



Programmation Orientée Objet Avancée (Partie 1)

Génie Informatique
Semestre 3

Enseignante : Ghada Feki

Plan

2

- **Introduction**
- **Composants SWING**
- **Gestionnaires de disposition**
- **Gestion des événements**

Plan

3

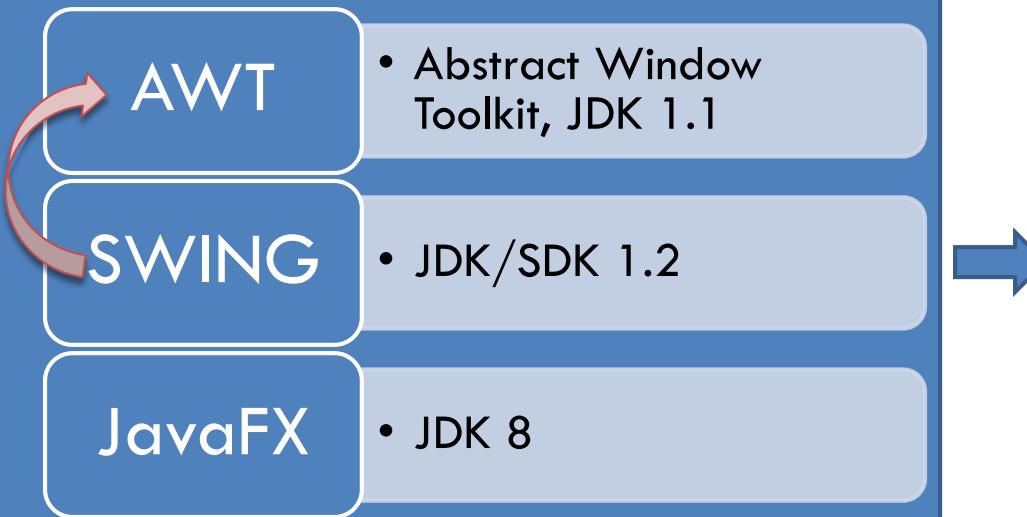
- **Introduction**
- **Composants SWING**
- **Gestionnaires de disposition**
- **Gestion des événements**

Introduction

4

- même gestion des événements
- les classes de Swing héritent des classes de AWT

Composants graphiques en JAVA



Interfaces graphiques

Introduction

5

Bibliothèque Java Foundation Classes (JFC)

API AWT (Abstract Windows Toolkit)

API SWING

Accessibility API

2D API

API pour l'impression et le cliquer/glisser



Ensemble de classes décrivant les composants graphiques, les polices, les couleurs et les images.

Introduction

6

Bibliothèque Java Foundation Classes (JFC)

API AWT (Abstract
Windows Toolkit)

API SWING



Accessibility API

2D API

API pour l'impression
et le cliquer/glisser

Même but de l'AWT.
Mode de fonctionnement et
d'utilisation est complètement
différent.

Introduction

7

Bibliothèque Java Foundation Classes (JFC)

API AWT (Abstract Windows Toolkit)

API SWING

Accessibility API

2D API

API pour l'impression et le cliquer/glisser

- Tous les composants de AWT ont leur équivalent dans Swing
 - **en plus joli**
 - **avec plus de fonctionnalités**
- Swing offre de nombreux composants qui n'existent pas dans AWT

Introduction

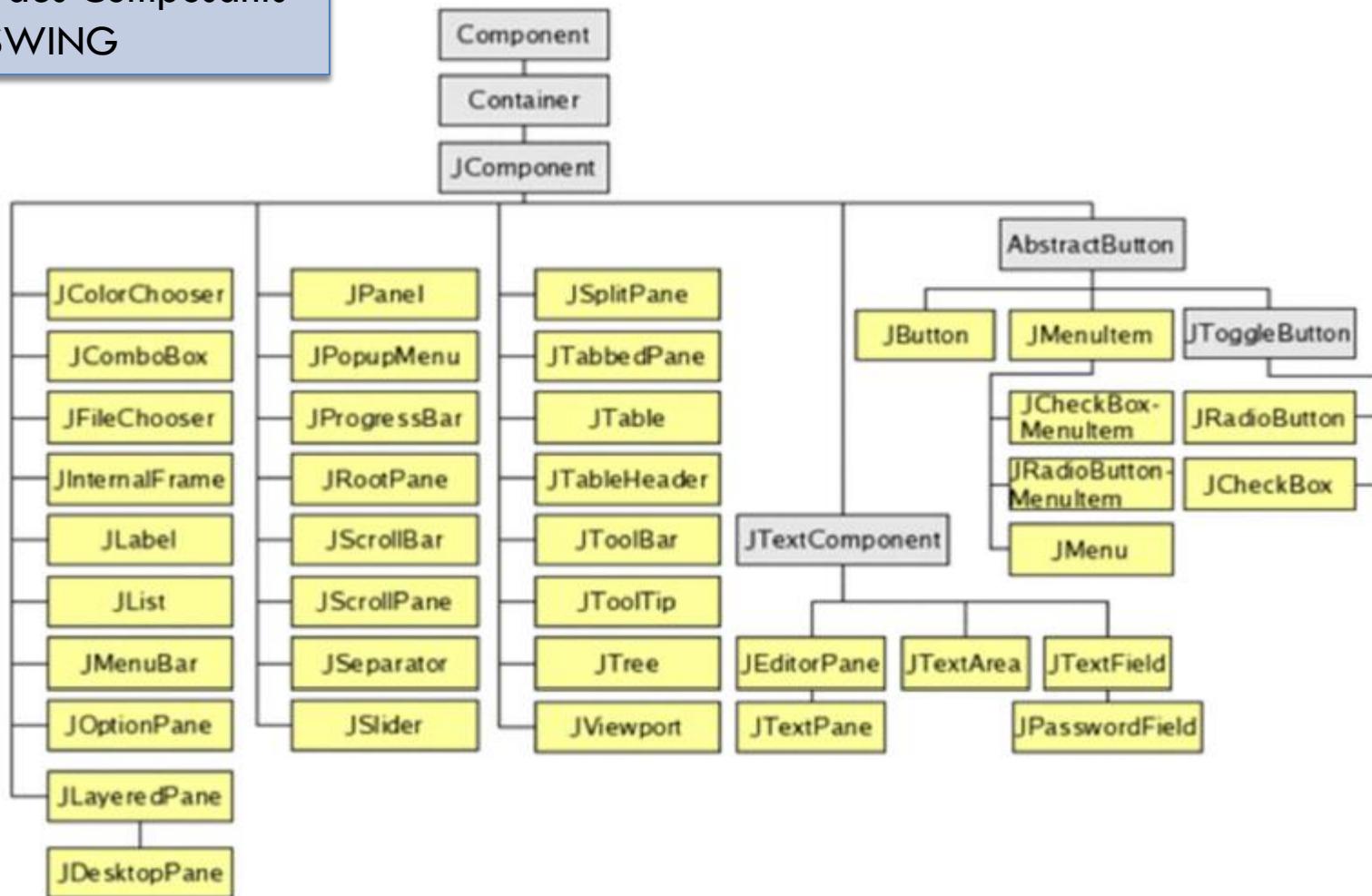
8

- Les composants Swing forment une nouvelle hiérarchie parallèle à celle de l'AWT.
- L'ancêtre de cette hiérarchie est le composant JComponent.
- Presque tous ces composants sont écrits en pure Java : ils ne possèdent aucune partie native sauf ceux qui assurent l'interface avec le système d'exploitation : JApplet, JDialog, JFrame, et JWindow.
- Cela permet aux composants de toujours avoir la même apparence quelque soit le système sur lequel l'application s'exécute.

Introduction

9

Hiérarchie des Composants SWING



Introduction

10

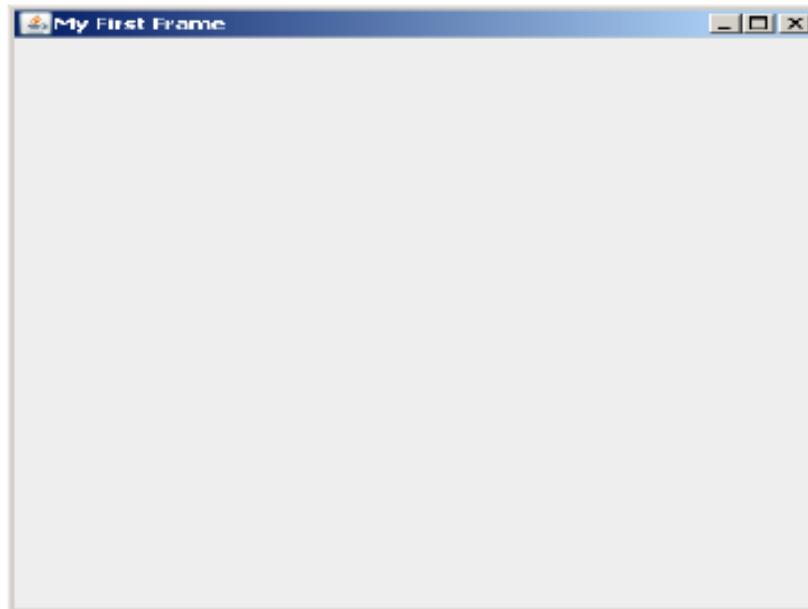
```
package graphique;
public class TestJFrame {
    public static void main(String[] args) {
        javax.swing.JFrame jFrame=new javax.swing.JFrame();
        jFrame.setTitle("My First Frame");
        jFrame.setSize(400,400);
        jFrame.setLocation(200, 200);
        jFrame.setDefaultCloseOperation(javax.swing.JFrame.EXIT_ON_CLOSE);
        jFrame.setVisible(true);
    }
}
```



Introduction

11

```
package graphique;
public class TestJFrame {
    public static void main(String[] args) {
        javax.swing.JFrame jFrame=new javax.swing.JFrame("My First Frame");
        jFrame.setBounds(200, 200, 400, 400);
        jFrame.setDefaultCloseOperation(javax.swing.JFrame.EXIT_ON_CLOSE);
        jFrame.setVisible(true);
    }
}
```



Introduction

12

```
package graphique;
public class TestJFrame {
    public static void main(String[] args) {
        javax.swing.JFrame jFrame=new javax.swing.JFrame("My First Frame");
        jFrame.setSize(400,400);
    // Permet de centrer la fenetre par rapport à l'écran
        jFrame.setLocationRelativeTo(null);
        jFrame.setDefaultCloseOperation(javax.swing.JFrame.EXIT_ON_CLOSE);
        jFrame.setVisible(true);
    }
}
```



Plan

13

- Introduction
- Composants SWING
- Gestionnaires de disposition
- Gestion des événements

Composants lourds & légers

14

- Les composants lourds sont dépendants du système hôte. En Java le composant lourd est identique en tant qu'objet Java et il est associé localement lors de l'exécution sur la plateforme hôte à un élément local dépendant du système hôte.
- Les composants légers sont entièrement écrits en Java. Un composant léger n'est pas dessiné visuellement par le système, mais en Java. Ceci apporte principalement une amélioration de portabilité.
- L'utilisation de composants lourds améliore la rapidité d'exécution mais nuit à la portabilité et nécessite de grandes ressources matérielles.

- En Java on ne peut pas se passer de composants lourds (communiquant avec le système) car la Java Machine doit communiquer avec son système hôte.
- Par exemple la fenêtre étant l'objet visuel de base dans les systèmes modernes elle est donc essentiellement liée au système d'exploitation et donc ce sera en Java un composant lourd.
- Dans le package swing le nombre de composants lourds est réduit au strict minimum soient 4 genres de fenêtres (dans ce cours on s'intéresse à JFrame).

Conteneurs (Container)

16

- Pour construire une interface graphique avec Swing, il faut créer un (ou plusieurs) container lourd et placer à l'intérieur les composants légers qui forment l'interface graphique
- Les conteneurs sont des objets graphiques qui sont destinés spécifiquement à recevoir d'autres éléments graphiques.
- Les conteneurs peuvent éventuellement contenir d'autres conteneurs.
- Ils héritent de la classe Container.
- Un composant graphique doit toujours être incorporer dans un conteneur.

Conteneurs (Container)

17

Il y a 4 sortes de containers lourds :

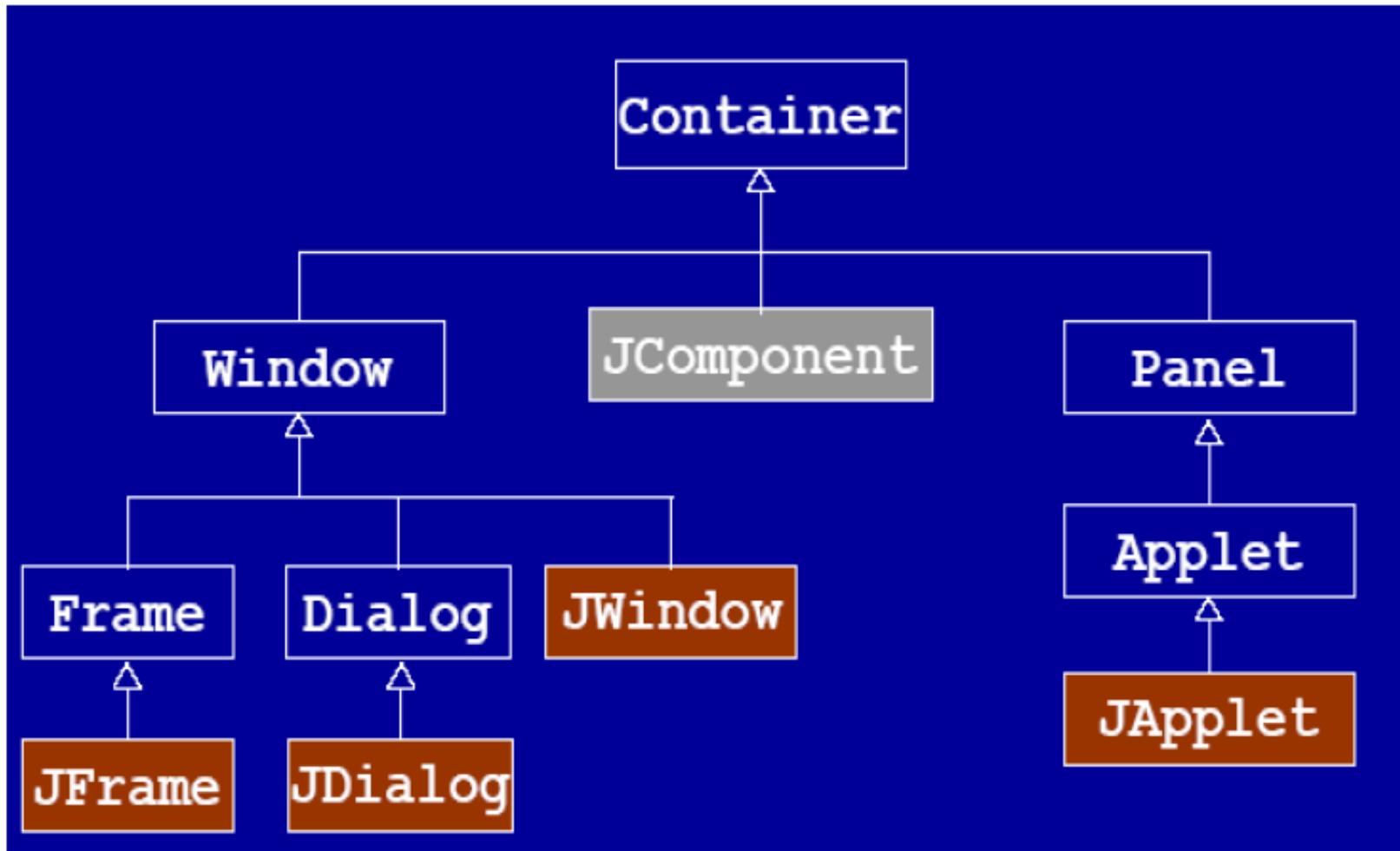
- JFrame fenêtre pour les applications
- JApplet pour les applets
- JDialog pour les fenêtres de dialogue
- JWindow, est plus rarement utilisé

Les containers « intermédiaires » légers :

- JPanel,
- JScrollPane,
- JSplitPane,
- JTabbedPane,
- Box (ce dernier est léger mais n'hérite pas de JComponent)

Hiérarchie d'héritage des containers lourds

18



Exemple de Container lourds: JFrame

19

La Classe `javax.swing.JFrame` :

- Elle représente une fenêtre principale qui possède un titre, une taille modifiable, éventuellement un menu, etc.
- Elle utilise un conteneur ou panneau de contenu (content pane) pour insérer des composants (ils ne sont plus insérer directement au JFrame mais à l'objet `contentPane` qui lui est associé).

Exemple de Container léger : JPanel

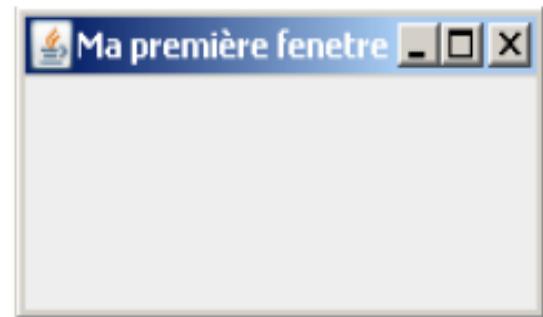
20

- **JPanel** est la classe mère des containers intermédiaires les plus simples.
- Un **JPanel** sert à regrouper des composants dans une zone d'écran.
- Il n'a pas d'aspect visuel déterminé ; son aspect visuel est donné par les composants qu'il contient.
- Il peut aussi servir de composant dans lequel on peut dessiner ce que l'on veut, ou faire afficher une image (par la méthode **paintComponent**).

Composant swing : JFrame

21

```
package graphique;
public class TestJFrame extends javax.swing.JFrame{
    public TestJFrame() {
        this("My First Frame");
    }
    public TestJFrame(String title) {
        super(title);
        this.setSize(200,100);
        this.setLocationRelativeTo(null);
        this.setDefaultCloseOperation(javax.swing.JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String[] args) {
        TestJFrame testJFrame=new TestJFrame("Ma première fenetre");
        testJFrame.setVisible(true);
    }
}
```



Composant swing : JFrame

22

- Il est possible de préciser comment un objet Jframe, JInternalFrame, ou JDialog réagit à sa fermeture grâce à la méthode setDefaultCloseOperation().
- Cette méthode attend en paramètre une valeur qui peut être :

Constante	Rôle
WindowConstants.DISPOSE_ON_CLOSE	détruit la fenêtre
WindowConstants.DO NOTHING_ON_CLOSE	rend le bouton de fermeture inactif
WindowConstants.HIDE_ON_CLOSE	cache la fenêtre
WindowConstants.EXIT_ON_CLOSE	Quitte le programme

Composant swing : JButton

23

javax.swing

Class JButton

```
java.lang.Object
└ java.awt.Component
  └ java.awt.Container
    └ javax.swing.JComponent
      └ javax.swing.AbstractButton
        └ javax.swing.JButton
```

All Implemented Interfaces:

[ImageObserver](#), [ItemSelectable](#), [MenuContainer](#), [Serializable](#), [Accessible](#), [SwingConstants](#)

Choisir votre photo

Composant swing : JButton

24

Constructor Summary

JButton()

Creates a button with no set text or icon.

JButton(Action a)

Creates a button where properties are taken from the Action supplied.

JButton(Icon icon)

Creates a button with an icon.

JButton(String text)

Creates a button with text.

JButton(String text, Icon icon)

Creates a button with initial text and an icon.

Composant swing : JLabel

25

javax.swing

Class JLabel

java.lang.Object

 └ java.awt.Component

 └ java.awt.Container

 └ javax.swing.JComponent

 └ javax.swing.JLabel

All Implemented Interfaces:

ImageObserver, MenuContainer, Serializable, Accessible, SwingConstants

A screenshot of a Java Swing application window. At the top, there is a menu bar with "File", "Edit", "View", "Help", and a "Recent Projects" dropdown. Below the menu is a toolbar with icons for "New", "Open", "Save", "Print", "Find", "Copy", "Paste", "Delete", and "Exit". The main area contains three text input fields: a blue one labeled "Nom" with the text "John", a grey one labeled "Prenom" with the text "Doe", and a grey one labeled "Password" with the text "password123".

A screenshot of a Java Swing application window. It shows a single text input field with the label "Date de naissance" above it. The field contains the text "2024-01-01".

Composant swing : JLabel

26

Constructor Summary

JLabel()

Creates a JLabel instance with no image and with an empty string for the title.

JLabel(Icon image)

Creates a JLabel instance with the specified image.

JLabel(Icon image, int horizontalAlignment)

Creates a JLabel instance with the specified image and horizontal alignment.

JLabel(String text)

Creates a JLabel instance with the specified text.

JLabel(String text, Icon icon, int horizontalAlignment)

Creates a JLabel instance with the specified text, image, and horizontal alignment.

JLabel(String text, int horizontalAlignment)

Creates a JLabel instance with the specified text and horizontal alignment.

Composant swing : JLabel

27

Parmi les valeurs que peut prendre horizontalAlignment, on peut citer :

- SwingConstants.CENTER
- SwingConstants.LEFT
- SwingConstants.RIGHT

String	getText() Returns the text string that the label displays.
Void	setHorizontalAlignment(int alignment) Sets the alignment of the label's contents along the X axis.
void	setText(String text) Defines the single line of text this component will display.

Composant swing : JLabel

28

- **Remarque 1 :** par défaut un JLabel n'est pas opaque. Donc, il faut le rendre opaque avant si on veut modifier la couleur de fond

```
javax.swing.JLabel jLabelNom = new javax.swing.JLabel("Nom");  
jLabelNom.setOpaque(true) ;  
jLabelNom.setBackground(java.awt.Color.BLUE) ;
```

- **Remarque 2 :** pour avoir une étiquette dont le libellé est sur 2 lignes, ou une étiquette dont le libellé est à moitié en rouge et l'autre moitié en vert, on utilise du HTML.

```
new javax.swing.JLabel("<html>nom <br>de la personne</html>");
```

- **Remarque 3 :** JLabel implémente l'interface SwingConstants. Donc on aurait pu écrire JLabel.CENTER

Composant swing : JTextField

29

Constructor Summary

JTextField()

Constructs a new TextField.

JTextField(Document doc, String text, int columns)

Constructs a new JTextField that uses the given text storage model and the given number of columns.

JTextField(int columns)

Constructs a new empty TextField with the specified number of columns.

JTextField(String text)

Constructs a new TextField initialized with the specified text.

JTextField(String text, int columns)

Constructs a new TextField initialized with the specified text and columns.

Composant swing : JTextField

30

Method Summary

`void setColumns(int columns)`

Sets the number of columns in this TextField, and then invalidate the layout.

`void setFont(Font f)`

Sets the current font.

`String getSelectedText()`

Returns the selected text contained in this TextComponent.

`Color getSelectedTextColor()`

Fetches the current color used to render the selected text.

`Color getSelectionColor()`

Fetches the current color used to render the selection.

`String getText()`

Returns the text contained in this TextComponent.

`void setEditable(boolean b)`

Sets the specified boolean to indicate whether or not this TextComponent should be editable.

`void setSelectedTextColor(Color c)`

Sets the current color used to render the selected text.

`void setText(String t)`

Sets the text of this TextComponent to the specified text.

Composant swing : JTextArea

31

Constructor Summary

[JTextArea\(\)](#)

Constructs a new TextArea.

[JTextArea\(Document doc\)](#)

Constructs a new JTextArea with the given document model, and defaults for all of the other arguments (null, 0, 0).

[JTextArea\(Document doc, String text, int rows, int columns\)](#)

Constructs a new JTextArea with the specified number of rows and columns, and the given model.

[JTextArea\(int rows, int columns\)](#)

Constructs a new empty TextArea with the specified number of rows and columns.

[JTextArea\(String text\)](#)

Constructs a new TextArea with the specified text displayed.

[JTextArea\(String text, int rows, int columns\)](#)

Constructs a new TextArea with the specified text and number of rows and columns.

Composant swing : JTextArea

32

Method Summary

`void append(String str)`

Appends the given text to the end of the document.

`int getLineCount()`

Determines the number of lines contained in the area.

`void setColumns(int columns)`

Sets the number of columns for this TextArea.

`void setFont(Font f)`

Sets the current font.

`void setRows(int rows)`

Sets the number of rows for this TextArea.

Composant swing : JCheckBox

33

Constructor Summary

JCheckBox()

Creates an initially unselected check box button with no text, no icon.

JCheckBox(Action a)

Creates a check box where properties are taken from the Action supplied.

JCheckBox(Icon icon)

Creates an initially unselected check box with an icon.

JCheckBox(Icon icon, boolean selected)

Creates a check box with an icon and specifies whether or not it is initially selected.

JCheckBox(String text)

Creates an initially unselected check box with text.

JCheckBox(String text, boolean selected)

Creates a check box with text and specifies whether or not it is initially selected.

JCheckBox(String text, Icon icon)

Creates an initially unselected check box with the specified text and icon.

JCheckBox(String text, Icon icon, boolean selected)

Creates a check box with text and icon, and specifies whether or not it is initially selected.

Composant swing : JCheckBox

34

Method Summary

`boolean isSelected()`

Returns the state of the button.

`void setSelected(boolean b)`

Sets the state of the button.

Composant swing : JRadioButton

35

Constructor Summary

JRadioButton()

Creates an initially unselected radio button with no set text.

JRadioButton(Action a)

Creates a radiobutton where properties are taken from the Action supplied.

JRadioButton(Icon icon)

Creates an initially unselected radio button with the specified image but no text.

JRadioButton(Icon icon, boolean selected)

Creates a radio button with the specified image and selection state, but no text.

JRadioButton(String text)

Creates an unselected radio button with the specified text.

JRadioButton(String text, boolean selected)

Creates a radio button with the specified text and selection state.

JRadioButton(String text, Icon icon)

Creates a radio button that has the specified text and image, and that is initially unselected.

JRadioButton(String text, Icon icon, boolean selected)

Creates a radio button that has the specified text, image, and selection state.

Composant swing : JRadioButton

36

□ Remarques :

Si on ajoute plusieurs boutons radios à une fenêtre, ils se comporteront comme s'ils étaient des cases à cocher, c.-à-d. qu'une sélection d'un bouton radio n'entraînera pas la désélection d'un autre bouton radio mais les deux restent sélectionnés. Or, ce n'est pas le comportement voulu. Pour réaliser le comportement voulu, on doit grouper tous les boutons radios concernés en utilisant la classe `javax.swing.ButtonGroup`.

Composant swing : JComboBox

37

Constructor Summary

JComboBox()

Creates a JComboBox with a default data model.

JComboBox(ComboBoxModel aModel)

Creates a JComboBox that takes its items from an existing ComboBoxModel.

JComboBox(Object[] items)

Creates a JComboBox that contains the elements in the specified array.

JComboBox(Vector<?> items)

Creates a JComboBox that contains the elements in the specified Vector.

Composant swing : JComboBox

38

Method Summary

`void addItem(Object anObject)`

 Adds an item to the item list.

`Object getItemAt(int index)`

 Returns the list item at the specified index.

`int getItemCount()`

 Returns the number of items in the list.

`int getSelectedIndex()`

 Returns the first item in the list that matches the given item.

`Object getSelectedItem()`

 Returns the current selected item.

`void removeItem(Object anObject)`

 Removes an item from the item list.

`void removeItemAt(int anIndex)`

 Removes the item at an Index This method works only if the JComboBox uses a mutable data model.

Composant swing : JList

39

Constructor Summary

JList()

Constructs a JList with an empty, read-only, model.

JList(ListModel dataModel)

Constructs a JList that displays elements from the specified, non-null, model.

JList(Object[] listData)

Constructs a JList that displays the elements in the specified array.

JList(Vector<?> listData)

Constructs a JList that displays the elements in the specified Vector.

Composant swing : JList

40

Method Summary

`int getSelectedIndex()`

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

`int[] getSelectedIndices()`

Returns an array of all of the selected indices, in increasing order.

`Object getSelectedValue()`

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

`Object[] getSelectedValues()`

Returns an array of all the selected values, in increasing order based on their indices in the list.

`boolean isSelectedIndex(int index)`

Returns true if the specified index is selected, else false.

`void setModel(ListModel model)`

Sets the model that represents the contents or "value" of the list, notifies property change listeners, and then clears the list's selection.

`void setSelectedValue(Object anObject, boolean shouldScroll)`

Selects the specified object from the list.

`void setVisibleRowCount(int visibleRowCount)`

Sets the visibleRowCount property, which has different meanings depending on the layout orientation: For a VERTICAL layout orientation, this sets the preferred number of rows to display without requiring scrolling; for other orientations, it affects the wrapping of cells.

Composant swing : JList

41

```
package graphique;

public class TestJList extends javax.swing.JFrame {

    public TestJList() {
        super("Test liste");
        this.setSize(200,200);
        this.setLocationRelativeTo(null);
        this.setDefaultCloseOperation(javax.swing.JFrame.EXIT_ON_CLOSE);
        this.getContentPane().setLayout(new java.awt.FlowLayout());

        String[]lettre={"A","B","C","D","E","F","G","H","I","J","K"};
        javax.swing.JList jList=new javax.swing.JList(lettre);
        this.getContentPane().add(jList);
    }

    public static void main(String[] args) {
        new TestJList().setVisible(true);
    }
}
```



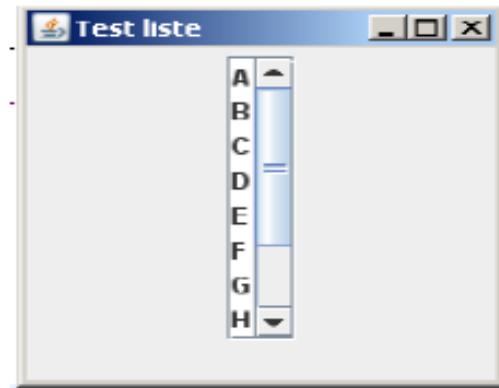
Composant swing : JList & JScrollPane

42

```
package graphique;
public class TestJList extends javax.swing.JFrame{
    public TestJList() {
        super("Test liste");
        this.setSize(200,200);
        this.setLocationRelativeTo(null);

        this.setDefaultCloseOperation(javax.swing.JFrame.EXIT_ON_CLOSE);
        this.getContentPane().setLayout(new java.awt.FlowLayout());
        String[]lettre={"A","B","C","D","E","F","G","H","I","J","K"};
        javax.swing.JList jList=new javax.swing.JList(lettre);
        javax.swing.JScrollPane jScrollPane=new javax.swing.JScrollPane(jList);
        this.getContentPane().add(jScrollPane);

    }
    public static void main(String[] args) {
        new TestJList().setVisible(true);
    }
}
```



Composant swing : JList with DefaultListModel

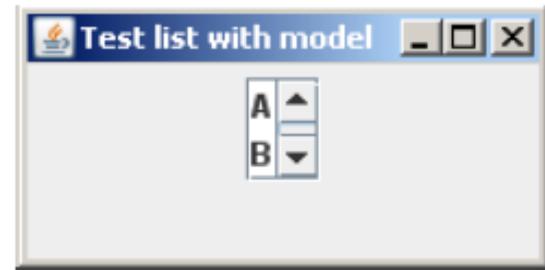
43

```
package graphique;
public class TestJListWithModel extends javax.swing.JFrame{
    public TestJListWithModel() {
        super("Test list with model");
        this.setSize(200,100);
        this.setLocationRelativeTo(null);
        this.setDefaultCloseOperation(javax.swing.JFrame.EXIT_ON_CLOSE);
        this.getContentPane().setLayout(new java.awt.FlowLayout());

        javax.swing.DefaultListModel defaultListModel=new javax.swing.DefaultListModel();
        defaultListModel.addElement("A");
        defaultListModel.addElement("B");
        defaultListModel.addElement("C");

        javax.swing.JList jList=new javax.swing.JList(defaultListModel);
        jList.setVisibleRowCount(2);

        javax.swing.JScrollPane jScrollPane=new javax.swing.JScrollPane(jList);
        this.getContentPane().add(jScrollPane);
    }
}
```



Composant swing : JList

44

void setSelectionMode(int selectionMode)

Sets the selection mode for the list.

Si selectionMode est égal à :

- **ListSelectionModel.SINGLE_SELECTION**, alors on ne peut sélectionner qu'une seule valeur à la fois
- **ListSelectionModel.SINGLE_INTERVAL_SELECTION**, alors on peut sélectionner une suite de valeurs successives
- **ListSelectionModel.MULTIPLE_INTERVAL_SELECTION**, alors on peut sélectionner plusieurs valeurs à la fois n'importe où dans la liste (c'est celle par défaut)

Composant swing : JDialog

45

Constructor Summary

JDialog()

Creates a modeless dialog without a title and without a specified Frame owner.

JDialog(Dialog owner)

Creates a modeless dialog without a title with the specified Dialog as its owner.

JDialog(Dialog owner, boolean modal)

Creates a dialog with the specified owner Dialog and modality.

JDialog(Dialog owner, String title)

Creates a modeless dialog with the specified title and with the specified owner dialog.

JDialog(Dialog owner, String title, boolean modal)

Creates a dialog with the specified title, modality and the specified owner Dialog.

JDialog(Dialog owner, String title, boolean modal, GraphicsConfiguration gc)

Creates a dialog with the specified title, owner Dialog, modality and GraphicsConfiguration.

JDialog(Frame owner)

Creates a modeless dialog without a title with the specified Frame as its owner.

JDialog(Frame owner, boolean modal)

Creates a dialog with the specified owner Frame, modality and an empty title.

JDialog(Frame owner, String title)

Creates a modeless dialog with the specified title and with the specified owner frame.

Composant swing : JOptionPane

46

- C'est une classe de Java qui permet de créer facilement des boîtes de dialogue. L'utilisation typique de cette classe consiste à des appels à une des méthodes statiques de la forme showXxxDialog qui suivent :

Méthodes	Description
showConfirmDialog	Demande une question qui se répond par oui/non/annulé
showInputDialog	Demande de taper une réponse
showMessageDialog	Transcrit un message à l'usage
showOptionDialog	Une méthode générale réunissant les trois précédente

Composant swing : JOptionPane

47

```
javax.swing.JOptionPane.showMessageDialog(null,"Message","Titre",messageType);
```

Le type de message est l'une des valeurs suivantes :

- *JOptionPane.PLAIN_MESSAGE*; 
- *JOptionPane.ERROR_MESSAGE*; 
- *JOptionPane.INFORMATION_MESSAGE*; 
- *JOptionPane.WARNING_MESSAGE*; 
- *JOptionPane.QUESTION_MESSAGE*;

```
javax.swing.JOptionPane.showConfirmDialog(null, "Message", "Title", messageType);
```

L'option de message est l'une des valeurs suivantes :

- *JOptionPane.DEFAULT_OPTION*
- *JOptionPane.YES_NO_OPTION*
- *JOptionPane.YES_NO_CANCEL_OPTION*
- *JOptionPane.OK_CANCEL_OPTION*

Composant swing : JTabbedPane

48

Constructor Summary

JTabbedPane()

Creates an empty TabbedPane with a default tab placement of JTabbedPane.TOP.

JTabbedPane(int tabPlacement)

Creates an empty TabbedPane with the specified tab placement of either: JTabbedPane.TOP, JTabbedPane.BOTTOM, JTabbedPane.LEFT, or JTabbedPane.RIGHT.

JTabbedPane(int tabPlacement, int tabLayoutPolicy)

Creates an empty TabbedPane with the specified tab placement and tab layout policy.

Composant swing : JTabbedPane

49

Method Summary

`void add(Component component, Object constraints)`

 Adds a component to the tabbed pane.

`void add(Component component, Object constraints, int index)`

 Adds a component at the specified tab index.

`Component add(String title, Component component)`

 Adds a component with the specified tab title.

`void addTab(String title, Component component)`

 Adds a component represented by a title and no icon.

`void addTab(String title, Icon icon, Component component)`

 Adds a component represented by a title and/or icon, either of which can be null.

`void addTab(String title, Icon icon, Component component, String tip)`

 Adds a component and tip represented by a title and/or icon, either of which can be null.

`Component getComponentAt(int index)`

 Returns the component at index.

`int getTabCount()`

 Returns the number of tabs in this tabbedpane.

`String getTitleAt(int index)`

 Returns the tab title at index.

`int indexOfTab(String title)`

 Returns the first tab index with a given title, or -1 if no tab has this title.

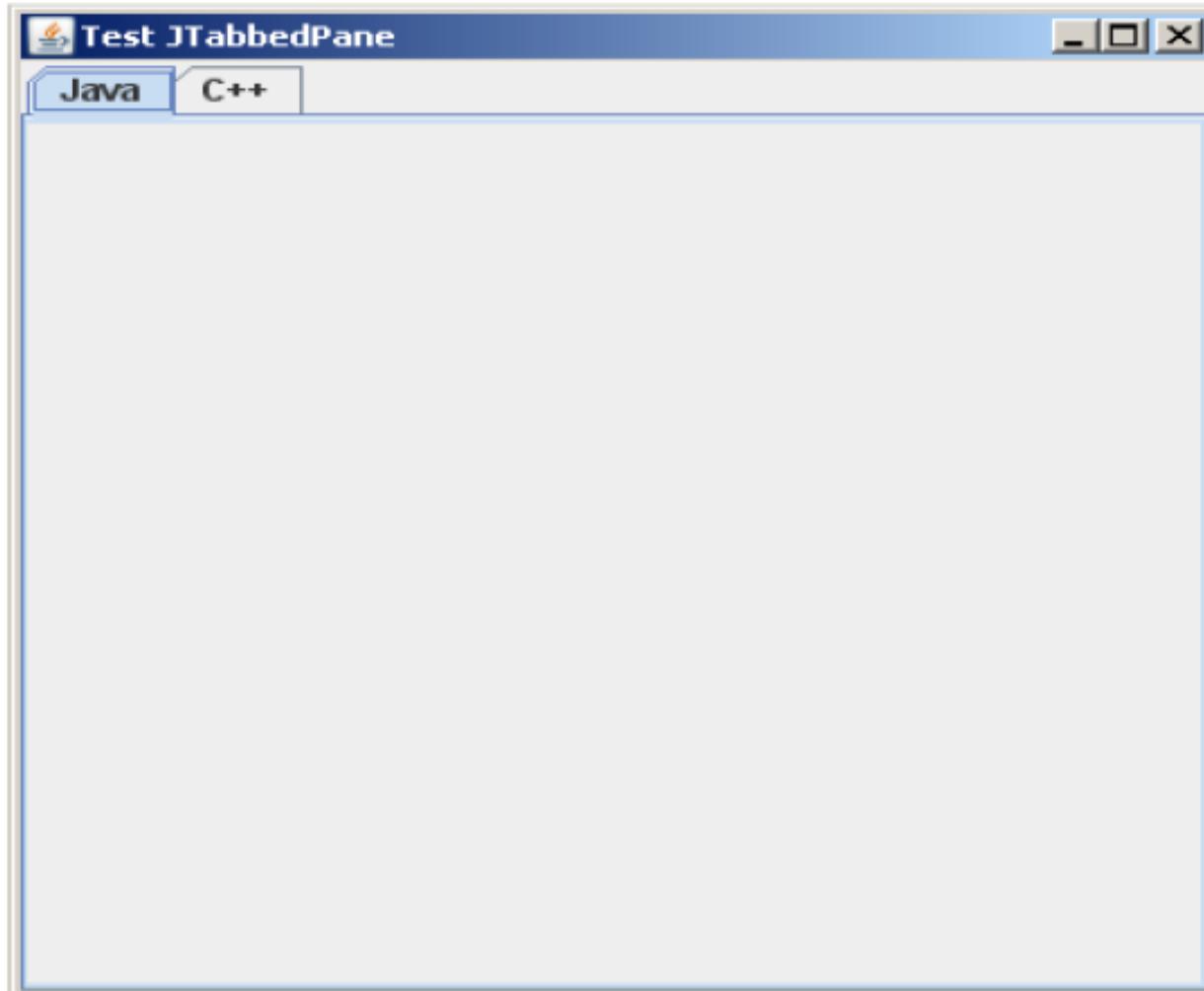
Composant swing : JTabbedPane

50

```
package graphique;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JTabbedPane;
import javax.swing.WindowConstants;
public class TestJTabbedPane extends JFrame{
    private JTabbedPane jTabbedPane;
    public TestJTabbedPane() {
        super("Test JTabbedPane");
        jTabbedPane = new JTabbedPane();
        jTabbedPane.addTab("Java", new JPanel());
        jTabbedPane.addTab("C++", new JPanel());
        this.getContentPane().add(jTabbedPane);
        this.setSize(400,400);
        this.setLocationRelativeTo(null);
    this.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
    }
}
```

Composant swing : JTabbedPane

51



Composant swing : JSplitPane

52

Constructor Summary

[JSplitPane\(\)](#)

Creates a new JSplitPane configured to arrange the child components side-by-side horizontally with no continuous layout, using two buttons for the components.

[JSplitPane\(int newOrientation\)](#)

Creates a new JSplitPane configured with the specified orientation and no continuous layout.

[JSplitPane\(int newOrientation, boolean newContinuousLayout\)](#)

Creates a new JSplitPane with the specified orientation and redrawing style.

[JSplitPane\(int newOrientation, boolean newContinuousLayout, Component newLeftComponent, Component newRightComponent\)](#)

Creates a new JSplitPane with the specified orientation and redrawing style, and with the specified components.

[JSplitPane\(int newOrientation, Component newLeftComponent, Component newRightComponent\)](#)

Creates a new JSplitPane with the specified orientation and with the specified components that do not do continuous redrawing.

Composant swing : JSplitPane

53

Method Summary

`int getOrientation()`

Returns the orientation.

`void setLeftComponent(Component comp)`

Sets the component to the left (or above) the divider.

`void setOneTouchExpandable(boolean newValue)`

Sets the value of the oneTouchExpandable property, which must be true for the JSplitPane to provide a UI widget on the divider to quickly expand/collapse the divider.

`void setRightComponent(Component comp)`

Sets the component to the right (or below) the divider.

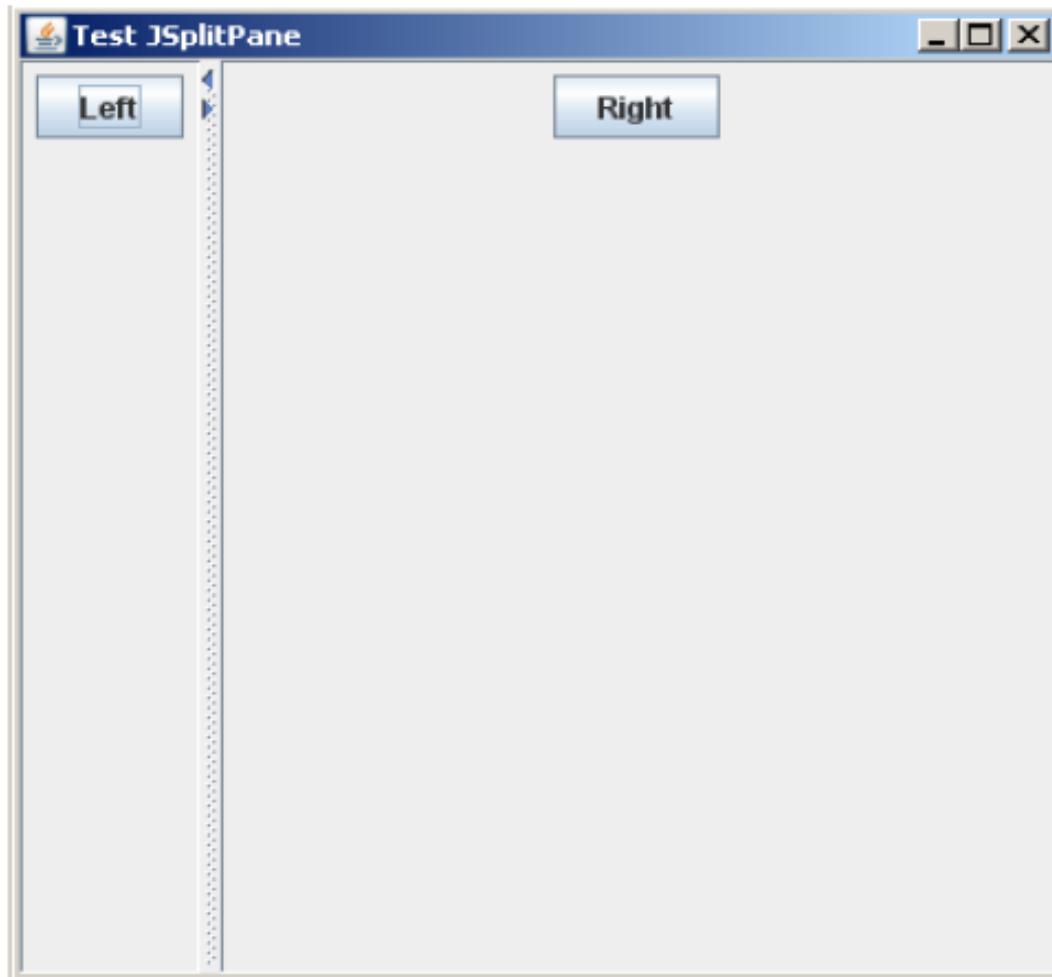
Composant swing : JSplitPane

54

```
package graphique;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JSplitPane;
public class TestJSplitPane extends JFrame{
    private JSplitPane jSplitPane;
    public TestJSplitPane() {
        super("Test JSplitPane");
        this.setSize(400,400);
        this.setLocationRelativeTo(null);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JPanel left=new JPanel();
        left.add(new JButton("Left"));
        JPanel right=new JPanel();
        right.add(new JButton("Right"));
        jSplitPane = new JSplitPane();
        jSplitPane.setOneTouchExpandable(true);
        jSplitPane.setLeftComponent(left);
        jSplitPane.setRightComponent(right);
        this.add(jSplitPane);
    }
}
```

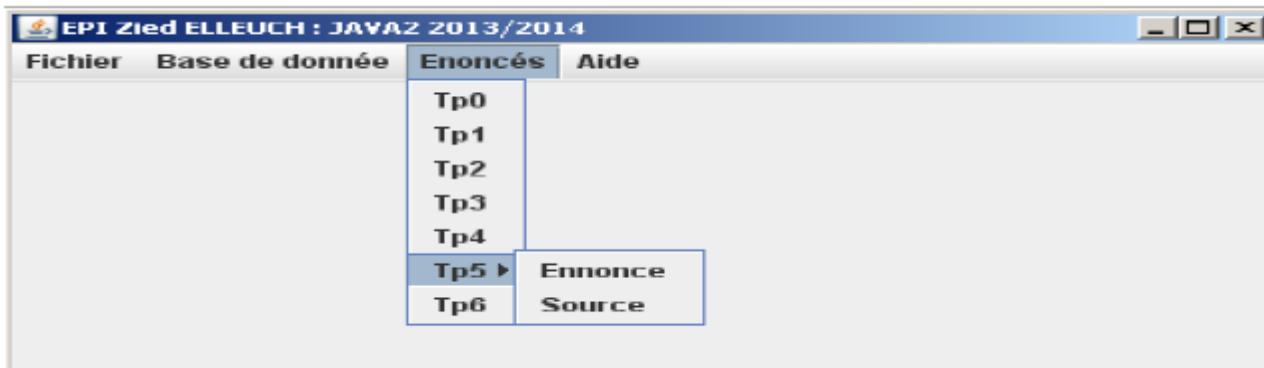
Composant swing : JSplitPane

55



Composant swing : JMenuBar

56



Composant swing : JMenuBar

57

Constructor Summary

JMenuBar()

Creates a new menu bar.

Method Summary

JMenu add(JMenu c)

Appends the specified menu to the end of the menu bar.

Component getComponentAtIndex(int i)

Deprecated. replaced by *getComponent(int i)*

boolean isBorderPainted()

Returns true if the menu bars border should be painted.

boolean isSelected()

Returns true if the menu bar currently has a component selected.

void setBorderPainted(boolean b)

Sets whether the border should be painted.

Composant swing : JMenu

58

Constructor Summary

JMenu()

Constructs a new JMenu with no text.

JMenu(Action a)

Constructs a menu whose properties are taken from the Action supplied.

JMenu(String s)

Constructs a new JMenu with the supplied string as its text.

JMenu(String s, boolean b)

Constructs a new JMenu with the supplied string as its text and specified as a tear-off menu or not.

Composant swing : JMenu

59

Method Summary

[JMenuItem add\(JMenuItem menuItem\)](#)

Appends a menu item to the end of this menu.

[void addSeparator\(\)](#)

Appends a new separator to the end of the menu.

[int getItemCount\(\)](#)

Returns the number of items on the menu, including separators.

[void setAccelerator\(KeyStroke keyStroke\)](#)

setAccelerator is not defined for JMenu.

Composant swing : JMenuItem

60

Constructor Summary

JMenuItem()

Creates a JMenuItem with no set text or icon.

JMenuItem(Action a)

Creates a menu item whose properties are taken from the specified Action.

JMenuItem(Icon icon)

Creates a JMenuItem with the specified icon.

JMenuItem(String text)

Creates a JMenuItem with the specified text.

JMenuItem(String text, Icon icon)

Creates a JMenuItem with the specified text and icon.

JMenuItem(String text, int mnemonic)

Creates a JMenuItem with the specified text and keyboard mnemonic.

Composant swing : JMenuItem

61

Method Summary

KeyStroke getAccelerator()

Returns the KeyStroke which serves as an accelerator for the menu item.

void setAccelerator(KeyStroke keyStroke)

Sets the key combination which invokes the menu item's action listeners without navigating the menu hierarchy.

void setEnabled(boolean b)

Enables or disables the menu item.

Composant swing : JSlider

62



Constructor Summary

JSlider()

Creates a horizontal slider with the range 0 to 100 and an initial value of 50.

JSlider(int orientation)

Creates a slider using the specified orientation with the range 0 to 100 and an initial value of 50.

JSlider(int min, int max)

Creates a horizontal slider using the specified min and max with an initial value equal to the average of the min plus max.

JSlider(int min, int max, int value)

Creates a horizontal slider using the specified min, max and value.

JSlider(int orientation, int min, int max, int value)

Creates a slider with the specified orientation and the specified minimum, maximum, and initial values.

Composant swing : JSlider

63

Method Summary

`int getMaximum()`

Returns the maximum value supported by the slider from the BoundedRangeModel.

`int getMinimum()`

Returns the minimum value supported by the slider from the BoundedRangeModel.

`int getValue()`

Returns the slider's current value from the BoundedRangeModel.

`void setOrientation(int orientation)`

Set the slider's orientation to either SwingConstants.VERTICAL or SwingConstants.HORIZONTAL.

`void setValue(int n)`

Sets the slider's current value to n.

Composant swing : Taille

64

- Tous les composants graphiques (classe Component) peuvent indiquer leurs tailles pour l'affichage
 - taille maximum
 - taille préférée
 - taille minimum
- Accesseurs et modificateurs associés :
{get | set}{Maximum | Preferred | Minimum}Size

Composant swing : Taille

65

Taille préférée

- La taille préférée est la plus utilisée par les *layout managers* ; un composant peut l'indiquer en redéfinissant la méthode « **Dimension getPreferredSize()** »
- On peut aussi l'imposer « de l'extérieur » avec la méthode « **void setPreferredSize(Dimension)** »
- Mais le plus souvent, les gestionnaires de mise en place nettiendront pas compte des tailles imposées de l'extérieur

Taille minimum

- Quand un composant n'a plus la place pour être affiché à sa taille préférée, il est affiché à sa taille minimum sans passer par des tailles intermédiaires.
- Si la taille minimum est très petite, ça n'est pas du plus bel effet ; il est alors conseillé de fixer une taille minimum, par exemple par **c.setMinimumSize(c.getPreferredSize());**

Plan

66

- Introduction
- Composants SWING
- **Gestionnaires de disposition**
- Gestion des événements

Gestionnaire de disposition

67

- La disposition des composants dans les conteneurs est gérée par les *LayoutManager*. Il existe plusieurs types de *LayoutManager* avec des algorithmes de placement différents.
- L'utilisation des gestionnaires de disposition a les principaux avantages suivants :
 - En cas de redimensionnement de la fenêtre, l'aménagement des composants graphiques est délégué aux layout manager (il est inutile d'utiliser les coordonnées absolues). Ainsi, les composants sont automatiquement agrandis ou réduits.
 - Il est inutile de préciser les coordonnées de chaque emplacement. Le gestionnaire s'en charge selon la taille de la fenêtre
 - ils permettent une indépendance vis à vis des plateformes

Gestionnaire de disposition

68

- Quand on ajoute un composant dans un container on ne donne pas la position exacte du composant. On donne plutôt des indications de positionnement au gestionnaire de mise en place.
 - explicites (BorderLayout.NORTH)
 - ou implicites (ordre d'ajout dans le container)
- Un *layout manager* place les composants « au mieux » suivant :
 - l'algorithme de placement qui lui est propre
 - les indications de positionnement des composants
 - la taille du container
 - les tailles préférées des composants
- Il existe plusieurs gestionnaires dans java à savoir **FlowLayout**, **BorderLayout**, **GridLayout**, **BoxLayout**, **GridBagLayout**, **GroupLayout**, **CardLayout**.

Gestionnaire de disposition : java.awt.FlowLayout

69

- Le gestionnaire *FlowLayout* (*mise en page flot*) place les composants les uns à la suite des autres, sur la même ligne. Chaque ligne est complétée progressivement jusqu'à être remplie, puis passe à la suivante. Toutefois on peut changer le type d'alignement en utilisant les constantes suivantes :
 - `FlowLayout.LEFT` : alignement à gauche
 - `FlowLayout.RIGHT` : alignement à droite
 - `FlowLayout.CENTER` : alignement au centre
- **JPanel** admet par défaut comme gestionnaire de disposition *FlowLayout* avec un alignement aux centre.

Gestionnaire de disposition : java.awt.FlowLayout

70

Constructor Summary

[FlowLayout\(\)](#)

Constructs a new FlowLayout with a centered alignment and a default 5-unit horizontal and vertical gap.

[FlowLayout\(int align\)](#)

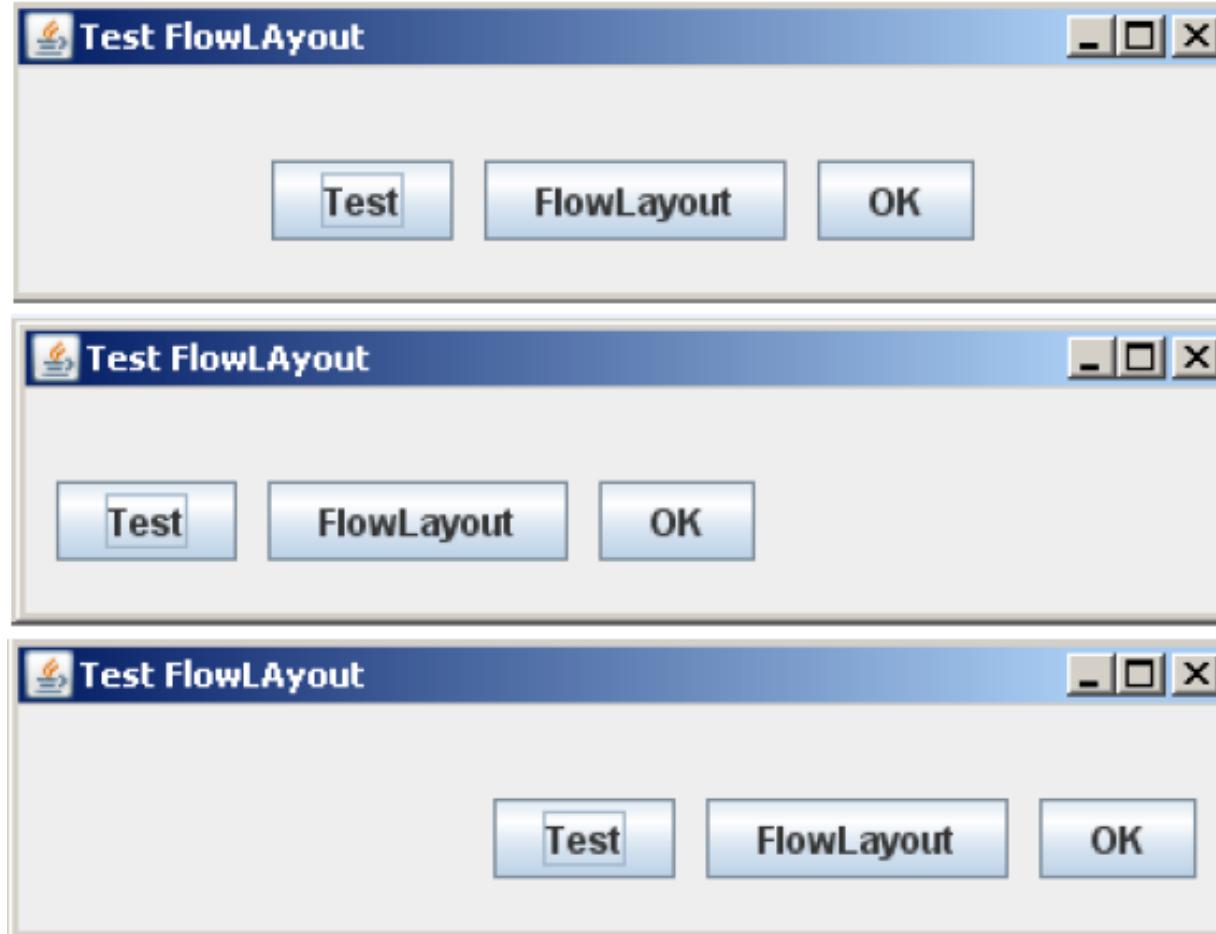
Constructs a new FlowLayout with the specified alignment and a default 5-unit horizontal and vertical gap.

[FlowLayout\(int align, int hgap, int vgap\)](#)

Creates a new flow layout manager with the indicated alignment and the indicated horizontal and vertical gaps.

Gestionnaire de disposition : java.awt.FlowLayout

71



Par défaut, les composants sont espacés de 5 pixels

Gestionnaire de disposition : java.awt.BorderLayout

72

- Le gestionnaire BorderLayout place les composants :
 - soit sur un des bords:
 - Nord (BorderLayout.NORTH)
 - Sud (BorderLayout.SOUTH)
 - Est (BorderLayout.EAST)
 - Ouest (BorderLayout.WEST)
 - soit au centre :
 - Centre (BorderLayout.CENTER)
- On peut librement utiliser une ou plusieurs zones. Lorsque le composant est placé sur les bords, il a une épaisseur fixe. Lorsqu'il est placé au centre, il occupe toute la place qui reste

Gestionnaire de disposition : java.awt.BorderLayout

73

Constructor Summary

[BorderLayout\(\)](#)

Constructs a new border layout with no gaps between components.

[BorderLayout\(int hgap, int vgap\)](#)

Constructs a border layout with the specified gaps between components.

Gestionnaire de disposition : java.awt.BorderLayout

74

- **JFrame** admet par défaut comme gestionnaire de disposition **BorderLayout**. On peut modifier le layout de n'importe quel container (**Jpanel**, **JFrame**, **JApplet**, **Container**, etc. en invoquant la méthode **setLayout**.
 - **jPanel.setLayout(new FlowLayout(FlowLayout.CENTER,10,30));**
 - **jFrame.setLayout(new FlowLayout(FlowLayout.LEFT,10,30));**
 - **jPanel.setLayout(new BorderLayout());**
- Pour choisir l'emplacement d'un composant, il faut fournir un 2^{ème} argument à la méthode **add** qui représente une des constantes suivantes :
 - **BorderLayout.NORTH**
 - **BorderLayout.SOUTH**
 - **BorderLayout.EAST**
 - **BorderLayout.WEST**
 - **BorderLayout.CENTER**

Gestionnaire de disposition : java.awt.BorderLayout

75

```
public class TestBorderLayout extends JFrame{
    public TestBorderLayout() {
        this.setTitle("TestBorderLayout");
        this.setSize(400,400);
        this.setLocationRelativeTo(null);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        Container container = this.getContentPane();
        container.add(new JButton("Center"));
        container.add(new JButton("North"), "North");
        container.add(new JButton("South"),BorderLayout.SOUTH);
        container.add(new JButton("East"),BorderLayout.EAST);
        container.add(new JButton("West"),BorderLayout.WEST);
    }
}
```



Gestionnaire de disposition : java.awt.BorderLayout

76

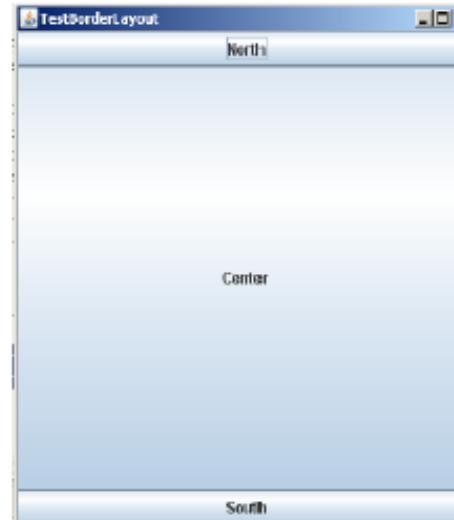
```
public class TestBorderLayout extends JFrame{
    public TestBorderLayout() {
        this.setTitle("TestBorderLayout");
        this.setSize(400,400);
        this.setLocationRelativeTo(null);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        Container container = this.getContentPane();
        container.setLayout(new BorderLayout(20,50));
        container.add(new JButton("Center"));
        container.add(new JButton("North"), "North");
        container.add(new JButton("South"), BorderLayout.SOUTH);
        container.add(new JButton("East"), BorderLayout.EAST);
        container.add(new JButton("West"), BorderLayout.WEST);
    }
}
```



Gestionnaire de disposition : java.awt.BorderLayout

77

```
public class TestBorderLayout extends JFrame{
    public TestBorderLayout() {
        this.setTitle("TestBorderLayout");
        this.setSize(400,400);
        this.setLocationRelativeTo(null);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        Container container = this.getContentPane();
        container.add(new JButton("Center"));
        container.add(new JButton("North"), "North");
        container.add(new JButton("South"), BorderLayout.SOUTH);
    }
}
```



Gestionnaire de disposition : java.awt.GridLayout

78

- Ce Layout Manager établit un réseau de cellules identiques qui forment une sorte de quadrillage invisible : les composants sont organisés en lignes et en colonnes. Les éléments insérés dans la grille ont tous la même taille. Les cellules du quadrillage se remplissent de droite à gauche ou de haut en bas.
- Par défaut, les composants sont espacés de 5 pixels.

Gestionnaire de disposition : java.awt.GridLayout

79

Constructor Summary

GridLayout()

Creates a grid layout with a default of one column per component, in a single row.

GridLayout(int rows, int cols)

Creates a grid layout with the specified number of rows and columns.

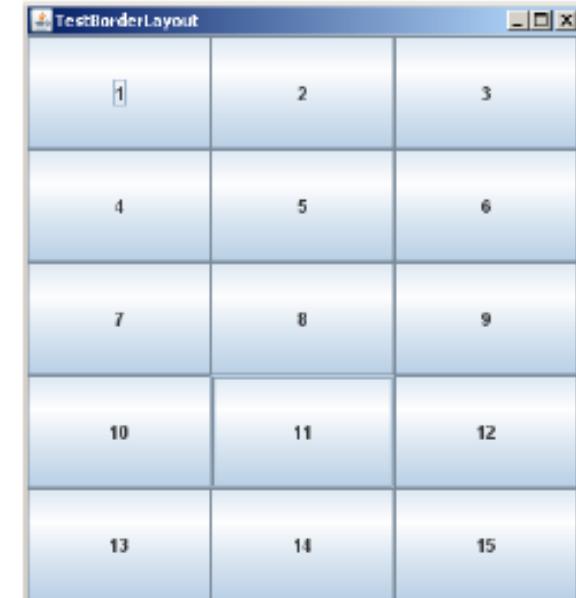
GridLayout(int rows, int cols, int hgap, int vgap)

Creates a grid layout with the specified number of rows and columns.

Gestionnaire de disposition : java.awt.GridLayout

80

```
public class TestGridLayout extends JFrame {  
    private JPanel jPanel;  
    public TestGridLayout() {  
        this.setTitle("TestBorderLayout");  
        this.setSize(400,400);  
        this.setLocationRelativeTo(null);  
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        jPanel = new JPanel();  
        jPanel.setLayout(new GridLayout(5, 3));  
        for(int i=0;i<15;i++)  
        {  
            jPanel.add(new JButton(""+(i+1)));  
        }  
        this.add(jPanel);  
    }  
}
```



Gestionnaire de disposition : java.awt.GridLayout

81

```
public class TestGridLayout extends JFrame {  
    private JPanel jPanel;  
    public TestGridLayout() {  
        this.setTitle("TestBorderLayout");  
        this.setSize(400,400);  
        this.setLocationRelativeTo(null);  
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        jPanel = new JPanel();  
        jPanel.setLayout(new GridLayout(5, 3,20,10));  
        for(int i=0;i<15;i++)  
        {  
            jPanel.add(new JButton(""+(i+1)));  
        }  
        this.add(jPanel);  
    }  
}
```



Gestionnaire de disposition : java.awt.GridLayout

82

```
public class TestGridLayout extends JFrame {  
    private JPanel jPanel;  
    public TestGridLayout() {  
        this.setTitle("TestBorderLayout");  
        this.setSize(400,400);  
        this.setLocationRelativeTo(null);  
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        jPanel = new JPanel();  
        jPanel.setLayout(new GridLayout(5, 3,20,10));  
        for(int i=0;i<13;i++)  
        {  
            jPanel.add(new JButton(""+(i+1)));  
        }  
        this.add(jPanel);  
    }  
}
```



Gestionnaire de disposition : javax.swing.BoxLayout

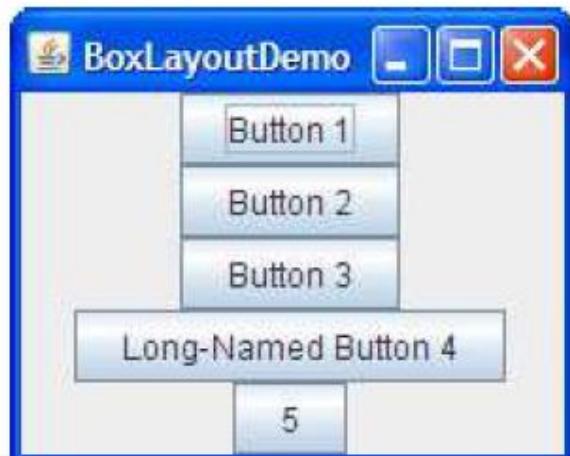
83

- Le gestionnaire BoxLayout dispose des composants suivant une seule ligne ou une seule colonne.

Constructor Summary

[BoxLayout](#)(Container target, int axis)

Creates a layout manager that will lay out components along the given axis.



Gestionnaire de disposition : javax.swing.BoxLayout

84

```
public testBoxLayout () {
    setTitle ("testBoxLayout");
    setLocationRelativeTo (getParent ());
    /////////////
    JLabel l1= new JLabel ("label text");
    JButton b1=new JButton ("Bouton 1");
    JButton b2=new JButton ("Bouton 2");

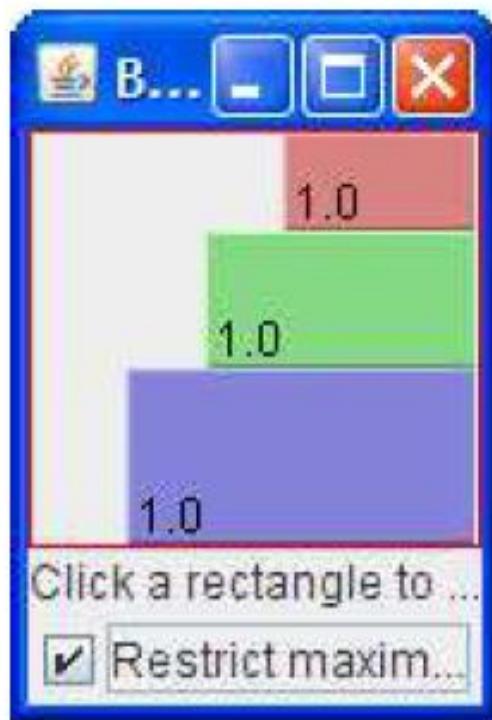
    JPanel pane = new JPanel ();
    setContentPane ( pane );
    //pane.setLayout (new BoxLayout (pane, BoxLayout.X_AXIS)); //horizontal
    pane.setLayout (new BoxLayout (pane, BoxLayout.Y_AXIS)); //vertical
    pane.add (l1);
    pane.add (b1);
    pane.add (b2);

    setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
    pack ();
    show ();
}
```

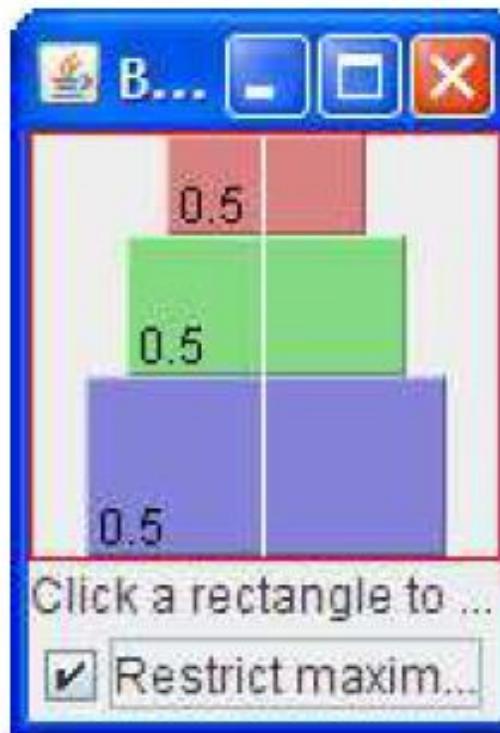
Gestionnaire de disposition : javax.swing.BoxLayout

85

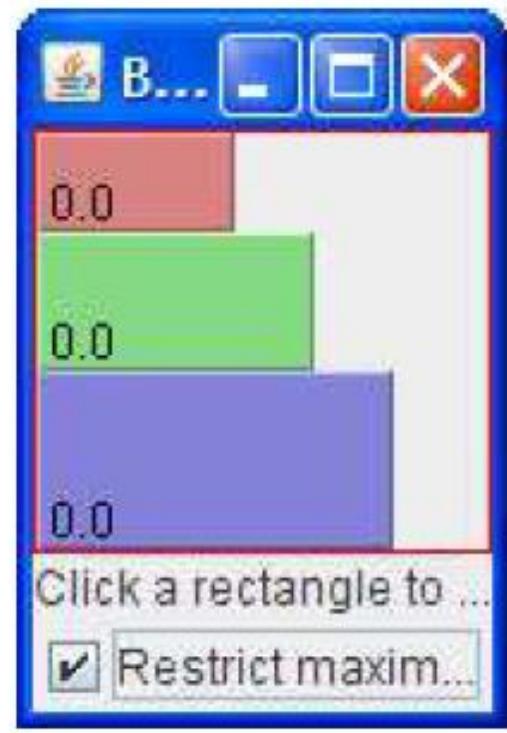
Component.CENTER_ALIGNMENT



Component.RIGHT_ALIGNMENT



Component.LEFT_ALIGNMENT



Gestionnaire de disposition : javax.swing.Box

86

- On peut créer un Box horizontal ou un Box vertical.

```
Box ligne = Box.createHorizontalBox () ; // box  
horizontal
```

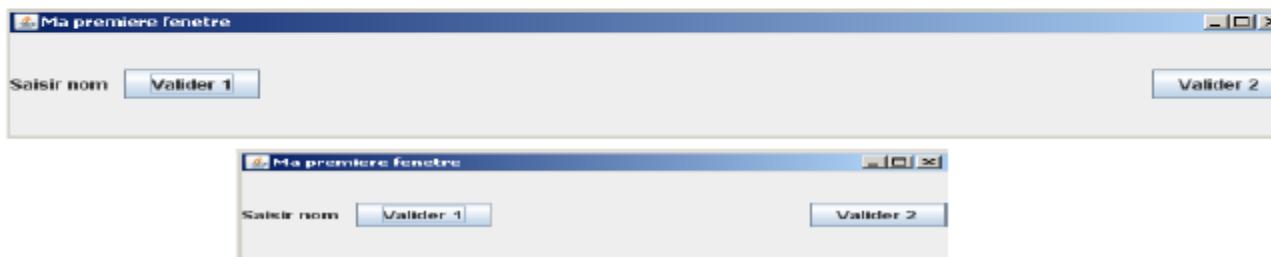
```
Box col = Box.createVerticalBox () ; // box vertical
```

- Pour un box horizontal : Les composants sont mis de gauche à droite sur une même ligne. Si on rétrécit la largeur de la fenêtre, certains composants qui sont à gauche peuvent ne plus apparaître.
- Pour un box vertical : Les composants sont mis de haut en bas sur une même colonne. Si on rétrécit la longueur de la fenêtre, certains composants qui sont en bas peuvent ne plus apparaître.

Gestionnaire de disposition : javax.swing.Box

87

```
Box box = Box.createHorizontalBox();
container.add(box);
jLabel = new JLabel ("Saisir nom");
box.add(jLabel);
box.add(Box.createRigidArea(new Dimension(10,0)));
//crée un composant virtuel représentant un espace vide et fixe de 10 px
jButtonValider = new JButton ("Valider 1");
box.add(jButtonValider);
box.add(Box.createGlue());
/* crée un emplacement virtuel de taille initialisée de façon à espacer au maximum les
composants situés de part et d'autre du glue. Cette taille augmente (ou diminue) lorsqu'on on
agrandit la feuille (ou on la rétrécit)*/
jButtonValider2 = new JButton ("Valider 2");
box.add(jButtonValider2);
```



Gestionnaire de disposition : java.awt.GridBagLayout

88

- **GridBagLayout** répartit ses filles dans une grille, selon les préférences exprimées par chaque fille.
- Chaque fille s'exprime dans un objet **GridBagConstraints**.
- Les contraintes sont ajoutées avec la fille.
- Les "contraintes" d'une fille concernent
 - la **zone** réservée dans la grille
 - la **place** occupée dans cette zone

Gestionnaire de disposition : java.awt.GridBagLayout

89

```
public class testGridBagLayaout extends JFrame{
    public testGridBagLayaout() {
        setTitle("testGridBagL");
        setLocationRelativeTo(getParent());
        JPanel jPanel = new JPanel();
        setContentPane ( jPanel );
        GridBagLayout layout = new GridBagLayout();
        GridBagConstraints lim = new GridBagConstraints();
        jPanel.setLayout(layout);
        JButton b0 = new JButton("B0");
        lim.gridx = 0;
        lim.gridy = 0;
        lim.weightx = 1;
        lim.weighty = 1;
        lim.fill = GridBagConstraints.NONE;
        layout.setConstraints(b0, lim);
        jPanel.add(b0);

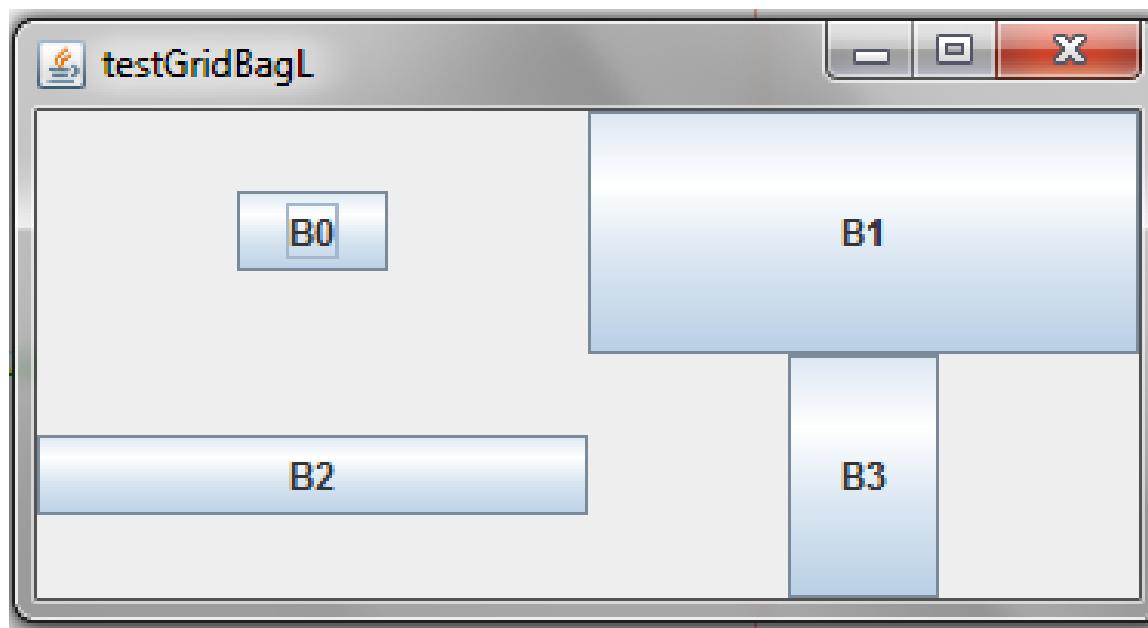
        JButton b1 = new JButton("B1");
        lim.gridx = 1;
        lim.gridy = 0;
        lim.weightx = 1;
        lim.weighty = 1;
        lim.fill = GridBagConstraints.BOTH;
        layout.setConstraints(b1, lim);
        jPanel.add(b1);

        JButton b2 = new JButton("B2");
        lim.gridx = 0;
        lim.gridy = 1;
        lim.weightx = 1;
        lim.weighty = 1;
        lim.fill = GridBagConstraints.HORIZONTAL;
        layout.setConstraints(b2, lim);
        jPanel.add(b2);

        JButton b3 = new JButton("B3");
        lim.gridx = 1;
        lim.gridy = 1;
        lim.weightx = 1;
        lim.weighty = 1;
        lim.fill = GridBagConstraints.VERTICAL;
        layout.setConstraints(b3, lim);
        jPanel.add(b3);
    }
}
```

Gestionnaire de disposition : java.awt.GridBagLayout

90



Gestionnaire de disposition : javax.swing.GroupLayout

91

- L'objectif est de placer des groupes de composants en distinguant le placement vertical et horizontal.
- GroupLayout utilise deux types d'arrangements: séquentiel et parallèle, associés à une composition hiérarchique.
- Avec l'arrangement séquentiel, les composants sont simplement placés les uns après les autres. La position de chaque composant est définie comme étant relative au composant précédent.
- La deuxième méthode place les composants en parallèle, les uns sur les autres, dans le même espace. Ils peuvent être alignés sur la ligne de base, le haut ou le bas le long de l'axe vertical. Sur l'axe horizontal, ils peuvent être alignés à gauche, à droite ou au centre si les composants ne sont pas tous de la même taille

Gestionnaire de disposition : javax.swing.GroupLayout

92

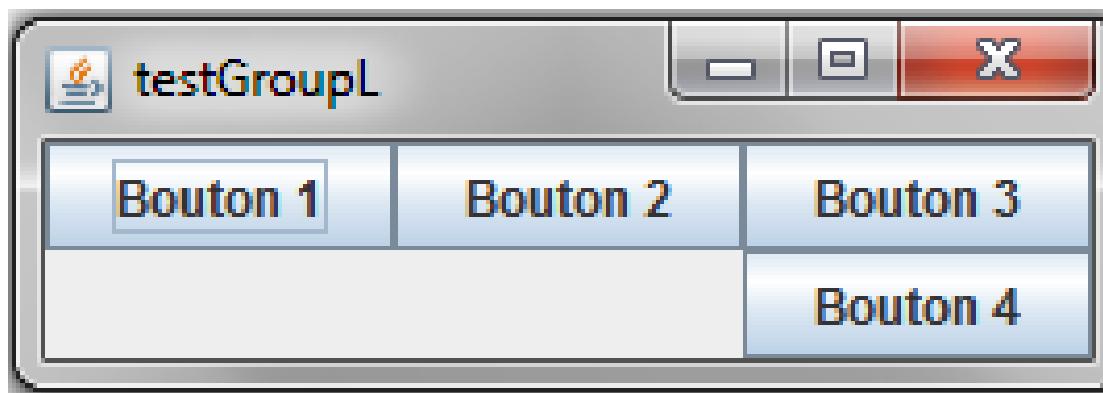
```
GroupLayout layout = new GroupLayout(getContentPane()) ;
jPanel.setLayout(layout);

layout.setHorizontalGroup(
    layout.createSequentialGroup()
    .addComponent(b1)
    .addComponent(b2)
    .addGroup(layout.createParallelGroup(GroupLayout.Alignment.LEADING)
    .addComponent(b3)
    .addComponent(b4))
);
layout.setVerticalGroup(
    layout.createSequentialGroup()
    .addGroup(layout.createParallelGroup(GroupLayout.Alignment.BASELINE)
    .addComponent(b1)
    .addComponent(b2)
    .addComponent(b3))
    .addComponent(b4));
);
```

Gestionnaire de disposition : javax.swing.GroupLayout

93

Sur l'axe horizontal, le nouveau composant occupe le même espace horizontal que b3, de sorte qu'il forme un groupe parallèle avec b3. Sur l'axe vertical, b4 forme un groupe séquentiel avec le groupe parallèle original des trois composants.



Gestionnaire de disposition : java.awt.CardLayout

94

- affiche un seul composant à la fois ; les composants sont affichés à tour de rôle.
- Ce layout manager aide à construire des boîtes de dialogue composées de plusieurs onglets. Un onglet se compose généralement de plusieurs contrôles : on insère des panneaux dans la fenêtre utilisée par le CardLayout Manager. Chaque panneau correspond à un onglet de boîte de dialogue et contient plusieurs contrôles. Par défaut, c'est le premier onglet qui est affiché.
- Ce layout manager est plus rarement utilisé que les autres.

Gestionnaire de disposition : java.awt.CardLayout

95

```
super();
setTitle("Titre de la Fenetre");
setSize(300,150);
CardLayout cl = new CardLayout();
setLayout(cl);

//création d'un panneau contenant les contrôles d'un onglet
Panel p = new Panel();

//ajouter les composants au panel
p.add(new Button("Bouton 1 panneau 1"));
p.add(new Button("Bouton 2 panneau 1"));

//inclure le panneau dans la fenetre sous le nom "Page1"
// ce nom est utilisé par show()

add("Page1",p);

//déclaration et insertion de l'onglet suivant
p = new Panel();
p.add(new Button("Bouton 1 panneau 2"));
add("Page2", p);

// affiche la fenetre
pack();
show();
```

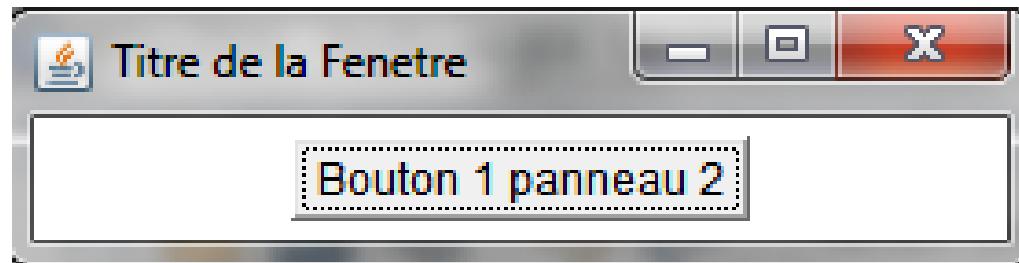


Gestionnaire de disposition : java.awt.CardLayout

96

- Lors de l'insertion d'un onglet, un nom doit lui être attribué. Les fonctions nécessaires pour afficher un onglet de boîte de dialogue ne sont pas fournies par les méthodes du conteneur, mais seulement par le Layout Manager. Il est nécessaire de sauvegarder temporairement le Layout Manager dans une variable où déterminer le gestionnaire en cours par un appel à getLayout(). Pour appeler un onglet donné, il faut utiliser la méthode show() du CardLayout Manager.

```
((CardLayout)getLayout()).show(this, "Page2");
```



Gestionnaire de disposition : java.awt.CardLayout

97

- Les méthodes `first()`, `last()`, `next()` et `previous()` servent à parcourir les onglets de boîte de dialogue :

```
((CardLayout)getLayout()).first(this);
```

Gestionnaire de disposition : null

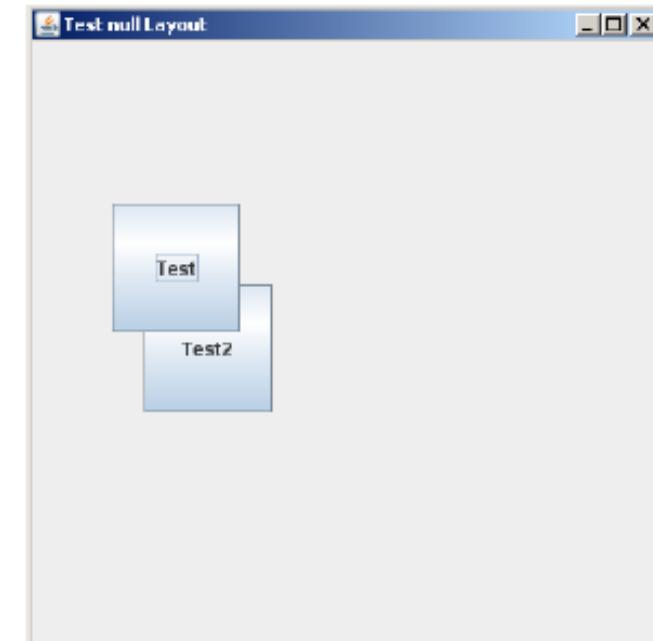
98

- On peut fixer les coordonnées dans l'écran de chaque composant lorsque on l'ajoutes à la fenêtre. Dans ce cas, notre classe doit annoncer explicitement qu'elle n'utilisera pas de gestionnaire de disposition par l'instruction : `setLayout(null);`
- A chaque ajout d'un composant il faut préciser sa taille et son emplacement en utilisant `setSize()`, et `setLocation()` ou tout simplement `setBounds()`

Gestionnaire de disposition : null

99

```
public class TestLayout extends JFrame{
    public TestLayout() {
        this.setTitle("Test null Layout");
        this.setSize(400,400);
        this.setLocationRelativeTo(null);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLayout(null);
        JButton jButton=new JButton("Test");
        jButton.setBounds(50,100, 80, 80);
        JButton jButton2=new JButton("Test2");
        jButton2.setBounds(70,150, 80, 80);
        this.add(jButton);
        this.add(jButton2);
    }
}
```



Plan

100

- **Introduction**
- **Composants SWING**
- **Gestionnaires de disposition**
- **Gestion des événements**

Gestion des événements

101

- La programmation d'interfaces graphiques recourt à la notion d'événement pour représenter les **interactions** entre l'interface et l'utilisateur.
- La gestion d'un événement met en jeu un objet **source** et un objet **écouteur**. L'objet qui a déclenché un événement est appelé source.
- Les événements sont des objets d'une classe dérivée de **java.util.EventObject**. La sous-classe **java.awt.Event** concerne les événements créés par des interactions avec les composantes d'interface : bouton pressé, clic souris, changement de taille des fenêtres, touches du clavier pressées, etc.

Gestion des événements

102

- Quelques unes de ses sous-classes sont : ActionEvent , ItemEvent , MouseEvent , MouseMotionEvent , TextEvent , KeyEvent , FocusEvent , WindowEvent.

- Pour traiter un événement, on associe à la source un **écouteur**. Cet écouteur doit être un objet dont la classe **implémente une interface** particulière correspondant à une catégorie d'événements.

Gestion des événements

103

Il existe :

- une interface correspondante à la catégorie d'événement mouse (événements souris),
- une interface correspondante à la catégorie d'événement key (événements clavier),
- une interface correspondante à la catégorie d'événement action (événements lié aux boutons et à d'autres composants), etc.

Gestion des événements

104

- Par exemple, l'interface correspondante la catégorie d'événement souris s'appelle **MouseListener** et elle déclare 5 méthodes correspondant chacun à un événement de cette catégorie :
 - `public void mouseClicked(MouseEvent e);`
 - `public void mouseEntered(MouseEvent e);`
 - `public void mouseExited(MouseEvent e);`
 - `public void mousePressed(MouseEvent e);`
 - `public void mouseReleased(MouseEvent e);`

```
import java.awt.event.MouseEvent;  
import java.awt.event.MouseListener;
```

Gestion des événements : 1ère démarche

105

- Supposons qu'on veut gérer l'événement clic de souris sur la fenêtre.
- **1ère démarche:** créer une classe qui implémente l'interface **MouseListener** et associer un objet de cette classe à la fenêtre. Cet objet représente l'écouteur de la fenêtre.
- Pour cela, on suivra les étapes suivantes :
 1. **Étape 1** : créer une classe qui implémente l'interface **MouseListener**. Les objets instanciés à partir de cette classe joueront le rôle d'écouteur. On appellera cette classe par exemple **MyJFrameMouseListener**
 2. **Étape 2** : associer un écouteur de type **MyJFrameMouseListener** à la fenêtre sur laquelle on va cliquer. Ceci se fait grâce à la méthode `addMouseListener`.

Gestion des événements : 1ère démarche

106

```
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import javax.swing.JOptionPane;
public class MyJFrameMouseListener implements MouseListener {Étape 1
    public void mouseClicked(MouseEvent e) {
        int x = e.getX();
        int y = e.getY();
        int xOnScreen = e.getXOnScreen();
        int yOnScreen = e.getYOnScreen();
        JOptionPane.showMessageDialog(null, "(x,y)=( " + x + ", " + y
                + ")\n(xOnScreen,yOnScreen)=( " + xOnScreen + ", " +
                yOnScreen
                + ")", "Location", JOptionPane.PLAIN_MESSAGE);
    }
    public void mouseEntered(MouseEvent e) {}
    public void mouseExited(MouseEvent e) {}
    public void mousePressed(MouseEvent e) {}
    public void mouseReleased(MouseEvent e) {}
}
```

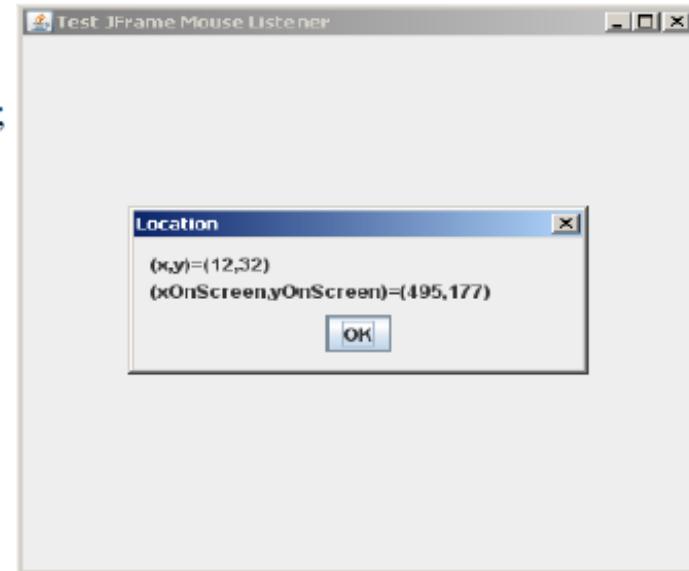
Même si on a besoin que de la méthode mouseClicked, on doit implémenter toutes les méthodes de l'interface MouseListener.

Gestion des événements : 1ère démarche

107

```
package evenement;  
import javax.swing.JFrame;  
public class TestMouseListener extends JFrame {  
    public TestMouseListener() {  
        super("Test JFrame Mouse Listener");  
        this.setSize(400, 400);  
        this.setLocationRelativeTo(null);  
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        this.addMouseListener(new MyJFrameMouseListener());  
    }  
    public static void main(String[] args) {  
        new TestMouseListener().setVisible(true);  
    }  
}
```

Étape 2



Gestion des événements : 2ème démarche

108

- **2ème démarche** : la fenêtre peut être elle-même son propre écouteur. Dans ce cas la fenêtre implémentera l'interface `MouseListener`, puis on associe l'objet courant (la fenêtre courante) à la fenêtre pour la désigner comme son propre écouteur.
- Pour cela, on suivra les étapes suivantes :
 1. **Étape 1** : changer la classe qui représente la fenêtre pour qu'elle implemente l'interface `MouseListener`
 2. **Étape 2** : considérer la fenêtre comme son propre écouteur

Gestion des événements : 2ème démarche

109

```
public class TestMouseListener2 extends JFrame implements MouseListener {  
    public TestMouseListener2() {  
        super("Test JFrame Mouse Listener");  
        this.setSize(400, 400);  
        this.setLocationRelativeTo(null);  
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    }  
    public void mouseClicked(MouseEvent e) {  
        int x = e.getX(); int xOnScreen = e.getXOnScreen();  
        int y = e.getY(); int yOnScreen = e.getYOnScreen();  
        JOptionPane.showMessageDialog(null, "(x,y)=( " + x + ", " + y  
                + " )\n(xOnScreen,yOnScreen)=( " + xOnScreen + ", " +  
                yOnScreen  
                + " )", "Location", JOptionPane.PLAIN_MESSAGE);  
    }  
    public void mouseEntered(MouseEvent e) {}  
    public void mouseExited(MouseEvent e) {}  
    public void mousePressed(MouseEvent e) {}  
    public void mouseReleased(MouseEvent e) {}  
}
```

Étape 1

Gestion des événements : 2ème démarche

110

```
public class TestMouseListener2 extends JFrame implements MouseListener {  
    public TestMouseListener2() {  
        super("Test JFrame Mouse Listener");  
        this.setSize(400, 400);  
        this.setLocationRelativeTo(null);  
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        this.addMouseListener(this);  
    }  
    public void mouseClicked(MouseEvent e) {  
        int x = e.getX(); int xOnScreen = e.getXOnScreen();  
        int y = e.getY(); int yOnScreen = e.getYOnScreen();  
        JOptionPane.showMessageDialog(null, "(x,y)=( " + x + ", " + y  
                + " )\n(xOnScreen,yOnScreen)=( " + xOnScreen + ", " +  
                yOnScreen  
                + " )", "Location", JOptionPane.PLAIN_MESSAGE);  
    }  
    public void mouseEntered(MouseEvent e) {}  
    public void mouseExited(MouseEvent e) {}  
    public void mousePressed(MouseEvent e) {}  
    public void mouseReleased(MouseEvent e) {}  
}
```

Étape 2

Gestion des événements : 3ème démarche

111

- **3ème démarche** : utiliser une classe prédéfinie appelée **MouseAdapter** qui **implémente** déjà l'interface **MouseListener**. Cette classe contient toutes les méthodes de l'interface avec une implémentation vide ({}). Il suffit donc de créer un classe écouteur qui **hérite** de **MouseAdapter** et qui permet de redéfinir seulement la méthode qu'on va utiliser : **MouseClicked**.
- Pour cela, on suivra les étapes suivantes :
 1. **Étape 1** : créer une classe qui hérite de **MouseAdapter**
 2. **Étape 2** : associer un écouteur de type **MyJFrameMouseListener2** à la fenêtre sur laquelle on va cliquer. Ceci se fait grâce à la méthode **addMouseListener**

Gestion des événements : 3ème démarche

112

```
package evenement;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import javax.swing.JOptionPane;
public class MyJFrameMouseListener2 extends MouseAdapter {
    public void mouseClicked(MouseEvent e) {
        int x = e.getX();
        int y = e.getY();
        int xOnScreen = e.getXOnScreen();
        int yOnScreen = e.getYOnScreen();
        JOptionPane.showMessageDialog(null, "(x,y)=( " + x + ", " + y
                + " )\n(xOnScreen,yOnScreen)=( " + xOnScreen +
                ", " + yOnScreen
                + " )", "Location",
        JOptionPane.PLAIN_MESSAGE);
    }
}
```

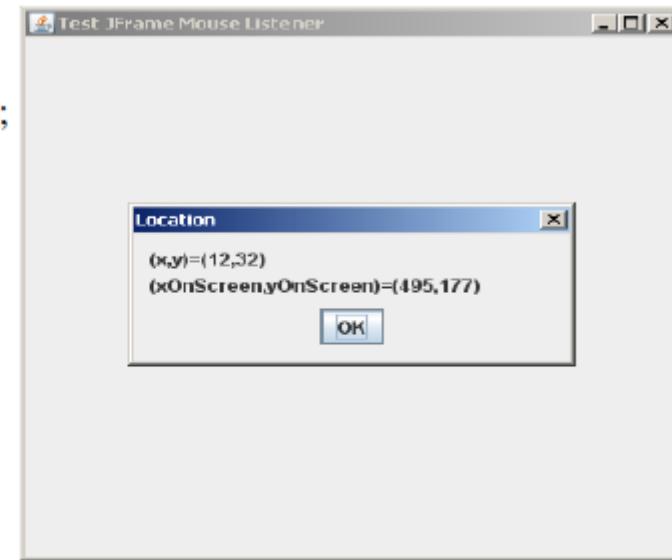
Étape 1

Gestion des événements : 3ème démarche

113

```
package evenement;
import javax.swing.JFrame;
public class TestMouseListener extends JFrame {
    public TestMouseListener() {
        super("Test JFrame Mouse Listener ");
        this.setSize(400, 400);
        this.setLocationRelativeTo(null);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.addMouseListener(new MyJFrameMouseListener2());
    }
    public static void main(String[] args) {
        new TestMouseListener().setVisible(true);
    }
}
```

Étape 2



Gestion des événements : 4ème démarche

114

- 4ème démarche : créer **une classe anonyme** qui hérite de **MouseAdapter**.
- Les classes **internes anonymes (anonymous inner-classes)** sont des classes internes qui ne possèdent pas de nom. Elles ne peuvent donc être instanciées qu'à l'endroit où elles sont définies.
- Ce type de classe est très pratique lorsqu'une classe doit être utilisée une seule fois : c'est par exemple le cas d'une classe qui doit être utilisée comme un callback.
- Une syntaxe particulière de l'opérateur new permet de déclarer et instancier une classe interne :

```
new ClassOrInterface (){  
    // définition des attributs et des méthodes de la classe interne  
}
```

Gestion des événements : 4ème démarche

115

```
public class TestMouseListener4 extends JFrame {  
    public TestMouseListener4() {  
        super("Test JFrame Mouse Listener");  
        this.setSize(400, 400);  
  
        this.addMouseListener(new MouseAdapter() {  
            public void mouseClicked(MouseEvent e) {  
                int x = e.getX(); int xOnScreen = e.getXOnScreen();  
                int y = e.getY(); int yOnScreen = e.getYOnScreen();  
  
                JOptionPane.showMessageDialog(null, "(x,y)=( " + x + "," + y+ ")\n" +  
                    "(xOnScreen,yOnScreen)=( " + xOnScreen + "," + yOnScreen  
                    + ")", "Location", JOptionPane.PLAIN_MESSAGE);  
            }  
        });  
    }  
}
```

Classe interne anonyme

Gestion des événements : ActionListener

116

java.awt.event

Interface ActionListener

All Superinterfaces:

[EventListener](#)

All Known Subinterfaces:

[Action](#)

Method Summary

void [actionPerformed\(ActionEvent e\)](#)

Invoked when an action occurs.

Gestion des événements : MouseListener

117

java.awt.event

Interface MouseListener

All Superinterfaces:

[EventListener](#)

All Known Subinterfaces:

[MouseInputListener](#)

Method Summary

void [mouseClicked\(MouseEvent e\)](#)

Invoked when the mouse button has been clicked (pressed and released) on a component.

void [mouseEntered\(MouseEvent e\)](#)

Invoked when the mouse enters a component.

void [mouseExited\(MouseEvent e\)](#)

Invoked when the mouse exits a component.

void [mousePressed\(MouseEvent e\)](#)

Invoked when a mouse button has been pressed on a component.

void [mouseReleased\(MouseEvent e\)](#)

Invoked when a mouse button has been released on a component.

Gestion des événements : MouseMotionListener

118

java.awt.event

Interface MouseMotionListener

All Superinterfaces:

[EventListener](#)

All Known Subinterfaces:

[MouseListener](#)

Method Summary

`void mouseDragged\(MouseEvent e\)`

Invoked when a mouse button is pressed on a component and then dragged.

`void mouseMoved\(MouseEvent e\)`

Invoked when the mouse cursor has been moved onto a component but no buttons have been pushed.

Gestion des événements : MouseWheelListener

119

`java.awt.event`

Interface MouseWheelListener

All Superinterfaces:
[EventListener](#)

Method Summary

`void mouseWheelMoved(MouseWheelEvent e)`

Invoked when the mouse wheel is rotated.

Gestion des événements : KeyListener

120

java.awt.event

Interface KeyListener

All Superinterfaces:

[EventListener](#)

Method Summary

`void keyPressed(KeyEvent e)`

Invoked when a key has been pressed.

`void keyReleased(KeyEvent e)`

Invoked when a key has been released.

`void keyTyped(KeyEvent e)`

Invoked when a key has been typed.

Gestion des événements : ItemListener

121

java.awt.event

Interface ItemListener

All Superinterfaces:

[EventListener](#)

Method Summary

void **itemStateChanged**(ItemEvent e)

Invoked when an item has been selected or deselected by the user.

Gestion des événements : FocusListener

122

java.awt.event

Interface FocusListener

All Superinterfaces:

[EventListener](#)

Method Summary

`void focusGained(FocusEvent e)`

Invoked when a component gains the keyboard focus.

`void focusLost(FocusEvent e)`

Invoked when a component loses the keyboard focus.

Gestion des événements : WindowListener

123

java.awt.event

Interface WindowListener

All Superinterfaces:
[EventListener](#)

Method Summary

`void windowActivated(WindowEvent e)`

Invoked when the Window is set to be the active Window.

`void windowClosed(WindowEvent e)`

Invoked when a window has been closed as the result of calling dispose on the window.

`void windowClosing(WindowEvent e)`

Invoked when the user attempts to close the window from the window's system menu.

`void windowDeactivated(WindowEvent e)`

Invoked when a Window is no longer the active Window.

`void windowDeiconified(WindowEvent e)`

Invoked when a window is changed from a minimized to a normal state.

`void windowIconified(WindowEvent e)`

Invoked when a window is changed from a normal to a minimized state.

`void windowOpened(WindowEvent e)`

Invoked the first time a window is made visible.

Gestion des événements : ListSelectionListener

124

javax.swing.event

Interface ListSelectionListener

All Superinterfaces:

[EventListener](#)

Method Summary

`void valueChanged(ListSelectionEvent e)`

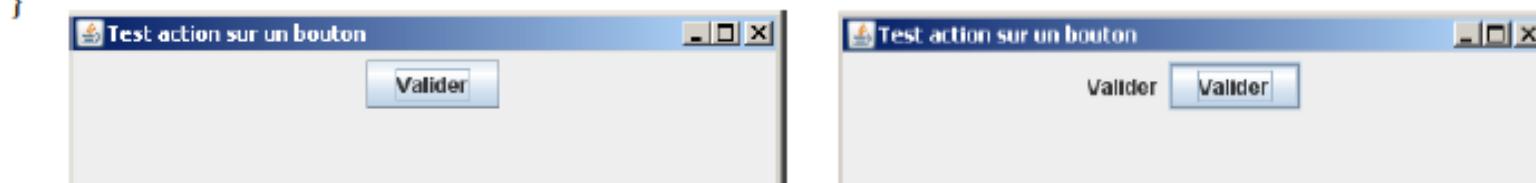
Called whenever the value of the selection changes.

Gestion des événements : Exemple 1

125

```
public class TestActionListener extends JFrame implements ActionListener{
    private JButton jButtonValider;
    private JLabel jLabelTest;
    public TestActionListener() {
        super("Test action sur un bouton");
        this.setSize(400,100);
        JPanel jPanelMain=new JPanel();
        jLabelTest = new JLabel();
        jButtonValider = new JButton("Valider");
        jButtonValider.addActionListener(this);

        jPanelMain.add(jLabelTest);
        jPanelMain.add(jButtonValider);
        this.getContentPane().add(jPanelMain);
    }
    public void actionPerformed(ActionEvent e) {
        jLabelTest.setText(jButtonValider.getText());
    }
}
```



Gestion des événements : Exemple 2

126

```
public class TestActionListener extends JFrame implements ActionListener{  
    private JButton jButtonValider;  
    private JLabel jLabelTest;  
    public TestActionListener() {  
        super("Test action sur un bouton");  
        this.setSize(400,100);  
        JPanel jPanelMain=new JPanel();  
        jLabelTest = new JLabel();  
        jButtonValider = new JButton("Valider");  
        jButtonValider.addActionListener(this);  
        jPanelMain.add(jLabelTest);  
        jPanelMain.add(jButtonValider);  
        this.getContentPane().add(jPanelMain);  
    }  
    public void actionPerformed(ActionEvent e) {  
        jLabelTest.setText(e.getActionCommand());  
    }  
}
```



- La méthode **getActionCommand()** : cette méthode permet de retourner la chaîne de commande (sous forme d'une chaîne de caractères) associée au bouton qui a déclenché l'événement. Par défaut cette chaîne est l'étiquette du bouton.

Gestion des événements : Exemple 3

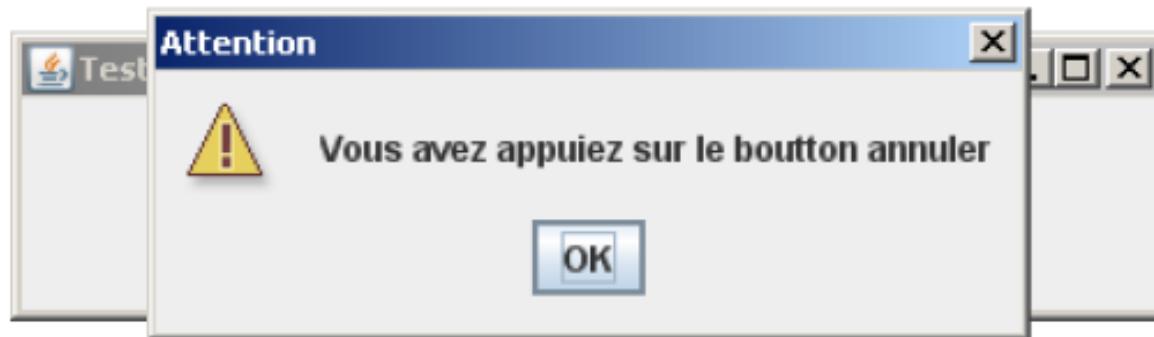
127

```
public class TestActionListener extends JFrame implements ActionListener{  
    private JButton jButtonValider;  
    private JLabel jLabelTest;  
    private JButton jButtonAnnuler;  
    public TestActionListener() {  
        jButtonAnnuler = new JButton("Annuler");  
        jButtonAnnuler.addActionListener(this);  
        jPanelMain.add(jButtonAnnuler);  
    }  
    public void actionPerformed(ActionEvent e) {  
        Object source=e.getSource();  
        if(source instanceof JButton)  
        {  
            JButton temp=(JButton) source;  
            if(temp.equals(jButtonValider))  
                jLabelTest.setText(jButtonValider.getText());  
            else if(temp.equals(jButtonAnnuler))  
                JOptionPane.showMessageDialog(TestActionListener.this,  
"Vous avez appuyez sur le boutton annuler","Attention",JOptionPane.WARNING_MESSAGE);  
        }  
    }  
}
```

La méthode `getSource()` permet de retourner une référence sur l'objet qui a déclenché l'événement *Action*.

Gestion des événements : Exemple 3

128



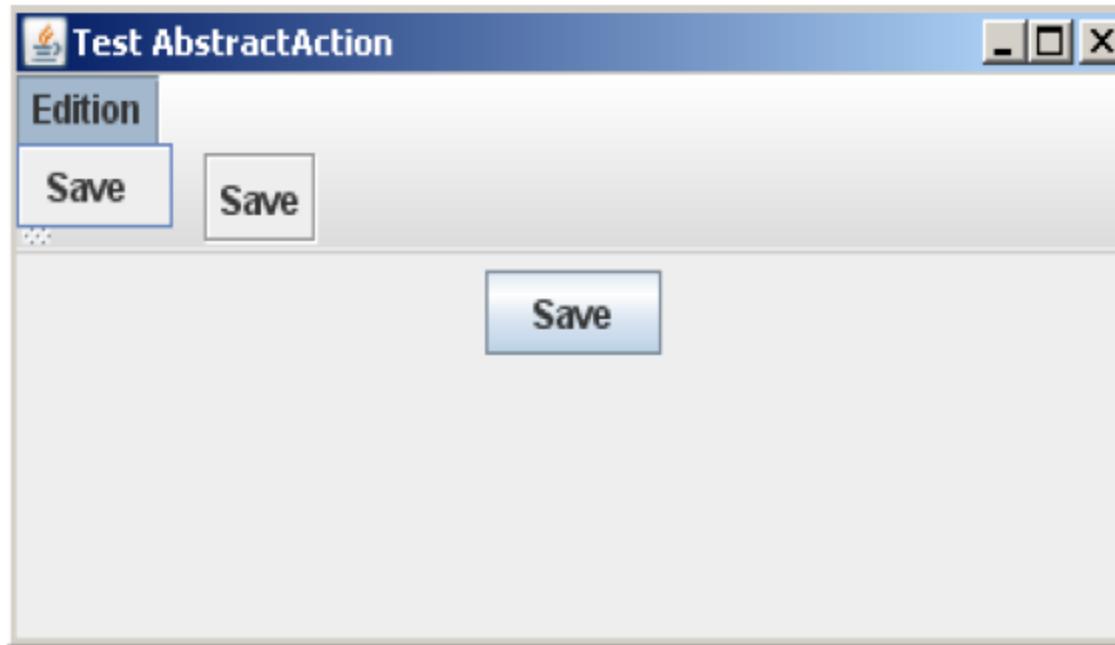
Gestion des événements : AbstractAction

129

- **AbstractAction** : Cette classe représente une action de Swing.
- On peut lui donner un texte et une icône et on peut évidemment lui dire que faire en cas de clic dans la méthode `actionPerformed`.
- Dans une application, une même action peut être déclencher à la fois par plusieurs manières (Ex. la fonctionnalité copier dans Word). Créer une seule fois le comportement voulu puis l'associer par exemple à une option d'un menu déroulant ou à une option d'un menu surgissant ou un bouton.

Gestion des événements : AbstractAction

130



Gestion des événements : AbstractAction

131

```
public class TestAbstractAction extends JFrame{
    public TestAbstractAction() {
        jMenuBar = new JMenuBar();
        jToolBar = new JToolBar();
        jMenuEdition = new JMenu("Edition");
        jMenuItemSave = new JMenuItem();
        jMenuBar.add(jMenuEdition);
        jMenuEdition.add(jMenuItemSave);
        this.setJMenuBar(jMenuBar);
        jPanel = new JPanel();
        abstractActionSave = new AbstractAction("Save") {
            public void actionPerformed(ActionEvent arg0) {
                System.out.println("Save");
            }
        };
        jButtonSave = new JButton(abstractActionSave);
        jMenuItemSave.setAction(abstractActionSave);
        jToolBar.addSeparator(new Dimension(50,10));
        jToolBar.add(abstractActionSave);
        jPanel.add(jButtonSave);
        this.add(jToolBar,"North");
        this.add(jPanel);
    }
}
```