Java Interfaces graphiques

Intervenant : Aina Lekira
aina.lekira@univ-lemans.fr

Interfaces graphiques en Java
Petit aperçu

- *Graphical User Interface* (GUI)
  - AWT (*Abstract Window Toolkit*)
    - La bibliothèque originelle d'interfaces graphiques en Java
  - Swing
    - Ajout dans Java 1.1
  - SWT (*Standard Widget Toolkit*)
    - Bibliothèque initiée par IBM et maintenue par Eclipse Foundation
    - Alternative à AWT et Swing de Sun
      - Non reconnue par la Java Community Process
  - Java FX
    - plateforme (initiée par Sun) construite sur la technologie Java
    - création et déploiement d'applications internet riches (RIA) se comportant de manière consistante sur différentes plateformes
    - Existence d'un ensemble fourni d'API multimedia et graphiques

AWT vs. Swing ?

- Swing est plus récent mais n'est pas le « remplaçant » de AWT
  - Ces deux APIs sont complémentaires
- Existence d'éléments équivalents dans les 2 API
  - Les composants boutons (`Button`), menu (`MenuBar`), barres de défilement (`ScrollBar`) , … de AWT

    existent dans SWING et leurs noms commencent (souvent) par J (`Jbutton`, `Jmenu`, …)
- Swing prend en charge plus d'éléments : les champs de mot de passe (`JPasswordField`), les barres de progression (`JProgressBar`), …

AWT vs. Swing ?

- Deux « philosophies » différentes
- AWT prend en charge des composants lourds (*heavyweight*)
  - Le GUI s'adapte à ce que l'utilisateur a l'habitude de voir
    - Apparence différente selon l'OS
      - Sous Windows un bouton aura l'aspect d'un bouton Windows, sous Mac OS d'un bouton mac
- SWING prend en charge des composants légers (*lightweight*)
  - Les éléments de contrôles sont implémentées en Java
  - L'apparence et les fonctionnalités des éléments de contrôles des composants n'est pas déterminé par l'OS
    - Même apparence quel que soit l'OS

AWT vs. Swing ?

- Deux « philosophies » différentes
- AWT prend en charge des composants lourds (*heavyweight*)
  - Le GUI s'adapte à ce que l'utilisateur a l'habitude de voir
    - Apparence différente selon l'OS
      - Sous Windows un bouton aura l'aspect d'un bouton Windows, sous Mac OS d'un bouton mac
- SWING prend en charge des composants légers (*lightweight*)
  - Les éléments de contrôles sont implémentées en Java
  - L'apparence et les fonctionnalités des éléments de contrôles des composants n'est pas déterminé par l'OS
    - Même apparence quel que soit l'OS

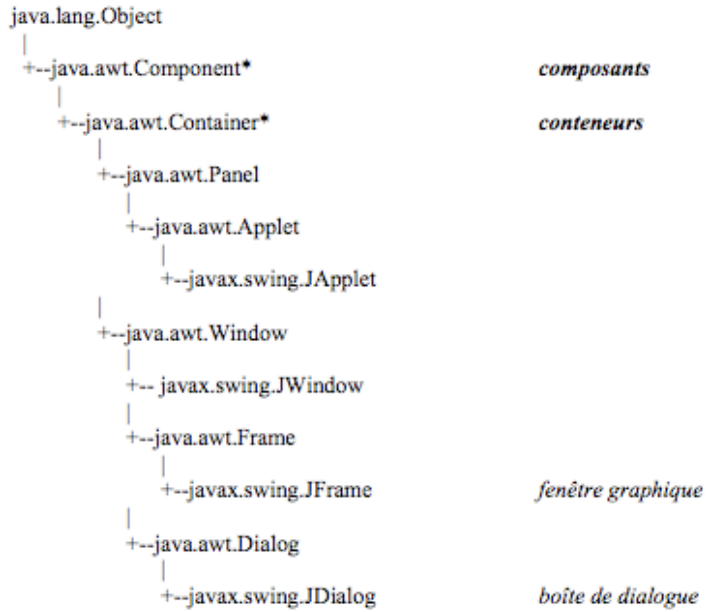
Structure d'une interface graphique

- Conteneurs
- Composants
- Gestionnaire de mise en page/placement/disposition/
- Gestionnaire d'événements

Swing

- Une GUI Swing est construite à partir d'un objet du système (*heavyweight*)
  - JFrame : fenêtre graphique
  - JWindow : fenêtre sans décoration, non déplaçable (utilisée pour les fenêtres de démarage)
  - JDialog : boîte de dialogue (modale ou non)
  - JApplet : application destinée à être exécutée dans un navigateur

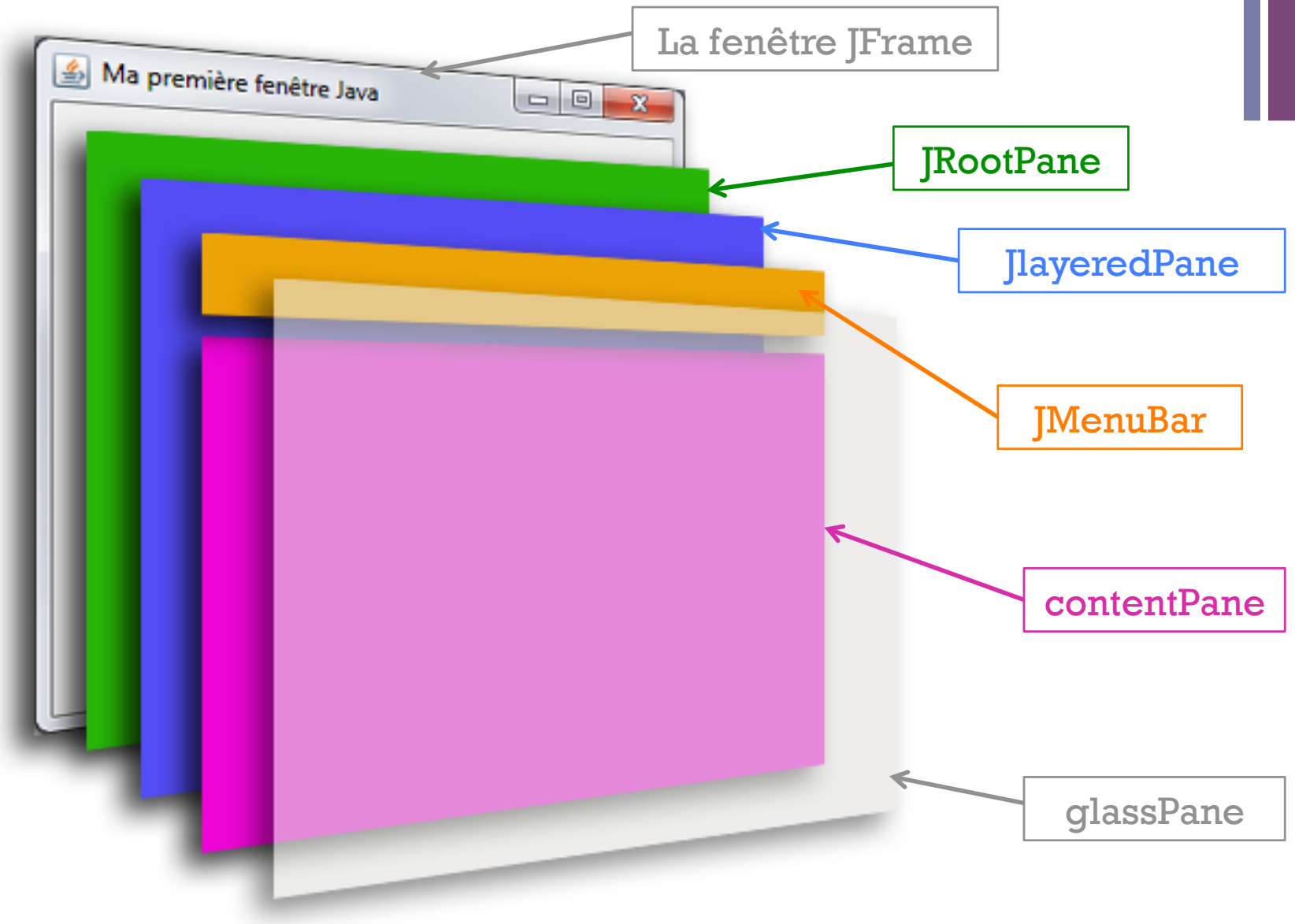
Ce sont des extensions de la version java.awt.XXX



Fenêtre graphique - JFrame

- Une instance de JFrame est composée d'un (seul) JRootPane
- JRootPane est composé de
  - LayeredPane
    - JMenuBar pour les menus
    - contentPane pour les composants
      - Accessible via getContentPane()
  - glassPane : couche utilisée pour intercepter les actions de l'utilisateur avant qu'elles ne parviennent aux composants
    - c'est un JPanel transparent !
  - Décoration
    - Bordure, titre, icône, boutons de fermeture, …

Les composants conteneurs (1)



Fenêtre graphique - JFrame

- Création d'une JFrame

```
JFrame fenetre = new JFrame() ;
fenetre.setTitle("Ma première fenêtre Java") ;
```

```
JFrame fenetre = new JFrame("Ma première fenêtre Java")
```

- préciser une taille : setSize(), setBounds()

sinon 

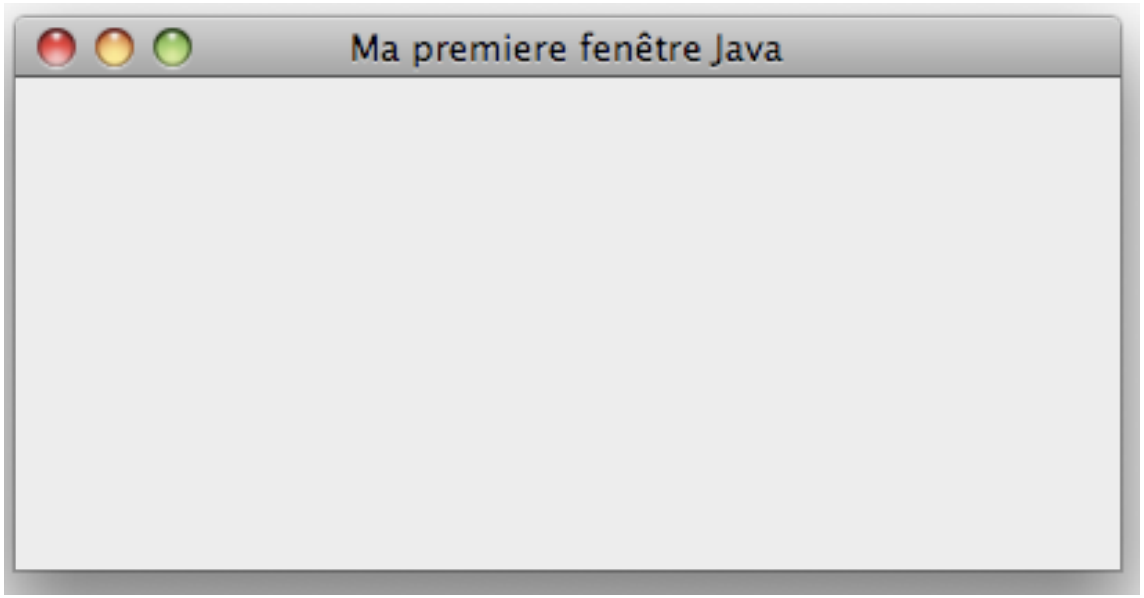
- rendre la fenêtre visible : setVisible(true)

Fenêtre graphique - JFrame

- Fermeture d'une JFrame
  - Définir le mode via setDefaultCloseOperation
    - DO_NOTHING_ON_CLOSE
    - HIDE_ON_CLOSE: rend la fenêtre invisible (mode par défaut)
    - DISPOSE_ON_CLOSE : cache la fenêtre et libère toutes les ressources systèmes associées; elles seront réallouées si la fenêtre redevient visible
    - EXIT_ON_CLOSE: termine le programme
- Propriétés d'une JFrame
  - redimensionnement : setResizable
  - décoration : setUndecorated

Exemple

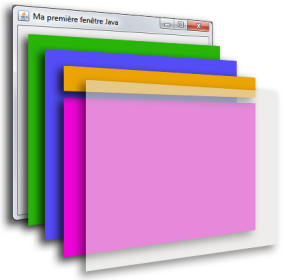
```
import java.awt.* ;
import javax.swing.* ;

public class MaFenetre extends JFrame {
    public MaFenetre() {
        this.setTitle("Ma premiere fenêtre Java");
        this.setSize(400, 200);
        this.setDefaultCloseOperation(EXIT_ON_CLOSE);
        this.setVisible(true);
    }
    public static void main(String[] args) {
        JFrame fen = new MaFenetre();
    }
}
```



Les composants

JFrame et ses composants

- Une fenêtre est destinée à contenir d'autres composants via `contentPane`
  - Accessible via *getContentPane()* qui fournit une référenc contenu de la fenêtre
  - Cette référence est de type `Container`



- Ajout et suppression d'un composant dans une fenêtre via les méthodes add() et remove() de `Container`

distinction entre la gestion logique des composants (ajout/ suppression de la fenêtre)de leurs placements dans la fenêtre (géré par un LayoutManager)

Les composants

- Un composant Swing est un `Jcomponent` (classe abstraite)
  - Extension de cette classe pour créer `Jpanel, JScrollPane, Jbutton, Jtable, ...`
- Quelques méthodes

Composant	Description
`setVisible(boolean b)`	Montre ou cache un composant
`setEnabled(boolean b)`	Active et désactive un composant de saisie de texte formaté (ex : les dates)
`isEnabled()`	Informe sur l'état du composant (activé ou non)
`setBackground(Color c)`	Modifie la couleur de fond d'un composant
`setSize(int largeur, int hauteur)`	Modifie la taille d'un composant
`setBounds(int x, int y, int l, int h)`	Modifie la position et la taille d'un composant

la taille des composants peut également être gérée via set/getPreferredSize, set/getMinimumSize, set/getMaximumSize

Les composants conteneurs

- Jpanel : panneau élémentaire
- JTabbedPane : panneau à onglets
- JScroolPane : panneau à ascenseurs
- JSplitPane : panneau partagé horizontalement ou verticalement
- JLayeredPane : panneau multicouches
- JDesktopTopPane panneau des fenêtres pour une application multi-documents
- JToolBar : barre d'outils détachable (pouvant se raccrocher et se décrocher à un autre panneau)

JPanel

- C'est un panneau élémentaire destiné à contenir d'autres composants
  - Il est muni d'un gestionnaire de placement

Creating a `JPanel`

Constructor	Purpose
JPanel() JPanel(LayoutManager)	Creates a panel. The `LayoutManager` parameter provides a layout manager for the new panel. By default, a panel uses a `FlowLayout` to lay out its components.

Managing a Container's Components

Method	Purpose
void add(Component) void add(Component, int) void add(Component, Object) void add(Component, Object, int) void add(String, Component)	Adds the specified component to the panel. When present, the `int` parameter is the index of the component within the container. By default, the first component added is at index 0, the second is at index 1, and so on. The `Object` parameter is layout manager dependent and typically provides information to the layout manager regarding positioning and other layout constraints for the added component. The `String` parameter is similar to the `Object` parameter.
int getComponentCount()	Gets the number of components in this panel.
Component getComponent(int) Component getComponentAt(int, int) Component getComponentAt(Point) Component[] getComponents()	Gets the specified component or components. You can get a component based on its index or *x*, *y* position.
void remove(Component) void remove(int) void removeAll()	Removes the specified component(s).

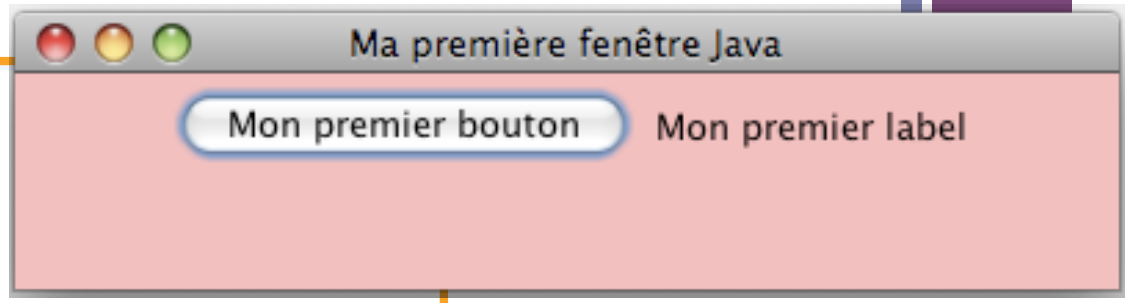
Setting or Getting the Layout Manager

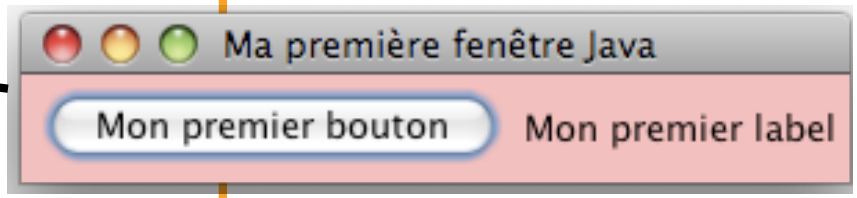
Method	Purpose
void setLayout(LayoutManager) LayoutManager getLayout()	Sets or gets the layout manager for this panel. The layout manager is responsible for positioning the panel's components within the panel's bounds according to some philosophy.

http://docs.oracle.com

Exemple

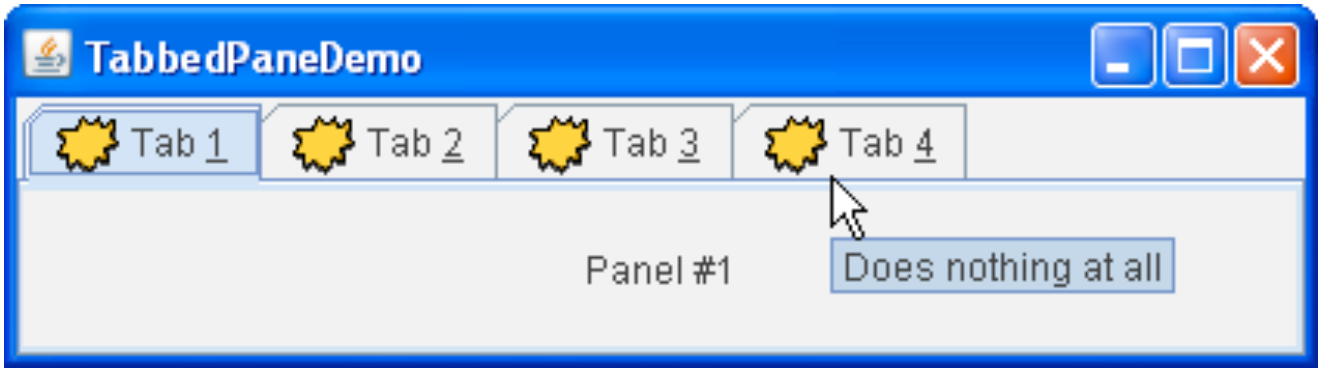
```
import java.awt.Color;
import javax.swing.*;

public class MaFenetre2 extends JFrame {

    public MaFenetre2(){
        this.setTitle("Ma première fenêtre Java");
        this.setSize(400, 400);
        JPanel monPanel = new JPanel();
        monPanel.setBackground(Color.PINK);

        JButton monBouton = new JButton("Mon premier bouton") ;
        JLabel monLabel = new JLabel("Mon premier label");
        monPanel.add(monBouton);
        monPanel.add(monLabel);

        /* affecte le JPanel comme content pane de  MaFenetre2 */
        this.setContentPane(monPanel);
        /* adapte la fenetre à la taille des composants */
        this.pack();
        this.setVisible(true);
    }

    public static void main(String args[]) {
        JFrame fen = new MaFenetre2() ;
    }
}
```





JTabbedPane

- C'est un conteneur pouvant contenir d'autres conteneurs
- Navigation entre ces différents conteneur se fait via des onglets
  - Un onglet est un JComponent (de type Jpanel généralement)
  - Un onglet a un titre, une icône, une bulle d'information (ToolTipText), un indice dans la liste des onglets (*setSelectedIndex()*)

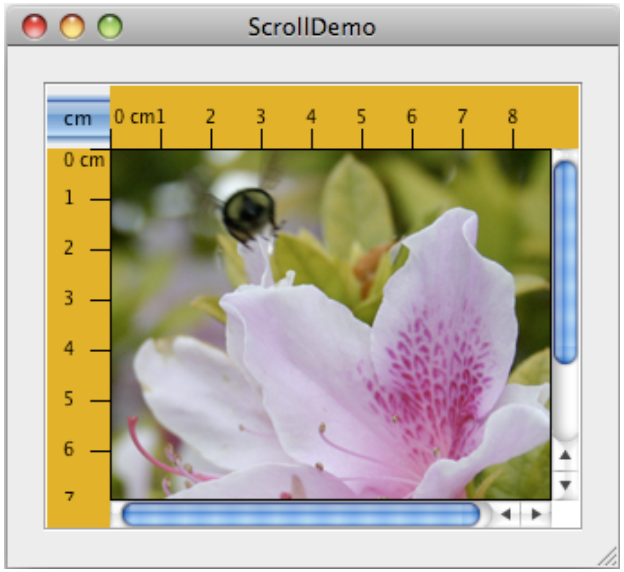


http://docs.oracle.com

JScrollPane

- C'est un conteneur qui fournit une barre de défilement à un autre composant
  - À utiliser pour afficher un composant
    - qui est grand
    - avec une taille qui peut changer dynamiquement

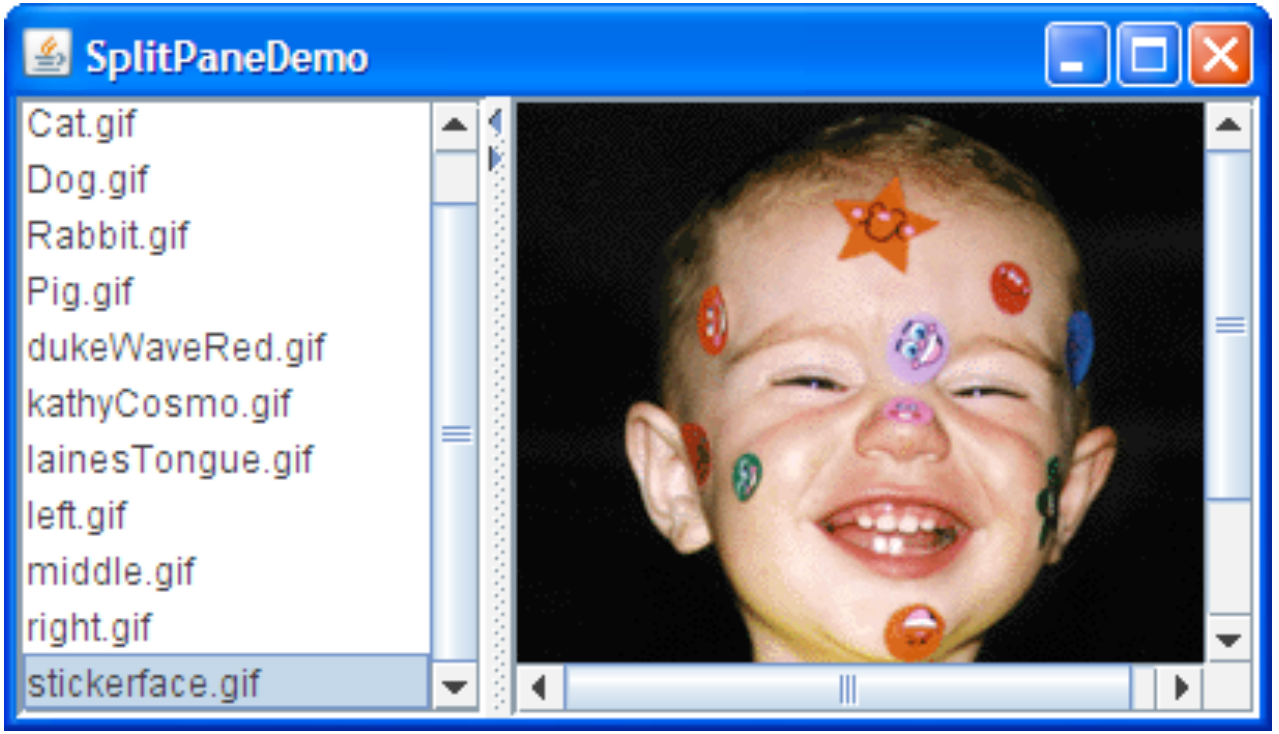
- Les barres de défilement sont munies d'uns stratégie d'affichage
  - barre verticale et/ou horizontale
  - Visibilité : toujours visible, jamais visible, si besoin



http://docs.oracle.com

JSplitPane

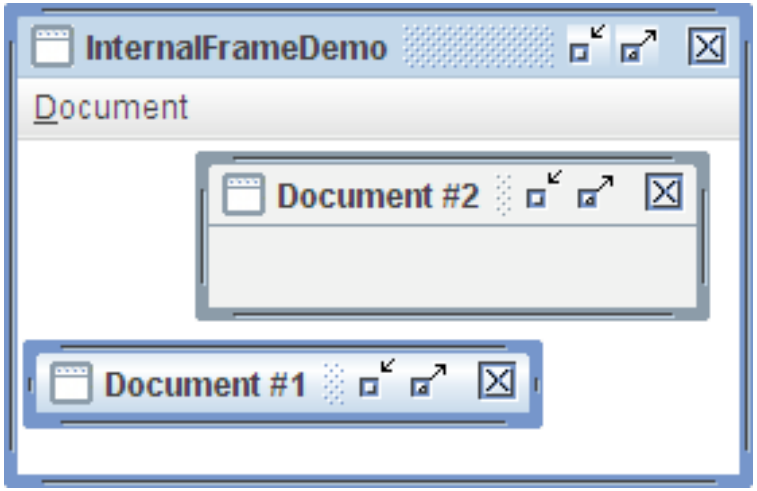
- C'est un conteneur qui permet de séparer deux composants verticalement ou horizontalement.
  - La barre de division entre les deux composants peut être déplaçable
- Possibilité de diviser la fenêtre en 3 ou plusieurs composants via des JSplitPane dans des JSplitPane



http://docs.oracle.com

JDesktopPane

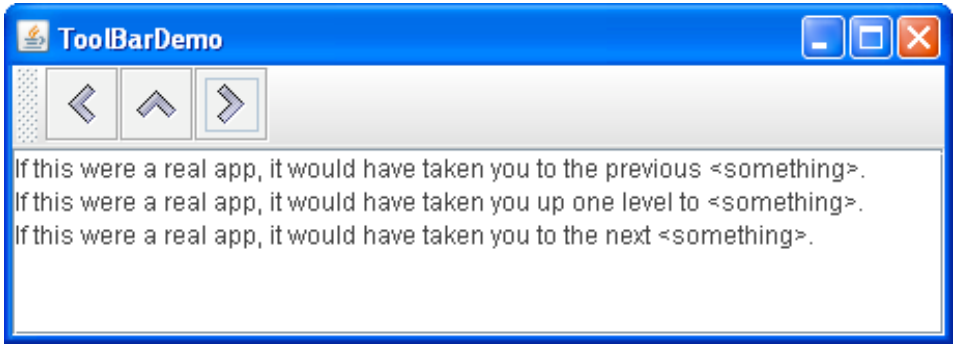
- C'est un conteneur qui permet de réaliser des applications MDI (*Multiple Document Interface*)
- Le *contentPane* de la fenêtre principale doit être une instance de ce conteneur
- Les fenêtres internes sont des instances de `JInternalFrame`
  - Ces instances doivent être raccrochées au *contentPane* de type JDesktopPane

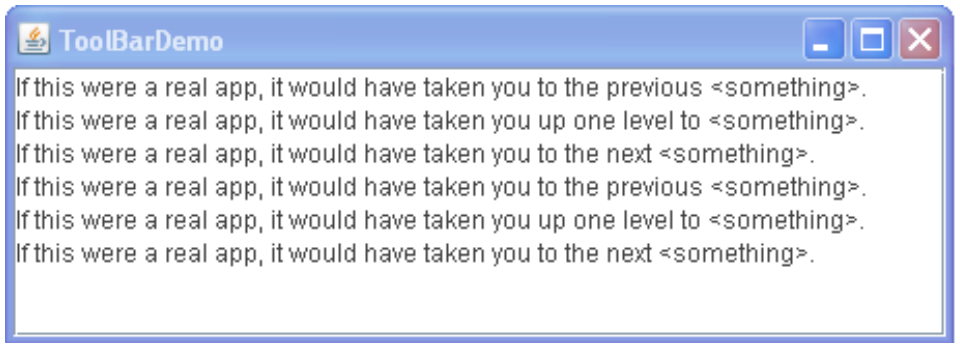


http://docs.oracle.com

JToolBar

- C'est un conteneur qui regroupe plusieurs composants (généralement des boutons avec des icônes)
  - Dans une barre verticale, horizontale ou indépendante de la fenêtre
- A utiliser généralement pour accéder à des fonctionnalités qui existent également dans les menus
  - Fournir la même fonctionnalité dans les options de menus et boutons de barre d'outils via un objet de type `Action`



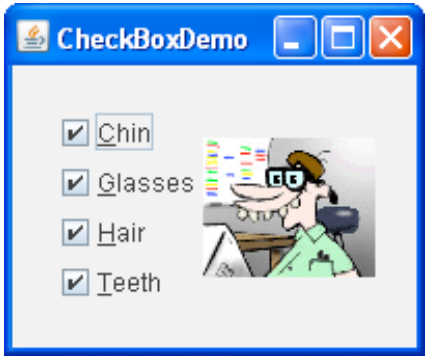


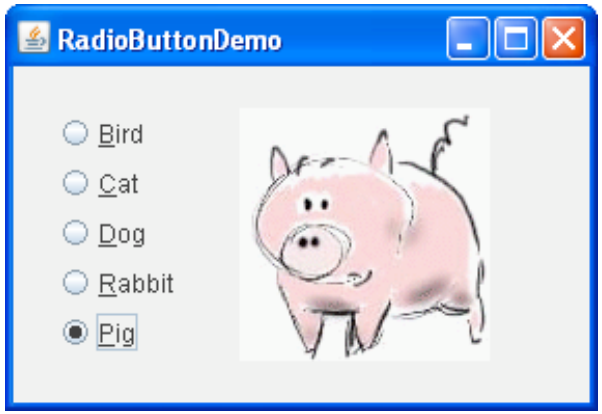
http://docs.oracle.com

Les composants de base

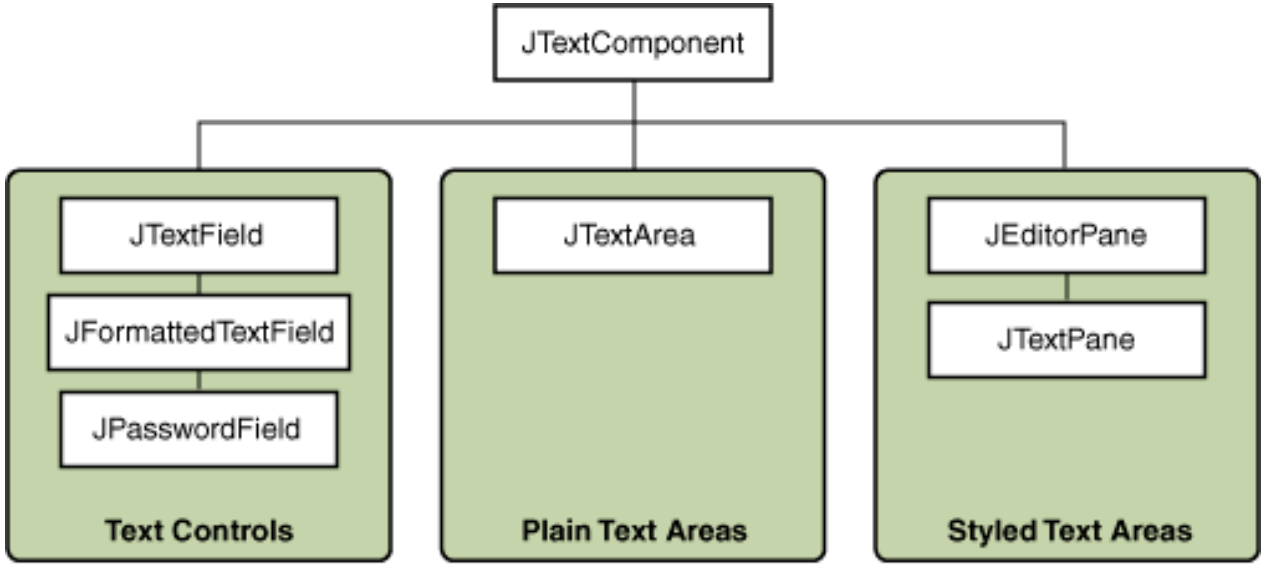
Composant	Description
`JLabel`	Texte (brut ou mis en forme) et/ou une icône
`JToolTip`	Info bulle affichée au survol du composant
`JButton`	Bouton
`JRadioButton`	Bouton radio
`JChekBox`	Case à cocher

http://docs.oracle.com





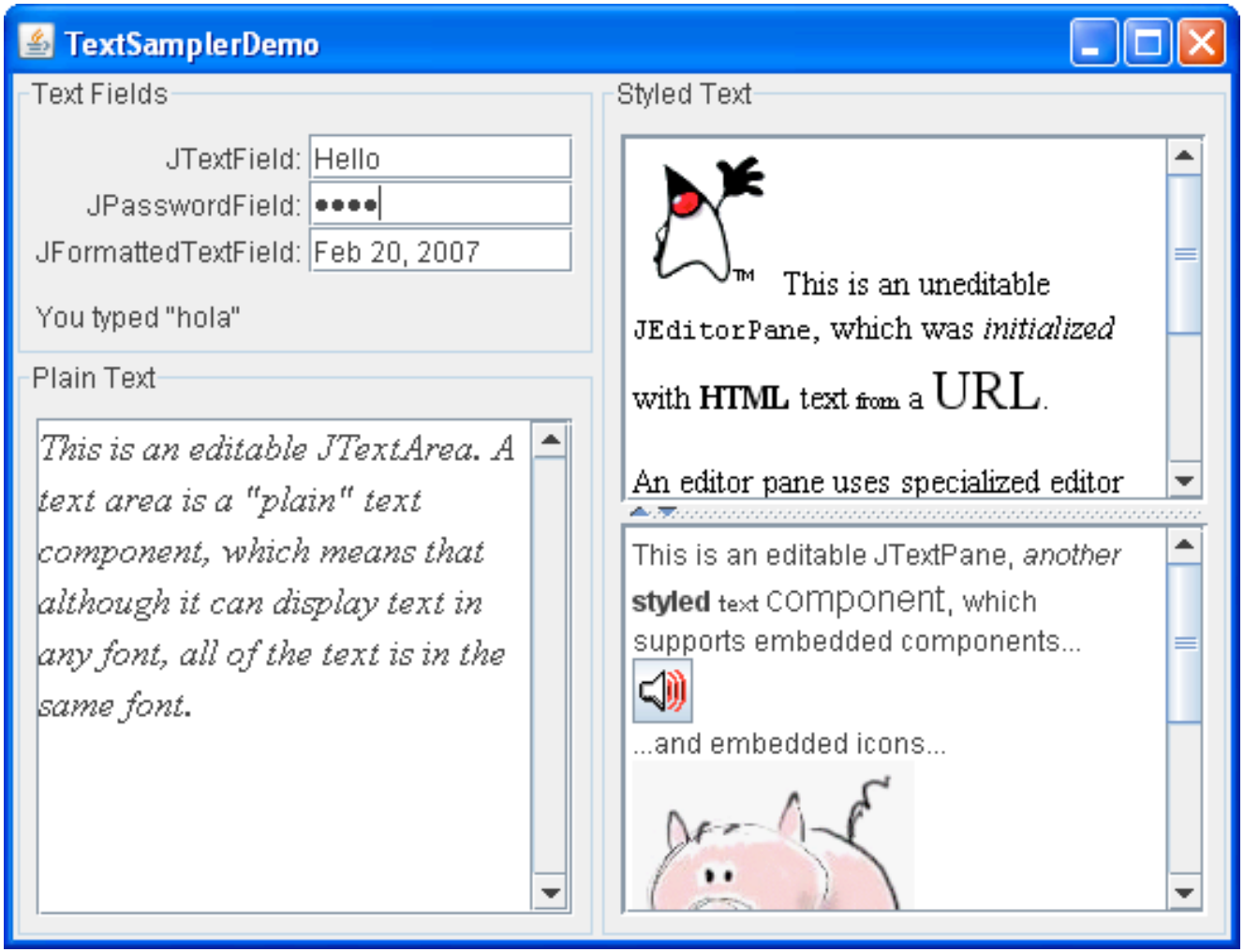
Les composants de saisie (1)



http://docs.oracle.com

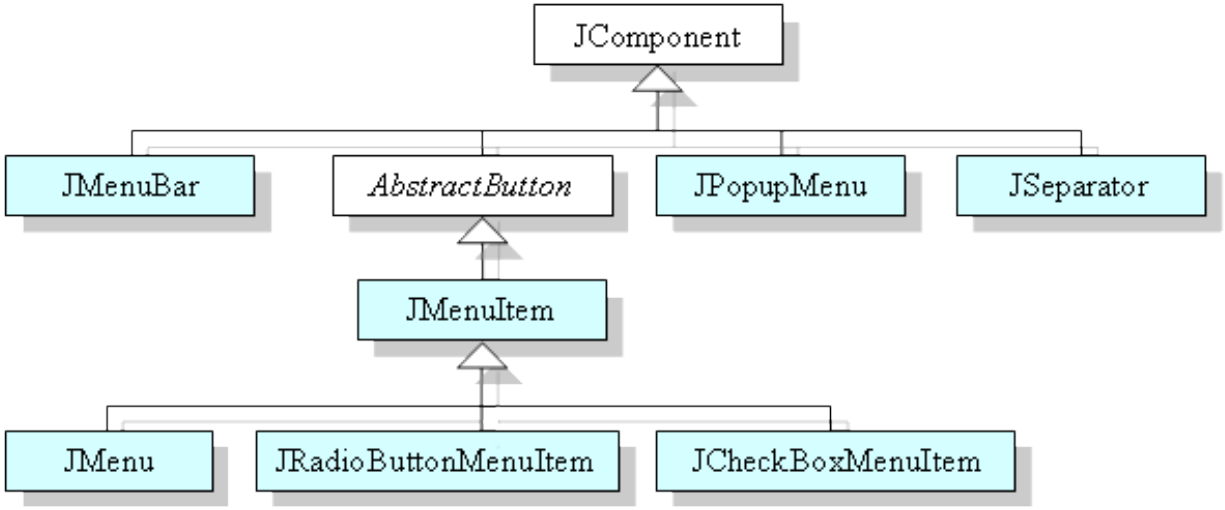
Composant	Description
`JTextField`	Champ de saisie simple
`JFormattedTextField`	Champ de saisie de texte formaté (ex : les dates)
`JPassworldField`	Champ de saisie pour les mots de passe
`JTextArea`	Zone de saisie sur plusieurs lignes
`JEditorPane`	Zone d'édition pour des contenus de différentes natures (plain, html, rtf, …)
`JTextPane`	Zone d'édition stylisé avec possibilité d'ajout d'images, …différentes différentes

Les composants de saisie (2)



http://docs.oracle.com

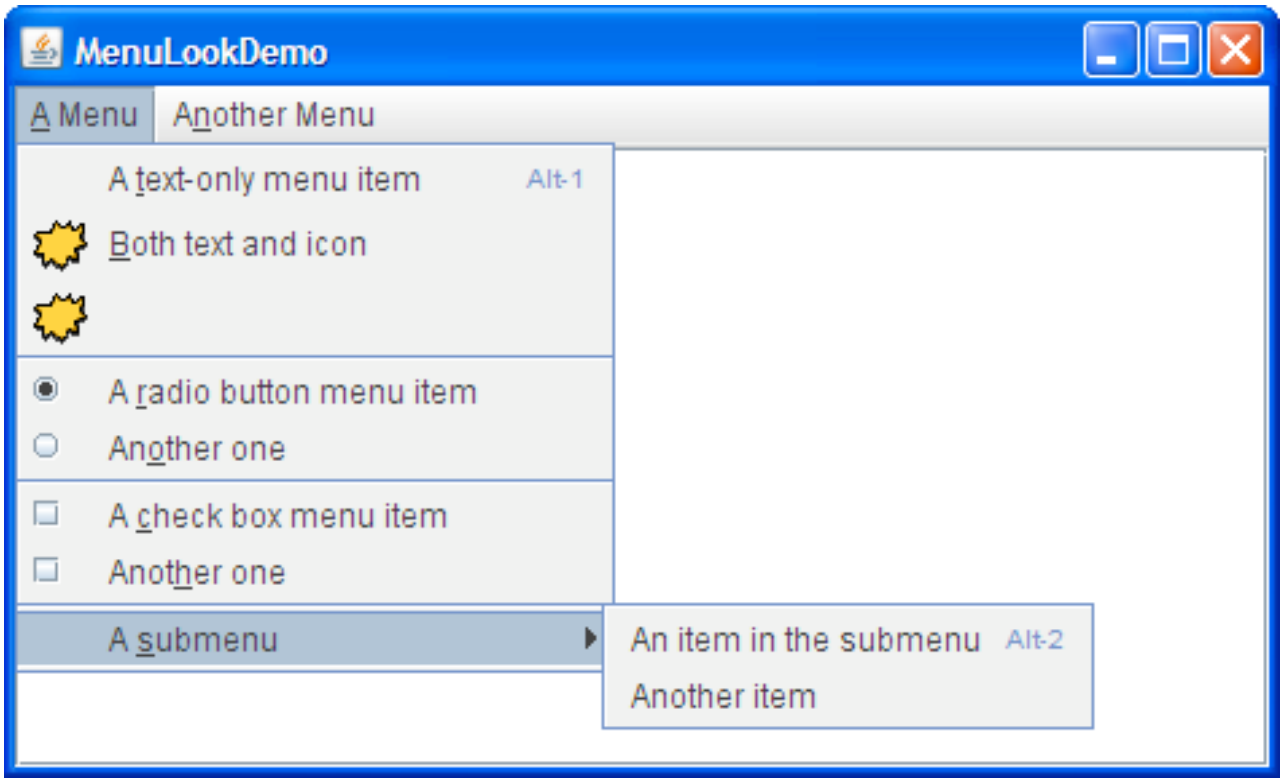
Les menus (1)



http://docs.oracle.com

Composant	Description
`JMenuBar`	Barre de menu
`JPopupMenu`	Menu surgissant
`JMenu`	Menu
`JMenuItem`	Item d'un menu
`JCheckBoxMenuItem`	Item de menu de type case à cocher
`JRadioButtonMenuItem`	Item de menu de type bouton radio
`JSeparator`	Séparateur

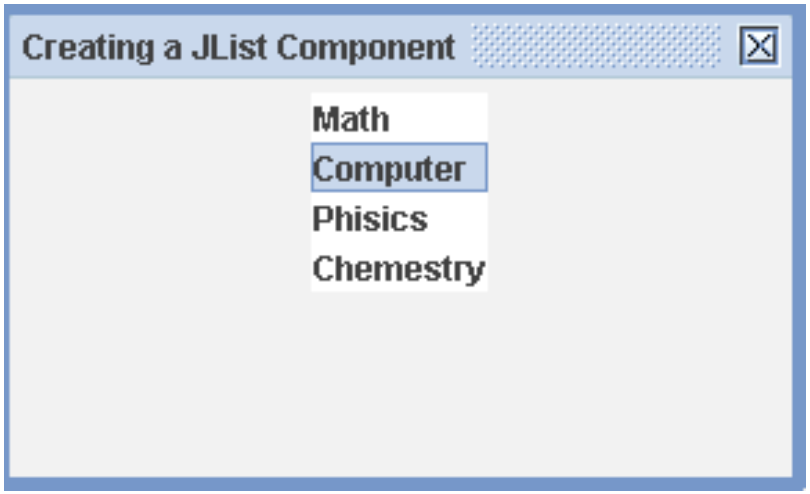
Les menus (2)

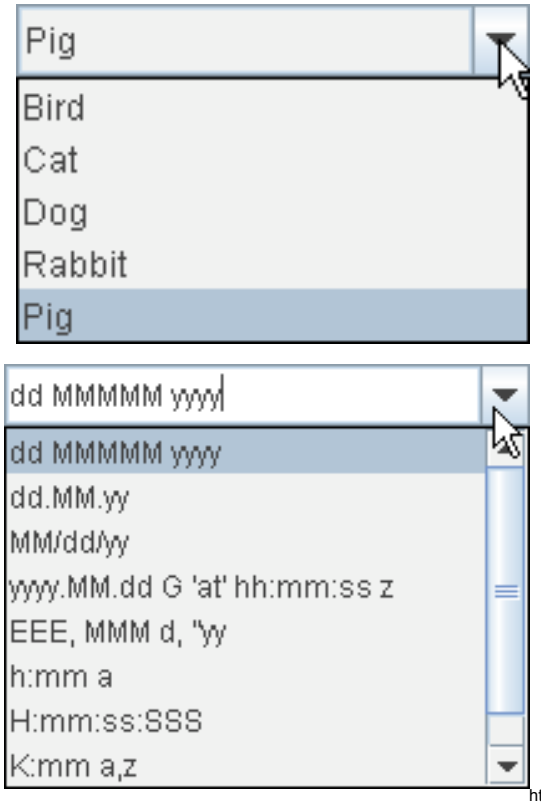


http://docs.oracle.com

Les listes

Composant	Description
`JList`	liste d'objets avec possibilité de sélection simple ou multiple
`JComboBox`	liste déroulante pouvant être éditable

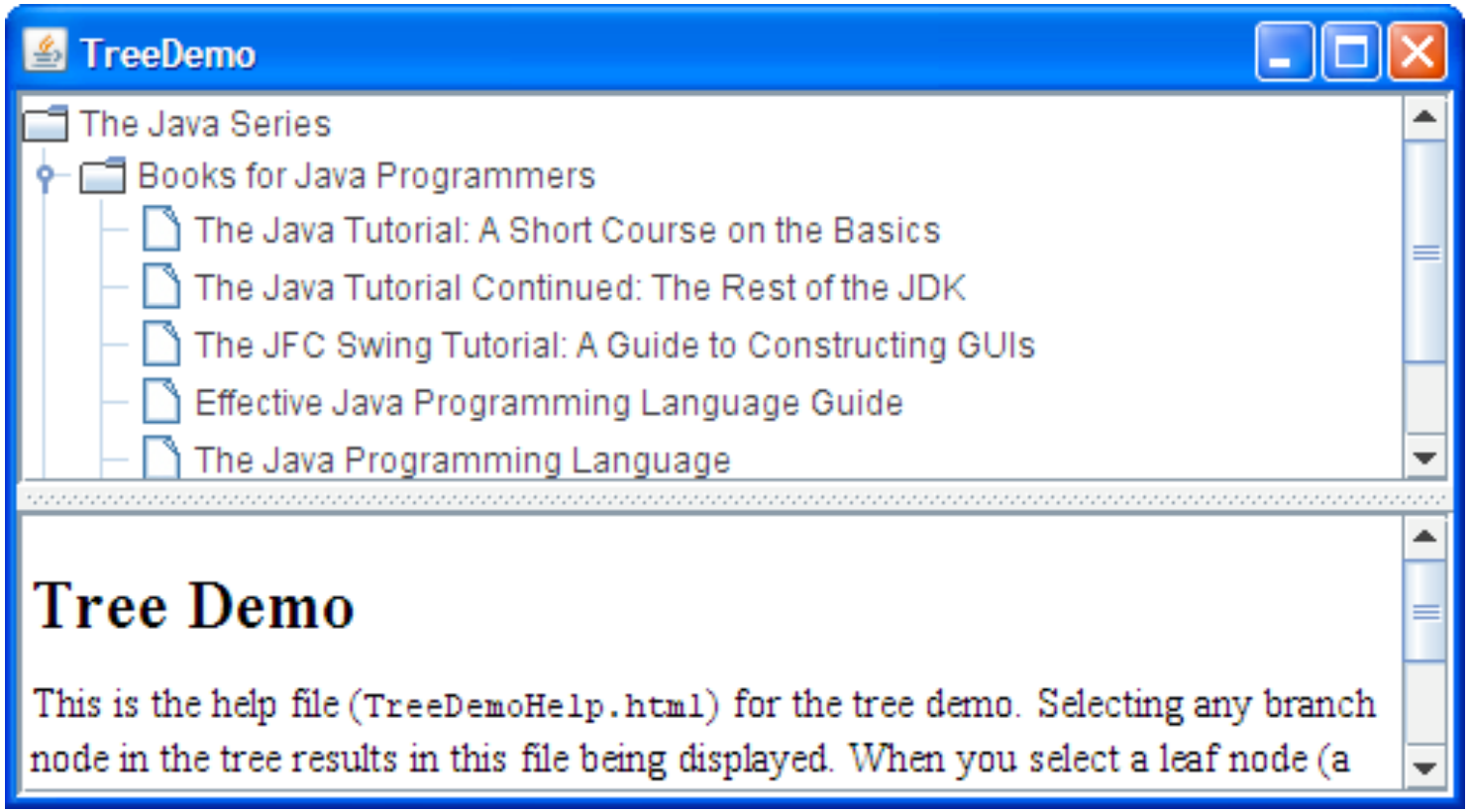




http://www.roseindia.net

Les arbres

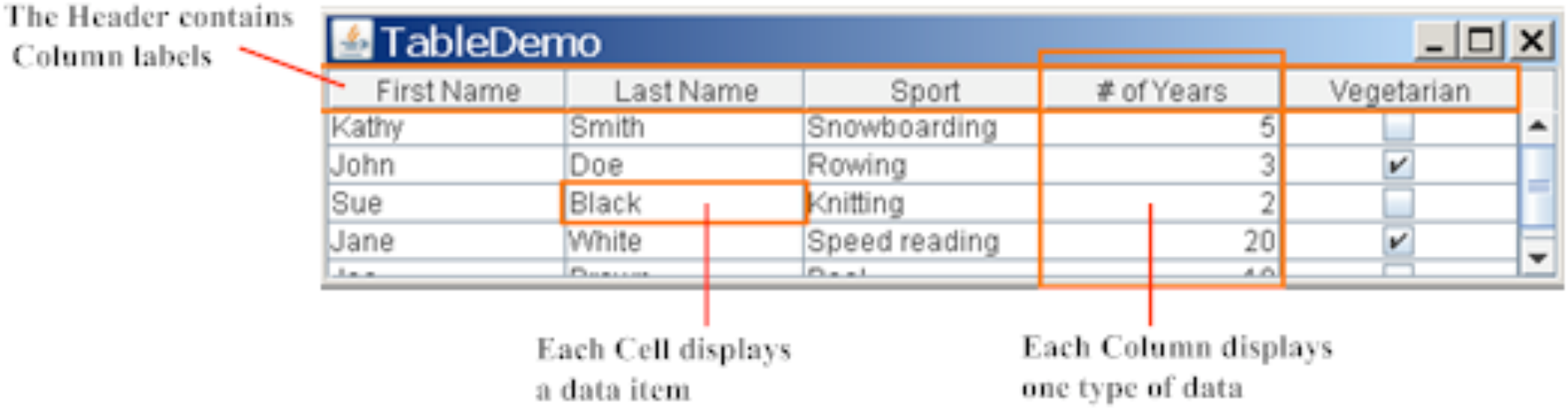
Composant	Description
`JTree`	Permet d'afficher des données hiérarchique dans une structure sous forme d'arbre



http://docs.oracle.com

Les tableaux

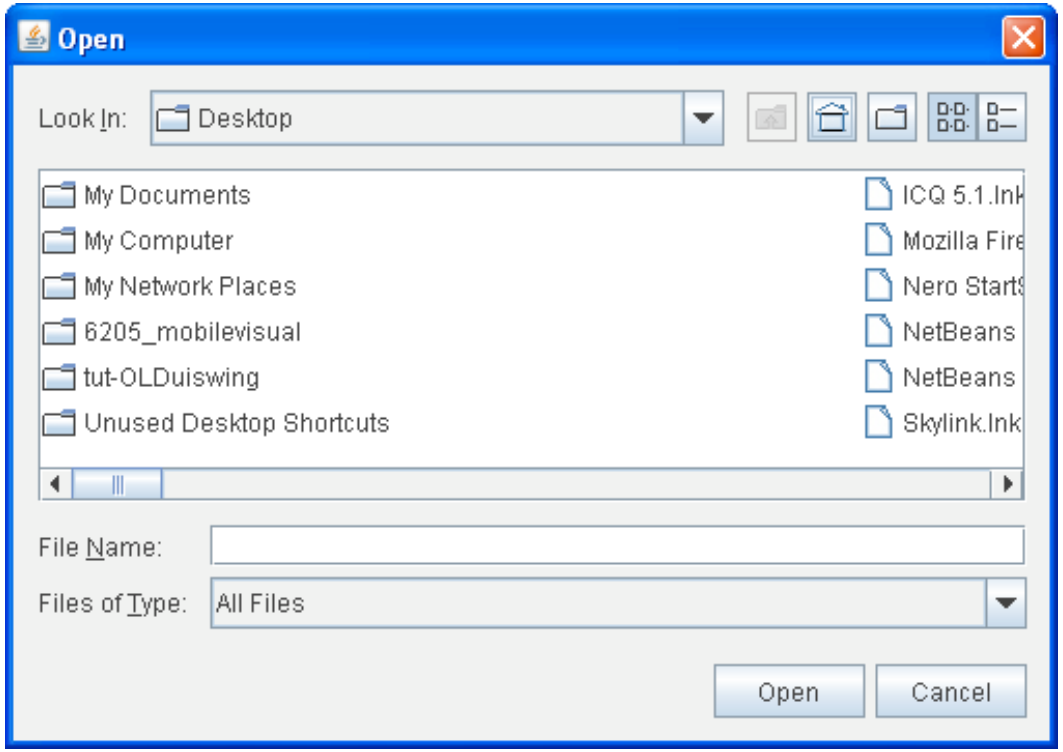
Composant	Description
`JTable`	Tableau éditable ou non pouvant afficher différents types de données

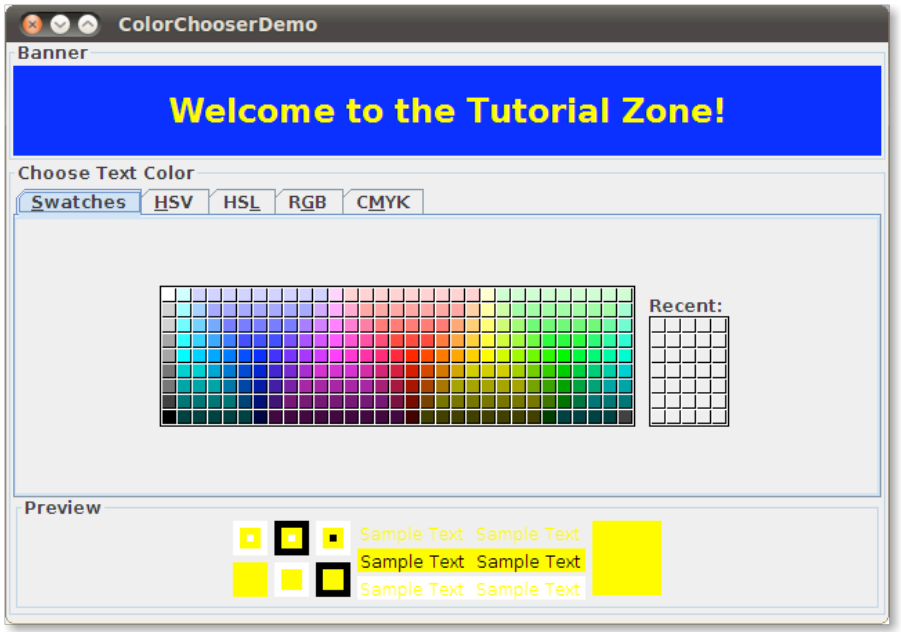


http://docs.oracle.com

Les boites de dialogue (1)

Composant	Description
`JFileChooser`	Choix de fichier
`JColorChooser`	Choix de couleur





http://docs.oracle.com

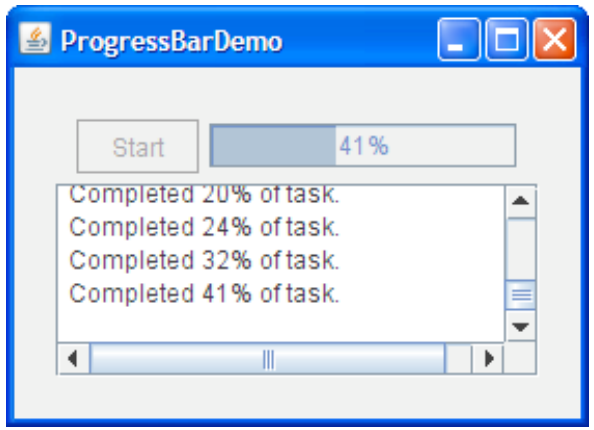
Les boites de dialogue (2)

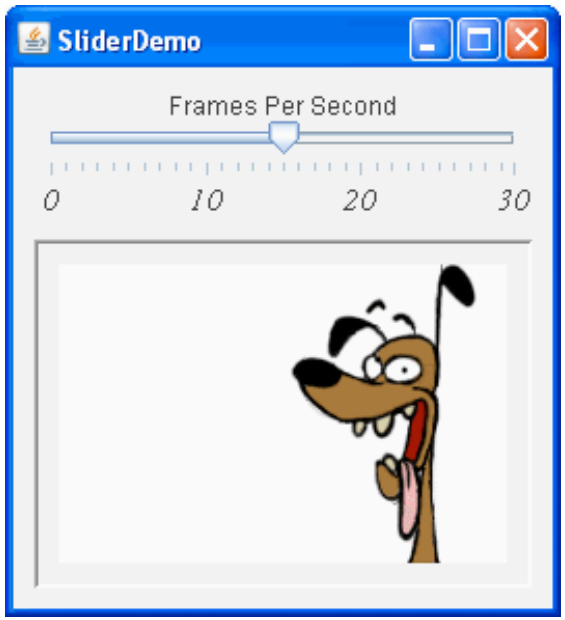
Composant	Description
`JOptionPane`	Choix standards (information, confirmation, saisie, personnalisé)

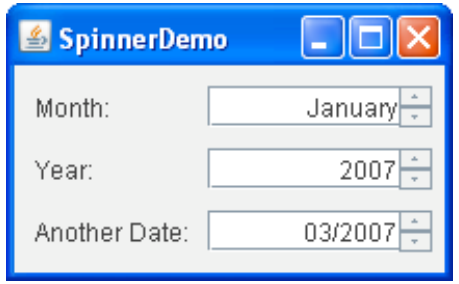
http://docs.oracle.com

Et encore d'autres composants

Composant	Description
`JProgressBar`	Barre de progression
`Jslider`	Choix de valeur avec glissière
`JSpinner`	Bouton fléché pour modifier une valeur







http://docs.oracle.com

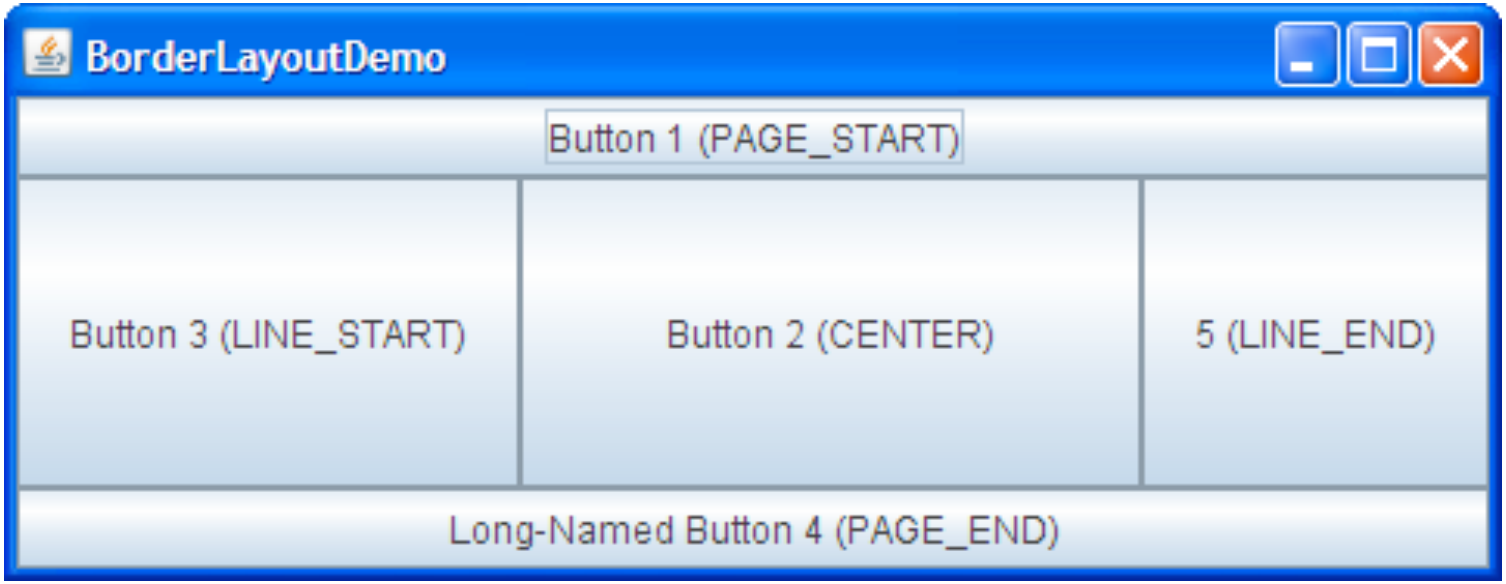
Les LayoutManager

Les Layout Managers

- Gestionnaires de mise en page (placement/disposition)
  → pour positionner les composants dans la fenêtre
  - BorderLayout
  - BoxLayout
  - CardLayout
  - FlowLayout
  - GridBagLayout
  - GridLayout
  - SpringLayout

BorderLayout

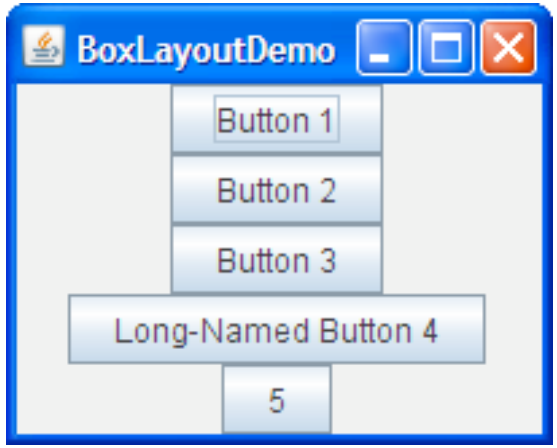
- Ce gestionnaire possède 5 zones possibles
  - PAGE_START, PAGE_END, LINE_START, LINE_END, CENTER
- C'est le gestionnaire par défaut pour un content pane

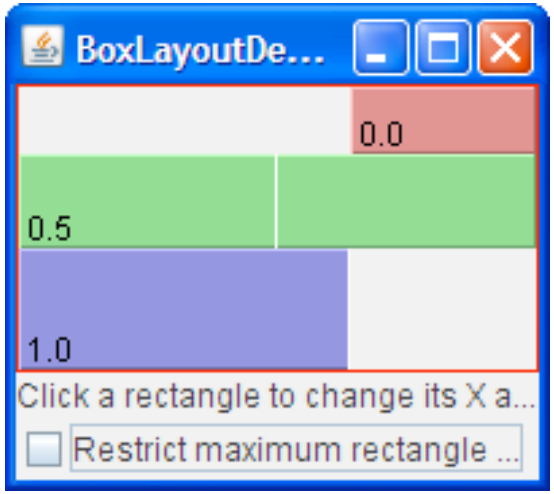


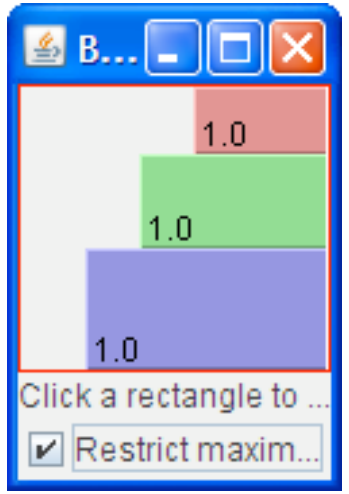
http://docs.oracle.com

BoxLayout

- Ce gestionnaire affiche les composants en ligne ou en colonne selon leur taille préférée (*preferredSize*)
- L'ordre d'apparition des composants correspond à l'ordre d'ajout dans le conteneur
- Tous les composants n'ont pas obligatoirement la même taille et la même hauteur



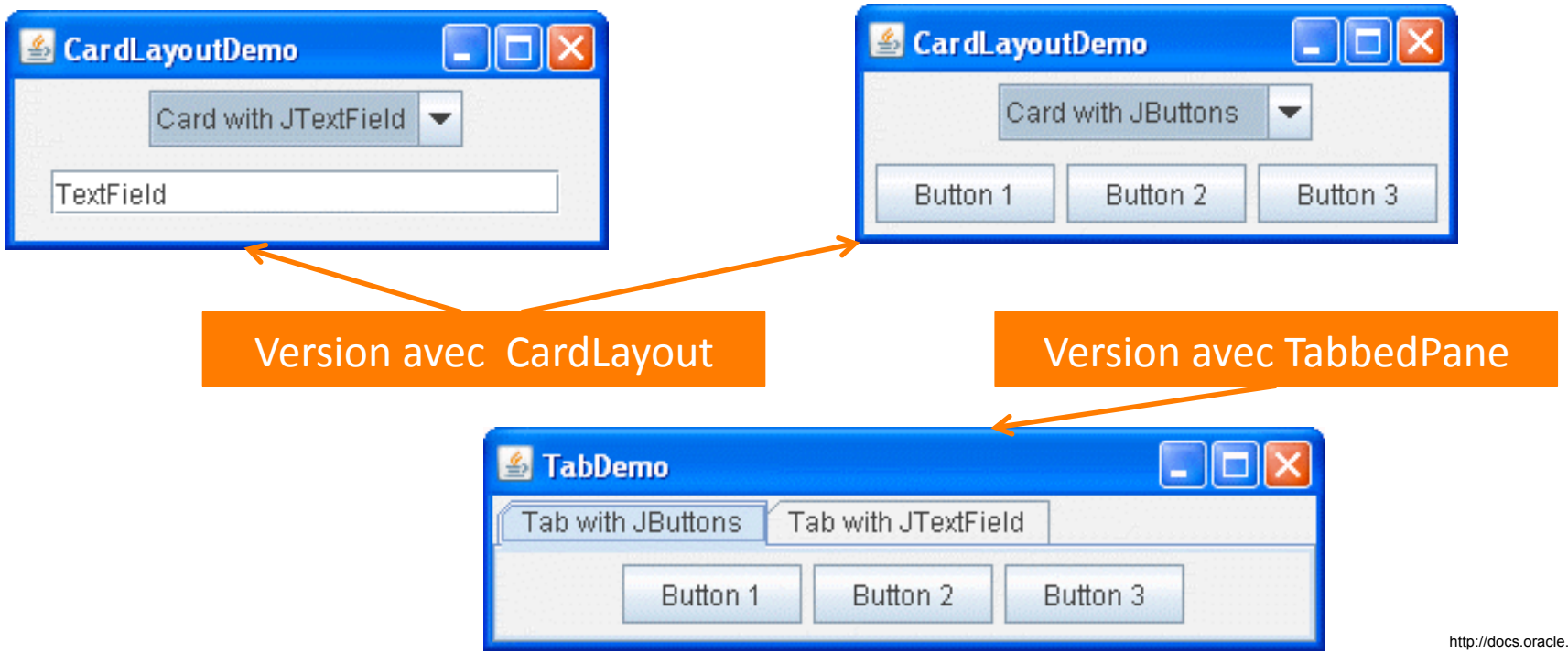




http://docs.oracle.com

CardLayout

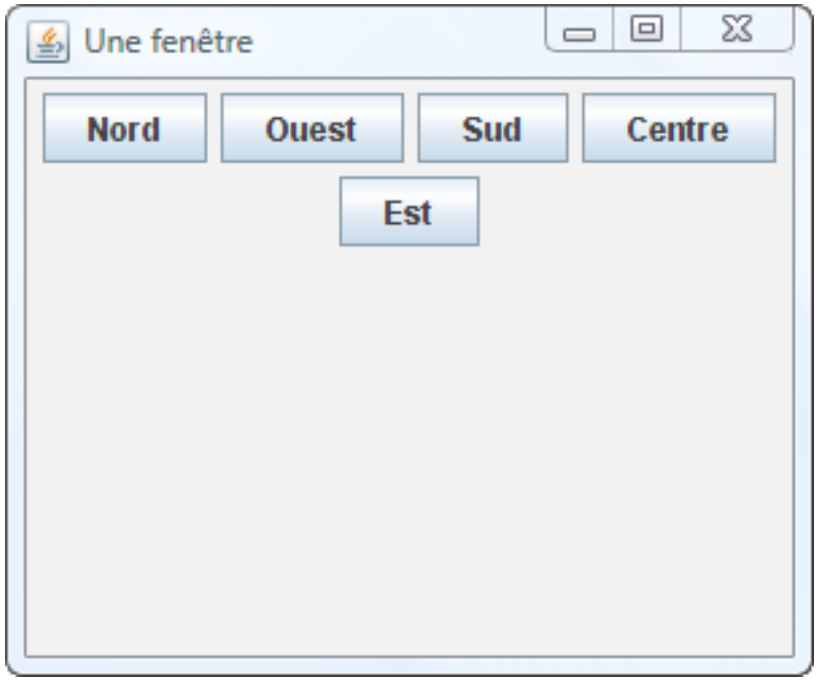
- Ce gestionnaire gère deux ou plusieurs composants (généralement des JPanel) qui partage le même espace d'affichage
- Pour choisir le panneau à afficher : choix via une combox par exemple



http://docs.oracle.com

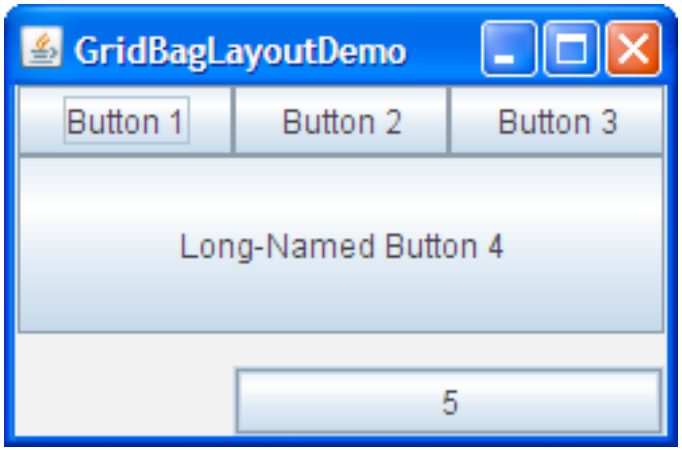
FLowLayout

- Ce gestionnaire essaie de placer les composants selon leur taille préférée (*preferredSize*) de gauche à droite et de haut en bas
- C'est le gestionnaire par défaut d'un JPanel



GridBagLayout

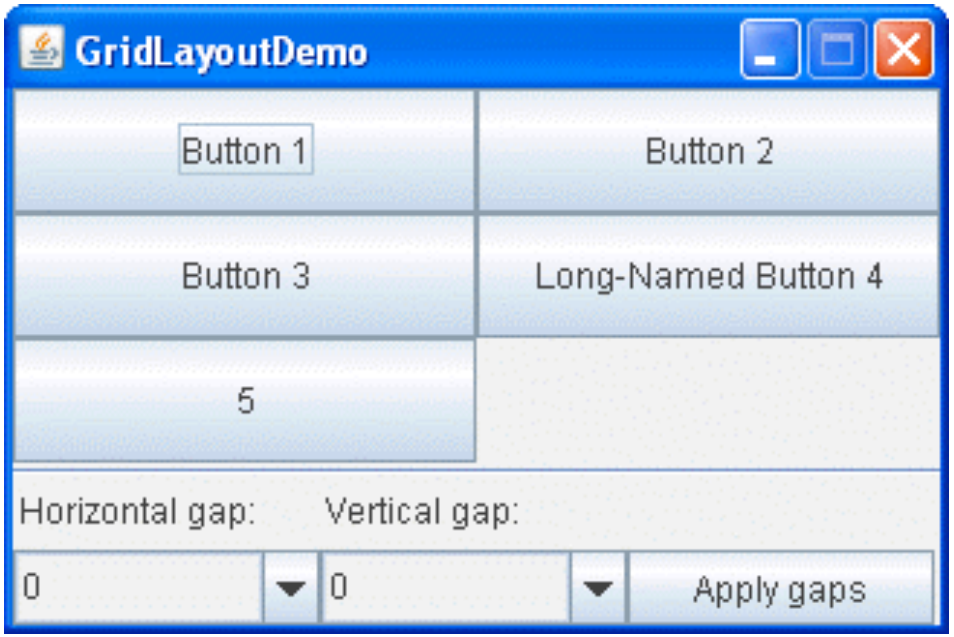
- Ce gestionnaire est l'un des plus flexible (et complexe)
- Les composants sont placés dans une grille (ligne et colonne)
  - certains composants peuvent s'étendre sur plusieurs lignes ou colonnes
  - Toutes les lignes (resp. les colonnes) n'ont pas la même hauteur (resp. largeur) .



http://docs.oracle.com

GridLayout

- Ce gestionnaire place les composants en ligne dans des cellules
  - Chaque cellule a la même taille
  - Chaque composant prend toute la place disponible dans la cellule



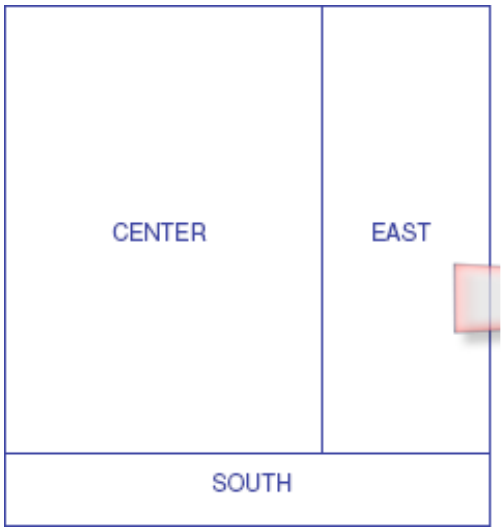
http://docs.oracle.com

Les Layouts

- Possibilité de grouper les layouts verticalement et horizontalement de manière séparée
  - Pour cela utiliser le gestionnaire GroupLayout
  - La mise en page est définie de manière indépendante pour chaque dimension
- Possibilité de personnaliser un layout manager
  - Créer soi-même son layout
  - Créer une classe qui implémente l'interface LayoutManager ou LayoutManager2
- Possibilité de combiner les layouts entre eux

http://docs.oracle.com

Combiner les panneaux



Petit aparté sur les classes internes

Les classes internes

- Une classe est dite interne lorsque sa définition est située à l'intérieure d'une autre classe (dite classe externe)
  - Un objet de la classe interne est toujours associé, au moment de son instanciation, à un objet de la classe externe
  - La classe interne a accès aux champs et méthodes (même privés !) de la classe externe

```
public class ClasseExterne
{
    private int compteur = 0;

    class ClasseInterne
    {
        private int index = 0;
        public ClasseInterne()
        {
            compteur++;
        }
    }
}
```

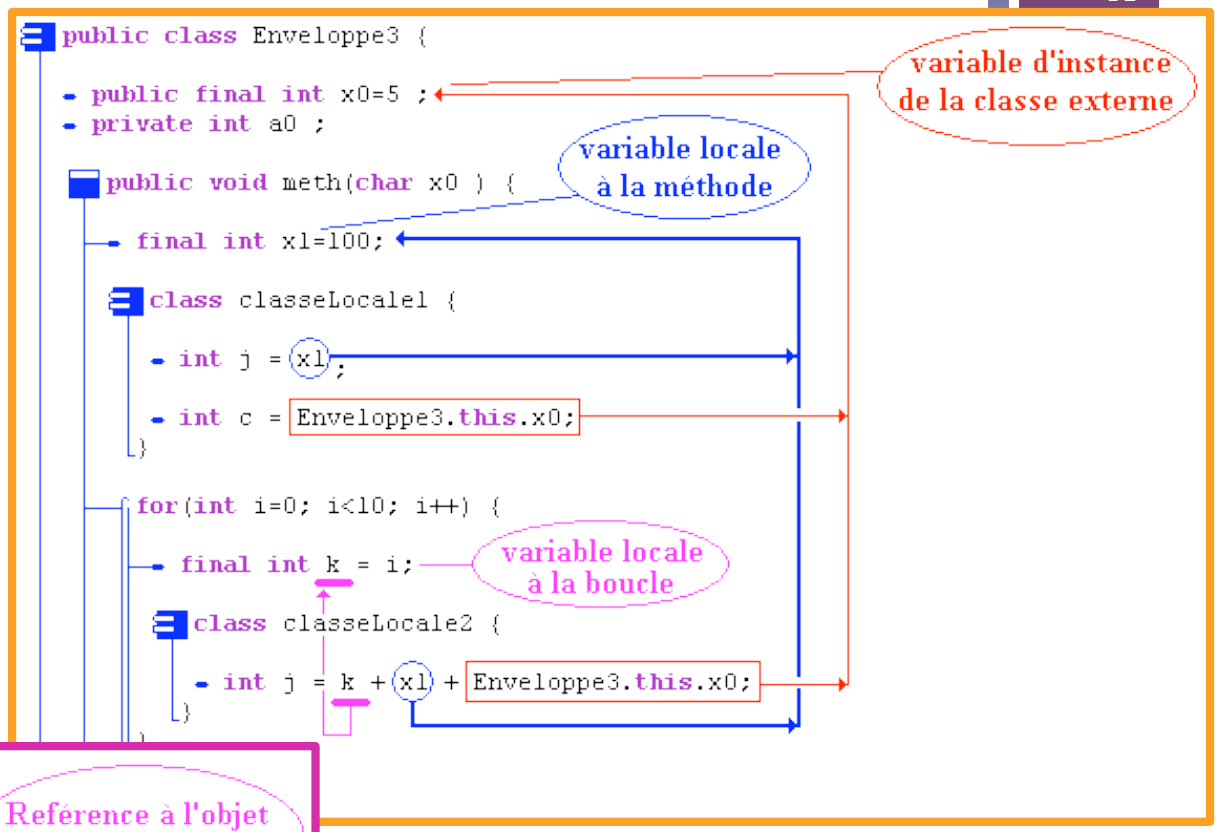
`ClasseExterne.this.compteur++`

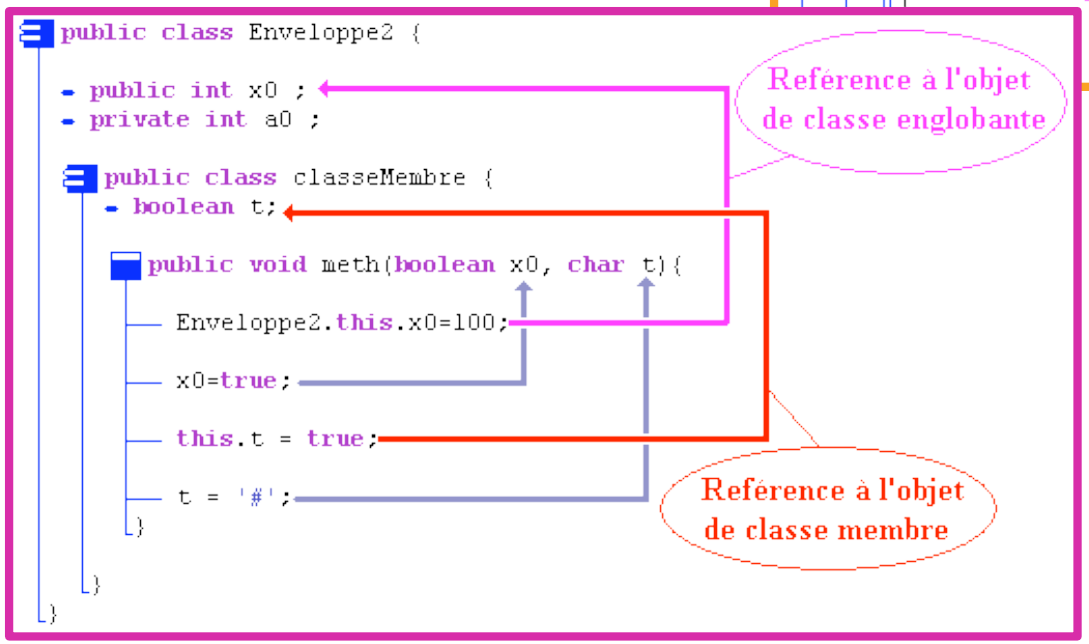
Les classes internes

- Une classe interne peut être déclarée
  - de manière globale dans la classe externe
    - → L'ensemble des membres de la classe peut y accéder
    - → La classe interne est considérée comme un membre de la classe externe
      - classe interne considérée comme attribut → on peut utiliser les modificateurs des membres d'une classe
      - Classe interne considérée comme locale → elle n'est utilisable que dans le code qui l'a défini (pas de modificateurs)
  - de manière locale à une méthode
    - → L'accès à la classe interne se fait uniquement via la méthode

Une classe interne statique ne peut accéder qu'aux membres statiques de sa classe externe

Les classes internes





Les classes anonymes

- Une classe est dite anonyme si elle n'a pas de nom
  - Elle peut dériver d'une autre classe
  - Elle peut implémenter une interface
- Une classe anonyme ne peut pas contenir de constructeurs
  - Le constructeur appelé est celui qui correspond à l'instruction d'instanciation

```
// Implémentation d'une interface :
Runnable task = new Runnable() {
        public void run() {
                System.out.println("Hello World");
        }
};

// Héritage d'une classe :
Thread thread = new Thread() {
        public void run() {
                System.out.println("Hello World");
        }
};

thread.start();
new Thread(task).start();
```

Elle ne peut être instanciée qu'une seule fois

Les événements

Les événements

- Une interface graphique Swing est dite *event-driven* (basée sur des événements)
- Un événement affectant une interface graphique peut provenir
  - de l'utilisateur

    Chaque action de utilisateur (clic sur un bouton, souris, clavier, …) génère la création d'un objet événement
  - de l'application
  - du système
- Tous les événements sont stockées dans une queue d'événements

  → Events Queue (structure de données similaire à une FIFO)

GUI et processus

Petit rappel

- Lors de l'exécution d'un programme Java, plusieurs sous-processus (threads) sont créés
  - le processus légers lié à la méthode *main*
  - le processus légers *Garbage Collector*
  - … et (peut-être) d'autres threads (cf. cours précédent)

Les threads peuvent interagir entre eux, etc..

GUI et processus

- Création d'une interface graphique avec Swing
  - → Création d'un nouveau thread : *Event Dispatch Thread* (appelé aussi thread Swing ou thread de gestion des événements)

Un seul processus a la droit de modifier l'interface graphique : c'est *Event Dispatch Thread*

Event Dispatch Thread est chargé de récupérer et traiter les événements de la queue d'événements (*Events Queue*)

```
public static void main(String[] args) {
    SwingUtilities.invokeLater(new Runnable() {
        public void run() {
            creerEtMontrerMaFenetre();
        }
    });
}
```

Poste la tâche dans le thread Swing

Revenons aux événements

- Un événement peut-être
  - de haut niveau
    - action sur un bouton, choix dans un menu
  - de bas niveau
    - clic dans une fenêtre, souris qui bouge
- Lorsqu'un utilisateur fait une action qui cible/affecte un composant un événement est crée
  - on dit que ce composant est la source d'un événement

→ Un événement dépend du composant qui l'a généré

Chaque événement est représenté par un objet qui donne des informations sur l'événement et qui identifie la source de cet événement

Les classes d'événements

- Tout événement qui se produit dans une interface graphique est de type *xxxEvent* en fonction de l'organe d'entrée de cet événement
  - *MouseEvent*
    - événements (de bas niveau) provoqués par la souris
  - *ActionEvent*
    - événements (de haut niveau) tels que : clic sur un bouton, choix dans un menu, etc.,
  - *KeyEvent*
    - événements (de bas niveau) traduisant des actions sur le clavier
  - *FocusEvent*
  - *WindowEvent*
  - …

Traitement d'un événement

- Un événement représente généralement une requête à laquelle il faut répondre
  - → Traiter un événement c'est répondre à cette requête
- A chaque classe d'événements est associé une (ou plusieurs) interface qu'on appelle un **listener** (auditeur)
  - → Cette interface indique « les choses » que devront satisfaire les objets prétendant pouvoir traiter l'événement lorsqu'il se produit

Traitement d'un événement

- En fonction des événements qu'il traite, un auditeur doit implémenter une interface particulière dérivée de *EventListener*
- Par exemple, *MouseListener* est l'une des interfaces associées à la classe *MouseEvent*

```
public interface MouseListener extends EventListener {
    void mouseClicked(MouseEvent e);
    void mouseEntered(MouseEvent e);
    void mouseExited(MouseEvent e);
    void mousePressed(MouseEvent e);
    void mouseReleased(MouseEvent e);
}
```

- Un objet qui prétend traiter les événements *MouseEvent* doit implémenter cette interface et définir ces 5 méthodes

Interception d'un événement

- Chaque composant maintient une liste de *listeners* pour chaque classe d'événements dont il peut être la source
  - Les *listeners* doivent s'enregistrer auprès de la source via la méthode

addXXXListener(XXXListener auditeur)

auditeur correspond au *listener* chargé de traiter l'événement

- Pour détecter un événement à partir d'un composant particulier, il faut savoir quels *listeners* utiliser

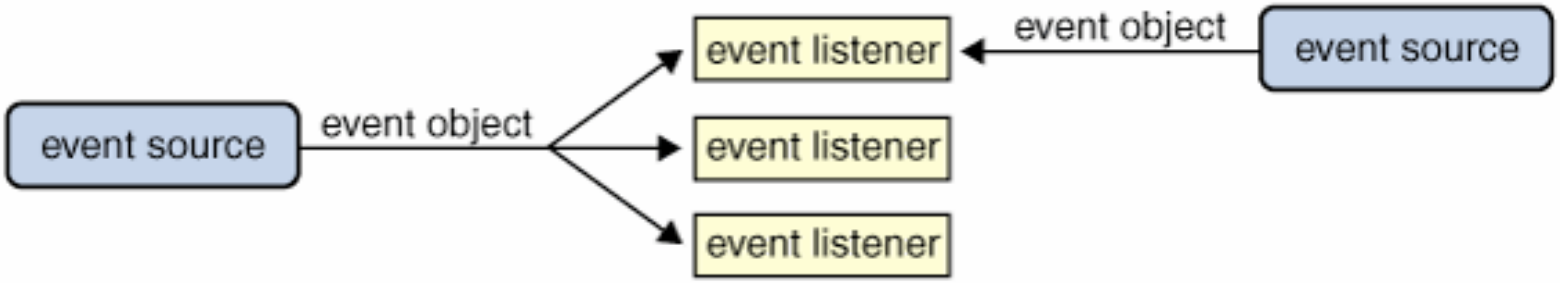
Les *listeners*

- Listeners supportés par par tous les composants
  - Component Listener
    - écoute pour les changements dans la visibilité, la position ou la taille d'un composant
  - Focus Listener
    - écoute si le composant acquiert ou perd le focus clavier
  - Key Listener
    - écoute les (pressions) touches du clavier
  - Mouse Listener
  - Mouse Motion Listener
  - Mouse Wheel Listener
  - Hierarchy Listener
  - Hierarchy Bounds Listener

Component	Action Listener	Caret Listener	Change Listener	Document Listener, Undoable Edit Listener	Item Listener	List Selection Listener	Window Listener	Other Types of Listeners
button	✔		✔		✔			
check box	✔		✔		✔			
color chooser			✔					
combo box	✔				✔			
dialog							✔	
editor pane		✔		✔				hyperlink
file chooser	✔							
formatted text field	✔	✔		✔				
frame							✔	
internal frame								internal frame
list						✔		list data
menu								menu
menu item	✔		✔		✔			menu key menu drag mouse
option pane								
password field	✔	✔		✔				
popup menu								popup menu
progress bar			✔					
radio button	✔		✔		✔			
slider			✔					
spinner			✔					
tabbed pane			✔					
table						✔		table model table column model cell editor
text area		✔		✔				
text field	✔	✔		✔				
text pane		✔		✔				hyperlink
toggle button	✔		✔		✔			
tree								tree expansion tree will expand tree model tree selection
viewport (used by scrollpane)			✔					

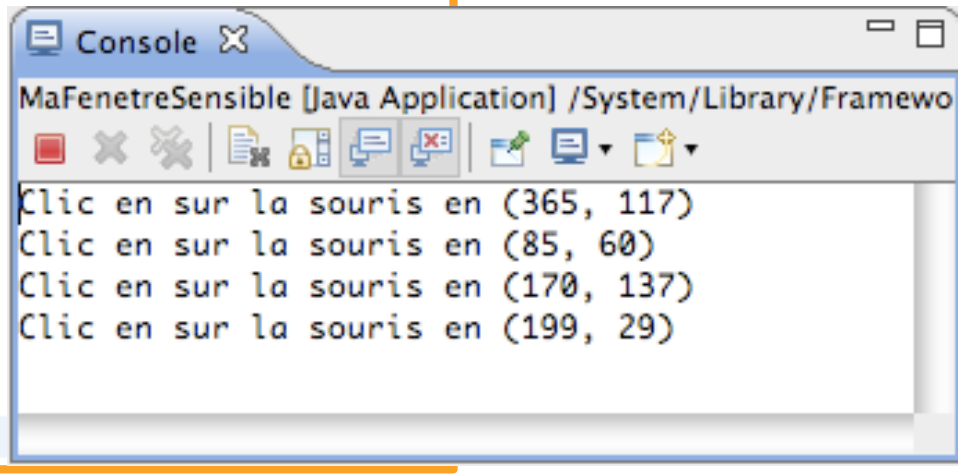
Remarques

- Un programme peut crée
  - un *listener* par source d'événements
  - un unique *listener* pour tous les événements de toutes les sources
  - plus d'un *listener* pour un unique type d'événements à partir d'une unique source



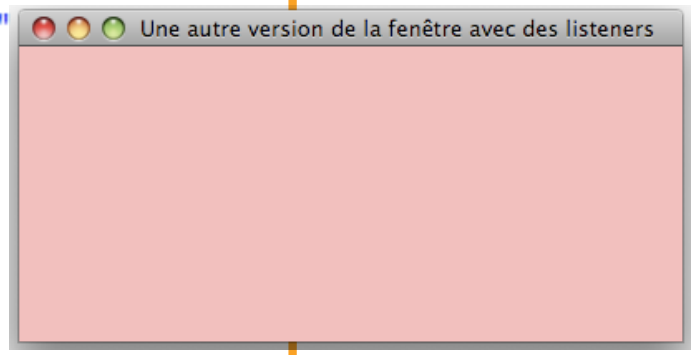
- Plusieurs *listeners* peuvent s'enregistrer pour être informé des événements d'un type particulier d'une source particulière

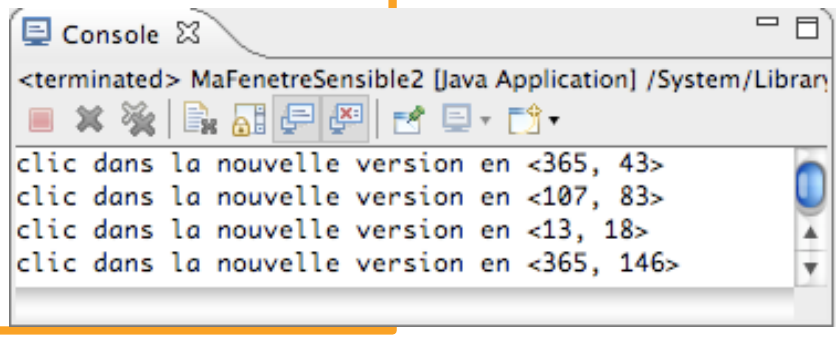
```
import java.awt.event.*;
import javax.swing.*;
public class MaFenetreSensible extends JFrame{
    public static void creerEtMontrerMaFenetre(){new MaFenetreSensible() ;}

    public MaFenetreSensible(){
        this.setTitle("Ma fenêtre avec des listeners");
        this.setSize(400, 200);
        JPanel monPanel = new JPanel();
        monPanel.setBackground(Color.PINK);
        monPanel.addMouseListener(new MonListenerSouris());
        this.getContentPane().add(monPanel);
        this.setDefaultCloseOperation(EXIT_ON_CLOSE);
        this.setVisible(true);
    }

    public class MonListenerSouris implements MouseListener {
        public void mousePressed(MouseEvent e) {
            System.out.println("Clic en sur la souris en ("
                    + e.getX() + ", " + e.getY() + ")");
        }
        public void mouseReleased(MouseEvent e) {}
        public void mouseClicked(MouseEvent e) {}
        public void mouseEntered(MouseEvent e) {}
        public void mouseExited(MouseEvent e) {}
    }

    public static void main(String args[]) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                creerEtMontrerMaFenetre();
            }
        });
    }
}
```



Une autre version

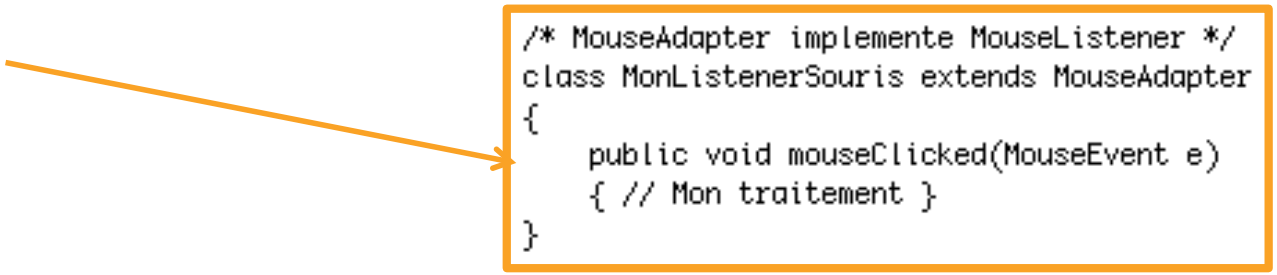

```
public class MaFenetreSensible2 extends JFrame implements MouseListener{
    public static void creerEtMontrerMaFenetre(){new MaFenetreSensible2() ;}
    public MaFenetreSensible2(){
        this.setTitle("Une autre version de la fenêtre avec des listeners");
        this.setSize(400, 200);
        JPanel monPanel = new JPanel();
        monPanel.setBackground(Color.PINK);
        monPanel.addMouseListener(this);
        this.getContentPane().add(monPanel);
        this.setDefaultCloseOperation(EXIT_ON_CLOSE);
        this.setVisible(true);
    }

    public void mouseClicked(MouseEvent e) {
        System.out.println("clic dans la nouvelle version en <"
                + e.getX() + ", " + e.getY() + ">" );
    }
    public void mousePressed(MouseEvent e) { }
    public void mouseReleased(MouseEvent e) { }
    public void mouseEntered(MouseEvent e) { }
    public void mouseExited(MouseEvent e) { }

    public static void main(String args[]) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                creerEtMontrerMaFenetre();
            }
        });
    }
}
```





Les adaptateurs

- Les interfaces *XXXListener* peuvent fournir et déclarer plus de méthodes que celles dont on a besoin réellement (*cf.* exemple précédent)
  - implémentation de l'interface induit la déclaration de nombreuses fonctions vides
- Solution : les adaptateurs (Adapters)
- Pour les interfaces possédant plusieurs méthodes, Java fournit des classes (non abstraites): les adaptateurs définissant une version par défaut de chaque méthode déclarée dans l'interface *XXXListener*
  - Il suffit d'étendre ces adaptateurs et de redéfinir uniquement les méthodes utiles



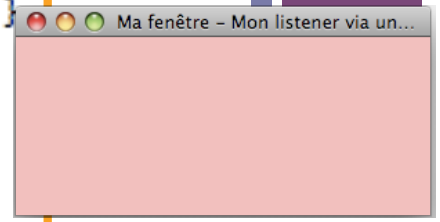
```
/* MouseAdapter implemente MouseListener */
class MonListenerSouris extends MouseAdapter
{
    public void mouseClicked(MouseEvent e)
    { // Mon traitement }
}
```

Listener Interface	Adapter Class	Listener Methods
ActionListener	*none*	actionPerformed(ActionEvent)
AncestorListener	*none*	ancestorAdded(AncestorEvent) ancestorMoved(AncestorEvent) ancestorRemoved(AncestorEvent)
CaretListener	*none*	caretUpdate(CaretEvent)
CellEditorListener	*none*	editingStopped(ChangeEvent) editingCanceled(ChangeEvent)
ChangeListener	*none*	stateChanged(ChangeEvent)
ComponentListener	ComponentAdapter	componentHidden(ComponentEvent) componentMoved(ComponentEvent) componentResized(ComponentEvent) componentShown(ComponentEvent)
ContainerListener	ContainerAdapter	componentAdded(ContainerEvent) componentRemoved(ContainerEvent)
DocumentListener	*none*	changedUpdate(DocumentEvent) insertUpdate(DocumentEvent) removeUpdate(DocumentEvent)
ExceptionListener	*none*	exceptionThrown(Exception)
FocusListener	FocusAdapter	focusGained(FocusEvent) focusLost(FocusEvent)
HierarchyBoundsListener	HierarchyBoundsAdapter	ancestorMoved(HierarchyEvent) ancestorResized(HierarchyEvent)
HierarchyListener	*none*	hierarchyChanged(HierarchyEvent)
HyperlinkListener	*none*	hyperlinkUpdate(HyperlinkEvent)
InputMethodListener	*none*	caretPositionChanged(InputMethodEvent) inputMethodTextChanged(InputMethodEvent)
InternalFrameListener	InternalFrameAdapter	internalFrameActivated(InternalFrameEvent) internalFrameClosed(InternalFrameEvent) internalFrameClosing(InternalFrameEvent) internalFrameDeactivated(InternalFrameEvent) internalFrameDeiconified(InternalFrameEvent) internalFrameIconified(InternalFrameEvent) internalFrameOpened(InternalFrameEvent)
ItemListener	*none*	itemStateChanged(ItemEvent)
KeyListener	KeyAdapter	keyPressed(KeyEvent) keyReleased(KeyEvent) keyTyped(KeyEvent)
ListDataListener	*none*	contentsChanged(ListDataEvent) intervalAdded(ListDataEvent) intervalRemoved(ListDataEvent)
ListSelectionListener	*none*	valueChanged(ListSelectionEvent)
MenuDragMouseListener	*none*	menuDragMouseDragged(MenuDragMouseEvent) menuDragMouseEntered(MenuDragMouseEvent) menuDragMouseExited(MenuDragMouseEvent) menuDragMouseReleased(MenuDragMouseEvent)
MenuKeyListener	*none*	menuKeyPressed(MenuKeyEvent) menuKeyReleased(MenuKeyEvent) menuKeyTyped(MenuKeyEvent)

Listener Interface	Adapter Class	Listener Methods
MenuListener	*none*	menuCanceled(MenuEvent) menuDeselected(MenuEvent) menuSelected(MenuEvent)
MouseInputListener (extends MouseListener and MouseMotionListener	MouseInputAdapter MouseAdapter	mouseClicked(MouseEvent) mouseEntered(MouseEvent) mouseExited(MouseEvent) mousePressed(MouseEvent) mouseReleased(MouseEvent) mouseDragged(MouseEvent) mouseMoved(MouseEvent) MouseAdapter(MouseEvent)
MouseListener	MouseAdapter, MouseInputAdapter	mouseClicked(MouseEvent) mouseEntered(MouseEvent) mouseExited(MouseEvent) mousePressed(MouseEvent) mouseReleased(MouseEvent)
MouseMotionListener	MouseMotionAdapter, MouseInputAdapter	mouseDragged(MouseEvent) mouseMoved(MouseEvent)
MouseWheelListener	MouseAdapter	mouseWheelMoved(MouseWheelEvent) MouseAdapter<MouseEvent>
PopupMenuListener	*none*	popupMenuCanceled(PopupMenuEvent) popupMenuWillBecomeInvisible(PopupMenuEvent) popupMenuWillBecomeVisible(PopupMenuEvent)
PropertyChangeListener	*none*	propertyChange(PropertyChangeEvent)
TableColumnModelListener	*none*	columnAdded(TableColumnModelEvent) columnMoved(TableColumnModelEvent) columnRemoved(TableColumnModelEvent) columnMarginChanged(ChangeEvent) columnSelectionChanged(ListSelectionEvent)
TableModelListener	*none*	tableChanged(TableModelEvent)
TreeExpansionListener	*none*	treeCollapsed(TreeExpansionEvent) treeExpanded(TreeExpansionEvent)
TreeModelListener	*none*	treeNodesChanged(TreeModelEvent) treeNodesInserted(TreeModelEvent) treeNodesRemoved(TreeModelEvent) treeStructureChanged(TreeModelEvent)
TreeSelectionListener	*none*	valueChanged(TreeSelectionEvent)
TreeWillExpandListener	*none*	treeWillCollapse(TreeExpansionEvent) treeWillExpand(TreeExpansionEvent)
UndoableEditListener	*none*	undoableEditHappened(UndoableEditEvent)
VetoableChangeListener	*none*	vetoableChange(PropertyChangeEvent)
WindowFocusListener	WindowAdapter	windowGainedFocus(WindowEvent) windowLostFocus(WindowEvent)
WindowListener	WindowAdapter	windowActivated(WindowEvent) windowClosed(WindowEvent) windowClosing(WindowEvent) windowDeactivated(WindowEvent) windowDeiconified(WindowEvent) windowIconified(WindowEvent) windowOpened(WindowEvent)
WindowStateListener	WindowAdapter	windowStateChanged(WindowEvent)

Classes anonymes et adaptateurs

```
public class MaFenetreSensible4 extends JFrame{
    public static void creerEtMontrerMaFenetre(){new MaFenetreSensible4() ;}
    public MaFenetreSensible4(){
        this.setTitle("Ma fenêtre - Mon listener via une classe anonyme");
        this.setSize(300, 150);
        JPanel monPanel = new JPanel();
        monPanel.setBackground(Color.PINK);
        monPanel.addMouseListener(new MouseAdapter() {
            public void mousePressed(MouseEvent e) {
                System.out.println("clic dans la nouvelle version en <"
                        + e.getX() + ", " + e.getY() + ">" );
            }
        });

        this.getContentPane().add(monPanel);
        this.setDefaultCloseOperation(EXIT_ON_CLOSE);
        this.setVisible(true);
    }

    public static void main(String args[]) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                creerEtMontrerMaFenetre();
            }
        });
    }
}
```

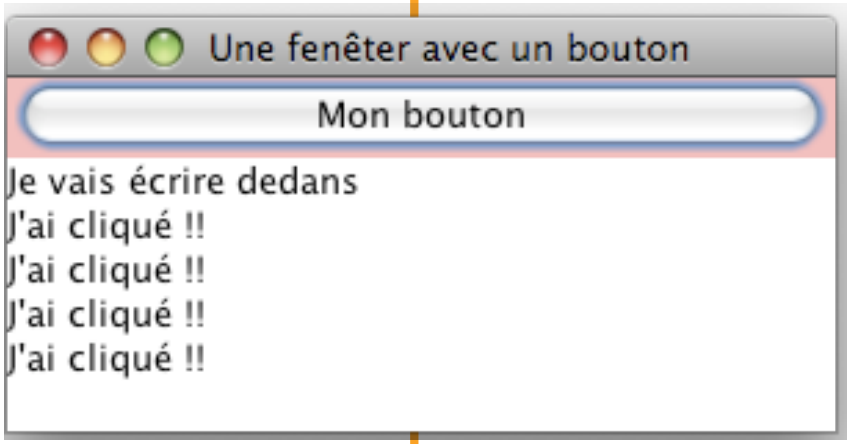



```
MouseListener monEcouteur = new MouseAdapter() {
    public void mousePressed(MouseEvent e) {
        System.out.println("clic dans la nouvelle version en <"
                + e.getX() + ", " + e.getY() + ">" );
    }
};
monPanel.addMouseListener(monEcouteur);
```

+ Un autre exemple


```
public class MaFenetreSensible3 extends JFrame {
    public static void creerEtMontrerMaFenetre(){new MaFenetreSensible3() ;}
    public MaFenetreSensible3(){
        this.setTitle("Une fenêter avec un bouton");
        this.setSize(300, 150);
        JPanel monPanel = new JPanel();
        monPanel.setBackground(Color.PINK);
        monPanel.setLayout(new BorderLayout());
        JButton monBouton = new JButton("Mon bouton") ;
        final JTextArea monTexte = new JTextArea("Je vais écrire dedans");
        monPanel.add(monBouton, BorderLayout.NORTH);
        monPanel.add(monTexte, BorderLayout.CENTER);
        monBouton.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e) {
                monTexte.append("\nJ'ai cliqué !!");
                }
        });
        this.getContentPane().add(monPanel);
        this.setDefaultCloseOperation(EXIT_ON_CLOSE);
        this.setVisible(true);
    }

    public static void main(String args[]) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                creerEtMontrerMaFenetre();
            }
        });
    }
}
```

Classe anonyme qui implémente une interface



MVC

Le modèle MVC

- Modèle de conception : Modèle – Vue – Contrôleur
- Principe : séparation des données, de la présentation et des traitements
- Trois couches :
  - Le modèle (de données) définit les données à manipuler
  - La vue affiche les données fournit par le modèle

    Plusieurs vues possibles pour un même modèle de données
  - Le contrôleur sert de pont entre la vue et le modèle
    - Gestion des actions provenant de la vue, des mises à jour, …

http://blog.jaycreation.net

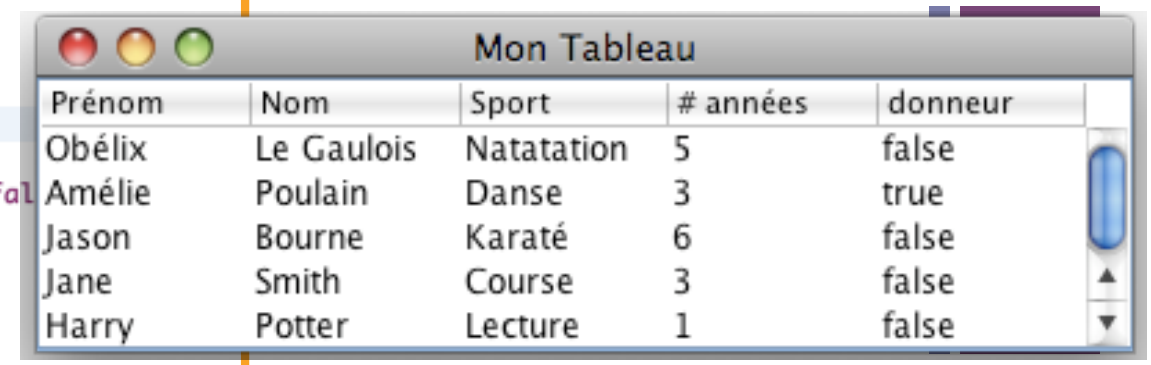
Le modèle MVC

- L'architecture des composants Swing s'inspire du modèle MVC
- A chaque composant est associé
  - Un modèle
  - Un couple Vue-Contôleur
- Par exemple, ButtonModel est le modèle associé à un bouton
- Certains composants peuvent avoir plusieurs modèles
  - Par exemple, une liste (JList)
    - ListModel pour s'occuper du contenu de la liste
    - ListSelectionModel poursuivre la sélection courante de la liste

Pourquoi utiliser des modèles ?

- La plupart du temps les modèles ne sont pas utiliser
  - un programme qui utilise un bouton interagit directement avec l'objet `JButton` sans passer par l'objet `ButtonModel`
- Toutefois, les modèles
  - donnent une grande flexibilité pour déterminer la manière dont les données sont stockées et récupérées
  - évitent les copies de données entre les structures de données d'un programme et ceux des composants Swing
  - Permettent la propagation automatique des modifications à tous les auditeurs intéressés
  - …

```java
public class DemoTable1 extends JPanel {
    public DemoTable1() {
        super(new GridLayout(1,0));
        String[] colonnes = {"Prénom","Nom",
                "Sport","# années","donneur"};
        Object[][] donnees = {
        {"Obélix", "Le Gaulois","Natatation", 5, fal
        {"Amélie", "Poulain","Danse", 3, true},
        {"Jason", "Bourne","Karaté", 6, false},
        {"Jane", "Smith","Course", 3, false},
        {"Harry", "Potter", "Lecture", 1, false}
        };

        final JTable table = new JTable(donnees, colonnes);
        table.addMouseListener(new MouseAdapter() {
                public void mouseClicked(MouseEvent e) {
                    affichage(table);
                }
           });
        JScrollPane scrollPane = new JScrollPane(table);
        add(scrollPane);
    }

    private void affichage(JTable table) {
        int numRows = table.getRowCount();
        int numCols = table.getColumnCount();
        javax.swing.table.TableModel model = table.getModel();

        System.out.println("Valeur des données");
        for (int i=0; i < numRows; i++) {
            System.out.print("    ligne " + i + ":");
            for (int j=0; j < numCols; j++) {
                System.out.print("   " + model.getValueAt(i, j));
            }
            System.out.println();
        }
        System.out.println("--------------------------");
    }
```




```java
rivate static void creerMaFenetre() {
    JFrame frame = new JFrame("Mon Tableau");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    DemoTable1 monConteneur = new DemoTable1();
    frame.setContentPane(monConteneur);
    frame.pack();
    frame.setVisible(true);


ublic static void main(String[] args) {
    SwingUtilities.invokeLater(new Runnable() {
        public void run() {
            creerMaFenetre();
        }
    });
```

Créer son modèle

- Chaque objet JTable utilise un objet *table model* pour gérer les données actuelles du tableau
  - Un objet *table model* doit implémenter l'interface `TableModel`
- A la création d'un tableau si aucun modèle n'est fourni, JTable crée automatiquement une instance de `DefaultTableModel`

Mon Tableau

Prénom	Nom	Sport	# années	donneur
Obélix	Le Gaulois	Natatation	5	false
Amélie	Poulain	Danse	3	true
Jason	Bourne	Karaté	6	false
Jane	Smith	Course	3	false
Harry	Potter	Lecture	1	false

Les modèles

Correspondance entre composant et modèle dans Swing

Component	Model Interface	Model Type
JButton	ButtonModel	GUI
JToggleButton	ButtonModel	GUI/data
JCheckBox	ButtonModel	GUI/data
JRadioButton	ButtonModel	GUI/data
JMenu	ButtonModel	GUI
JMenuItem	ButtonModel	GUI
JCheckBoxMenuItem	ButtonModel	GUI/data
JRadioButtonMenuItem	ButtonModel	GUI/data
JComboBox	ComboBoxModel	data
JProgressBar	BoundedRangeModel	GUI/data
JScrollBar	BoundedRangeModel	GUI/data
JSlider	BoundedRangeModel	GUI/data
JTabbedPane	SingleSelectionModel	GUI
JList	ListModel	data
JList	ListSelectionModel	GUI
JTable	TableModel	data
JTable	TableColumnModel	GUI
JTree	TreeModel	data
JTree	TreeSelectionModel	GUI
JEditorPane	Document	data
JTextPane	Document	data
JTextArea	Document	data
JTextField	Document	data
JPasswordField	Document	data

Créer son modèle

- Le constructeur utilisé dans notre exemple

```
new AbstractTableModel() {
    public String getColumnName(int col) {
        return columnNames[col].toString();
    }
    public int getRowCount() { return rowData.length; }
    public int getColumnCount() { return columnNames.length; }
    public Object getValueAt(int row, int col) {
        return rowData[row][col];
    }
    public boolean isCellEditable(int row, int col)
        { return true; }
    public void setValueAt(Object value, int row, int col) {
        rowData[row][col] = value;
        fireTableCellUpdated(row, col);
    }
}
```

Créer son modèle

- Créer son propre modèle via l'implémentation d'une classe qui hérite de `AbstractTableModel`
- Le modèle peut stocker ses données dans un tableau, un vecteur, une table de hachage
  - les données peuvent provenir d'une source externe (ex : une base de données)
  - les données peuvent être générées à l'exécution

Créer son modèle


```
class MyTableModel extends AbstractTableModel {
    private String[] columnNames = ...//same as before...
    private Object[][] data = ...//same as before...

    public int getColumnCount() {
        return columnNames.length;
    }

    public int getRowCount() {
        return data.length;
    }

    public String getColumnName(int col) {
        return columnNames[col];
    }

    public Object getValueAt(int row, int col) {
        return data[row][col];
    }

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

    /*
     * Don't need to implement this method unless your table's
     * editable.
     */
    public boolean isCellEditable(int row, int col) {
        //Note that the data/cell address is constant,
        //no matter where the cell appears onscreen.
        if (col < 2) {
            return false;
        } else {
            return true;
        }
    }

    /*
     * Don't need to implement this method unless your table's
     * data can change.
     */
    public void setValueAt(Object value, int row, int col) {
        data[row][col] = value;
        fireTableCellUpdated(row, col);
    }
    ...
}
```

```
...
JTable table = new JTable(new MyTableModel());
...
```

Method	Change
`fireTableCellUpdated`	Update of specified cell.
`fireTableRowsUpdated`	Update of specified rows
`fireTableDataChanged`	Update of entire table (data only).
`fireTableRowsInserted`	New rows inserted.
`fireTableRowsDeleted`	Existing rows Deleted
`fireTableStructureChanged`	Invalidate entire table, both data and structure.

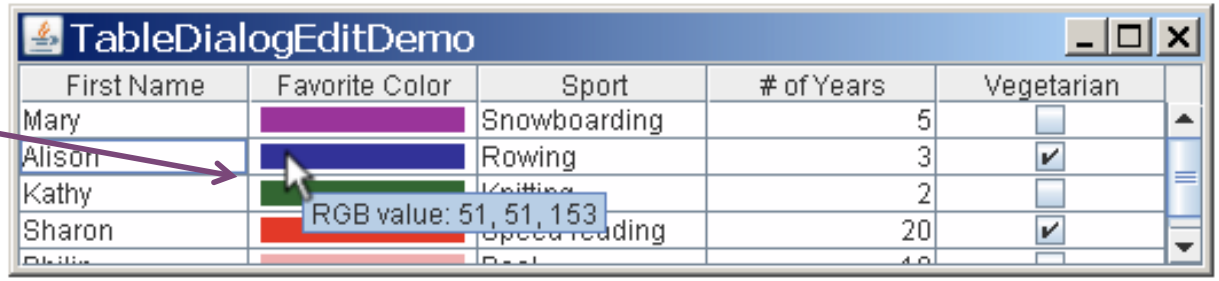
En cas de modification des données

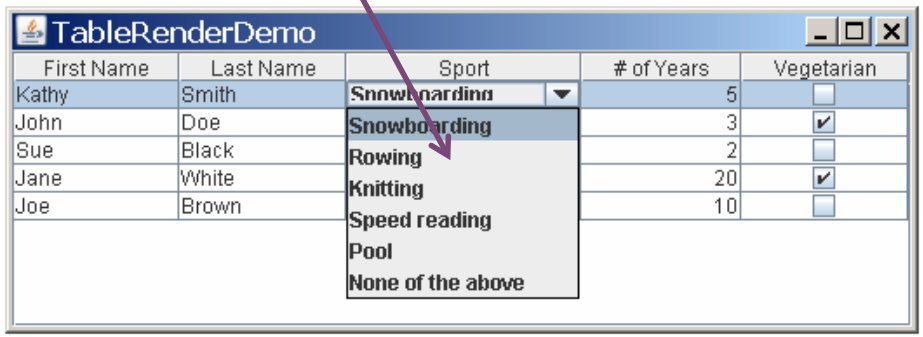
N'oubliez pas les *listeners*

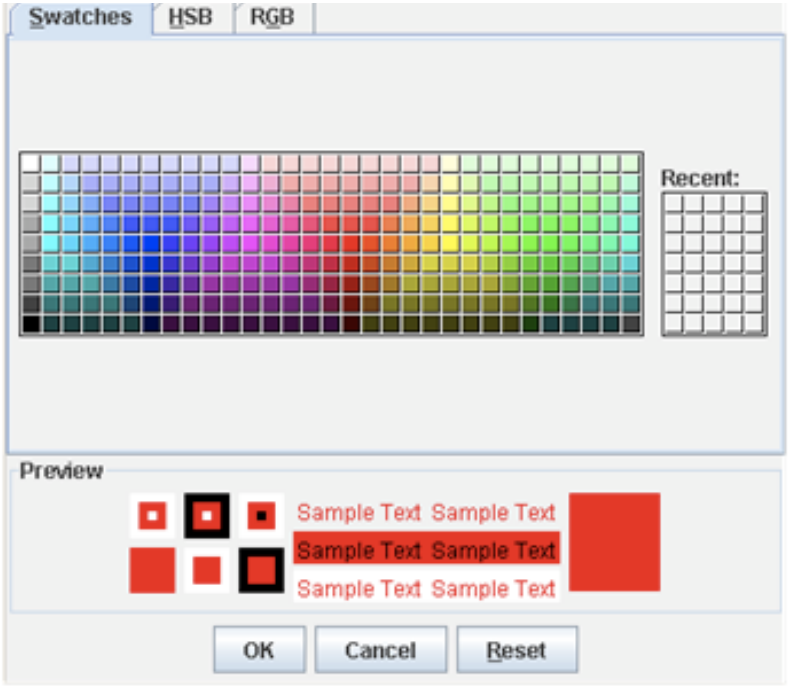
```
import javax.swing.event.*;
import javax.swing.table.TableModel;

public class SimpleTableDemo ... implements TableModelListener {
    ...
    public SimpleTableDemo() {
        ...
        table.getModel().addTableModelListener(this);
        ...
    }

    public void tableChanged(TableModelEvent e) {
        int row = e.getFirstRow();
        int column = e.getColumn();
        TableModel model = (TableModel)e.getSource();
        String columnName = model.getColumnName(column);
        Object data = model.getValueAt(row, column);

        ...// Do something with the data...
    }
    ...
}
```

… et encore plus

Utiliser les renderers pour le rendu des données du tableau







Possibilité de trier et filtrer

```
JTable table = new JTable();
table.setAutoCreateRowSorter(true);
```

Pour finir …

…. on peut aussi

- Dessiner

> Pour en savoir plus : http://docs.oracle.com/javase/tutorial/uiswing/painting/index.html

- Modifier le Look and Feel
- Faire du Drag and Drop sur les composants (« atomiques »)
- …

Les applets

- Ce sont des applications Java exécutées dans un navigateur prenant en charge Java (*plug-in* Java installé)
- Ils sont dérivés à partir de la classe Applet (java.applet.*) ou JApplet (manipulation d'objet Swing)
  - pas de méthodes main()
  - Héritage des méthodes :
    - init() → appel au moment du chargement de l'applet
    - start() → appel lorsque le navigateur démarre l'applet
    - stop() → appel lorsque le navigateur arrête l'applet
    - destroy() : appel lorsque le navigateur ferme l'applet
    - paint() : appel pour redessiner l'applet (entièrement ou en partie)
- Intégration à un document (X)HTML via la balise <applet>

Pour en savoir plus : http://docs.oracle.com/javase/tutorial/deployment/applet/index.html

Java FX

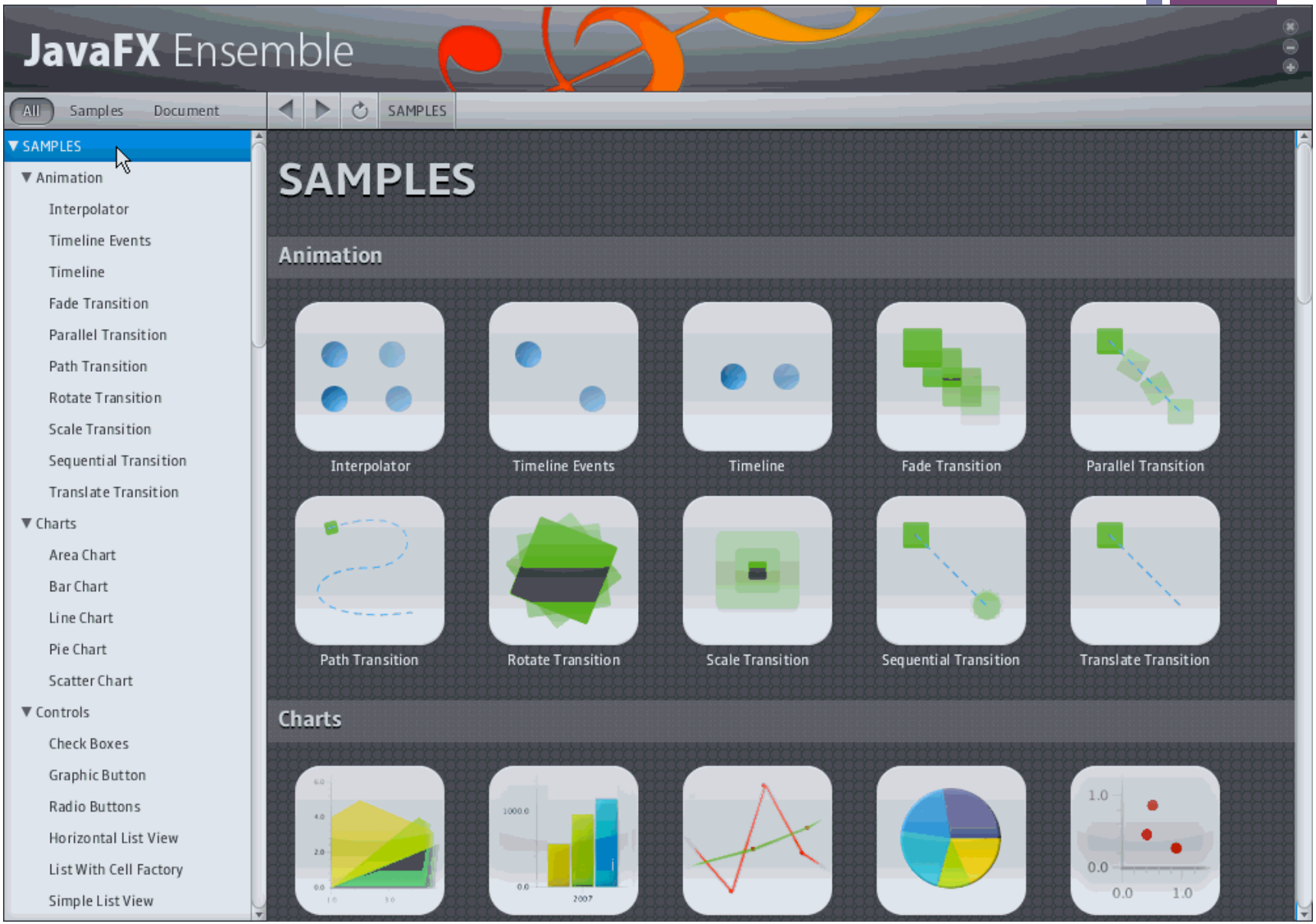
- Objectif : permettre aux développeurs de créer et déployer des applications internet riches (RIA) se comportant de manière consistante sur différentes plateformes
- Caractéristiques
  - → Actuellement : intégration totale avec la JDK 7
  - → APIs Java pour Java FX
    - Possibilité de continuer à utiliser les outils tels que les IDEs, les debuggers, les profileurs, le *refactoring*, …
  - → Nouveau moteur graphique : accélération du pipeline graphique couplé à un nouveau *windowing toolkit*
  - → Nouveau moteur multimédia permettant la lecture de contenu web

Java FX

- Un composant web : possibilité d'intégrer des pages web dans une application Java FX en utilisant la technologie du rendu (rendering) WebKit HTML
- Une mise en page XML-based via FXML
- Une variété de contrôles intégrés pour les interfaces utilisateurs (*Charts, Tables, Menus, Panes*)
- Disponible sur les plateformes Windows, Linux et Mac OS

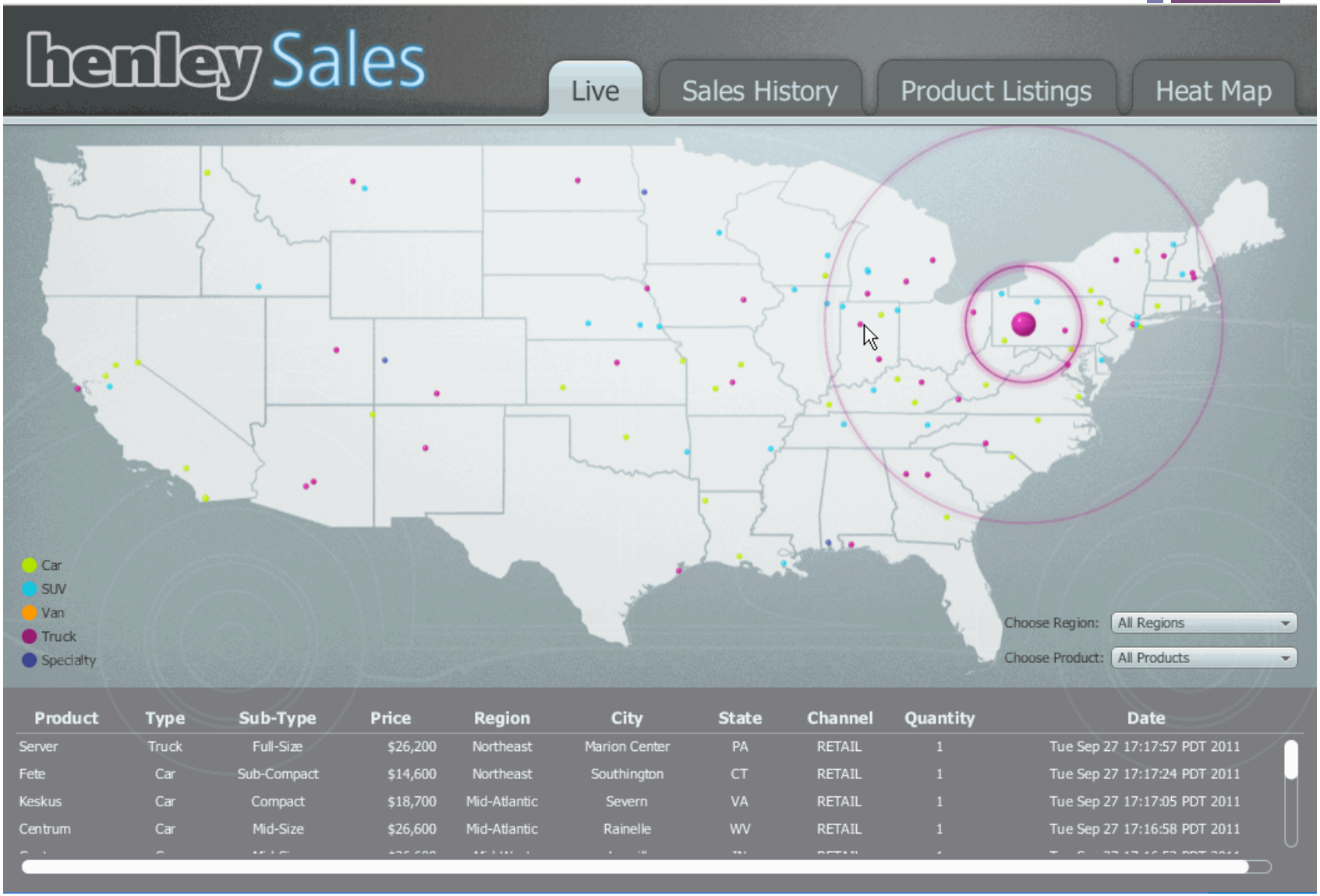
Pour en savoir plus : http://docs.oracle.com/javafx/index.html
APIs de Java FX : http://docs.oracle.com/javafx/2/api/index.html

Java FX



http://docs.oracle.com/javafx/2/overview/ensemble.gif

Java FX



http://docs.oracle.com/javafx/2/overview/ensemble.gif