

Interface Graphique

Mathieu RAYNAL

mathieu.raynal@irit.fr

<http://www.irit.fr/~Mathieu.Raynal>

Les couleurs

- Elles sont gérées par la classe Color

- Objet construit

- à partir de ses composantes
 - rouge/vert/bleu
 - alpha en option
- Soit avec des float (0.0-1.0) ou des int (0-255)

```
Color(float r, float g, float b)  
Color(float r, float g, float b, float a)  
Color(int r, int g, int b)  
Color(int r, int g, int b, int a)
```

- Ensemble de couleurs prédéfinies sous forme de static

Color.BLACK

Color.BLUE

Color.CYAN

Color.DARK_GRAY

Color.GRAY

Color.GREEN

Color.LIGHT_GRAY

Color.MAGENTA

Color.ORANGE

Color.PINK

Color.RED

Color.WHITE

Color.YELLOW

Les images / icônes

- Dans les interfaces, les images sont transmises au moyen de classes implémentant l'interface **Icon**
- La classe la plus utilisée : **ImageIcon**

ImageIcon(String filename)
ImageIcon(URL location)

- Une image peut être redimensionnée dans la classe **Image**
 - Méthode dans ImageIcon **Image getImage()**
 - Méthode de redimensionnement dans Image
- Image getScaledInstance(int width, int height, int hints)**
- Hints : algorithme utilisé : Image. **SCALE_DEFAULT**

Utiliser une fonte

- Classe **Font** pour gérer les fontes

```
Font(String name, int style, int size)
```

- Des familles de fontes prédéfinies

```
Font.SANS_SERIF
```

```
Font.SERIF
```

```
Font.DIALOG_INPUT
```

```
Font.DIALOG
```

```
Font.MONOSPACED
```

- Des styles

```
Font.BOLD
```

```
Font.ITALIC
```

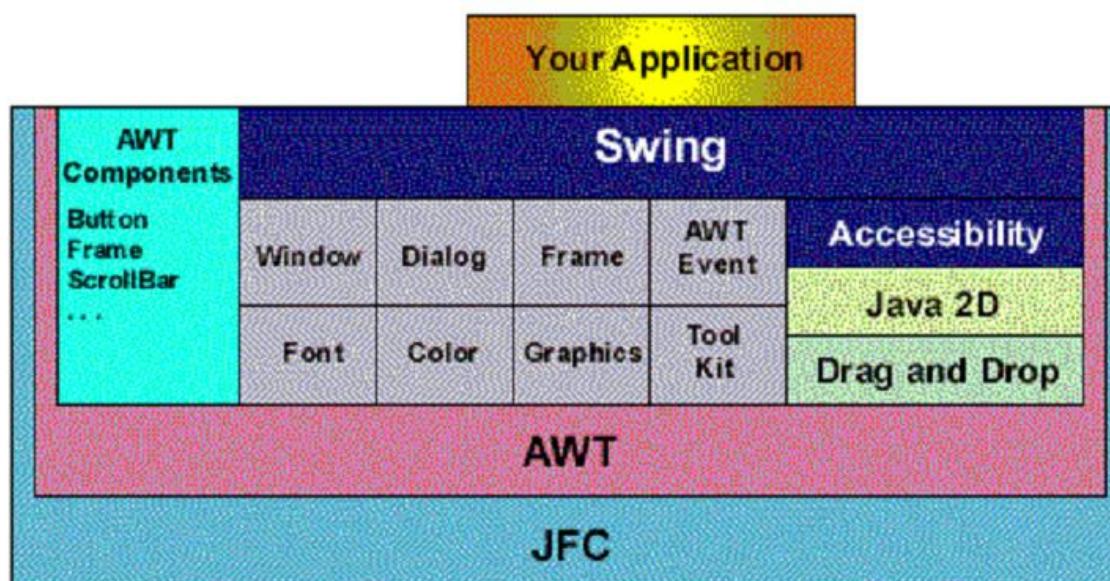
```
Font.PLAIN
```

- Pour connaître l'ensemble des fontes installées

```
Font [] allFonts = GraphicsEnvironment.getLocalGraphicsEnvironment().getAllFonts();
```

AWT et Swing

- AWT – Abstract Window Toolkit: depuis Java 1.1
- SWING depuis Java 1.2
- Graphique, composants, gestionnaire



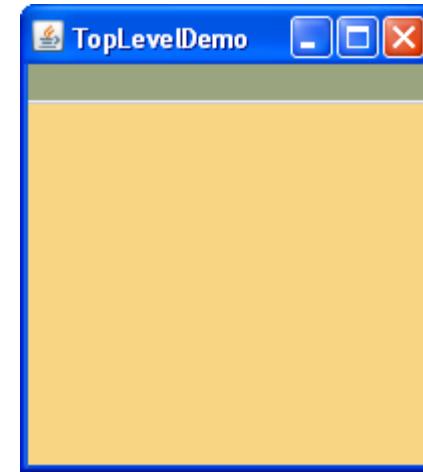
- Abstract Window Toolkit
- Conçue pour la portabilité des applications
- Ressources systèmes encapsulées par des abstractions
- Affichage délégué au système
 - Utilise le système de gestion des fenêtre du système
- Eléments graphiques ont l'apparence fournie par l'OS

SWING

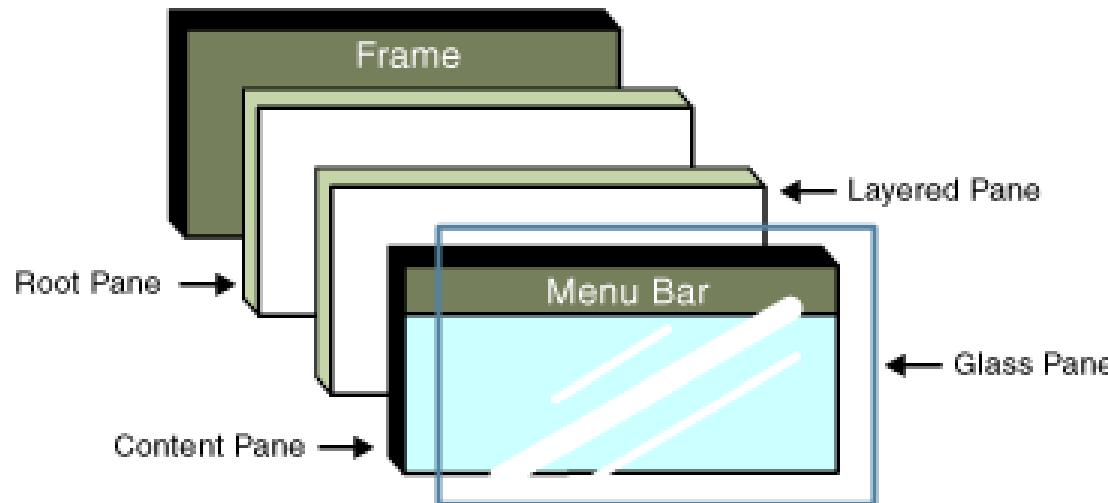
- Extension d'AWT
 - Architecture plus souple
 - Plus de possibilités graphiques
- Affichage non délégué au système.
- Un élément Swing peut
 - avoir la même apparence d'un système à l'autre
 - être paramétré: Pluggable Look and Feel
 - n'avoir aucun équivalent dans le système d'exploitation sur lequel il s'exécute

Une fenêtre

- Toute interface graphique a une fenêtre principale
- Elle contient
 - Un espace pour la barre de menu
 - Conteneur principal
 - Racine de l'arbre de composants
 - Contient les composants de l'interface



Les autres couches de la fenêtre



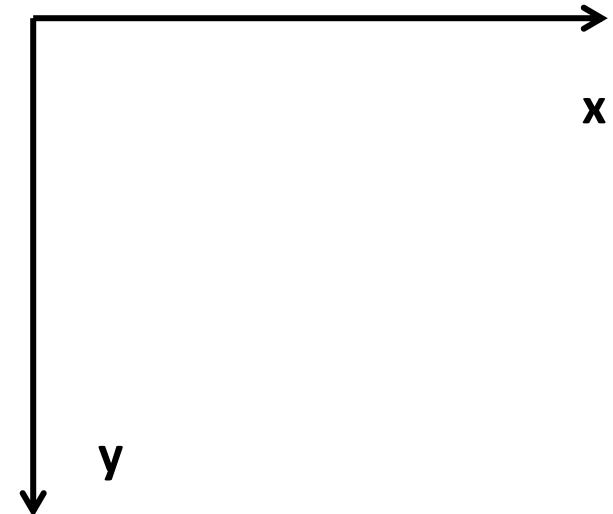
- **GlassPane**
 - Par défaut transparent
 - Possibilité d'ajouter des composants
 - Gérer des événements plus haut niveau
- **LayeredPane**
 - Permet de gérer les différentes couches

Accès aux éléments de la fenêtre

- Accès au Menu
 - JMenuBar getJMenuBar()
 - setJMenuBar(JMenuBar)
- Accès au conteneur principal
 - Container getContentPane()
 - setContentPane(Container)

Système de coordonnées

- Les fenêtres et composants sont positionnés par rapport à leur coin supérieur gauche
- La coordonnées (0,0) d'un élément est le coin supérieur gauche



Positionnement et taille d'une fenêtre

- Donner la dimension optimale à la fenêtre

```
public void pack()
```

- Positionner la fenêtre

```
public void setLocation(int x, int y)
```

- Dimensionner la fenêtre

```
public void setSize(int width, int height)
```

- Positionner et dimensionner la fenêtre

```
public void setBounds(int x, int y, int width, int height)
```

- Connaitre la taille de l'écran

```
Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
```

Bien fermer une interface graphique

- **public void setDefaultCloseOperation(int operation)**
 - DO NOTHING ON CLOSE
 - HIDE ON CLOSE
 - DISPOSE ON CLOSE
 - EXIT ON CLOSE
- **public void dispose()**
 - Libère la mémoire (tous les objets et références contenus via le garbage collector)

Principales étapes de construction

- Créer la fenêtre `JFrame frame = new JFrame("Titre");`

- Paramétrier la fermeture de la fenêtre

```
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

- Récupérer le conteneur principal

```
Container c = frame.getContentPane();
```

- Créer, paramétrier et positionner les différents composants

- Positionner et dimensionner la fenêtre

```
frame.pack();
```

- Rendre la fenêtre visible

```
frame.setVisible(true);
```

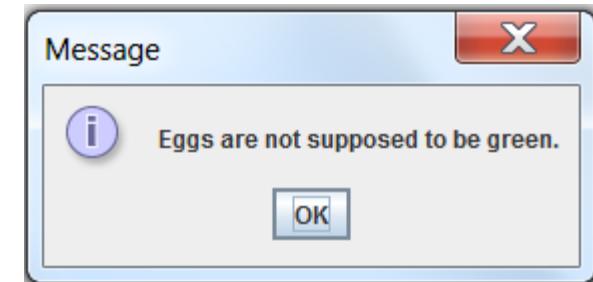
Boite de dialogue

- Classe : JDialog
- Elle vient en complément d'une fenêtre principale
- Elle a exactement le même fonctionnement que la JFrame
- Principale différence avec une Jframe
 - La fenêtre peut être modale
 - Bloque la fenêtre principale

JDialog(Frame owner, String title, boolean modal)

JOptionPane

- Production de boîtes de dialogues standardisées
- Modaux
- Plusieurs types
 - Messages d'erreur, d'information
 - Demande d'information
 - Choix
- Agencement type

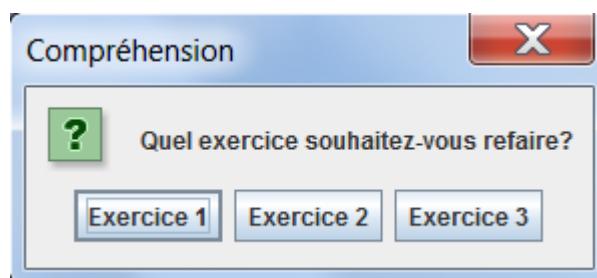


icon

message

input value

option buttons



Différents types de dialogues

- Demander une confirmation
 - `showConfirmDialog`
- Donner une information à l'attention de l'utilisateur
 - `showMessageDialog`
- Demander des informations particulières
 - `showInputDialog`
- Donner des informations et demander des actions
 - `showOptionDialog`

JOptionPane - showMessageDialog

- public static void **showMessageDialog**(



Component parentComponent,
Object message,
[String title,
int messageType],
[Icon icon])

```
ImageIcon ii = new ImageIcon("index.jpg");
Image i = ii.getImage().getScaledInstance(80, -1, java.awt.Image.SCALE_SMOOTH);
JOptionPane.showMessageDialog(ExempleOptionPane.this,
                            "J'aime le JAVA",
                            "Mon langage favori",
                            JOptionPane.PLAIN_MESSAGE,
                            new ImageIcon(i));
```

JOptionPane - Paramètres

- parentComponent : Fenêtre mère
 - JFrame, JDesktopPane, null
- message: contenu du message
- Title : titre de la fenêtre de dialogue

- messageType
 - ERROR_MESSAGE
 - INFORMATION_MESSAGE
 - WARNING_MESSAGE
 - QUESTION_MESSAGE
 - PLAIN_MESSAGE

Icons used by JOptionPane			
Icon description	Java look and feel	Windows look and feel	
question			
information			
warning			
error			

JOptionPane - showConfirmDialog

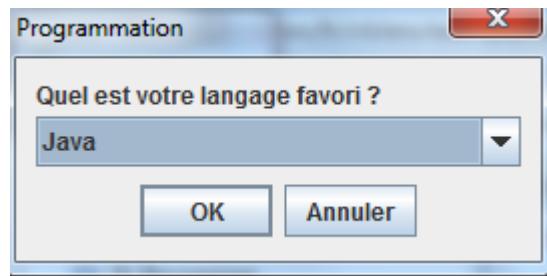
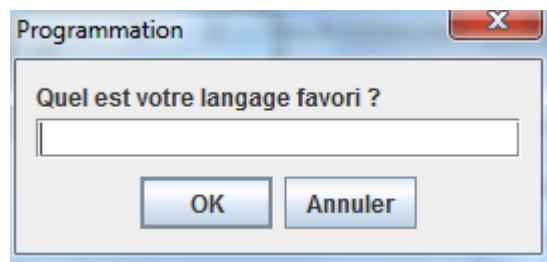
- public static int **showConfirmDialog**(
 Component parentComponent,
 Object message,
 [String title,
 int optionType],
 [int messageType],
 [Icon icon])

JOptionPane – Paramètres (2)

- optionType: configurer le nombre et le type de boutons
 - DEFAULT_OPTION
 - 1 bouton (“ok”)
 - YES_NO_OPTION
 - 2 boutons (“oui”, “non”)
 - YES_NO_CANCEL_OPTION
 - 3 boutons (“oui”, “non”, “annuler”)
 - OK_CANCEL_OPTION
 - 2 boutons (“ok”, “annuler”)
- Lors de l'appui sur un des boutons des boites de dialogue de JOptionPane
 - La boite de dialogue se ferme
 - La méthode static renvoie un entier
 - DEFAULT_OPTION (-1)
 - CLOSED_OPTION (-1)
 - OK_OPTION (0)
 - YES_OPTION (0)
 - NO_OPTION (1)
 - CANCEL_OPTION (2)

JOptionPane - showInputDialog

- public static Object showInputDialog(

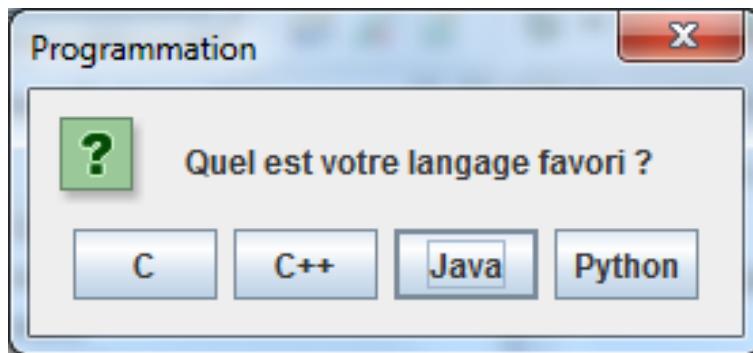


Component parentComponent,
Object message,
String title, int messageType,
Icon icon, Object[] selectionValues,
Object initialValue)

```
String[] liste = {"C","C++","Java","Python"};
Object o = JOptionPane.showInputDialog(ExempleOptionPane.this,
                                         "Quel est votre langage favori ?",
                                         "Programmation",
                                         JOptionPane.PLAIN_MESSAGE,
                                         null, liste, "Java");
```

JOptionPane - showOptionDialog

- public static int showOptionDialog(

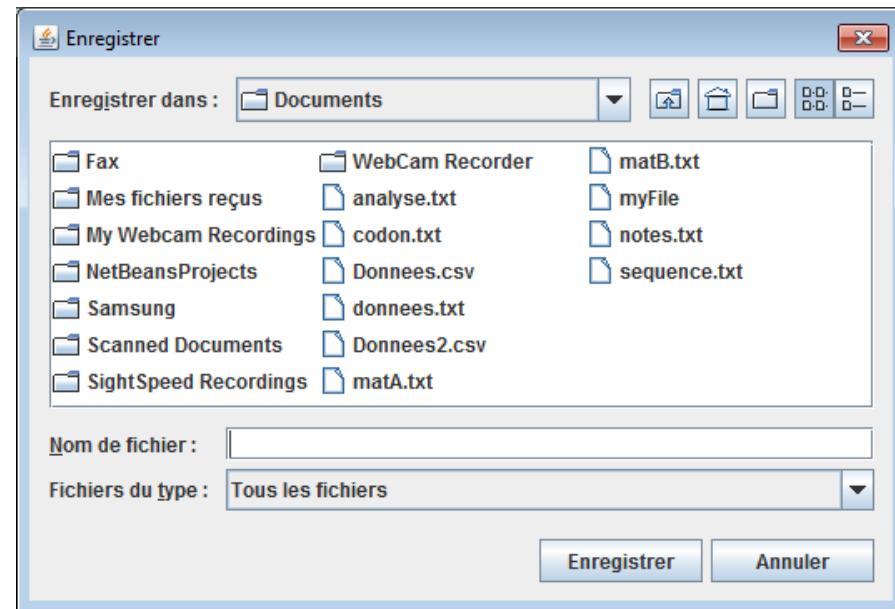


Component parentComponent,
Object message,
String title,
int optionType,
int messageType,
Icon icon, Object[] options,
Object initialValue)

```
String[] liste = {"C","C++","Java","Python"};
int n = JOptionPane.showOptionDialog(ExempleOptionPane.this,
                                    "Quel est votre langage favori ?",
                                    "Programmation",
                                    JOptionPane.DEFAULT_OPTION,
                                    JOptionPane.QUESTION_MESSAGE,
                                    null, liste, "Java");
```

JFileChooser

- JFileChooser chooser = new JFileChooser();
- Afficher la boite de dialogue
 - showOpenDialog
 - showSaveDialog
 - showDialog
- Choisir
 - un fichier
 - File getFileSelected()
 - Des fichiers
 - File[] getSelectedFiles()



Filtrer les types de fichier

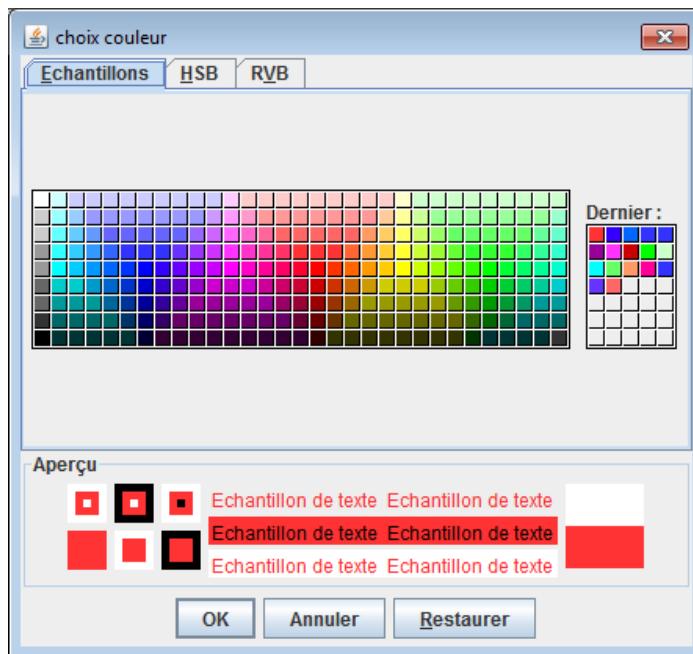
- `setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);`
 - Type de fichiers affichés
 - FILES_AND_DIRECTORIES
 - FILES_ONLY
 - DIRECTORIES_ONLY
- Etendre FileFilter
 - Masquer public boolean accept(File f)
 - Renvoie true si le fichier doit être affiché
 - False sinon
- `addChoosableFileFilter(FileFilter);`

JColorChooser

public static Color showDialog(

Component component,
String title,
Color initialColor)

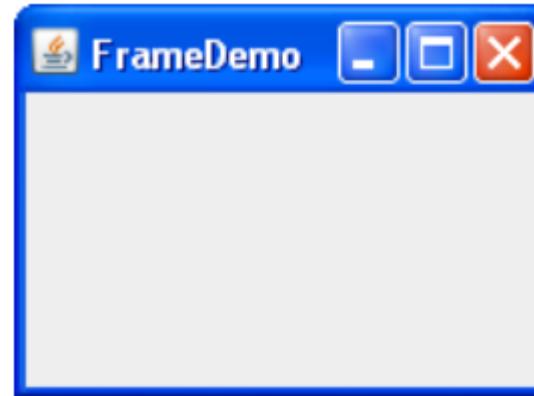
throws HeadlessException



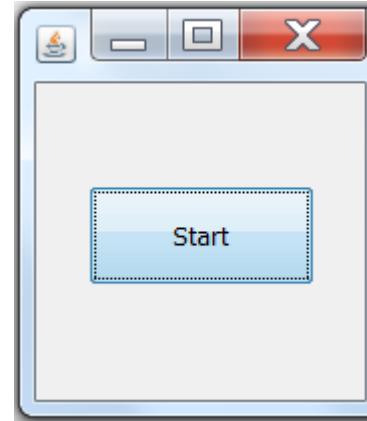
Principaux composants

Types de composants

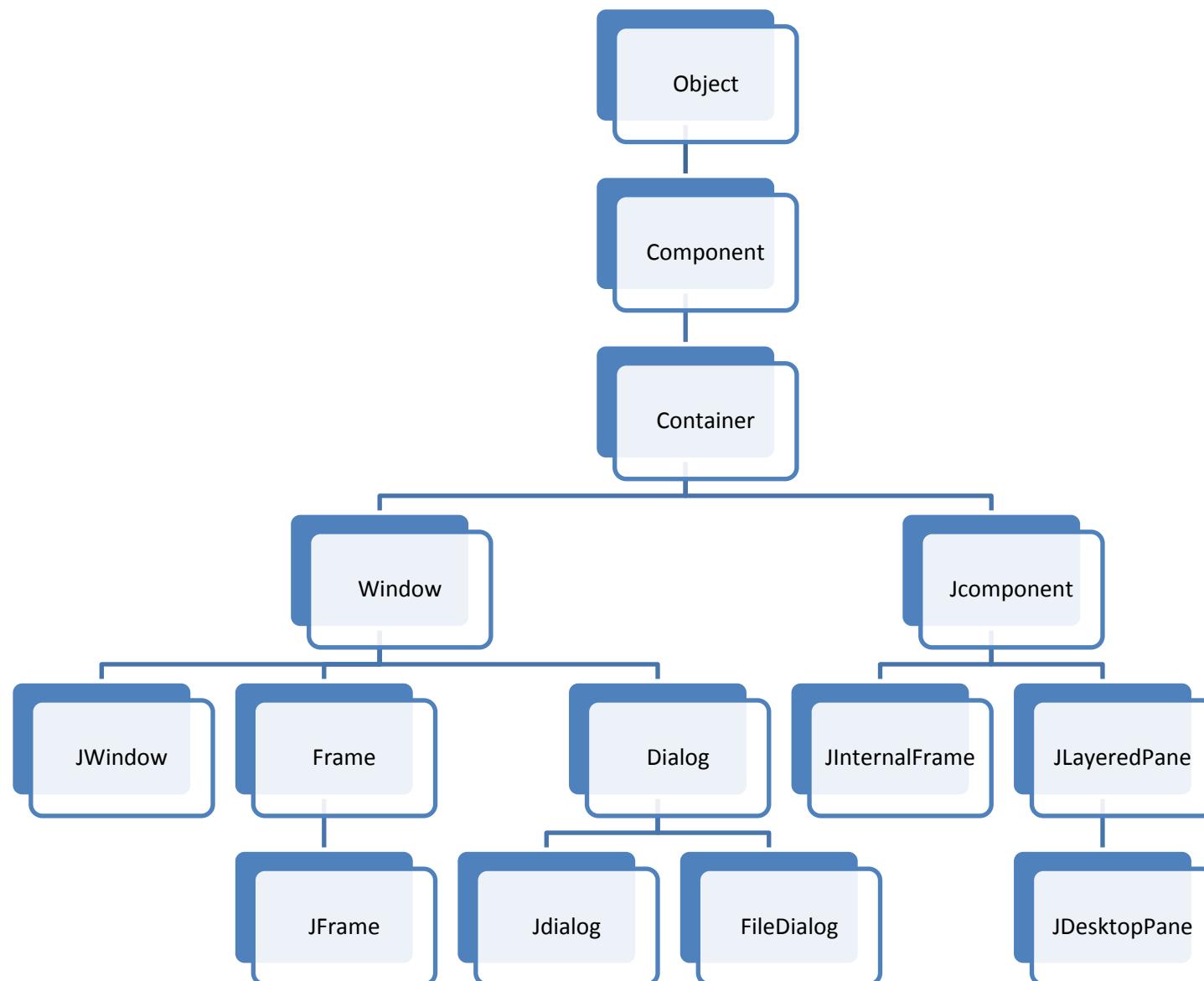
- Containers
 - JPanel
 - JSplitPane
 - JTabbedPane



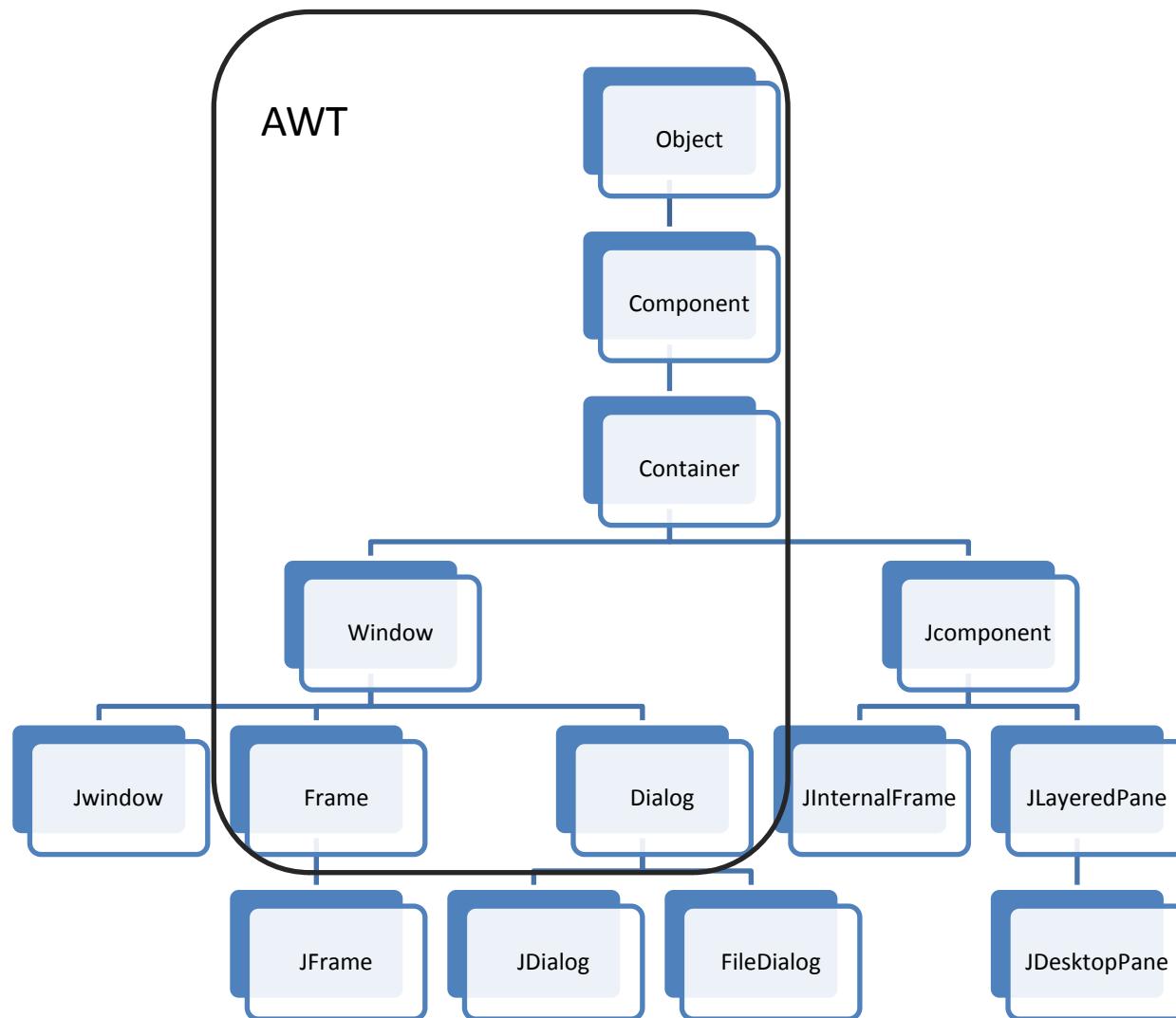
- Interacteurs
 - JButton
 - JCheckBox
 - JRadioButton
 - JList
 - JComboBox



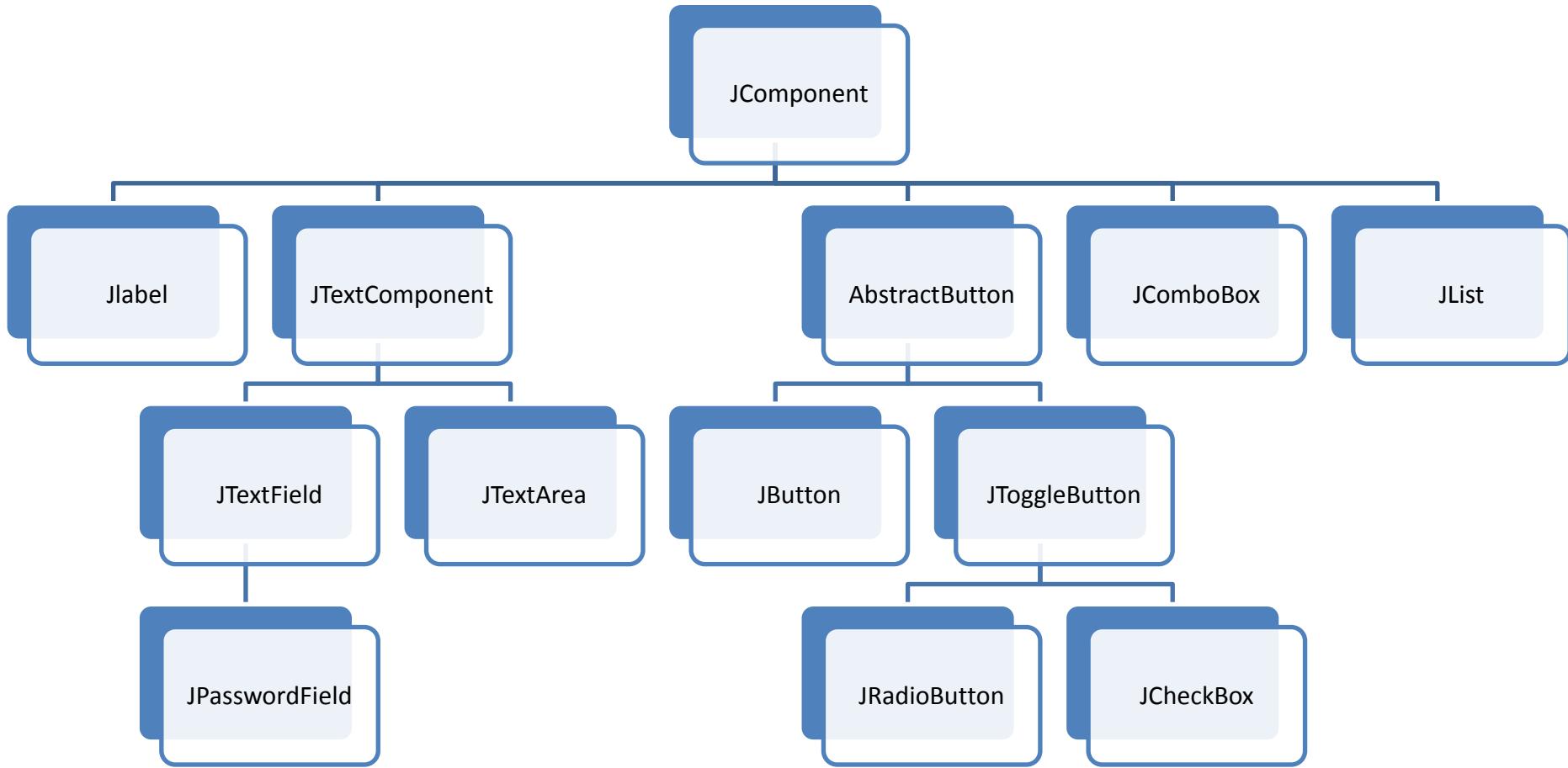
Hiérarchie de classes



Rappel: hiérarchie de classes SWING



JComponent



Caractéristiques communes à tous les composants

- Apparence (Bordure, Couleur, Font)
 - Taille, position
 - Disposition (layout)
 - Etat (nom, visible/invisible, actif/inactif,...)
 - Gestion d'évènements
 - Dessin (surcharge de la méthode paint,
rafraîchissement)
 - Hiérarchie de conteneurs

Couleur

- Arrière plan

```
Color getBackground()  
void setBackground(Color)
```

- Couleur d'écriture, de dessin

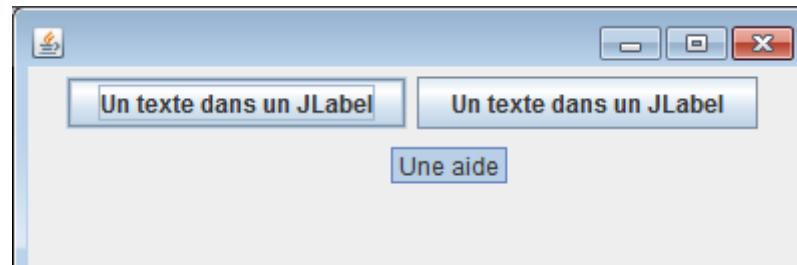
```
Color getForeground()  
void setForeground(Color)
```

- Transparence

```
void setOpaque(boolean)  
boolean isOpaque()
```

Aide - Tooltip

- `public void setToolTipText(String text)`



Bordure - Cadre

- Initialiser/configurer une bordure

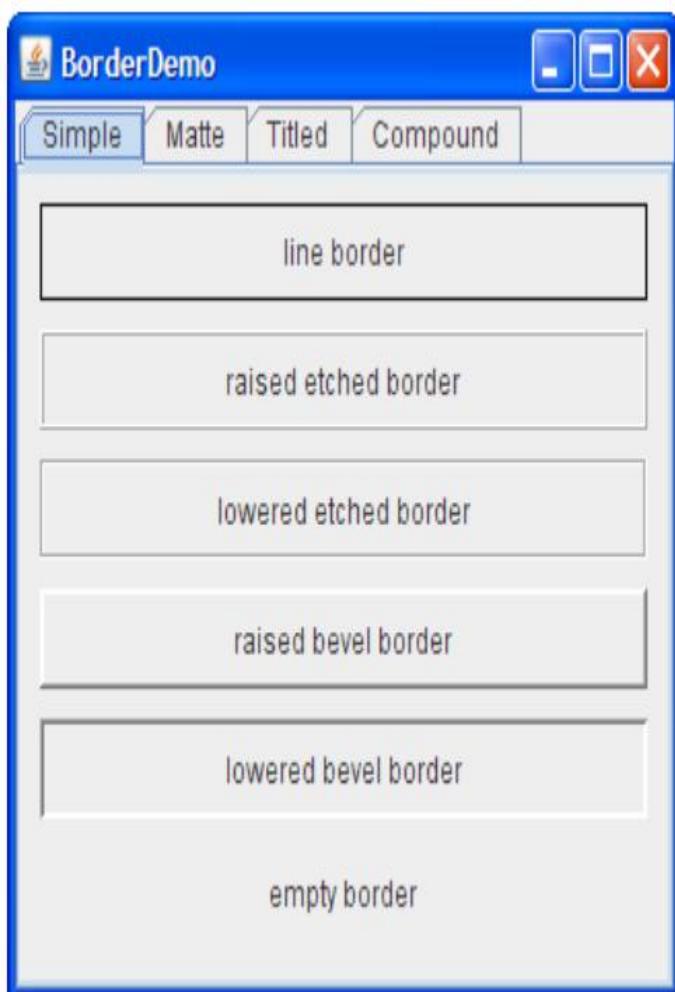
`void setBorder(Border)`

`Border getBorder()`

- `BorderFactory` : que des méthodes statiques

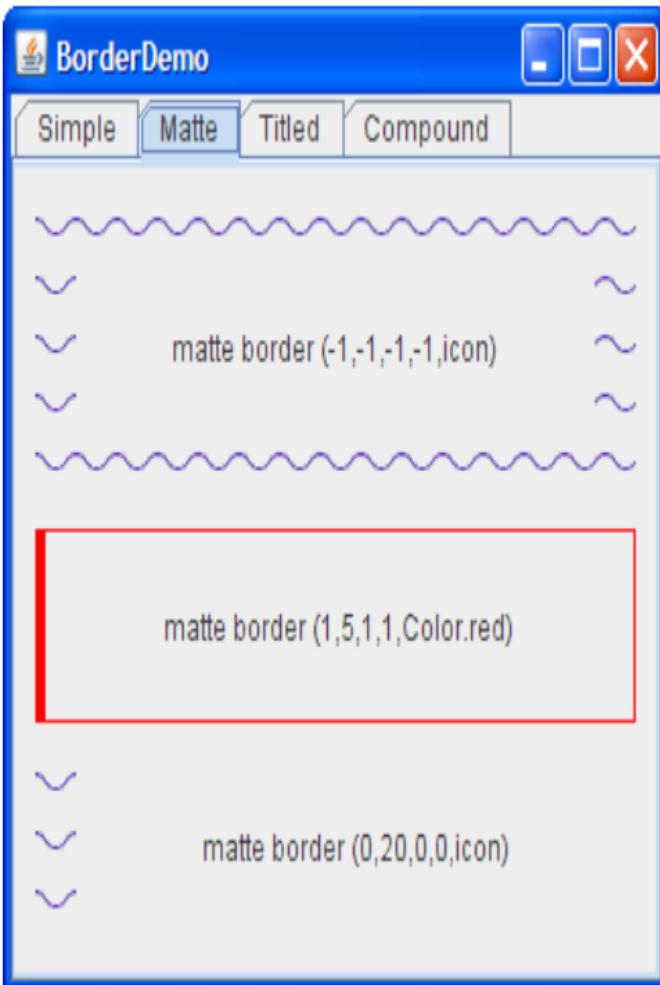
- Simple
- Matte (“Emmêlé”)
- Titled
- Compound

Simple border



- static Border `createLineBorder(Color color, int thickness, boolean rounded)`
 - Couleur,
 - épaisseur,
 - coins arrondis
- static Border `createEtchedBorder(int type, Color highlight, Color shadow)`
 - EtchedBorder.RAISED ou EtchedBorder.LOWERED,
 - Couleur partie illuminée,
 - Couleur partie ombragée
- static Border `createBevelBorder(int type, Color highlightOuter, Color highlightInner, Color shadowOuter, Color shadowInner)`
 - BevelBorder.LOWERED ou BevelBorder.RAISED

Matte border



- Static Border

`createMatteBorder(int top,
int left, int bottom, int right,
Icon tileIcon)`

- Largeur de chaque bord (pixel)
- Icône

Titled border

- **Static Border**

```
createTitledBorder(Border border,  
String title, int titleJustification, int  
titlePosition, Font titleFont, Color  
titleColor)
```

- Justification du titre

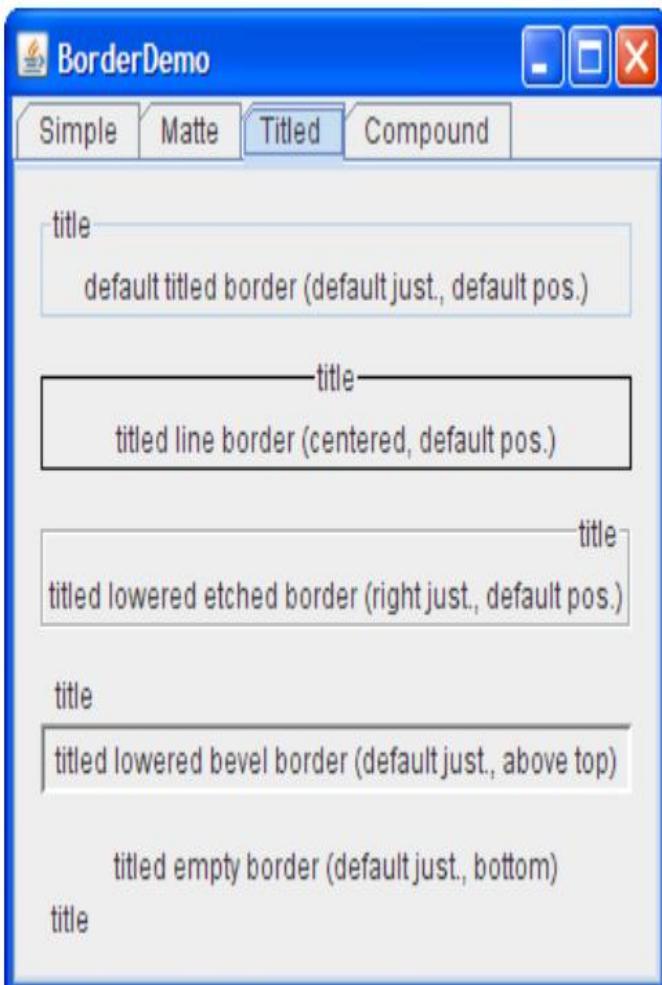
- TitledBorder.LEFT, TitledBorder.CENTER ,
TitledBorder.RIGHT , TitledBorder.LEADING
, TitledBorder.TRAILING ,
TitledBorder.DEFAULT JUSTIFICATION

- Position

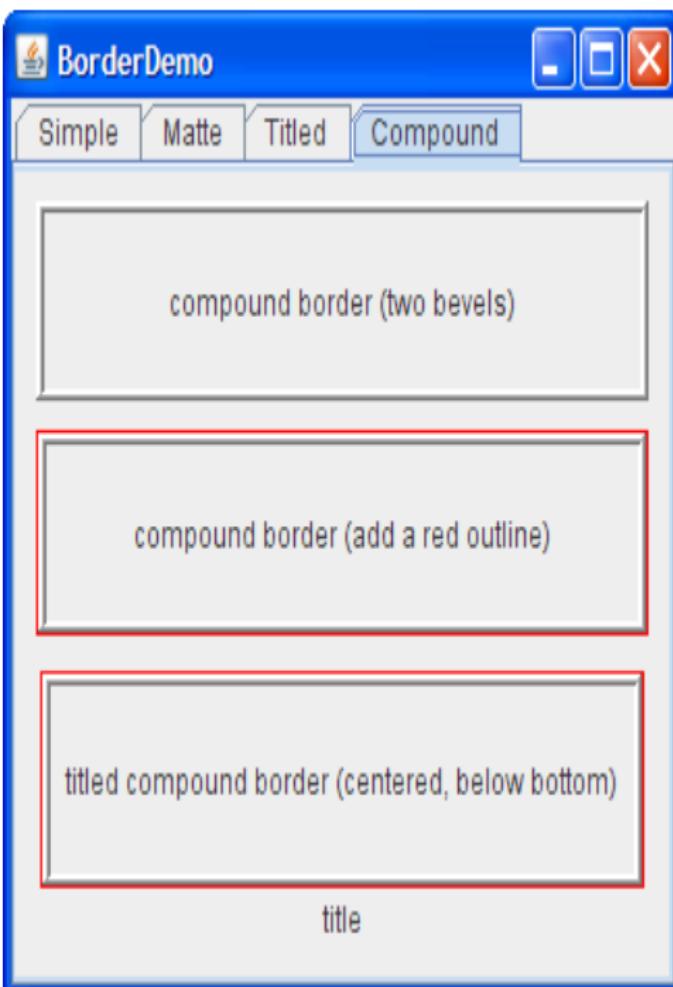
- TitledBorder.ABOVE_TOP ,
– TitledBorder.TOP ,
– TitledBorder.BELOW_TOP ,
– TitledBorder.ABOVE_BOTTOM,
– TitledBorder.BOTTOM ,
– TitledBorder.BELOW_BOTTOM,
– TitledBorder.DEFAULT POSITION

- Police

- Couleur du titre



Compound border



- Static Border
`createCompoundBorder(Border outsideBorder, Border insideBorder)`
 - Combinaison de deux types de bordures

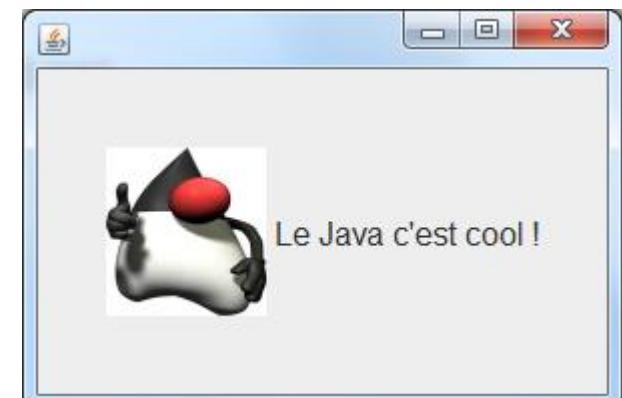
Afficher un peu de texte et/ou une image

- Classe JLabel

```
JLabel(String text, [int horizontalAlignment])  
JLabel(Icon image, [int horizontalAlignment])  
JLabel(String text, Icon icon, int horizontalAlignment)
```

- L'alignement est géré sous forme de constantes

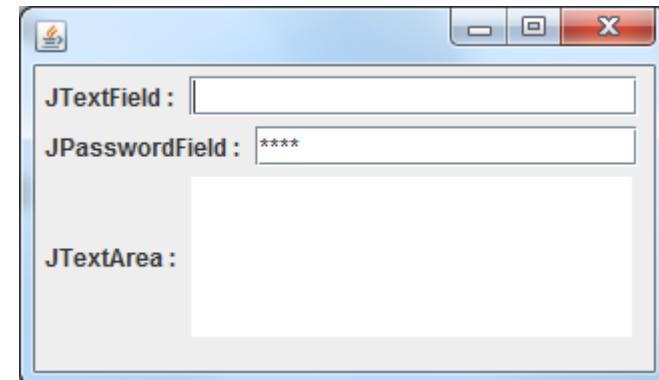
```
SwingConstants.LEFT  
SwingConstants.CENTER  
SwingConstants.RIGHT  
SwingConstants.LEADING  
SwingConstants.TRAILING.
```



Champ de saisie

- Classe principale : **JTextComponent**

```
public void setText(String t)  
public String getText()
```



- Sur une ligne simple : **JTextField** **JTextField(String text, int columns)**
 - En cachant les caractères saisis : **JPasswordField**
JPasswordField(String text, int columns)
void setEchoChar(char c)
- Sur plusieurs lignes : **JTextArea**

```
JTextArea(String text, int rows, int columns)  
void append(String str)  
void insert(String str, int pos)
```

Boutons

- Une classe principale : AbstractButton
 - JButton
 - JToggleButton
 - JRadioButton
 - JCheckBox
 - JMenuItem
 - JMenu
 - JRadioButtonMenuItem
 - JCheckBoxMenuItem

AbstractButton

- Activer / désactiver un bouton

```
public void setEnabled(boolean b)
```

- Ajouter un raccourci clavier

```
void setMnemonic(int mnemonic)
```

- mnemonic correspond à un code de caractère du clavier

```
KeyEvent.VK_V
```

Les boutons

- Les boutons simples : **JButton**

```
JButton(String text)  
JButton(Icon image)  
JButton(String text, Icon icon)
```

- Les boutons à double état : **JToggleButton**

```
JToggleButton(String text, boolean selected)  
JToggleButton(Icon image , boolean selected)  
JToggleButton(String text, Icon icon , boolean selected)
```



Les boutons radio

- Classe : **JRadioButton**

```
JRadioButton(String text, boolean selected)
```

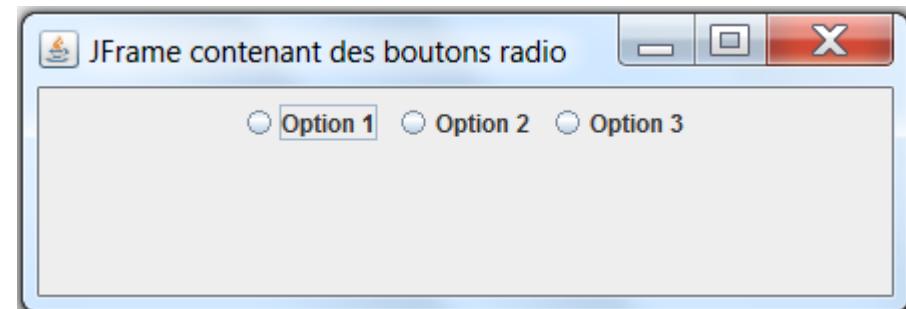
```
JRadioButton(Icon image , boolean selected)
```

```
JRadioButton(String text, Icon icon , boolean selected)
```

- Un seul choix possible à la fois

- Besoin de les regrouper pour désélectionner un choix si on fait un autre choix
- Classe pour les regrouper : **ButtonGroup**

```
public void add(AbstractButton b)
```



Les cases à cocher

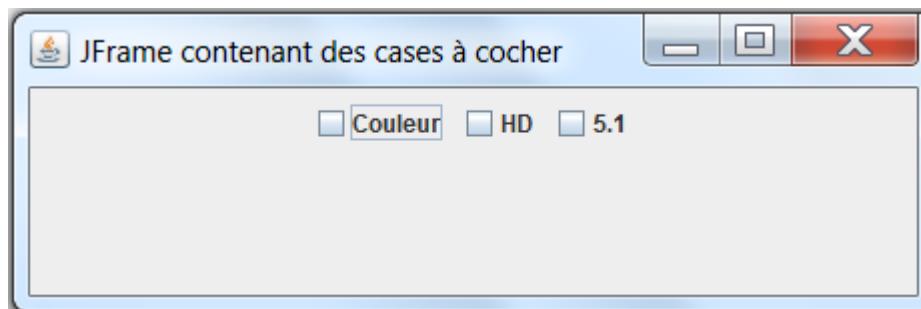
- Classe : **JCheckBox**

```
JCheckBox(String text, boolean selected)
```

```
JCheckBox(Icon image , boolean selected)
```

```
JCheckBox(String text, Icon icon , boolean selected)
```

- Plusieurs cases peuvent être cochées simultanément

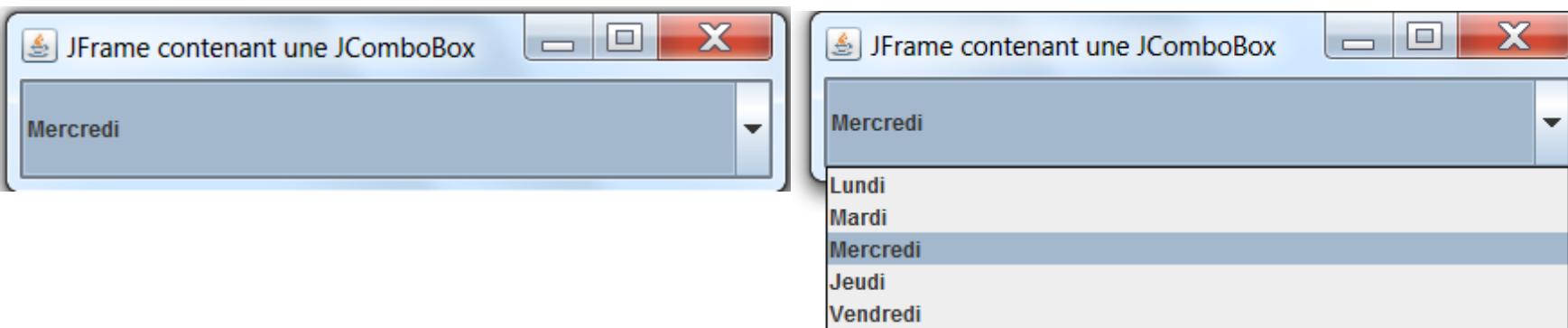


JComboBox

- Liste déroulante pour choisir un item (fermée, ouverte)

```
public JComboBox(E[] items)
```

```
public JComboBox(Vector<E> items)
```

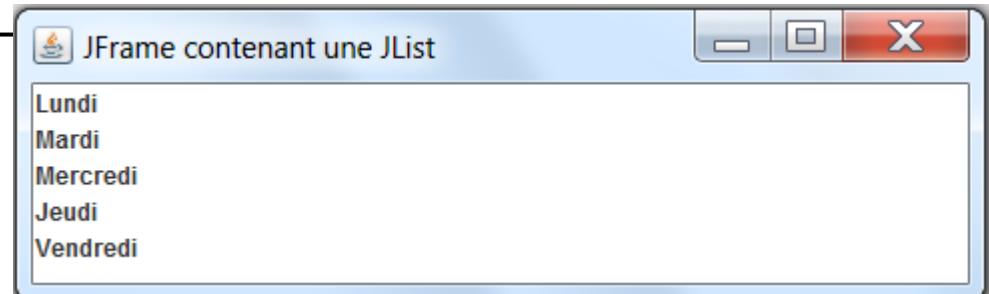


JComboBox - suite

- Pré-selection
 - `public void setSelectedIndex(int anIndex)`
- Nombre d'items à afficher quand déroulée
 - `public void setMaximumRowCount(int count)`
- Récupération de l'élément sélectionné
 - `public Object getSelectedItem()`

JList

- Liste de valeurs
 - `public JList(E[] listData)`
 - `public JList(Vector<? extends E> listData)`
- Pré-selection
 - `public void setSelectedIndex(int anIndex)`
- Doit être ajoutée à un container particulier pour être défilante (JScrollPane)



Gestionnaire de géométrie

Mathieu RAYNAL

mathieu.raynal@irit.fr

<http://www.irit.fr/~Mathieu.Raynal>

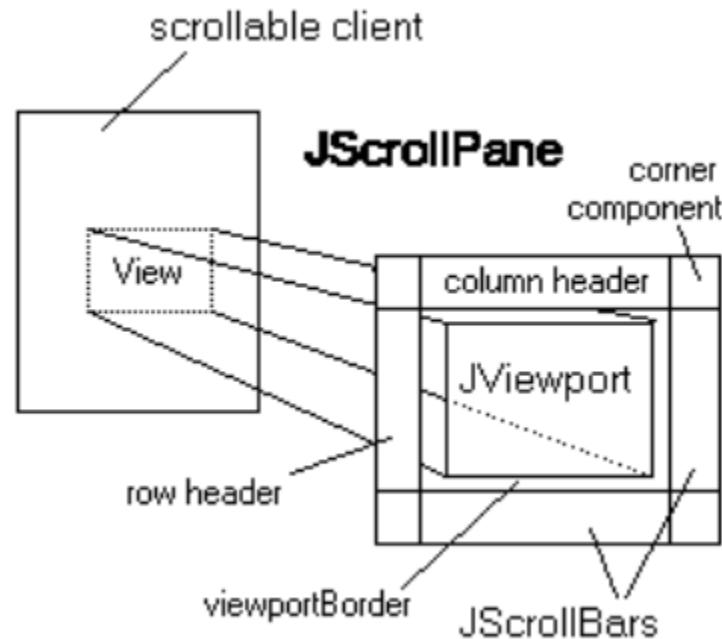
Les composants / containers

- Composants qui ont pour but principal de contenir d'autres composants
- Les principaux conteneurs :
 - JPanel
 - JScrollPane
 - JSplitPane
 - JTabbedPane

- Contenant générique
- N'est pas visuellement détectable
- Permet de structurer une interface en rassemblant un ensemble de composants liés à une activité de l'utilisateur

JScrollPane

- Contenant pour 1 seul composant
- A utiliser pour afficher un composant dont la taille est telle qu'il ne peut être affiché intégralement dans l'emplacement prévu
 - Exemples: texte, liste, tableau, ...



JScrollPane

- Permet d'avoir des barres de défilement lorsque le composant qu'il contient est trop grand par rapport à la taille qui lui est allouée.



```
JLabel texte = new JLabel("un texte trop  
long par rapport à la taille du JLabel");
```

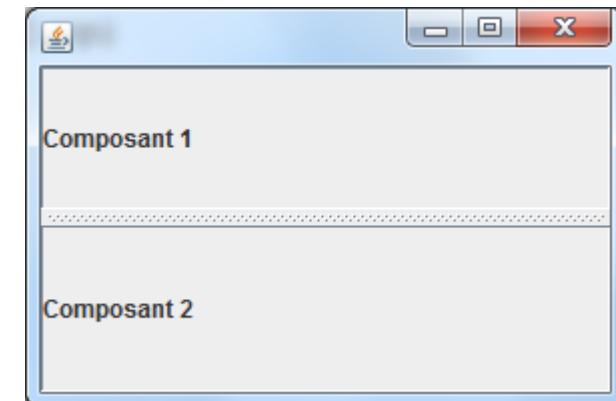
```
JScrollPane scroll = new JScrollPane(texte);
```

JSplitPane

- Permet de séparer un espace en 2 zones
- Ces 2 zones sont séparées par une barre
 - verticale : JSplitPane.VERTICAL_SPLIT
 - ou horizontale : JSplitPane.HORIZONTAL_SPLIT
 - qui peut être déplacée dynamiquement

```
void setDividerLocation(int location)
```

- Par défaut, le composant en haut ou à gauche prend la place dont il a besoin

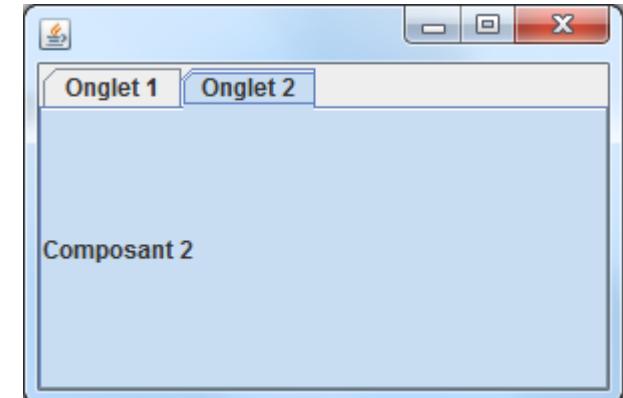


```
JLabel l1 = new JLabel("Composant 1");
JLabel l2 = new JLabel("Composant 2");
JSplitPane sp = new JSplitPane(JSplitPane.VERTICAL_SPLIT, l1, l2);
```

JTabbedPane

- Reprend le système des onglets
- Permet d'avoir plusieurs panneaux sur la même surface

```
JLabel l1 = new JLabel("Composant 1");
JLabel l2 = new JLabel("Composant 2");
JTabbedPane tp = new JTabbedPane();
tp.add("Onglet 1", l1);
tp.add("Onglet 2", l2);
```



Container - Disposition

- Comment positionner les composants les uns par rapport aux autres?
- Comment un container agence-t-il les composants qu'il contient?
 - Gestion du redimensionnement dynamique
- Utilisation de différents types de layout

Layout

Une stratégie par type de layout

- Dispositions simples, peu d'éléments graphiques
 - BorderLayout (5 zones)
 - FlowLayout
 - BoxLayout
- Disposition matricielle
 - GridLayout
- Réalisation de formulaires, dispositions recherchées
 - FormLayout (n'appartient pas au JDK)
 - GridBagLayout
 - **GroupLayout (recommandé avec Netbeans)**

Positionner les composants

- Sur des containers
 - Fenêtre
 - Composants / containers

add(Component comp)

- Au moyen d'un gestionnaire de géométrie :
LayoutManager

setLayout(LayoutManager mgr)

Positionnement sans LayoutManager

- `setLayout(null)`
- Positionnement et taille au pixel près
 - `setLocation(x,y)`
 - `setSize(new Dimension(l,h))`
 - `setBounds(x,y,l,h)`
- Problème : redimensionnement de la fenêtre

JComponent - taille

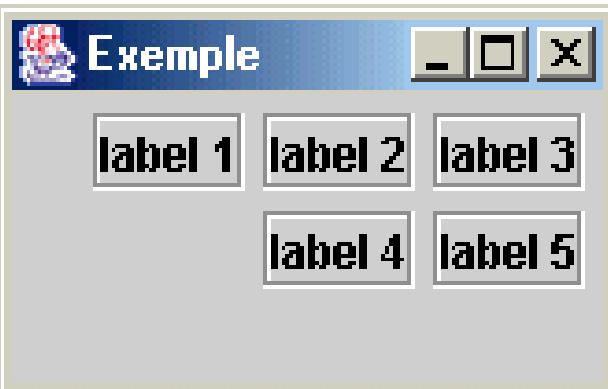
- Taille maximum : maximumSize
- Taille minimum : minimumSize
- Taille effective : size
- Taille idéale : preferredSize

Les gestionnaires de géométrie

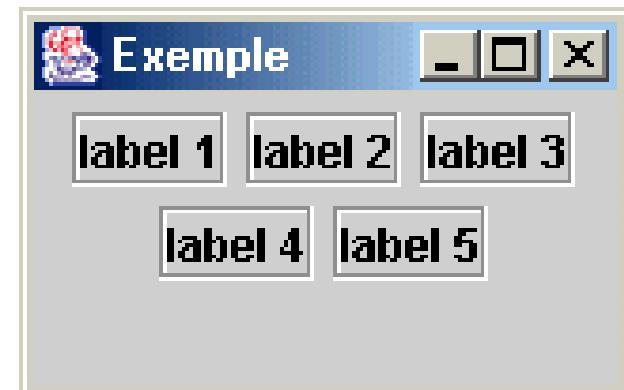
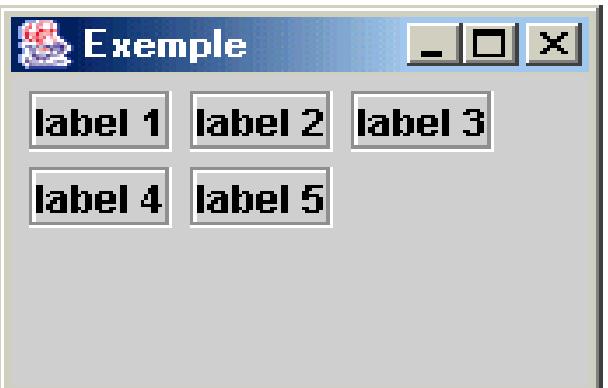
- Gestionnaire de géométrie simple
 - FlowLayout
 - GridLayout
 - BoxLayout
- Gestionnaire de géométrie avec contraintes
 - BorderLayout
 - GridBagLayout
- Possibilité de créer son propre Layout

Principe du FlowLayout

- Arrange les composants de gauche à droite.
- On peut demander à tout aligner à droite, à gauche, au centre ...



```
JFrame f = new JFrame("Exemple");
Container content = f.getContentPane();
content.setLayout (new FlowLayout(FlowLayout.LEFT));
content.add(new JLabel("label 1"));
content.add(new JLabel("label 2"));
content.add(new JLabel("label 3"));
content.add(new JLabel("label 4"));
content.add(new JLabel("label 5"));
```

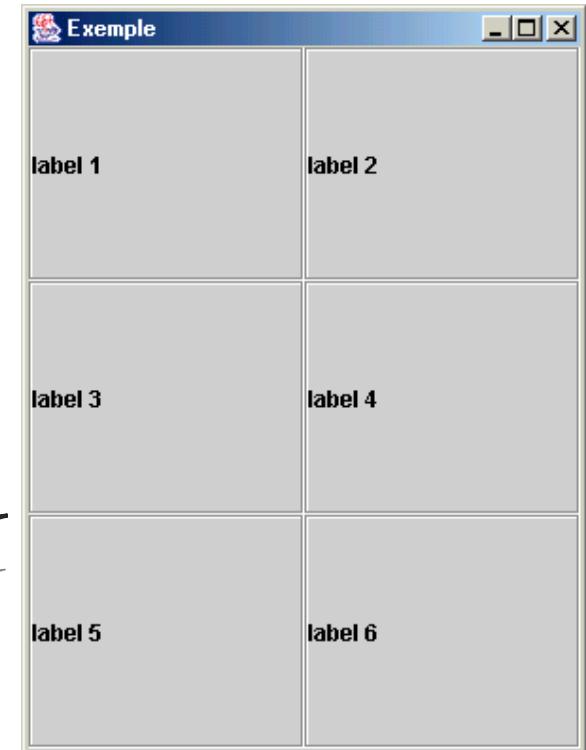


Options du FlowLayout

- Alignement de chaque ligne : align
 - LEFT
 - CENTER (par défaut),
 - RIGHT,
 - LEADING,
 - TRAILING (selon l'orientation).
- Espaces entre composants : Hgap et Vgap
- Paramétrable
 - dans le constructeur : **FlowLayout(int align,int hgap,int vgap)**
 - Via les Getters et Setters associés
- Revalidate() pour prendre en compte les modifications

Principe du GridLayout

- Dispose les éléments sur une grille de n lignes et m colonnes.
- Chaque cellule a la même taille.
- **GridLayout(int lignes, int cols, int hgap, int vgap)**



```
JFrame f = new JFrame("Exemple");
Container content = f.getContentPane();
content.setLayout (new GridLayout(3,2));
content.add(new JLabel("label 1"));
content.add(new JLabel("label 2"));
content.add(new JLabel("label 3"));
content.add(new JLabel("label 4"));
content.add(new JLabel("label 5"));
content.add(new JLabel("label 6"));
```

Principe du BoxLayout

- Arrange les composants
 - sur une ligne (X_AXIS)
 - sur une colonne (Y_AXIS)
 - En prenant la taille préférée des composants
- **BoxLayout(Container c, int axe)**
 - Le container doit être créé avant le BoxLayout

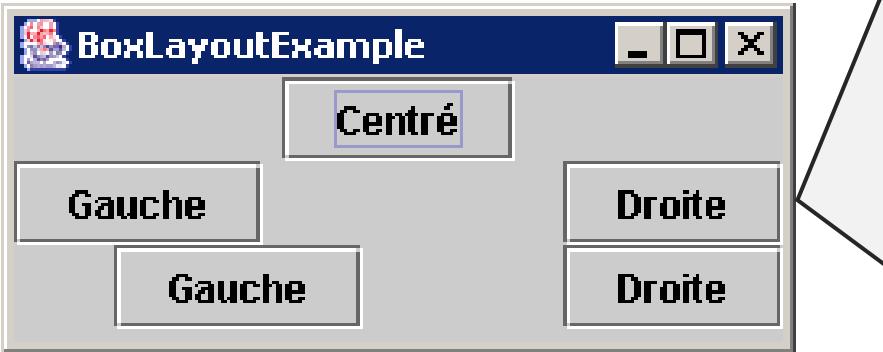
Le composant Box

- Composant transparent avec un BoxLayout associé
- Il peut contenir des glues pour « remplir » des espaces entre composants
 - Un composant entre deux glues sera centré
 - Une glue entre deux composants plaquera les composants sur les bords

static Component createHorizontalGlue()

static Component createVerticalGlue()

Exemple



```
Container c = f.getContentPane();
```

```
Box line1=new Box(BoxLayout.X_AXIS);
line1.add(Box.createHorizontalGlue());
line1.add(new JButton("Centré"));
line1.add(Box.createHorizontalGlue());
```

```
Box line2=new Box(BoxLayout.X_AXIS);
line2.add(new JButton("Gauche"));
line2.add(Box.createHorizontalGlue());
line2.add(new JButton("Droite"));
```

```
Box line3=new Box(BoxLayout.X_AXIS);
line3.add(Box.createHorizontalGlue());
line3.add(new JButton("Gauche"));
line3.add(Box.createHorizontalGlue());
line3.add(Box.createHorizontalGlue());
line3.add(new JButton("Droite"));
```

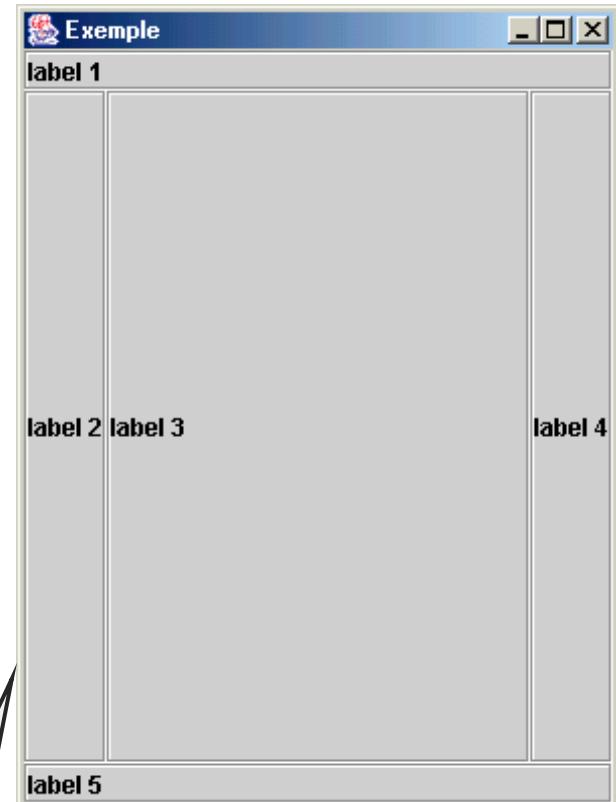
```
c.setLayout(new BoxLayout(c,BoxLayout.Y_AXIS));
c.add(line1);
c.add(line2);
c.add(line3);
```

Principe du BorderLayout

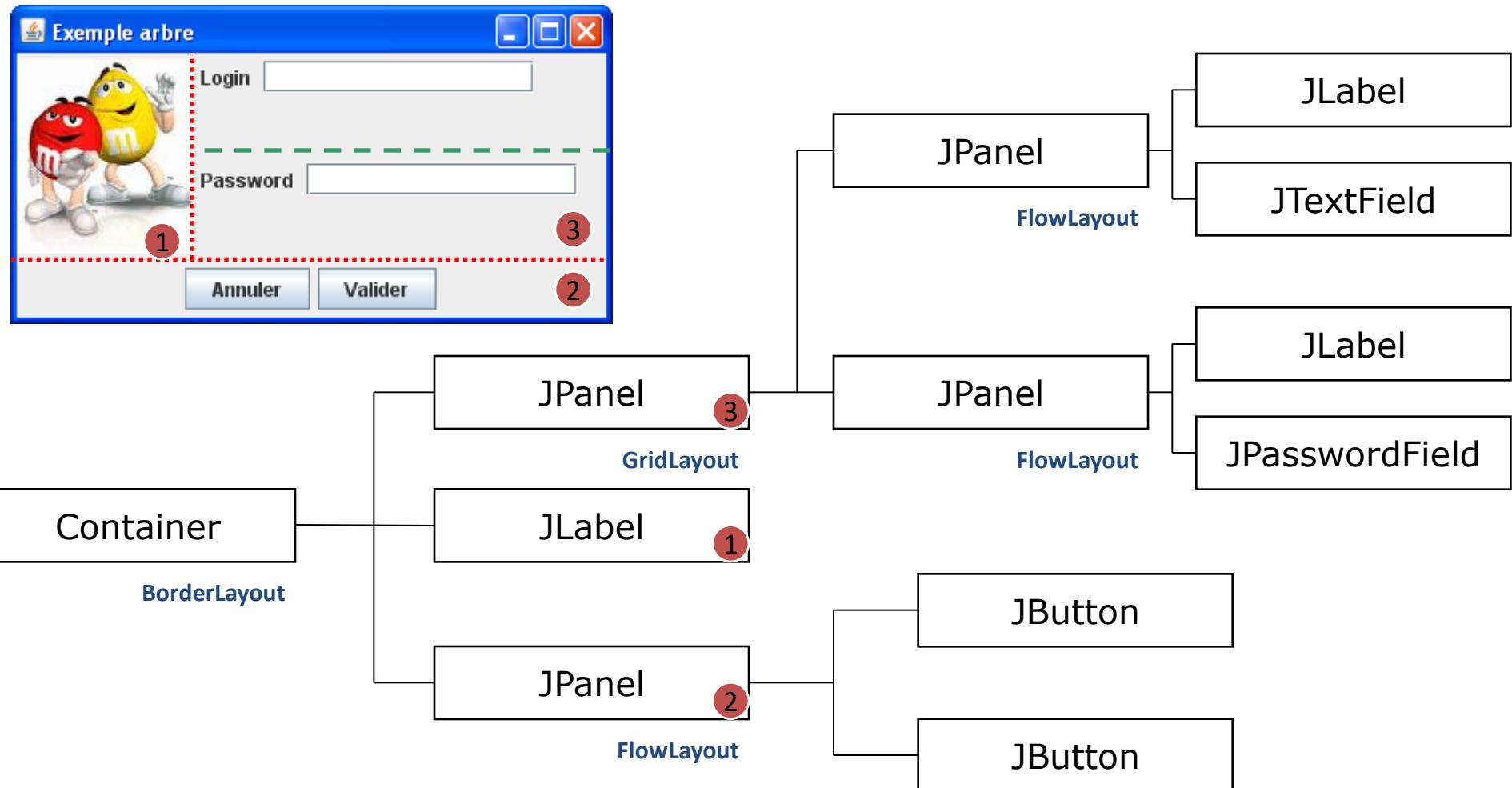
- Peut contenir jusqu'à 5 éléments

- haut,
- bas,
- droite,
- gauche
 - juste la place dont ils ont besoin
- un élément central
 - qui prend toute la place qui reste.

```
JFrame f = new JFrame("Exemple");
Container content = f.getContentPane();
content.setLayout(new BorderLayout());
content.add(new JLabel("label 1"),BorderLayout.NORTH);
content.add(new JLabel("label 2"),BorderLayout.WEST);
content.add(new JLabel("label 3"),BorderLayout.CENTER);
content.add(new JLabel("label 4"),BorderLayout.EAST);
content.add(new JLabel("label 5"),BorderLayout.SOUTH);
```



Exemple d'arbre de composants



La barre de menu

- Espace spécifique pour le menu
 - Pas géré sur le contentPane
- Une classe spécifique : JMenuBar
 - Qui est liée à la JFrame
 - JMenuBar getJMenuBar()
 - setJMenuBar(JMenuBar)
 - Sur laquelle on ajoute les menus
 - add(JMenu)

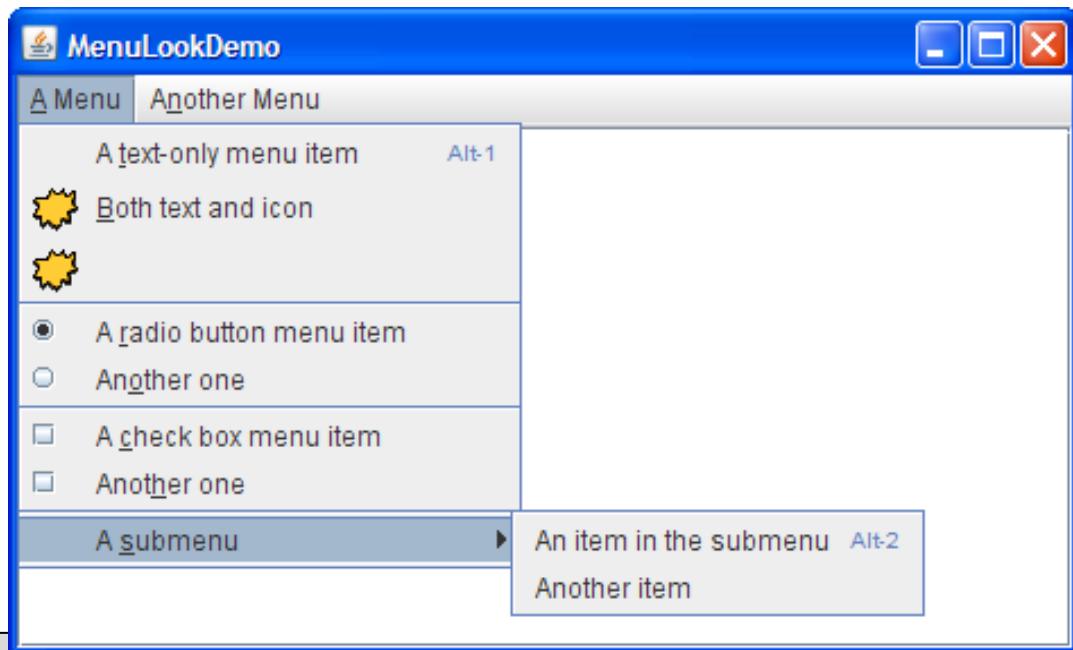
Créer un item de menu

- Un item simple : JMenuItem
 - Même fonctionnement qu'un bouton classique (hérite de AbstractButton)
 - Affiche un texte et/ou une icône
 - JMenuItem(Icon icon)
 - JMenuItem(String text)
 - JMenuItem(String text, Icon icon)

—

Les autres types d'items

- `JRadioButtonMenuItem`
- `JCheckboxMenuItem`
- Fonctionne de la même manière que `JRadioButton` et `JCheckbox`

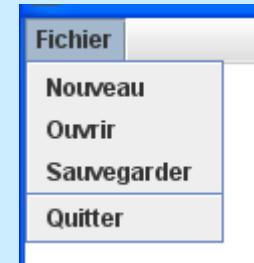


Créer un menu ou sous menu

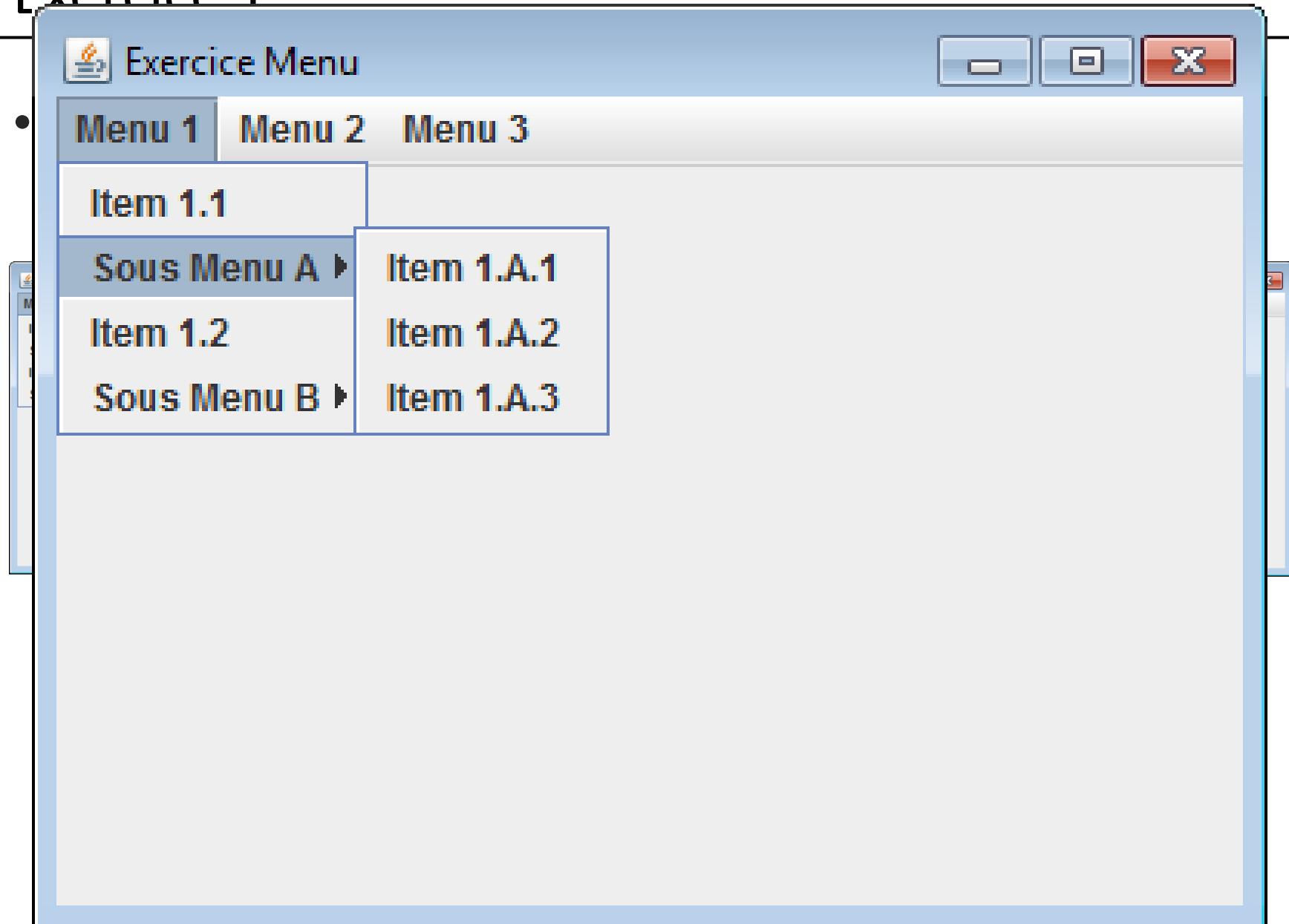
- La barre de menu (JMenuBar)
 - Contient des Menus (JMenu) qui contiennent
 - Des items (JMenuItem, JRadioButtonMenuItem, JCheckboxMenuItem)
 - Des sous menus (JMenu)
 - Des séparateurs (Jseparator)
- Positionne les menus à l'aide d'un BoxLayout
 - Possibilité d'utiliser les glues

Création d'un menu (exemple)

```
JMenuBar barreMenu = new JMenuBar();  
  
JMenu menuFichier = new JMenu("Fichier");  
  
 JMenuItem itemNouveau = new JMenuItem("Nouveau");  
 JMenuItem itemOuvrir = new JMenuItem("Ouvrir");  
 JMenuItem itemSauvegarder = new JMenuItem("Sauvegarder");  
 JMenuItem itemQuitter = new JMenuItem("Quitter");  
  
menuFichier.add(itemNouveau);  
menuFichier.add(itemOuvrir);  
menuFichier.add(itemSauvegarder);  
menuFichier.add(new JSeparator());  
menuFichier.add(itemQuitter);  
  
barreMenu.add(menuFichier);  
  
setJMenuBar(barreMenu);
```



Exercice 1



Gestion des tableaux et arbres

Mathieu RAYNAL

mathieu.raynal@irit.fr

<http://www.irit.fr/~Mathieu.Raynal>

Les tableaux

- Un objet graphique : JTable
 - Permet d'afficher un tableau
 - Permet l'édition de son contenu

The Header contains Column labels

First Name	Last Name	Sport	# of Years	Vegetarian
Kathy	Smith	Snowboarding	5	
John	Doe	Rowing	3	<input checked="" type="checkbox"/>
Sue	Black	Knitting	2	<input type="checkbox"/>
Jane	White	Speed reading	20	<input checked="" type="checkbox"/>
			40	<input type="checkbox"/>

Each Cell displays a data item

Each Column displays one type of data

- Présenter une vision structurée d'un grand nombre de données et permettre leur manipulation
 - 2 dimensions
 - Synthèse
 - Classification

Un composant graphique : la JTable

- Constructeurs

```
JTable(Object[][] rowData, Object[] columnNames)
```

```
JTable(Vector rowData, Vector columnNames)
```

- Principales méthodes

```
int[] getSelectedRows()
```

```
Object getValueAt(int row, int column)
```

```
void setValueAt(Object aValue, int row, int column)
```

Gestion d'une JTable

- Données à afficher dans le tableau
 - Quelles données affichées ? Dans quelle cellule ?
 - Interface : TableModel
 - Classe abstraite : AbstractTableModel
 - Classe par défaut : DefaultTableModel
- Rendu visuel des cellules du tableau
 - Comment afficher les données présentes dans la table ?
 - Interface : TableCellRenderer
 - Classe par défaut : DefaultTableCellRenderer
- Edition des données contenues dans le tableau
 - Comment éditer les données présentes dans la table ?
 - Interface : TableCellEditor

Modèle de données

- **Objectif :** Indiquer à la Jtable comment afficher un ensemble de données
- Par défaut, JTable contient une instance de DefaultTableModel
 - Créé par défaut à la création de la Jtable
 - Les données sont stockées sous forme de Vector
 - Objet qui implémente TableModel
- Possibilité de modifier le modèle
 - Personnalisation du modèle grâce à AbstractTableModel
 - Passage au constructeur de la JTable

JTable(TableModel dm)

Personalisation du modèle

- Définir une classe héritant de `AbstractTableModel`
- Exige de définir les méthodes d'accès aux données
 - `public int getRowCount()`
 - `public int getColumnCount()`
 - `public Object getValueAt(int row, int column)`
- Masquage possible d'autres méthodes
 - `boolean isCellEditable(int rowIndex, int colIndex)`
 - `String getColumnName(int colIndex)`
 - `Class getColumnClass(int colIndex)`

Surcharges intéressantes (2/2)

- Modifier une valeur dans le modèle

```
void setValueAt(Object val, int rowIndex, int collIndex)
```

- **Nécessite de « prévenir » des changements dans le modèle**

- **fireTableCellUpdated(int row, int column)** : une cellule
- **fireTableRowsUpdated(int firstRow, int lastRow)** : une ligne
- **fireTableDataChanged()** : la table entière
- **fireTableStructureChanged()** : structure de la table
- **fireTableRowsInserted(int firstRow, int lastRow)** : insertion de ligne
- **fireTableRowsDeleted(int firstRow, int lastRow)**: suppression de ligne

Modifier une valeur de la table

- `void setValueAt(Object val, int rowIndex, int colIndex)`
- Nécessite de « prévenir » des changements dans le modèle
 - `fireTableCellUpdated(int row, int column)`
 - `fireTableRowsUpdated(int firstRow, int lastRow)`
 - `fireTableDataChanged()`
 - `fireTableStructureChanged()`
 - `fireTableRowsInserted(int firstRow, int lastRow)`
 - `fireTableRowsDeleted(int firstRow, int lastRow)`

Exercice 1

Exemple tableau

Nom	Prenom	Age	Club	Poste
AHAMADA	Ali	22	TFC	GARDIEN
AKPA AKPRO	Jean Daniel	21	TFC	DEFENSEUR
BALMONT	Florent	34	LOSC	MILIEU
BAYSSE	Paul	26	ASSE	DEFENSEUR
BEN YEDDER	Wissam	23	TFC	ATTAQUANT
BERIA	Franck	30	LOSC	DEFENSEUR
BRAITHWAITE	Martin	22	TFC	ATTAQUANT
CAVANI	Edinson	27	PSG	ATTAQUANT
CHANTÔME	Clément	26	TFC	MILIEU
CLEMENT	Jérémy	29	ASSE	MILIEU
CLERC	François	30	ASSE	DEFENSEUR
DIDOT	Etienne	30	TFC	MILIEU
DIGNE	Lucas	20	PSG	DEFENSEUR
ELANA	Steeve	33	LOSC	GARDIEN
ERDING	Mevlut	27	ASSE	ATTAQUANT

- A partir de la structure de données ci-jointe, créez le modèle permettant d'afficher les données comme sur ce tableau

```
public class Footballeur
{
    public String nom, prenom;
    public int age;
    public Club club;
    public Poste poste;

    ...
}
```

```
public class ListeJoueur
{
    TreeSet<Joueur> listeJoueurs;

    public ListeJoueur()
    {
        listeJoueurs = new TreeSet<Joueur>(new JoueurComparatorAlpha());
        ...
    }
}
```

Correction (1/2)

```
public class ModeleTable extends AbstractTableModel
{
    ListeJoueur liste;

    public ModeleTable()
    {
        liste = new ListeJoueur();
    }

    public int getColumnCount()
    {
        return 5;
    }

    public int getRowCount()
    {
        return liste.listeJoueurs.size();
    }
}
```

Correction (2/2)

```
public String getColumnName(int idC)
{
    switch (idC)
    {
        case 0: return new String("Nom");
        case 1: return new String("Prenom");
        case 2: return new String("Age");
        case 3: return new String("Club");
        case 4: return new String("Poste");
        default: return null;
    }
}
```

```
public Object getValueAt(int idL, int idC)
{
    Object[] tab = liste.listeJoueurs.toArray();
    Joueur p = (Joueur)tab[idL];
    switch (idC)
    {
        case 0: return p.nom;
        case 1: return p.prenom;
        case 2: return p.age;
        case 3: return p.club;
        case 4: return p.poste;
        default: return null;
    }
}
```

```
JTable table = new JTable(new ModeleTable());
```

Rendu des cellules

- Comment afficher les données dans une cellule ?
 - En fonction du type d'Objet
- Des rendus par défaut
 - Boolean : case à cocher
 - Number : label aligné à droite
 - Object : chaîne de caractères correspondante à l'objet
- Si modèle de données personnalisé, pensez à définir le type d'objet présent dans chaque cellule

```
public Class getColumnClass(int c) {  
    return getValueAt(0, c).getClass();  
}
```

Personnalisation du rendu des cellules

- Implémenter l'interface TableCellRenderer
- ou étendre de la classe DefaultTableCellRenderer

```
public Component getTableCellRendererComponent(  
    JTable table,  
    Object value,  
    boolean isSelected,  
    int row,  
    int column)
```

Personnalisation du rendu des cellules

- Associer le nouveau rendu aux cellules
 - Des colonnes souhaitées

```
public TableColumn getColumn(Object identifier)
```

Dans TableColumn:

```
void setCellRenderer(TableCellRenderer cellRenderer)
```

- Des cellules ayant ce type d'objet

```
public void setDefaultRenderer(Class<?> columnClass,  
                             TableCellRenderer renderer)
```

Exercice 2

- Ecrire la classe permettant d'avoir le même rendu que sur la tableau :
 - Les joueurs du TFC en violet
 - Les joueurs du LOSC en rouge
 - Les joueurs du PSG en bleu
 - Les joueurs de l'ASSE en vert

Exemple tableau

Nom	Prenom	Age	Club	Poste
AHAMADA	Ali	22	TFC	GARDIEN
AKPA AKP...	Jean Daniel	21	TFC	DEFENSEUR
BALMONT	Florent	34	LOSC	MILIEU
BAYSSE	Paul	26	ASSE	DEFENSEUR
BEN YEDDER	Wissam	23	TFC	ATTAQUANT
BERIA	Franck	30	LOSC	DEFENSEUR
BRAITHWA...	Martin	22	TFC	ATTAQUANT
CAVANI	Edinson	27	PSG	ATTAQUANT
CHANTOME	Clément	26	TFC	MILIEU
CLEMENT	Jérémy	29	ASSE	MILIEU
CLERC	François	30	ASSE	DEFENSEUR
DIDOT	Etienne	30	TFC	MILIEU
DIGNE	Lucas	20	PSG	DEFENSEUR
ELANA	Steeve	33	LOSC	GARDIEN
ERDING	Mevlut	27	ASSE	ATTAQUANT

Correction : la classe de rendu ...

```
public class RenduJoueur extends DefaultTableCellRenderer
{
    public Component getTableCellRendererComponent(JTable table, Object value,
        boolean isSelected, boolean hasFocus, int row, int column)
    {
        setFont(new Font(Font.SERIF,Font.BOLD,14));
        if(((Club)table.getValueAt(row,3))==Club.LOSC)
            setBackground(new Color(250,80,80));
        if(((Club)table.getValueAt(row,3))==Club.PSG)
            setBackground(new Color(80,80,250));
        if(((Club)table.getValueAt(row,3))==Club.ASSE)
            setBackground(new Color(80,250,80));
        if(((Club)table.getValueAt(row,3))==Club.TFC)
            setBackground(new Color(250,80,250));
        setText(value.toString());
        return this;
    }
}
```

Correction : ... mais aussi

- Associer le rendu à la table
 - Pour une colonne donnée

```
TableColumn tc = table.getColumn("Age");  
tc.setCellRenderer(new RenduJoueur());
```

- Pour un type d'objet

```
table.setDefaultRenderer(Object.class, new RenduJoueur());
```

- Penser à spécifier le type d'objet dans le modèle

```
public Class getColumnClass(int col)  
{  
    switch (col)  
    {  
        case 0: return String.class;  
        case 1: return String.class;
```

```
        case 2: return Integer.class;  
        case 3: return Club.class;  
        case 4: return Poste.class;  
        default: return Object.class;
```

```
}
```

```
}
```

Editor - Editeur

- Différentes manières de saisir une donnée
 - Champ texte
 - Case à cocher
 - Liste déroulante
 - Color picker
 - ...

Editeur par défaut

- DefaultCellEditor
 - JTextField, JCheckBox, JComboBox

The screenshot shows a Java Swing application window titled "TableRenderDemo". The window contains a table with five columns: "First Name", "Last Name", "Sport", "# of Years", and "Vegetarian". The "Sport" column uses a JComboBox editor. A dropdown menu is open over the cell for "Kathy Smith", listing the following options: "Snowboarding", "Snowboarding", "Rowing", "Knitting", "Speed reading", "Pool", and "None of the above". The option "Snowboarding" is highlighted. The "# of Years" column contains the values 5, 3, 2, 20, and 10. The "Vegetarian" column contains five checkboxes, three of which are checked.

First Name	Last Name	Sport	# of Years	Vegetarian
Kathy	Smith	Snowboarding	5	<input type="checkbox"/>
John	Doe	Snowboarding	3	<input checked="" type="checkbox"/>
Sue	Black	Rowing	2	<input type="checkbox"/>
Jane	White	Knitting	20	<input checked="" type="checkbox"/>
Joe	Brown	Speed reading	10	<input type="checkbox"/>
		Pool		
		None of the above		

Edition par liste déroulante - exemple

```
 TableColumn sportColumn =  
 table.getColumnModel().getColumn(2);
```

...

```
JComboBox comboBox = new JComboBox();  
comboBox.addItem("Snowboarding");  
comboBox.addItem("Rowing");  
comboBox.addItem("Chasing toddlers");  
comboBox.addItem("Speed reading");  
comboBox.addItem("Teaching high school");  
comboBox.addItem("None");  
sportColumn.setCellEditor(new DefaultCellEditor(comboBox));
```

Personnalisation de l'édition des cellules

- Définir une classe qui
 - met en oeuvre l'interface **TableCellEditor**
 - `public Component getTableCellEditorComponent(JTable table, Object value, boolean isSelected, int row, int column)`
 - et étend la classe **AbstractCellEditor**
 - `public Object getCellEditorValue()`
- Configurer l'éditeur
 - `JTable: void setDefaultEditor(Class<?> columnClass, TableCellEditor editor)`
 - `TableColumn: void setCellEditor(TableCellEditor cellEditor)`

Exercice 3

- Ecrire le code qui permette à l'utilisateur de changer le club d'un joueur

Exemple tableau

Nom	Prenom	Age	Club	Poste
AHAMADA	Ali	22	TFC	GARDIEN
AKPA AKPRO	Jean Daniel	21	TFC	DEFENSEUR
BALMONT	Florent	34	LOSC	MILIEU
BAYSSE	Paul	26	ASSE	DEFENSEUR
BEN YEDDER	Wissam	23	TFC	ATTAQUANT
BERIA	Franck	30	PSG	DEFENSEUR
BRAITHWAITE	Martin	22	TFC	ATTAQUANT
CAVANI	Edinson	27	LOSC	ATTAQUANT
CHANTOME	Clément	26	ASSE	MILIEU
CLEMENT	Jérémy	29	ASSE	MILIEU
CLERC	François	30	ASSE	DEFENSEUR
DIDOT	Etienne	30	TFC	MILIEU
DIGNE	Lucas	20	PSG	DEFENSEUR
ELANA	Steeve	33	LOSC	GARDIEN
ERDING	Mevlut	27	ASSE	ATTAQUANT
IBRAHIMOVIC	Zlatan	32	PSG	ATTAQUANT
KALOU	Salomon	28	LOSC	ATTAQUANT
LEMOINE	Fabien	27	ASSE	MILIEU
MARTIN	Marvin	26	LOSC	MILIEU

Correction

```
public void setValueAt(Object o, int idL, int idC)
{
    Object[] tab = liste.listeJoueurs.toArray();
    Joueur p = (Joueur)tab[idL];
    switch (idC)
    {
        case 0: p.nom = (String)o; break;
        case 1: p.prenom = (String)o; break;
        case 2: p.age = Integer.parseInt((String)o); break;
        case 3: p.club = (Club)o; break;
        case 4: p.poste = (Poste)o; break;
        default: break;
    }
    fireTableRowsUpdated(idL, idC);
}
```

```
JComboBox comboClub = new JComboBox(Club.values());
tc = table.getColumn("Club");
tc.setCellEditor(new DefaultCellEditor(comboClub));
```

Récap. model/rendu/edition

	Classe utilisée par défaut	Personnalisation
Modèle de données	Classe concrète DefaultTableModel	Extension de la classe abstraite AbstractTableModel
Rendu	Classe concrète DefaultCellRenderer	Extension d'un élément graphique (héritant de Jcomponent) + mise en oeuvre de l'interface TableCellRenderer
Edition	Classe concrète DefaultCellEditor	Extension de la classe abstraite AbstractCellEditor + mise en oeuvre de l'interface TableCellEditor

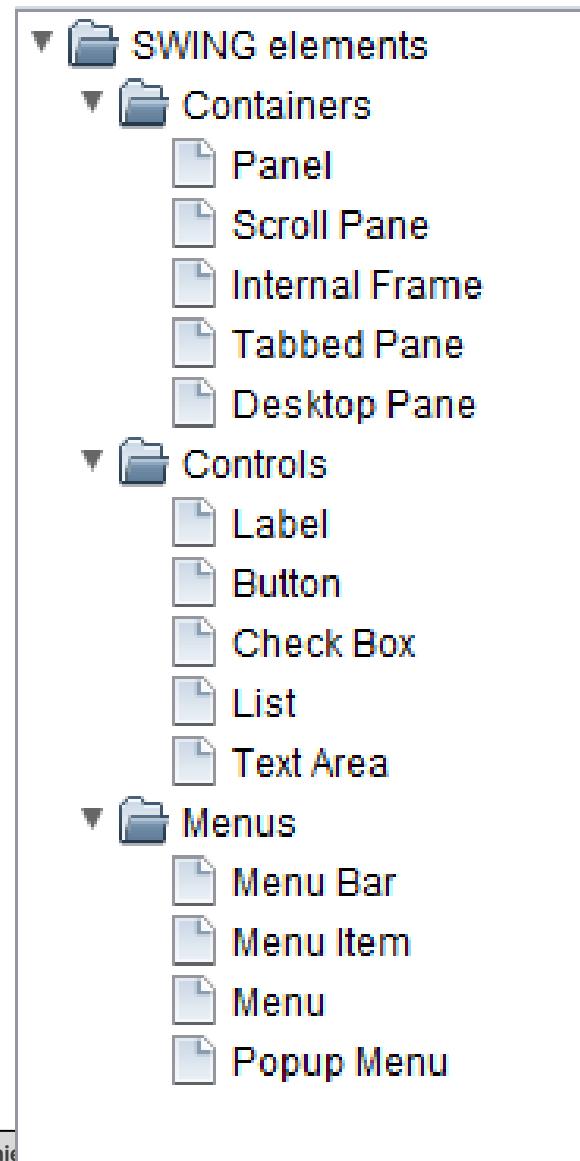
Les arbres

Arbre: utilité et composition

- Visualisation d'informations
- Hiérarchie
- Composition d'un arbre
 - Racine
 - Branches
 - Feuilles

SWING - JTree

- Racine - Root
- Noeud - Node
 - Branche – Branch node
 - Feuille – Leaf node



Création simple d'un JTree

- Arbre – Jtree
 - public JTree(TreeNode root)
- Noeuds/Données – DefaultMutableTreeNode
 - public DefaultMutableTreeNode(Object userObject)
- Ajout d'un sous-noeud
 - public void add(MutableTreeNode newChild)

Création simple d'un JTree - exemple

```
DefaultMutableTreeNode treeNode1;
```

```
DefaultMutableTreeNode treeNode2;
```

```
DefaultMutableTreeNode treeNode3;
```

```
treeNode1 = new DefaultMutableTreeNode("JTree");
```

```
treeNode2 = new DefaultMutableTreeNode("Branch 1");
```

```
treeNode3 = new DefaultMutableTreeNode("leaf 1");
```

```
//Branche "Branch 1"
```

```
treeNode2.add(treeNode3);
```

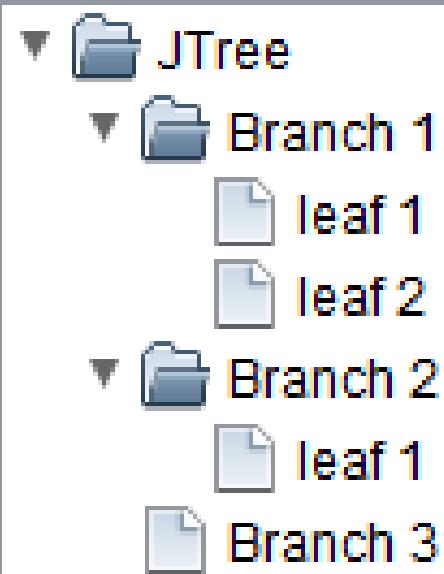
```
treeNode3 = new DefaultMutableTreeNode("leaf 2");
```

```
treeNode2.add(treeNode3);
```

```
treeNode1.add(treeNode2);
```

```
...
```

```
JTree jtree = new JTree(treeNode1);
```



Personnalisation du rendu (simple)

- Affichage de l'élément racine

```
public void setRootVisible(boolean rootVisible)
```

- Affichage des “poignées de dépliage/repliage”

```
public void setShowsRootHandles(boolean newValue)
```

Création d'un JTree associé à un modèle

- Constructeur JTree

JTree(TreeModel newModel)

- DefaultTreeModel

— associé par défaut à une instance de classe JTree

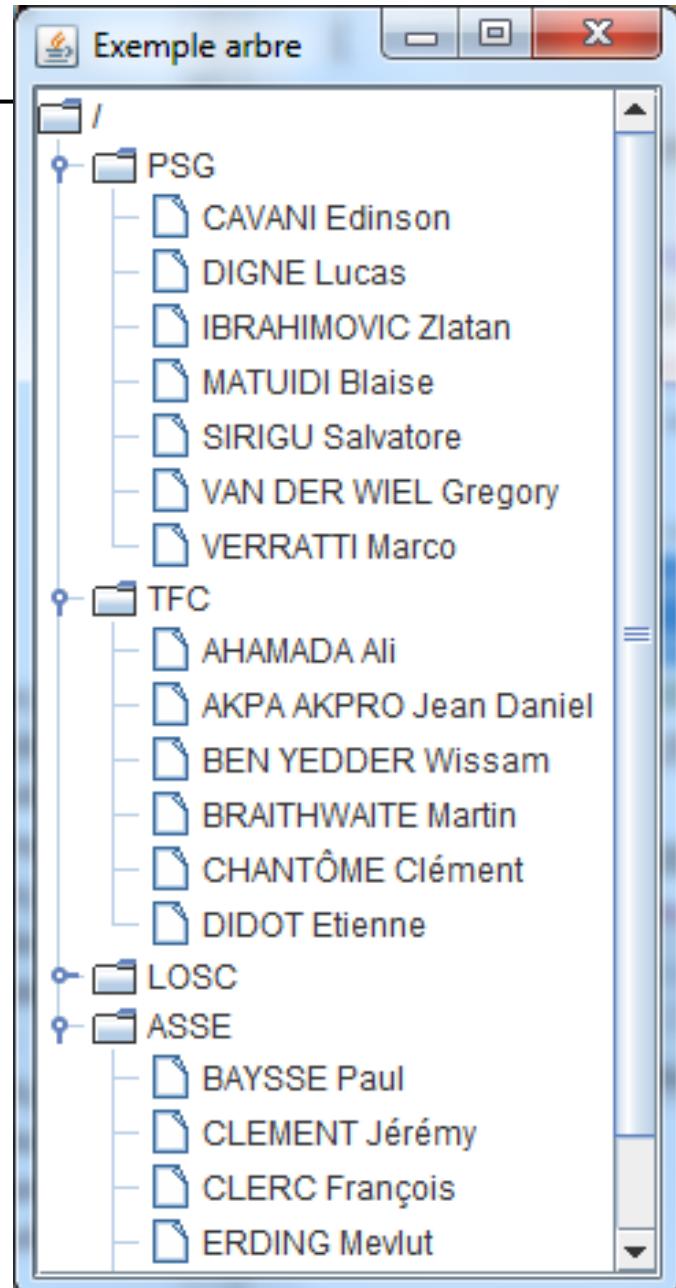
DefaultTreeModel(TreeNode root)

Création d'un JTree associé à un modèle

- Implémentation de l'interface TreeModel
 - void **addTreeModelListener(TreeModelListener l)**
 - Object **getChild(Object parent, int index)**
 - int **getChildCount(Object parent)**
 - int **getIndexOfChild(Object parent, Object child)**
 - Object **getRoot()**
 - boolean **isLeaf(Object node)**
 - void **removeTreeModelListener(TreeModelListener l)**
 - void **valueForPathChanged(TreePath path, Object newValue)**

Exercice 4

- Toujours à partir de la même structure de données, réalisez le modèle qui permet d'afficher l'arbre ci-contre



Correction

```
public class ModeleArbre implements TreeModel
{
    ListeJoueur listeJoueur;

    public Object getRoot()
    {
        return "/";
    }

    public Object getChild(Object arg0, int arg1)
    {
        if(arg0 instanceof String && ((String)arg0).equals("/"))
            return Club.values()[arg1];
        if(arg0 instanceof Club)
            return listeJoueur.getJoueur((Club)arg0, arg1);
        return null;
    }

    public int getChildCount(Object arg0)
    {
        if(arg0 instanceof String && ((String)arg0).equals("/"))
            return Club.values().length;
        if(arg0 instanceof Club)
            return listeJoueur.NombreJoueur((Club)arg0);
        return 0;
    }
}
```

Personnalisation du rendu

- Reconfiguration du renderer associé par défaut
 - DefaultTreeCellRenderer : public
DefaultTreeCellRenderer()
 - Ou Extension de la classe **DefaultTreeCellRenderer**
 - Implémentation de l’interface de rendu
TreeCellRenderer
- `public void setCellRenderer(TreeCellRenderer x)`

DefaultTreeCellRenderer

- Icône associée aux feuilles

`void setLeafIcon(Icon newIcon)`

- Icône associée noeuds repliés

`void setClosedIcon(Icon newIcon)`

- Icône associée aux noeuds dépliés

`void setOpenIcon(Icon newIcon)`

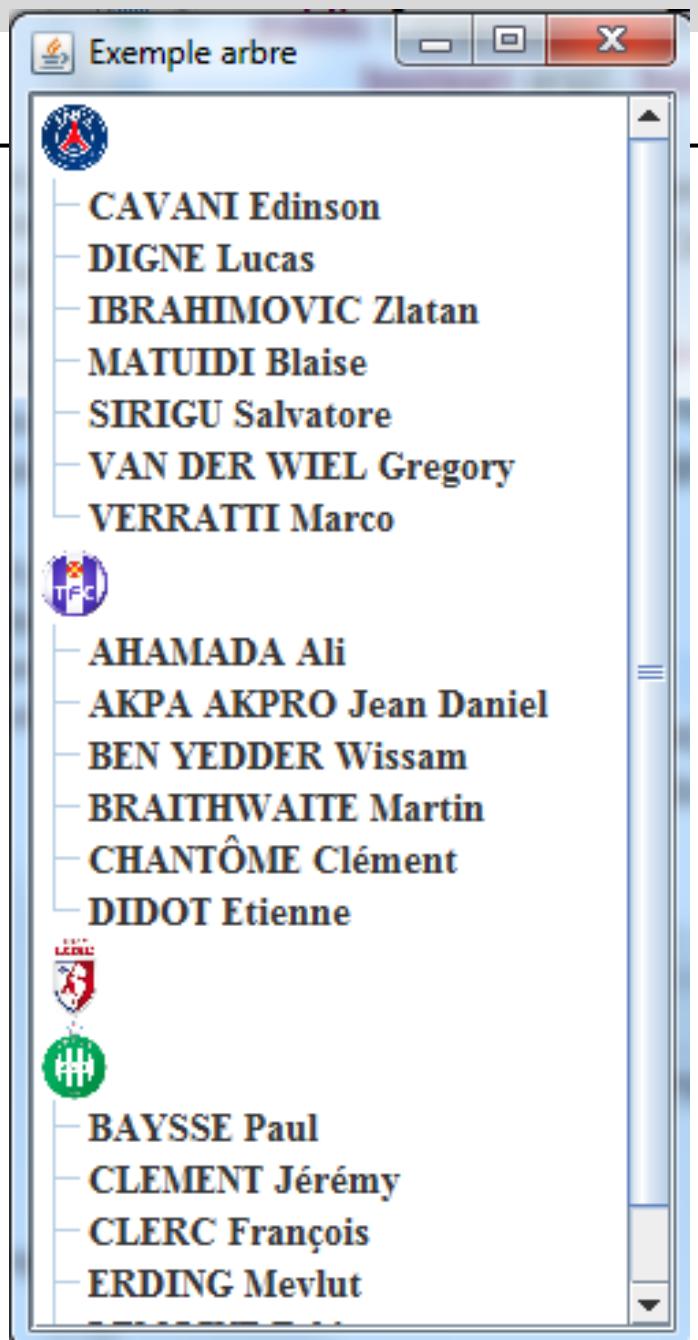
Mise en oeuvre d'une classe de rendu

- Extension de la classe DefaultTreeCellRenderer
- Surcharge de la méthode

```
public Component getTreeCellRendererComponent(  
    JTree tree,  
    Object value,  
    boolean sel,  
    boolean expanded,  
    boolean leaf,  
    int row,  
    boolean hasFocus)
```

Exercice 5

- Ecrire la classe de rendu pour avoir le logo des clubs, sans leur nom et le nom des joueurs écrit en gras sans icônes devant



Correction

```
public class RenduClub extends DefaultTreeCellRenderer {  
    public Component getTreeCellRendererComponent(JTree arg0, Object arg1,  
        boolean arg2, boolean arg3, boolean arg4, int arg5, boolean arg6)  
    {  
        if(arg1 instanceof Club)  
        {  
            ImageIcon ii = new ImageIcon(((Club)arg1).getIconName());  
            Image i = ii.getImage().getScaledInstance(30,-1,Image.SCALE_DEFAULT);  
            setIcon(new ImageIcon(i));  setText("");  
        }  
        else  
        {  
           setFont(new Font(Font.SERIF,Font.BOLD,14));  
            setText(arg1.toString());  setIcon(null);  
        }  
        return this;  
    }  
}
```

Utilisation du Timer

Javax.swing.Timer

- Envoie un événement de type ActionEvent toutes les N millisecondes
 - Evenements gérés de la même manière que pour les boutons

- Constructeur

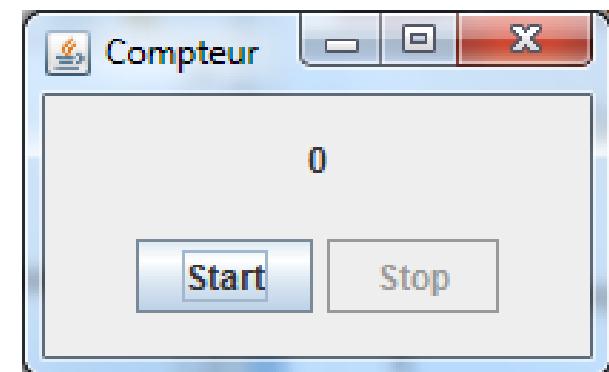
Timer(int delay, ActionListener listener)

- Principales méthodes

**void setDelay(int delay)
void start()
void stop()**

Exercice : Compteur

- Réalisez le compteur suivant
 - Au démarrage seul le bouton Start est activé et le compteur est initialisé à 0
 - Après un clic sur le bouton Start
 - Le compteur est réinitialisé à 0 si ce n'est pas le cas
 - le bouton Stop est activé
 - le bouton Start est désactivé
 - Jusqu'à l'appui sur Stop, le compteur est incrémenté de 1 toutes les secondes
 - Après un clic sur le bouton Stop
 - Le compteur est arrêté
 - le bouton Stop est désactivé
 - le bouton Start est activé



Dessiner sur un composant

Dessiner en JAVA

- Où ?
 - Sur le composant Canvas en AWT
 - Sur tous les composants SWING héritant de JComponent
- Comment ?
 - Pour SWING, redéfinir la méthode

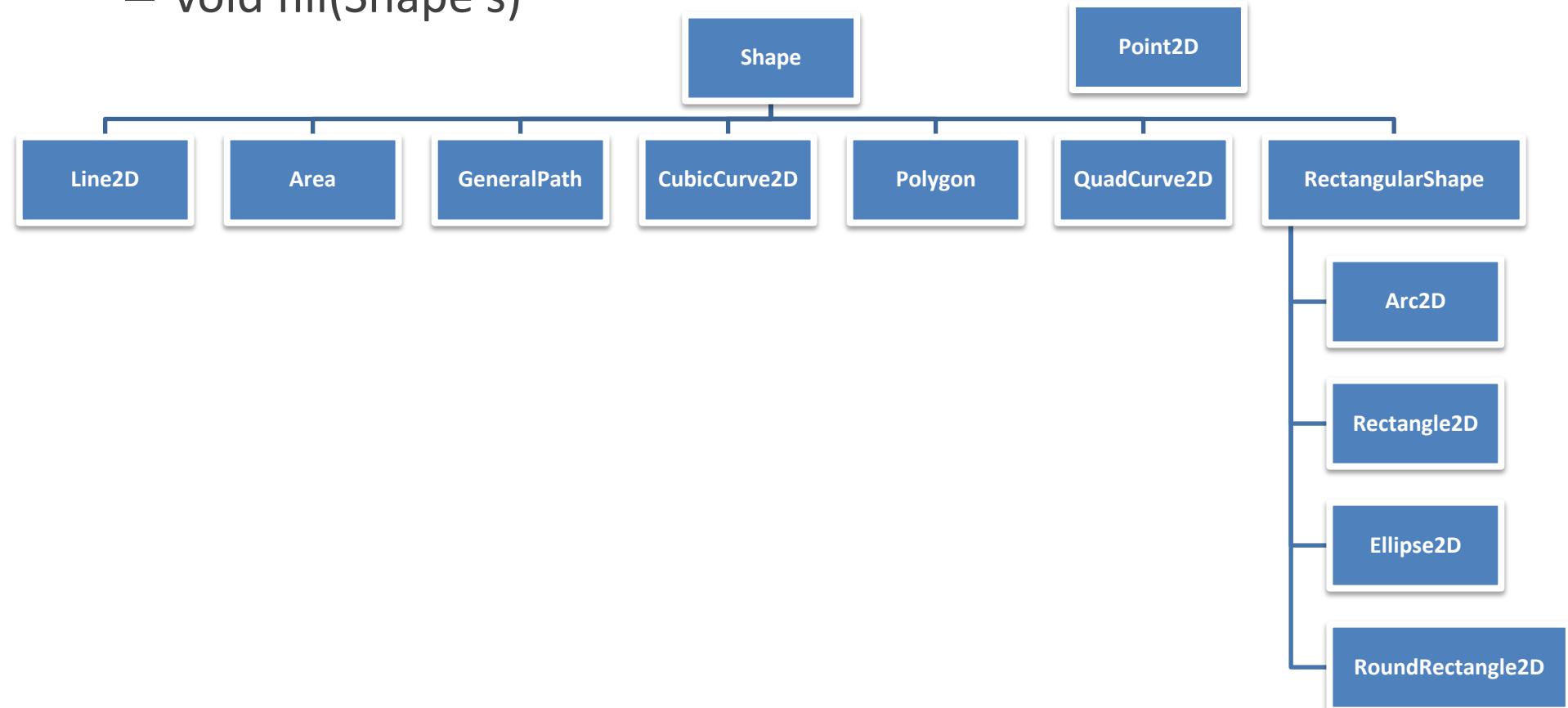
```
void paintComponent(Graphics g)
```
 - Les méthodes de dessin se trouvent dans l'objet **Graphics** ou **Graphics2D** qui en hérite

La classe Graphics

- **Dessiner les contours**
 - **draw3DRect(int x, int y, int width, int height, boolean raised)**
 - **drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)**
 - **drawLine(int x1, int y1, int x2, int y2)**
 - **drawOval(int x, int y, int width, int height)**
 - **drawPolygon(int[] xPoints, int[] yPoints, int nPoints)**
 - **drawPolygon(Polygon p)**
 - **drawPolyline(int[] xPoints, int[] yPoints, int nPoints)**
 - **drawRect(int x, int y, int width, int height)**
 - **drawRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)**
- Remplir la forme : fill à la place de draw

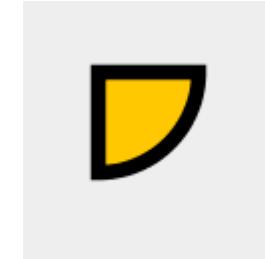
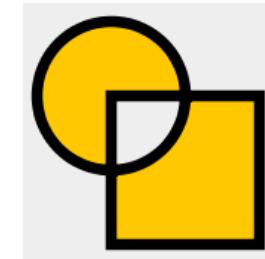
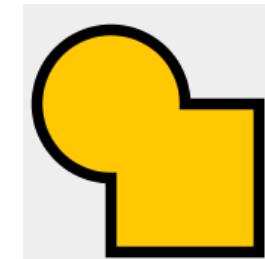
La classe Graphics2D

- Affichage de Shape
 - void draw(Shape s)
 - void fill(Shape s)



Gestion des Aires

- La class Area
 - Area(Shape s)
- Méthodes de combinaison de forme
 - add(Area rhs)
 - exclusiveOr(Area rhs)
 - intersect(Area rhs)
 - subtract(Area rhs)



Méthodes de Graphics2D

- Fonctionnalités supplémentaires avec Graphics2D
 - Propriété d'affichage : RenderingHints

```
void setRenderingHint(RenderingHints.Key hintKey, Object hintValue)
```

- Style de trait : Stroke

```
void setStroke(Stroke s)
```

- Outil de remplissage : Paint

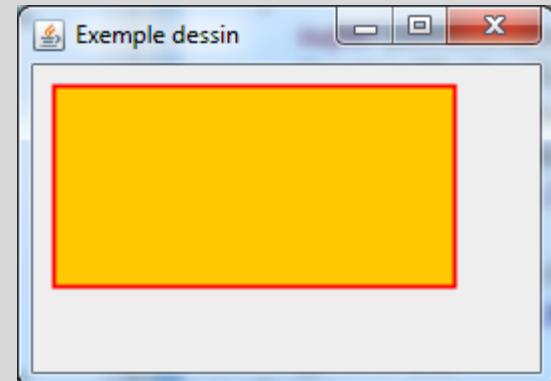
```
void setPaint(Paint paint)
```

- Transformation géométrique : AffineTransform

```
void setTransform(AffineTransform Tx)
```

Exemple

```
public class PanneauDessin extends JComponent
{
    public void paintComponent(Graphics g)
    {
        super.paintComponent(g);
        Graphics2D g2 = (Graphics2D)g;
        g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                            RenderingHints.VALUE_ANTIALIAS_ON);
        Rectangle2D.Double r = new Rectangle2D.Double(10,10,200,100);
        g2.setColor(Color.ORANGE);
        g2.fill(r);
        g2.setStroke(new BasicStroke(2.0f));
        g2.setColor(Color.RED);
        g2.draw(r);
    }
}
```



Exercice : tracé de ligne

- Sur une interface ne contenant qu'un JComponent
 - afficher la trace du déplacement du pointeur de la souris quand le bouton gauche de la souris est enfoncé
 - Créer un menu pour gérer la couleur de la trace et l'épaisseur de celle-ci

