Les containers spécialisés

- Environnement Graphique
- Les Fenêtres et Boite de dialogue
- LayeredPane et TabbedPane
- ScrollPane et JSplitPane

Les containers généraux

- Il existe en Swing deux containers généraux :
- **`JPanel`**
  - dérive de **`JComponent`**.
  - Contient un **`FlowLayout`** par défaut.
  - Gère en plus couleur d'avant plan et d'arrière plan.
  - Est opaque, ce qui importe pour les dessins.
- **`Box`**
  - dérive de **`JComponent`**.
  - Utilise par défaut un **`BoxLayout`**.
  - Est transparent.
  - Ne peut donc pas être utiliser comme *contentPane*.

Les containers spécialisés

- Les containers spécialisés sont des containers qui peuvent :
  - imposé un LayoutManager
  - imposé des fils particuliers

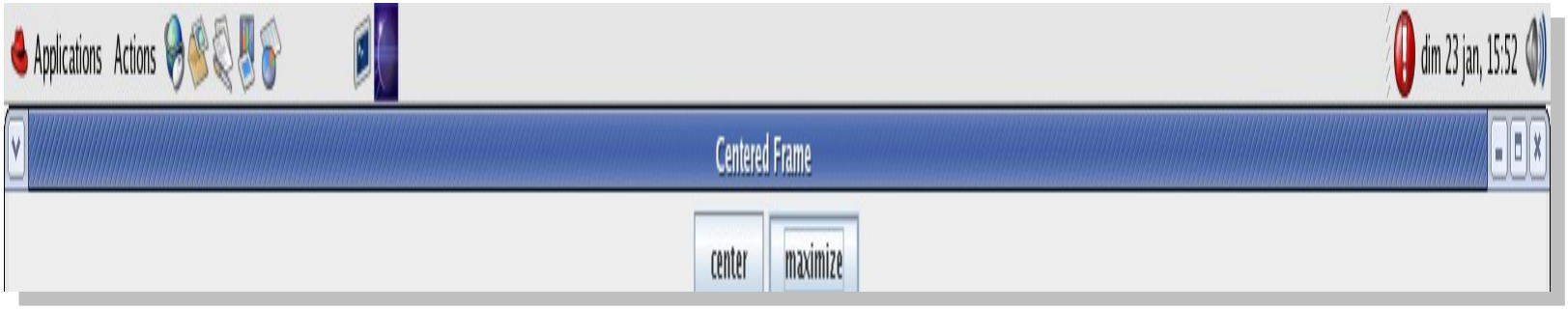
- Il y a plusieurs sortes :
  - Les fenêtres (JFrame, JDIalog, JInternalFrame)
  - Les conteneurs particuliers (JSplitPane, JTabbedPane)
  - Les conteneurs cachés (JRootPane, JLayeredPane, JDesktopPane)

Environnement graphique

- Le GraphicsEnvironnement permet d'obtenir :
  - Les fontes du système (Font)
    - Obtenir toutes les fontes
      Font[] getAllFonts()
    - Obtenir le nom des familles de fontes
      String[] getAvailableFontFamilyNames()
  - Les écrans graphiques (GraphicsDevice)
    - Obtenir l'écran graphique courant
      GraphicsDevice getDefaultScreenDevice()
    - Obtenir tous les écrans graphiques
      GraphicsDevice[] getScreenDevices()
- Obtenir l'environnement graphique courant :
  getLocalGraphicsEnvironment()

Centrer/Agrandir une fenêtre

- Pour centrer une fenêtre :
  la méthode getCenterPoint() renvoie le centre de l'écran, ou des écran (en multi-display)
- Pour maximiser une fenêtre :
  la méthode getMaximumWindowBounds() renvoie la taille de la fenêtre maximal en tenant compte des barres du bureau



Exemple

```
private static JButton createResizeButton(String text,final JFrame frame,
  final Rectangle bounds) {
  JButton button=new JButton(text);
  button.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      frame.setBounds(bounds);
    }
  });
  return button;
}

public static void main(String[] args) {
  GraphicsEnvironment env=GraphicsEnvironment.getLocalGraphicsEnvironment();
  Point center=env.getCenterPoint();
  Rectangle maximumBounds=env.getMaximumWindowBounds();

  JFrame frame=new JFrame("Centered Frame");
  JButton maximizeButton=createResizeButton("maximize",frame,maximumBounds);
  Rectangle centerBounds=new Rectangle(center.x-200,center.y-150,400,300).
    intersection(maximumBounds);
  JButton centerButton=createResizeButton("center",frame,centerBounds);
  ...
```

Les écran graphiques

- La classe GraphicsDevice représente un endroit sur lequel il est possible de dessiner :
  - un écran graphique
  - une imprimante
  - une image
- Chaque GraphicsDevice possède plusieurs configurations (GraphicsConfiguration).
- Chaque configurartion qui correspond un mode d'affichage (displayMode) particulier

Pleine écran

- Mettre une fenêtre en pleine écran :
  - mode supporté par la plateforme :
    boolean isFullScreenSupported()
  - met une fenêtre en plein écran :
    get/setFullScreenWindow(Window)
- Le mode plein écran :
  - met la fenêtre devant les autres (always on top)
  - la fenêtre occupe totalement l'écran et reçoit tous les évènements
  - la fenêtre est automatiquement rendu visible
- Si le mode n'est pas supporté, il est émulé
- Il est des fois préférable d'utiliser une JFrame plutôt qu'une JWindow

Exemple

```
GraphicsEnvironment env=GraphicsEnvironment.getLocalGraphicsEnvironment();
final GraphicsDevice screen=env.getDefaultScreenDevice();
System.out.println(screen.isFullScreenSupported());

final JFrame frame=new JFrame("example");
frame.setUndecorated(true);

JLabel label=new JLabel("FullScreen",SwingConstants.CENTER);
label.setFocusable(true);
label.setFont(new Font("SansSerif",Font.BOLD,96));
label.addKeyListener(new KeyAdapter() {
  public void keyTyped(KeyEvent e) {
    if (e.getKeyChar()=='q') {
      screen.setFullScreenWindow(null);
      frame.dispose();
    }
  }
});

frame.setContentPane(label);
screen.setFullScreenWindow(frame);
```

Mode graphique et pleine écran

- Un GraphicDevice possède plusieurs mode d'affichage :
  - Le mode d'affichage par défaut :
    DisplayMode getDisplayMode()
  - Les modes existants :
    DisplayMode[] getDisplayModes()
  - Changer de mode graphique :
    isDisplayChangeSupported()
    setDisplayMode(DisplayMode mode)
- Changer le mode d'affichage ne marche souvent que lors que l'on est en mode plein écran non émulé (bref sous Windows)

Mode graphique

- La classe DisplayMode correspond à un mode graphique :
  - une largeur : getWidth()
  - une hauteur : getHeight()
  - un nombre de bits pour les couleurs 8/16/24/32 : getBitDepth()
  - une féquence de raffraichissement en Hz : getRefreshRate()

```
GraphicsEnvironment env=GraphicsEnvironment.getLocalGraphicsEnvironment();
if (env.isHeadlessInstance())
  throw new IllegalStateException("pas de serveur graphique détecté");

GraphicsDevice[] devices=env.getScreenDevices();
for(GraphicsDevice device:devices)
  for(DisplayMode mode:device.getDisplayModes())
    System.out.println(device+" "+mode.getWidth()+"x"+mode.getHeight());
```

Configuration graphique

- Une configuration graphique correspond à une configuration particulière d'un écran graphique
- Tous les composants heavyweights sont crées à partir de cette configuration
- Obtenir la configuration par défaut : graphicsDevice.getDefaultConfiguration()
- Obtenir toutes les configurations : graphicsDevice.getConfigurations()

Configuration graphique (2)

- A partir d'une configuration graphique, il est possible de :
  - obtenir la taille de l'écran
    - rectangle getBounds()
  - créer des images compatibles avec la configuration :
    - image en mémoire
      createCompatibleImage(int width, int height)
    - image dans la carte video :
      createCompatibleVolatileImage(int width, int height)
  - La transformation affine par défaut (cf Dessin)
    - getDefaultTransform()
  - Les capacités de la carte graphique :
    - pour les images : getImageCapabilities()
    - pour la gestion des buffers : getBufferCapabilities()

Fenêtre principale

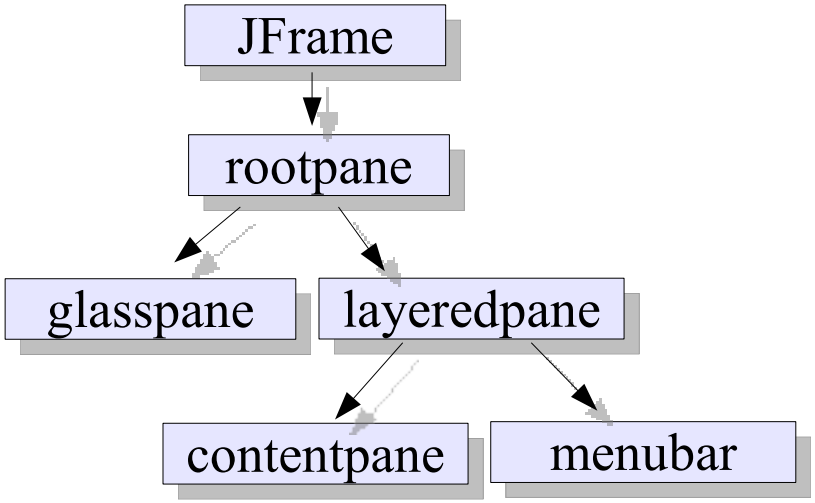
- Il existe en Swing quatre containers principaux
  - JWindow : fenêtre sans bordure
  - JFrame : fenêtre *top-level*
  - JDialog : boite de dialogue
  - JApplet : conteneur gérer par un navigateur

- Ces conteneurs implante l'interface RootPaneContainer et possède donc comme unique composant fils un composant JRootPane

JRootPane

- Composant qui est le seul fils possible pour **JWindow**, **JDialog**, **JFrame**, **JApplet** et **JInternalFrame**.

- Un **JRootPane** a deux fils : *glasspane* et *layeredpane.* Le *layeredpane* à aussi deux fils *menubar* et *contentpane*.



- Les *menuBar* et *contentPane* sont créés et gérés par **JRootPane**.

JRootPane (2)

- Le *glasspane* :
  composant transparent héritant de JComponent placé devant tous les composants du *contentpane*, il permet d'intercepter les évènements souris avant que ceux-ci soit distribués
- Le *layeredpane* :
  container sachant gérer la notion de profondeur
- Le *menubar* :
  barre de menu de l'application dont l'ensemble des menus sera affiché devant le *contentPane*
- le *contentPane* :
  zone permettant d'ajouter de nouveaux composants

JWindow

- Fenêtre *top-level* sans bordure ni bouton de gestion de fenêtre et qui par défaut ne gère pas les évènements
- Constructeurs :
  JWindow()
  JWindow(GraphicsConfiguration)
- Est utilisé habituellement :
  - pour les splashscreen
  - pour des fenêtres popup heavyweight, comme par exemple la notification de mail

JWindow (2)

- Propriétés des fenêtres JWindow :
  - placement de la fenêtre :
    - relative par rapport à un composant (coller à coté) setLocationRelativeTo(Component)
    - centrée setLocationRelativeTo(null)
    - indique si on doit laissé le window manager placer le fenêtre setLocationByPlatform(boolean)
  - Mettre la fenêtre toujours en avant plan is/setAlwaysOnTop(boolean)
  - changer le curseur get/setCursor(Cursor)
- JFrame hérite de toutes ces propriétés

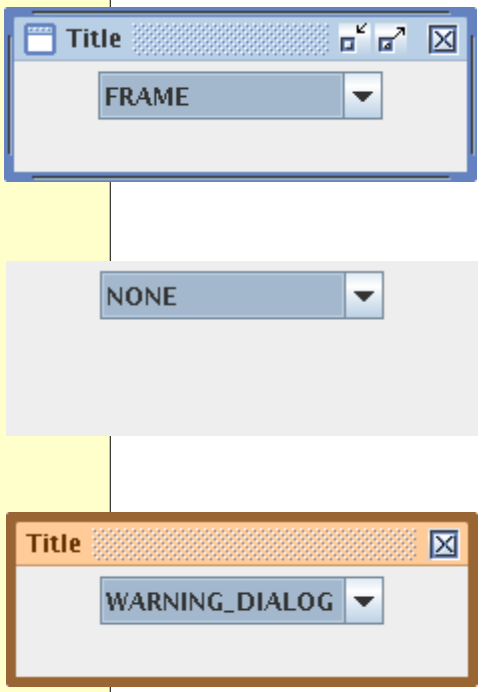
JFrame

- JFrame correspond à une fenêtre *top-level* avec bordure, hérite de Frame qui hérite de Window
- Constructeurs :
  JFrame(String title)
  JFrame(String title,GraphicsConfiguration)
- Méthodes :
  - changer l'icône : get/setIconImage(Image)
  - ne pas avoir de bordure : is/setUndecorated()
  - être resizable/icônifiable : is/setResizable, is/setIconifiable
  - changer l'état de la fenêtre : get/setExtendedState(int state)
    parmis
    NORMAL, ICONIFIED, MAXIMIZED_HORIZ, MAXIMIZED_VERT, MAXIMIZED_BOTH

Fenêtre décorée/non décorée

- Il est possible de gérer les décoration au niveau LaF

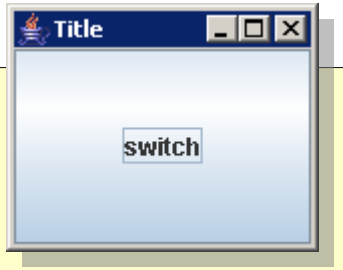
```
enum DecorationType {
  FRAME(JRootPane.FRAME), NONE(JRootPane.NONE),
  PLAIN_DIALOG(JRootPane.PLAIN_DIALOG),
  ERROR_DIALOG(JRootPane.ERROR_DIALOG),
  WARNING_DIALOG(JRootPane.WARNING_DIALOG);
  public DecorationType(int value) {
    this.value=value;
  }
  public void apply(JFrame frame) {
    frame.getRootPane().setWindowDecorationStyle(value);
  }
  private final int value;
}
public static void main(String[] args) {
  final JFrame frame=new JFrame("Title");
  frame.setUndecorated(true);
  final JComboBox combo=new JComboBox(DecorationType.values());
  combo.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      ((DecorationType)combo.getSelectedItem()).apply(frame);
    }
  });
```

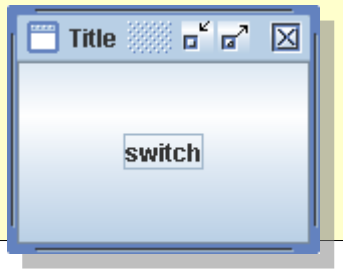


Decoration dynamique

- Les décorations ne peuvent être changée si le *peer* de la fenêtre a été créé (setVisible()/dispose())

```
final JFrame frame=new JFrame("Title");
JButton button=new JButton("switch");
button.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {
    frame.dispose();
    if (frame.isUndecorated()) {
      frame.setUndecorated(false);
      frame.getRootPane().setWindowDecorationStyle(JRootPane.NONE);
    } else {
      frame.setUndecorated(true);
      frame.getRootPane().setWindowDecorationStyle(JRootPane.FRAME);
    }
    frame.setVisible(true);
  }
});
frame.setContentPane(button);
frame.setSize(400,300);
frame.setVisible(true);
```





JDialog

- JDialog correspond à une fenêtre fille de l'application, elle correspond à un dialogue entre l'application et l'utilisateur
- Constructeurs :
  JDialog(Frame owner,String title,boolean modal,GraphicsConfiguration)
  JDialog(JDialog owner,String title,boolean modal,GraphicsConfiguration)
- Une JDialog est **modale** si elle ne permet pas composant des autres fenêtre de recevoir des évènements (sauf affichage)

JOptionPane

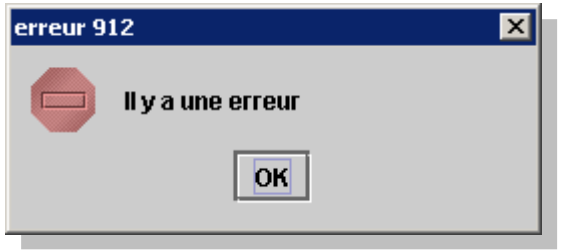
- La classe **`JOptionPane`** a les méthodes de fabrique (*factory*) statiques
  - **`showMessageDialog`**
  - **`showConfirmDialog`**
  - **`showInputDialog`**
  - **`showOptionDialog`** (permet de créer les 3 autres types)
- Une boîte de dialogue contient les champs
  - une icône/un message/un titre
  - un ou plusieurs boutons d'options
  - et, pour les **`InputDialog`**, un champ
    - de texte
    - ou une liste déroulante

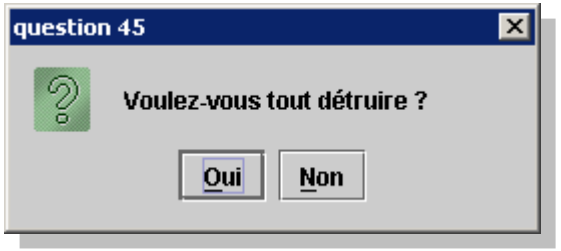
JOptionPane - 2

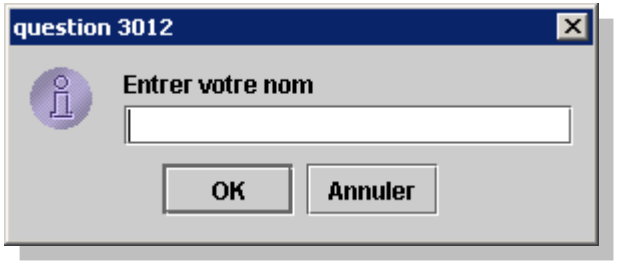
- Toute les méthodes show créée un boites de dialogue **`JDialog`** modale.
- La valeur de retour d'une boite à question :
  - **`YES_OPTION`**
  - **`NO_OPTION`**
  - **`CANCEL_OPTION`**
  - **`OK_OPTION`**
  - **`CLOSE_OPTION`** (bouton de fermeture)
- La valeur de retour d'une saisie d'information correspond au texte ou à null.

Catégorie de boite de dialogue

- Il existe trois types de boites de dialogues
  - Message d'Information ou d'Erreur
  - Question ou confirmation
  - Saisie utilisateur

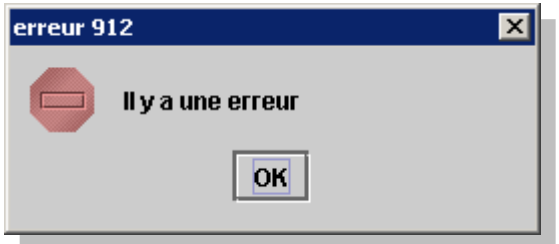




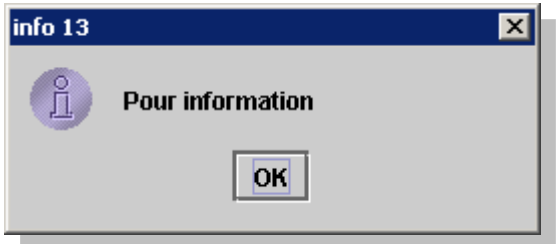


Message

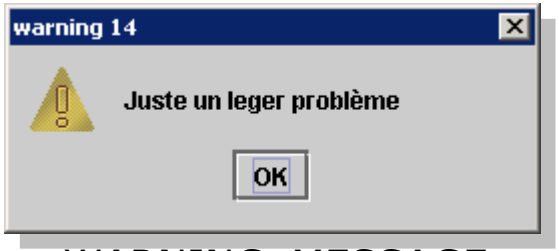
- Il existe quatres types de message :
- Information
  - Informe l'utilisateur
    - fin du travail
    - erreur
  - Simple confirmation



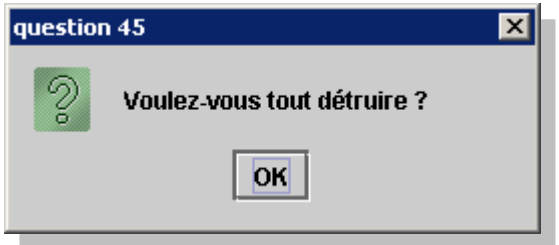
ERROR_MESSAGE



INFORMATION_MESSAGE



WARNING_MESSAGE



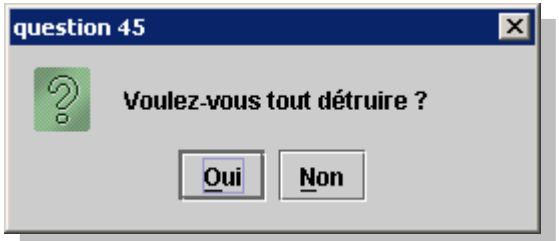
QUESTION_MESSAGE

```
JOptionPane.showMessageDialog(Component parent,
  Object message, String title,int type_de_message);
```

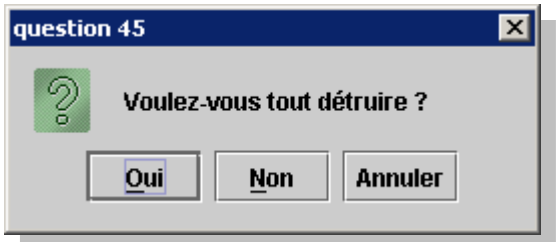
```
JOptionPane.showMessageDialog(frame,"Il y a une erreur",
  "erreur 912",JOptionPane.ERROR_MESSAGE);
```

Question

- Il existe deux types de questions
- Question
  - Prise de décision
    - abandon/annulation/confirmation
  - Réponse binaire ou ternaire



YES_NO_OPTION



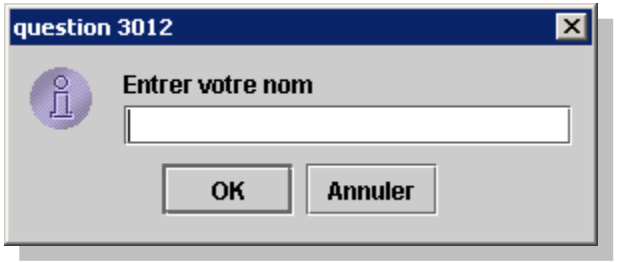
YES_NO_CANCEL_OPTION

```
JOptionPane.showConfirmDialog(Component parent,
  Object message, String title,
  int type_d_option, int type_de_message);
```

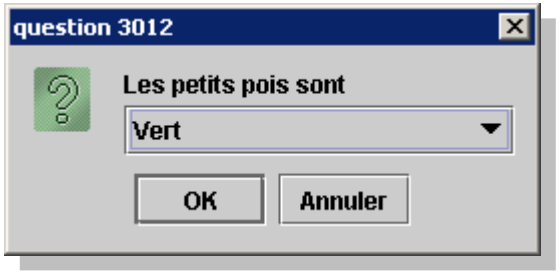
```
JOptionPane.showConfirmDialog(
  frame, "Voulez-vous tous détruire ?","question 45",
  JoptionPane.YES_NO_OPTION, JOptionPane.QUESTION_MESSAGE);
```

Saisie utilisateur

- Il existe principalement deux types de boite de saisie d'information



- Saisie d'information
  - saisie textuelle
    - nom
    - valeur
  - saisie par choix




```
JOptionPane.showInputDialog(Component parent,
  Object message, String title, int type_de_message);
```

```
JOptionPane.showInputDialog(Component parent,
  Object message, String title, int type_de_message,
  Icon icon, Object[] selectionValues,
  Object initialSelectionValue);
```

JOptionPane - Example

- Un exemple d'InputDialog

```
JButton button=new JButton("Add");
button.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {
    String text=JOptionPane.showInputDialog(
      (JComponent)e.getSource(),"Entrer un texte");
    if (text!=null)
       System.out.println(text);
  }
});
```

JFileChooser

- Permet à l'utilisateur de spécifier un ou plusieurs fichiers/répertoires
- Constructeur :
  JFileChooser(File currentDir,FileSystemView view)
- FileSystemView correspond à un vue Java d'un système de fichier particulier
- Utilise les classes du paquetage javax.swing.filechooser

FileSystemView

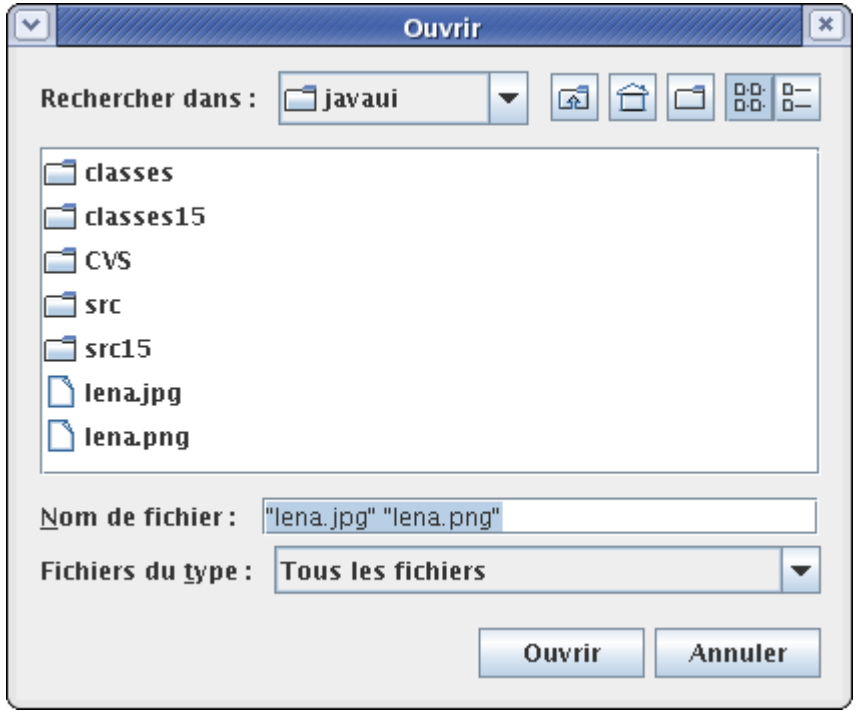
- La classe **java.io.File** du JDK ne permet pas certaines informations habituellement disponible dans un *file chooser*.
- **javax.swing.filechooser.FileSystemView** fourni :
  - La Home directory de l'utilisateur courant :
    File getHomeDirectory()
  - Lister un répertoire mais sans les fichiers cachées :
    File[] getFiles(File directory, boolean useFileHiding)
  - Les disques/racines du SGF :
    File[] getRoots()
  - Affichage usuel, description du type de fichier, icône associée :
    getSystemDisplayName(File), getSystemTypeDescription(File), getSystemIcon(File)

Mettre des données/Afficher/Sortir le résultat

- Afficher la boite de dialog :
  - showOpenDialog(Component parent)
  - showSaveDialog(Component parent)
  - showDialog(Component parent,String text) [version générique]
- La valeur de retour :
  APPROVE_OPTION, CANCEL_OPTION, ERROR_OPTION (si l'on ferme avec la croix)
- Indique une sélection multiple :
  is/setMultiSelectionEnabled(boolean)
- Fichiers sélectionnées par l'utilisateur :
  get/setSelectedFile(File), get/setSelectedFiles(File[])

Open et Save / Exemple

- Ouvrir de multiples fichiers et Sauver un seul fichier



Open et Save / Exemple

```
final JFrame frame=new JFrame("OpenSaveExample");
final JFileChooser chooser=new JFileChooser(new File("."));
Action openAction=new AbstractAction("Open") {
  public void actionPerformed(ActionEvent e) {
    chooser.setMultiSelectionEnabled(true);
    int value=chooser.showOpenDialog(frame);
    if (value!=JFileChooser.APPROVE_OPTION)
      return;
    System.out.println(Arrays.toString(chooser.getSelectedFiles()));
  }
};
Action saveAction=new AbstractAction("Save") {
  public void actionPerformed(ActionEvent e) {
    chooser.setMultiSelectionEnabled(false);
    int value=chooser.showSaveDialog(frame);
    if (value!=JFileChooser.APPROVE_OPTION)
      return;
    System.out.println(chooser.getSelectedFile());
  }
};
```

Filtrer l'affichage

- Il y a deux façon de filtrer l'affichage :
  - en implantant la classe abstraite FileFilter

```
abstract class FileFilter {
  abstract boolean accept(File f);
  abstract String getDescription()
}
```

  - en indiquant un mode setFileSelectionMode(mode) parmi FILES_ONLY, DIRECTORIES_ONLY, FILES_AND_DIRECTORIES

Filtrer affichage - 2

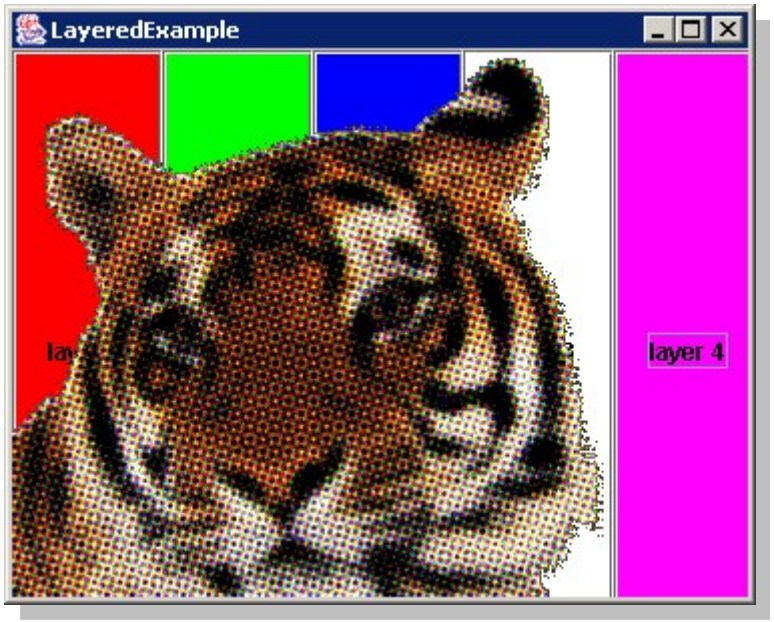
```
final JFrame frame=new JFrame("OpenSaveExample");
File file=new File(".");
final JFileChooser chooser=new JFileChooser(file);
chooser.setFileSelectionMode(JFileChooser.FILES_ONLY);
chooser.setFileFilter(new FileFilter() {
  public boolean accept(File f) {
    String name=f.getName().toLowerCase();
    return name.endsWith(".gif") || name.endsWith(".jpg");
  }
  public String getDescription() {
    return "select only GIF and JPG";
  }
});
Action openAction=new AbstractAction("Open") {
  public void actionPerformed(ActionEvent e) {
    int value=chooser.showOpenDialog(frame);
    if (value!=JFileChooser.APPROVE_OPTION)
       return;
    System.out.println(chooser.getSelectedFile());
  }
};
```

JLayeredPane

- Conteneur général pour des composants en couche.
- On peut donner une valeur de profondeur comme contrainte aux composants (attention à l'auto-boxing)

```
layeredPane.add(button,Integer.valueOf(3));
```

- L'affichage s'effectue en ordre croissant
- Les composants doivent être positionnés avec `setBounds()`.



Méthodes

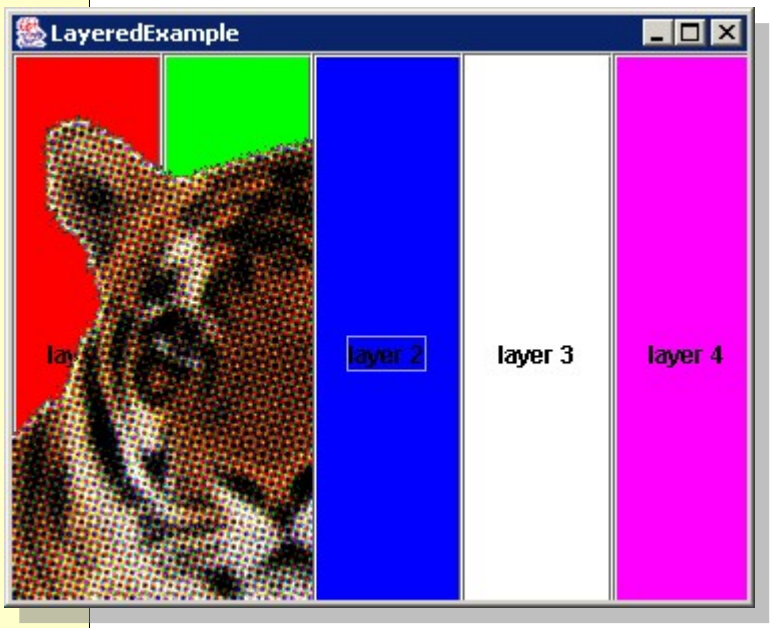
- setLayer(composant,int level)
- moveToFront(), moveToBack()

JLayeredPane (2)

- La **profondeur** d'une couche est utilise un `Integer`.
- Six profondeurs sont 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

JLayeredPane – Exemple

```
final JLayeredPane pane=new JLayeredPane();
final JLabel tiger=new JLabel(
  new ImageIcon("tiger.gif"));
Dimension prefs=tiger.getPreferredSize();
tiger.setBounds(0,0,prefs.width,prefs.height);
pane.add(tiger,Integer.valueOf(5));

Color colors[]=new Color[] {
  Color.RED,Color.GREEN,Color.BLUE,
  Color.WHITE,Color.MAGENTA
};
for(int i=0;i<colors.length;i++) {
  JButton button=new JButton("layer "+i);
  button.setBackground(colors[i]);
  final int index=i;
  button.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      pane.setLayer(tiger, index);
    }
  });
  button.setBounds(i*75,0,75,300);
  pane.add(button,Integer.valueOf(i));
}
```



- **setLayer()** permet d'assigner à un composant un niveau

JInternalFrame

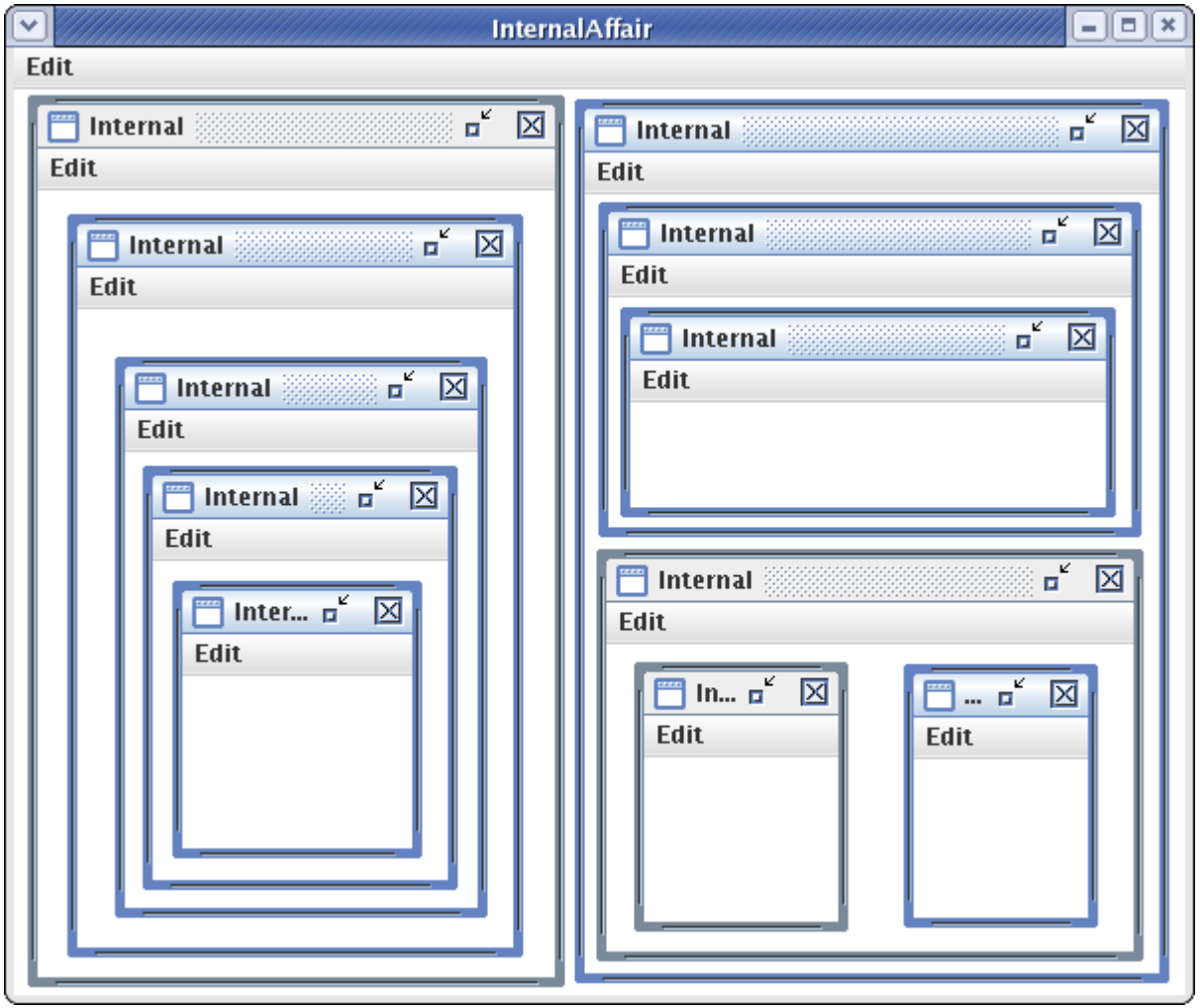
- Les fenêtre internes sont des fenêtres que l'on peut placer à l'intérieur d'une fenêtre existante (JFrame, JDialog, etc)
- Constructeur :
  JInternalFrame(String title, boolean resizable,closable,maximisable,iconifiable)
- JInternalFrame est un RootPaneContainer il possède donc un glasspane, un contentpane, etc.
- Les **JInternalFrame** :
  - n'ont pas de taille prédéfinie
  - ne sont pas visibles par défaut (à partir du JDK1.3)
  - doivent avoir comme père un JDesktopPane pour pouvoir être déplacée comme des fenêtre classiques

JDesktopPane

- Conteneur pour gérer les fenêtres internes (**`JInternalFrames`**).
- Hérite de **`JLayeredPane`**.
- Comme **`JLayeredPane`**, les fenêtres internes doivent être positionnées avec **`setBounds()`**.
- Utilise un objet de l'interface **`DesktopManager`** pour gèrer les opérations (maximiser,iconifier, etc) sur les **`JInternalFrames`**. **`DefaultDesktopManager`** est l'implémentation par défaut

Un exemple de JInternalFrame

- Un exemple récursif



Exemple de JInternalFrame

- Un exemple récursif

```
static Action createAddAction(final JDesktopPane desktop) {
  return new AbstractAction("Add...") {
    public void actionPerformed(ActionEvent e) {
      JInternalFrame jif=new JInternalFrame("Internal",true,true,false,true);
      JDesktopPane contentPane=new JDesktopPane();
      jif.setJMenuBar(createMenuBar("Edit",createAddAction(contentPane)));
      jif.setContentPane(contentPane);
      jif.pack();
      jif.setVisible(true);
      desktop.add(jif);
      desktop.revalidate();
    }
  };
}
```

```
static JMenuBar createMenuBar(
  String title,Action... actions) {

  JMenu menu=new JMenu(title);
  for(Action action:actions)
    menu.add(action);
  JMenuBar bar=new JMenuBar();
  bar.add(menu);
  return bar;
}
```

```
public static void main(String[] args) {
  JFrame frame=new JFrame("InternalAffair");
  JDesktopPane desktop=new JDesktopPane();
  frame.setJMenuBar(createMenuBar("Edit",createAddAction(desktop)));
  frame.setContentPane(desktop);
  frame.setSize(400,300);
  frame.setVisible(true);
}
```

JTabbedPane

- Conteneur de composants repérés par des onglets.
- Création:

```
JTabbedPane(int placementOnglet, int layoutOnglet)
```

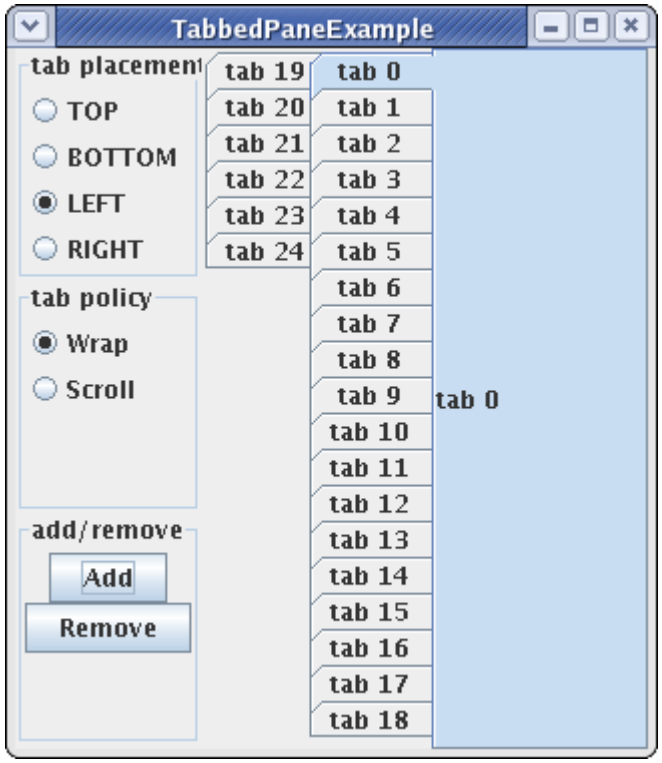
placement: TOP, BOTTOM, LEFT, RIGHT
layout: à la ligne : WRAP_TAB_LAYOUT,
scroll : SCROLL_TAB_LAYOUT

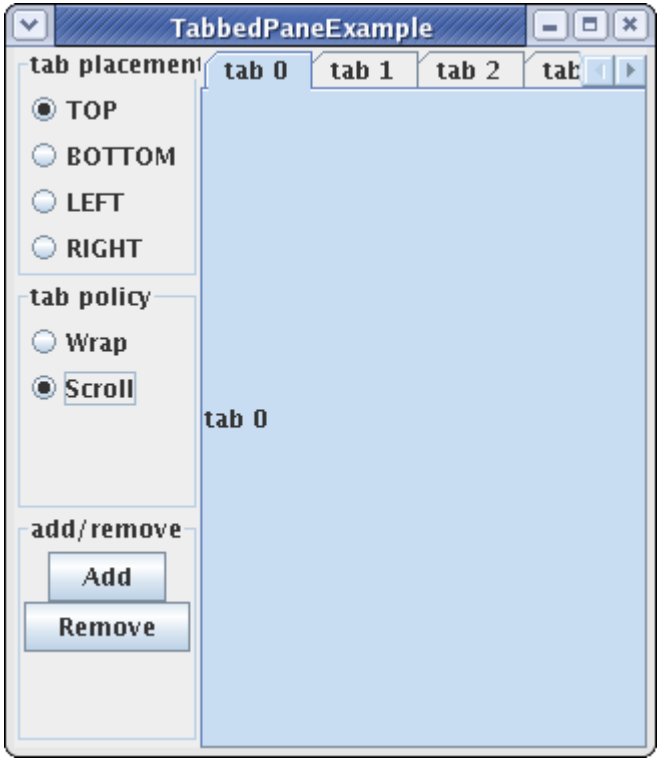
- Ajout de conteneurs à la boite à onglet :

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

JTabbedPane - 2

- Placement des onglets dynamiquement : setTabPlacement(int placement)
- Changement de politique d'affichage des onglets : setTabLayoutPolicy(int policy)





JTabbedPane - Onglets

- Ajouter ou retirer des onglets

```
private static Box createAddRemoveControls(final JTabbedPane pane) {
  Box box=new Box(BoxLayout.Y_AXIS);
  box.setBorder(BorderFactory.createTitledBorder("add/remove"));
  JButton add=new JButton("Add");
  add.setAlignmentX(0.5f);
  add.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      String text="tab "+(count++);
      pane.addTab(text, new JLabel(text));
    }
  });
  box.add(add);
  JButton remove=new JButton("Remove");
  remove.setAlignmentX(0.5f);
  remove.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      int index=pane.getSelectedIndex();
      if (index!=-1)
        pane.removeTabAt(index);
    }
  });
  box.add(remove);
  return box;
}
static int count;
```

JTabbedPane – Placement des onglets

- Changer la zone de placement des onglets

```
private static Box createTabPlacementControls(final JTabbedPane pane) {
  int[] placements=new int[] {
    JTabbedPane.TOP, JTabbedPane.BOTTOM,
    JTabbedPane.LEFT,JTabbedPane.RIGHT
  };
  String[] labels=new String[] {
    "TOP","BOTTOM","LEFT","RIGHT"
  };
  Box box=new Box(BoxLayout.Y_AXIS);
  box.setBorder(BorderFactory.createTitledBorder("tab placement"));
  ButtonGroup group=new ButtonGroup();
  for(int i=0;i<placements.length;i++) {
    JRadioButton button=new JradioButton(labels[i]);
    group.add(button);
    final int placement=placements[i];
    button.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        pane.setTabPlacement(placement);
      }
    });
    box.add(button);
  }
  return box;
}
```

JTabbedPane – Placement des onglets (2)

- Changer la politique de gestion des onglets : même code que pour le placement des onglets
- Partager le code pour écrire un seul code :

```
interface Command {
  String name();
  void performs(JTabbedPane pane);
}
enum TabPlacement implements Command {
  TOP(JTabbedPane.TOP), BOTTOM(JTabbedPane.BOTTOM),
  LEFT(JTabbedPane.LEFT), RIGHT(JTabbedPane.RIGHT);
  public TabPlacement(int placement) {
    this.placement=placement;
  }
  public void performs(JTabbedPane pane) {
    pane.setTabPlacement(placement);
  }
  private final int placement;
}
```

Utilise le design-pattern *command*

JTabbedPane – code générique

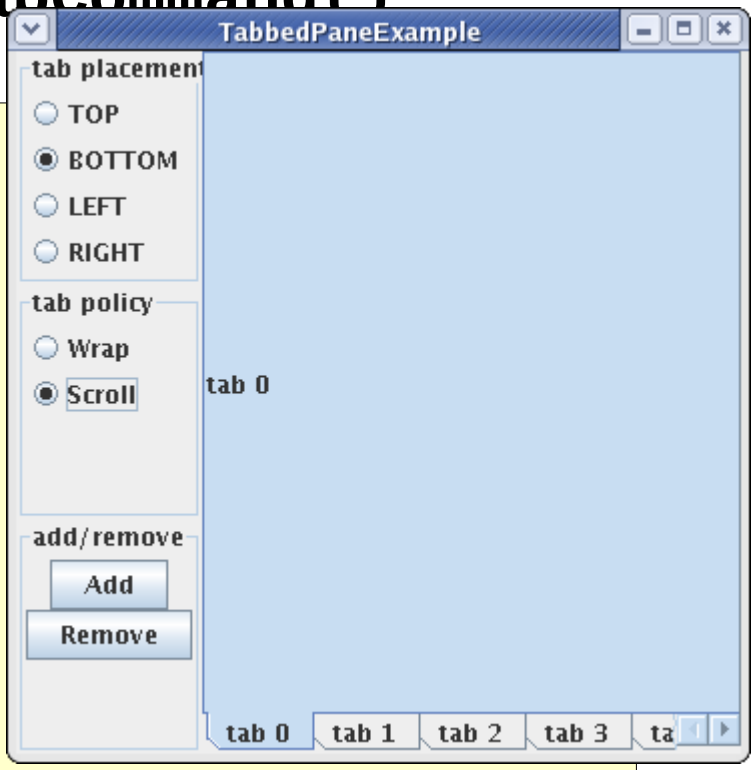
- Un seul code indépendant des commandes à effectuer
- Crée les boutons radios en fonction des commandes

```
private static Box createTabCommand(final JTabbedPane pane,
  String title,Command[] commands) {

  Box box=new Box(BoxLayout.Y_AXIS);
  box.setBorder(BorderFactory.createTitledBorder(title));
  ButtonGroup group=new ButtonGroup();
  for(final Command command:commands) {
    JRadioButton button=new JRadioButton(command.name());
    group.add(button);
    button.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        command.performs(pane);
      }
    });
    box.add(button);
  }
  return box;
}
```

JTabbedPane – Politique d'affichage des onglets

- Définie une autre commande en utilisant une énumération
- Puis on utilise la méthode **createTabCommand()**

```
enum TabPolicy implements Command {
  Wrap(JTabbedPane.WRAP_TAB_LAYOUT),
  Scroll(JTabbedPane.SCROLL_TAB_LAYOUT);
  public TabPolicy(int policy) {
    this.policy=policy;
  }
  public void performs(JTabbedPane pane) {
    pane.setTabLayoutPolicy(policy);
  }
  private final int policy;
}
public static void main(String[] args) {
  JTabbedPane pane=new JTabbedPane();

  JPanel panel=new JPanel(new GridLayout(3,0));
  panel.add(createTabCommand(pane,"tab placement",TabPlacement.values()));
  panel.add(createTabCommand(pane,"tab policy",TabPolicy.values()));
  panel.add(createAddRemoveControls(pane));
```



Composant comme onglet

- A partir de la version 1.6, il est possible de spécifier un composant comme onglet

- Et donc d'implanter la croix qui ferme l'onglet

Composant comme onglet

- On prépare les icônes et l'on crée les différents composants (**JTabbedPane**, **JTextField** et **Box** qui contient les deux)

```
BufferedImage image=ImageIO.read(MustangTab.class.getResource("close.jpg"));
final ImageIcon overCloseIcon=new ImageIcon(image);
image=new ColorConvertOp(
  ColorSpace.getInstance(ColorSpace.CS_GRAY),null).filter(image,null);
final ImageIcon closeIcon=new ImageIcon(image);

final JFrame frame=new JFrame("Tabbed with no Pain");
final JTabbedPane pane=new JTabbedPane();
final JTextField field=new JTextField();
field.addActionListener(new ActionListener() {
  ...
});
Box box=new Box(BoxLayout.X_AXIS);
box.add(new JLabel("Location: "));
box.add(field);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.add(pane);
frame.add(box,BorderLayout.NORTH);
frame.setSize(400,300);
frame.setVisible(true);
```

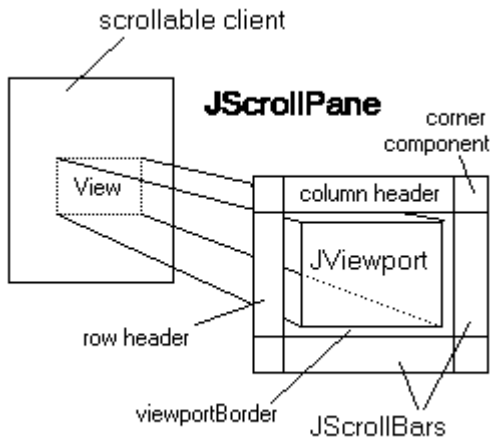
Composant comme onglet

```
public void actionPerformed(ActionEvent event) {
  final JTextPane textPane=new JtextPane();
  URI location;
  try {
    location = new URI(field.getText());
    text.setPage(location.toString());
  } catch (Exception e) {
    JOptionPane.showMessageDialog(frame,e.getMessage(),
      e.getClass().getName(),JOptionPane.ERROR_MESSAGE);
  }
  textPane.addTab(null,text); // on passe null ici
  JButton button=new JButton(closeIcon);
  button.setPreferredSize(new Dimension(
    closeIcon.getIconWidth(),closeIcon.getIconHeight()));
  button.setBorderPainted(false);
  button.setRolloverIcon(overCloseIcon);
  button.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      int index=textPane.indexOfComponent(text);
      textPane.removeTabAt(index);
    }
  });
  Box box=new Box(BoxLayout.X_AXIS);
  box.add(new JLabel(location.getHost()));
  box.add(Box.createHorizontalStrut(8));
  box.add(button);
  textPane.setTabComponentAt(pane.getTabCount()-1,box);
```

JScrollPane

- Gère automatiquement des ascenseurs autour de son composant central qui est un **JViewPort**.
- Constructeur

```
JScrollPane(Component view)
```

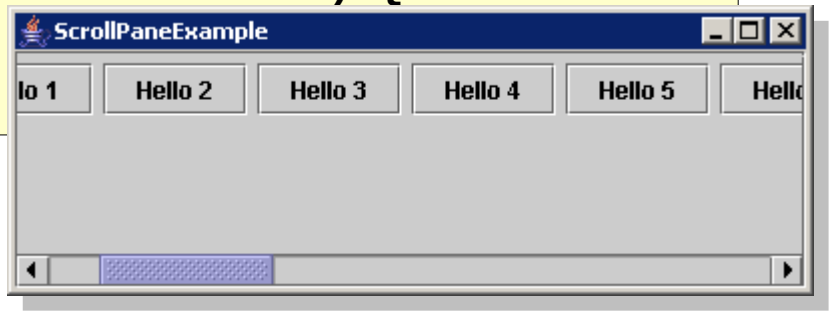


- La “vue” est gérée par le **JViewPort** et doit implanter l'interface **Scrollable**.
- Les entêtes (*headers*) scrolle en même temps que la vue

Scrollable

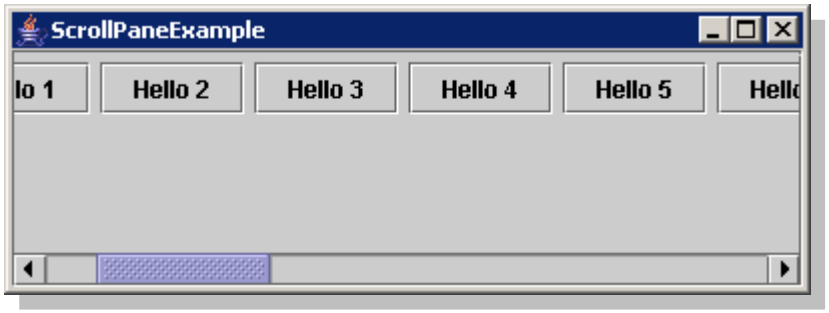
- Il faut implanter l'interface **Scrollable**.

```
public class ScrollPaneExample extends JPanel implements Scrollable {
  public boolean getScrollableTracksViewportHeight() {
    return true;
  }
  public boolean getScrollableTracksViewportWidth() {
    return false;
  }
  public Dimension getPreferredScrollableViewportSize() {
    return new Dimension(200,100);
  }
  public int getScrollableBlockIncrement(
    Rectangle visibleRect, int orientation, int direction) {
    return 30;
  }
  public int getScrollableUnitIncrement(
    Rectangle visibleRect, int orientation, int direction) {
    return 5;
  }
```



JScrollPane - Exemple

- On créé un **JScrollPane** en passant en paramètre le composant implantant l'interface **Scrollable**.




```
public static void main(String[] args) {
  ScrollPaneExample example=new ScrollPaneExample();
  for(int i=0;i<20;i++)
    example.add(new JButton("Hello "+i));
  JFrame frame=new JFrame("ScrollPaneExample");
  frame.setContentPane(
    new JScrollPane(example));
  frame.pack();
  frame.setVisible(true);
}
```

JSplitPane

- Panneau définissant deux compartiments ajustables suivant une séparation.
- Constructeur

```
JSplitPane(int orientation, boolean
 dessinContinu, Component gauche/haut,Component
 droit/bas);
```

- Orientation :
  séparation verticale (VERTICAL_SPLIT)
  séparation horizontale (HORIZONTAL_SPLIT)

JSplitPane - 2

- La taille de la barre de séparation peut être réglée par
  `setDividerSize(int)`
- Le poid d'agrandissement entre les zones
  `setResizeWeight(double)`
- L'affichage continue spécifié explicitement par
  `setContinuousLayout(boolean)`
- Poignée d'ouverture/fermeture spécifiées par
  `setOneTouchExpandable(boolean)`

JSplitPane - Example

- Créé récursivement des **JSplitPane** jusqu'à une certaine profondeur.

```
private static JComponent createSplits(int depth) {
  if (depth==0) {
    JLabel label=new JLabel();
    label.setOpaque(true);
    label.setBackground(randomColor());
    return label;
  }
  JSplitPane pane=new JSplitPane(
    randomOrientation(),true,
    createSplits(depth-1),
    createSplits(depth-1));
  pane.setOneTouchExpandable(true);
  pane.setResizeWeight(0.5);
  return pane;
}
private static final Random random=new Random(0);
```

```
private static Color randomColor() {
  return new Color(
    random.nextInt(256),
    random.nextInt(256),
    random.nextInt(256));
}
```

```
private static int randomOrientation() {
  return (random.nextInt(2)==0)?
    JSplitPane.HORIZONTAL_SPLIT:
    JSplitPane.VERTICAL_SPLIT;
}
```