

Chapitre V

# Interfaces graphiques

---

Swing

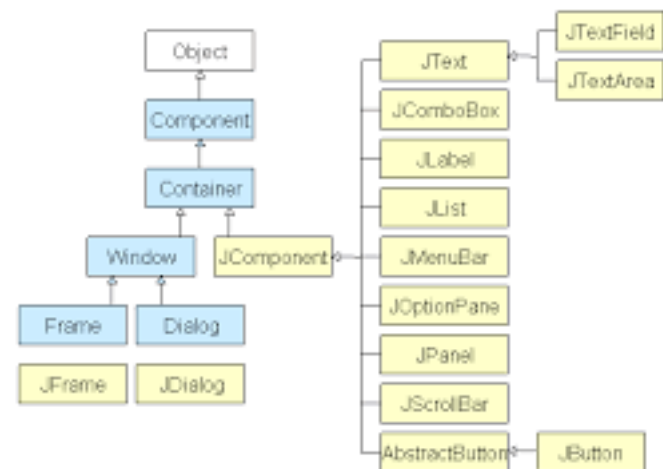
# Principes de base

---

- Dans un système d'interface graphique, le programme doit réagir aux interactions avec l'utilisateur
- Les interactions génèrent des « événements » qui provoquent l'exécution de code
- Le programme est dirigé par les événements (event-driven)

# Principes de base

- Des composants graphiques  
(exemple: JFrame, JButton ...)
  - Hiérarchie de classes
- Placement des composants graphiques
- Des événements et les actions à effectuer  
(exemple: presser un bouton)
- (Et d'autres choses...)



# Principes de base

---

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

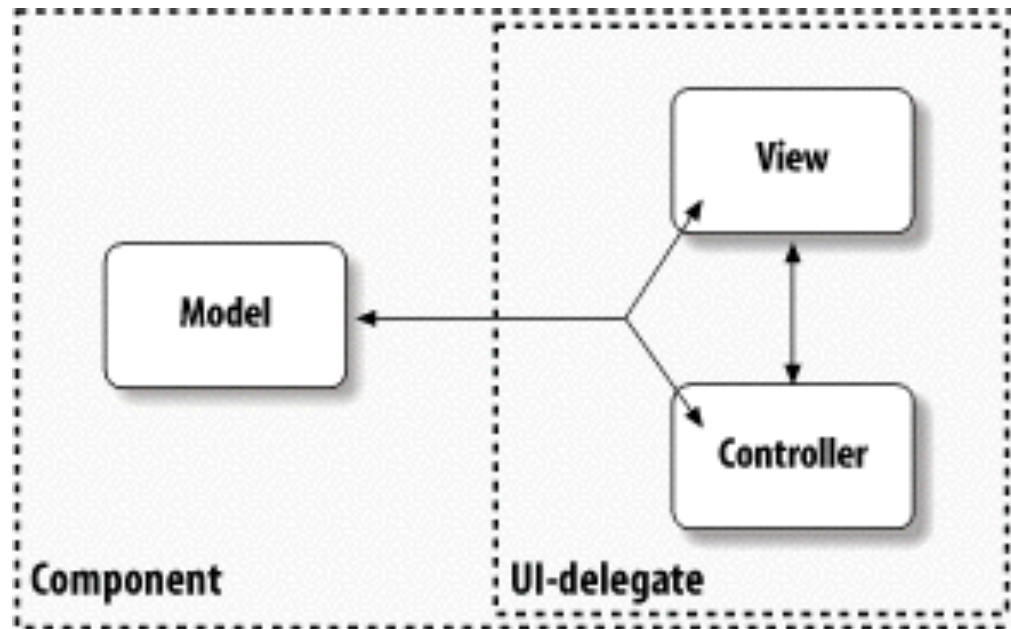


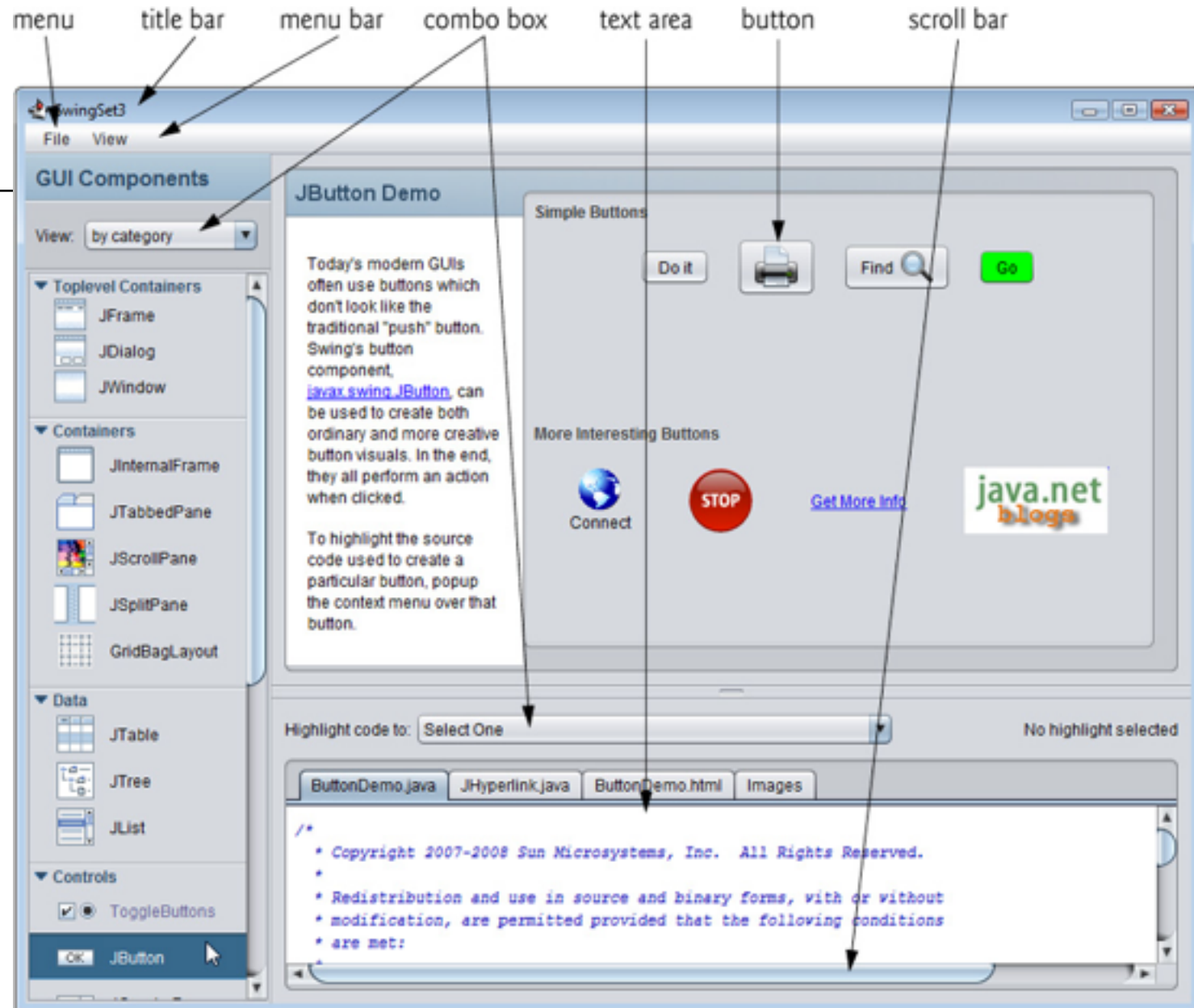
# Composants: JComponent

---

# Composants

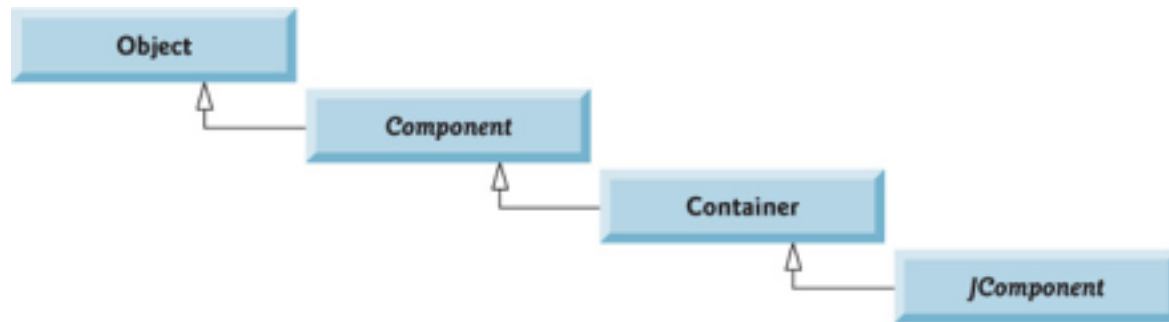
- Modèle Vue Contrôleur





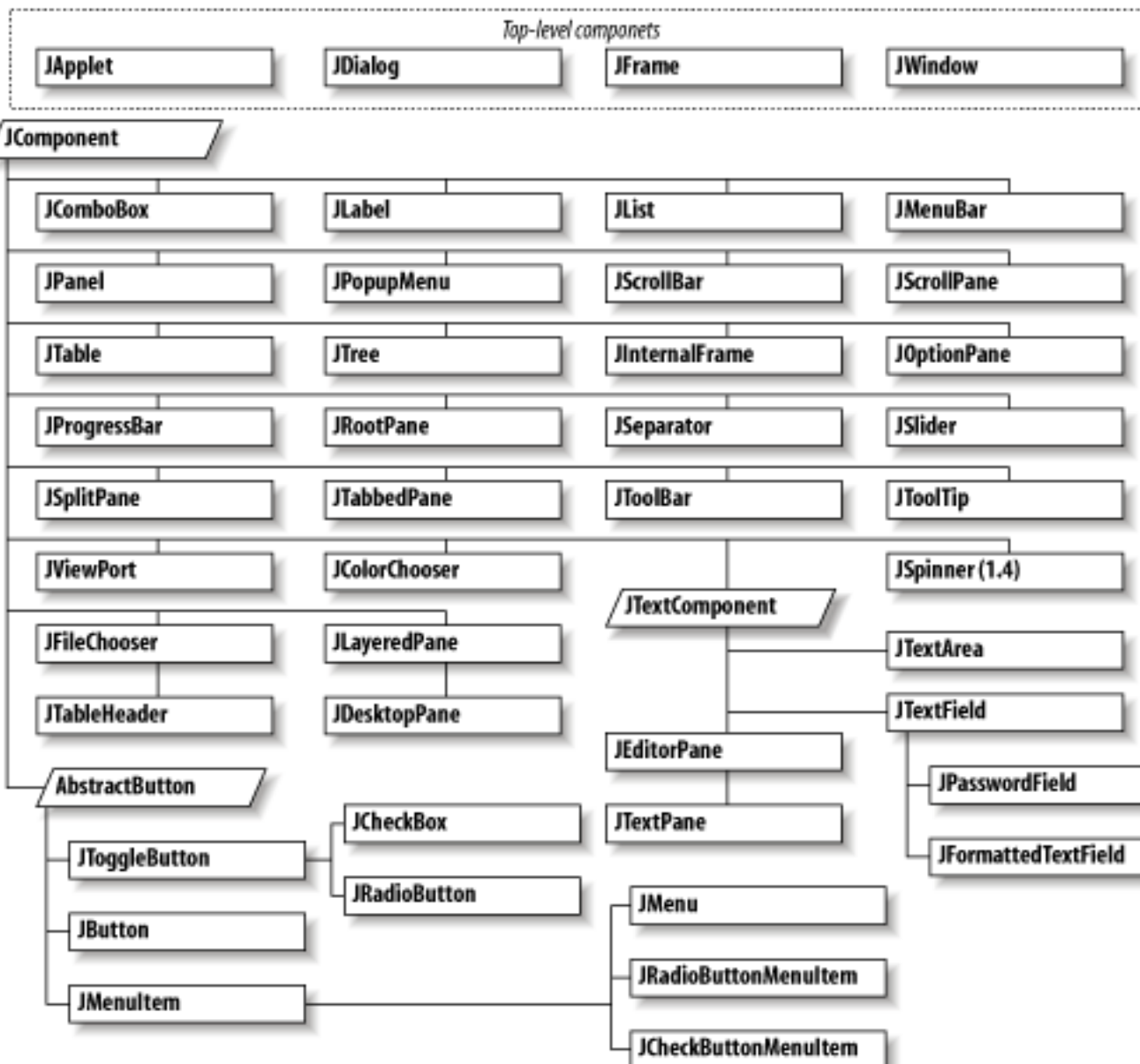
# Hiérarchie:

---



- les composants Swing  
descendent de JComponent (classe  
abstraite)



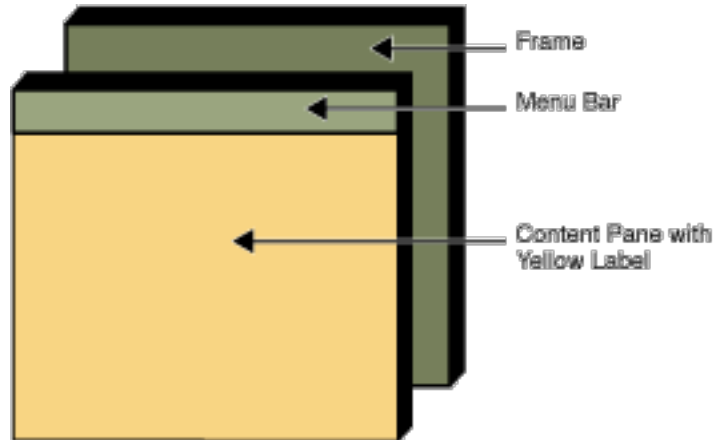
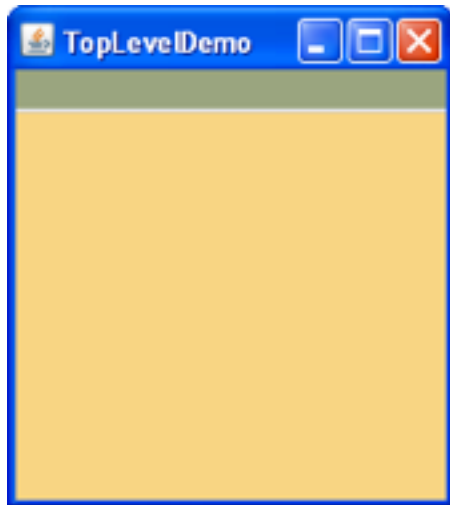


# Afficher... top-level

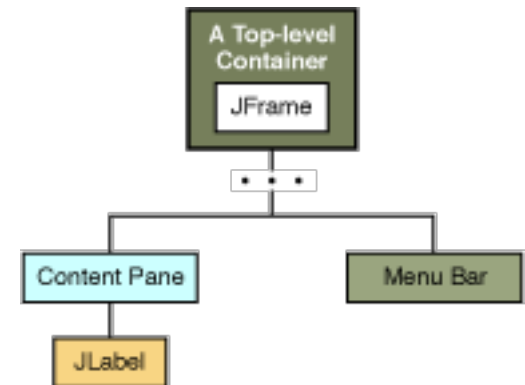
---

- Pour pouvoir être affiché, il faut que le composant soit dans un top-level conteneur:  
(JFrame, JDialog et JApplet)
- Hiérarchie des composants: arbre racine top-level

# Exemple



- Correspond à la hiérarchie:



# Le code

---

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

public class TopLevel {
    /**
     * Affiche une fenêtre JFrame top level
     * avec une barre de menu JMenuBar verte
     * et un JLabel jaune
     */
    private static void afficherMaFenetre() {
        //créer la JFrame
        //créer la JMenuBar
        //créer le JLabel
        // mettre le JMenuBar et le JLabel dans la JFrame
        //afficher la JFrame
    }
}
```

# Le code

---

```
//Créer la JFrame
JFrame frame = new JFrame("TopLevelDemo");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
//Créer la JMenuBar
JMenuBar greenMenuBar = new JMenuBar();
greenMenuBar.setOpaque(true);
greenMenuBar.setBackground(new Color(0, 200, 0));
greenMenuBar.setPreferredSize(new Dimension(200, 20));
//Créer le JLabel
JLabel yellowLabel = new JLabel();
yellowLabel.setOpaque(true);
yellowLabel.setBackground(new Color(250, 250, 0));
yellowLabel.setPreferredSize(new Dimension(200, 180));
//mettre la JMenuBar et position le JLabel
frame.setJMenuBar(greenMenuBar);
frame.getContentPane().add(yellowLabel, BorderLayout.CENTER);
//afficher...
frame.pack();
frame.setVisible(true);
```

# Et le main

---

```
public class TopLevel { //afficherMaFenetre()
    public static void main(String[] args) {
        javax.swing.SwingUtilities.invokeLater(
            new Runnable() {
                public void run() {
                    afficherMaFenetre();
                }
            });
    }
}
```



# Événements

---

# Événements: principes

---

- Dans un système d'interface graphique:
  - Quand l'utilisateur presse un bouton, un "événement" est posté et va dans une boucle d'événements
  - Les événements dans la boucle d'événements sont transmis aux applications qui se sont enregistrées pour écouter.



# Événements

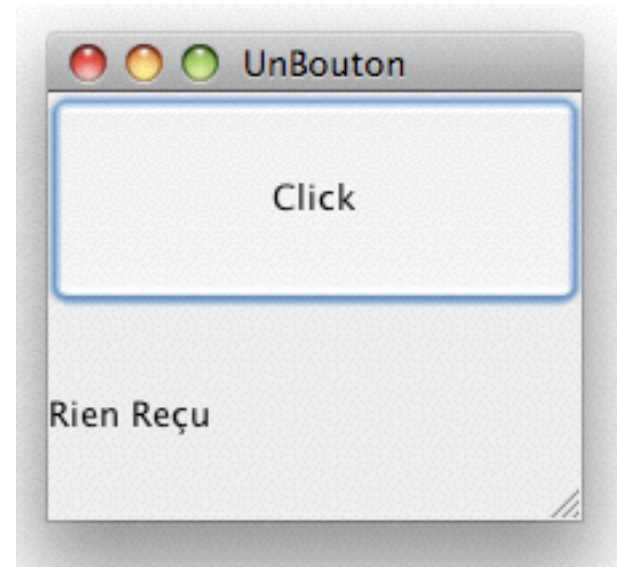
---

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

# Un exemple

---

- Un bouton qui réagit



# Le code:

---

- Un JButton
- Un JLabel
- Implementer ActionListener
  - actionPerformed définit ce qui se passe quand le bouton est cliqué
- Placer le bouton et le label

# Code:

---

```
import java.awt.*;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JButton;
import javax.swing.JComponent;
import java.awt.Toolkit;
import java.awt.BorderLayout;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import javax.swing.JLabel;

public class UnBouton extends JPanel implements ActionListener {
    JButton bouton;
    String contenu="Rien Reçu";
    JLabel label=new JLabel(contenu);
    int cmp=0;
    public UnBouton() { //...}
    public void actionPerformed(ActionEvent e) {//...}
    private static void maFenetre(){//...}
    public static void main(String[] args) {//...}
}
```

# Code

---

```
public UnBouton() {
    super(new BorderLayout());
    bouton = new JButton("Click");
    bouton.setPreferredSize(new Dimension(200, 80));
    add(bouton, BorderLayout.NORTH);
    label = new JLabel(contenu);
    label.setPreferredSize(new Dimension(200, 80));
    add(label, BorderLayout.SOUTH);
    bouton.addActionListener(this);
}

public void actionPerformed(ActionEvent e) {
    Toolkit.getDefaultToolkit().beep();
    label.setText("clické " + (++cmp) + " fois");
}
```

# Code

---

```
private static void maFenetre() {
    JFrame frame = new JFrame("UnBouton");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    JComponent newContentPane = new UnBouton();
    newContentPane.setOpaque(true);
    frame.setContentPane(newContentPane);
    frame.pack();
    frame.setVisible(true);
}
public static void main(String[] args) {
    //Formule magique
    javax.swing.SwingUtilities.invokeLater(new Runnable() {
        public void run() {
            maFenetre();
        }
    });
}
```

# Variante

---

```
public class UnBoutonBis extends JPanel {  
    //...  
    bouton.addActionListener(new ActionListener() {  
        public void actionPerformed(ActionEvent e) {  
            Toolkit.getDefaultToolkit().beep();  
            label.setText("clické " + (++cmp) + " fois");  
        } });  
    }  
    //...  
}
```

# Compléments

---

- AWT-Swing
- Multithreading



# Préliminaires...

---

- Lightweight et heavyweight composants
  - Dépendent ou non du système d'interface graphique
    - Lightweight écrit en Java et dessiné dans un heavyweight composant- indépendant de la plateforme (swing)
    - Les heavyweight composants s'adressent directement à l'interface graphique du système (awt)
  - (certaines caractéristiques dépendent du « look and feel »).

# Plus précisément

---

- Swing prend en charge la gestion des composants qui sont dessinés en code Java (lightweight)
- Les composants AWT sont eux liés aux composants natifs (heavyweight)
- Swing dessine le composant dans un canevas AWT et utilise le traitement des événements de AWT

# Look and feel

---

- Look and feel:

Possibilité de choisir l'apparence de l'interface graphique.

UIManager gère l'apparence de l'interface

```
public static void main(String[] args) {  
    try {  
        UIManager.setLookAndFeel(  
            UIManager.getCrossPlatformLookAndFeelClassName());  
    } catch (Exception e) { }  
  
    new SwingApplication(); //Create and show the GUI.  
}
```

# Multithreading

---

- Attention au « modèle, contrôleur, vue » en cas de multithreading:
  - Tous les événements de dessin de l'interface graphiques sont dans une unique file d'event-dispatching dans une seule thread.
  - La mise à jour du modèle doit se faire tout de suite après l'événement de visualisation dans cette thread.

# Suite

## Les threads

- Main application thread
- Toolkit thread
- Event dispatcher thread
- Toutes Les opérations d'affichage ont lieu dans une seule thread l'EDT



# Principes

---

- Une tâche longue ne doit pas être exécutée dans l'EDT
- Un composant Swing doit s'exécuter dans l'EDT
  - threads initiales (code de l'application de départ)
  - event dispatcher thread: les événements
  - « workers » threads les tâches qui consomment du temps

# Exemple

---

```
public void actionPerformed(ActionEvent e{  
    try {  
        Thread.sleep(4000);  
    } catch (InterruptedException e) { }  
}
```

Provoque une interruption de l'affichage pendant 4 secondes

# Une solution

---

```
public void actionPerformed(ActionEvent e){
    try{
        SwingUtilities.invokeLater(new Runnable(
            {
                public void run() {
                    //opération longue
                }
            }));
    } catch (InterruptedException ie) {}
        catch (InvocationTargetException ite) {}
    }
}
```

le code sera exécuté dans la thread des événements



# Attendre le résultat:

---

```
try {
    SwingUtilities.invokeAndWait(new Runnable() {
        public void run() {
            show();
        }
    });
} catch (InterruptedException ie) {}
    catch (InvocationTargetException ite) {}
```

# workers

- La classe javax.swing.SwingWorker permet d'exécuter des tâches longues
  - `doInBackground()`: code à exécuter en background
  - `done()` sera exécuté dans Event Dispatch Thread quand `doInBackground` sera terminé
  - (il est aussi possible d'obtenir des résultats intermédiaire)



# Un premier exemple: éditeur

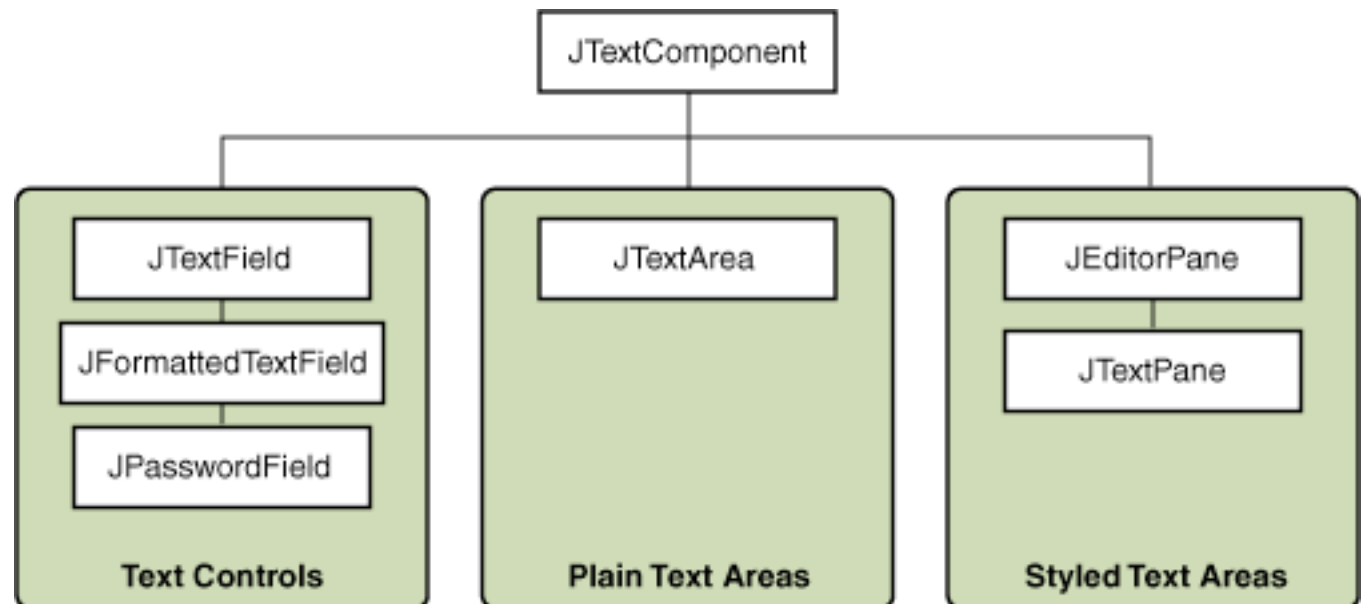
---

# Exemple JTextComponent

Modèle: document contenu du composant

Vue: l'affichage du composant sur l'écran

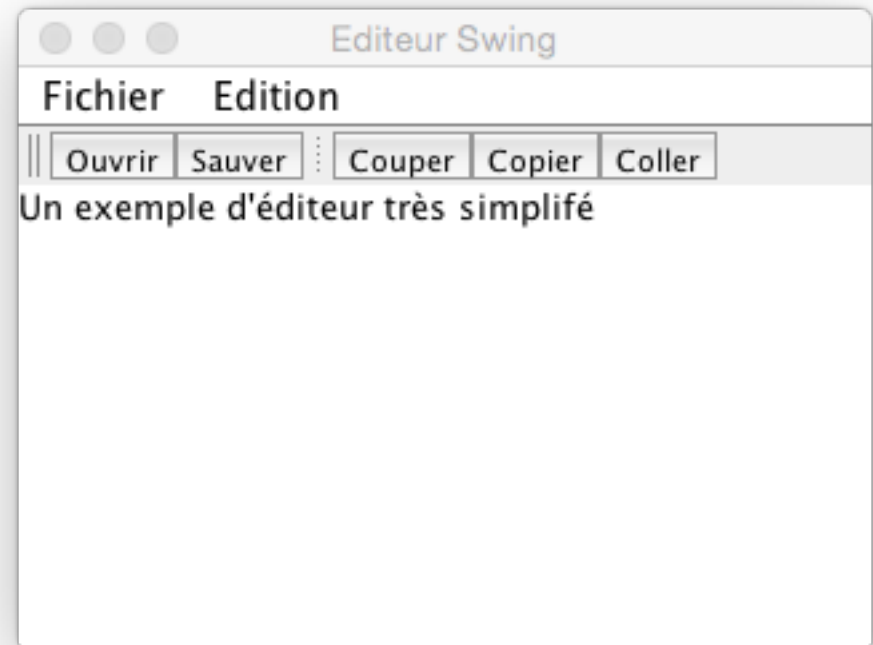
Contrôleur: editor kit avec actions



# Un éditeur simple

---

- un menu, une barre d'outil,
- actions:
  - couper, coller, copier, tout sélectionner provenant de `DefaultEditorKit`
  - ouvrir un fichier, sauvegarder un fichier avec un `JFileChooser`





# Structure globale

---

Créer un JTextArea

Définir les différentes Actions

Créer une barre de menu

associer les Actions à des boutons de la  
barre de menu

Créer un menu

associer les Actions aux éléments de menu

# SimpleEditeur

```
public SimpleEditeur() {
    super("Editeur Swing");
    textComp = createTextComponent();
    mesActions();
    Container content = getContentPane();
    content.add(textComp, BorderLayout.CENTER);
    content.add(createToolBar(), BorderLayout.NORTH);
    setJMenuBar(createMenuBar());
    setSize(320, 240);
}

// ici le JTextComponent en un JTextArea: plain text
protected JTextComponent createTextComponent() {
    JTextArea ta = new JTextArea();
    ta.setLineWrap(true);
    return ta;
}
```

# Créations des actions

---

```
protected void mesActions() {
    Action a;
    a = textComp.getActionMap().get(DefaultEditorKit.cutAction);
    a.putValue(Action.SMALL_ICON, new ImageIcon("images/Couper.gif"));
    a.putValue(Action.NAME, "Couper");

    a = textComp.getActionMap().get(DefaultEditorKit.copyAction);
    a.putValue(Action.SMALL_ICON, new ImageIcon("images/Copier.gif"));
    a.putValue(Action.NAME, "Copier");

    a = textComp.getActionMap().get(DefaultEditorKit.pasteAction);
    a.putValue(Action.SMALL_ICON, new ImageIcon("images/Coller.gif"));
    a.putValue(Action.NAME, "Coller");

    a = textComp.getActionMap().get(DefaultEditorKit.selectAllAction);
    a.putValue(Action.NAME, "Tout Sélectionner");
}
```



# Barre d'outils

---

```
// JToolBar avec boutons
protected JToolBar createToolBar() {
    JToolBar bar = new JToolBar();
    // Ouvrir et Sauver
    bar.add(openAction).setText("Ouvrir");
    bar.add(saveAction).setText("Sauver");
    bar.addSeparator();
    // couper-coller
    bar.add(textComp.getActionMap().get(DefaultEditorKit.cutAction))
        .setText("Couper");
    bar.add(textComp.getActionMap().get(DefaultEditorKit.copyAction))
        .setText("Copier");
    bar.add(textComp.getActionMap().get(DefaultEditorKit.pasteAction))
        .setText("Coller");
    return bar;
}
```

# Menu

---

```
protected JMenuBar createMenuBar() {
    JMenuBar menubar = new JMenuBar();
    JMenu file = new JMenu("Fichier");
    JMenu edit = new JMenu("Edition");
    menubar.add(file);
    menubar.add(edit);
    file.add(openAction);
    file.add(saveAction);
    file.add(new ExitAction());
    edit.add(textComp.getActionMap()
        .get(DefaultEditorKit.cutAction));
    edit.add(textComp.getActionMap()
        .get(DefaultEditorKit.copyAction));
    edit.add(textComp.getActionMap()
        .get(DefaultEditorKit.pasteAction));
    edit.add(textComp.getActionMap()
        .get(DefaultEditorKit.selectAllAction));
    return menubar;
}
```

# OpenAction

```
class OpenAction extends AbstractAction {
    public OpenAction() {
        super("Ouvrir", new ImageIcon("icons/open.gif"));
    }
    // avec un JFileChooser
    public void actionPerformed(ActionEvent ev) {
        JFileChooser chooser = new JFileChooser();
        if (chooser.showOpenDialog(SimpleEditeur.this) != JFileChooser.APPROVE_OPTION) {
            return;
        }
        File file = chooser.getSelectedFile();
        if (file == null) {return;}
        FileReader reader = null;
        try {
            reader = new FileReader(file);
            textComp.read(reader, null);
        } catch (IOException ex) {
            JOptionPane.showMessageDialog(SimpleEditeur.this,
                "Pas de fichier", "ERROR", JOptionPane.ERROR_MESSAGE);
        } finally {
            if (reader != null) {
                try {
                    reader.close();
                } catch (IOException x) {}
            }
        }
    }
}
```

# SaveAction

```
class SaveAction extends AbstractAction {
    public SaveAction() {
        super("Save", new ImageIcon("icons/save.gif"));
    }
    public void actionPerformed(ActionEvent ev) {
        JFileChooser chooser = new JFileChooser();
        if (chooser.showSaveDialog(SimpleEditeur.this) != JFileChooser.APPROVE_OPTION) {
            return;
        }
        File file = chooser.getSelectedFile();
        if (file == null) {return;}
        FileWriter writer = null;
        try {
            writer = new FileWriter(file);
            textComp.write(writer);
        } catch (IOException ex) {
            JOptionPane.showMessageDialog(SimpleEditeur.this,
                "Erreur", "ERROR", JOptionPane.ERROR_MESSAGE);
        } finally {
            if (writer != null) {
                try {
                    writer.close();
                } catch (IOException x) {}
            }
        }
    }
}
```

# Le reste...

---

```
public class SimpleEditeur extends JFrame {
    private Action openAction = new OpenAction();
    private Action saveAction = new SaveAction();
    private JTextComponent textComp;
    private static void createAndShowGUI() {
        SimpleEditeur editeur = new SimpleEditeur();
        editeur.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        editeur.setVisible(true);
    }
    //le main
    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                UIManager.put("swing.boldMetal", Boolean.FALSE);
                createAndShowGUI();
            }
        });
    }
}
```