Bases de Swing Composants, Conteneurs

1. PRINCIPE
2. **LES CONTENEURS DE HAUT NIVEAU**
3. LES CONTENEURS DE NIVEAU INTERMÉDIAIRE
4. LES COMPOSANTS
5. LES COMPOSANTS DE MENU

Elle permet de réaliser des interfaces en devenant le réceptacle où viendront se placer les divers composants d'interface. Elle hérite de la classe Component et ajute les méthodes propres à la gestion des fenêtres suivantes :

JFrame(String) qui construit la fenêtre avec son titre

Container getContentPane() qui retourne le contenant associé à cette fenêtre qui servira de base à tout ce que l'on y placera (composants ou autres contenants).Si on ne l'a pas modifié, ce contenant est de la classe **JPanel.**

setContentPane(Container) qui redéfinit le contenant associé à cette fenêtre qui servira de base à tout ce que l'on y placera (composants ou autres contenants)

Image getIconImage() retourne l'image utilisée comme icône de cette fenêtre

void setIconImage(Image) définit l'image utilisée comme icône de cette fenêtre

String getTitle() retourne le titre de la fenêtre

void setTitle(String) définit le titre de la fenêtre

void setJMenuBar(MenuBar) définit la barre de menu de la fenêtre

void pack() donne à chaque élément contenu dans la fenêtre sa taille préférentielle

void setVisible(boolean) qui rend la fenêtre visible à l'écran

void addWindowListener(WindowListener) permet de prendre en compte les événements concernant la fenêtre (fermeture…)

Ce n'est pas directement dans un objet de classe **JFrame** que l'on place les composants mais dans un contenant qui lui est associé

La classe JFrame

Fermeture d'une Jframe: 4 modes possibles, à définir avec la méthode **setDefaultCloseOperation:**

- **DO_NOTHING_ON_CLOSE :Ignore la demande de fermeture**
- **HIDE_ON_CLOSE: rend la fenêtre invisible** (mode par défaut)

— Cache la fenêtre sans quitter l'application

— Quand on a plusieurs fenêtres, on peut souhaiter fermer une fenêtre et continuer l'application

- **DISPOSE_ON_CLOSE: cache la fenêtre et libère** toutes les ressources systèmes associées; elles seront réallouées si la fenêtre redevient visible
- **EXIT_ON_CLOSE: termine le programme et quitte l'appli**

Exemple:

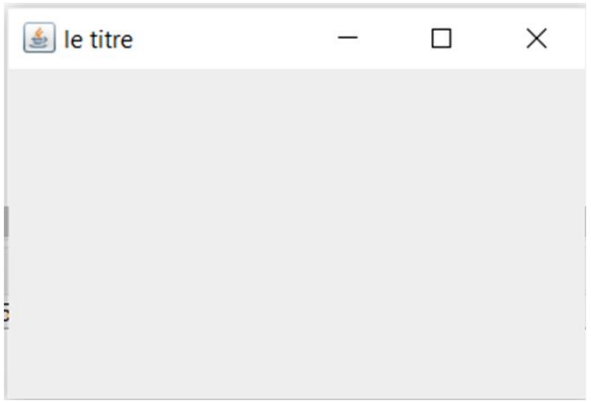
JFrame f=**new JFrame("Hello world!"); …**

f.setDefaultCloseOperation(JFrame.*EXIT_ON_CLOSE);*

La classe JFrame

- L'exemple suivant démarre une application avec titre

```
import java.awt.Dimension;
import javax.swing.JFrame;
public class Appli0 {
   public static void main(String[] args) { // création de l'application
 JFrame f = new JFrame();
// affectation du titre
   f.setTitle("le titre");
// affectation de l'opération à effectuer lors de la fermeture de la fenêtre
   f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
// taille et position
   f.setPreferredSize(new Dimension(300, 200));
   f.setLocation(100,100); // la fenêtre est en position 100, 100
   f.setLocationRelativeTo(null); // la fenêtre est centrée à l'écran
   f.pack(); // pack fait en sorte que tous les composants
        //de l'application soient à leur preferredSize, ou au-dessus
   f.setVisible(true); // rendre la fenêtre visible,
} }
```



- La classe *JDialog* permet de créer des boîtes de dialogue, qui permettent des interactions avec l'utilisateur de l'application.Les dialogues sont des conteneurs de premier niveau (comme *JFrame*) : ils ne sont pas contenus dans d'autres cadres, mais dépendent d'un autre cadre. Lorsqu'un cadre est fermé (ou iconifié) les cadres qui en dépendent sont fermés (ou iconifiés).

Un dialogue peut être :

- **modal** : lorsqu'il est actif toute interaction avec les autres fenêtres sont bloquées. Les dialogues *JOptionPane* sont modaux
- **non modal** : ne bloque pas les interactions avec les autres fenêtres : pour créer un dialogue non modal, on est obligé de passer par la classe *JDialog*.

Elle est utilisée pour faire apparaître des fenêtres temporaires permettant d'afficher un message ou de faire une saisie. Il est possible de rendre ces fenêtres prioritaires dans le sens où on ne peut rien faire d'autre tant qu'on ne les a pas fermées. Ceci permet d'obliger la lecture d'un message ou une saisie avant toute autre utilisation de l'interface principale. Elle hérite de la classe **Component** et ajoute les méthodes propres à la gestion des fenêtres suivantes :

JDialog (Frame, boolean) Construit une boîte de dialogue sans titre attachée à la fenêtre passée en premier paramètre. Le second paramètre indique si cette fenêtre est ou non prioritaire (modale). Si ce dernier est omis, sa valeur est false et la fenêtre de dialogue est non prioritaire, non modale.

JDialog (Frame, String, boolean) Construit une boîte de dialogue dépendant d'une autre fenêtre, avec un titre, et une modalité.

La classe JDialog

Elle a les mêmes méthodes que **Jframe**

Container getContentPane() qui retourne le contenant associé à cette fenêtre qui servira de base à tout ce que l'on y placera (composants ou autres contenants).Si on ne l'a pas modifié, ce contenant est de la classe **JPanel.**

setContentPane(Container) qui redéfinit le contenant associé à cette fenêtre qui servira de base à tout ce que l'on y placera (composants ou autres contenants)

void dispose() supprime la fenêtre, elle disparait à l'écran

String getTitle() retourne le titre de la fenêtre

void setTitle(String) définit le titre de la fenêtre

boolean isResizable() indique si la fenêtre peut être redimensionnée ou pas

void setResizable(boolean) autorise ou non le redimensionnement de la fenêtre

void setJMenubar(MenuBar) définit la barre de menu de la fenêtre

void setLocationRelativeTo(Component) définit la position de la fenêtre de dialogue comme relative au composant passée en paramètre, elle sera centrée sur ce composant

void pack() donne à chaque élément contenu dans la fenêtre sa taille préférentielle

void setVisible(boolean) qui rend la fenêtre visible à l'écran

void addWindowListener(WindowListener) permet de prendre en compte les événements concernant la fenêtre (fermeture…)

La classe JDialog

Void setDefaultCloseOpeation(int) définit l'opération qui sera faite lorsque l'user tente de fermer le dialogue. 4 modes possibles, à définir avec la méthode **setDefaultCloseOperation:**

- **JDialog.DO_NOTHING_ON_CLOSE**
- **JDialog.HIDE_ON_CLOSE: rend la fenêtre invisible** (mode par défaut)
- **JDialog.DISPOSE_ON_CLOSE: cache la fenêtre et libère** toutes les ressources systèmes associées; elles seront réallouées si la fenêtre redevient visible
- **JDialog.EXIT_ON_CLOSE: termine le programme**

Ce n'est pas directement dans un objet de classe **JDialog** que l'on place les composants mais dans un contenant qui lui est associé

La classe JDialog

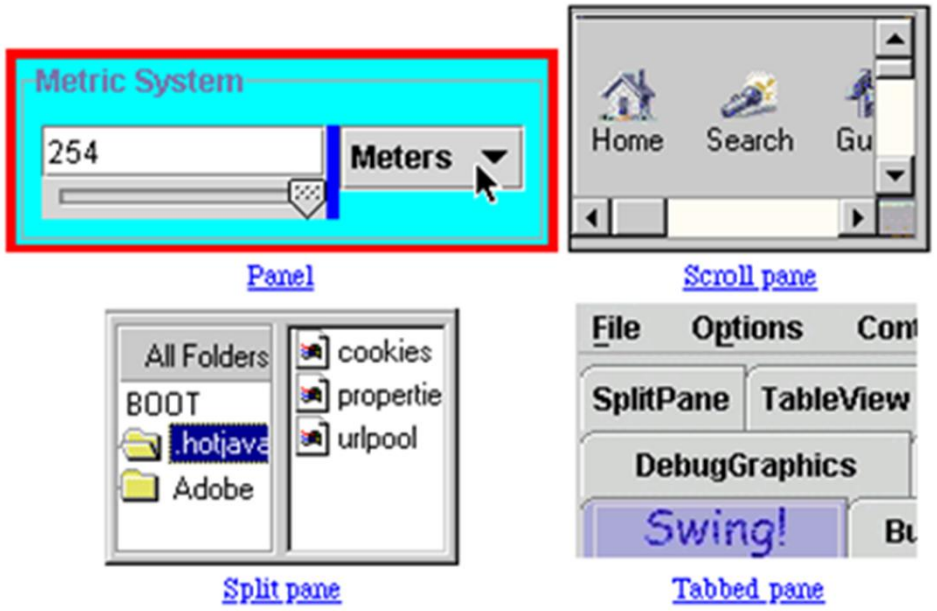
Bases de Swing Composants, Conteneurs

1. PRINCIPE
2. **LES CONTENEURS DE HAUT NIVEAU**
3. **LES CONTENEURS DE NIVEAU INTERMÉDIAIRE**
4. LES COMPOSANTS
5. LES COMPOSANTS DE MENU

Les conteneurs de niveau intermédiaire

Les contenants sont des objets appartenant à des classes qui héritent de **Container** qui elle-même hérite de **Component.** Les principaux contenants définis par SWING sont :

- JPanel
- JScrollPane
- JSplitPane
- JTabbedPane



Dans ces contenants on ajoute les composants de l'interface ou d'autres contenants.

Les conteneurs de niveau intermédiaire

Les principales méthodes communes à toutes ces classes :

void setLayout(LayoutManager) qui associe au contenant l'objet de placement passé en paramètre

void add(Component) qui ajoute ce composant au contenant sans désignation de zone lorsque l'objet de placement n'en a pas besoin (**FlowLayout, GridLayout ou GridBagLayout**)

void add(Component, Object) qui ajoute ce composant au contenant en utilisant une désignation de zone qui dépend de l'objet de placement qui a été associé à ce contenant (**int pour un BorderLayout**)

void remove(Component) qui enlève ce composant au contenant

void removeAll() qui enlève tous les composants au contenant

Component locate(int, int) retourne le composant situé aux coordonnées données en paramètre

La classe JPanel

Elle correspond au contenant le plus simple. Elle offre deux constructeurs:

JPanel() Création d'un panneau avec *FlowLayout* comme gestionnaire de placement par défaut.

JPanel(LayoutManager lm) Création d'un panneau avec *lm* comme gestionnaire de placement.

Void updateUI() permet le rafraichissement du panel

La classe JScrollPane

C'est une version améliorée de **Jpanel** qui possède des ascenseurs de défilement vertical et horizontal. Ses principales méthodes sont :

JScrollPane() crée un **JScrollPane** vide

JScrollPane(Component) crée un **JScrollPane** contenant un seul composant (celui passé en paramètre)

JScrollPane(int, int) crée un **JScrollPane** vide en précisant le comportement des ascenseurs

JScrollPane(Component, int, int) crée un **JScrollPane** contenant un seul composant (celui passé en paramètre) en précisant le comportement des ascenseurs

Le premier entier définit le comportement de l'ascenseur vertical,
il peut prendre les valeurs:

- *VERTICAL_SCROLLBAR_AS_NEEDED* l'ascenseur vertical n'est visible que si il est nécessaire.
- *VERTICAL_SCROLLBAR_NEVER* l'ascenseur vertical n'est jamais visible
- *VERTICAL_SCROLLBAR_ALWAYS* l'ascenseur vertical est toujours visible

Le dernier entier définit le comportement de l'ascenseur horizontal, il peut prendre les valeurs:

- *HORIZONTAL_SCROLLBAR_AS_NEEDED* la *ScrollBar* horizontale n'est visible que si elle est nécessaire.
- *HORIZONTAL_SCROLLBAR_NEVER* la *ScrollBar* horizontale n'est jamais visible
- *HORIZONTAL_SCROLLBAR_ALWAYS* la *ScrollBar* horizontale est toujours visible

Elle permet de définir deux zones adjacentes avec une barre de séparation pouvant être déplacée à la souris. Ces zones peuvent être cote à cote ou l'une sous l'autre. Ses principales méthodes sont :

JSplitPane(int) qui construit une **JSplitPane** avec une indication d'orientation. Le paramètre peut être JSplitPane.VERTICAL_SPLIT ou JSplitPane.HORIZONTAL_SPLIT

JSplitPane(int,Component,Component) qui construit une **JSplitPane** avec une indication d'orientation en premier paramètre (JSplitPane.VERTICAL_SPLIT ou JSplitPane.HORIZONTAL_SPLIT), et les deux composants à y placer.

Remarque : On peut ajouter à ces deux constructeurs un dernier paramètre booléen qui indique si, lors du déplacement de la barre de séparation des deux zones, leurs contenus doivent être redessinés en permanence (true) ou seulement lorsque la barre cessera de bouger (false).

Component getLeftComponent() qui retourne le composant de gauche (ou celui de dessus).

Component getRightComponent() qui retourne le composant de droite (ou celui de dessous).

Component getTopComponent() qui retourne le composant de dessus (ou celui de droite).

Component getBottomComponent() qui retourne le composant de dessous (ou celui de gauche).

int getDividerSize() qui retourne la taille en pixels de la séparation.

int getDividerLocation() qui retourne la position de la séparation (en pixels).

int getMaximumDividerLocation() qui retourne la position maximale possible de la séparation (en pixels).

int getMinimumDividerLocation() qui retourne la position minimale possible de la séparation (en pixels).

int setDividerLocation(double) qui positionne la séparation. La valeur est un réel représentant un pourcentage de 0.0 à1.0.

void setDividerSize(int) qui définit l'épaisseur de la barre de séparation (en pixels).

La classe JSplitPane

Elle permet de placer les éléments selon des onglets. Un onglet est un *JComponent* (en général un *JPanel*)

Un onglet a

- un titre
- une icône
- un *TooltipText*
- un indice dans la liste des onglets

Ses principales méthodes sont :

JTabbedPane(int) crée l'objet en précisant où se placent les onglets. Le paramètre peut prendre les valeurs : JTabbedPane.TOP, JTabbedPane.BOTTOM, JTabbedPane.LEFT,JTabbedPane.RIGHT

void addTab(String, Component) ajoute onglet dont le libellé est donné par le premier paramètre et y place le composant donné en second paramètre.

void addTab(String, ImageIcon,Component) ajoute onglet dont le libellé est donné par le premier paramètre et auquel est associé une icône (second paramètre) et y place le composant donné en dernier paramètre.

void addTab(String, ImageIcon,Component, String) ajoute onglet dont le libellé est donné par le premier paramètre et auquel est associé une icône (second paramètre) et y place le composant donné en troisième paramètre. Le dernier est une bulle d'aide qui apparaîtra lorsque la souris passera sur l'onglet.

void insertTab(String, ImageIcon,Component, String, int) insère un onglet dont le libellé est donné par le premier paramètre et auquel est associée une icône (second paramètre) et y place le composant donné en troisième paramètre. Le quatrième paramètre est une bulle d'aide. Le dernier indique la position à laquelle doit être inséré l'onglet.

La classe JTabbedPane

void removeTabAt(int) qui supprime l'onglet désigné.
void setIconAt(int, ImageIcon) associe une icône à l'onglet désigné.
int getSelectedIndex() qui retourne le numéro de l'onglet sélectionné
void setSelectedIndex(int) qui sélectionne l'onglet dont le numéro est passé en paramètre.

int getTabCount() qui retourne le nombre d'onglets disponibles.
int indexOfTab(ImageIcon) retourne le premier onglet associé à l'icône passée en paramètre
int indexOfTab(String) retourne le premier onglet associé au nom passé en paramètre

Remarque : les deux méthodes précédentes peuvent lever une exception de classe **ArrayIndexOutOfBoundsException** si la position de l'onglet n'est pas correcte.

String getTitleAt(int) qui retourne le libellé de l'onglet désigné
void setTitleAt(int,String) qui définit le libellé de l'onglet désigné.
boolean isEnabledAt(int) qui indique si l'onglet désigné est accessible ou pas.
boolean setEnabledAt(int,boolean) qui rend accessible ou pas l'onglet désigné.
int getTabCount() qui retourne le nombre d'onglets.
int indexOfTab(String) qui retourne le numéro correspondant à l'onglet désigné par son libellé.
void setBackgroundAt(int,Color) qui définit la couleur de fond pour la page correspondant à l'onglet désigné par le premier paramètre.
void setForegroundAt(int,Color) qui définit la couleur de tracé pour la page correspondant à l'onglet désigné par le premier paramètre.

void addChangeListener(ChangeListener) pour associer l'objet qui traitera les changements d'onglet

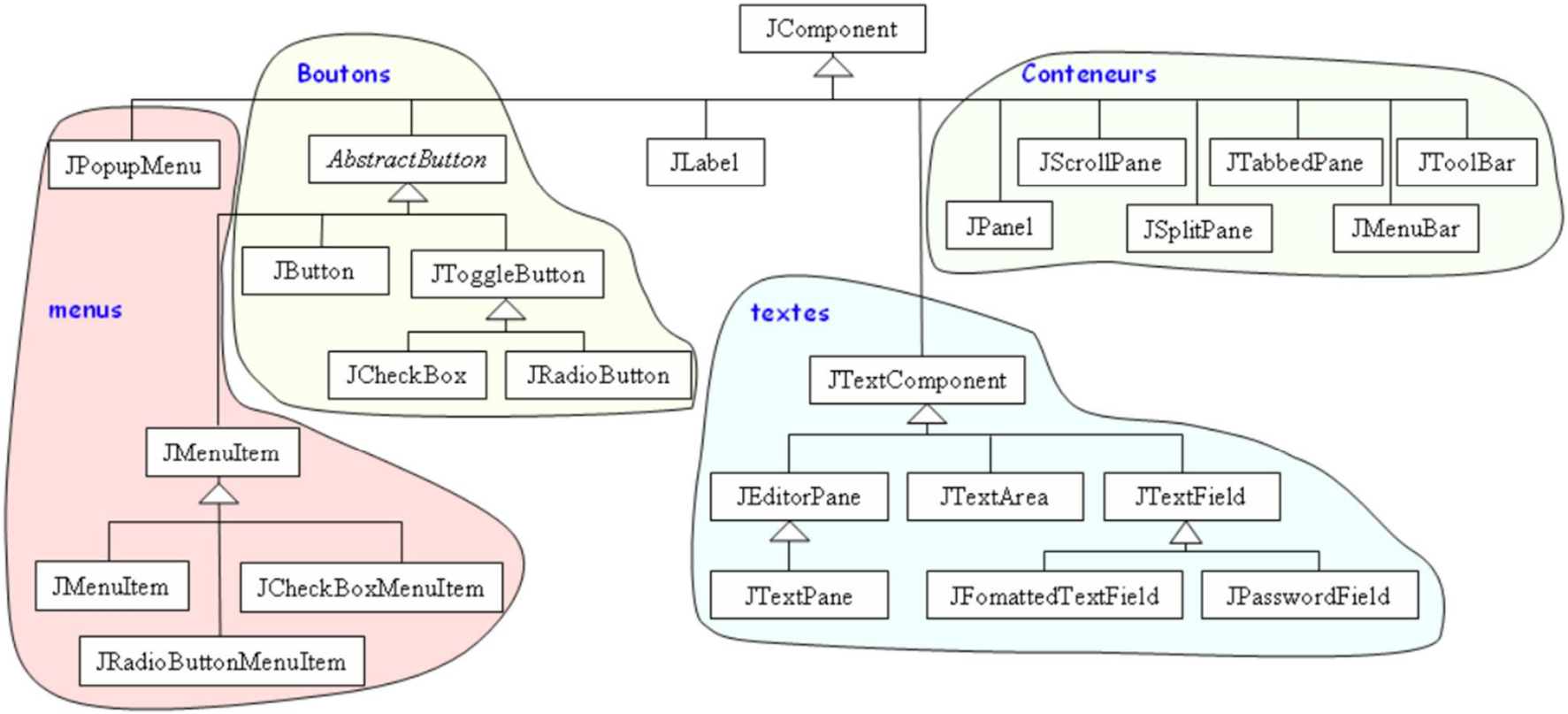
La classe JTabbedPane

Bases de Swing Composants, Conteneurs

1. PRINCIPE
2. LES CONTENEURS DE HAUT NIVEAU
3. LES CONTENEURS DE NIVEAU INTERMÉDIAIRE
4. **LES COMPOSANTS**
5. LES COMPOSANTS DE MENU

Les classes Component et JComponent

JComponent est la classe de base de tous les composants *Swing*, à part *JFrame*, *JDialog* et *JApplet*.



Les classes Component et JComponent

int getx() et **int gety()** retournent les coordonnées de ce composant

void setLocation(int , int) définit la position du composant

Celles propres à la classe JComponent :

void setPreferredSize(Dimension) définit la taille préférentielle du composant

void setMinimumSize(Dimension) définit la taille minimale du composant

void setMaximumSize(Dimension) définit la taille maximale du composant

void setToolTipText(String) définit le texte qui sera affiché lorsque la souris passera sur ce composant. Ce texte disparaît lorsque la souris s'éloigne du composant ou au bout d'un certain délai.

Les composants de l'interface

Notre fenêtre que l'on va définir (héritée de **JFrame**) contiendra en tant que **membres** les **noms des composants** d'interface (boutons, zones de texte,…) dont on souhaite disposer.

Il conviendra ensuite de les **initialiser** et de définir **leur disposition**. Pour cela, le **constructeur de la classe d'interface** (via une méthode) devra se charger:

- De **créer** les objets associés (**new**)
- De les **initialiser** par **leurs constructeurs** ou par tout autre moyen disponible
- De les **placer**
- De leur **associer les actions** liées aux évènements qu'ils peuvent recevoir

Composants Texte de la Swing

JTextField

```
JTextField numTel = new JTextField("",10);
```

JTextArea

```
JTextArea commentaire=new JTextArea("Enter more text to see scrollbars", 10, 10);
```

Enter more text to see scrollbars

JLabel et JButton

JLabel

```
JLabel texte=new JLabel("Texte");
```

Texte

JButton

```
JButton bouton=new JButton("OK");
```

OK

JCheckBox

```
JCheckBox box1=new JCheckBox("BOX1", true);
JCheckBox box2=new JCheckBox("BOX2");
```

BOX1 BOX2

JRadioButton

pour cocher un élément par défaut (facultatif)

```
JRadioButton radio1=new JRadioButton("Radio1", true);
JRadioButton radio2=new JRadioButton("Radio2");
ButtonGroup groupRadio=new ButtonGroup();
groupRadio.add(radio1);
groupRadio.add(radio2);
```

Radio1 Radio2

Groupe : pour gérer choix exclusif (facultatif)

La classe JButton

Elle permet de définir des boutons sur lesquels on peut cliquer. Ses principales méthodes sont :

JButton(String) construction avec définition du texte contenu dans le bouton

JButton(ImageIcon) construction avec une icone dans le bouton

JButton(String,ImageIcon) construction avec définition du texte et d'une icône dans le bouton

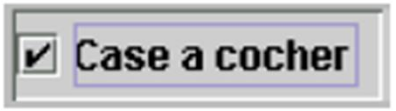
String getText() qui retourne le texte contenu dans le bouton

void setText(String) qui définit le texte contenu dans le bouton

void addActionListener(ActionListener) pour associer l'objet qui traitera les clics sur le bouton

La classe JCheckBox

Elle permet de définir des cases à cocher. Ses principales méthodes sont :



JCheckBox(String) construction avec définition du texte contenu dans la case à cocher

JCheckBox(String,boolean) construction avec en plus définition de l'état initial de la case à cocher

boolean isSelected() qui retourne l'état de la case à cocher (cochée ou non).

void setSelected(boolean) qui définit l'état de la case à cocher (cochée ou non).

String getText() qui retourne le texte contenu dans la case à cocher

void setText(String) qui définit le texte contenu dans la case à cocher

void addActionListener(ActionListener) pour associer l'objet qui traitera les actions sur la case à cocher

La classe JRadioButton

- Un bouton radio est un bouton qui peut être sélectionné ou non.
  Un *ButtonGroup* permet de grouper des boutons radio, de façon qu'il n'y ait qu'un seul bouton sélectionné dans le groupe. Un bouton radio est muni d'un texte, et éventuellement d'icônes :
- *setIcon(...);* : l'icône "normale" du bouton radio.
- *setSelectedIcon(...);* : l'icône quand le bouton est sélectionné.
- *setPressedIcon(...);* : l'icône quand le bouton est pressé
- Exemple :

```
ButtonGroup bg=new ButtonGroup();
        JRadioButton br1 = new JRadioButton("un");
        JRadioButton br2 = new JRadioButton("deux");
        JRadioButton br3 = new JRadioButton("trois");
        // ajout des boutons radio dans le groupe bg
        bg.add(br1); bg.add(br2); bg.add(br3);
```

La méthode *isSelected()* permet de savoir si le bouton radio est sélectionné ou non.

La classe JLabel

Elle permet de définir des textes fixes. Ses principales méthodes sont :

JLabel(String) qui construit le titre avec son contenu
void setText(String) qui définit le texte
String getText() qui retourne le texte

JLabel(String, Icon, int)	Crée une instance de JLabel avec le texte, l'image et l'alignement horizontal spécifiés
void setFont(Font f)	Affecte une police au label.
Font getFont()	Retourne la police du label.
void setOpaque(**boolean** b)	Rend le label opaque si *b* vaut *true*,par défaut un label est transparent.
boolean isOpaque()	Retourne la propriété opaque du label.
void setIcon(Icon i)	Affecte une icône au label.
Icon getIcon()	Retourne l'icône du label.
void setHorizontalAlignment(int a)	Affecte l'alignement horizontal : LEFT, RIGHT, CENTER, LEADING, TRAILING
void setVerticalAlignment(int a)	Affecte l'alignement vertical : TOP, BOTTOM, CENTER

La classe JComboBox

Elle permet de définir des boîtes permettant de choisir une valeur parmi celles proposées. Ses principales méthodes sont :

JComboBox(Object[]) construction avec définition de la liste. On peut utiliser un tableau de chaînes de caractères (**String**) ou de toute autre classe d'objets.
void addItem(Object) qui ajoute une valeur possible de choix
int getSelectedIndex() qui retourne le numéro du choix actuel
Object getSelectedItem() qui retourne l'objet associé au choix actuel. Attention l'objet retourné est de classe **Object**, il faudra utiliser l'opérateur de coercition pour le transformer en sa classe d'origine (**String** par exemple).
void setSelectedIndex(int) qui sélectionne un élément défini par son numéro
void addActionListener(ActionListener) pour associer l'objet qui traitera les choix faits

La classe JList

Elle permet de définir des listes non déroulantes (pour disposer de listes avec ascenseur il faut faire appel à un contenant possédant des ascenseurs comme **JScrollPane**). La selection dans ces listes peut porter sur 1 ou plusieurs objets. Ses principales méthodes sont :

JList(Object[]) construction avec définition de la liste. On peut utiliser un tableau de chaînes de caractères (**String**) ou de toute autre classe d'objets.

setListData(Object[]) définition de la liste. On peut utiliser un tableau de chaînes de caractères (**String**) ou de toute autre classe d'objets.

void setVisibleRowCount(int) qui définit le nombre d'éléments visibles sans ascenseur

int getSelectedIndex() qui retourne le numéro du premier élément sélectionné

int[] getSelectedIndices() qui retourne les numéros des éléments sélectionnés

Object getSelectedValue() qui retourne le premier objet selectionné. Attention l'objet retourné est de classe **Object**, il faudra utiliser l'opérateur de coercition pour le transformer en sa classe d'origine (**String** par exemple).

Object[] getSelectedValues() qui retourne les objets actuellement sélectionnés.

void setSelectedIndex(int) qui sélectionne l'élément désigné par son numéro

void setSelectedIndices(int[]) qui sélectionne les éléments désignés par leurs numéros

void clearSelection() qui annulle toutes les sélections

void addListSelectionListener(ListSelectionListener) pour associer l'objet qui traitera les sélections dans la liste

La classe JScrollbar

Elle permet de définir des ascenseurs horizontaux ou verticaux. Ses principales méthodes sont :

JScrollbar(int,int,int,int,int) qui définit l'ascenseur avec dans l'ordre des paramètres son orientation (on peut y mettre les constantes **JScrollbar.HORIZONTAL** ou **JScrollbar.VERTICAL**), sa position initiale, le pas avec lequel on le déplace en mode page à page, ses valeurs minimales et maximales.

int getValue() qui retourne la positon actuelle de l'ascenseur

void setValue(int) qui définit la position de l'ascenseur

int getBlockIncrement() qui retourne la valeur utilisée pour le pas en mode page à page

void setBlockIncrement (int) qui définit la valeur utilisée pour le pas en mode page à page

int getUnitIncrement() qui retourne la valeur utilisée pour le pas unitaire

void setUnitIncrement (int) qui définit la valeur utilisée pour le pas unitaire

int getMaximum() qui retourne la valeur maximale actuelle

void setMaximum (int) qui définit la valeur maximale actuelle

int getMinimum() qui retourne la valeur minimale actuelle

void setMinimum (int) qui définit la valeur minimale actuelle

void addAdjustmentListener(AdjustmentListener) pour associer l'objet qui traitera les déplacements de l'ascenseur

Les classes JTextField et JTextArea

Elles permettent de définir des zones de texte éditable sur une ou plusieurs lignes. Elles ont en commun un certain nombre de méthodes dont voici les principales :

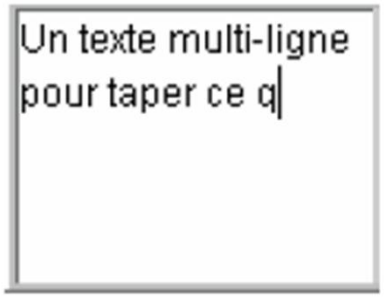
void copy() qui copie dans le bloc-notes du système la partie de texte sélectionnée
void cut() qui fait comme **copy** puis supprime du texte la partie sélectionnée
void paste() qui fait copie dans le texte le contenu du bloc-notes du système
String getText() qui retourne le texte contenu dans la zone de texte
void setText(String) qui définit le texte contenu dans la zone de texte
int getCaretPosition() qui retourne la position du curseur d'insertion dans le texte (rang du caractère)
int setCaretPosition(int) qui place le curseur d'insertion dans le texte au rang indiqué (rang du caractère)
int moveCaretPosition(int) qui déplace le curseur d'insertion dans le texte en sélectionnant le texte depuis la position précédente.
setEditable(boolean) qui rend la zone de texte modifiable ou pas
int getSelectionStart() qui retourne la position du début du texte sélectionné (rang du caractère)
int getSelectionEnd() qui retourne la position de la fin du texte sélectionné (rang du caractère)
void setSelectedTextColor(Color c) qui définit la couleur utilisée pour le texte sélectionné
String getSelectedText() qui retourne le texte sélectionné
void select(int,int) qui sélectionne le texte compris entre les deux positions données en paramètre
void selectAll() qui sélectionne tout le texte
Document getDocument() qui retourne le gestionnaire de document associé à cette zone de texte. C'est ce gestionnaire qui recevra les événements de modification de texte
void addCaretListener(CaretListener) pour associer l'objet qui traitera les déplacements du curseur de saisie dans le texte

La classe JTextArea

Elle permet de définir des zones de texte sur plusieurs lignes modifiables sans ascenseurs (pour disposer d'ascenseurs il faut faire appel à un contenant en possédant

JTextArea(String) qui crée une zone de texte avec un contenu initial

JTextArea(String,int,int) qui crée une zone de texte avec un contenu initial et précise le nombre de lignes et de colonnes de la zone de texte



void append(String) qui ajoute la chaîne à la fin du texte affiché

void insert(String,int) qui insère la chaîne au texte affiché à partir du rang donné

void setTabSize(int) qui définit la distance entre tabulations.

void setLineWrap(boolean) qui détermine si les lignes longues doivent ou non être repliées.

void setWrapStyleWord(boolean) qui détermine si les lignes sont repliées en fin de mot (true) ou pas.

Remarque : Lorsque l'on saisit du texte dans une zone de texte celle-ci adapte sa taille au texte saisi au fur et à mesure. Ce comportement n'est pas toujours souhaitable, on peut l'éviter en mettant ce composant dans un **JScrollPane** pour disposer d'ascenseurs.

La classe Document

Prend en charge l'édition de texte. Ses principales méthodes sont :

int getLength() qui retourne le nombre de caractères du document

String getText(int,int) qui retourne la partie du texte qui commence à la position donnée en premier paramètre et dont la longueur est donnée par le second paramètre

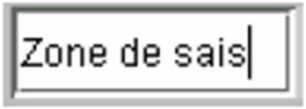
void removeText(int,int) qui supprime la partie du texte qui commence à la position donnée en premier paramètre et dont la longueur est donnée par le second paramètre

void addDocumentListener(DocumentListener) pour associer l'objet qui traitera les modifications du texte

Remarque : Ces deux dernières méthodes peuvent lever une exception de classe **BadLocationException** si les paramètres sont incorrects

La classe JTextField

Elle permet de définir des zones de texte sur une seule ligne modifiables Ses principales méthodes sont :



JTextField(String) qui la crée avec un contenu initial

JTextField(String,int) qui la crée avec un contenu initial et définit le nombre de colones

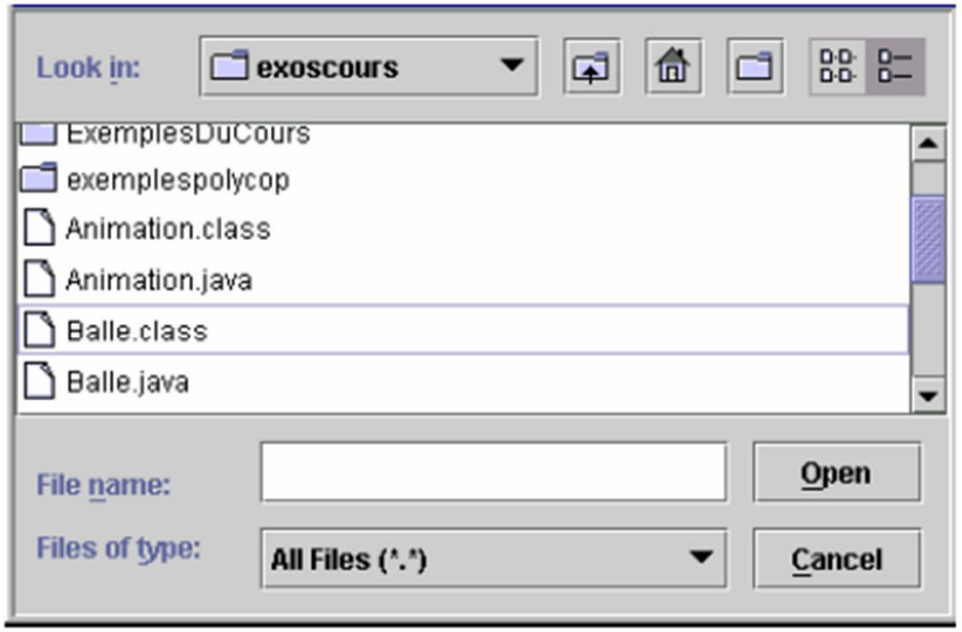
void addActionListener(ActionListener) pour associer l'objet qui traitera les modifications du texte

Remarque : Lorsque l'on saisit du texte dans un tel composant celui-ci adapte sa taille au texte saisi au fur et à mesure. Ce comportement n'est pas toujours souhaitable, on peut l'éviter en lui définissant une taille préférentielle par sa méthode **setPreferredSize** et une taille minimale par sa méthode **SetMinimumSize**

On procédera comme suit : texte.**setPreferedSize**(texte.**getPreferredSize**()) ;
texte.**setMinimumSize**(texte.**getPreferredSize**()) ;

La classe JFileChooser

Elle permet de faire apparaître une zone de choix de fichier capable de gérer le parcours des répertoires. Ses principales méthodes sont :



- **JFileChooser()** construction avec le répertoire courant comme point de départ.
- **JFileChooser(String)** construction avec comme point de départ le répertoire donné en paramètre
- **File getSelectedFile()** qui retourne le fichier sélectionné
- **File[] getSelectedFiles()** qui retourne les fichiers sélectionnés quand on peut en sélectionner plusieurs
- **Void setDialogTitle(String)** qui définit le titre de la zone de choix de fichier
- **void setFileSelectionMode(int)** qui définit le mode de sélection de fichier. La paramètre peut être **JFileChooser.**FILES_ONLY, **JFileChooser.**DIRECTORIES_ONLY ou **FileChooser.**FILES_AND_DIRECTORIES pour autoriser seulement les fichiers, seulement les répertoires ou les deux.
- **Void setMultipleSelectionEnabled(boolean)** qui autorise ou interdit les sélections multiples.
- **void addActionListener(ActionListener)** pour associer l'objet qui traitera l'événement de sélection d'un fichier