

# Conteneurs de Swing

- Conteneurs généraux
- Les couches de **JFrame**
- Menus et Toolbars
- Conteneurs spécialisés

# Conteneurs généraux

---

## JPanel

- Conteneur très général, dérive immédiatement de **JComponent**
- Contient un **FlowLayout** par défaut
- Est opaque, ce qui importe pour les dessins.

## Box

- Ne descend pas de **JComponent**
- Sert comme conteneur avec remplissage, car utilise **BoxLayout**
- *Ne peut avoir de bordure* car ne descend pas de **JComponent**.

# Les couches de JFrame

---

## JLayeredPane

- Conteneur général pour des composants en couche.
- On peut donner des valeurs de niveau aux composants indiquant qui est affiché au dessus.
- Utilise le *null* **Layout**, donc positionner ses enfants avec **setBounds ( )**.
- Classe mère de **JDesktopPane**.

# JLayeredPane

---

- La *profondeur* d'une couche est représenté par un objet **Integer**.
- Six profondeurs prédéfinies
  - **FRAME\_CONTENT\_LAYER (-30000)**  
le `ContentPane` est de ce niveau
  - **DEFAULT\_LAYER (0)**  
niveau "par défaut"
  - **PALETTE\_LAYER (100)**  
pour les palettes, boîtes à outils déplaçables
  - **MODAL\_LAYER (200)**  
pour les dialogues modaux
  - **POPUP\_LAYER (300)**  
pour les menus glissants, les tooltips
  - **DRAG\_LAYER (400)**  
pour le glisser-déposer
- L'affichage est évidemment en ordre croissant

# JDesktopPane

---

- Conteneur pour gérer des **JInternalFrames**.
- Ces composants peuvent être retailés, minimisés, etc.
- Comme **JLayeredPane**, utilise un Layout *null*.
- Deux autres classes utilisées
  - **DesktopManager** qui gère les opérations des **JInternalFrames** dépendant du look-and-feel. **JDesktopPane** crée un **DefaultDesktopManager**.
  - **DesktopPaneUI** qui gère le graphique du **DesktopPane** (mais pas des internal frames). Obtenu du look-and-feel courant.

# JRootPane

---

- Composant avec un rôle très précis.
- Le seul fils possible pour **JWindow**, **JDialog**, **JFrame** et **JInternalFrame**
- Un **JRootPane** a deux parties : *glassPane* et *layeredPane*, et cette deuxième a deux parties : *menuBar* et *contentPane*.
- Les *menuBar* et *contentPane* sont créés et gérés par **JRootPane**.
- Le *glassPane* sert à capter des évènements souris.

# Menus et Toolbars

---

## JMenu

- Un menu a des entrées qui sont **JMenuItem**, **JCheckBoxMenuItem**, **JRadioButtonMenuItem**, **JSeparator**, et **JMenu**

## JPopupMenu

- Sert pour les menus dynamiques et les menus déroulants.
- Utilise **SingleSelectionModel**, qui contient l'entrée sélectionnée.

## JToolBar

- Conteneur général, qui se retaille et peut d'être déplacé.
- Utilise un **BoxLayout** horizontal, et ses composants peuvent donc être espacés ou groupés.

# Conteneurs spécialisés

---

- Le **JTabbedPane** est un conteneur à navigation par onglets
- Le **JScrollPane** permet d'afficher une partie d'une zone
- Le **JSplitPane** introduit un volet mobile entre deux composants

# JTabbedPane

---

- Groupe une liste de conteneurs repérés par des onglets.
- Création:

```
JTabbedPane()  
JTabbedPane(int cotéOnglets)
```

- Ajout de conteneurs à un **tabbedPane**:

```
addTab(String texteOnglet, Component composant)  
addTab(String texteOnglet, Icon icone, Component composant)  
addTab(String texteOnglet, Icon icone, Component composant,  
String toolTipText)
```

# JTabbedPane

---

- Feuille initiale

```
tabbedPane.setSelectedIndex(int numero)
```

- Récupérer le choix

```
int tabbedPane.getSelectedIndex()
```

- Et la feuille elle-même

```
Component tabbedPane.getComponentAt(int numero);
```

- Nombre total de feuilles

```
int tabbedPane.getTabCount();
```

# JTabbedPane : exemple (1)

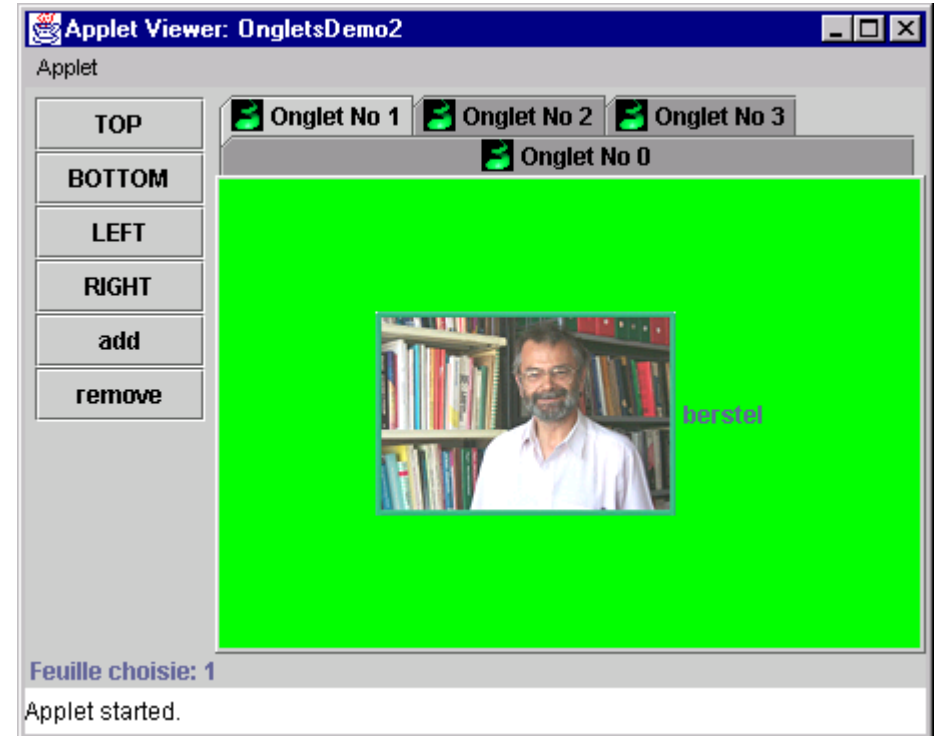


Pour

- naviguer
- ajouter, enlever les feuilles
- choisir la position des onglets

De plus

- un message affiche le numéro de l'onglet, à chaque changement



# JTabbedPane : exemple (2)

## ■ Acteurs principaux

```
class Panneau extends JPanel implements ActionListener {
    String [] imageNames = { "arques","berstel","crochemore","desarmenien", ...};
    ImageIcon[] images = new ImageIcon[imageNames.length]; les images montrées
    ImageIcon tabimage; l'icône dans les onglets
    JTabbedPane tabbedPane; le panneau à feuilles
    String[] boutonNames = {"TOP","BOTTOM","LEFT","RIGHT","add","remove"};
    JButton[] boutons= new JButton[boutonNames.length]; les boutons de gestion
    JLabel statut; le message d'état
    AudioClip layoutson, tabson; les sons des actions

    Panneau() {} création de la scène
    void createTab() {} ajoute une feuille et son onglet
    void killTab() {} supprime une feuille
    void setStatus(int index) {...} gestion du message
    public void actionPerformed(ActionEvent e) {...} les actions des boutons
}
```

# JTabbedPane : exemple (3)

## ■ Création / suppression de feuilles

```
public void createTab() {
    JLabel feuille = null;
    int ong = tabbedPane.getTabCount();
    feuille = new JLabel(imageNames[ong % images.length],
        images[ong % images.length], SwingConstants.CENTER);
    feuille.setOpaque(true);
    feuille.setBackground(Color.green);
    tabbedPane.addTab("Feuille No " + ong, tabImage, feuille);
    tabbedPane.setSelectedIndex(ong);
    setStatus(ong);
}

public void killTab() { // dernière
    if (tabbedPane.getTabCount() > 0) {
        tabbedPane.removeTabAt(tabbedPane.getTabCount()-1);
        setStatus(tabbedPane.getSelectedIndex());
    }
}

public void setStatus(int index) {
    if (index > -1) statut.setText(" Feuille choisie: " + index);
    else statut.setText(" Pas de feuille choisie");
}
```

# JTabbedPane : exemple (4)

- Les actions des boutons
- La classe **SwingConstants** contient les constantes de placement

```
public void actionPerformed(ActionEvent e) {
    String lib = ((JButton) e.getSource()).getActionCommand();
    if (lib.equals(boutonNames[0])) {
        tabbedPane.setTabPlacement(SwingConstants.TOP);
        layoutson.play(); }
    else if(lib.equals(boutonNames[1])) {
        tabbedPane.setTabPlacement(SwingConstants.BOTTOM);
        layoutson.play(); }
    else if(lib.equals(boutonNames[2])) {
        tabbedPane.setTabPlacement(SwingConstants.LEFT);
        layoutson.play(); }
    else if(lib.equals(boutonNames[3])) {
        tabbedPane.setTabPlacement(SwingConstants.RIGHT);
        layoutson.play();}
    else if(lib.equals(boutonNames[4]))
        createTab();
    else if(lib.equals(boutonNames[5]))
        killTab();
}
```

# JTabbedPane : exemple (5)

```
Panneau() {
    tabimage = new ImageIcon("gifs/tabimage.gif");
    for (int i = 0 ; i < images.length; i++)
        images[i] = new ImageIcon("gifs/" + imageNames[i] + ".jpg");
    for (int i = 0; i < boutons.length; i++)
        boutons[i] = new JButton(boutonNames[i]);
    statut = new JLabel();
    JPanel boutonPanel = new JPanel();
    boutonPanel.setLayout(new GridLayout(0,1));
    for (int i = 0; i < boutons.length ; i++){
        boutons[i].addActionListener(this); boutonPanel.add(boutons[i]);
    }
    JPanel leftPanel = new JPanel();
    leftPanel.add(boutonPanel);
    tabbedPane = new JTabbedPane(SwingConstants.TOP);
    createTab(); createTab(); createTab(); createTab();
    tabbedPane.addChangeListener(new ChangeListener(){
        public void stateChanged(ChangeEvent e) {
            setStatus(((JTabbedPane) e.getSource()).getSelectedIndex());
            tabson.play();
        }
    });
    setLayout(new BorderLayout());
    add(leftPanel, BorderLayout.WEST);
    add(statut, BorderLayout.SOUTH);
    add(tabbedPane, BorderLayout.CENTER);
}
```

# JTabbedPane : exemple (fin)

```
public void init() {
    JLabel loading = new JLabel("Initialisation en cours...", JLabel.CENTER);
    setContentPane(loading);
    setVisible(true);
    getRootPane().revalidate();
    try { Thread.sleep(1000); } catch (InterruptedException e) {};
    layoutson = getAudioClip(getCodeBase(), "switch.wav");
    tabson = getAudioClip(getCodeBase(), "tab.wav");
    Panneau panneau = new Panneau();
    panneau.addAudioClips(layoutson, tabson);
    setContentPane(panneau);
}
}
```

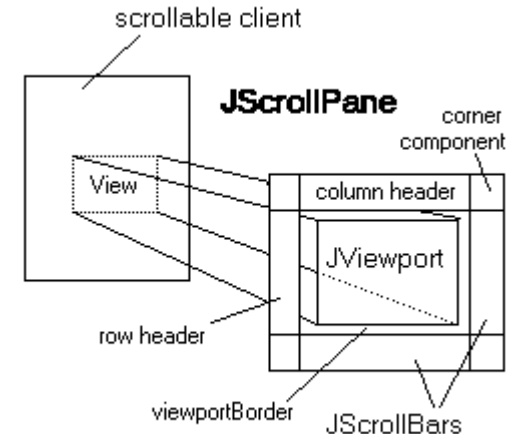
- On modifie le contenu après le chargement
- La méthode **revalidate** sert à “forcer” le réaffichage.

# JScrollPane

- Gère automatiquement des ascenseurs autour de son composant central qui est un **JViewport**.
- Constructeurs principaux

```
JScrollPane()
```

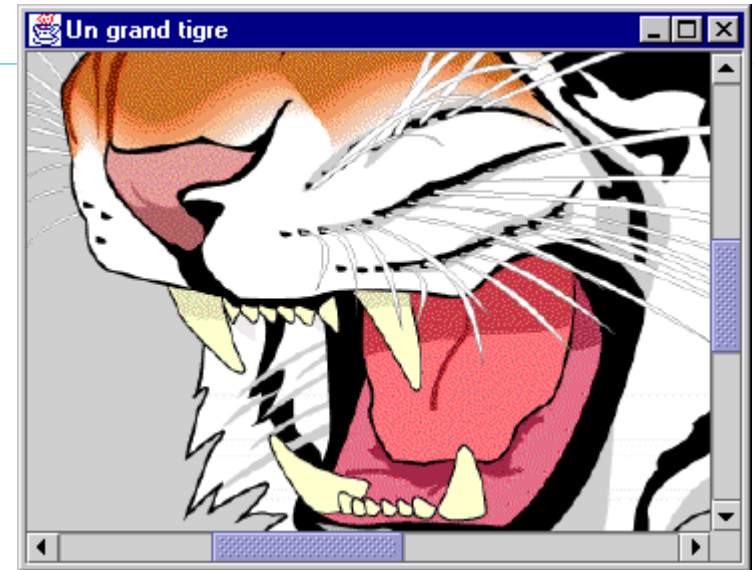
```
JScrollPane(Component view)
```



- Une “vue” s’ajoute au **JViewport**, si elle ne l’est dans le constructeur, par

```
scrollPane.setViewportView().add(view)
```

# Exemple

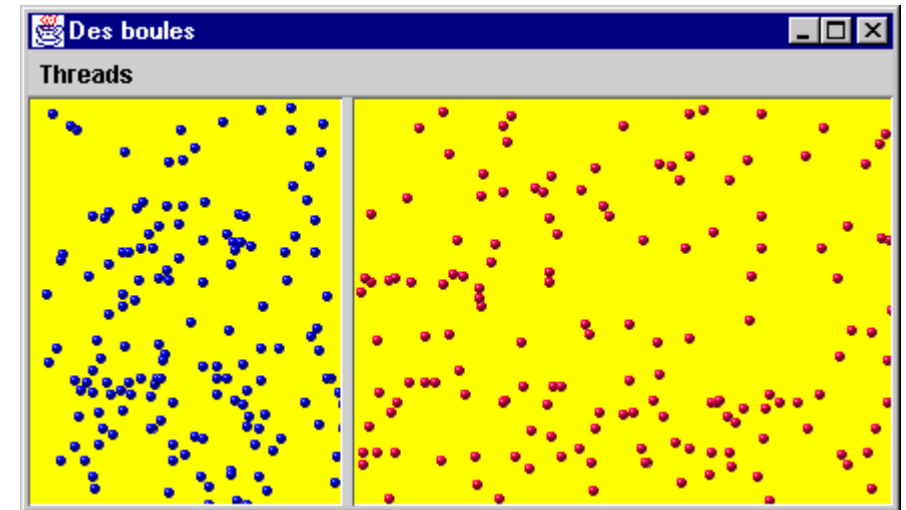


```
class ScrollPanel extends JPanel {  
    public ScrollPanel() {  
        setLayout(new BorderLayout());  
        Icon iconeTigre = new ImageIcon("BigTiger.gif");  
        JLabel etiquetteTigre = new JLabel(iconeTigre);  
        JScrollPane scrollPane = new JScrollPane(etiquetteTigre);  
        add(scrollPane, BorderLayout.CENTER);  
    }  
}
```



Split.bat

- Panneau à compartiments, chaque compartiment est ajustable
- Seule une classe de look-and-feel est nécessaire.
- Panneau à séparation verticale ou horizontale
- Constructeurs



```
JSplitPane(int orientation, boolean dessinContinu, Component gauche,  
           Component droit);  
JSplitPane(int orientation, Component gauche, Component droit);  
JSplitPane(int orientation, boolean dessinContinu)  
JSplitPane(int orientation)  
JSplitPane() //horizontal par défaut
```

```
JSplitPane.HORIZONTAL_SPLIT  
JSplitPane.VERTICAL_SPLIT
```

# JSplitPane

```
ImageIcon bleue = new ImageIcon("bleue.gif");
aireGauche = new PanneauBoules(150, bleue.getImage());
ImageIcon rouge = new ImageIcon("rouge.gif");
aireDroite = new PanneauBoules(150, rouge.getImage());
JSplitPane sp = new JSplitPane( JSplitPane.HORIZONTAL_SPLIT, aireGauche, aireDroite);
sp.setDividerSize(5);
sp.setContinuousLayout(true);
getContentPane().add(sp, BorderLayout.CENTER);
```

- La taille de la barre de séparation peut être réglée par `setDividerSize(int taille)`
- L'affichage continue spécifié explicitement par `setContinuousLayout(boolean dessinContinu)`
- Poignée d'ouverture/fermeture spécifiées par `setOneTouchExpandable(boolean ouvrable)`

# JSplitPane (2)

```
public class Split extends JFrame {
    protected PanneauBoules aireGauche, aireDroite;

    public Split() { ...
        ImageIcon bleue = new ImageIcon("bleue.gif");
        aireGauche = new PanneauBoules(150, bleue.getImage());
        ImageIcon rouge = new ImageIcon("rouge.gif");
        aireDroite = new PanneauBoules(150, rouge.getImage());
        JSplitPane sp = new JSplitPane( JSplitPane.HORIZONTAL_SPLIT,
            aireGauche, aireDroite);
        sp.setDividerSize(5);
        sp.setContinuousLayout(true);
        getContentPane().add(sp, BorderLayout.CENTER);
        setVisible(true);
        ...
        new Thread(aireGauche).start();
        new Thread(aireDroite).start();
    }
}
```

# JSplitPane (3)

```
class PanneauBoules extends JPanel implements Runnable, ComponentListener {
    Boule[] boules;
    Image img;
    Dimension dim;
    int sommeil;

    public PanneauBoules(int nBoules, Image img) {
        sommeil = 10;
        this.img = img;
        setBackground(Color.yellow);
        setPreferredSize(new Dimension(200,300));
        addComponentListener(this);
        boules = new Boule[nBoules];
        dim = getPreferredSize();
        for (int k=0; k < nBoules; k++)
            boules[k] = new Boule(dim);
    }
    public void run() {...}
}
```

# JSplitPane (4)

## ■ Runnable

```
public void run() {
    for(;;) {
        for (int k = 0; k < boules.length; k++)
            boules[k].move(dim);
        repaint();
        if (sommeil != 0) {
            try {
                Thread.sleep(sommeil);
            }
            catch(InterruptedException e) {}
        }
    }
}
```

```
public void paintComponent(Graphics g) {
    g.setColor(getBackground());
    g.fillRect(0,0, dim.width, dim.height);

    for (int k=0; k<boules.length; k++)
        g.drawImage(img,
            (int)boules[k].x,
            (int)boules[k].y, this);
}
```

## ■ ComponentListener

```
public void componentHidden(ComponentEvent e){}
public void componentShown(ComponentEvent e){}
public void componentMoved(ComponentEvent e){}
public void componentResized(ComponentEvent e){
    dim = getSize();
    for (int k = 0; k < boules.length; k++)
        boules[k].moveIntoRect(dim);
}
```

# JSplitPane (5)

## ■ Les boules

```
class Boule {
    protected double x, y, vx, vy;
    public Boule(Dimension dim) {
        x = dim.width * Math.random();
        y = dim.height * Math.random();
        double angle = 2*Math.PI*Math.random();
        vx = 2*Math.cos(angle);
        vy = 2*Math.sin(angle);
    }

    public void move(Dimension dim) {
        double nx = x + vx;
        double ny = y + vy;
        if ((nx < 0) || (nx > dim.width)) {
            vx = - vx;
            nx = x + vx;
        }
        if ((ny < 0) || (ny > dim.height)) {
            vy = - vy;
            ny = y + vy;
        }
        x = nx;
        y = ny;
    }
}
```

```
public void moveIntoRect(Dimension dim) {
    x = Math.max(x, 0);
    x = Math.min(x, dim.width);
    y = Math.max(y, 0);
    y = Math.min(y, dim.height);
}
}
```

# JSplitPane : exemple simple



DesPanneaux.bat

- Panneaux mouvants emboîtés
- Accélérateurs de mouvements ont trois positions
  - fermé
  - partagé
  - ouvert



# JSplitPane : exemple simple

```
class Fragments extends JPanel {
    JSplitPane doubleur(int orientation, String un, String deux) {
        return new JSplitPane(orientation, new JButton(un), new JButton(deux));
    }

    Fragments() {
        setLayout(new BorderLayout(2,2));
        setBackground(Color.blue);
        JSplitPane p;
        p = doubleur(JSplitPane.HORIZONTAL_SPLIT, "Un", "Deux");
        p.setOneTouchExpandable(true);
        add(p, BorderLayout.NORTH);
        p = doubleur(JSplitPane.VERTICAL_SPLIT, "Trois", "Quatre");
        add(p, BorderLayout.WEST);
        add(doubleur(JSplitPane.VERTICAL_SPLIT, "Cinq", "Six"), BorderLayout.EAST);
        add(doubleur(JSplitPane.HORIZONTAL_SPLIT, "Sept", "Huit"), BorderLayout.SOUTH);
        p = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT, true,
            new JTextArea(10,10), new JTextArea());
        p = new JSplitPane(JSplitPane.VERTICAL_SPLIT, true, new JButton("haut"), p);
        p.setOneTouchExpandable(true);
        add(p, BorderLayout.CENTER);
    }
}
```

# JComboBox

---

- Cache la liste des entrées possibles à l'exception de l'entrée sélectionnée. Si elle est éditable, on peut taper dans la partie visible.
- L'entrée nouvellement sélectionnée déclenche un **ItemEvent**. Inversement, un changement dans les données provoque un **ListDataEvent**.
- Classes associées
  - **ComboBoxModel** contient les entrées et génère les **ListDataEvent**
  - **ComboBoxEditor** : l'éditeur utilisé dans une ComboBox éditable
  - **ListCellRenderer** : pour dessiner les entrées