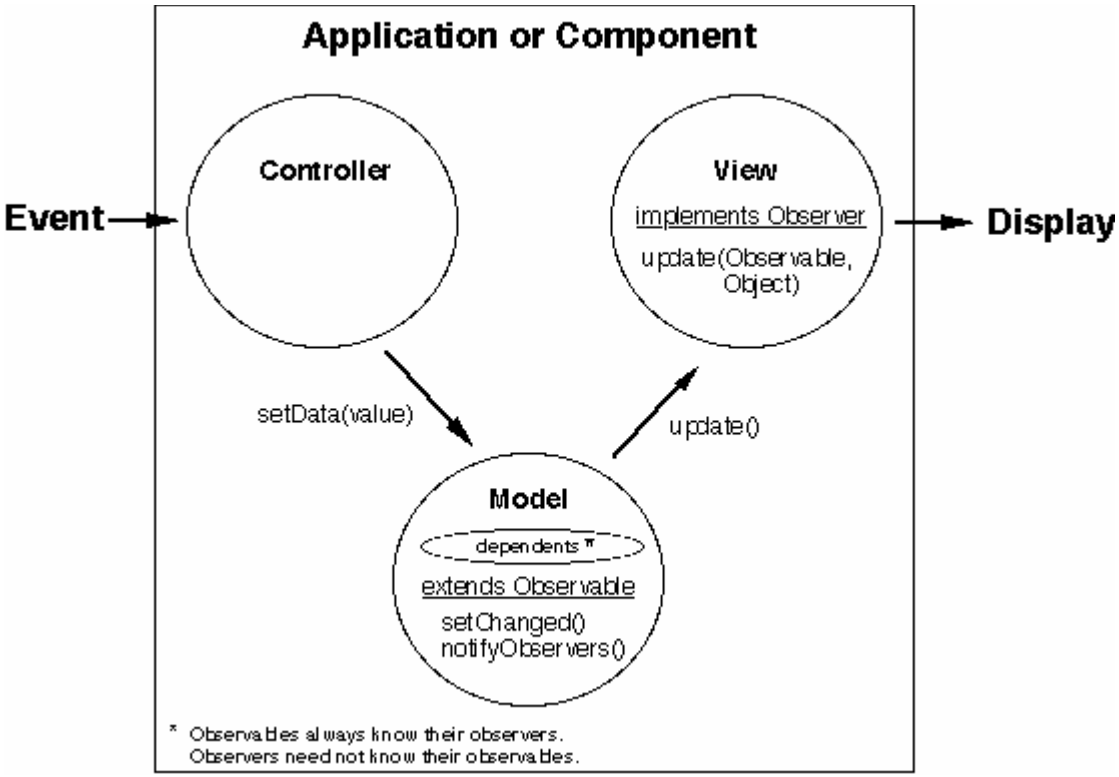


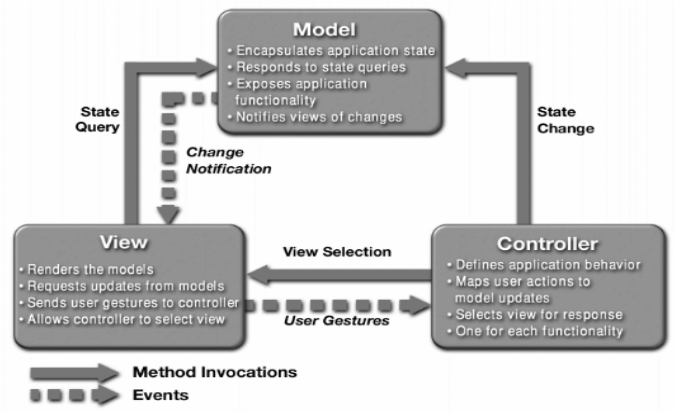
- **Déploiement ?**
  - **IHM ?**
  - **Contrôle ?**
  - **Modèle ?**



Une JFrame

Déploiement ? Choix de classes

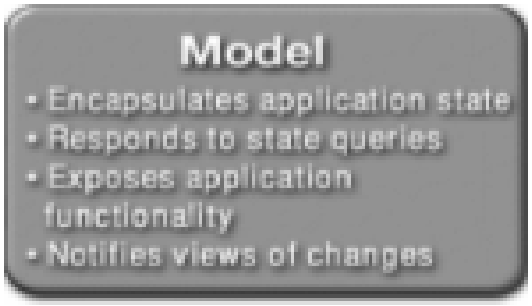




- **Discussion …**
  - **Le modèle :** **la classe Nombre**
  - **La Vue :** **Une JFrame, une jauge, un affichage, des boutons**
  - **Le Contrôle :** **Réactions aux actions de l'utilisateur …**

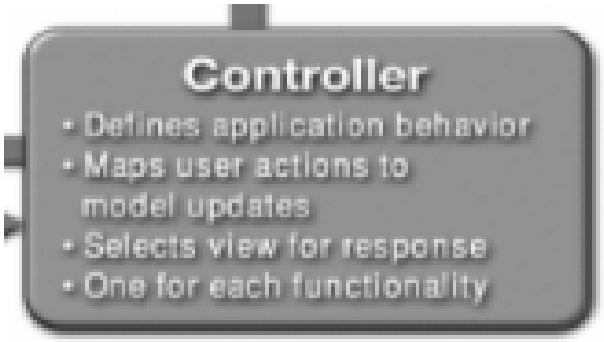
Un déploiement possible, démonstration

- **La classe Nombre est Observable**
  - **Elle hérite de java.util.Observable**

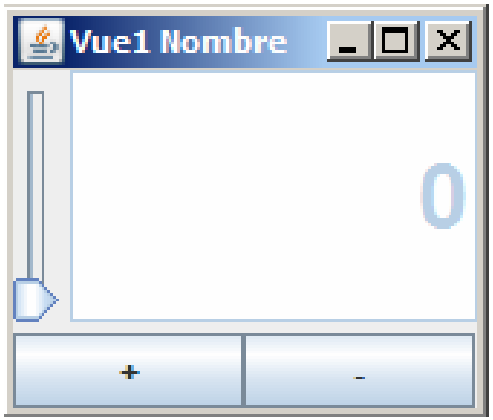


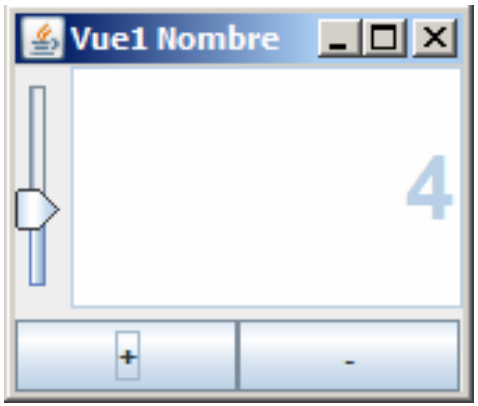
- **Les Vues sont des observateurs**
  - **Elles implémentent java.util.Observer,**
  - **Peuvent être des IHM, des « Container » swing,**
  - **Sans interface : des vues sans être vues ?…**
    - **Par exemple : un journal des évènements**
  - **…**

- **Les Contrôles gèrent les actions de l'utilisateur**
  - **Elles implémentent les XXXXListener des composants swing**
  - **Une classe par action ?**



Exemple un nombre, une vue, un contrôle





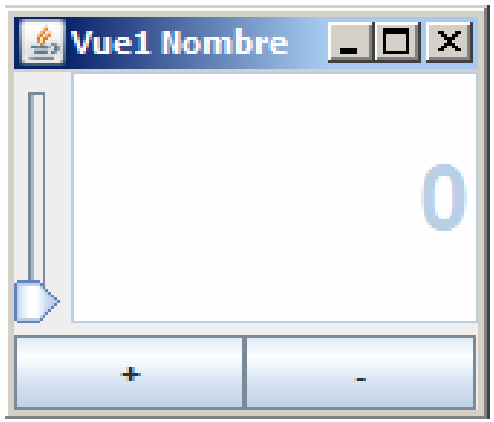

```
public class MainNombreMVC{

  public static void main(String[] args){
    Nombre nombre = new Nombre(0,10); // le modèle

    Vue1 vue1 = new Vue1(nombre);

    ControleVue1 controle1 = new ControleVue1(nombre, vue1);

  }
}
```

Le modèle

```
public class Nombre extends java.util.Observable{
  public final int VALEUR_MIN;
  public final int VALEUR_MAX;
  private int valeur;

  public Nombre(int min, int max){
   this.VALEUR_MIN = this.valeur = min;
   this.VALEUR_MAX = max;
  }

  public void inc(){
    if(valeur < VALEUR_MAX){
      this.valeur++;
      setChanged();
      notifyObservers();
    }
  }

  public void dec(){ // idem -- …
  … getValeur() + setValeur(…) + toString()
```

La Vue est une JFrame et implémente Observer

```
public class Vue1 extends JFrame implements Observer{

  private JButton     plus;
  private JButton     moins;
  private JTextField  valeur;
  private JSlider     jauge;

  public Vue1(Nombre nombre) {
    super("Vue1 Nombre");
     nombre.addObserver(this);
    // Mise en place des composants graphiques
```



en quelques lignes …,
un arbre instance du composite

```
    pack();
    setVisible(true);
  }
```

La Vue implémente java.util.Observer

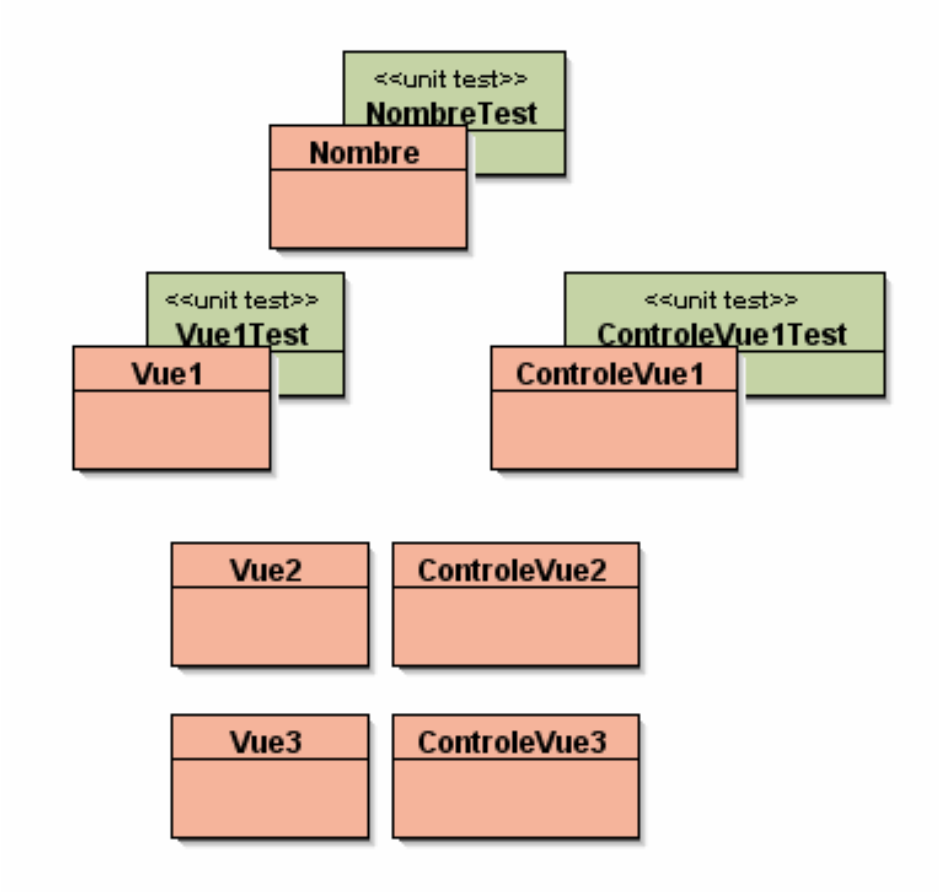
```
@Override
  public void update(Observable obs, Object arg){
  if(obs instanceof Nombre){
      Nombre n = (Nombre)obs;
      this.valeur.setText(n.toString());
      this.jauge.setValue(n.getValeur());
    }
  }


// accesseurs
  public JButton getPlus(){return plus;}
  public JButton getMoins(){return moins;}

}
```

Le Contrôle implémente les XXXXListener

```
public class ControleVue1{
  private Nombre nombre;

  public ControleVue1(Nombre nombre, Vue1 vue){
    this.nombre = nombre;
    vue.getMoins().addActionListener(
      new ActionListener(){
        public void actionPerformed(ActionEvent ae){
          ControleVue1.this.nombre.dec();
        }
      });

    vue.getPlus().addActionListener(
      new ActionListener(){
        public void actionPerformed(ActionEvent ae){
          ControleVue1.this.nombre.inc();
        }
      });

  }
}
```

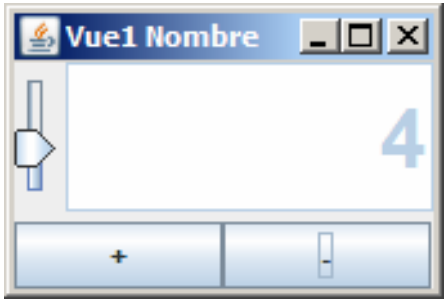
adéquation actions de l'utilisateur / opérations sur le modèle

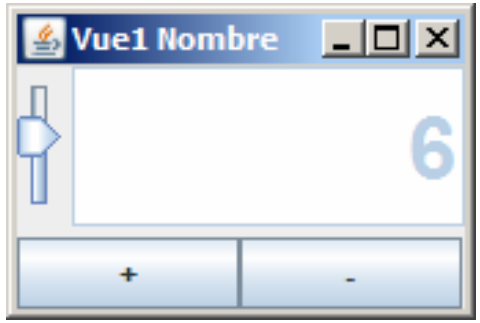
Démonstration



- **M V C**
- **http://jfod.cnam.fr/NFP121/cours4_swing_mvc.jar**

Une nouvelle vue, un nouveau contrôle, critiques






```
public class MainNombreMVC{

  public static void main(String[] args){
    Nombre nombre = new Nombre(0,10); // le modèle

    Vue1 vue1 = new Vue1(nombre);
    ControleVue1 controle1 = new ControleVue1(nombre, vue1);

    Vue2 vue2 = new Vue2(nombre);
    ControleVue2 controle2 = new ControleVue2(nombre, vue2);

  }
}
```

l'utilisateur peut maintenant modifier la valeur …

ajout d'un observateur auprès d'un JTextField
soit jTextField.addActionListener…

La classe Vue2, est un observateur

```
public class Vue2 extends JFrame implements Observer{

  private JTextField  valeur;
  private JSlider     jauge;

  public Vue2(Nombre nombre) {
    super("Vue2 Nombre");
    nombre.addObserver(this);
    Container content = getContentPane();
```

en quelques lignes …

```
  }

  public void update(Observable obs, Object arg){
    if(obs instanceof Nombre){  // instanceof prévention …
      Nombre n = (Nombre)obs;
      this.valeur.setText(n.toString());
      this.jauge.setValue(n.getValeur());
    }
  }
  public JTextField getValeur(){return valeur;}}
```

Le Contrôle2 de la vue 2, implements XXXXListener

```
public class ControleVue2{
  private Nombre nombre;
  private Vue2   vue2;

  public ControleVue2(Nombre nombre, Vue2 v){
    this.nombre = nombre;
    this.vue2 = v;

    vue2.getValeur().addActionListener(
      new ActionListener(){
        public void actionPerformed(ActionEvent ae){
          try{
            int valeur = Integer.parseInt(vue2.getValeur().getText());
            ControleVue2.this.nombre.setValeur(valeur);
          }catch(NumberFormatException nfe){
          }

        }
      });
  }
}
```

Une nouvelle vue engendre ici une nouvelle classe de contrôle, critiques ?
Faire autrement ? mieux ?

Deux nombres … 4 vues, 4 Contrôles

```
public class MainNombreMVC{

  public static void main(String[] args){
    Nombre nombre = new Nombre(0,10); // le modèle
    Vue1 vue1 = new Vue1(nombre);
    ControleVue1 controle1 = new ControleVue1(nombre, vue1);
    Vue2 vue2 = new Vue2(nombre);
    ControleVue2 controle2 = new ControleVue2(nombre, vue2);
    Vue1 vue11 = new Vue1(nombre);
    ControleVue1 controle11 = new ControleVue1(nombre, vue11);

    Nombre nombreBis = new Nombre(0,100);
    Vue1 vueBis = new Vue1(nombreBis);
    ControleVue1 controle = new ControleVue1(nombreBis, vueBis);

  }
}
```

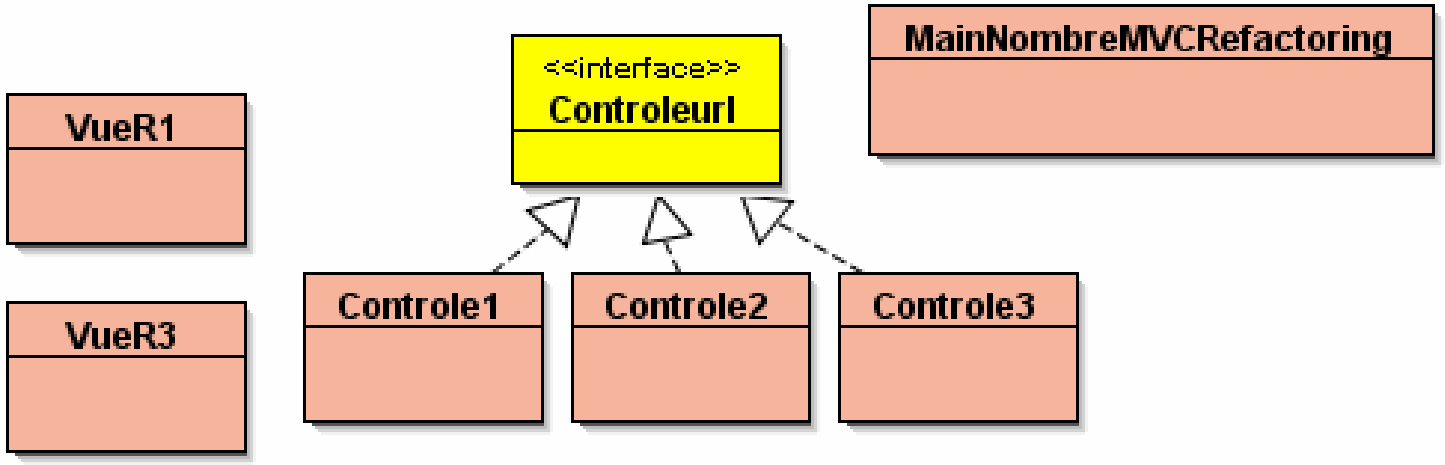
Démonstration …

Un dernier exemple : deux nombres … 8 vues, 25 Contrôles, non merci

Critiques de l'architecture

- **Vue et contrôleur sont fortement couplés**
  - **Serait-ce un oubli d'utilisation de patron ?**
- **Proposition:**
  - **Patron Stratégie pour le contrôleur**
    - **Une interface ControleurI**
      - **Les opérations sur un nombre**
    - **Plusieurs implémentations**
      - **Plusieurs stratégies concrètes**
- **Refactoring : pour un couplage faible de VC** (M ne change pas)

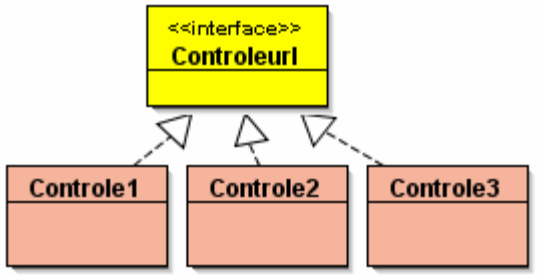
Architecture




```
public class MainNombreMVCRefactoring{

  public static void main(String[] args){
    Nombre nombre = new Nombre(0,10); // le modèle

    Controle1 controle1 = new Controle1(nombre);
    VueR1 vue1 = new VueR1(nombre, controle1);
    VueR1 vue2 = new VueR1(nombre, controle1);

    VueR3 vue3 = new VueR3(nombre, controle1);
  }
}
```

Patron Stratégie

```
public interface ControleurI{

  public void inc();
  public void dec();
  public void setValeur(int valeur);
}
```




```
public class Controle1 implements ControleurI{
  private Nombre nombre;

  public Controle1(Nombre nombre){
    this.nombre = nombre;
  }
  public void inc(){
    nombre.inc();
  }
  public void dec(){
    nombre.dec();
  }
  public void setValeur(int i){
    nombre.setValeur(i);
  }
}
```

Le Contrôleur s'est affranchi de la Vue

La Vue utilise la stratégie transmise

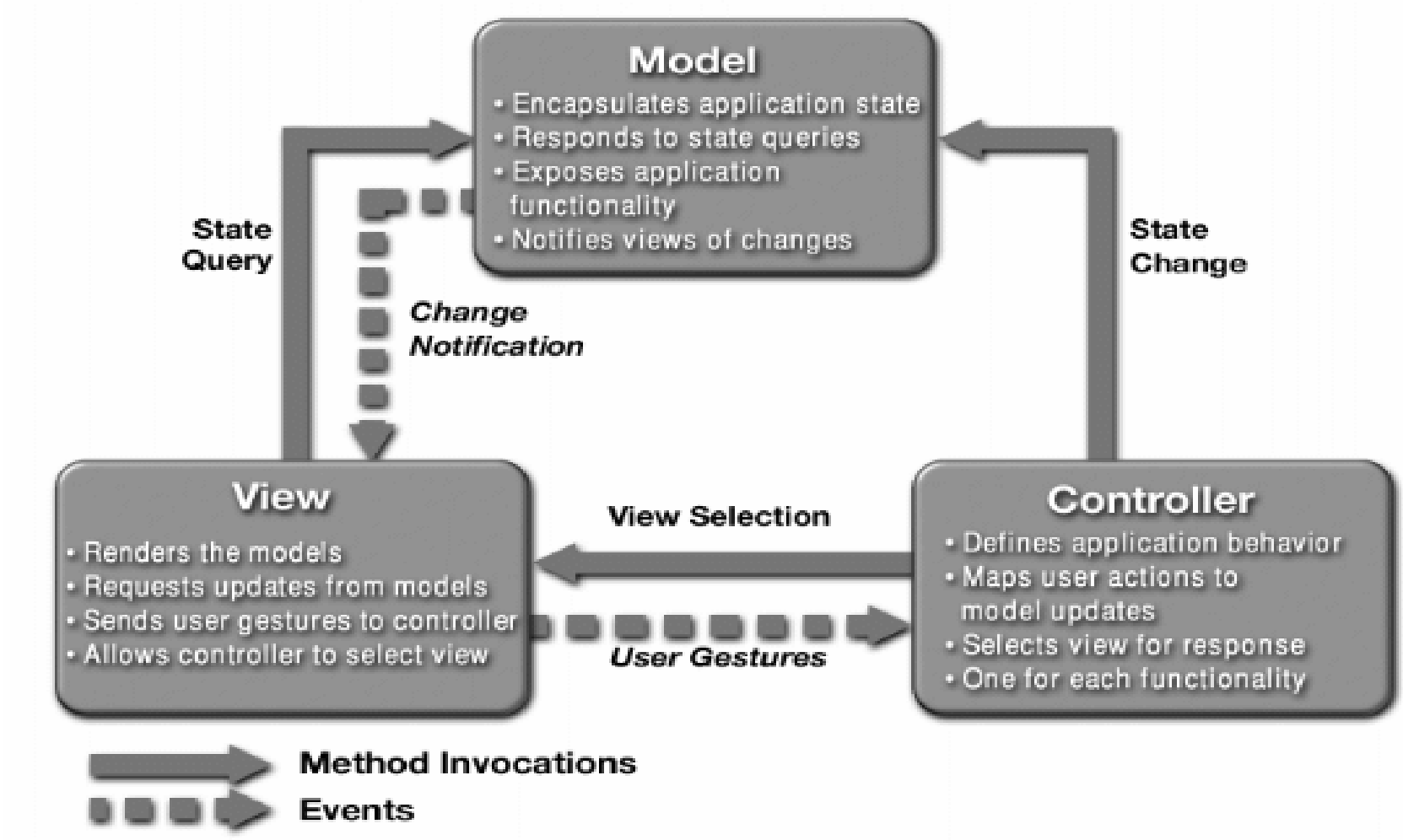
```
public class Vue1R extends JFrame implements Observer{
  private JTextField  valeur;
  private JSlider     jauge;
  private ControleurI controleur;

  public Vue1R(Nombre nombre, ControleurI controleur) {
    super("Vue1 Refactoring");
    nombre.addObserver(this);
    this.controleur = controleur;
    Container content = getContentPane();
```

en quelques lignes …

```
 this.moins.addActionListener(
     new ActionListener(){
       public void actionPerformed(ActionEvent ae){
         VueR1.this.controleur.dec();
       }
     });
   ….
```

Démonstration

Discussion / l'exemple



- **Remarquons que : View Selection**
  - **Est absent de l'exemple**

Discussion sur l'implémentation du Modèle

- **public class Nombre …**
- **Le Modèle est une classe**
  - **Cette classe hérite de java.util.Observable sur cet exemple**
- **Ce modèle peut devenir un composant logiciel**
  - **Un JavaBean**
  - **Peu de différences à part quelques règles d'écritures**
    - **Ces règles permettront une utilisation de ce composant par des outils d'introspection,**
    - **Vers une meilleure réutilisation ?**
    - **Vers une industrie du composant logiciels ?**
      - **EJB ? Enterprise JavaBeans**

        » Un nouveau langage au dessus de Java?

Observable/Observer *à la mode* JavaBeans

- **Paquetage java.beans**
- **Champs d'instance**
  - **int valeur comme property**
- **Observer comme PropertyChangeListener**
  - **public void update(Observable obs, Object o)**

  comme
  - **public void propertyChange(PropertyChangeEvent evt)**
- **Observable comme PropertyChangeSupport**
  - **void notifyObservers (Object o)**

  comme
  - **void firePropertyChange (String property,Object oldValue,Object newValue)**

Un bean

- **Un Bean est avant tout une classe …**
  - ***Un bean est un (extends) POJO, une classe quelconque***

Avec

- **Le respect de certaines conventions**
  - **implements Serializable**
  - **Un constructeur par défaut**
  - **Un *getter* et/ou un *setter* pour chaque variable d'instance**
  - **firePropertyChange au sein du setter**

  - **Simple n'est-ce pas ?**

- **La suite**
  - **Nombre devient Bean**
  - **La BeanBox, l'outil d'assemblage de Beans : un outil historique**
  - **Introspection systématique**

La classe Nombre devient Bean : NombreBean

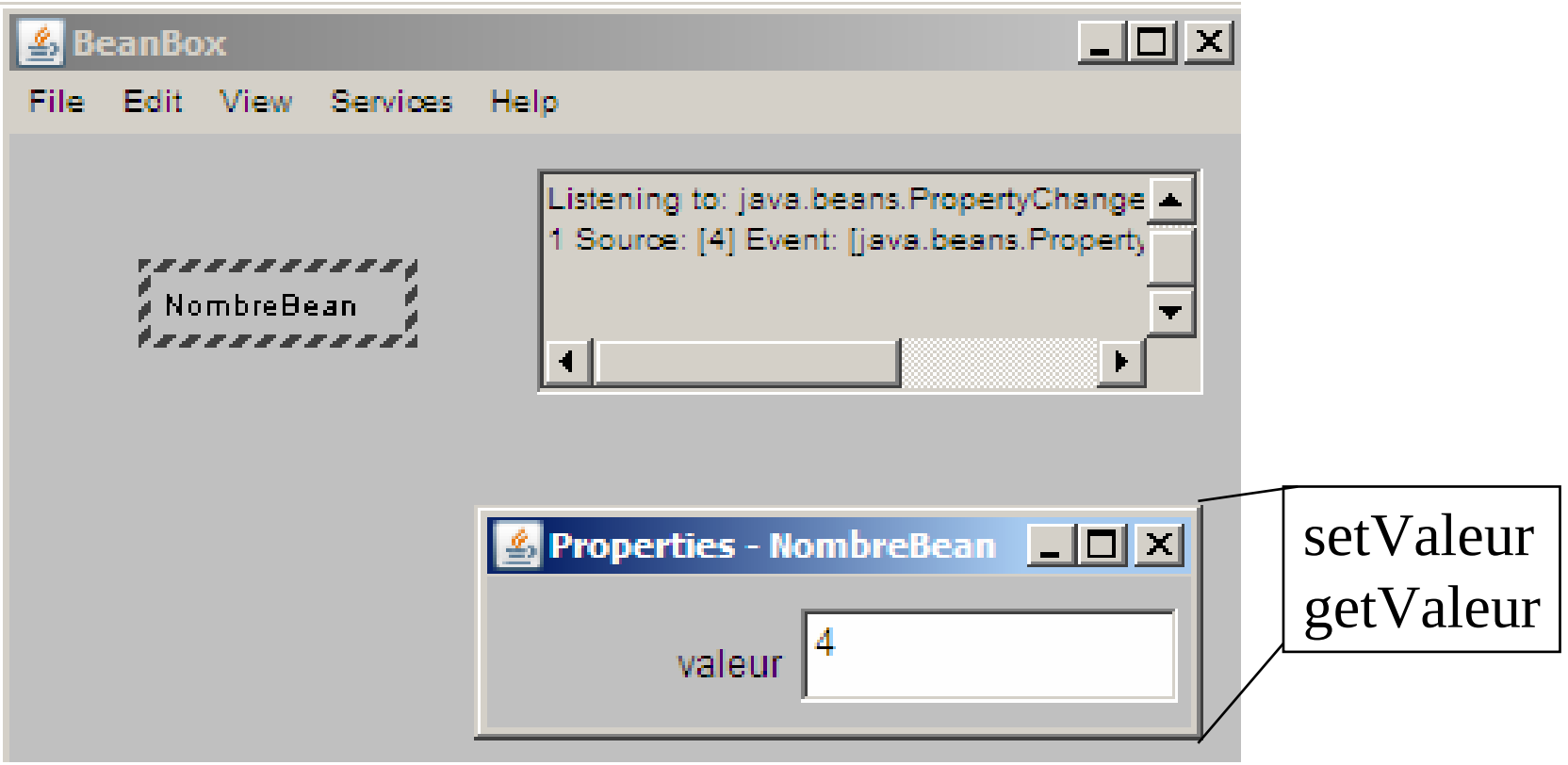
```
public class NombreBean implements Serializable{

  public final int VALEUR_MIN;
  public final int VALEUR_MAX;
  private int valeur;
  private PropertyChangeSupport propertySupport;

  public NombreBean(){
   this.VALEUR_MIN = this.valeur = 0;
   this.VALEUR_MAX = 10;
   this.propertySupport = new PropertyChangeSupport(this);
  }

  public void inc(){
    if(valeur < VALEUR_MAX){
      int old = valeur;
      this.valeur++;
      propertySupport.firePropertyChange("valeur",old,valeur);
    }
  }

  public void addPropertyChangeListener(PropertyChangeListener l) {
    propertySupport.addPropertyChangeListener(l);
  }
     // public int getValeur()  et setValeur()
```

La Vue est à l'écoute de son Bean

```
public class Vue1Bean extends JFrame implements PropertyChangeListener{

  private JButton     plus;
  private JButton     moins;
  private JTextField  valeur;
  private JSlider     jauge;

  public Vue1Bean(NombreBean nombre) {
    super("Vue1 Nombre");
    nombre.addPropertyChangeListener(this);

   // Le dessin ici
  }

  @Override
  public void propertyChange(PropertyChangeEvent evt){
    assert evt.getPropertyName().equals("valeur");
    this.valeur.setText(evt.getNewValue().toString());
    this.jauge.setValue((Integer)evt.getNewValue());
  }
```

Les contrôles ne changent pas, le MVC reste en l'état

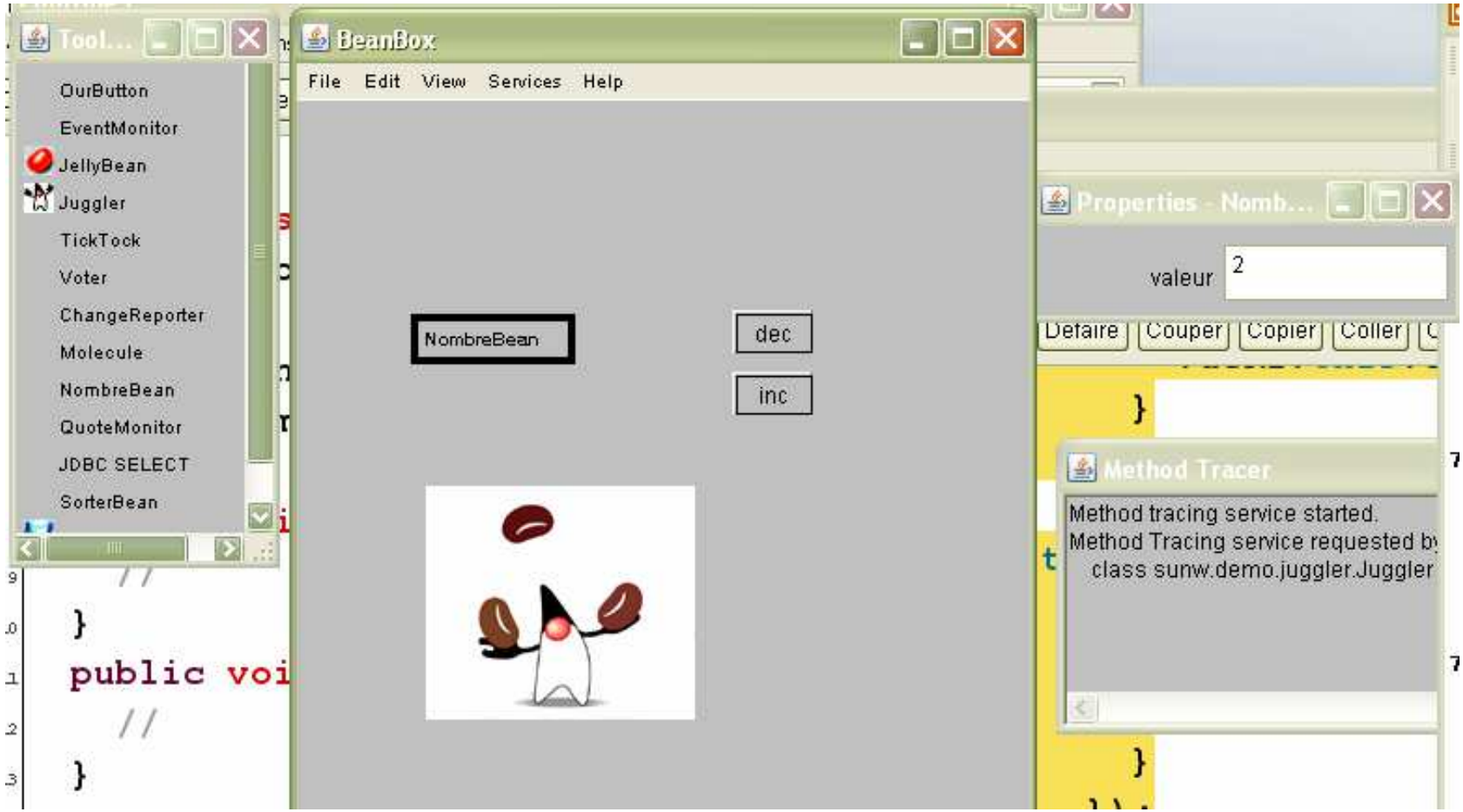
NombreBean intègre la BeanBox



- **Ajout par l'outil BeanBox d'un listener (EventMonitor)**
  à chaque changement de valeur de *valeur* …
  démonstration pour l'histoire c'est un outil du siècle dernier

Usage de l'introspection

Le jongleur de Beans



- **Attention avec BeanBox et la version du jdk…**

MVC un autre schéma, encore !, tjs d'accord ?

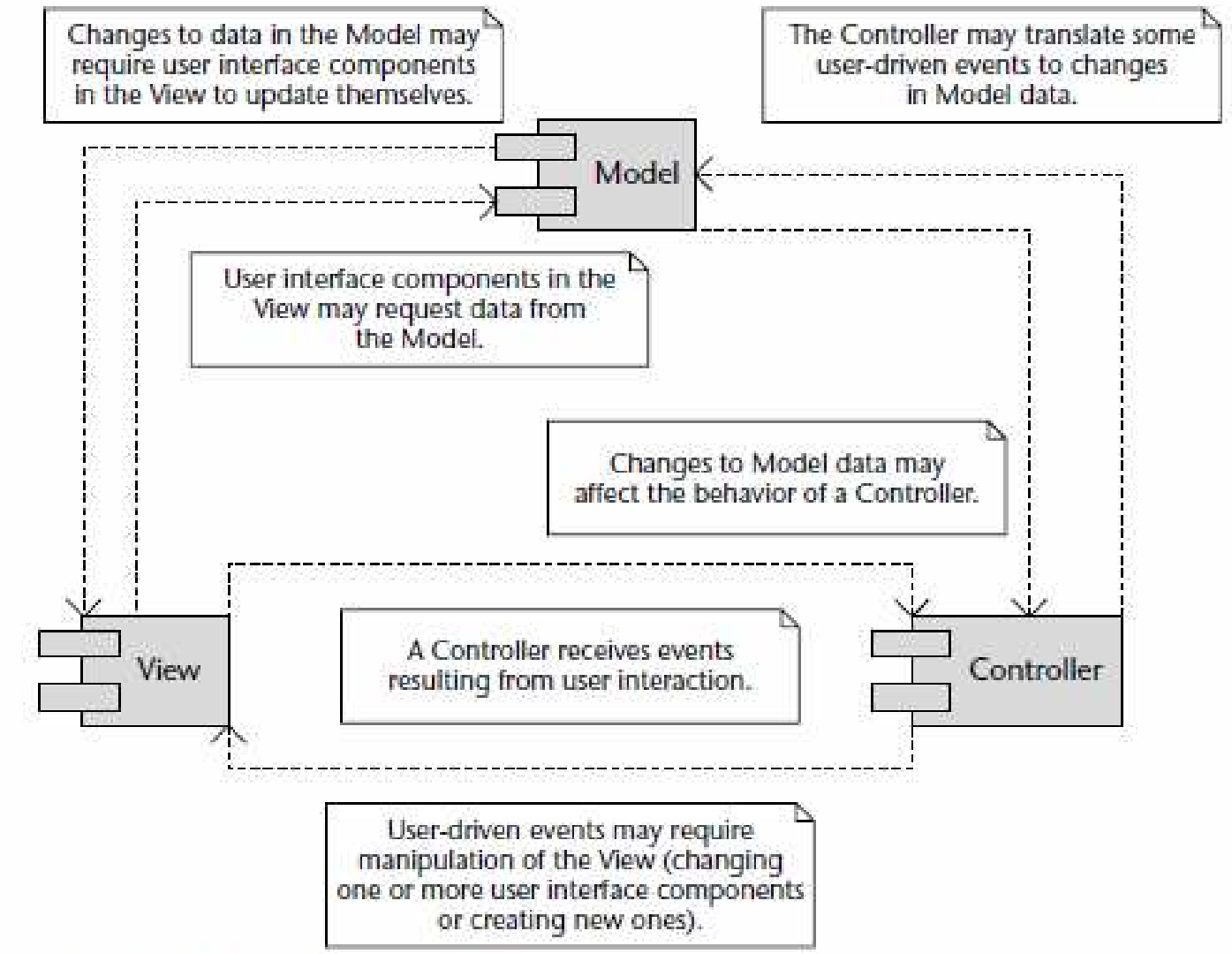
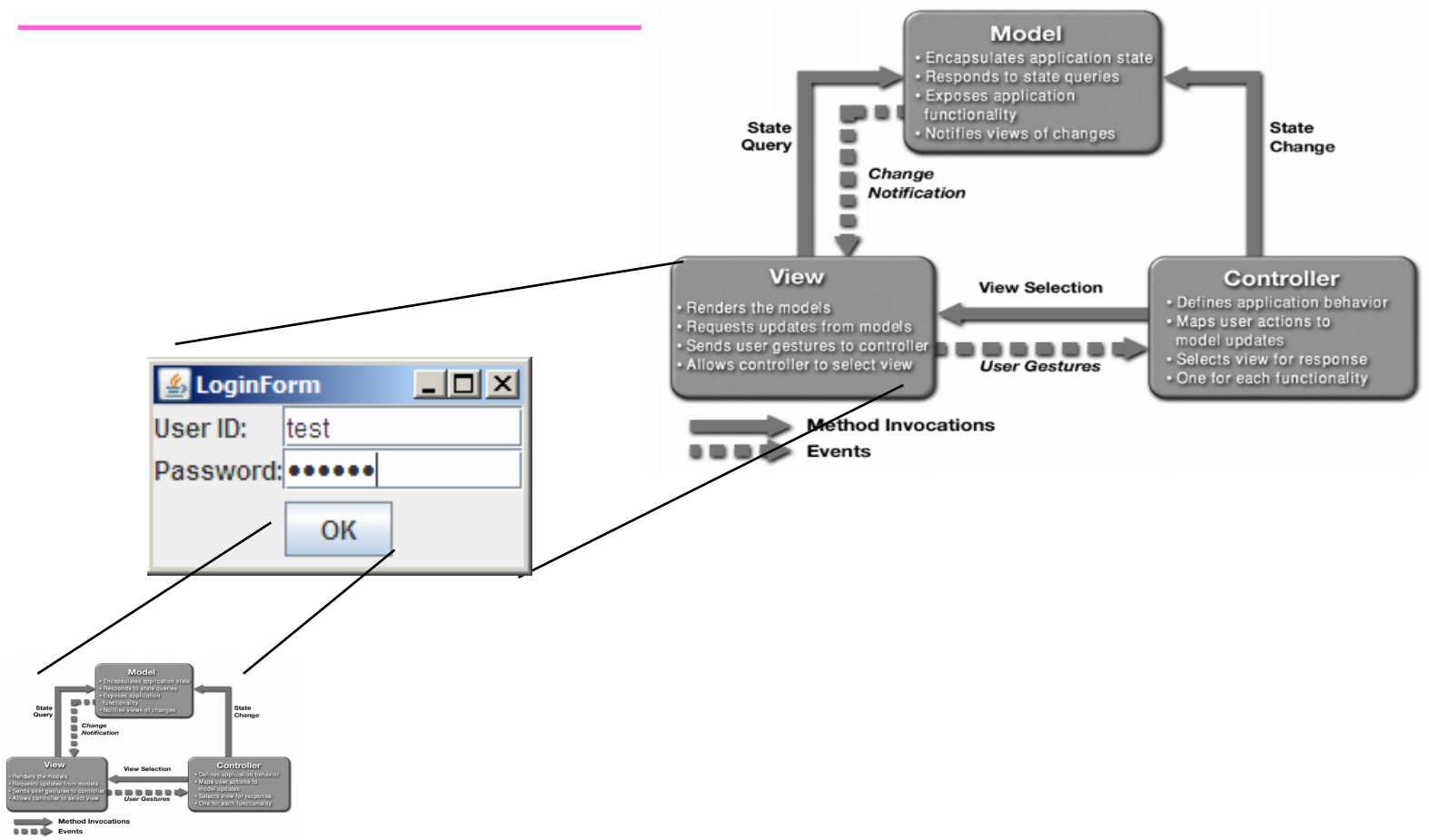


Figure 1.1 MVC Pattern structure.

src : Mastering JSF page 6

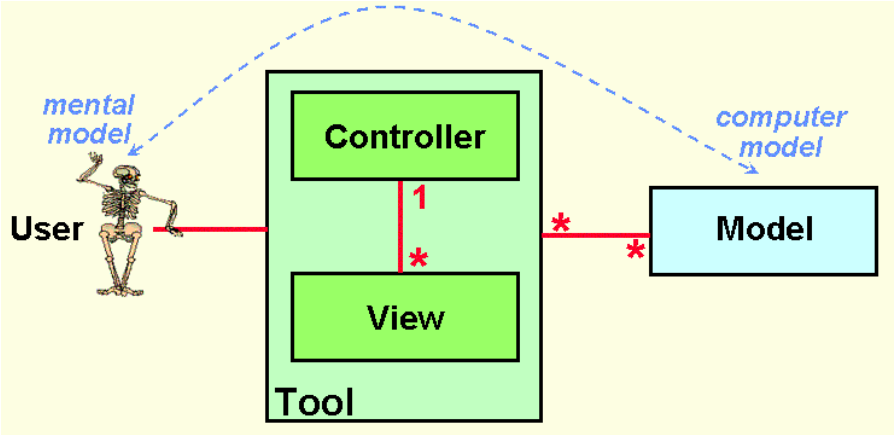
Conclusion ? discussion



- **MVC au niveau d'une application**
- **Et Il y a aussi un MVC au niveau de chaque composant swing**

MVC à l'origine

- **L'original en 1979**
- **http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html**



Donner l'illusion à l'utilisateur de manipuler les données du modèle

Conclusion, discussion

- **MVC**
  - **Mature**

La suite

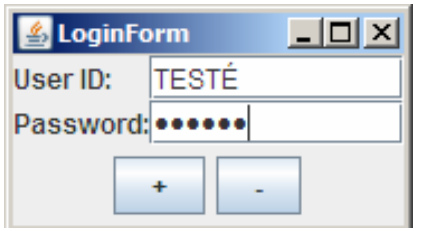
- **MVC et composant swing**
  - **Certains composants proposent de changer le modèle et l'IHM…**

Un Composant Swing et MVC

- **Par exemple**
  - **JTextField**
    - **Une zone de texte** *(classe JTextComponent)*
  - **Le Modèle**
    - **Le texte** *(classe PlainDocument)*
  - **L'interface Utilisateur**
    - **Un rectangle** *(classe BasicTextFieldUI)*
- **En standard**
  - **JTextField field = new JTextField();**
- **Avec un autre modèle**
  - **JTextField field = new JTextField (new unAutreModèleDeDocument());**
- **Avec une autre interface utilisateur (vue)**
  - **field.setUI(new uneAutreInterface());**

Un exemple, avec démo possible…

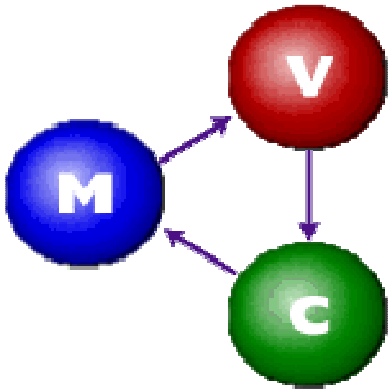
- **Le « nouveau » JTextField ne peut contenir que des majuscules !**
  - **Une contrainte sur une entrée du formulaire**
  - **Un nouveau modèle, ici une sous-classe de Document**
  - **this.userIdField = new JTextField(new UnAutreModèleDeTexte(), "", 10);**

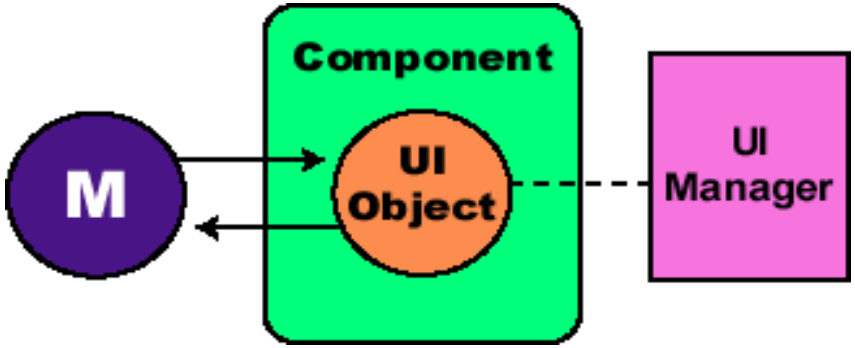



```
public class UnAutreModèleDeTexte extends PlainDocument {
        public void insertString(int offs, String str, AttributeSet a)
            throws BadLocationException {
            if (str == null) {
                return;
            }
            char[] upper = str.toCharArray();
            for (int i = 0; i < upper.length; i++) {
                upper[i] = Character.toUpperCase(upper[i]);
            }
            super.insertString(offs, new String(upper), a);
        }
}
```

Donc nous avons MVC et SMA…

- **SMA : Separable Model Architecture**





- **ou bien**

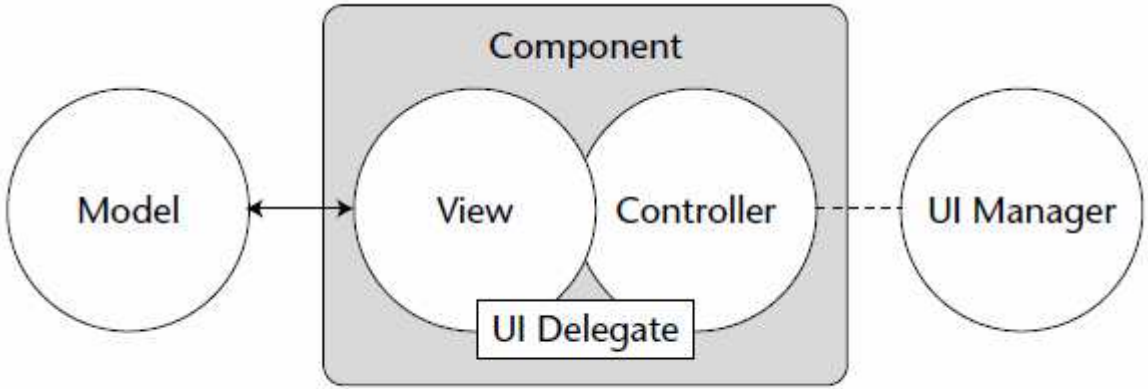
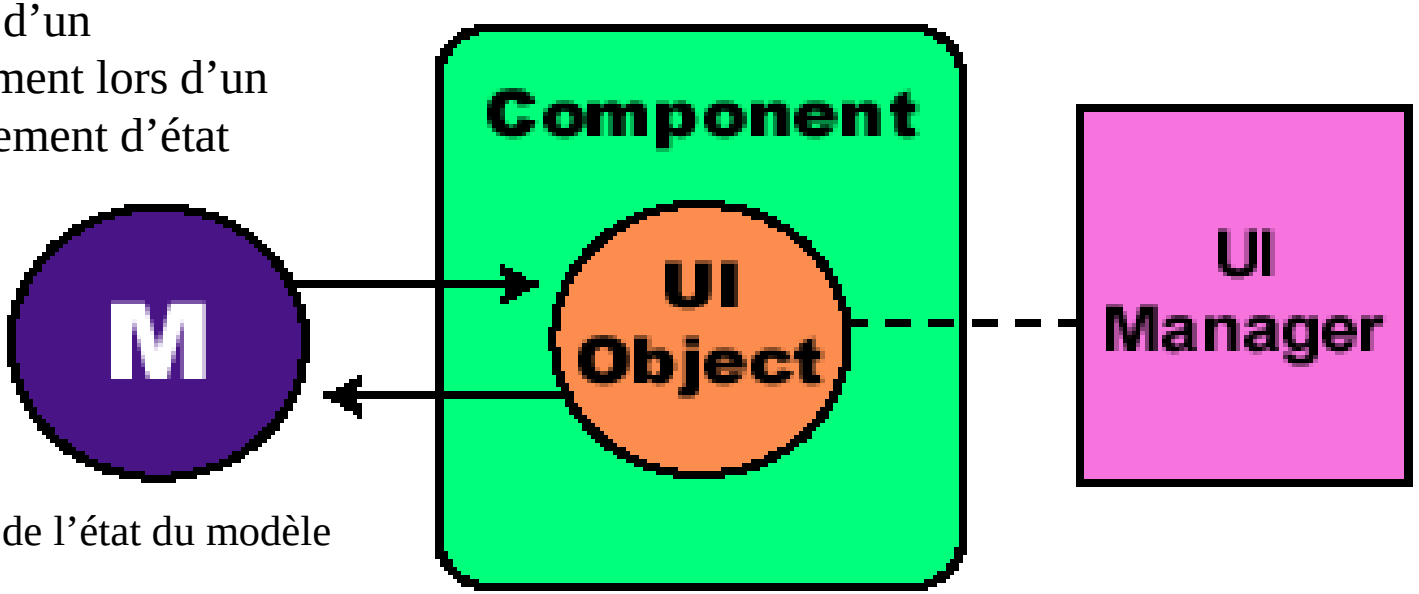


Figure 1.3 Swing's modified MVC.

SMA

Envoi d'un événement lors d'un changement d'état



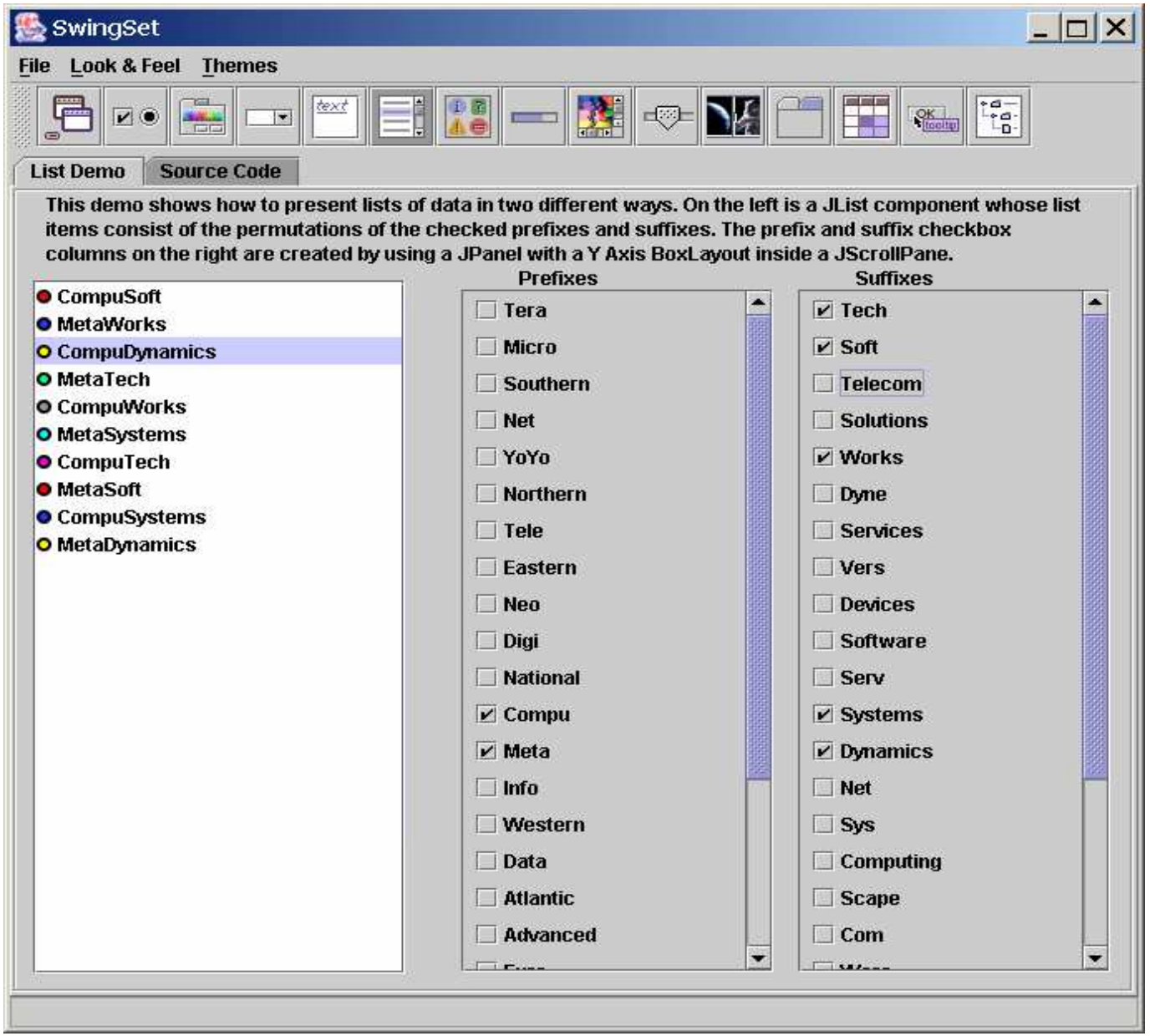
Lecture de l'état du modèle

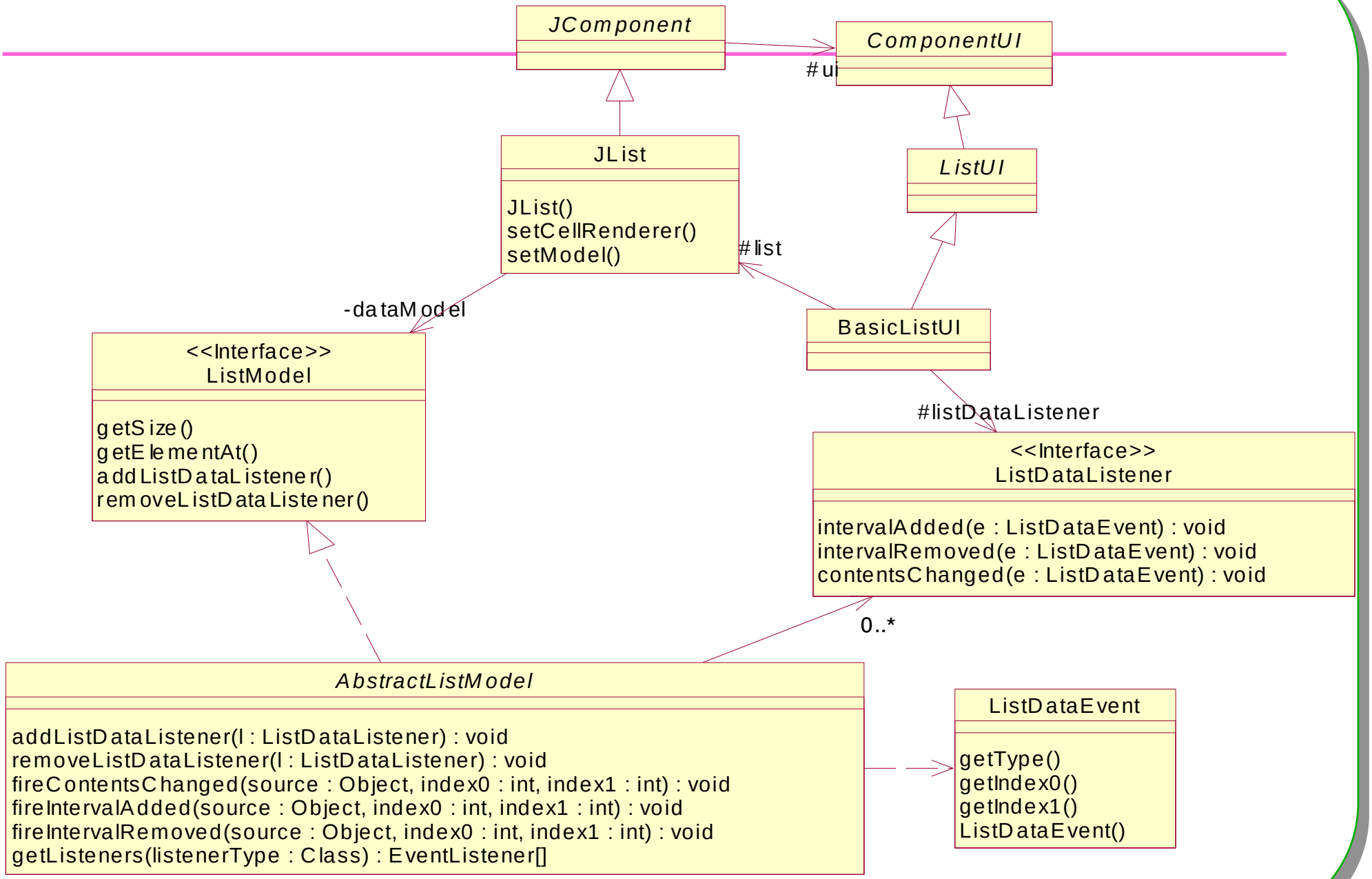
Component	Model Interface	Model Type
JButton	ButtonModel	GUI
JToggleButton	ButtonModel	GUI/data
JCheckBox	ButtonModel	GUI/data
JRadioButton	ButtonModel	GUI/data
JMenu	ButtonModel	GUI
JMenuItem	ButtonModel	GUI
JCheckBoxMenuItem	ButtonModel	GUI/data
JRadioButtonMenuItem	ButtonModel	GUI/data
JComboBox	ComboBoxModel	data
JProgressBar	BoundedRangeModel	GUI/data
JScrollBar	BoundedRangeModel	GUI/data
JSlider	BoundedRangeModel	GUI/data
JTabbedPane	SingleSelectionModel	GUI
JList	ListModel	data
JList	ListSelectionModel	GUI
JTable	TableModel	data
JTable	TableColumnModel	GUI
JTree	TreeModel	data
JTree	TreeSelectionModel	GUI
JEditorPane	Document	data
JTextPane	Document	data
JTextArea	Document	data
JTextField	**Document**	**data**
JPasswordField	Document	data

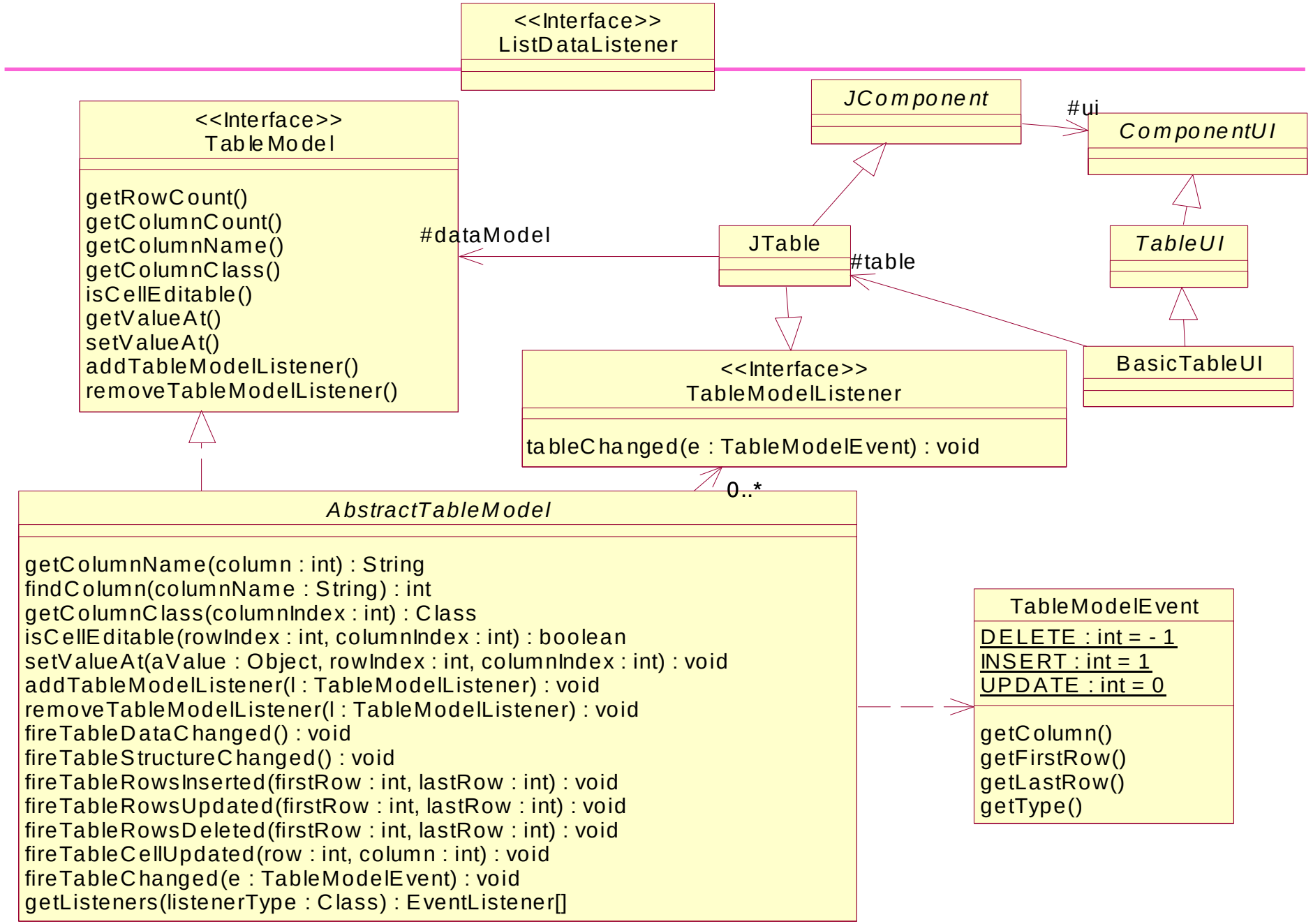
« Model » et Notification…

Model	Listener	Event
ListModel	ListDataListener	ListDataEvent
ListSelectionModel	ListSelectionListener	ListSelectionEvent
ComboBoxModel	ListDataListener	ListDataEvent
TreeModel	TreeModelListener	TreeModelEvent
TreeSelectionModel	TreeSelectionListener	TreeSelectionEvent
TableModel	TableModelListener	TableModelEvent
TableColumnModel	TableColumnModel-Listener	TableColumnModel-Event
Document	DocumentListener	DocumentEvent
Document	UndoableEditListener	UndoableEditEvent

Jlist and ListModel…exemple fourni.. À suivre




JComponent
ComponentUI
#ui
JList
JList()
setCellRenderer()
setModel()
ListUI
#list
-dataModel
BasicListUI
<<Interface>>
ListModel
getSize()
getElementAt()
addListDataListener()
removeListDataListener()
#listDataListener
<<Interface>>
ListDataListener
intervalAdded(e : ListDataEvent) : void
intervalRemoved(e : ListDataEvent) : void
contentsChanged(e : ListDataEvent) : void
0..*
AbstractListModel
addListDataListener(l : ListDataListener) : void
removeListDataListener(l : ListDataListener) : void
fireContentsChanged(source : Object, index0 : int, index1 : int) : void
fireIntervalAdded(source : Object, index0 : int, index1 : int) : void
fireIntervalRemoved(source : Object, index0 : int, index1 : int) : void
getListeners(listenerType : Class) : EventListener[]
ListDataEvent
getType()
getIndex0()
getIndex1()
ListDataEvent()


<<Interface>>
ListDataListener
<<Interface>>
TableModel
getRowCount()
getColumnCount()
getColumnName()
getColumnClass()
isCellEditable()
getValueAt()
setValueAt()
addTableModelListener()
removeTableModelListener()
#dataModel
JComponent
#ui
ComponentUI
JTable
#table
TableUI
BasicTableUI
<<Interface>>
TableModelListener
tableChanged(e : TableModelEvent) : void
0..*
AbstractTableModel
getColumnName(column : int) : String
findColumn(columnName : String) : int
getColumnClass(columnIndex : int) : Class
isCellEditable(rowIndex : int, columnIndex : int) : boolean
setValueAt(aValue : Object, rowIndex : int, columnIndex : int) : void
addTableModelListener(l : TableModelListener) : void
removeTableModelListener(l : TableModelListener) : void
fireTableDataChanged() : void
fireTableStructureChanged() : void
fireTableRowsInserted(firstRow : int, lastRow : int) : void
fireTableRowsUpdated(firstRow : int, lastRow : int) : void
fireTableRowsDeleted(firstRow : int, lastRow : int) : void
fireTableCellUpdated(row : int, column : int) : void
fireTableChanged(e : TableModelEvent) : void
getListeners(listenerType : Class) : EventListener[]
TableModelEvent
DELETE : int = - 1
INSERT : int = 1
UPDATE : int = 0
getColumn()
getFirstRow()
getLastRow()
getType()

Conclusion