Table des matières

[1 CONCEPTS BASIQUES DE L’INTERFACE GRAPHIQUE 2](#_Toc25920578)

[1.1 Une fenêtre simple 2](#_Toc25920579)

[1.1.1 GERER UN EVENEMENT 4](#_Toc25920580)

[1.1.2 UN FORMULAIRE AVEC BOUTON 8](#_Toc25920581)

[1.1.3 LES GESTIONNAIRES D'EVENEMENTS 13](#_Toc25920582)

[1.1.4 LES METHODES DES GESTIONNAIRES D'EVENEMENTS 15](#_Toc25920583)

[1.1.5 LES CLASSES ADAPTATEURS 16](#_Toc25920584)

[1.1.6 CONCLUSION 17](#_Toc25920585)

[1.2 LES GESTIONNAIRES DE PLACEMENT DES COMPOSSANTS. 18](#_Toc25920586)

[1.2.1 Introduction 18](#_Toc25920587)

[1.2.2 La classe FlowLayout 18](#_Toc25920588)

[1.2.3 Le gestionnaire BorderLayout 19](#_Toc25920589)

[1.2.4 La classe BoxLayout 20](#_Toc25920590)

[1.2.5 La classe CardLayout 22](#_Toc25920591)

[1.2.6 Le gestionnaire GridLayout 23](#_Toc25920592)

[1.2.7   Le gestionnaire GridBagLayout 25](#_Toc25920593)

[1.2.8 Le gestionnaire SpringLayout 30](#_Toc25920594)

[1.2.9 Le gestionnaire GroupLayout 33](#_Toc25920595)

[1.3 Les composants Swing 34](#_Toc25920596)

[1.3.1 La classe JFrame 34](#_Toc25920597)

[1.3.2 Les étiquettes : la classe JLabel 36](#_Toc25920598)

[1.3.3 Les panneaux : la classe Jpanel 38](#_Toc25920599)

[1.3.4 Les boutons 39](#_Toc25920600)

[1.3.5 Les composants de saisie de texte 44](#_Toc25920601)

[1.3.6 Les menus 49](#_Toc25920602)

[1.3.7 La classe Jtable 56](#_Toc25920603)

# CONCEPTS BASIQUES DE L’INTERFACE GRAPHIQUE

## Une fenêtre simple

// classes importées

import javax.swing.\*;

import java.awt.\*;

// la classe formulaire

public class form1 extends JFrame {

// le constructeur

public form1() {

// titre de la fenêtre

this.setTitle("Mon premier formulaire");

// dimensions de la fenêtre

this.setSize(new Dimension(300,100));

}//constructeur

// fonction de test

public static void main(String[] args) {

// on affiche le formulaire

new form1().setVisible(true);

}

}//classe

***Explication du code :***

Une interface graphique dérive en général de la classe de base *JFrame*

public class form1 extends JFrame {

}

La classe de base *JFrame* définit une fenêtre de base avec des boutons de fermeture, agrandissement/réduction, une taille ajustable, etc .. et gère les événements sur ces objets graphiques. Ici on spécialise la classe de base en lui fixant un titre et ses largeurs (300 pixels) et hauteur (100 pixels). Ceci est fait dans son constructeur :

// le constructeur

*public form1() {*

// titre de la fenêtre

this.setTitle("Mon premier formulaire");

// dimensions de la fenêtre

this.setSize (new Dimension(300,100));

0

}//constructeur

Le titre de la fenêtre est fixée par la méthode *setTitle* et ses dimensions par la méthode *setSize*. Cette méthode accepte pour paramètre un objet *Dimension (largeur, hauteur)* où *largeur* et *hauteur* sont les largeur et hauteur de la fenêtre exprimées en pixels. La méthode *main* lance l'application graphique de la façon suivante :

new form1 ().setVisible (true);

Un formulaire de type f*orm1* est alors créé (*new form1()*) et affiché (*setVisible(true)*), puis l'application se met à l'écoute des événements qui se produisent sur le formulaire (clics, déplacements de souris, etc...) et fait exécuter ceux que le formulaire gère. Ici, notre formulaire ne gère pas d'autres événements que ceux gérés par la classe de base *JFrame* (clics sur boutons fermeture, agrandissement/réduction, changement de taille de la fenêtre, déplacement de la fenêtre, ...).

L'exécution du programme se fait de la façon suivante :

- au départ, un premier thread d'exécution est lancé pour exécuter la méthode *main*

- lorsque celle-ci crée le formulaire et l'affiche, un second thread est créé pour gérer spécifiquement les événements liés au formulaire

- après cette création et dans notre exemple, le thread de la méthode *main* se termine et seul reste alors le thread d'exécution de l'interface graphique.

- lorsqu'on ferme la fenêtre, celle-ci disparaît mais n'interrompt pas le thread dans lequel elle s'exécutait

- on est obligé pour l'instant d'arrêter ce thread en faisant Ctrl-C dans la fenêtre Dos d'où a été lancée l'exécution du programme.

### GERER UN EVENEMENT

Dans l'exemple précédent, il nous faudrait gérer la fermeture de la fenêtre nous-mêmes pour que lorsqu'elle se produit on arrête l'application, ce qui pour l'instant n'est pas fait. Pour cela il nous faut créer un objet qui "écoute" les événements qui se produisent sur la fenêtre et détecte l'événement "fermeture de la fenêtre". On appelle cet objet un "listener" ou gestionnaire d'événements. Il existe différents types de gestionnaires pour les différents événements qui peuvent se produire sur les composants d'une interface graphique. Pour le composant *JFrame*, le listener s'appelle *WindowListener* et est une interface définissant les méthodes suivantes :

* void **windowActivated**(WindowEvent e) La fenêtre devient la fenêtre active;
* void **windowClosed**(WindowEvent e) La fenêtre a été fermée ;
* void **windowClosing**(WindowEvent e) L'utilisateur ou le programme a demandé la fermeture de la fenêtre ;
* void **windowDeactivated**(WindowEvent e) La fenêtre n'est plus la fenêtre active ;
* void **windowDeiconified**(WindowEvent e) La fenêtre passe de l'état réduit à l'état normal
* void **windowIconified**(WindowEvent e) La fenêtre passe de l'état normal à l'état réduit
* void **windowOpened**(WindowEvent e) La fenêtre devient visible pour la première fois

Il y a donc sept événements qui peuvent être gérés. Les gestionnaires reçoivent tous en paramètre un objet de type *WindowEvent* que nous ignorons pour l'instant. L'événement qui nous intéresse ici est la fermeture de la fenêtre, événement qui devra être traité par la

méthode *windowClosing*. Pour gérer cet événement, on pourra créer un objet *Windowlistener* à l'aide d'une classe anonyme de la façon suivante :

// création d'un gestionnaire d'événements

WindowListener win=new WindowListener(){

public void windowActivated(WindowEvent e){}

public void windowClosed(WindowEvent e){}

**public void windowClosing(WindowEvent e){**

**System.exit(0);**

**}**

public void windowDeactivated(WindowEvent e){}

public void windowDeiconified(WindowEvent e){}

public void windowIconified(WindowEvent e){}

public void windowOpened(WindowEvent e){}

};//définition win

Notre gestionnaire d'événements implémentant l'interface *WindowListener* doit définir les sept méthodes de cette interface. Comme nous ne voulons gérer que la fermeture de la fenêtre, nous ne définissons du code que pour la méthode **windowClosing**. Lorsque les autres événements se produiront, nous en serons avertis mais nous ne ferons rien. Que ferons-nous lorsqu'on sera averti que la fenêtre est en cours de fermeture (*windowClosing*) ? Nous arrêterons l'application :

**public void windowClosing(WindowEvent e){System.exit(0);}**

Nous avons là un objet capable de gérer les événements d'une fenêtre en général. Comment l'associe-t-on à une fenêtre particulière? La classe *JFrame* a une méthode *addWindowListener(WindowListener win)* qui permet d'associer un gestionnaire d'événements

"fenêtre" à une fenêtre donnée. Ainsi ici, dans le constructeur de la fenêtre nous écrirons :

// création d'un gestionnaire d'événements

WindowListener win=new WindowListener(){

public void windowActivated(WindowEvent e){}

public void windowClosed(WindowEvent e){}

public void windowClosing(WindowEvent e){System.exit(0);}

public void windowDeactivated(WindowEvent e){}

public void windowDeiconified(WindowEvent e){}

public void windowIconified(WindowEvent e){}

public void windowOpened(WindowEvent e){}

};//définition win

*// ce gestionnaire d'événements va gérer les évts de la fenêtre courante*

*this.addWindowListener(win);*

Le programme complet est le suivant :

// classes importées

import javax.swing.\*;

import java.awt.\*;

import java.io.\*;

import java.awt.event.\*;

// la classe formulaire

public class form2 extends JFrame {

// le constructeur

public form2() {

// titre de la fenêtre

this.setTitle("Mon premier formulaire");

// dimensions de la fenêtre

this.setSize(new Dimension(300,100));

// création d'un gestionnaire d'événements

WindowListener win=new WindowListener(){

public void windowActivated(WindowEvent e){}

public void windowClosed(WindowEvent e){}

public void windowClosing(WindowEvent e){System.exit(0);}

public void windowDeactivated(WindowEvent e){}

public void windowDeiconified(WindowEvent e){}

public void windowIconified(WindowEvent e){}

public void windowOpened(WindowEvent e){}

};//définition win

// ce gestionnaire d'événements va gérer les évts de la fenêtre courante

this.addWindowListener(win);

}//constructeur

// fonction de test

public static void main(String[] args) {

// on affiche le formulaire

new form2().setVisible(true);

}//main

}//classe

Le paquetage *java.awt.event* contient l'interface *WindowListener*. Lorsqu'on exécute ce programme et qu'on ferme la fenêtre qui s'est affichée on constate dans la fenêtre Dos où a été lancé le programme, la fin d'exécution du programme ce qu'on n'avait pas auparavant.

Dans notre programme, la création de l'objet chargé de gérer les événements de la fenêtre est un peu lourde dans la mesure où on est obligé de définir des méthodes même pour des événements qu'on ne veut pas gérer. Dans ce cas, au lieu d'utiliser l'**interface** *WindowListener* on peut utiliser la **classe *WindowAdapter***.

Celle-ci implémente l'interface *WindowListener*, avec sept méthodes vides. En dérivant la classe *WindowAdapter* et en redéfinissant les seules méthodes qui nous intéressent, nous arrivons au même résultat qu'avec l'interface *WindowListener* mais sans avoir besoin de définir les méthodes qui ne nous intéressent pas. La séquence

- définition du gestionnaire d'événements

- association du gestionnaire à la fenêtre peut se faire de la façon suivante

dans notre exemple :

// création d'un gestionnaire d'événements

WindowAdapter win=new WindowAdapter(){

public void windowClosing(WindowEvent e){System.exit(0);}

};//définition win

// ce gestionnaire d'événements va gérer les évts de la fenêtre courante

this.addWindowListener(win);

Nous utilisons ici une classe anonyme qui dérive la classe *WindowAdapter* et redéfinit sa méthode *windowClosing*. Le programme devient alors :

// classes importées

import javax.swing.\*;

import java.awt.\*;

import java.io.\*;

import java.awt.event.\*;

// la classe formulaire

public class form2 extends JFrame {

// le constructeur

public form2() {

// titre de la fenêtre

this.setTitle("Mon premier formulaire");

Interfaces graphiques 102

// dimensions de la fenêtre

this.setSize(new Dimension(300,100));

// création d'un gestionnaire d'événements

WindowAdapter win=new WindowAdapter(){

public void windowClosing(WindowEvent e){System.exit(0);

}

};//définition win

// ce gestionnaire d'événements va gérer les évts de la fenêtre courante

this.addWindowListener(win);

}//fin constructeur

// fonction de test

public static void main(String[] args) {

// on affiche le formulaire

new form2().setVisible(true);

}//main

}//classe

Il donne les mêmes résultats que le précédent programme mais est plus simple d'écriture.

### UN FORMULAIRE AVEC BOUTON

Ajoutons maintenant un bouton à notre fenêtre :

// classes importées

import javax.swing.\*;

import java.awt.\*;

import java.io.\*;

import java.awt.event.\*;

// la classe formulaire

public class form3 extends JFrame {

// un bouton

JButton btnTest=null;

Container conteneur=null;

// le constructeur

public form3() {

// titre de la fenêtre

this.setTitle("Formulaire avec bouton");

// dimensions de la fenêtre

this.setSize(new Dimension(300,100));

// création d'un gestionnaire d'événements

WindowAdapter win=new WindowAdapter(){

public void windowClosing(WindowEvent e){

System.exit(0);

}

};//définition win

// ce gestionnaire d'événements va gérer les évts de la fenêtre courante

this.addWindowListener(win);

// on récupère le conteneur de la fenêtre

conteneur=this.getContentPane();

// on choisit un gestionnaire de mise en forme des composants dans ce conteneur

conteneur.setLayout(new FlowLayout());

// on crée un bouton

btnTest=new JButton();

// on fixe son libellé

btnTest.setText("Test");

// on ajoute le bouton au conteneur

conteneur.add(btnTest);

}//constructeur

// fonction de test

public static void main(String[] args) {

// on affiche le formulaire

new form3().setVisible(true);

}//main

}//classe

**Quelques explications du code ci-haut :**

Une fenêtre *JFrame* a un conteneur dans lequel on peut déposer des composants graphiques (bouton, cases à cocher, listes déroulantes, ...). Ce conteneur est disponible via la méthode *getContentPane* de la classe *JFrame* :

***Container conteneur=null;***

// on récupère le conteneur de la fenêtre

***conteneur=this.getContentPane();***

Tout composant est placé dans le conteneur par la méthode *add* de la classe *Container* . Ainsi pour déposer le composant C dans l'objet *conteneur* précédent, on écrira :

***conteneur.add(C) ;***

Où ce composant est-il placé dans le conteneur ? Il existe divers gestionnaires de disposition de composants portant le nom **XXXLayout**, avec XXX égal par exemple à *Border*, *Flow*, ... Chaque gestionnaire de disposition a ses particularités.

Par exemple, le gestionnaire *FlowLayout* dispose les composants en ligne en commençant par le haut du formulaire. Lorsqu'une ligne a été remplie, les composants sont placés sur la ligne suivante. Pour associer un gestionnaire de mise en forme à une fenêtre *JFrame*, on utilise la méthode *setLayout* de la classe *JFrame* sous la forme :

SetLayout (objet XXXLayout); Ainsi dans notre exemple, pour associer un gestionnaire de type *FlowLayout* à la fenêtre, on a écrit :

// on choisit un gestionnaire de mise en forme des composants dans ce conteneur

***conteneur.setLayout(new FlowLayout());***

On peut ne pas utiliser de gestionnaire de disposition et écrire :

***setLayout(null);***

Dans ce cas, on devra donner les coordonnées précises du composant dans le conteneur sous la forme (x,y,largeur,hauteur) où (x,y) sont les coordonnées du coin supérieur gauche du composant dans le conteneur. C'est cette méthode que nous utiliserons le plus souvent par la suite. On sait maintenant comment sont déposés les composants au conteneur (*add*) et où (*setLayout*). Il nous reste à connaître les composants qu'on peut placer dans un conteneur. Ici nous plaçons un bouton modélisé par la classe *javax.swing.JButton* :

JButton btnTest=null;

..........

// on crée un bouton

***btnTest=new JButton();***

// on fixe son libellé

***btnTest.setText("Test");***

// on ajoute le bouton au conteneur

***conteneur.add(btnTest);***

Si on clique sur le bouton *Test*, il ne se passe rien. En effet, nous n'avons pas encore associé de gestionnaire d'événements au bouton. Pour connaître le type de gestionnaires d'événements disponibles pour un composant donné, on peut chercher dans la définition de sa classe des méthodes *addXXXListener* qui permettent d'associer au composant un gestionnaire d'événements. La classe *javax.swing.JButton* dérive de la classe *javax.swing.AbstractButton* dans laquelle on trouve les méthodes suivantes :

void **addActionListener**(ActionListener k);

void **addChangeListener**(ChangeListener k);

void **addItemListener**(ItemListener k);

C'est l'interface ***ActionListener*** qui gére les événement du clis sur un bouton. Celle-ci ne définit qu'une méthode :

***void actionPerformed(ActionEvent e) ;***

La méthode reçoit un paramètre *ActionEvent* que nous ignorerons pour le moment. Pour gérer le clic sur le bouton *btntest* de notre programme on lui associe d'abord un gestionnaire d'événements :

btnTest.addActionListener(new ActionListener()

{

public void actionPerformed(ActionEvent evt){

btnTest\_clic(evt);

}

}//classe anonyme

);//gestionnaire d'evt

Ici, la méthode *actionPerformed* renvoie à la méthode *btnTest\_clic* que nous définissons de la façon suivante :

public void btnTest\_clic(ActionEvent evt){

// suivi console

System.out.println("clic sur bouton");

}//btnTest\_click

***Le programme complet est le suivant :***

// classes importées

import javax.swing.\*;

import java.awt.\*;

import java.io.\*;

import java.awt.event.\*;

// la classe formulaire

public class form4 extends JFrame {

// un bouton

JButton btnTest=null;

Container conteneur=null;

// le constructeur

public form4() {

// titre de la fenêtre

this.setTitle("Formulaire avec bouton");

// dimensions de la fenêtre

this.setSize(new Dimension(300,100));

// création d'un gestionnaire d'événements

WindowAdapter win=new WindowAdapter(){

public void windowClosing(WindowEvent e){

System.exit(0);

}

};//définition win

// ce gestionnaire d'événements va gérer les évts de la fenêtre courante

this.addWindowListener(win);

// on récupère le conteneur de la fenêtre

conteneur=this.getContentPane();

// on choisit un gestionnaire de mise en forme des composants dans ce //conteneur

conteneur.setLayout(null);

// on crée un bouton

btnTest=new JButton();

// on fixe son libellé

btnTest.setText("Test");

// on fixe son emplacement et ses dimensions

btnTest.setBounds(10,20,100,20);

// on lui associe un gestionnaire d'évt

btnTest.addActionListener(new ActionListener()

{

public void actionPerformed(ActionEvent evt){

btnTest\_clic(evt);

}

}//classe anonyme

);//gestinnaire d'evt

// on ajoute le bouton au conteneur

conteneur.add(btnTest);

}//constructeur

public void btnTest\_clic(ActionEvent evt){

// suivi console

System.out.println("clic sur bouton");

}//btnTest\_click

// fonction de test

public static void main(String[] args) {

// on affiche le formulaire

new form4().setVisible(true);

}//main

}//classe

### LES GESTIONNAIRES D'EVENEMENTS

Les principaux composants swing que nous allons présenter sont les fenêtres (JFrame), les boutons (*JButton*), les cases à cocher (*JCheckBox*), les boutons radio (*JButtonRadio*), les listes déroulantes (*JComboBox*), les listes (*JList*), les variateurs (*JScrollBar*), les étiquettes (*JLabel*), les boîtes de saisie monoligne (*JTextField*) ou multilignes (*JTextArea*), les menus (*JMenuBar*), les éléments de menu (*JMenuItem*).

***Les tableaux suivants donnent une liste de quelques gestionnaires d'événements et les événements auxquels ils sont liés.***

|  |  |  |  |
| --- | --- | --- | --- |
| **Gestionnaire** | **Composant(s)** | **Méthode d'enregistrement** | **Evénement** |
| *ActionListener* | *JButton ,JCheckbox, JButtonRadio, JMenuItem, JTextField* | public void addActionListener(ActionListener) | clic sur le bouton, la case à cocher,  le bouton radio, l'élément de menu  l'utilisateur a tapé [Entrée] dans la  zone de saisie |
| *ItemListener* | *JComboBox, JList* | public void addItemListener(ItemListener) | L'élément sélectionné a changé |
| *InputMethodListener* | *JTextField, JTextArea* | public void addMethodInputListener(InputMethodListener) | le texte de la zone de saisie a changé  ou le curseur de saisie a changé de  position |
| *CaretListener* | *JTextField, JTextArea* | public void addcaretListener(CaretListener) | Le curseur de saisie a changé de  position |
| *AdjustmentListener*  *MouseMotionListener* | *JScrollBar* | public void addAdjustmentListener(AdjustmentListener)  public void addMouseMotionListener(MouseMotionListener) | la valeur du variateur a changé.  la souris a bougé |
| *WindowListener ;*  *MouseListener* | *JFrame* | public void addWindowlistener(WindowListener)  public void addMouselistener(MouseListener) | événement fenêtre  événements souris (clic,  entrée/sortie du domaine d'un  composant, bouton pressé, relâche) |
| *FocusListener* | *JFrame* | public void addFocuslistener(FocusListener) | événement focus (obtenu, perdu) |
| *KeyListener* | *JFrame* | public void addKeylistener(KeyListener) | événement clavier( touche tapée,  pressée, relachée) |

|  |  |
| --- | --- |
| **Composant** | **Méthode d'enregistrement des gestionnaires d'évenements** |
| *JButton* | public void addActionListener(ActionListener) |
| *JCheckbox* | public void addItemListener(ItemListener) |
| *JCheckboxMenuItem* | public void addItemListener(ItemListener) |
| *JComboBox* | public void addItemListener(ItemListener)  public void addActionListener(ActionListener) |
| *JList* | public void addItemListener(ItemListener) |
| *Container* | public void addContainerListener(ContainerListener) |
| *JComponent* | public void addComponentListener(ComponentListener)  public void addFocusListener(FocusListener)  public void addKeyListener(KeyListener)  public void addMouseListener(MouseListener)  public void addMouseMotionListener(MouseMotionListener) |
| *JFrame* | public void addWindowlistener(WindowListener) |
| *JMenuItem* | public void addActionListener(ActionListener) |
| *JPanel* | comme Container |
| *JScrollPane* | comme Container |
| *JScrollBar* | public void addAdjustmentListener(AdjustmentListener) |
| *JTextComponent* | public void addInputMethodListener(InputMethodListener)  public void addCaretListener(CaretListener) |
| *JTextArea* | comme JTextComponent |
| *JTextField* | comme JTextComponent  public void addActionListener(ActionListener) |

Tous les composants, sauf ceux du type *TextXXX*, étant dérivés de la classe J*Component,* possédent également les méthodes associées à cette classe.

### LES METHODES DES GESTIONNAIRES D'EVENEMENTS

Le tableau suivant liste les méthodes que doivent implémenter les différents gestionnaires d'événements.

|  |  |
| --- | --- |
| **Interface** | **Méthodes** |
| *ActionListener* | public void actionPerformed(ActionEvent) |
| *AdjustmentListener* | public void adjustmentValueChanged(AdjustmentEvent) |
| *ComponentListener* | public void componentHidden(ComponentEvent)  public void componentMoved(ComponentEvent)  public void componentResized(ComponentEvent)  public void componentShown(ComponentEvent) |
| *ContainerListener* | public void componentAdded(ContainerEvent)  public void componentRemoved(ContainerEvent) |
| *FocusListener* | public void focusGained(FocusEvent)  public void focusLost(FocusEvent) |
| *ItemListener* | public void itemStateChanged(ItemEvent) |
| *KeyListener* | public void keyPressed(KeyEvent)  public void keyReleased(KeyEvent)  public void keyTyped(KeyEvent) |
| *MouseListener* | public void mouseClicked(MouseEvent)  public void mouseEntered(MouseEvent)  public void mouseExited(MouseEvent)  public void mousePressed(MouseEvent)  public void mouseReleased(MouseEvent) |
| *MouseMotionListener* | public void mouseDragged(MouseEvent)  public void mouseMoved(MouseEvent) |
| *TextListener* | public void textValueChanged(TextEvent) |
| *InputmethodListener* | public void InputMethodTextChanged(InputMethodEvent)  public void caretPositionChanged(InputMethodEvent) |
| *CaretLisetner* | public void caretUpdate(CaretEvent) |
| *WindowListener* | public void windowActivated(WindowEvent)  public void windowClosed(WindowEvent)  public void windowClosing(WindowEvent)  public void windowDeactivated(WindowEvent)  public void windowDeiconified(WindowEvent)  public void windowIconified(WindowEvent)  public void windowOpened(WindowEvent) |

### LES CLASSES ADAPTATEURS

*XXXListener* avec des méthodes vides. Un gestionnaire d'événements dérivé d'une classe *XXXAdapter* peut alors n'implémenter qu'une partie des méthodes de l'interface *XXXListener*, celles dont l’application a besoin. Supposons qu'on veuille gérer les clics de souris sur un composant *Frame f1*. On pourrait lui associer un gestionnaire d'événements par :

f1.addMouseListener(new gestionnaireSouris());

et écrire :

public class gestionnaireSouris implements MouseListener{

// on écrit les 5 méthodes de l'interface MouseListener

// mouseClicked, ..., mouseReleased)

}// fin classe

Comme on ne souhaite gérer que les clics de souris, il est cependant préférable d'écrire :

public class gestionnaireSouris extends MouseAdapter{

// on écrit une seule méthode, celle qui gère les clics de souris

public void mouseClicked(MouseEvent evt){

…

}

}// fin classe

Le tableau suivant donne les classes adapteurs des différents gestionnaires d'événements :

|  |  |
| --- | --- |
| **Gestionnaire d'événements** | **Adaptateur** |
| *ComponentListener* | ComponentAdapter |
| *ContainerListener* | ContainerAdapter |
| *FocusListener* | FocusAdapter |
| *KeyListener* | KeyAdapter |
| *MouseListener* | MouseAdapter |
| *MouseMotionListener* | MouseMotionAdapter |
| MouseMotionAdapter | WindowAdapter |

### CONCLUSION

Nous venons de présenter les concepts de base de la création d'interfaces graphiques en Java :

* création d'une fenêtre
* création de composants
* association des composants à la fenêtre avec un gestionnaire de disposition
* association de gestionnaires d'évènements aux composants

## LES GESTIONNAIRES DE PLACEMENT DES COMPOSSANTS.

### Introduction

Les gestionnaires de placement (layout manager) permettent de disposer des composants dans un panneau de contenu en fonction des caractéristiques qui leurs sont propres. L'ensemble des gestionnaires de placement implémente l'interface *java.awt.LayoutManager*.

### [La classe FlowLayout](file:///D:\personnel\Lac%20Tanganyika\Backup_Lac\Cours_Evenementielle\java_swing\coursjava\swing_flowlayout.html)

L'instance de la classe *FlowLayout* dispose les composants dans un flux directionnel déterminé. Les alignements utilisés correspondent aux directions d'écritures et aux alignements de texte.

|  |  |
| --- | --- |
| **FlowLayout.CENTER** | : centrage des composants. |
| **FlowLayout.LEFT** | : justification à gauche des composants. |
| **FlowLayout.RIGHT** | : justification à droite des composants. |
| **FlowLayout.LEADING** | : justification par rapport au bord de départ fourni par l'orientation RTL (Right To Left) ou LTR (Left To Right) du conteneur. |
| **FlowLayout.TRAILING :** | justification par rapport au bord d'arrivée fourni par l'orientation du conteneur. |

**FlowLayout flux = new FlowLayout(FlowLayout.LEFT);**

**Container panneauContenu = getContentPane();**

**panneauContenu.setLayout(flux));**

Des intervalles entre les composants peuvent être fournis par le biais d'un constructeur (*FlowLayout (int alignement, int h, int l)*) ou de méthodes spécifiques (*setHGap()* et *setVGap()*).

L'objet *FlowLayout* constitue le gestionnaire de placement par défaut du conteneur des objets du type *JPanel*.

import java.awt.FlowLayout;

import javax.swing.JButton;

import javax.swing.JFrame;

import javax.swing.JPanel;

import javax.swing.JTextField;

public class FluxPlacement extends JFrame {

public FluxPlacement() {

super("Démonstration FlowLayout");

this.creerInterface();

super.setSize(300, 300);

super.setVisible(true);

}

public void creerInterface() {

JPanel pan = (JPanel)this.getContentPane();

JTextField texte1 = new JTextField(20);

JButton btn1 = new JButton("Premier bouton");

JTextField texte2 = new JTextField(20);

JButton btn2 = new JButton("Second bouton");

**FlowLayout gestionnaire = new FlowLayout();**

**pan.setLayout(gestionnaire);**

**pan.add(texte1);**

**pan.add(btn1);**

**pan.add(texte2);**

**pan.add(btn2);**

}

public static void main(String[] args) {

new FluxPlacement();

}

}

### [Le gestionnaire BorderLayout](file:///D:\personnel\Lac%20Tanganyika\Backup_Lac\Cours_Evenementielle\java_swing\coursjava\swing_borderlayout.html)

L'instance de la classe *BorderLayout* constitue le gestionnaire de placement par défaut d'un panneau de contenu. L'objet *BorderLayout* comporte quatre zones en bordures (Nord, Est, Sud et Ouest) et une zone centrale. Chaque région ne peut contenir qu'un seul composant.

BorderLayout PlacementBordure = new BorderLayout();

Le dimensionnement des composants s'effectue selon leur positionnement. Les composants des régions Nord et Sud peuvent s'étirer horizontalement, ceux de l'Est et de l'Ouest s'étirent verticalement et celui du centre dans les deux directions.

import java.awt.BorderLayout;

import java.awt.Dimension;

import javax.swing.JButton;

import javax.swing.JFrame;

import javax.swing.JPanel;

import javax.swing.JTextField;

public class PlacementBordure extends JFrame {

public PlacementBordure() {

super("Démonstration BorderLayout");

this.creerInterface();

super.pack();

super.setVisible(true);

}

public void creerInterface() {

JPanel pan = (JPanel)this.getContentPane();

JTextField texte = new JTextField("Centre");

texte.setPreferredSize(new Dimension(120, 120));

JButton btnNord = new JButton("Nord");

JButton btnEst = new JButton("Est");

JButton btnSud = new JButton("Sud");

JButton btnOuest = new JButton("Ouest");

**BorderLayout gestionnaire = new BorderLayout();**

**pan.setLayout(gestionnaire);**

**pan.add(texte, BorderLayout.CENTER);**

**pan.add(btnNord, BorderLayout.NORTH);**

**pan.add(btnEst, BorderLayout.EAST);**

**pan.add(btnSud, BorderLayout.SOUTH);**

**pan.add(btnOuest, BorderLayout.WEST);**

}

public static void main(String[] args) {

PlacementBordure boite = new PlacementBordure();

}

}

### [La classe BoxLayout](file:///D:\personnel\Lac%20Tanganyika\Backup_Lac\Cours_Evenementielle\java_swing\coursjava\swing_boxlayout.html)

L'instance de la classe *BoxLayout* permet de disposer les composants selon un axe horizontal ou vertical. Un objet de type *Box* peut être créé verticalement ou horizontalement, par l'intermédiaire de deux méthodes statiques de la classe *Box*, respectivement *createVerticalBox()* et *createHorizontalBox()*.

Box boiteV = Box.createVerticalBox ();

Box boiteH = Box.createHorizontalBox();

Ce gestionnaire de placement n'autorise pas un saut de ligne ou de colonne en cas de dépassement de la taille maximum possible. Dans ce cas, l'objet *BoxLayout* tente de réduire la taille des composants ou les tronque si cela n'est pas possible.

BoxLayout PlacementBoite = new BoxLayout(Container cible, int axe);

**Quatre axes d'orientation**, représentés par des constantes statiques de la classe *BoxLayout*, sont possibles :

|  |  |
| --- | --- |
| ***BorderLayout.X\_AXIS*** | : Axe vertical. |
| ***BorderLayout.Y\_AXIS*** | : Axe horizontal. |
| ***BorderLayout.LINE\_AXIS*** | : Axe de la direction d'une ligne fournie par la propriété *ComponentOrientation* du panneau de contenu. |
| ***BorderLayout.PAGE\_AXIS*** | : Axe de la direction d'une page fournie par la propriété *ComponentOrientation* du panneau de contenu. |

import javax.swing.BoxLayout;

import javax.swing.JButton;

import javax.swing.JFrame;

import javax.swing.JPanel;

import javax.swing.JTextField;

public class Boite extends JFrame {

public Boite() {

super("Démonstration BoxLayout");

this.creerInterface();

super.pack();

super.setVisible(true);

}

public void creerInterface() {

JPanel pan = (JPanel)this.getContentPane();

JTextField texte1 = new JTextField(25);

JButton btn1 = new JButton("Premier bouton");

JTextField texte2 = new JTextField(25);

JButton btn2 = new JButton("Second bouton");

**BoxLayout gestionnaire = new BoxLayout(pan, BoxLayout.X\_AXIS);**

**pan.setLayout(gestionnaire);**

**pan.add(texte1);**

**pan.add(btn1);**

**pan.add(texte2);**

**pan.add(btn2);**

}

public static void main(String[] args) {

Boite boite = new Boite();

}

}

### [La classe CardLayout](file:///D:\personnel\Lac%20Tanganyika\Backup_Lac\Cours_Evenementielle\java_swing\coursjava\swing_cardlayout.html)

L'instance de la classe *CardLayout* construit une pile de composants se plaçant les uns au dessus des autres. L'objet *CardLayout* traite les composants comme des cartes dont une seule peut être visible, par défaut, la première de la pile.

CardLayout pileCarte = new CardLayout();

Le gestionnaire de placement fournit plusieurs méthodes destinées à parcourir les cartes (*first()*, *last()*, *previous()* et *next()*) et à les afficher (*show()*).

carte.**next(getContentPane())**;

Il est possible de fournir des intervalles horizontaux et verticaux entre le conteneur et le composant par l'intermédiaire d'un constructeur (*CardLayout(int h, int l)*) ou de méthodes spécifiques (*setHGap()* et *setVGap()*).

import java.awt.BorderLayout;

import java.awt.CardLayout;

import java.awt.event.ItemEvent;

import java.awt.event.ItemListener;

import javax.swing.JButton;

import javax.swing.JComboBox;

import javax.swing.JFrame;

import javax.swing.JPanel;

import javax.swing.JTextField;

public class CartePlacement extends JFrame implements ItemListener {

private JPanel cartes;

public CartePlacement() {

super("Démonstration CardLayout");

this.creerInterface();

this.pack();

this.setVisible(true);

}

public void creerInterface() {

JPanel pan = (JPanel) this.getContentPane();

JPanel panMenu = new JPanel();

String menus[] = { "Première carte", "Seconde carte" };

JComboBox jcbMenu = new JComboBox(menus);

jcbMenu.addItemListener(this);

panMenu.add(jcbMenu);

JPanel carte1 = new JPanel();

carte1.add(new JButton("Premier Bouton"));

carte1.add(new JTextField(20));

JPanel carte2 = new JPanel();

carte2.add(new JTextField(20));

carte2.add(new JButton("Second bouton"));

cartes = new JPanel(new CardLayout());

cartes.add(carte1, menus[0]);

cartes.add(carte2, menus[1]);

pan.add(panMenu, BorderLayout.NORTH);

pan.add(cartes, BorderLayout.CENTER);

}

public void itemStateChanged(ItemEvent e) {

CardLayout carte = (CardLayout) (cartes.getLayout());

carte.show(cartes, (String) e.getItem());

}

public static void main(String[] args) {

CartePlacement doite = new CartePlacement();

}

}

### [Le gestionnaire GridLayout](file:///D:\personnel\Lac%20Tanganyika\Backup_Lac\Cours_Evenementielle\java_swing\coursjava\swing_gridlayout.html)

L'instance de la classe *GridLayout* propose un gestionnaire de placement disposant les composants à l'intérieur des cellules d'une grille rectangulaire régulière.

La classe *GridLayout* dispose de trois constructeurs.

|  |
| --- |
| GridLayout GL = new GridLayout(); |
| Le constructeur par défaut permet d'instancier un objet *GridLayout* composé d'une seule ligne et d'une seule colonne. |
| GridLayout GL = new GridLayout(lignes, colonnes); |
| Le second constructeur permet de créer un objet *GridLayout* composé du nombre de lignes et de colonnes passés en argument. |
| GridLayout GL = new GridLayout(lignes, colonnes, espacesH, espacesV); |
| Le dernier constructeur crée un gestionnaire de placement composé d'un certain nombre de colonnes et de lignes, ainsi que d'espaces horizontaux et verticaux entre les composants. |

Il faut qu'au moins une des valeurs représentant le nombre de colonnes et de lignes soit différente de zéro. De cette façon, un objet *GridLayout* pourra être soit une ligne, soit une colonne, composée d'un certain nombre de cellules.

GridLayout GL = new GridLayout(0, 10);

GridLayout GL = new GridLayout(10, 0);

import java.awt.GridLayout;

import javax.swing.JButton;

import javax.swing.JFrame;

import javax.swing.JPanel;

import javax.swing.JTextField;

public class Grille extends JFrame {

public Grille() {

super("Démonstration GridLayout");

this.creerInterface();

super.pack();

super.setVisible(true);

}

public void creerInterface() {

JPanel pan = (JPanel)this.getContentPane();

JTextField texte1 = new JTextField(20);

JButton btn1 = new JButton("Premier bouton");

JTextField texte2 = new JTextField(20);

JButton btn2 = new JButton("Second bouton");

JTextField texte3 = new JTextField(20);

JButton btn3 = new JButton("Troisième bouton");

JTextField texte4 = new JTextField(20);

JButton btn4 = new JButton("Quatrième bouton");

**GridLayout gestionnaire = new GridLayout(4, 4);**

**pan.setLayout(gestionnaire);**

**pan.add(texte1);**

**pan.add(btn1);**

**pan.add(texte2);**

**pan.add(btn2);**

**pan.add(texte3);**

**pan.add(btn3);**

**pan.add(texte4);**

**pan.add(btn4);**

}

public static void main(String[] args) {

Grille boite = new Grille();

}

}

### [Le gestionnaire GridBagLayout](file:///D:\personnel\Lac%20Tanganyika\Backup_Lac\Cours_Evenementielle\java_swing\coursjava\swing_gridbaglayout.html)

L'instance de la classe *GridBagLayout* représente le gestionnaire de placement le plus complexe et le plus flexible de la plateforme Java.

L'objet *GridBagLayout* aligne les composants en les plaçant dans une grille composée de colonnes et de lignes. Chaque cellule peut contenir un à plusieurs composants. D'autre part, contrairement à l'objet *GridLayout*, il est possible que les composants soient placés sur une ou plusieurs cellules. De même, les lignes et les colonnes dans la grille ne sont pas nécessairement toutes de la même hauteur et de la même largeur.

La gestion du placement des composants nécessitent l'instanciation de la classe *GridBagConstraints* chargée d'exposer les contraintes de positionnement.

Container contentPane = getContentPane();

GridBagLayout grille = new GridBagLayout();

GridBagConstraints contraintes = new GridBagConstraints();

contentPane.setLayout(grille);

Chaque composant est placé dans une ou plusieurs cellules dont les caractéristiques doivent être fournies à un objet *GridBagConstraints*. Ce dernier contient un certain nombre de champs qui permettront au gestionnaire de placement de positionner précisément chaque composant.

*// Instanciation du composant*

ClasseGraphique obj = new ClasseGraphique();

*// Définition des contraintes*

contraintes.champs = valeur;

...

*// Application des contraintes*

grille.setConstraints(obj, contraintes);

*// Ajout de l'objet dans le panneau de contenu*

panneauContenu.add(obj);

Il est important de noter que ce processus doit être répété pour chaque composant à ajouter dans un panneau de contenu géré par le gestionnaire de placement *GridBagLayout*, et que les champs de l'objet *GridBagConstraints* doivent être réinitialisés pour chacun des objets graphiques, puisqu'en général une seule instance de la classe *GridBagConstraints* est employée pour l'ensemble de ces objets.

Le champs *fill* est utilisé pour la gestion du remplissage de la zone d'affichage du composant lorsque celui-ci possède une taille inférieure à l'espace disponible.

**contraintes.fill = GridBagConstraints.CONSTANTE;**

|  |  |
| --- | --- |
| *NONE* | : ne fournit aucune contrainte de taille (valeur par défaut). |
| *HORIZONTAL* | : redimensionne horizontalement le composant afin de totalement remplir la zone d'affichage horizontale. |
| *VERTICAL* | : redimensionne verticalement le composant afin de totalement remplir la zone d'affichage verticale. |
| *BOTH* | : redimensionne horizontalement et verticalement le composant afin de totalement remplir la zone d'affichage. |

L'ajout d'un composant dans ce gestionnaire de placement demande d'indiquer dans quelle cellule devra s'afficher le composant. L'objet *GridBagConstraints* dispose de deux champs *gridx* et *gridy* spécifiant les coordonnées de la cellule devant contenir le composant.

JButton bouton = new JButton("Un bouton");

**contraintes.gridx = 0;**

**contraintes.gridy = 0;**

grille.setConstraints(bouton, contraintes);

panneauContenu.add(bouton);

Il est également possible de spécifier une zone d'affichage comprenant plusieurs cellules. Un composant peut être placé sur des cellules alignées horizontalement ou verticalement par l'intermédiaire des champs *gridheight* et *gridwidth*.

JTextArea zoneTextuelle = new JTextArea("Une zone de texte");

*// Spécification du nombre de cellules verticales*

*// sur lesquelles s'afficheront le composant.*

**contraintes.gridwidth = 3;**

*// Spécification du nombre de cellules horizontales*

*// sur lesquelles s'afficheront le composant.*

**contraintes.gridheight = 5;**

*// Affichage du composant sur la première colonne et la seconde ligne.*

contraintes.gridx = 0;

contraintes.gridy = 1;

grille.setConstraints(zoneTextuelle, contraintes);

panneauContenu.add(zoneTextuelle);

Les espaces de remplissage au sein des composants peuvent être founis par l'intermédiaire des champs *ipadx* et *ipady* spécifiant respectivement des marges horizontales et verticales.

JButton bouton = new JButton("Soumettre");

contraintes.gridx = 1;

contraintes.gridy = 5;

*// Spécifications de marges horizontales*

*// de 10 pixels et verticales de 8 pixels.*

**contraintes.ipadx = 10;**

**contraintes.ipady = 8;**

grille.setConstraints(bouton, contraintes);

panneauContenu.add(bouton);

**Les espaces entre les composants** sont définis par le biais d'un champs ***insets*** contenant un objet de type *Insets*, auquel sont passées quatre valeurs entières représentant les espaces supérieurs, inférieurs, droites et gauches.

**constraintes.insets = new Insets(valSup, valGau, valInf, valInf);**

Les champs *weightx* et *weighty* permettent respectivement une redistribution des espaces libres horizontaux et verticaux entre les composants.

**contraintes.weightx = 10;**

**contraintes.weighty = 12;**

Le champs *anchor* fournit la position du composant au sein de sa zone d'affichage dans le panneau de contenu.

contraintes.anchor = GridBagConstraints.SOUTH;

Les valeurs possibles pour ce champs se décomposent en deux catégories.

|  |  |  |
| --- | --- | --- |
| les valeurs relatives | : | **CENTER (valeur par défaut), NORTH, NORTHEAST, EAST, SOUTHEAST, SOUTH, SOUTHWEST, WEST et NORTHWEST** |
| les valeurs absolues | : | **PAGE\_START, PAGE\_END, LINE\_START, LINE\_END, FIRST\_LINE\_START, FIRST\_LINE\_END, LAST\_LINE\_START et LAST\_LINE\_END** |

La classe *GridBagLayout* possède deux constructeurs. Le premier ne contient aucun argument et le second demande de fournir toutes les valeurs explicitées précédemment à appliquer par défaut aux composants à ajouter dans le conteneur.

GridBagConstraints GBC =

new **GridBagConstraints(**val\_gridx,

val\_gridy,

val\_gridwidth,

val\_gridheight,

val\_weightx,

val\_weighty,

val\_anchor,

val\_fill,

obj\_insets,

val\_ipadx,

val\_ipady**)**;

import java.awt.Dimension;

import java.awt.GridBagConstraints;

import java.awt.GridBagLayout;

import javax.swing.JButton;

import javax.swing.JFrame;

import javax.swing.JPanel;

import javax.swing.JTextField;

public class GrilleComplexe extends JFrame {

public GrilleComplexe() {

super("Démonstration GridBagLayout");

this.creerInterface();

super.pack();

super.setVisible(true);

}

public void creerInterface() {

JPanel pan = (JPanel)this.getContentPane();

JTextField texte = new JTextField("Centre");

texte.setPreferredSize(new Dimension(120, 120));

JButton btnNord = new JButton("Nord");

JButton btnEst = new JButton("Est");

JButton btnSud = new JButton("Sud");

JButton btnOuest = new JButton("Ouest");

**GridBagConstraints contraintes = new GridBagConstraints();**

**GridBagLayout gestionnaire = new GridBagLayout();**

**pan.setLayout(gestionnaire);**

**contraintes.gridx = 0;**

**contraintes.gridy = 0;**

**contraintes.gridheight = 1;**

**contraintes.gridwidth = 3;**

**contraintes.weightx = 0;**

**contraintes.weighty = 0;**

**contraintes.fill = GridBagConstraints.HORIZONTAL;**

**gestionnaire.setConstraints(btnNord, contraintes);**

**pan.add(btnNord);**

**contraintes.gridx = 0;**

**contraintes.gridy = 1;**

**contraintes.gridheight = 1;**

**contraintes.gridwidth = 1;**

**contraintes.weightx = 0;**

**contraintes.weighty = 0;**

**contraintes.fill = GridBagConstraints.VERTICAL;**

**gestionnaire.setConstraints(btnOuest, contraintes);**

**pan.add(btnOuest);**

**contraintes.gridx = 1;**

**contraintes.gridy = 1;**

**contraintes.gridheight = 1;**

**contraintes.gridwidth = 1;**

**contraintes.weightx = 1;**

**contraintes.weighty = 1;**

**gestionnaire.setConstraints(texte, contraintes);**

**pan.add(texte);**

**contraintes.gridx = 2;**

**contraintes.gridy = 1;**

**contraintes.gridheight = 1;**

**contraintes.gridwidth = 1;**

**contraintes.weightx = 0;**

**contraintes.weighty = 0;**

**contraintes.fill = GridBagConstraints.VERTICAL;**

**gestionnaire.setConstraints(btnEst, contraintes);**

**pan.add(btnEst);**

**contraintes.gridx = 0;**

**contraintes.gridy = 2;**

**contraintes.gridheight = 1;**

**contraintes.gridwidth = 3;**

**contraintes.weightx = 0;**

**contraintes.weighty = 0;**

**contraintes.fill = GridBagConstraints.HORIZONTAL;**

**gestionnaire.setConstraints(btnSud, contraintes);**

**pan.add(btnSud);**

}

public static void main(String[] args) {

GrilleComplexe boite = new GrilleComplexe();

}

}

### [Le gestionnaire SpringLayout](file:///D:\personnel\Lac%20Tanganyika\Backup_Lac\Cours_Evenementielle\java_swing\coursjava\swing_springlayout.html)

Le gestionnaire de placement *SpringLayout* s'appuie sur les bords des composants graphiques afin de les placer au sein d'un panneau de contenu. Il définit des relations entre les bords des composants.

SpringLayout ressort = new SpringLayout();

La classe *SpringLayout.Constraints* permet de spécifier les contraintes à appliquer à l'objet *SpringLayout* pour placer des composants graphiques. Cette classe permet de fixer ou de récupérer la valeur de quatre propriétés, les coordonnées (*x* et *y*) et la taille (*height* et *width*) à appliquer à un composant.

JtextField texte = new JTextField();

SpringLayout.Constraints contraintes = new SpringLayout.Constraints(texte);

*//ou*

SpringLayout.Constraints contraintes = SpringLayout.Constraints getConstraints(texte);

Les méthodes de la classe *SpringLayout.Constraints* utilise exclusivement des objets *Spring* pour spécifier ou obttenir les valeurs de contraintes précitées. Cette objet contient trois propriétés qui caractérise son comportement, les valeurs minimum, maximum et préférée. Chacune de ces propriétés peut être impliqué dans la définition de sa quetrième valeur, soit une propriété basée sur une série de règles.

JPanel pan = (JPanel)frame.getContentPane();

SpringLayout gestionnaire = new SpringLayout();

pan.setLayout(gestionnaire);

JButton bouton = new JButton("Bouton");

JTextField texte = new JTextField(20);

pan.add(bouton);

pan.add(texte);

SpringLayout.Constraints contraintesBouton =

gestionnaire.getConstraints(bouton);

contraintesBouton.setX(Spring.constant(8));

contraintesBouton.setY(Spring.constant(8));

SpringLayout.Constraints contraintesTexte =

gestionnaire.getConstraints(textField);

contraintesTexte.setX(

Spring.sum(

Spring.constant(5),

contraintesBouton.getConstraint(SpringLayout.EAST)));

contraintesTexte.setY(Spring.constant(5));

La classe *SpringLayout* fournit quatre constantes désignant chacun des bords du composant graphique contraint (*EAST*, *NORTH*, *SOUTH* et *WEST*). Une constante passée à la méthode *getConstraint()* indique qu'un objet *Spring* concernant le bord spécifié doit être retourné.

Un instance de la classe *Spring* peut être assimilé à un ressort qui est capable de se comprimer ou de s'étirer entre des valeurs minimum et maximum autour de la valeur préférée. A cet effet, plusieurs méthodes de la classe *Spring* sont disponibles afin de déterminer les valeurs maximum, minimum et préférée, d'effectuer des soustractions et additions entre des objets *Spring*.

JTextField texte = new JTextField();

Spring ressortTexte = Spring.width(texte);

JButton bouton = new JButton("Bouton");

Spring ressortBouton = Spring.width(bouton);

int max = ressort.getMaximumValue();

int min = ressort.getMinimumValue();

int pref = ressort.getPreferredValue();

Spring resultatAddition = Spring(

ressortTexte,

ressortBouton);

Spring resultatSoustraction = Spring(

ressortTexte,

Spring.minus(ressortBouton));

import java.awt.Component;

import javax.swing.JButton;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JPanel;

import javax.swing.JTextField;

import javax.swing.Spring;

import javax.swing.SpringLayout;

public class RessortPlacement extends JFrame {

public RessortPlacement() {

super("Démonstration SpringLayout");

this.creerInterface();

super.pack();

super.setVisible(true);

}

public void creerInterface() {

JPanel pan = (JPanel) this.getContentPane();

JLabel label1 = new JLabel("Premier label");

JTextField textField1 = new JTextField(20);

JButton bouton1 = new JButton("Premier bouton");

JLabel label2 = new JLabel("Second label");

JTextField textField2 = new JTextField(20);

JButton bouton2 = new JButton("Second bouton");

JLabel label3 = new JLabel("Troisième label");

JTextField textField3 = new JTextField(20);

JButton bouton3 = new JButton("Troisième bouton");

**SpringLayout ressort = new SpringLayout();**

**pan.setLayout(ressort);**

**pan.add(label1);**

**pan.add(textField1);**

**pan.add(bouton1);**

**pan.add(label2);**

**pan.add(textField2);**

**pan.add(bouton2);**

**pan.add(label3);**

**pan.add(textField3);**

**pan.add(bouton3);**

**this.creerGrilleCompacte(pan, 3, 3, 5, 5, 5, 5);**

}

public void creerGrilleCompacte(

JPanel panneau,

int lignes, int colonnes,

int initialX, int initialY,

int espacementX, int espacementY) {

SpringLayout gestionnaire = null;

if (panneau.getLayout() instanceof SpringLayout)

**gestionnaire = (SpringLayout) panneau.getLayout();**

else throw new ClassCastException();

Spring x = Spring.constant(initialX);

for (int col = 0; col < colonnes; col++) {

Spring largeur = Spring.constant(0);

for (int ligne = 0; ligne < lignes; ligne++) {

largeur = Spring.max(largeur, getConstraintesComposant(ligne,

col, panneau, colonnes).getWidth());

}

for (int ligne = 0; ligne < lignes; ligne++) {

SpringLayout.Constraints contraintes = getConstraintesComposant(

ligne, col, panneau, colonnes);

contraintes.setX(x);

contraintes.setWidth(largeur);

}

x = Spring.sum(x, Spring.sum(largeur, Spring.constant(espacementX)));

}

Spring y = Spring.constant(initialY);

for (int ligne = 0; ligne < lignes; ligne++) {

Spring hauteur = Spring.constant(0);

for (int colonne = 0; colonne < colonnes; colonne++) {

hauteur = Spring.max(hauteur, getConstraintesComposant(ligne,

colonne, panneau, colonnes).getHeight());

}

for (int c = 0; c < colonnes; c++) {

SpringLayout.Constraints contraintes = getConstraintesComposant(

ligne, c, panneau, colonnes);

contraintes.setY(y);

contraintes.setHeight(hauteur);

}

y = Spring

.sum(y, Spring.sum(hauteur, Spring.constant(espacementY)));

}

SpringLayout.Constraints contraintesPanneau = gestionnaire

.getConstraints(panneau);

contraintesPanneau.setConstraint(SpringLayout.SOUTH, y);

contraintesPanneau.setConstraint(SpringLayout.EAST, x);

}

private SpringLayout.Constraints getConstraintesComposant(

int ligne,

int colonne,

JPanel parent,

int colonnes) {

SpringLayout gestionnaire = (SpringLayout) parent.getLayout();

Component composant = parent.getComponent(

ligne \* colonnes + colonne);

return gestionnaire.getConstraints(composant);

}

public static void main(String[] args) {

RessortPlacement ressort = new RessortPlacement();

}

}

### Le gestionnaire GroupLayout

**Principe de fonctionnement: Dimensions**

GroupLayout travaille avec les dispositions horizontales et verticales. La mise en page est définie pour chaque dimension de façon indépendante. Vous n'avez pas besoin de s'inquiéter de la dimension verticale lors de la définition de la présentation horizontale, et vice versa, que la disposition le long de chaque axe est totalement indépendante de la mise en page sur l'autre axe.

En se concentrant sur une seule dimension, on résout la moitié du problème en une seule fois. C'est plus facile que de gérer les deux dimensions à la fois. Cela signifie, bien sûr, que chaque composant doit être défini deux fois dans la mise en page. Si vous oubliez de le faire, GroupLayout générera une exception.

## Les composants Swing

### La classe JFrame

L'utilisation d'un panneau de contenu (contentPane) pour insérer des composants (ils ne sont plus insérés directement au JFrame mais à l'objet contentPane qui lui est associé). Elle représente une fenêtre principale qui possède un titre, une taille modifiable et éventuellement un menu.

La classe possède plusieurs constructeurs :

|  |  |
| --- | --- |
| Constructeur | Rôle |
| JFrame() |  |
| JFrame(String) | Création d'une instance en précisant le titre |

#### Le comportement par défaut à la fermeture

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 |

#### La personnalisation de l'icône

La méthode setIconImage () permet de modifier l'icône de la JFrame.

import javax.swing.\*;

public class TestJFrame4 {

public static void main(String argv[]) {

JFrame f = new JFrame("ma fenetre");

f.setSize(300,100);

JButton b =new JButton ("Mon bouton");

f.getContentPane ().add (b);

f.setDefaultCloseOperation(WindowConstants.DISPOSE\_ON\_CLOSE);

ImageIcon image = new ImageIcon ("book.gif");

f.setIconImage (image.getImage ());

f.setVisible (true);

}

}

Si l'image n'est pas trouvée, alors l'icône est vide. Si l'image est trop grande, elle est redimensionnée.

#### Centrer une JFrame à l'écran

Par défaut, une JFrame est affichée dans le coin supérieur gauche de l'écran. Pour la centrer dans l'écran, il faut procéder comme pour une Frame : déterminer la position de la Frame en fonction de sa dimension et de celle de l'écran et utiliser la méthode setLocation () pour affecter cette position.

import javax.swing.\* ;

import java.awt.\* ;

public class TestJFrame5 {

public static void main(String argv[]) {

JFrame f = new JFrame("ma fenetre");

f.setSize(300,100);

JButton b =new JButton("Mon bouton");

f.getContentPane().add(b);

f.setDefaultCloseOperation(WindowConstants.DISPOSE\_ON\_CLOSE);

Dimension dim = Toolkit.getDefaultToolkit ().getScreenSize ();

f.setLocation(dim.width/2 - f.getWidth()/2, dim.height/2 - f.getHeight ()/2);

f.setVisible (true);

}

}

### Les étiquettes : la classe JLabel

Le composant JLabel propose les mêmes fonctionnalités que les intitulés AWT mais ils peuvent en plus contenir des icônes .

Cette classe possède plusieurs constructeurs :

|  |  |
| --- | --- |
| Constructeurs | rôles |
| JLabel() | Création d'une instance sans texte ni image |
| JLabel(Icon) | Création d'une instance en précisant l'image |
| JLabel(Icon, int) | Création d'une instance en précisant l'image et l'alignement horizontal |
| JLabel(String) | Création d'une instance en précisant le texte |
| JLabel(String, Icon, int) | Création d'une instance en précisant le texte, l'image et l'alignement horizontal |
| JLabel(String, int) | Création d'une instance en précisant le texte et l'alignement horizontal |

Le composant JLabel permet d'afficher un texte et/ou une icône en précisant leur alignement. L'icône doit être au format GIF et peut être une animation dans ce format.

import javax.swing.\*;

import java.awt.\*;

public class TestJLabel1 {

public static void main(String argv[]) {

JFrame f = new JFrame("ma fenetre");

f.setSize(100,200);

JPanel pannel = new JPanel();

JLabel jLabel1 =new JLabel("Mon texte dans JLabel");

pannel.add(jLabel1);

ImageIcon icone = new ImageIcon("book.gif");

JLabel jLabel2 =new JLabel(icone);

pannel.add(jLabel2);

JLabel jLabel3 =new JLabel("Mon texte",icone,SwingConstants.LEFT);

pannel.add(jLabel3);

f.getContentPane().add(pannel);

f.setVisible(true);

}

}

La classe JLabel définit plusieurs méthodes pour modifier l'apparence du composant :

|  |  |
| --- | --- |
| **Méthodes** | **Rôle** |
| setText () | Permet d'initialiser ou de modifier le texte affiché |
| setOpaque() | Indique si le composant est transparent (paramètre false) ou opaque (true) |
| setBackground() | Indique la couleur de fond du composant (setOpaque doit être à true) |
| setVerticalAlignment() | Permet de modifier l'alignement vertical du texte et de l'icône |
| setHorizontalTextAlignment() | Permet de modifier l'alignement horizontal du texte uniquement |
| setVerticalTextAlignment() | Permet de modifier l'alignement vertical du texte uniquement  Exemple :  jLabel.setVerticalTextPosition(SwingConstants.TOP); |
| setDisabledIcon() | Permet d'assigner une icône dans un état désactivée |
| setIcon() | Permet d'assigner une icône |
| setFont() | Permet de préciser la police du texte |
| setForeGround() | Permet de préciser la couleur du texte |
| setHorizontalAlignment() | Permet de modifier l'alignement horizontal du texte et de l'icône |

L'alignement vertical par défaut d'un JLabel est centré. L'alignement horizontal par défaut est soit à droite si il ne contient que du texte, soit centré si il contient un image avec ou sans texte. Pour modifier cet alignement, il suffit d'utiliser les méthodes ci dessus en utilisant des constantes en paramètres :

SwingConstants.LEFT, SwingConstants.CENTER, SwingConstants.RIGHT, SwingConstants.TOP, SwingConstants.BOTTOM

Par défaut, un JLabel est transparent : son fond n'est pas dessiné. Pour le dessiner, il faut utiliser la méthode setOpaque() :

import javax.swing.\*;

import java.awt.\*;

public class TestJLabel2 {

public static void main(String argv[]) {

JFrame f = new JFrame("ma fenetre");

f.setSize(100,200);

JPanel pannel = new JPanel();

JLabel jLabel1 =new JLabel("Mon texte dans JLabel 1");

jLabel1.setBackground(Color.red);

pannel.add(jLabel1);

JLabel jLabel2 =new JLabel("Mon texte dans JLabel 2");

jLabel2.setBackground(Color.red);

jLabel2.setOpaque(true);

pannel.add(jLabel2);

f.getContentPane().add(pannel);

f.setVisible(true);

}

}

### Les panneaux : la classe Jpanel

La classe JPanel est un conteneur utilisé pour regrouper et organiser des composants grâce à un gestionnaire de présentation (layout manager). Le gestionnaire par défaut d'un JPanel est un objet de la classe FlowLayout.

### Les boutons

Il existe plusieurs boutons définis par Swing.

#### La classe AbstractButton

C'est une classe abstraite dont héritent les boutons Swing JButton, JMenuItem et JToggleButton. Cette classe définit de nombreuses méthodes dont les principales sont :

|  |  |
| --- | --- |
| **Méthode** | **Rôle** |
| AddActionListener | Associer un écouteur sur un évenement de type ActionEvent |
| AddChangeListener | Associer un écouteur sur un évenement de type ChangeEvent |
| AddItemListener | Associer un écouteur sur un évenement de type ItemEvent |
| doClick() | Déclencher un clic par programmation |
| getText() | Obtenir le texte affiché par le composant |
| setDisabledIcon() | Associer une icône affichée lorsque le composant à l'état désélectionné |
| setDisabledSelectedIcon() | Associer une icône affichée lors du passage de la souris sur le composant à l'état  désélectionné |
| setEnabled () | Activer/désactiver le composant |

Tous les boutons peuvent afficher du texte et/ou une image. Il est possible de préciser une image différente lors du passage de la souris sur le composant et lors de l'enfoncement du bouton : dans ce cas, il faut créer trois images pour chacun des états (normal, enfoncé et survolé). L'image normale est associée au bouton grâce au constructeur, l'image enfoncée grâce à la méthode setPressedIcon () et l'image lors d'un survole grâce à la méthode setRolloverIcon (). Il suffit enfin d'appeler la méthode setRolloverEnable () avec en paramètre la valeur true.

import javax.swing.\*;

import java.awt.event.\*;

public class swing4 extends JFrame {

public swing4() {

super("titre de l'application");

WindowListener l = new WindowAdapter() {

public void windowClosing(WindowEvent e){

System.exit(0);

}

};

addWindowListener(l);

ImageIcon imageNormale = new ImageIcon("arrow.gif");

ImageIcon imagePassage = new ImageIcon("arrowr.gif");

ImageIcon imageEnfoncee = new ImageIcon("arrowy.gif");

JButton bouton = new JButton("Mon bouton",imageNormale);

bouton.setPressedIcon(imageEnfoncee);

bouton.setRolloverIcon(imagePassage);

bouton.setRolloverEnabled(true);

getContentPane().add(bouton, "Center");

JPanel panneau = new JPanel();

panneau.add(bouton);

setContentPane(panneau);

setSize(200,100);

setVisible(true);

}

public static void main(String [] args){

JFrame frame = new swing4();

}

}

Un bouton peut recevoir des évenements de type ActionEvents (le bouton a été activé), ChangeEvents, et ItemEvents.

import javax.swing.\*;

import java.awt.event.\*;

public class TestJButton3 {

public static void main(String argv[]) {

JFrame f = new JFrame("ma fenetre");

f.setSize(300,100);

JPanel pannel = new JPanel();

JButton bouton1 = new JButton("Bouton1");

bouton1.addActionListener( new ActionListener() {

public void actionPerformed(ActionEvent e) {

**System.exit(0);**

**}**

}

);

pannel.add(bouton1);

f.getContentPane().add(pannel);

f.setVisible(true);

}

}

#### La classe JButton

JButton est un composant qui représente un bouton : il peut contenir un texte et/ou une icône. Les constructeurs sont :

|  |  |
| --- | --- |
| **Constructeur** | **Rôle** |
| JButton() |  |
| JButton(String) | préciser le texte du bouton |
| JButton(Icon) | préciser une icône |
| JButton(String, Icon) | préciser un texte et une icone |

Il ne gère pas d'état. Toutes les indications concernant le contenu du composant JLabel sont valables pour le composant JButton.

#### La classe JToggleButton

Cette classe définit un bouton à deux états : c'est la classe mère des composants JCheckBox et JRadioButton. La méthode setSelected() héritée de AbstractButton permet de mettre à jour l'état du bouton. La méthode isSelected() permet de connaître cet état.

#### La classe ButtonGroup

La classe ButtonGroup permet de gérer un ensemble de boutons en garantissant qu'un seul bouton du groupe sera sélectionné. Pour utiliser la classe ButtonGroup, il suffit d'instancier un objet et d'ajouter des boutons (objets héritant de la classe AbstractButton) grâce à la méthode add(). Il est préférable d'utiliser des objets de la classe JToggleButton ou d'une de ces classes filles car elles sont capables de gérer leurs états.

import javax.swing.\*;

public class TestGroupButton1 {

public static void main(String argv[]) {

JFrame f = new JFrame("ma fenetre");

f.setSize(300,100);

JPanel pannel = new JPanel();

ButtonGroup groupe = new ButtonGroup();

JRadioButton bouton1 = new JRadioButton("Bouton 1");

groupe.add(bouton1);

pannel.add(bouton1);

JRadioButton bouton2 = new JRadioButton("Bouton 2");

groupe.add(bouton2);

pannel.add(bouton2);

JRadioButton bouton3 = new JRadioButton("Bouton 3");

groupe.add(bouton3);

pannel.add(bouton3);

f.getContentPane().add(pannel);

f.setVisible(true);

}

}

#### Les cases à cocher : la classe JCheckBox

Les constructeurs sont les suivants :

|  |  |
| --- | --- |
| **Constructeur** | **Rôle** |
| JCheckBox(String) | précise l'intitulé |
| JCheckBox(String, boolean) | précise l'intitulé et l'état |
| JCheckBox(Icon) | précise une icône comme intitulé |
| JCheckBox(Icon, boolean) | précise une icône comme intitulé et l'état |
| JCheckBox(String, Icon) | précise un texte et une icône comme intitulé |
| JCheckBox(String, Icon, boolean) | précise un texte et une icône comme intitulé et l'état |

Un groupe de cases à cocher peut être défini avec la classe ButtonGroup. Dans ce cas, un seul composant du groupe peut être sélectionné. Pour l'utiliser, il faut créer un objet de la classe ButtonGroup et utiliser la méthode add() pour ajouter un composant au groupe.

#### Les boutons radio : la classe JRadioButton

Les constructeurs sont les mêmes que ceux de la classe JCheckBox.

import javax.swing.\*;

public class TestJRadioButton1 {

public static void main(String argv[]) {

JFrame f = new JFrame("ma fenetre");

f.setSize(300,100);

JPanel pannel = new JPanel();

JRadioButton bouton1 = new JRadioButton("Bouton 1");

pannel.add(bouton1);

JRadioButton bouton2 = new JRadioButton("Bouton 2");

pannel.add(bouton2);

JRadioButton bouton3 = new JRadioButton("Bouton 3");

pannel.add(bouton3);

f.getContentPane().add(pannel);

f.setVisible(true);

}

}

Pour regrouper plusieurs boutons radio, il faut utiliser la classe CheckboxGroup

### Les composants de saisie de texte

Swing possède plusieurs composants pour permettre la saisie de texte.

#### La classe JTextComponent

La classe abstraite JTextComponent est la classe mère de tout les composants permettant la saisie de texte. Les données du composant (le modèle dans le motif de conception MVC) sont encapsulées dans un objet qui implémente l'interface Document. Deux classes implémentant cette interface sont fournies en standard : PlainDocument pour du texte simple et StyledDocument pour du texte riche pouvant contenir entre autre plusieurs polices de caractères, des couleurs, des images, ...

La classe JTextComponent possède de nombreuses méthodes dont les principales sont :

|  |  |
| --- | --- |
| Méthode | Rôle |
| void cut() | Couper le contenu du texte et le mettre dans le presse papier système  Document |
| getDocument() | Renvoyer l'objet de type Document qui encapsule le texte saisi String |
| getSelectectedText() | Renvoyer le texte sélectionné dans le composant. |
| int getSelectionEnd () | Renvoyer la position de la fin de la sélection |
| int getSelectionStart () | Renvoyer la position du début de la sélection |
| String getText () | Renvoyer le texte saisi |
| String getText (int, int) | Renvoyer une portion du texte incluse à partir de la position donnée par le premier  paramètre et la longueur donnée dans le second paramètre |
| bool isEditable () | Renvoyer un booléen qui précise si le texte est éditable ou non |
| void paste () | Coller le contenu du presse papier système dans le composant |
| void select (int, int) | Sélectionner une portion du texte dont les positions de début et de fin sont fournies en paramètres |
| Void setCaretPosition (int) | Déplacer la curseur à la position dans le texte précisé en paramètre |
| Void setEditable(boolean) | Permet de préciser si les données du composant sont éditables ou non |
| void setSelectionEnd(int) | Modifier la position de la fin de la sélection |
| Void setSelectionStart(int) | Modifier la position du début de la sélection |
| void setText(String) | Modifier le contenu du texte |

Toutes ces méthodes sont donc accessibles grâce à l'héritage pour tous les composants de saisie de texte proposés par Swing.

#### La classe JTextField

La classe javax.Swing.JTextField est un composant qui permet la saisie d'une seule ligne de texte simple. Son modèle utilise un objet de type PlainDocument.

import javax.swing.\*;

public class JTextField1 {

public static void main(String argv[]) {

JFrame f = new JFrame("ma fenetre");

f.setSize(300, 100);

JPanel pannel = new JPanel();

JTextField testField1 = new JTextField ("mon texte");

pannel.add(testField1);

f.getContentPane().add(pannel);

f.setVisible(true);

}

}

La propriété horizontalAligment permet de préciser l'alignement du texte dans le composant en utilisant les valeurs JTextField.LEFT , JTextField.CENTER ou JTextField.RIGHT.

#### La classe JPasswordField

La classe JPasswordField permet la saisie d'un texte dont tous les caractères saisis seront affichés sous la forme d'un caractère particulier ('\*' par défaut). Cette classe hérite de la classe JTextField.

import java.awt.Dimension;

import javax.swing.\*;

public class JPasswordField1 {

public static void main(String argv[]) {

JFrame f = new JFrame("ma fenetre");

f.setSize(300, 100);

JPanel pannel = new JPanel();

JPasswordField passwordField1 = new JPasswordField ("");

passwordField1.setPreferredSize(new Dimension(100,20 ));

pannel.add(passwordField1);

f.getContentPane().add(pannel);

f.setVisible(true);

}

}

La méthode setEchoChar(char) permet de préciser le caractère qui sera utilisé pour afficher la saisie d'un caractère. Il ne faut pas utiliser la méthode getText() qui est déclarée deprecated mais la méthode getPassword() pour obtenir la valeur du texte saisi.

Les méthodes copy() et cut() sont redéfinies pour empêcher l'envoi du contenu dans le composant et émettre simplement un beep.

#### La classe JTextArea

La classe JTextArea est un composant qui permet la saisie de texte simple en mode multi-lignes. Le modèle utilisé par ce composant est le PlainDocument : il ne peut donc contenir que du texte brut sans éléments multiples de formatage. JTexteArea propose plusieurs méthodes pour ajouter du texte dans son modèle :

* soit fournir le texte en paramètre du constructeur utilisé
* soit utiliser la méthode setText() qui permet d'initialiser le texte du composant

Soit utiliser la méthode append() qui permet d'ajouter du texte à la fin de celui contenu dans le texte du composant

* soit utiliser la méthode **insert()** permet d'insérer un texte à une position données en caractères dans le texte du composant
* La méthode replaceRange () permet de remplacer une partie du texte désignée par la position du caractère de début et la position de son caractère de fin par le texte fourni en paramètre.

La propriété rows permet de définir le nombre de ligne affichée par le composant : cette propriété peut donc être modifié lors d'un redimensionnement du composant. La propriété lineCount en lecture seule permet de savoir le nombre de lignes dont le texte est composé. Il ne faut pas confondre ces deux propriétés.

Par défaut, la taille du composant augmente au fur et à mesure de l'augmentation de la taille du texte qu'il contient. Pour éviter cet effet, il faut encapsuler le JTexteArea dans un JScrollPane.

#### Les onglets

La classe javax.swing.JTabbedPane encapsule un ensemble d'onglets. Chaque onglet est constitué d'un titre, d'un

composant et éventuellement d'une image.

Pour utiliser ce composant, il faut :

* instancier un objet de type JTabbedPane
* créer le composant de chaque onglet
* ajouter chaque onglet à l'objet JTabbedPane en utilisant la méthode addTab ()

import java.awt.Dimension;

import java.awt.event.KeyEvent;

import javax.swing.\*;

public class TestJTabbedPane1 {

public static void main(String[] args) {

JFrame f = new JFrame("Test JTabbedPane");

f.setSize(320, 150);

JPanel pannel = new JPanel();

JTabbedPane onglets = new JTabbedPane(SwingConstants.TOP);

JPanel onglet1 = new JPanel();

JLabel titreOnglet1 = new JLabel("Onglet 1");

onglet1.add(titreOnglet1);

onglet1.setPreferredSize(new Dimension(300, 80));

onglets.addTab("onglet1", onglet1);

JPanel onglet2 = new JPanel();

JLabel titreOnglet2 = new JLabel("Onglet 2");

onglet2.add(titreOnglet2);

onglets.addTab("onglet2", onglet2);

onglets.setOpaque(true);

pannel.add(onglets);

f.getContentPane().add(pannel);

f.setVisible(true);

}

}

A partir du JDK 1.4, il est possible d'ajouter un raccourci clavier sur chacun des onglets en utilisant la méthode setMnemonicAt (). Cette méthode attend deux paramètres : l'index de l'onglet concerné (le premier commence à 0) et la touche du clavier associée sous la forme d'une constance KeyEvent.VK\_xxx. Pour utiliser ce raccourci, il suffit d'utiliser la touche désignée en paramètre de la méthode avec la touche Alt. La classe JTabbedPane possède plusieurs méthodes qui permettent de définir le contenu de l'onglets :

|  |  |
| --- | --- |
| **Méthodes** | **Rôles** |
| addTab(String, Component) | Permet d'ajouter un nouvel onglet dont le titre et le composant sont fournis en  paramètres. Cette méthode possède plusieurs surcharges qui permettent de préciser une icône et une bulle d'aide |
| insertTab(String, Icon, Component, String, index) | Permet d'insérer un onglet dont la position est précisée dans le dernier paramètre |
| remove (int) | Permet de supprimer l'onglet dont l'index est fourni en paramètre |
| setTabPlacement | Permet de préciser le positionnement des onglets dans le composant JTabbedPane. Les valeurs possibles sont les constantes TOP, BOTTOM, LEFT et RIGHT définies dans la  classe JTabbedPane. |

### Les menus

Les menus de Swing proposent certaines caractéristiques intéressantes en plus de celle proposées par un menu standard :

* les éléments de menu peuvent contenir une icône
* les éléments de menu peuvent être de type bouton radio ou case à cocher
* les éléments de menu peuvent avoir des raccourcis clavier (accelerators)

Les menus sont mis en oeuvre dans Swing avec un ensemble de classe :

* JMenuBar : encapsule une barre de menu
* JMenu : encapsule un menu
* JMenuItem : encapsule un élément d'un menu
* JCheckBoxMenuItem : encapsule un élément d'un menu sous la forme d'une case à cocher
* JRadioButtonMenuItem : encapsule un élément d'un menu sous la forme d'un bouton radio
* JSeparator : encapsule un élément d'un menu sous la forme d'un séparateur
* JPopupMenu : encapsule un menu contextuel

Les éléments de menus cliquables héritent de la classe JAbstractButton.

JMenu hérite de la classe JMenuItem et non pas l'inverse car chaque JMenu contient un JMenuItem implicite qui encapsule le titre du menu. La plupart des classes utilisées pour les menus implémentent l'interface MenuElement. Cette interface définit des méthodes pour la gestion des actions standards de l'utilisateur. Ces actions sont gérées par la classe MenuSelectionManager.

import java.awt.\*;

import java.awt.event.\*;

import javax.swing.\*;

public class TestMenuSwing1 extends JMenuBar {

public TestMenuSwing1() {

// Listener géeacut;nérique qui affiche l'action du menu utilisé

ActionListener afficherMenuListener = new ActionListener() {

public void actionPerformed(ActionEvent event) {

System.out.println("Elément de menu [" + event.getActionCommand()

+ "] utilisé.");

}

};

// Création du menu Fichier

JMenu fichierMenu = new JMenu("Fichier");

JMenuItem item = new JMenuItem("Nouveau", 'N');

item.addActionListener(afficherMenuListener);

fichierMenu.add(item);

item = new JMenuItem("Ouvrir", 'O');

item.addActionListener(afficherMenuListener);

fichierMenu.add(item);

item = new JMenuItem("Sauver", 'S');

item.addActionListener(afficherMenuListener);

fichierMenu.insertSeparator(1);

fichierMenu.add(item);

item = new JMenuItem("Quitter");

item.addActionListener(afficherMenuListener);

fichierMenu.add(item);

// Création du menu Editer

JMenu editerMenu = new JMenu("Editer");

item = new JMenuItem("Copier");

item.addActionListener(afficherMenuListener);

item.setAccelerator(KeyStroke.getKeyStroke('C', Toolkit.getDefaultToolkit()

.getMenuShortcutKeyMask(), false));

editerMenu.add(item);

item = new JMenuItem("Couper");

item.addActionListener(afficherMenuListener);

item.setAccelerator(KeyStroke.getKeyStroke('X', Toolkit.getDefaultToolkit()

.getMenuShortcutKeyMask(), false));

editerMenu.add(item);

item = new JMenuItem("Coller");

item.addActionListener(afficherMenuListener);

item.setAccelerator(KeyStroke.getKeyStroke('V', Toolkit.getDefaultToolkit()

.getMenuShortcutKeyMask(), false));

editerMenu.add(item);

// Création du menu Divers

JMenu diversMenu = new JMenu("Divers");

JMenu sousMenuDiver1 = new JMenu("Sous menu 1");

item.addActionListener(afficherMenuListener);

item = new JMenuItem("Sous menu 1 1");

sousMenuDiver1.add(item);

item.addActionListener(afficherMenuListener);

JMenu sousMenuDivers2 = new JMenu("Sous menu 1 2");

item = new JMenuItem("Sous menu 1 2 1");

sousMenuDivers2.add(item);

sousMenuDiver1.add(sousMenuDivers2);

diversMenu.add(sousMenuDiver1);

item = new JCheckBoxMenuItem("Validé");

diversMenu.add(item);

item.addActionListener(afficherMenuListener);

diversMenu.addSeparator();

ButtonGroup buttonGroup = new ButtonGroup();

item = new JRadioButtonMenuItem("Cas 1");

diversMenu.add(item);

item.addActionListener(afficherMenuListener);

buttonGroup.add(item);

item = new JRadioButtonMenuItem("Cas 2");

diversMenu.add(item);

item.addActionListener(afficherMenuListener);

buttonGroup.add(item);

diversMenu.addSeparator();

diversMenu.add(item = new JMenuItem("Autre",

new ImageIcon("about\_32.png")));

item.addActionListener(afficherMenuListener);

// ajout des menus à la barre de menu

add(fichierMenu);

add(editerMenu);

add(diversMenu);

}

public static void main(String s[]) {

JFrame frame = new JFrame("Test de menu");

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.setJMenuBar(new TestMenuSwing1());

frame.setMinimumSize(new Dimension(250, 200));

frame.pack();

frame.setVisible(true);

}

}

#### La classe JMenuBar

La classe JMenuBar encapsule une barre de menu qui contient zéro ou plusieurs menus. La classe JMenuBar utilise la classe DefaultSingleSelectionModel comme modèle de données : un seul de ces menus peut être activé à un instant T. Pour ajouter des menus à la barre de menu, il faut utiliser la méthode add() de la classe JMenuBar qui attend en paramètre l'instance du menu.

Pour ajouter la barre de menu à une fenêtre, il faut utiliser la méthode setJMenuBar() d'une instance des classes JFrame, JInternalFrame, JDialog ou JApplet. Comme la classe JMenuBar hérite de la classe JComponent, il est aussi possible d'instancier plusieurs JMenuBar et de les insérer dans un gestionnaire de positionnement comme n'importe quel composant. Ceci permet aussi de placer le menu à sa guise.

...

public static void main(String s[]) {

JFrame frame = new JFrame("Test de menu");

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

TestMenuSwing1 menu = new TestMenuSwing1();

frame.getContentPane().add(menu, BorderLayout.SOUTH);

frame.setMinimumSize(new Dimension(250, 200));

frame.pack();

frame.setVisible(true);

}

...

La classe JMenuBar ne possède qu'un seul constructeur sans paramètre.

Les principales méthodes de la classe JMenuBar sont :

|  |  |
| --- | --- |
| **Méthodes** | **Rôle** |
| JMenu add(JMenu) | Ajouter un menu à la barre de menu |
| JMenu getMenu(int) | Obtenir le menu dont l'index est fourni en paramètre |
| int getMenuCount() | Obtenir le nombre de menu de la barre de menu |
| MenuElement[] getSubElements() | Obtenir un tableau de tous les menus |
| boolean isSelected() | précise si un menu est affiché |
| void setMenuHelp (JMenu) | Cette méthode n'est pas implémentée et lève systématiquement une exception |

#### La classe JMenuItem

La classe JMenuItem encapsule les données d'un élément de menu (libellé et/ou image). Elle hérite de la classe AbstractButton. Le comportement est similaire mais différent de celui d'un bouton : avec la classe JMenuItem, le composant est considéré comme sélectionné dès que le curseur de la souris passe dessus.

Les éléments de menu peuvent être associés à deux types de raccourcis clavier :

les accelerators : ils sont hérités de JComponent : ce sont des touches (par exemple les touches de fonctions) ou des combinaisons de touches avec les touches shift, Ctrl ou Alt qui sont affichées à la droite du libellé de l'élément du menu

* les mnemonics : ils apparaissent sous la forme d'une lettre soulignée. Ils sont utilisables seulement sur certaines plate-formes (par exemple en combinaison avec la touche Alt sous Windows).
* La méthode setAccelerator() permet d'associer un accelerator à un élément de type JMenuItem. Un mnemonic peut être associé à JMenuItm de deux façons :
* soit dans la surcharge du constructeur prévu à cet effet
* soit en utilisant la méthode setMnemonic()

Le mnemonic correspond à un caractère qui doit obligatoirement être contenu dans le libellé. Un élément de menu peut contenir uniquement une image ou être composé d'un libellé et d'une image. Une image peut être associée à un JMenuItem de deux façons :

soit dans une des surcharges du constructeur prévu à cet effet

item = new JMenuItem("Autre", new ImageIcon("about\_32.png"));

item = new JMenuItem(new ImageIcon("about\_32.png"));

soit en utilisant la méthode setIcon item.setIcon(new ImageIcon("about\_32.png"));

#### La classe JPopupMenu

La classe JPopupMenu encapsule un menu flottant qui n'est pas rattaché à une barre de menu mais à un composant. La création d'un JPopMenu est similaire à la création d'un JMenu. Il est préférable d'ajouter un élément de type JMenuItem grâce à la méthode add() de la classe JPopupMenu mais il est aussi d'ajouter n'importe quel élément qui hérite de la classe Component en utilisant une surcharge de la méthode add(). Il est possible d'ajouter un élément à un index précis en utilisant la méthode insert(). La méthode addSeparator() permet d'ajouter un élément séparateur.

Pour afficher un menu flottant, il faut ajouter un listener sur l'événement déclanchant et utiliser la méthode show() de la classe JPopupMenu.

import java.awt.Dimension;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.event.MouseAdapter;

import java.awt.event.MouseEvent;

import javax.swing.JFrame;

import javax.swing.JMenuBar;

import javax.swing.JMenuItem;

import javax.swing.JPopupMenu;

import javax.swing.JTextField;

public class TestMenuSwing2 extends JMenuBar {

public JPopupMenu popup;

public TestMenuSwing2() {

JMenuItem item = null;

// Listener générique qui affiche l'action du menu utilisé

ActionListener afficherMenuListener = new ActionListener() {

public void actionPerformed(ActionEvent event) {

System.out.println("Elément de menu [" + event.getActionCommand()

+ "] utilisé.");

}

};

popup = new JPopupMenu();

item = new JMenuItem("Copier");

item.addActionListener(afficherMenuListener);

popup.add(item);

item = new JMenuItem("Couper");

item.addActionListener(afficherMenuListener);

popup.add(item);

}

public void processMouseEvent(MouseEvent e) {

}

public static void main(String s[]) {

final JFrame frame = new JFrame("Test de menu divers");

final JTextField texte = new JTextField();

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

final TestMenuSwing2 tms = new TestMenuSwing2();

frame.add(texte);

texte.addMouseListener(new MouseAdapter() {

public void mouseClicked(MouseEvent e) {

System.out.println("mouse clicked");

afficherPopup(e);

}

public void mousePressed(MouseEvent e) {

System.out.println("mouse pressed");

afficherPopup(e);

}

public void mouseReleased(MouseEvent e) {

System.out.println("mouse released");

afficherPopup(e);

}

private void afficherPopup(MouseEvent e) {

if (e.isPopupTrigger()) {

tms.popup.show(texte, e.getX(), e.getY());

}

}

});

frame.setMinimumSize(new Dimension(250, 200));

frame.pack();

frame.setVisible(true);

}

}

### La classe Jtable

Les tableaux sont des composants qui permettent d'afficher des données de façon structurée :

import javax.swing.JFrame;

import javax.swing.JPanel;

import javax.swing.JScrollPane;

import javax.swing.JTable;

public class Fenetre extends JFrame {

public Fenetre(){

this.setLocationRelativeTo(null);

this.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

this.setTitle("JTable");

this.setSize(300, 120);

//Les données du tableau

Object[][] data = { {"Cysboy", "28 ans", "1.80 cm"},

{"BZHHydde", "28 ans", "1.80 cm"},

{"IamBow", "24 ans", "1.90 cm"},

{"FunMan", "32 ans", "1.85 cm"}

};

//Les titres des colonnes

String title[] = {"Pseudo", "Age", "Taille"};

JTable tableau = new JTable(data, title);

//On ajoute notre tableau à notre contentPane dans un scroll

//Sinon les titres des colonnes ne s'afficheront pas ! !

this.getContentPane().add(new JScrollPane(tableau));

}

public static void main(String[] args){

Fenetre fen = new Fenetre();

fen.setVisible(true);

}

}

On instancie un objet **JTable** en lui passant en paramètres les données qu'il doit utiliser. Le tableau correspondant aux titres des colonnes de votre tableau peuvent être des **String** ou de type **Object** Tandis que les données, elles, sont des **Object**.

Les données doivent être de type **Object car on peux** mettre plusieurs types d'éléments dans un tableau. Le tableau se met dans le scroll pour que les titres soient visibles.

Le scroll indique automatiquement à notre tableau où il doit afficher ses titres !  
Si on ne fait pas ceci, **on est obligé de spécifier où mettre l'entête du tableau**, comme ceci :

|  |  |
| --- | --- |
|  | //On indique que l'entête doit être au nord, donc au-dessus  this.getContentPane().add(tableau.getTableHeader(), BorderLayout.NORTH);  //Et le corps au centre !  this.getContentPane().add(tableau, BorderLayout.CENTER);  Maintenant, on va créer notre propre modèle du tableau !  Pour cela, il vous suffit de créer une classe héritant de **AbstractTableModel** qui est  une classe abstraite...    //CLASSE MODÈLE PERSONNALISÉE  class ZModel extends AbstractTableModel{  private Object[][] data;  private String[] title;  /\*\*  \* Constructeur  \* @param data  \* @param title  \*/  public ZModel(Object[][] data, String[] title){  this.data = data;  this.title = title;  }    /\*\*  \* Retourne le nombre de colonnes  \*/  public int getColumnCount() {  return this.title.length;  }    /\*\*  \* Retourne le nombre de lignes  \*/  public int getRowCount() {  return this.data.length;  }    /\*\*  \* Retourne la valeur à l'emplacement spécifié  \*/  public Object getValueAt(int row, int col) {  return this.data[row][col];  }  /\*\*  \* Retourne le titre de la colonne à l'indice spécifé  \*/  public String getColumnName(int col) {  return this.title[col];  }  /\*\*  \* Retourne vrai si la cellule est éditable : celle-ci sera donc éditable  \* @return boolean  \*/  public boolean isCellEditable(int row, int col){  //On appelle la méthode getValueAt qui retourne la valeur d'une cellule  //Et on fait un traitement spécifique si c'est un JButton  if(getValueAt(0, col) instanceof JButton)  return false;  return true;  }    }  Son application est la suivante :  ZModel model = new ZModel(data, title);  this.tableau = new JTable(model);  Les modèles servent à faire un pont entre ce qu'affiche **JTable** et ce que fait l'utilisateur.  Pour changer la façon dont sont affichées les cellules, nous devrons utiliser  **DefaultCellRenderer**. Le but du jeu est de définir une nouvelle façon  de dessiner les composants dans les cellules. On va dire à notre tableau que la valeur  qu'il a dans telle ou telle cellule est un composant (bouton ou autre).  Il suffit de créer une classe héritant de **DefaultTableCellRenderer** et de redéfinir  la méthode  public Component getTableCellRendererComponent  (JTable table, Object value, boolean isSelected, boolean hasFocus, int row, int column) ;  Une fois cette classe créée, il suffit juste de dire à notre tableau d'utiliser ce rendu de cellules, avec cette instruction : this.tableau.setDefaultRenderer(JButton.class, new TableComponent());  Le premier paramètre permet de dire à notre tableau de faire attention à ce type d'objet et enfin,  le second lui dit d'utiliser ce modèle de cellules.  class TableComponent extends DefaultTableCellRenderer {  public Component getTableCellRendererComponent(JTable table,  Object value, boolean isSelected, boolean hasFocus, int row,  int column) {    if (value instanceof JButton){  return (JButton) value;  }  //LIGNE RAJOUTÉE  else if(value instanceof JComboBox){  return (JComboBox) value;  }  else  return this;  }  }  Une autre façon d'initialiser un tableau :  //data et title sont toujours nos tableaux d'objets !  JTable tableau = new JTable(new DefaultTableModel(data, title)); |
|  |  |