Interface Graphique en Java 1.6

Introduction à Swing

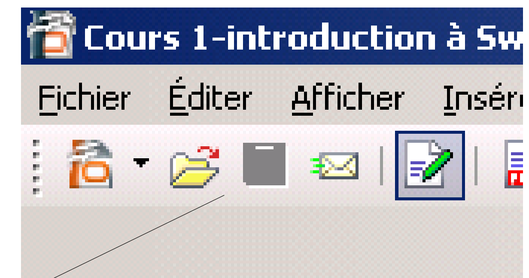
Sébastien Paumier

Objectifs du cours

- comprendre le fonctionnement d'une interface graphique:
  - positionnement et rendu des composants
  - gestion des événements
- maîtriser les composants les plus courants (fenêtres, boutons, listes, menus, arbres, etc)
- être capable de concevoir et d'implémenter proprement une interface

Principe d'une IG

- intermédiaire entre l'utilisateur et la partie "métier" d'un logiciel



- ce qui relève de l'IG:
  - gérer le fait que le bouton soit actif ou non (idem pour la commande "Enregistrer")
  - lancer la sauvegarde quand on clique
- ce qui ne relève pas de l'IG:
  - la sauvegarde elle-même!

La spécificité de Java

- Java est un langage multi-plateforme
- il faut donc une gestion des IG qui marche partout
- 3 toolkits graphiques:
  - AWT: l'ancêtre, vieux, usé et fatigué
  - SWT: la version IBM utilisée dans Eclipse
  - Swing: grosse API, bien maintenue, très utilisée, base de ce cours

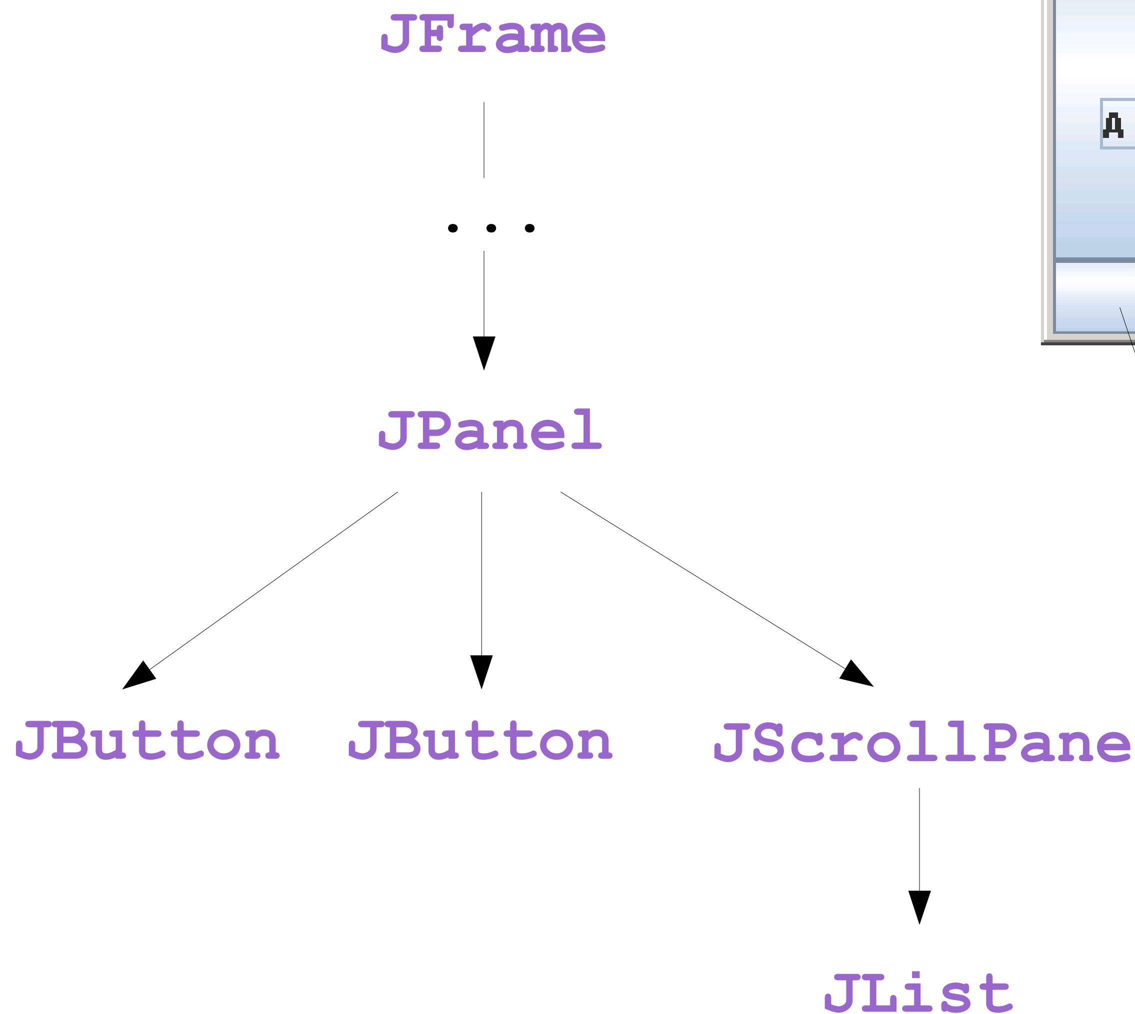
Swing

- les noms de composants Swing commencent (souvent) par `J`
- tous héritent du composant graphique `JComponent`
- `JComponent` hérite du composant graphique *AWT* `Container` qui permet à un composant d'en contenir d'autres

Les IG sont arborescentes

- une IG Swing est un arbre qui part d'un objet du système (*heavyweight*):
  - **`JFrame`**: fenêtre normale
  - **`JWindow`**: fenêtre non décorée avec gestion partielle des événements (utilisée pour les splash screen)
  - **`JDialog`**: boîte de dialogue
  - **`JApplet`**
- cet objet racine contient des objets gérés par Java (*lightweight*)

Les IG sont arborescentes



style du système, car **JFrame** est heavyweight

style de Swing, car **JButton** est lightweight

La structure de JFrame

JFrame

JRootPane (un seul par JFrame)

glassPane (un JPanel transparent par-dessus tout l'intérieur de la JFrame)

JLayeredPane

JMenuBar

contentPane (un JPanel)

Le contentPane

- par défaut, c'est un **JPanel** opaque
- peut être récupéré ou changé:
  - **getContentPane**/**setContentPane**
- exemple:

```
public static void main(String[] args) {
JFrame f=new JFrame("Hello world!");
f.getContentPane().add(new JButton("A button"),BorderLayout.WEST);
f.getContentPane().add(new JButton("Another button"),BorderLayout.SOUTH);
String[] s={"a","bb","ccc","dd","e","ff","ggg","hh","i","jj"};
f.getContentPane().add(new JScrollPane(new JList(s)));
f.setSize(300,150);
f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
f.setVisible(true);
}
```

Le glassPane

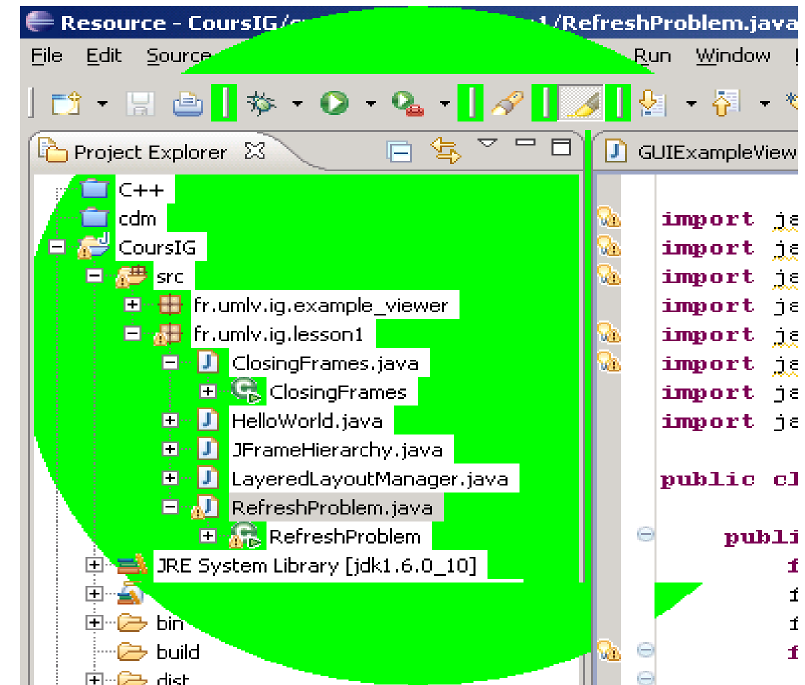
- par défaut, c'est un **JPanel** transparent et invisible (ce n'est pas un pléonasme)
- peut être récupéré ou changé:
  - **getGlassPane**/**setGlassPane**
- exemple:

```
f.setGlassPane(new JComponent() {
    @Override
    protected void paintComponent(Graphics g) {
    Graphics2D g2=null;
    try{
        g2=(Graphics2D)g.create();
        g2.setStroke(new BasicStroke(13));
        g2.setColor(Color.RED);
        g2.drawLine(0,getHeight(),getWidth(),0);
        g2.drawString("This is the glass pane",110,40);
    } finally {
        g2.dispose();
    }
}
});
```

Création d'une JFrame

- on doit préciser une taille, sinon:

- on doit la rendre visible
- on ne touche plus à rien dans la fenêtre après `setVisible(true)` sinon, risque de soucis...



Fermeture d'une JFrame

- 4 modes possibles, à définir avec **`setDefaultCloseOperation`**:
  - **`DO_NOTHING_ON_CLOSE`**
  - **`HIDE_ON_CLOSE`**: rend la fenêtre invisible (mode par défaut)
  - **`DISPOSE_ON_CLOSE`**: cache la fenêtre et libère toutes les ressources systèmes associées; elles seront réallouées si la fenêtre redevient visible
  - **`EXIT_ON_CLOSE`**: termine le programme

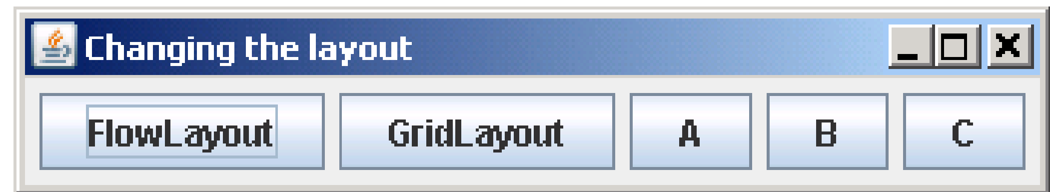
Propriétés de JFrame

- on peut bloquer le redimensionnement avec **setResizable**
- on peut enlever les décorations avec **setUndecorated**, mais seulement quand la fenêtre n'est pas associée à des ressources système

```
JCheckBox b2=new JCheckBox("Decorated",true);
b2.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        JCheckBox b=(JCheckBox) e.getSource();
        f.dispose();
        f.setUndecorated(!b.isSelected());
        f.setVisible(true);
}});
```

Les Container

- en Swing, on distingue la gestion logique des composants:
  - méthodes `add`/`remove` de `Container`
- de leur placement qui est géré par un `LayoutManager` (gestionnaire de géométrie)



Propriétés des JComponent

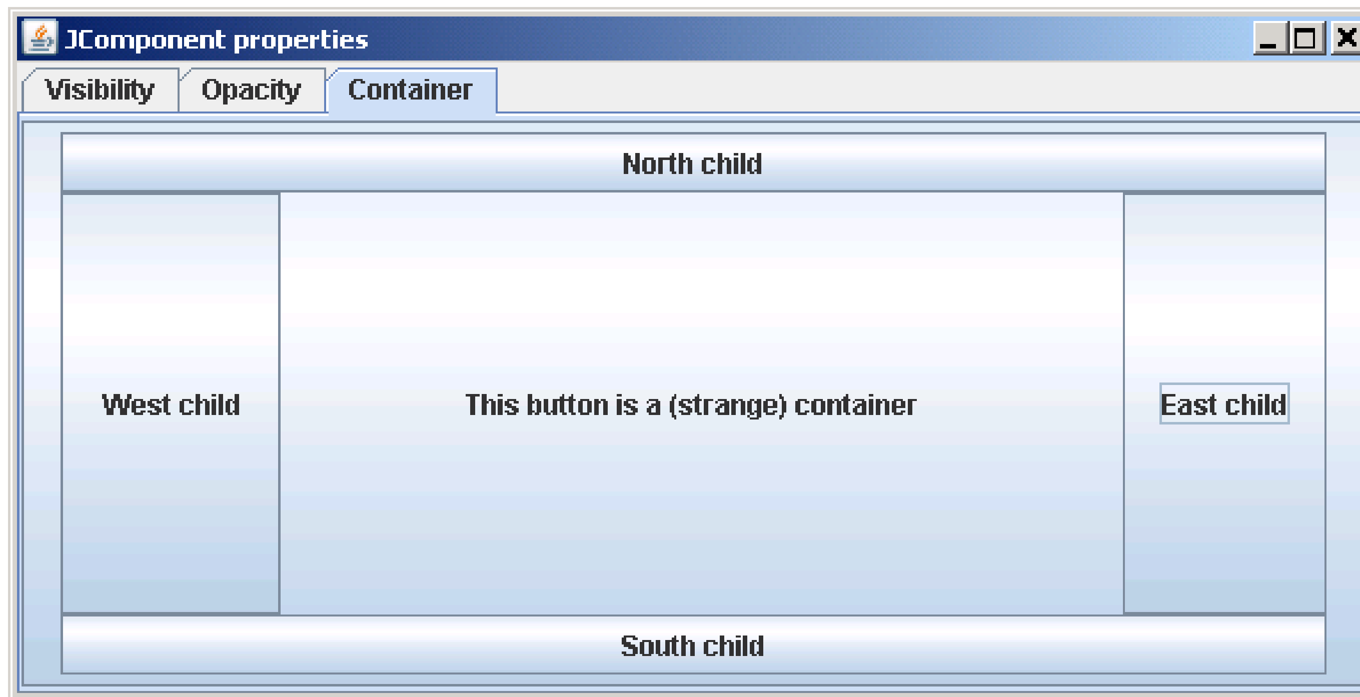
- visibilité: est-ce que le composant est dessiné ou non ?
  - **setVisible**/**isVisible**
  - le composant n'est plus pris en compte par le **LayoutManager**
  - le composant ne capte plus les événements souris et clavier

Propriétés des JComponent

- opacité: est-ce que le fond du composant doit être dessiné ou non ?
  - `setOpaque`
  - si oui, la couleur utilisée est celle définie par `setBackground`
  - la plupart des composants sont opaques par défaut, mais pas le `JLabel`
  - **ATTENTION:** `setOpaque` ne rafraîchit pas le composant!!

Propriétés des JComponent

- être un container, même si c'est parfois bizarre...

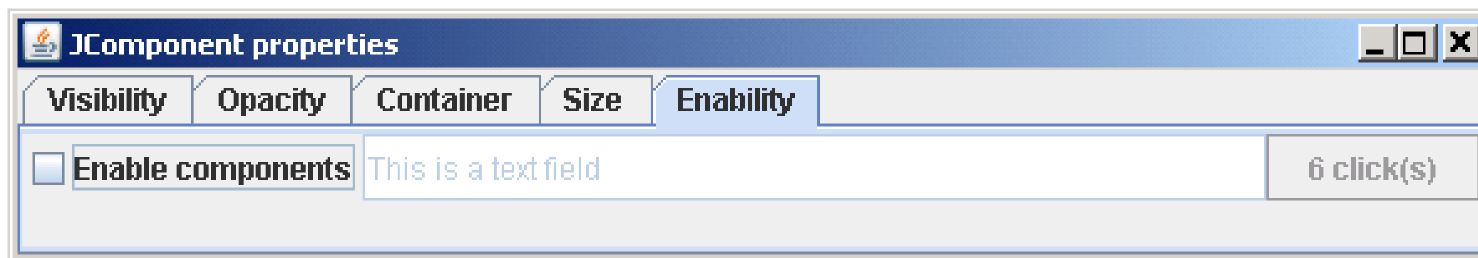


Propriétés des JComponent

- la taille courante:`set`/`getSize`
- + 3 autres tailles:
  - la préférée: `set`/`getPreferredSize`
  - la minimum: `set`/`getMinimumSize`
  - la maximum: `set`/`getMaximumSize`
- qui sont utilisées, entre autres, par les `LayoutManager` pour calculer la place accordée à chaque composant

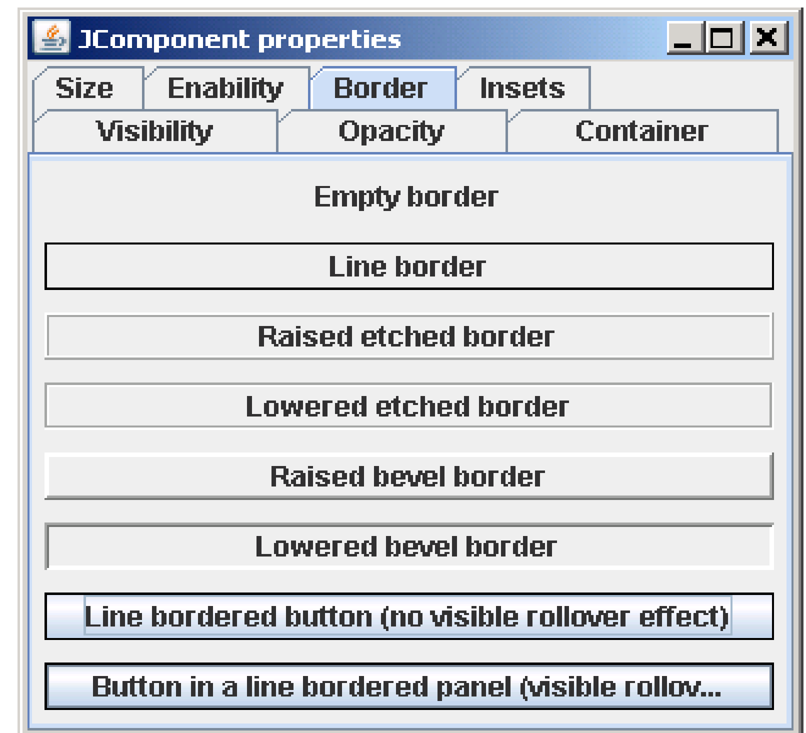
Propriétés des JComponent

- actif/inactif
  - **setEnabled**/**isEnabled**
- un composant inactif ne réagit plus aux événements provoqués par l'utilisateur (souris, clavier)
- correspond à un rendu grisé



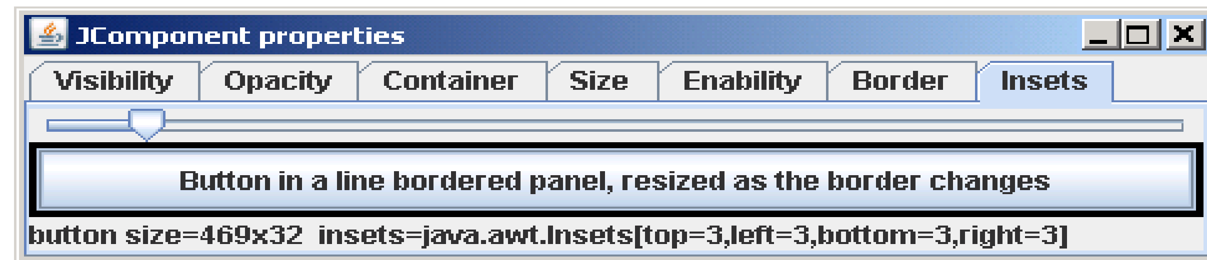
Propriétés des JComponent

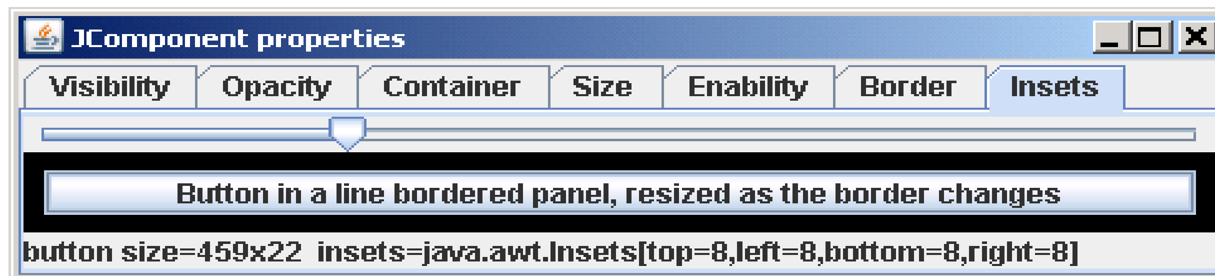
- bordures définies avec **setBorder**, créées avec **BorderFactory.createXXXBorder** (ou à la main)
- à utiliser de préférence sur les **JPanel** et les **JLabel**
- pour les autres composants, il peut y avoir des problèmes de rendu



Propriétés des JComponent

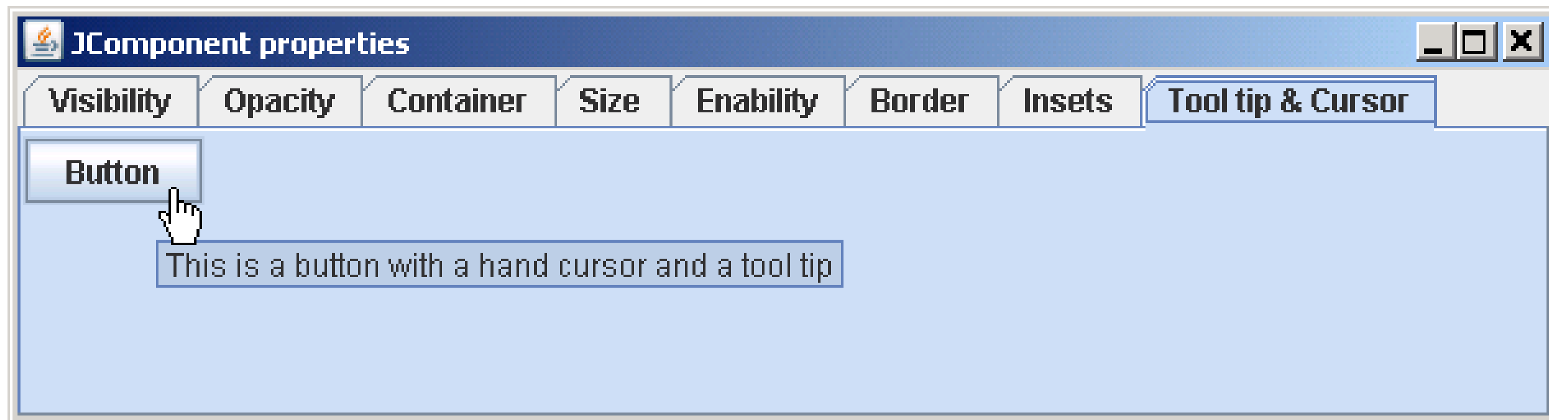
- le **JPanel** tient compte de la bordure pour dessiner les composants qu'il contient (*insets*=taille de la bordure; pour une **JFrame**, ça inclue la barre de titre):





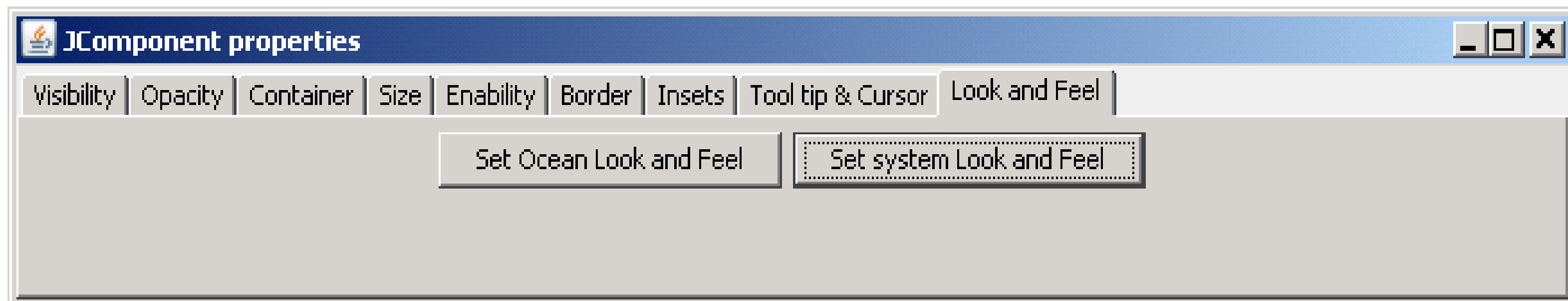
Propriétés des JComponent

- bulle d'aide définie avec `setToolTipText`
- curseur défini avec `setCursor`
- curseurs prédéfinis:
  `Cursor.getPredefinedCursor(Cursor.XXX_CURSOR))`



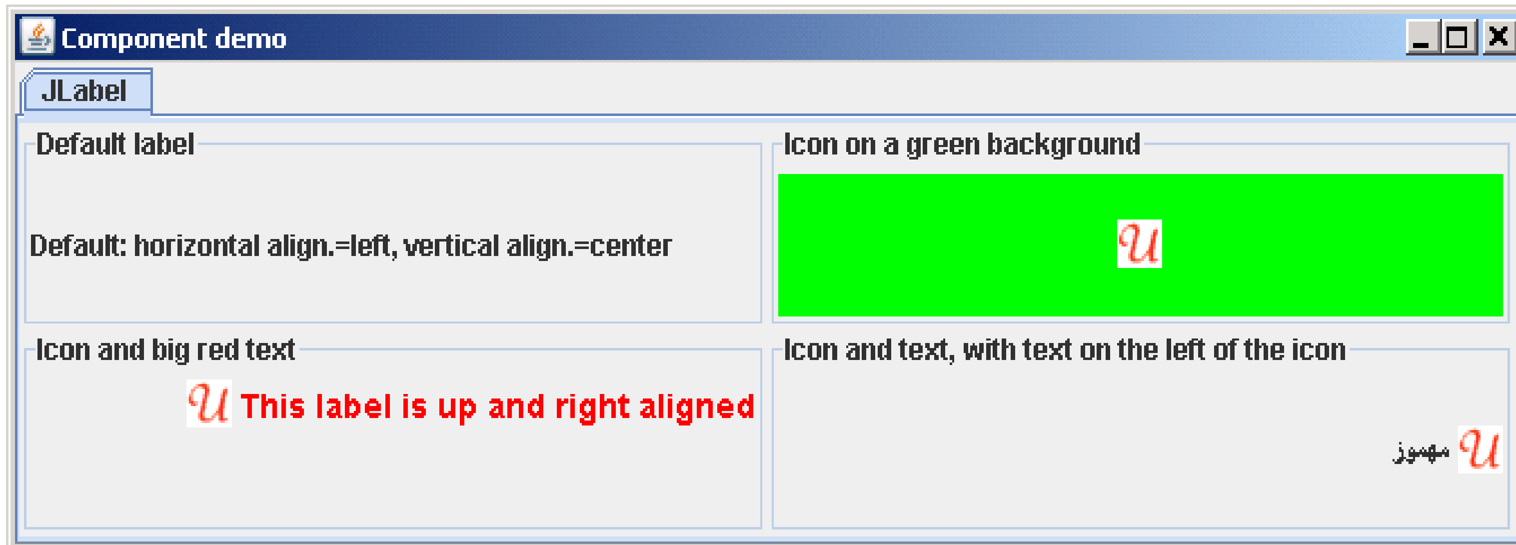
Propriétés des JComponent

- le rendu est délégué au Look and Feel



Les labels

- JLabel=texte non éditable sur une ligne
  - peut être accompagné d'une icône
  - transparent par défaut
  - on peut gérer la position du texte et de l'icône, les couleurs (fond et texte) et la fonte



Obtenir une image

"Je suis dans la classe **`Biniou`**, *et je veux utiliser l'image* `pouet.png` *comme icône pour un label. Comment faire ?"*

- 1) mettre le fichier `pouet.png` dans le même répertoire que `Biniou.class` (sous Eclipse, si on le met avec `Biniou.java`, il est copié automatiquement)

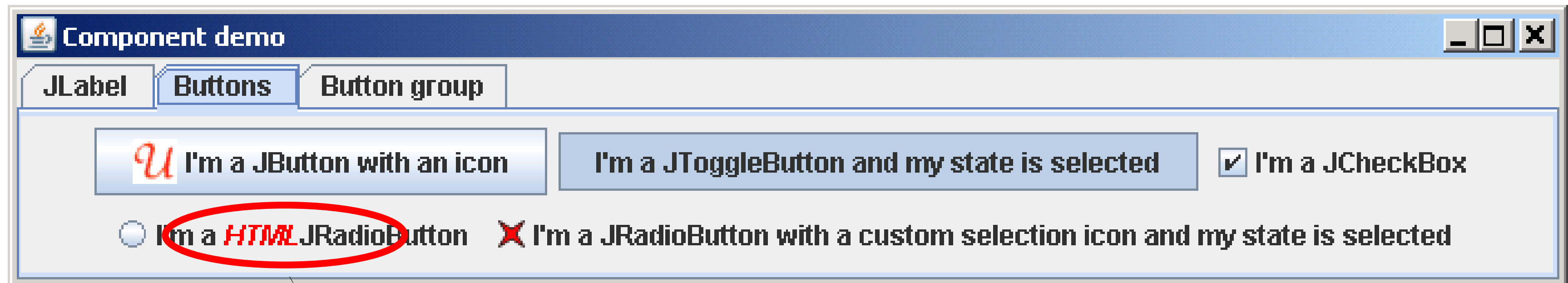
2) utiliser le code suivant:

```
ImageIcon icon=new ImageIcon(Biniou.class.getResource("pouet.png"));
JLabel label=new JLabel(icon);
```

3) mettre `icon` en champ `static final` si elle sert plusieurs fois

Les boutons

- JButton=bouton classique
- JToggleButton=bouton qui reste enfoncé:
  - test avec isSelected
  - 2 sous-classes: JCheckBox et JRadioButton
- on peut ajouter une icône



ASTUCE: beaucoup de composants Swing tolèrent du HTML très simple

Les boutons

- pour les faire réagir aux événements, on utilise la méthode **`addActionListener`**
- cette interface ne possède que la méthode **`actionPerformed`**, invoquée quand le bouton est activé à la souris ou au clavier:
- Note: un **`JToggleButton`** est activé soit quand on l'enfonce, soit quand on le désenfonce

Les boutons

- exemple d'utilisation d'un listener sur un **JToggleButton**:

```
final JToggleButton b=new JToggleButton();
b.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        /* IMPORTANT: b must be final to be used here */
        b.setText("I'm a JToggleButton and my state is "
                  +(b.isSelected()?"pressed":"unpressed"));
        }
    });
/* We simulate a click to initialize b's caption */
b.doClick();
```

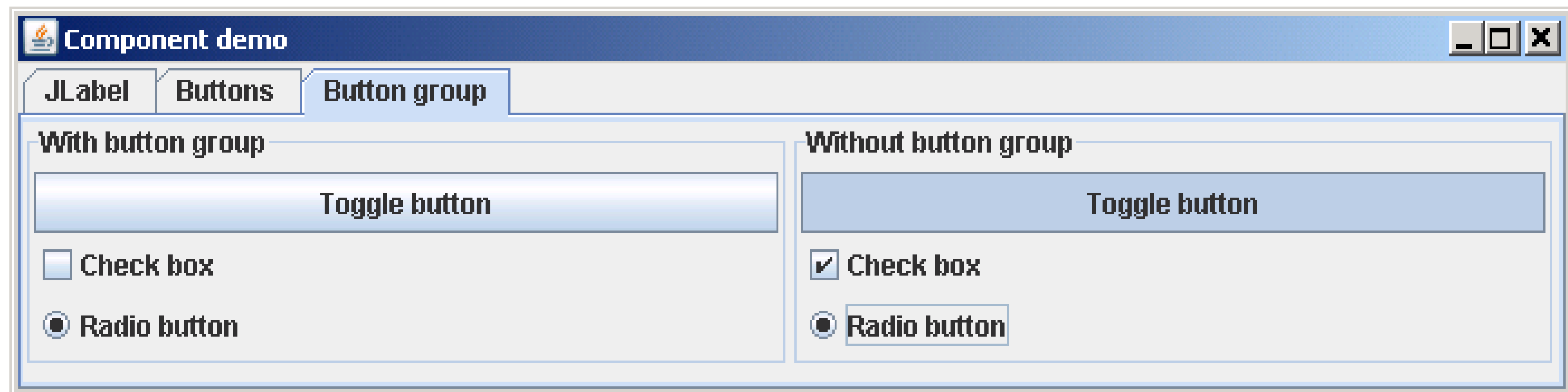
toute variable déclarée en dehors de **actionPerformed** doit être finale pour pouvoir être utilisée ici (cf. cours sur les événements)

Les groupes de boutons

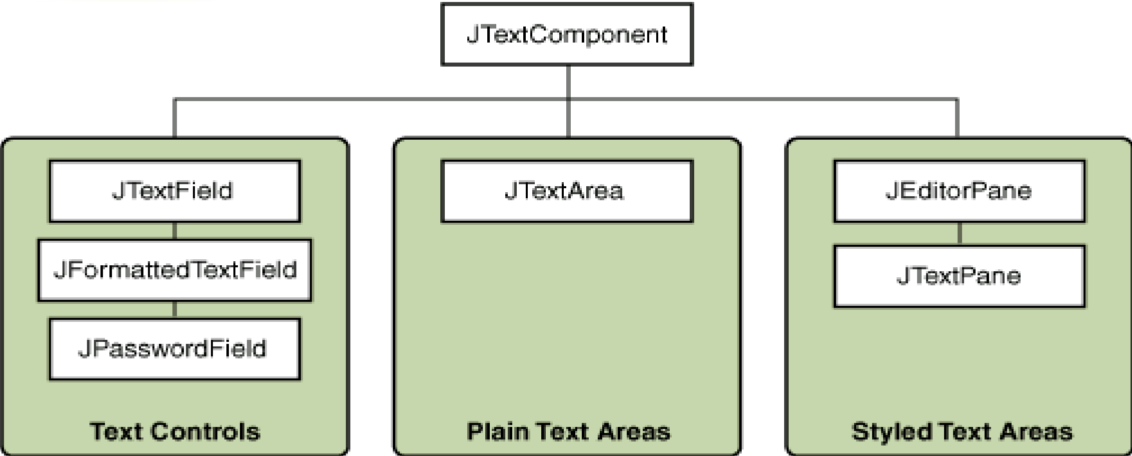
- on peut créer un groupe de boutons, pour n'avoir qu'un bouton sélectionné parmi plusieurs grâce à **`ButtonGroup`**
- à utiliser avec **`JToggleButton`**, **`JCheckBox`** et **`JRadioButton`** (crétin avec **`JButton`**)
- on ajoute un bouton au groupe avec **`add`**
- **`ATTENTION:`** il faut aussi ajouter le bouton au container, car **`ButtonGroup`** n'est qu'un container logique!

Les groupes de boutons

- NOTE: quand un bouton est dans un **ButtonGroup**, par défaut, on ne peut pas le désélectionner en cliquant sur lui

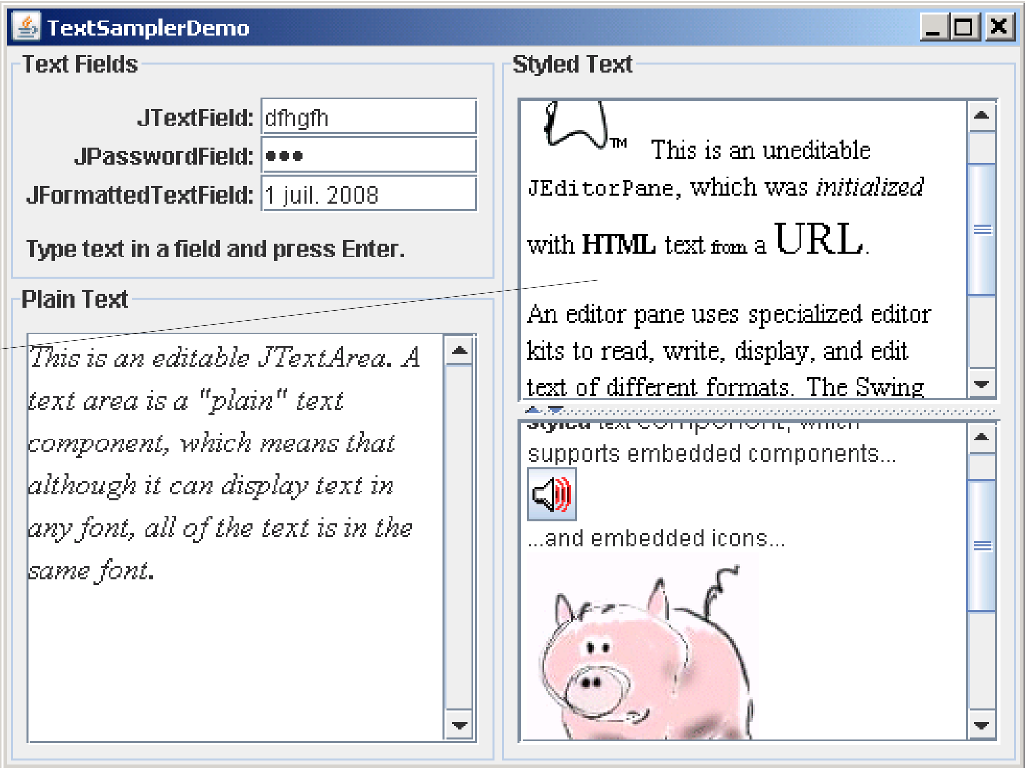


Les zones de texte



les zones de texte peuvent être éditables ou non (`setEditable`)

pratique pour charger une page HTML; beaucoup moins pour éditer du texte enrichi en direct



IG et héritage

- sauf cas ***très*** particulier on n'hérite pas d'un composant graphique:

```
public class BadHelloWorld extends JFrame {

public BadHelloWorld() {
super("Hello world!");
getContentPane().add(new JButton("BAD :("));
pack();
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
setVisible(true);
}

public static void main(String[] args) {
new BadHelloWorld();
}

}
```

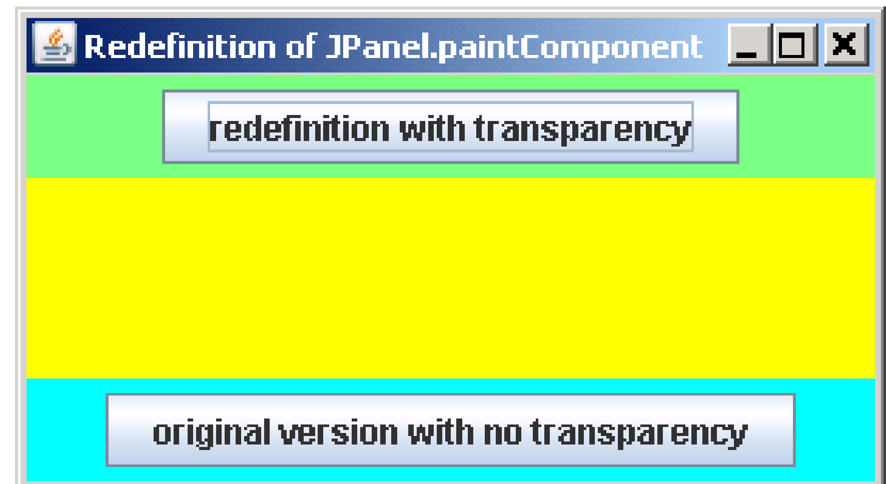
```
public class GoodHelloWorld {

public static void main(String[] args) {
JFrame f=new JFrame("Hello world!");
f.getContentPane().add(new JButton("GOOD :)"));
f.pack();
f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
f.setVisible(true);
}

}
```

utilisation normale de **JFrame**, donc pas d'héritage

IG et redéfinition de méthode

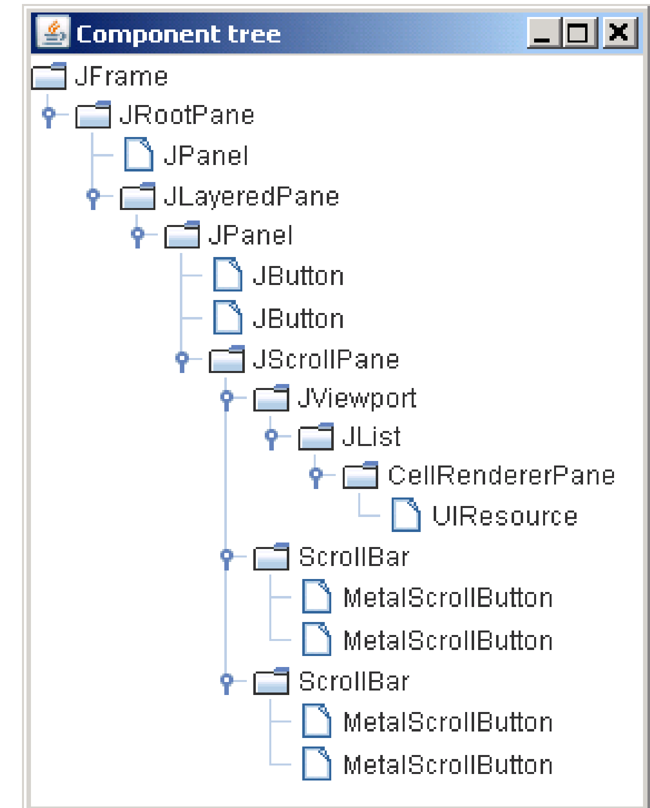
- en revanche, on peut vouloir modifier le comportement d'un composant
- exemple: ajout de transparence à un **`JPanel`**

```
JPanel p = new JPanel() {
    @Override
    protected void paintComponent(Graphics g) {
        Graphics2D g2 = (Graphics2D) g;
        Composite old = g2.getComposite();
        g2.setComposite(composite);
        super.paintComponent(g2);
        g2.setComposite(old);
    }
};
```



Ne pas stocker les objets

- chaque composant est déjà stocké par son père
- inutile de le gérer en plus à côté!




```
private static DefaultMutableTreeNode buildTreeNode(Component c) {
    DefaultMutableTreeNode node =
        new DefaultMutableTreeNode(c.getClass().getSimpleName());
    try {
        /* If the component is a container, it may have children */
        Container c2 = (Container) c;
        int n = c2.getComponentCount();
        for (int i = 0; i < n; i++) {
            node.add(buildTreeNode(c2.getComponent(i)));
        }
    } catch (ClassCastException e) {
    }
    return node;
}
```

Pas de copier-coller !

- si on doit faire la même chose plusieurs fois, factoriser le code
- méthode 1: écrire une *factory method* **createMyBiniou**

```
private static JPanel createPanel(Color c) {
     JPanel p = new JPanel() {
          @Override
          protected void paintComponent(Graphics g) {
               Graphics2D g2 = (Graphics2D) g;
               Composite old = g2.getComposite();
               g2.setComposite(composite);
               super.paintComponent(g2);
               g2.setComposite(old);
          }
     };
     p.setBackground(c);
     return p;
}
```

Pas de copier-coller !

- méthode 2: la bonne vieille boucle

```
String[] labels = { "DO_NOTHING_ON_CLOSE", "HIDE_ON_CLOSE (for 2 seconds)",
            "DISPOSE_ON_CLOSE", "EXIT_ON_CLOSE" };
int[] values = { JFrame.DO_NOTHING_ON_CLOSE, JFrame.HIDE_ON_CLOSE,
            JFrame.DISPOSE_ON_CLOSE, JFrame.EXIT_ON_CLOSE };
ButtonGroup group = new ButtonGroup();
Box box = new Box(BoxLayout.Y_AXIS);
for (int i = 0; i < labels.length; i++) {
    final JRadioButton b = new JRadioButton(labels[i], i == 0);
    final int value = values[i];
    group.add(b);
    box.add(b);
    b.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            if (b.isSelected()) {
                f2.setDefaultCloseOperation(value);
            }
        }
    });
}
```

Trouver une info sur un composant

- LA source: la Javadoc
- l'heuristique:
  - mettre le biniou qu'on cherche en anglais
  - chercher **set/getBagpipes**
- exemple: indice de la cellule sélectionnée dans une liste

JList (Java Platform SE 6) - Mozilla Firefox

Fichier Édition Affichage Historique Marque-pages Outils ?

file:///E:/jdk-6-doc/docs/api/index.html

Google

Java™ Platform

All Classes

int **getSelectedIndex**()

Returns the smallest selected cell index; *the selection* when only a single item is selected in the list.

Rechercher : validate Suivant Précédent Surligner tout Respecter la casse

Terminé Get video