



Composants de base des IHM en Java / SWING

Cours IHM
Frédéric Moal
2011/2012



Objectif

Donner aperçu de l'utilisation des
composants de base de SWING
en particulier :

- boutons
- zones de texte
- fenêtres de dialogue
- menus



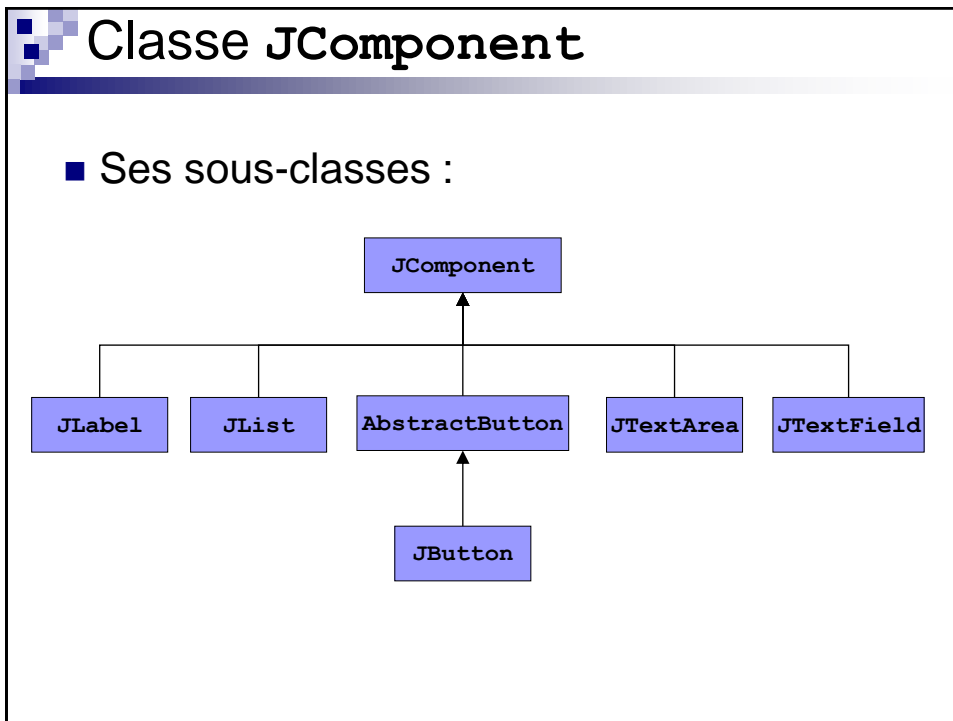
Plan de cette présentation

- Classe **JComponent**
- Boutons
- Composants pour du texte
- Fenêtres de dialogue
- Menus et barres d'outils
- Barres de progression
- Autres composants et décorations divers
- Classe **SwingUtilities**



Plan

- Classe **JComponent**
- Boutons
- Composants pour du texte
- Fenêtres de dialogue
- Menus et barres d'outils
- Barres de progression
- Autres composants et décorations divers
- Classe **SwingUtilities**



Classe JComponent

- La classe **JComponent** dérive des classes (AWT) **Component** et **Container** et elle ajoute elle-même de nombreuses méthodes
- Elle offre donc de très nombreuses fonctionnalités qui sont héritées par tous les composants graphiques



Fonctionnalités de base

■ *Bulle d'information*

une bulle peut être affichée quand le pointeur de souris est depuis un certain temps au dessus d'un composant

■ *Association de touches*

on peut associer des actions aux frappes de certaines touches du clavier quand le focus appartient à un composant



Fonctionnalités de base

■ *Dimensionner un composant*

des méthodes permettent de donner les dimensions préférées, minimum et maximum des composants (pas nécessairement utilisées par les layout managers)

■ *Bordures*

on peut ajouter une bordure à un composant

■ *Transparence*

par défaut les composants sont opaques (le "fond" est dessiné) mais on peut les rendre transparents (**`setOpaque(false)`**)

Fonctionnalités de base

■ *Validation*

- un composant peut être mis "hors-service" ou remis en service en utilisant la méthode **setEnabled(boolean)**
- Quand le composant est hors-service, il est affiché en grisé et il ne répond plus aux actions de l'utilisateur

Fonctionnalités techniques

■ Double buffer

les composants sont dessinés dans un buffer avant d'être affichés à l'écran ; on évite ainsi des scintillements désagréables

■ Facilités pour la mise au point

une option permet de demander un affichage lent des composants, avec couleur spéciale et clignotement pour repérer plus facilement les erreurs dans la programmation des interfaces graphiques (**setDebugGraphicsOptions**)



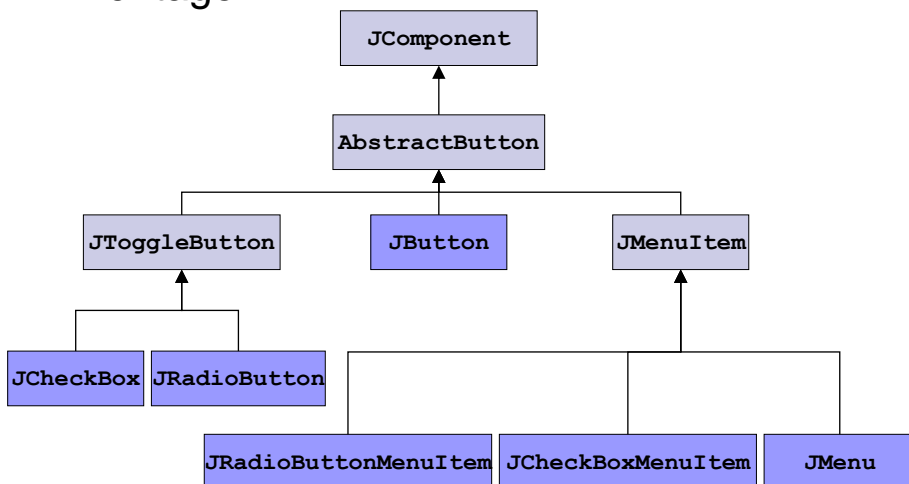
Plan

- Classe **JComponent**
- Boutons
- Composants pour du texte
- Fenêtres de dialogue
- Menus et barres d'outils
- Barres de progression
- Autres composants et décorations divers
- Classe **SwingUtilities**

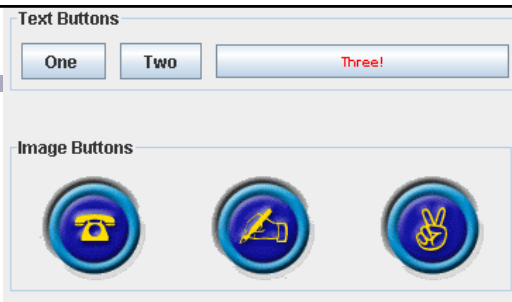


Les boutons

■ Héritage



Boutons



- Un bouton peut contenir du texte (qui peut être du code HTML) mais aussi une image (et on peut choisir la position relative des 2)
- Le caractère souligné du texte indique le caractère mnémonique (pour simuler un clic avec le clavier)
- Un bouton peut être invalidé (texte en « gris »)

HTML dans les boutons

- Les dernières versions de swing ont ajouté la possibilité d'inclure du code HTML dans les labels et les boutons (tous les types de boutons)
- Il suffit pour cela d'inclure le texte dans les balises **<html>** et **</html>** (en fait, seul le **<html>** du début est indispensable) :

```
new JButton("<html>Ligne1<p>Ligne2</html>")
```

HTML dans les composants Swing

Actuellement, les composants suivants permettent l'utilisation de texte HTML :

`JButton`, `JLabel`, `JMenuItem`,
`JMenu`, `JRadioButtonMenuItem`,
`JCheckBoxMenuItem`, `JTabbedPane`,
`JToolTip`, `JToggleButton`,
`JCheckBox`, `JRadioButton`

Code pour un bouton

```
JButton b1, b2, b3;

ImageIcon leftIcon = new
    ImageIcon("images/right.gif");

b1 = new JButton("Invalidier le bouton", leftIcon);

// Position par défaut : CENTER, RIGHT
b1.setVerticalTextPosition(AbstractButton.CENTER);
b1.setHorizontalTextPosition(AbstractButton.LEFT);
b1.setMnemonic('I');

b1.setActionCommand("invalidier");
. . .

b3.setEnabled(false);
```


Écouteur pour un bouton

```
b1.addActionListener(this);  
. . .  
}  
  
public void actionPerformed(ActionEvent e) {  
    if (e.getActionCommand().equals("invalider")) {  
        b2.setEnabled(false);  
        b1.setEnabled(false);  
        b3.setEnabled(true);  
    } else { . . . }  
}
```

Actions

- Dès que l'action effectuée quand on clique sur un bouton est partagée avec d'autres composants (menus, barre d'outils,...), il est intéressant de créer le bouton à partir d'une instance de **Action**



Interface Action

- Une action permet de centraliser
 - un texte (qui s'affiche sur les boutons ou les menus)
 - une icône
 - un traitement à exécuter
 - le fait que ce traitement est permis ou pas
 - un texte « `actionCommand` »
 - un mnémonique
 - un raccourci clavier
 - un texte qui décrit l'action (version longue ou courte utilisée par les bulles d'aide)



Utilisation des actions

- Cette action peut être utilisée par plusieurs composants de types éventuellement différents
- Certains attributs de ces composants sont alors fixés par l'action
 - le texte des boutons ou des menus
 - l'action leur est ajoutée comme **ActionListener**
 - ...

Classe et interface pour les actions

- L'interface **Action** (hérite de **ActionListener**) permet de représenter une telle action
- On héritera le plus souvent de la classe abstraite **AbstractAction**
- Des constructeurs de cette classe prennent en paramètres (optionnels) un texte et une icône

L'interface Action

- L'interface **Action** a les méthodes suivantes
 - **actionPerformed** (héritée de **ActionListener**)
 - **setEnabled** et **isEnabled** indiquent si l'action peut être lancée ou non
 - **putValue** et **getValue** permettent d'ajouter des attributs (paire "nom-valeur") à l'action
 - 2 attributs prédéfinis : **Action.NAME** et **Action.SMALL_ICON** (utilisés si l'action est associée à un menu ou à une barre d'outils)
 - **{add|remove}PropertyChangeListener** pour, par exemple, notifier un menu associé à une action que l'action est invalidée



Classes utilisant les Actions

- Les sous-classes de **AbstractButton** ; elles ont en particulier un constructeur qui prend en paramètre une action :
 - Boutons (**JButton**)
 - Boutons radio (**JRadioButton**), y compris dans un menu (**JRadioButtonMenuItem**)
 - Boîtes à cocher (**JCheckBox**), y compris dans un menu (**JCheckBoxMenuItem**)
 - Menu (**JMenu**)
 - Choix de menu (**JMenuItem**)



Utilisation des Actions

```
ImageIcon image = new ImageIcon("gauche.gif");

actionPrecedent =
new AbstractAction("Question précédente", image) {
    public void actionPerformed(ActionEvent e) {
        . . .
    }
};

. . .
JButton bPrecedent = new JButton(actionPrecedent);
panel.add(bPrecedent);
bPrecedent.setText("");
// bouton "image" avec tooltip
```



Utilisation des Actions

- On peut associer une action à un bouton (un **AbstractButton**, c'est-à-dire **JButton**, **JMenuItem**, **JToggleButton**) par la méthode **setAction(Action)**
- Cette méthode fait tout ce qu'on peut en attendre ; elle ajoute en particulier l'action comme écouteur du bouton, met le nom (propriété **NAME**), l'icône (**SMALL_ICON**), le texte de la bulle d'aide du bouton (**SHORT_DESCRIPTION**)
- La méthode **getAction()** récupère l'action



ToggleButton

- Représente les boutons qui ont 2 états : sélectionnés ou pas
- On utilise plutôt ses 2 sous-classes
 - – **CheckBox** : l'aspect visuel du bouton change suivant son état
 - – **RadioButton** : un seul bouton d'un groupe de boutons radio (**ButtonGroup**) peut être sélectionné à un moment donné

Aspect

The screenshot displays a Java Swing window titled "Aspect" with a light gray background. It contains four distinct widget sets, each enclosed in a rounded rectangle:

- Text Radio Buttons:** A group box containing three radio buttons labeled "Radio One", "Radio Two", and "Radio Three". The radio buttons are small circles with a blue border.
- Image Radio Buttons:** A group box containing three radio buttons labeled "Radio One", "Radio Two", and "Radio Three". The radio buttons are small circles with a blue border, each containing a grayscale image of a light bulb.
- Text CheckBoxes:** A group box containing three checkboxes labeled "One", "Two", and "Three". The checkboxes are small squares with a blue border.
- Image CheckBoxes:** A group box containing three checkboxes labeled "One", "Two", and "Three". The checkboxes are small squares with a blue border, each containing a grayscale image of a light bulb.

Traitement des événements

- Un clic sur un **ToggleButton** génère un **ActionEvent** et un **ItemEvent**
- Avec une boîte à cocher, le plus simple est de traiter les **ItemEvent** avec un **ItemListener**
- Mais les boîtes à cocher n'ont souvent pas d'écouteurs et on relève leur valeur quand on en a besoin avec **isSelected()**



Boîtes à cocher

```
JCheckBox boite = new JCheckBox("Label");
boite.addItemListener(itemListener);
// La boîte ne sera pas cochée
boite.setSelected(false);

. . .
// Récupère l'état de la boîte
boolean on = boite.isSelected();
```



ItemListener

```
class BoiteListener implements ItemListener
{
public void itemStateChanged(ItemEvent e) {
    Object source = e.getItemSelectable();
    if (source == boite) {
        if (e.getStateChange() ==
            ItemEvent.DESELECTED)
        . . .
    }
}
```



Bouton radio

- Avec les boutons radio, le plus simple est d'écouter avec un **ActionListener**
- Cet événement est engendré quand le bouton radio est sélectionné (pas quand il est désélectionné)
- L'utilisation d'un **ItemListener** serait plus complexe car la sélection d'un bouton du groupe désélectionne tous les autres et engendre donc un grand nombre de **ItemEvent**



Bouton radio

```
// Le groupe de boutons
ButtonGroup group = new ButtonGroup();
// Les boutons
JRadioButton b1 = new JRadioButton("Label1");
b1.setSelected(true); // Sélectionne le bouton
JRadioButton b2 = new JRadioButton("Label2");
// Ajoute les boutons au groupe
group.add(b1); . . .
// Ajoute les boutons dans l'interface graphique
panel.add(b1); . . .
// Ajoute les actionListeners ; les "actionCommands"
// peuvent différencier les boutons dans les
// listeners
b1.setActionCommand("label1"); . . .
b1.addActionListener(boutonListener); . . .
```




Plan

- Classe **JComponent**
- Boutons
- Composants pour du texte
- Fenêtres de dialogue
- Menus et barres d'outils
- Barres de progression
- Autres composants et décorations divers
- Classe **SwingUtilities**



Composant Texte

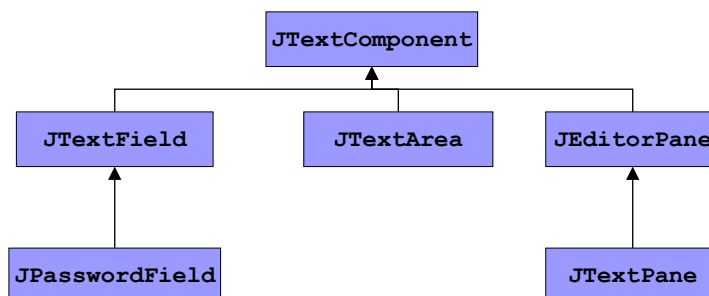
- **JLabel** : texte non modifiable par l'utilisateur
- **TextField** : entrer une seule ligne
- **PasswordField** : entrer un mot de passe non affiché sur l'écran
- **TextArea** : quelques lignes de texte avec une seule police de caractères

■ Composant texte formaté

- **JEditorPane** : affiche des pages à un format donné, avec plusieurs polices et des images et composants inclus
- Cette classe est en particulier adaptée aux format RTF (Microsoft) ou HTML
- **JTextPane** : sous-classe de **JEditorPane** ; documents décomposés en paragraphes et travaillant avec des styles nommés

■ Arbre d'héritage

- **JTextComponent** et **JLabel** héritent directement de **JComponent**





JTextComponent

- Dans le paquetage **javax.swing.text**
- Fonctionne sur le modèle MVC avec une instance de **javax.swing.text.Document** pour les données
- Elle fournit beaucoup de fonctionnalités à toutes les classes filles qui manipulent du texte (mais elles sont surtout utilisées avec les **JTextPane**)



JTextComponent : Fonctionnalités

- Un modèle séparé
- Une vue séparée
- Un contrôleur séparé, que l'on appelle un kit d'édition, qui lit et écrit du texte et offre des fonctionnalités grâce à des actions
- Possède des associations de clés (*key bindings*) pour lancer les actions
- Supporte les undo/redo



JTextComponent

- Contient les méthodes de base pour traiter une zone de saisie ou/et d'affichage de texte :
 - **{get/set}Text** pour obtenir ou mettre le texte (ou une partie du texte) contenu dans le composant
 - **setEditable()** pour indiquer si l'utilisateur peut modifier le texte
 - copier/couper/coller avec le *clipboard* du système
 - utilisation et gestion du point d'insertion
 - ...



JTextField

```
JTextField textfield = new JTextField("Texte initial");

textfield.addActionListener(new MyActionListener());

class MyActionListener implements ActionListener {
    public void actionPerformed(ActionEvent evt) {
        JTextField tf = (JTextField)evt.getSource();
        String texte = tf.getText();
        // Traitement du texte
        . . .
    }
}
```

! Taper la touche « Entrée » déclenche un **ActionEvent**. On peut aussi travailler directement avec **getText()** pour récupérer le texte

Mots de passe

```
JPasswordField pwf =  
    new JPasswordField("Texte initial");  
pwf.setEchoChar('#');  
pwf.addActionListener(actionListener);
```

- Les mots de passe doivent être récupérés dans un tableau de **char** et pas dans une **String** pour pouvoir les effacer de la mémoire après utilisation

Vérification mot de passe

```
public void actionPerformed(ActionEvent e) {  
    char[ ] mdp = pwf.getPassword();  
    if (isPasswordCorrect(mdp)) {  
        . . .  
    } else {  
        . . .  
    }  
    // On n'est jamais trop prudent...  
    java.util.Arrays.fill(mdp, 0);
```

Méthode privée pour
vérifier le mot de passe

JTextArea

- Constructeurs :

```
textArea = new  
    JTextArea("Ligne1\nLigne2");  
// nb de lignes et de colonnes en  
    paramètres  
textArea = new JTextArea(5, 40);
```

- Souvent la zone de texte dans un **ScrollPane** :

```
JScrollPane sc = new JScrollPane(textArea);
```

- **getText()** récupère le texte du **JTextArea**
- **setText(String)** donne la valeur du texte
- **append(String)** ajoute du texte à la fin

Les lignes dans un JTextArea

- Tous les composants qui peuvent contenir du texte utilisent le caractère **\n** pour passer à la ligne, quel que soit le système sous-jacent
- **setLineWrap(boolean)** indique si les lignes trop longues sont coupées ou affichées sur la ligne suivante (faux par défaut)
- **setWrapStyleWord(boolean)** indique si les mots seront coupés en fin de ligne ou non (faux par défaut)

Écouteur pour composant de texte

- Si on veut traiter tout de suite les différentes modifications introduites dans le texte (cf plus loin) :

```
textArea.getDocument()  
    .addDocumentListener(new MyDocumentListener());  
  
class MyDocumentListener implements DocumentListener {  
    public void insertUpdate(DocumentEvent evt) { ... }  
    public void removeUpdate(DocumentEvent evt) { ... }  
    public void changedUpdate(DocumentEvent evt) { ... }  
}
```

JFormattedTextField

- Cette classe fille de **JTextField** a été introduite par le SDK 1.4
- Permet de donner un format pour la saisie des données (et aussi éventuellement pour leur affichage)
- De très nombreuses possibilités sont offertes au programmeur ; par exemple, pour dire ce qui se passe si la valeur saisie ne correspond pas au format

Exemple (1)

```
DateFormat f =  
    new SimpleDateFormat("dd/MM/yyyy");  
  
DateFormatter df = new DateFormatter(f);  
  
JFormattedTextField ftf =  
    new JFormattedTextField(df);  
  
// Montrer le format de saisie :  
ftf.setValue(new Date());
```

Exemple (2)

```
try {  
    MaskFormatter mf =  
        new MaskFormatter("(##)## ## ## ##");  
  
    // Pour indiquer les emplacements :  
    mf.setPlaceholderCharacter('_');  
  
    JFormattedTextField ftf =  
        new JFormattedTextField(mf);  
  
} catch (ParseException e) { . . . }
```


JEditorPane

- Comme cette classe peut afficher facilement une page HTML, on peut l'utiliser pour afficher une page d'un serveur HTML ou une aide en ligne

Exemple

```
JEditorPane ep = new JEditorPane();
ep.setEditable(false);

...

URL url = new URL("file:///truc/aide.html");
try {
    ep.setPage(url);
} catch (IOException e) {
    . . .
}
```

Composants texte et threads

- Les méthodes **append()** et **setText()** peuvent être invoquées dans un *thread* indépendant sans risquer de problèmes (cf « Swing et threads »)

Document

- Les modèles pour les composants texte sont de la classe **`javax.swing.text.Document`**
- Représente un container de texte

Pour récupérer le texte

- `int getLength()`
- `String getText(int debut, int longueur)`
- `void getText(int debut, int longueur, Segment txt)`
(le texte récupéré est mis dans **txt**)

Structure d'un document

- Un document a souvent une structure hiérarchique (chapitres, sections,...)
- L'interface **Element** représente un noeud de l'arbre qui représente la hiérarchie
- Pour les cas complexes, plusieurs arbres peuvent être associés à un document
- **Element getDefaultRootElement()** renvoie la racine de l'arbre
- **Element[] getRootElementes()** renvoie les racines des différents arbres



Méthodes de l'interface Element

- **AttributeSet getAttributes()** renvoie les attributs d'un élément de l'arbre ; un attribut est une paire nom-valeur qui peut, par exemple, correspondre à la fonte d'un caractère
- **int getElementCount()** renvoie le nombre de fils
- **Element getElement(int indice)** renvoie le fils numéro indice de l'élément
- **int getStartOffset()** et **int getEndOffset()** renvoient le numéro du début et de la fin de l'élément dans le document



Modifications d'un document

- **void insertString(int debut, String texte, AttributeSet attributs)** insert un texte avec des attributs (éventuellement **null**) ; la modification de la structure qui s'ensuit dépend du document
- **void remove(int debut, int longueur)** supprime les caractères indiqués
- **Position createPosition(int debut)** permet de repérer une position dans le texte indépendamment des ajouts et suppressions (**int getOffset()** de **Position** redonne le numéro de caractère de la position à un moment donné)

Écouteurs

- Les modifications d'un document peuvent être écoutés par des instances de l'interface **DocumentListener** du paquetage **javax.swing.event**
- **{add|remove}DocumentListener(DocumentListener ecouteur)** ajoute et enlève un écouteur
- Si on veut faire une action dans le cas où une zone de texte (**TextArea** par exemple) change, il faut ajouter un écouteur qui fera cette action

DocumentListener

- **void insertUpdate(DocumentEvent evt)** et **void removeUpdate(DocumentEvent evt)**
action à faire en cas d'insertion ou de suppression dans le document
- **void changeUpdate(DocumentEvent evt)**
action à faire en cas de modification d'attributs dans le document

■ ■ ■ Propriétés

- On peut associer des propriétés à un document, par exemple, un titre
- Une propriété est une paire nom-valeur
- **Object getProperty(Object nomProp)** renvoie la valeur de **nomProp**
- **Void putProperty(Object nom, Object valeur)** met la valeur de **nomProp** à **valeur**

■ ■ ■ Actions pour le texte

- Les composants texte viennent avec un lot d'actions
- Ces actions sont de sous-classes de la classe **Action**, plus précisément de la classe **TextAction**
- Quand on exécute la méthode **actionPerformed(e)** de l'action, l'action est effectuée sur le composant retourné par **e.getTextComponent(e)**, ou sur le composant qui a le focus si cette méthode renvoie **null**

■ Actions pour le texte

- Les types d'actions fournies par Swing sont des constantes de la classe **DefaultEditorKit** :
beepAction, **copyAction**, **cutAction**,
pasteAction, **forwardAction**, ... (beaucoup)
- On peut ajouter de nouvelles actions aux actions fournies par Swing à l'aide de la méthode **augmentList()**
- Comme la plupart des classes liées aux textes, la classe **TextAction** est dans le paquetage **javax.swing.text**

■ Keymaps

- Un des rôles importants d'un composant texte est de déclencher une action à la suite de la frappe d'une touche par l'utilisateur (instance de la classe **KeyStroke**)
- Chaque composant a une *keymap* qui relie des touches à des actions
- *Keymap* est une interface de **javax.swing.text**
- Les *keymaps* sont hiérarchisées en arbre



Création d'une keymap

```
Keymap maMap =  
    JTextComponent.addKeymap("maMap", textArea.  
        getKeyMap());  
KeyStroke toucheAvancer =  
    KeyStroke.getKeyStroke(KeyEvent.VK_F,  
        InputEvent.ALT_MASK);  
Action actionAvancer =  
   .getAction(DefaultEditorKit.forwardAction);  
maMap.addActionForKeyStroke(toucheAvancer,  
    actionAvancer);
```

- crée la *map* "maMap" qui ajoute la touche **Alt-F** pour avancer, à la map associée à un *textArea*



Sélection d'une keymap

```
textArea.setKeymap(maMap);
```




Visualisation d'une page HTML

```
public class AfficheHTML {
    public AfficheHTML() {
        try {
            String url = "http://www.google.fr/";
            JEditorPane editorPane = new JEditorPane(url);
            editorPane.setEditable(false);

            JFrame frame = new JFrame();
            frame.getContentPane().add(editorPane,
                BorderLayout.CENTER);
            frame.setSize(200, 300);
            frame.setVisible(true);
        } catch (IOException e) { . . . }
    }
}
```



avec traitement des liens

```
public class Browser extends JPanel {
    public Browser() {
        setLayout (new BorderLayout (5, 5));
        final JEditorPane pWeb =
            new JEditorPane();
        final JTextField input =
            new JTextField("http://google.fr/");
        pWeb.setEditable(false);
    }
}
```

avec traitement des liens

```
// Ecouteur pour clics sur liens
pWeb.addHyperlinkListener(new HyperlinkListener () {
    public void hyperlinkUpdate(final HyperlinkEvent e)
    {
        if (e.getEventType() ==
HyperlinkEvent.EventType.ACTIVATED) {
            // Traité par event dispatch thread
            SwingUtilities.invokeLater(new Runnable() {
                public void run() {
                    try {
                        URL url = e.getURL();
                        pWeb.setPage(url);
                        input.setText(url.toString());
                    } catch (IOException io) { . . . }
                } // fin run
            }); // fin runnable
        }
    } // fin hyperlinkUpdate
}); // fin hyperlinkListener
```

avec traitement des liens

```
JScrollPane pane = new JScrollPane(pWeb);
add(pane, BorderLayout.CENTER);
input.addActionListener (new ActionListener() {
    public void actionPerformed (ActionEvent e) {
        try {
            pWeb.setPage(input.getText());
        } catch (IOException ex) {
            JOptionPane.showMessageDialog (
                Browser.this, "URL Invalide",
                "Saisie invalide",
                JOptionPane.ERROR_MESSAGE);
        }
    }
}); // fin ActionListener
add (input, BorderLayout.SOUTH);
} // fin Browser
```



JTextPane

- La classe JTextPane qui hérite de JEditorPane permet d'afficher des documents complexes qui contiennent des images et différents styles d'écriture



Plan

- Classe JComponent
- Boutons
- Composants pour du texte
- Fenêtres de dialogue
- Menus et barres d'outils
- Barres de progression
- Autres composants et décorations divers
- Classe SwingUtilities



Généralités sur les dialogues

- Une fenêtre de dialogue dépend d'une **JFrame**
- Elle peut être modale (l'utilisateur doit répondre avant de faire autre chose) ou non



Les classes pour les dialogues

- **JOptionPane** est un composant léger, classe fille de **JComponent**
 - elle permet d'avoir très simplement les cas les plus fréquents de fenêtres de dialogue
 - elle affiche une fenêtre modale
- Pour les cas non prévus par **JOptionPane**, on doit utiliser la classe **JDialog** (composant lourd)



Utilisation de JOptionPane

- 4 méthodes **static** de la classe qui font afficher des fenêtres de dialogue modales de divers types :
 - message d'information avec bouton OK (**showMessageDialog**)
 - demande de confirmation avec boutons Oui, Non et Cancel (**showConfirmDialog**)
 - saisie d'une information sous forme de texte, de choix dans une liste ou dans une combobox (**showInputDialog**)
 - fenêtres plus complexes car on peut configurer les composants (**showOptionDialog**)



Signatures des méthodes

- Toutes ces méthodes sont surchargées
- Le message que l'on passe à la méthode est de la classe **Object**
- L'affichage du « message » dépend du type réel du paramètre
 - les cas particuliers sont : **String**, **Icon**, **Component** ou même un tableau d'objets qui sont affichés les uns sous les autres
 - les autres cas sont affichés en utilisant la méthode **toString()**



Look de la fenêtre de dialogue

- Chaque type de fenêtre de dialogue a un aspect différent
- Cet aspect est donné par
 - l'icône placée en haut à gauche de la fenêtre
 - les boutons placés en bas de la fenêtre



Types de messages

- On peut indiquer un type de message qui indiquera l'icône affichée en haut, à gauche de la fenêtre (message d'information par défaut)
- Ce type est spécifié par des constantes :
 - `JOptionPane.INFORMATION_MESSAGE`
 - `JOptionPane.ERROR_MESSAGE`
 - `JOptionPane.WARNING_MESSAGE`
 - `JOptionPane.QUESTION_MESSAGE`
 - `JOptionPane.PLAIN_MESSAGE`

Boutons placés dans la fenêtre

- Ils dépendent des méthodes appelées :
 - **showMessageDialog** : bouton Ok
 - **showInputDialog** : Ok et Cancel
 - **showConfirmDialog** : dépend du paramètre passé ; les différentes possibilités sont
 - **DEFAULT_OPTION**
 - **YES_NO_OPTION**
 - **YES_NO_CANCEL_OPTION**
 - **OK_CANCEL_OPTION**
 - **showOptionDialog** : selon le tableau d'objets passé en paramètre

Valeurs retournées par les méthodes

- **showConfirmDialog** : une des constantes
 - **OK_OPTION**
 - **CANCEL_OPTION**
 - **YES_OPTION**
 - **NO_OPTION**
 - **CLOSED_OPTION**
- **showInputDialog** : le texte (**String**) qu'a choisi ou tapé l'utilisateur
- **showOptionDialog** : le numéro du bouton sur lequel l'utilisateur a cliqué ou **CLOSED_OPTION**

Message d'information

```
JOptionPane.showMessageDialog(frame,  
    "Eggs aren't supposed to be green.");
```

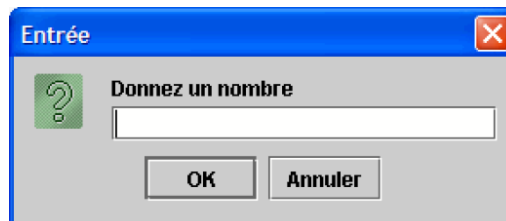


Message d'erreur

```
JOptionPane.showMessageDialog(frame,  
    "Le message d'erreur", "Titre fenêtre  
    dialogue", JOptionPane.ERROR_MESSAGE);
```


Saisie d'une valeur

```
String nombre =JOptionPane.  
showInputDialog("Donnez un nombre");
```



! Utilisable sans env. graphique

Confirmation pour quitter

```
setDefaultCloseOperation(WindowConstants.DO_NOTHING_  
ON_CLOSE);  
  
...  
int reponse = JOptionPane.showConfirmDialog(  
    this,  
    "Voulez-vous vraiment quitter ?",  
    "Quitter l'application",  
    JOptionPane.YES_NO_OPTION);  
  
if (reponse == JOptionPane.YES_OPTION) {  
    System.exit(0);  
}
```



Franciser une fenêtre de dialogue

- Ce problème a été résolu dans les dernières versions du SDK
- Dans les anciennes versions les textes des boutons étaient en anglais
- Un des moyens pour les avoir en français :
UIManager.put("OptionPane.yesButtonText", "Oui");
UIManager.put("OptionPane.noButtonText", "Non");
- Si vous voulez connaître les autres propriétés de la classe **UIManager** associées au *look and feel* :
System.out.println(
UIManager.getLookAndFeelDefaults());



Fenêtres de dialogue complexes

- Pour les fenêtres de dialogue plus complexes ou non modales, il faut hériter de la classe **JDialog**
- Exemple : une fenêtre de dialogue pour un nom d'utilisateur et un mot de passe (les caractères tapés ne sont pas visibles)
- Montre comment initialiser la fenêtre de dialogue, afficher la fenêtre, et finalement récupérer les valeurs saisies par l'utilisateur



Exemple

```
public class SaisieMDP extends JDialog implements
    ActionListener {
    private JPasswordField zoneMDP;
    private JButton valider, annuler;
    private boolean ok;

    public SaisieMDP(JFrame parent) {
        super(parent, "Connexion", true);
        Container contentPane = getContentPane();
        // Panel pour mot de passe
        JPanel p1 = new JPanel();
        p1.setLayout(new GridLayout(2, 2));
        p1.add(new JLabel("Mot de passe :"));
        p1.add(zoneMDP = new JPasswordField(""));
        contentPane.add(p1, BorderLayout.CENTER);
```



Exemple

```
        // Panel pour les boutons
        JPanel p2 = new JPanel();
        valider = new JButton(Valider);
        valider.addActionListener(this);
        p2.add(valider);
        annuler = new JButton(Annuler);
        valider.addActionListener(this);
        p2.add(valider);
        contentPane.add(p2, BorderLayout.SOUTH);
    }
    public char[] getMDP() {
        return zoneMDP.getPassword();
    }
}
```

Exemple

```
public void actionPerformed(ActionEvent evt) {
    if (evt.getSource() == valider)
        ok = true;
    else if (evt.getSource() == annuler)
        ok = false;
    setVisible(false);
}
public boolean afficheDialog() {
    ok = false;
    setVisible(true);
    // modal sur valider ou annuler
    return ok;
}
}
```

Exemple : utilisation

```
// this est une instance d'une sous-classe de JFrame
SaisieMDP smdp = null;
char[] mdp; // le mot de passe
. . .
if (smdp == null)
    smdp = new SaisieMDP(this);
    // afficheDialog() affiche la fenêtre de dialogue.
    // Le programme ne passe à la ligne suivante que
    // lorsque cette fenêtre est effacé par le clic du
    // bouton de validation ou d'annulation de la
    // fenêtre
if (smdp.afficheDialog()) {
    // récupère le mot de passe
    mdp = smdp.getMdp();
}
```



Fenêtres de dialogue complexes

- Dans la pratique, on peut le plus souvent éviter l'utilisation de la classe **JDialog**
- En effet, les méthodes de la classe **JOptionPane** accepte en paramètre toute instance de la classe **Object** et pas seulement une **String**
- On peut ainsi passer des composants graphiques complexes
- ! Exercice : utiliser **JOptionPane** pour la saisie d'un mot de passe



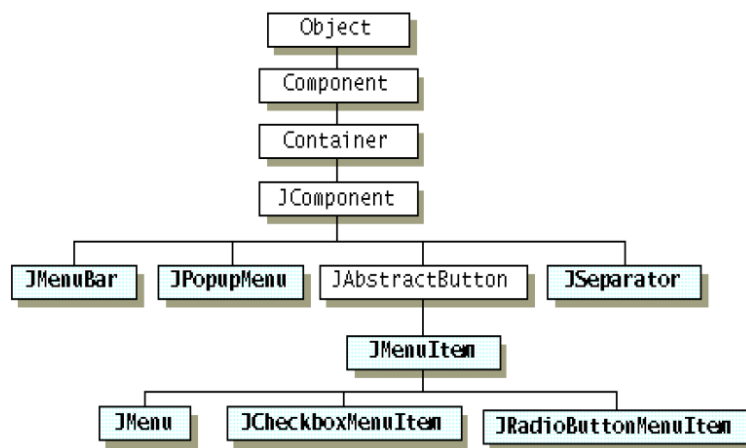
Fenêtres de dialogue particulières

- **JFileChooser** pour choisir un fichier
- 3 types :
 - ☐ pour ouvrir
 - ☐ pour sauvegarder
 - ☐ à personnaliser
- **JColorChooser** pour choisir une couleur
cf démo/td

Plan

- Classe **JComponent**
- Boutons
- Composants pour du texte
- Fenêtres de dialogue
- Menus et barres d'outils
- Barres de progression
- Autres composants et décorations divers
- Classe **SwingUtilities**

Les classes





Exemple de menu

```
JMenuBar menuBar = new JMenuBar();
frame.setJMenuBar(menuBar);

// Un menu de la barre de menu
JMenu menu = new JMenu("Un Menu");
menu.setMnemonic(KeyEvent.VK_M);
menuBar.add(menu);

// Choix des menus
JMenuItem menuItem;
menuItem = new JMenuItem("Texte", KeyEvent.VK_T);
menuItem.setAccelerator(KeyStroke.getKeyStroke(
    KeyEvent.VK_1, ActionEvent.ALT_MASK));
menuItem.setActionCommand("text");
menu.add(menuItem);
```



Exemple de menu

```
menuItem = new JMenuItem("Texte et image",
    new ImageIcon("images/middle.gif"));
menuItem.setMnemonic(KeyEvent.VK_B);
menuItem.setActionCommand("text-image");
menu.add(menuItem);

// Image seulement
menuItem =
    new JMenuItem(new ImageIcon("images/middle.gif"));
menu.add(menuItem);

// Le menu d'aide toujours à droite de la fenêtre
menuBar.add(Box.createHorizontalGlue());
menuBar.add(menuAide);
```

Exemple de menu

```
// Groupe de boutons radio
menu.addSeparator();
ButtonGroup group = new ButtonGroup();
JRadioButtonMenuItem rbMenuItem =
    new JRadioButtonMenuItem("Un choix bouton radio");
rbMenuItem.setSelected(true);
rbMenuItem.setMnemonic(KeyEvent.VK_R);

group.add(rbMenuItem);
menu.add(rbMenuItem);

rbMenuItem = new JRadioButtonMenuItem("Autre choix");
rbMenuItem.setMnemonic(KeyEvent.VK_A);

group.add(rbMenuItem);
menu.add(rbMenuItem);
```

Exemple de menu

```
// Groupe de boîtes à cocher
menu.addSeparator();
JCheckBoxMenuItem cbMenuItem =
    new JCheckBoxMenuItem("boîte à cocher");
cbMenuItem.setMnemonic(KeyEvent.VK_C);
menu.add(cbMenuItem);

cbMenuItem =
    new JCheckBoxMenuItem("Une autre");
cbMenuItem.setMnemonic(KeyEvent.VK_U);
menu.add(cbMenuItem);
```


Exemple de menu

```
// Un sous-menu
menu.addSeparator();

JMenu submenu = new JMenu("Un sous-menu");
submenu.setMnemonic(KeyEvent.VK_S);
menuItem =
    new JMenuItem("Un choix du sous-menu");
menuItem.setAccelerator(KeyStroke.getKeyStroke(
    KeyEvent.VK_2, ActionEvent.ALT_MASK));

submenu.add(menuItem);
```

Récupérer des entrées de menu

- Il faut utiliser les méthodes qui renvoient un **Component** et pas un **JMenuItem** (existent pour compatibilité avec AWT)
- Exemple :
Component getMenuComponent(int n)
- Ces méthodes renvoient aussi les séparateurs et pas seulement des **JMenuItem**



Actions

- Comme avec les boutons en général, les éléments d'un menu peuvent être créés à partir d'une instance de **Action** dès que l'action déclenchée est partagée avec d'autres éléments graphiques



Mnémoniques et accélérateurs

- On peut faciliter l'utilisation des actions associées aux menus de 2 façons :
- un mnémonique permet d'associer un caractère à un choix de menu ; il permet de se déplacer dans l'arborescence des menus avec le clavier
 - Le choix doit être visible au moment de l'utilisation du clavier
 - la combinaison de touches à taper dépend du look & feel (Alt + caractère sous Windows)
 - un accélérateur permet de déclencher une action d'un menu avec le clavier, sans passer par les menus



Attacher des actions à un menu

- Pour chaque choix des menus, on ajoute un écouteur : **menuItem.addActionListener(this);**
- Si un écouteur écoute plusieurs choix, on peut distinguer les choix avec **getActionCommand()**
- Pour les boîtes à cocher, on ajoute un **ItemListener** :
cbMenuItem.addItemListener(this);



Invalider des choix de menus

- Un choix de menu peut être invalidé par **itemMenu.setEnabled(false);**
il apparaîtra alors en grisé
- On peut valider ou invalider un choix d'un menu juste avant l'affichage du menu dans un écouteur de menu (dans la méthode **menuSelected()** de l'interface **MenuListener**, qui est exécutée juste avant l'affichage d'un menu)



Menus *popup*

- Ce sont des menus qui apparaissent quand l'utilisateur fait une certaine action sur le composant dans lequel ils sont définis
- L'action dépend du système sur lequel le programme s'exécute (bouton droit de la souris sous Windows par exemple)
- Si on veut faire afficher le menu *popup* quand l'utilisateur fait cette action sur un composant, on utilise un *MouseListener* du composant



Actions à exécuter

- On les code dans des **ActionListener** associés aux choix du menu popup (comme dans un menu ordinaire)
- Ce code peut utiliser la méthode **getInvoker()** qui renvoie le composant sur lequel le menu popup a été appelé

Code avec menus *popup*

```
// Création du menu
JPopupMenu popup=new JPopupMenu("Titre menu");
menuItem = new JMenuItem("Un choix du menu
    popup");
menuItem.addActionListener(this);
popup.add(menuItem);
menuItem = new JMenuItem("Autre choix");
...
// Ajoute un écouteur qui affiche le menu popup
// aux composants qui pourront afficher ce menu
MouseListener ml = new EcouteurPourPopup();
textArea.addMouseListener(ml);
panel.addMouseListener(ml);
...
```

Code avec menus *popup*

```
class EcouteurPourPopup extends MouseAdapter {
    public void mouseReleased(MouseEvent e) {
        if (e.isPopupTrigger()) {
            popup.show(e.getComponent(),
                e.getX(), e.getY());
        }
    }
}
```

Si l'action avec la souris correspond à la demande d'affichage d'un menu popup (dépendant du système)

Barre d'outils

- Permet de lancer une action en cliquant sur une des petites icônes de la barre
- Une barre d'outils peut être placée horizontalement ou verticalement sur un des bords d'une fenêtre
- Elle peut aussi apparaître dans une fenêtre « flottante »



Barre d'outils

- Utilise un **BoxLayout** pour placer les composants
- Comme les composants d'une barre d'outils déclenchent des actions que l'on peut déclencher d'une autre façon (menus ou boutons), on y ajoute souvent des Actions



Barre d'outils

```
JToolBar toolbar = new JToolBar();

// Ajoute les boutons qui représentent les "outils"
ImageIcon icon = new ImageIcon("image.gif");
JButton button1 = new JButton(icon);
button.addActionListener(actionListener);
toolbar.add(button);
// Idem pour les autres boutons
. . .
. . .
JPanel contentPane = new JPanel();
contentPane.setLayout(new BorderLayout());
...
contentPane.add(toolbar, BorderLayout.NORTH);
contentPane.add(scrollPane, BorderLayout.CENTER);
```



Plan

- Classe **JComponent**
- Boutons
- Composants pour du texte
- Fenêtres de dialogue
- Menus et barres d'outils
- Barres de progression
- Autres composants et décorations divers
- Classe **SwingUtilities**



Les différentes possibilités

- Elles servent à afficher la progression d'une ou plusieurs tâches qui se déroulent en parallèle
- On peut les implanter avec plusieurs classes :
 - **ProgressMonitor** pour afficher la progression d'une seule tâche dans une fenêtre de dialogue
 - **ProgressMonitorInputStream**, sous-classe de **java.io.FilterInputStream**, pour suivre la lecture d'un flot
 - **JProgressBar**, barre de progression élémentaire pour construire une solution dans les cas plus complexes



Utilisation

- La difficulté de l'utilisation de ces barres de progression viennent de ce qu'elles sont faites pour être mises à jour par un autre *thread* que le « *event dispatch thread* »
- On doit donc utiliser les méthodes **invokeLater()** ou **invokeAndWait()**
- On peut aussi les faire mettre à jour par un timer (qui utilise le *event dispatch thread*)



Création d'un **ProgressMonitor**

```
progressM = new
    ProgressMonitor(fenetre, "Charge...", "", 0,
        100);

progressM.setProgress(0);

// S'affiche au bout de 2 secondes par défaut
progressM.setMillisToDecideToPopup(3000);
```



Utilisation du **ProgressMonitor**

- La tâche longue peut modifier les informations affichées par le monitor par les méthodes
 - **setNote(String message)** qui affiche un message (par exemple, le nom du fichier en cours de chargement)
 - **setProgress(int progression)** qui indique la progression de l'opération avec un nombre compris entre le minimum et le maximum donné à la création du monitor (on peut les modifier en cours de route)

Exemple d'un **ProgressMonitor**

```
// Dans la tâche longue à exécuter
SwingUtilities.invokeLater(
    new Runnable() {
        public void run() {
            // pm désigne le progressMonitor
            pm.setNote(...);
            // i représente l'état d'avancement
            pm.setProgress(i);
        }
    });
```

Plan

- Classe **JComponent**
- Boutons
- Composants pour du texte
- Fenêtres de dialogue
- Menus et barres d'outils
- Barres de progression
- Autres composants et décorations divers
- Classe **SwingUtilities**

Types de containers

- Fenêtres internes : fenêtres internes à une fenêtre
- **SplitPane** : permet de diviser une fenêtre en plusieurs sous-parties dont l'utilisateur peut changer les dimensions
- **TabbedPane** : onglets qui permettent à l'utilisateur de choisir facilement (simple clic) parmi plusieurs interfaces

Fenêtres internes

- Permet de travailler sur plusieurs documents en même temps, ou sur des parties différentes d'un même document
- Les fenêtres internes peuvent être manipulées comme des fenêtres externes : déplacées, iconifiées, redimensionnées,...

Fonctionnement des Fenêtres internes

1. On ajoute un **JDesktopPane** à une fenêtre
 2. On ajoute les **JInternalFrame** au **JDesktopPane**
- Remarques :
 - le gestionnaire de placement d'un **JDesktopPane** est **null**
 - pas de **WindowEvent** pour une **JInternalFrame** mais des **InternalFrameEvent**

Exemple de Fenêtres internes

```
desktop = new JDesktopPane();  
setContentPane(desktop);  
  
FenetreInterne fenetre = new FenetreInterne();  
  
fenetre.setVisible(true);  
desktop.add(fenetre);  
  
try {  
    fenetre.setSelected(true);  
} catch (java.beans.PropertyVetoException e) {}
```

Exemple de Fenêtres internes

```
class FenetreInterne extends JInternalFrame {
    static int numFenetre = 0;
    static final int x = 30, y = 30; // haut-gauche
    public FenetreInterne() {
        super("Document #" + (++numFenetre),
            true, // redimensionnable
            true, // la fermer
            true, // « maximiser » avec un clic
            true); //iconifiable
        // Ajouter des composants dans la fenêtre
        . . . // (comme avec JFrame, avec contentPane)
        setLocation(x*numFenetre, y*numFenetre);
        setSize(200, 200); // INDISPENSABLE !!
    }
}
```

SplitPane

- Très utile pour travailler avec beaucoup d'informations diverses dans une fenêtre
- Quand on travaille sur une des informations, on peut occuper toute la place en hauteur ou en largeur en cliquant sur les petits triangles, ou donner la bonne taille à la zone de travail

Exemple de SplitPane

```
splitPane = new
    JSplitPane(JSplitPane.HORIZONTAL_SPLIT,
        paneListe,
        paneImages);

splitPane.setOneTouchExpandable(true);
splitPane.setDividerLocation(150);
minimumSize = new Dimension(100, 50);

paneListe.setMinimumSize(minimumSize);
paneImages.setMinimumSize(minimumSize);
```

Onglets

- Une autre façon que les splitpane pour travailler avec beaucoup d'informations diverses dans une fenêtre
- La différence est qu'on ne voit qu'un seul type d'information à la fois avec les onglets

Exemple d'Onglet

```
int position = JTabbedPane.BOTTOM; // TOP par défaut
JTabbedPane pane = new JTabbedPane(position);
ImageIcon icon = new ImageIcon("image.gif");
pane.addTab("Nom de l'onglet", icon, panel,
    "Texte de la bulle");

// Autre exemple :
ImageIcon icon = new ImageIcon("images/middle.gif");
JTabbedPane tabbedPane = new JTabbedPane();
Component panel1 = makeTextPanel("Blah");
tabbedPane.addTab("Un", icon, panel1, "Fait ...");
tabbedPane.setSelectedIndex(0);
Component panel2 = makeTextPanel("Bla bla");
tabbedPane.addTab("Deux", icon, panel2, ". . .");
Component panel3 = makeTextPanel("Bla bla bla");
tabbedPane.addTab("Trois", icon, panel3, ". . .");
```

ComboBox

- Non modifiable : on choisit dans une liste
- Modifiable : on choisit dans une liste ou on entre un autre choix
- Constructeurs : Sans paramètre, ou avec un paramètre de type
 - **Object[]**
 - **Vector**
 - **ComboBoxModel**

Exemple

```
String[] items = {"item1", "item2"};

JComboBox combo = new JComboBox(items);

combo.addActionListener(actionListener);

// Un JComboBox éditable
combo.setEditable(true);
```

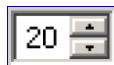
2 types de **ComboBox**

- Non modifiable : on l'écoute par un ActionListener qui reçoit un ActionEvent quand l'utilisateur fait un choix dans la liste
- Modifiable : un ActionEvent sera lancé aussi quand l'utilisateur tape Return dans la zone de texte modifiable

Écouter un **ComboBox**

- Dans la méthode `actionPerformed` de l'écouteur, on utilise le plus souvent la méthode **Object `getSelectedItem()`** qui retourne l'objet sélectionné par l'utilisateur
- Autre méthode : **int `getSelectedIndex()`** retourne le numéro du choix (-1 si le choix n'est pas dans la liste pour un `comboBox` modifiable)

Spinner



- Composant semblable à un `comboBox` : l'utilisateur peut choisir une valeur dans une liste ou taper la valeur de son choix
- La différence est que le choix doit se faire en parcourant une liste, élément par élément ; pas de liste déroulante affichée
- Ainsi, on ne risque pas de cacher d'autres valeurs de l'interface graphique

Spinner

- Implémenté par la classe **Jspinner**
- Repose sur un **SpinnerModel** ; 3 modèles sont disponibles dans l'API :
 - **SpinnerListModel**
 - **SpinnerDateModel**
 - **SpinnerNumberModel**

Exemple

```
String[] joursSemaine =  
    new DateFormatSymbols().getWeekdays();  
SpinnerModel model =  
    new SpinnerListModel(joursSemaine);  
JSpinner spinner = new JSpinner(model);  
String jour = spinner.getValue().toString();  
  
SpinnerDateModel model =  
    new SpinnerDateModel();  
JSpinner spinner = new JSpinner(model);  
Date dateChoisie = model.getDate();
```

JSlider

- Pour changer une valeur en faisant glisser un curseur

```
int min = 0, max = 100, init = 50;
JSlider hSlider =
    new JSlider(JSlider.HORIZONTAL, min, max, init);
hSlider.addChangeListener(new MyChangeListener());

class MyChangeListener implements ChangeListener {
    public void stateChanged(ChangeEvent evt) {
        JSlider slider = (JSlider)evt.getSource();
        if (!slider.getValueIsAdjusting()) {
            int value = slider.getValue();
            process(value);
        }
    }
}
```

Ascenseurs

- Tout composant peut être placé dans un **JScrollPane**, ce qui lui ajoute des barres de défilement (« ascenseurs ») :

```
textArea = new JTextArea(5, 30);
JScrollPane scrollPane = new JScrollPane(textArea);
scrollPane.setPreferredSize(new Dimension(400, 100));
contentPane.add(scrollPane, BorderLayout.CENTER);
```

- Remarque : les barres de défilement n'apparaissent que lorsque le composant ne tient pas dans la zone d'écran qui lui est réservée



Bordures

- On peut ajouter un espace autour d'un composant quelconque ; l'espace peut être vide ou comporter le dessin d'une bordure

```
panel.setBorder(  
    BorderFactory.createEmptyBorder(30, 30, 10,  
    30)  
);
```

- De nombreux types de bordures sont disponibles



Types de bordures

```
JComponent c = new . . .  
c.setBorder(BorderFactory.createEmptyBorder());  
c.setBorder(BorderFactory.createLineBorder(  
Color.black));  
c.setBorder(BorderFactory.createEtchedBorder());  
c.setBorder(BorderFactory.createRaisedBevelBorder());  
c.setBorder(BorderFactory.createLoweredBevelBorder());  
ImageIcon icon = new ImageIcon("image.gif");  
c.setBorder(BorderFactory.createMatteBorder(-1,-1,-1,  
-1, icon));  
c.setBorder(BorderFactory.createTitledBorder("Title"));  
TitledBorder titledBorder =  
BorderFactory.createTitledBorder(border, "Title");  
Border newBorder =  
BorderFactory.createCompoundBorder(border1, border2);  
c.setBorder(newBorder);
```



Bulle d'aide

- On peut ajouter une bulle d'aide ou de description à n'importe quel composant :

`composant.setToolTipText("Texte");`

- Cette bulle s'affiche lorsque le pointeur de la souris est positionné depuis un certain temps sur le composant



Plan

- Classe **JComponent**
- Boutons
- Composants pour du texte
- Fenêtres de dialogue
- Menus et barres d'outils
- Barres de progression
- Autres composants et décorations divers
- Classe **SwingUtilities**



Fonctionnalités

- La classe **SwingUtilities** offre des méthodes utilitaires diverses :
 - inclusion, intersection, différence et réunions de 2 rectangles
 - calcul de la taille d'une chaîne dans une certaine fonte
 - conversions entre les systèmes de coordonnées de 2 composants dont l'un est inclus dans l'autre, ou entre celui d'un composant et celui de l'écran
 - gestion du multitâche (**invokeLater** et **invokeAndWait**)
 - facilités pour reconnaître le bouton de la souris
- Cf doc en ligne



Plan

- Classe **JComponent**
- Boutons
- Composants pour du texte
- Fenêtres de dialogue
- Menus et barres d'outils
- Barres de progression
- Autres composants et décorations divers
- Classe **SwingUtilities**
- Compléments +++



Timer

- Un timer est une sorte de réveil que l'on remonte et qui « sonne » après un certain délai
- Un timer est associé à un (ou plusieurs) **ActionListener** qui reçoit un **ActionEvent** quand le timer « sonne »
- Un timer est représenté par la classe **javax.Timer**
- Un timer peut sonner une seule fois ou à intervalles réguliers



Timer et Threads

- Le timer s'exécute à chacun de ses réveils dans le thread de répartition des événements et il en est donc ainsi des méthodes `actionPerformed()` appelées par le timer
 - ces méthodes peuvent donc manipuler des éléments graphiques
 - mais elles doivent s'exécuter rapidement pour ne pas figer l'interface graphique

■ Arrêt et démarrage d'un timer

- **stop()** stoppe l'envoi des **ActionEvent**
- **start()** reprend l'envoi des **ActionEvent**
- **restart()** reprend tout depuis le début comme si le timer venait d'être démarré pour la 1ère fois

■ Timer : Exemple

```
Timer chaqueSeconde =  
    new Timer(1000, new ActionListener1());  
Timer uneFois =  
    new Timer(5000, new ActionListener2());  
Timer delaiInitial =  
    new Timer(1000, new ActionListener3());  
  
uneFois.setRepeats(false);  
delaiInitial.setInitialDelay(3000);  
  
chaqueSeconde.start();  
uneFois.start();  
delaiInitial.start();
```


java.util.Timer

- La version JDK 1.3 a ajouté les classes **Timer** et **TimerTask** dans le paquetage **java.util** pour lancer une tâche à un moment donné ou à des intervalles répétés
- Au contraire du timer de swing, les tâches ne sont pas nécessairement associées à l'interface graphique et ne sont donc pas déposées dans la file d'attente des événements

JTable

- Représenter des données sous forme tabulaire :

First Name	Last Name	Favorite Color	Favorite Movie	Favorite Number	Favorite Food
Mike	Albers	Green	Brazil	44	
Mark	Andrews	Blue	Curse of the Dem...	3	
Brian	Beck	Black	The Blues Brothers	2,718	
Lara	Bunni	Red	Airplane (the who...	15	
Roger	Brinkley	Blue	The Man Who Kn...	13	

- Les lignes et les colonnes sont identifiées par un nombre entier (en commençant à 0)

Jtable : utilisation

- Un modèle de données qui représente les données dans la **JTable**
- Un modèle de colonnes qui représente les colonnes de la table
- Un (ou plusieurs) *renderer* pour afficher les données du modèle
- Un (ou plusieurs) éditeur de cellule pour saisir les modifications des données par l'utilisateur

Tables simples

- Les tables les plus simples sont construites avec les constructeurs qui prennent en paramètres 2 tableaux d'**Object** ou 2 **Vector** (1 pour les noms des colonnes et 1 pour les données)
- On n'a pas à créer une classe pour le modèle de données
- La table utilise les tableaux ou vecteurs passés en paramètres comme modèles (ils sont donc modifiés si les données de la table le sont)

■ ■ ■ Table simple : Exemple

```
Object[][] data =
    {{ "a11", "a12"}, {"a21", "a22"} };
String[] nomsColonnes = { "Col1", "Col2" };

JTable table =
    new JTable(data, nomsColonnes);
JScrollPane sp = new JScrollPane(table);

// Si on veut définir la taille de la zone
// d'affichage :
table.setPreferredSize(
    new Dimension(500, 70));
```

■ ■ ■ Constructeurs

- **JTable()** : crée une table avec un modèle de données de type **DefaultTableModel** (utilise un **Vector** de **Vector** pour ranger les données) et un modèle de colonnes de type **DefaultTableColumnModel**
- **JTable(int ligne, int colonne)** : des lignes et des colonnes dont les cellules ne contiennent rien (**null**)
- **JTable(Object[][] données, Object[] descriptionColonnes)** : les données sont rangées dans un tableau
- **JTable(Vector données, Vector descriptionColonnes)** : les données sont dans un **Vector** de **Vector**



Constructeurs

- **JTable(TableModel modèleDonnées)** : crée une table avec un modèle de données
- **JTable(TableModel modèleDonnées, TableColumnModel modèleColonnes)** : le modèle pour les colonnes n'est pas nécessairement un **DefaultTableColumnModel**
- **JTable(TableModel modèleDonnées, TableColumnModel modèleColonnes, ListSelectionModel modèleSélection)** : le modèle de sélection n'est pas nécessairement un **DefaultListSelectionModel**



Propriétés des tables simples

- Toutes leurs cellules sont éditables
- Tous les types de données sont affichés sous forme de chaînes de caractères
- Toutes les données sont dans un tableau ou un **Vector**, ce qui ne convient pas si, par exemple, les données viennent d'une base de données