01 cours Java Swing DB

Book · July 2014

CITATIONS
0

READS
2,272

1 author:

Djamel Berrabah
University of Sidi-Bel-Abbes
19 PUBLICATIONS **25** CITATIONS

SEE PROFILE

Université Djillali Liabès –Sidi Bel Abbès
Faculté des sciences exactes
Département d'Informatique

Développement d'Interfaces Graphiques
Java Swing

3ème année licence
SITW

Réalisé par : Dr. Djamel BERRABAH

01 juillet , 2014

Préface

Ce support de cours est destiné aux étudiants ayant déjà programmé en langage Java. Il faut qu'ils connaissent les aspects un peu avancés de ce langage comme par exemple, les classes et les interfaces. Ce cours est donné aux étudiants de la troisième année licence SITW, au sein du département d'Informatique, faculté des sciences exactes à l'université de Sidi Bel Abbès.

Le but de ce cours est de permettre aux étudiants de créer leurs propres interfaces graphiques appelées aussi IHM (pour Interfaces Homme Machine) ou encore GUI (pour Graphical User Interfaces). Il s'agit de la programmation événementielle. C'est-à-dire que son programme tourne en boucle et attend les actions de l'utilisateur sur des composants graphiques (menus, boutons, cases à cocher, etc). Le langage Java propose un nombre de bibliothèques dédiées à la création d'interfaces graphiques, mais dans cet ouvrage, nous utiliserons principalement le package javax.swing.

Ce support de cours comporte six chapitres. Le premier étant une introduction générale sur la programmation événementielle. Le deuxième chapitre présente les différents types de conteneurs et comment ces derniers comprennent les objets de l'IHM. Dans le chapitre trois, nous aborderons les composants SWING. Ces derniers occupent des positions dans les conteneurs. Le chapitre quatre nous montre comment gérer le placement des composants dans les conteneurs. Dans le cinquième chapitre, nous verrons comment les événements émis par les éléments de l'interface graphiques sont gérés. Finalement, le dernier chapitre comporte les codes des exemples des interfaces présentes dans les différents chapitres de ce manuscrit. Une autres série d'exercices avec solutions est données dans ce chapitre.

Table des matières

Chapitre 1

1 Introduction générale

1.1 Une interface graphique

Une interface graphique **IG** (appelée aussi Interface Homme Machine **IHM** ou encore Graphical User Interface **GUI**) est un dispositif de dialogue entre l'homme (l'utilisateur) et la machine (une application). Les objets à manipuler, dans ce dispositif, sont présentés à l'utilisateur sous la forme d'icones dans une fenêtre à l'écran de la machine. Cette présentation permet à l'utilisateur d'effectuer différentes opérations à l'aide d'un dispositif de pointage, le plus souvent une souris, sans avoir à connaître les commandes codées pour effectuer ces opérations.

En 1981, la compagnie Xerox propose pour la première fois son ordinateur **Star 8010** dotée de la technologie à interface graphique et une souri. Deux ans après (en 1983), la compagnie Apple développe son premier ordinateur baptisé "***le***

Lisa" doté lui aussi d'une interface graphique et une souris. De nos jours, la plupart des appareils utilisés par l'homme sont dotés d'interfaces graphiques et un dispositif de pointage comme par exemple les écrans tactiles. Ces interfaces sont développées en utilisant un langage de programmation, notamment le langage Java.

1.2 Le langage Java

Le langage Java a été introduit par la société SUN en 1995. C'est un langage évolué orienté objet. Il est basé en grande partie sur la syntaxe du langage C. il est dit langage interprété, c'est-à-dire, que le code source est compilé en pseudo code ou bytecode puis exécuté par un interpréteur Java : la Java Virtual Machine (JVM). En effet, le bytecode, s'il ne contient pas de code spécifique à une plate-forme particulière, peut être exécuté et donner quasiment les mêmes résultats sur toutes les machines disposant d'une JVM.

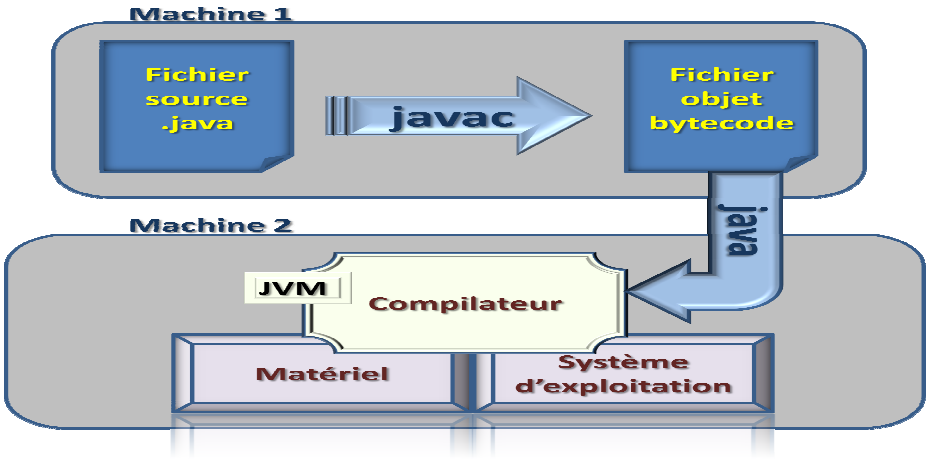


Figure 1 - La machine virtuelle java JVM

Le langage Java est cependant largement utilisé pour le développement d'applications d'entreprises et mobiles. Il propose des APIs pour le développement d'interfaces graphiques permettant aux utilisateurs d'interagir

avec ces applications. Dans un premier temps, Java proposait l'API AWT (Abstract Window Toolkit, JDK 1.1). Ensuite, il propose une nouvelle API nommée Swing (JDK 1.2). AWT et Swing font partie du Framework JFC (Java Foundation Classes) qui offre des facilités pour construire des interfaces graphiques. A tout moment, l'utilisateur peut interférer avec plusieurs composants (bouton, menu, liste déroulante, etc.) de l'interface graphique. Ces actions peuvent modifier totalement le cheminement du programme. Par conséquent, il n'est pas possible de prévoir un ordre d'exécution des instructions à l'écriture du code.

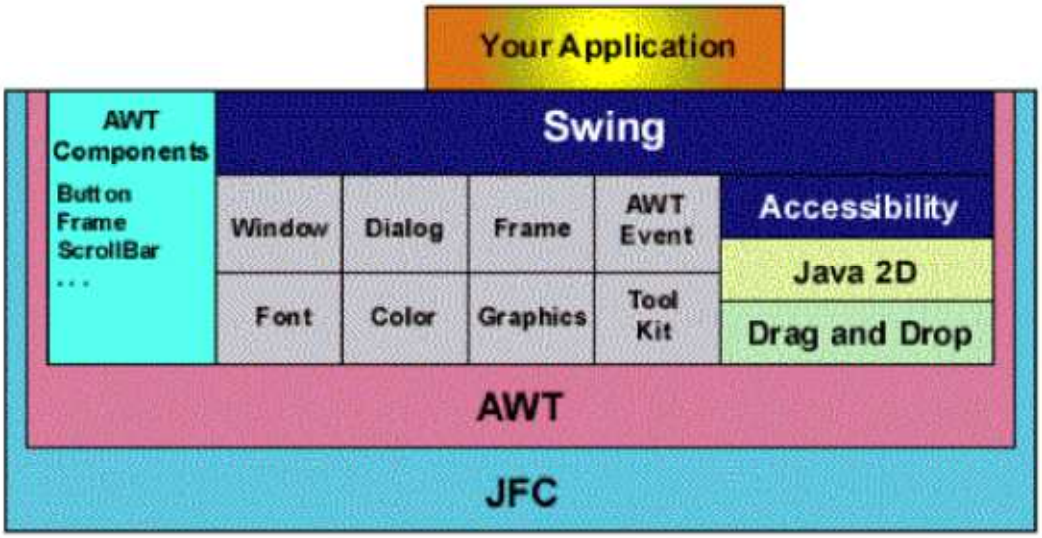


Figure 2 - La boite à outils JFC

1.3 AWT

La librairie AWT (Abstract Window Toolkit) a été introduite dès les premières versions de Java. Comme son nom l'indique, AWT est une abstraction. Elle a été conçue pour qu'elle soit portable elle aussi, d'ailleurs c'est le principe fondamental du langage Java. Ses fonctionnalités sont les mêmes pour toutes les implémentations Java. Les classes d'AWT permettent de développer des interfaces graphiques indépendantes du système d'exploitation sur lesquels elles vont fonctionner. Par ailleurs, cette librairie utilise les ressources propres au

système graphique de la plateforme d'exécution (Windows, MacOS, X-Window) encapsulées dans des abstractions. L'affichage des composants est géré par le système ainsi leurs apparences sont différentes selon les plateformes.

La librairie AWT contient deux classes principales : Component et Container. Tout objet de l'interface graphique doit être une classe dérivée de la classe Component. La classe Container hérite, elle aussi, de la classe Component. Elle sert à contenir les objets de l'interface graphique. Ces objets que nous appelons aussi composants, sont définis à partir de simples classes qui héritent elles aussi de la classe Component (Figure 3). Chaque classe sert à créer l'objet graphique correspondant. Par exemple, la classe `Frame` est utilisée pour créer un cadre de fenêtre et la classe `TextField` c'est pour les zones de saisie de texte.

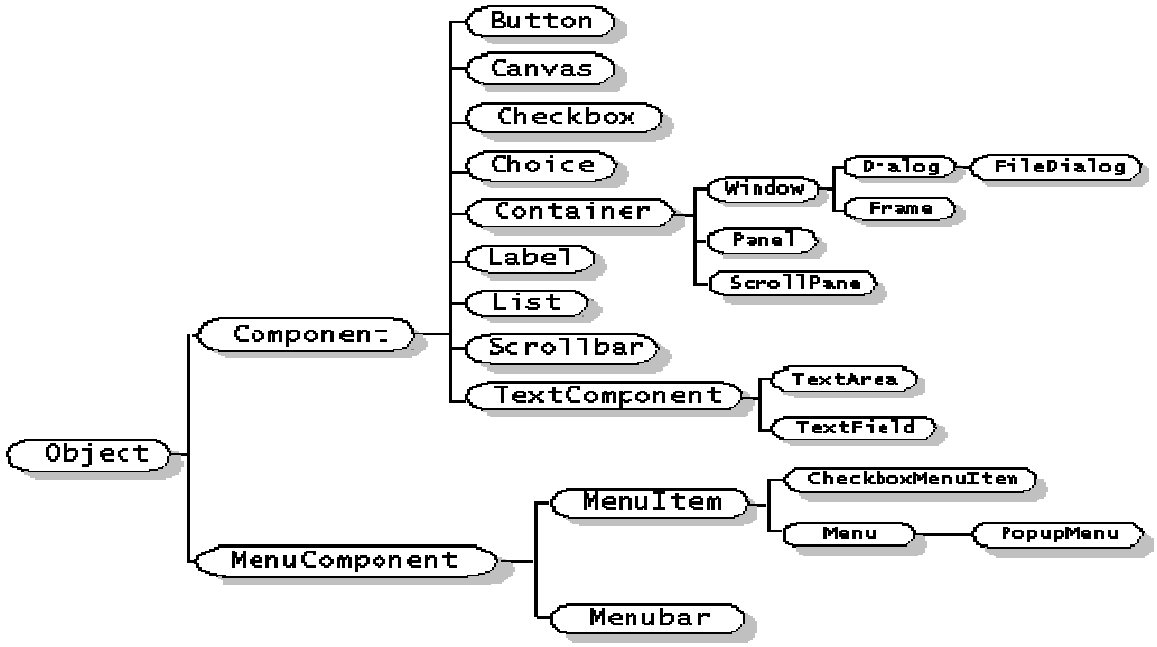


Figure 3 - hiérarchie des classes de la librairie AWT

1.4 Swing

AWT utilise la boîte à outils de la plateforme sur laquelle est crée l'interface graphique. Par conséquent, l'apparence de cette dernière peut être différente d'une plateforme (système d'exploitation) à l'autre. Pour cette raison,

les concepteurs de Java ont pensé à créer une nouvelle API, appelée Swing, moins dépendante de la plateforme sur laquelle est créée l'interface. La solution était d'utiliser une fenêtre de cette plateforme qui soit totalement blanche. Sur cette fenêtre, le programmeur peut peindre entièrement ce qu'il désire. De cette façon, les objets graphiques ne change pas d'apparence quelque soit le système d'exploitation.

Swing fait partie de la bibliothèque Java Foundation Classes (JFC). C'est une API dont le but est similaire à celui de l'API AWT seulement les modes de leur fonctionnement et leur utilisation sont totalement différents. Swing utilise quelques éléments d'AWT mais propose en même temps un grand nombre de nouveaux composants. Tous les composants Swing ont leur équivalent dans AWT. Seulement les composants Swing sont plus légers et se distinguent par la lettre J: JButton dans Swing correspond à Button dans AWT. Swing peut être considéré comme étant une amélioration de la librairie AWT car il offre de nombreux composants qui n'existent pas dans cette librairie et certains possèdent des fonctions étendues. Il propose également une utilisation des mécanismes de gestion d'événements performants et une apparence modifiable à la volée ; une interface graphique qui emploie le style du système d'exploitation Windows ou Motif ou un nouveau style spécifique à Java nommé Metal. Toutes les classes de Swing héritent des classes d'AWT. Par ailleurs, Swing ne peut pas remplacer AWT puisqu'il est basé dessus. En outre, certains éléments d'AWT comme la classe `Color` et la gestion des événements, ainsi que quelques autres éléments ne sont pas définis dans Swing.

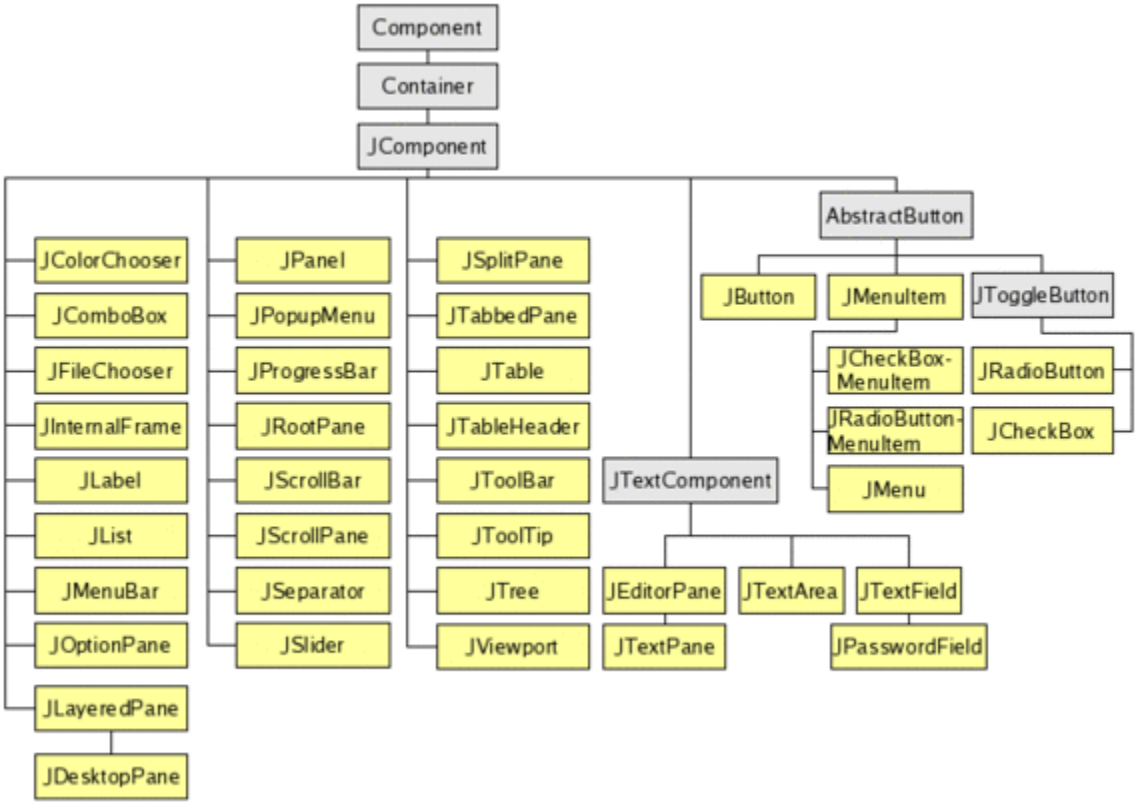


Figure 4 - hiérarchie des classes de la librairie Swing

Chapitre 2

2 Les conteneurs

Les conteneurs sont des objets graphiques pouvant inclure d'autres objets graphiques. Une interface graphique doit avoir un conteneur principal (appelé aussi conteneur de haut niveau) comme composant racine. Ce conteneur sert à inclure tous les autres composants de l'interface. Ces derniers peuvent être des conteneurs. Dans une interface graphique, on imbrique généralement trois niveaux d'objets graphiques : les conteneurs de haut niveau, les composants-conteneurs intermédiaires et les composants atomiques.

2.1 Les conteneurs de haut niveau

Chaque interface graphique est dotée d'un conteneur principal permettant d'encapsuler tous ses objets. Dans le cas d'une fenêtre sans bordure, on utilise la classe JWindow. S'il s'agit d'une application autonome, c'est la classe JFrame qu'il faut utiliser. La classe JDialog sert à créer une boite de dialogue. Enfin, pour

créer une applette, on utilise la classe JApplet. La Figure 5 montre la hiérarchie de ces classes.

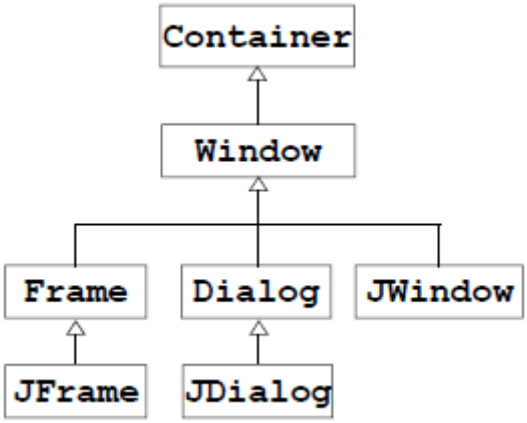


Figure 5 Hiérarchies des classes conteneurs hot niveau

2.1.1 La JFrame

C'est une fenêtre de haut niveau, avec une bordure et un titre. Une `JFrame` (cadre) possède les contrôles de fenêtre standard, tels un menu système, des boutons pour réduire, agrandir et fermer la fenêtre. Elle peut aussi contenir une barre de menus. La Figure 6 montre un exemple d'une `JFrame` vide.

```
import javax.swing.JFrame;

public class JFrame_cont {
  public static void main(String[] args) {
    JFrame f = new JFrame();
    f.setSize(200, 100);
    f.setVisible(true);
  }
}
```

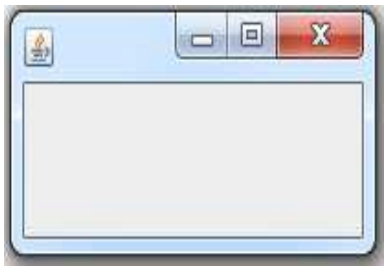


Figure 6 Une JFrame vide

Bien que ce conteneur soit utilisé pour encapsuler les objets de l'interface, il fortement déconseillé de lui ajouter ces objets directement (c-à-d, à ne pas faire appel à la méthode `add()`). En effet, pour gérer au mieux les objets de l'interface, l'unique fils de la `JFrame` est le `JRootPane`. C'est ce dernier qui

contient réellement les objets. L'exemple suivant nous montre comment ajouter les objets d'une interface graphique (un bouton ici) à une `JFrame`.

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

public class JFrame_cont {
  public static void main(String[] args) {
    JFrame f = new JFrame();
    Container c = f.getContentPane();
    c.add(new JButton("bouton"));
    f.setSize(200, 100);
    f.setVisible(true);
  }
}
```

Figure 7 - Une JFrame contenant un bouton

Pour configurer l'action à exécuter lors de la fermeture de la fenêtre, on utilise la méthode `setDefaultCloseOperation(int operation)`. En effet, l'objet instanciant la `JFrame` n'est pas automatiquement détruit quand on clique sur le bouton de fermeture de la fenêtre. Les différentes valeurs pour `operation` lors de la fermeture de la fenêtre sont :

- `DISPOSE_ON_CLOSE` : cette option provoque l'arrêt de l'application lors de la fermeture de la dernière fenêtre prise en charge par la machine virtuelle.
- `DO_NOTHING_ON_CLOSE` : rien ne se passe. Dans ce cas, il est du rôle de l'utilisateur de définir des événements (voir chapitre 5) pour gérer l'arrêt de l'application.
- `EXIT_ON_CLOSE` : l'application s'arrête même si d'autres fenêtres sont encore visibles.
- `HIDE_ON_CLOSE` : la fenêtre est masquée par un appel à sa méthode `setVisible(false)`.

2.1.2 Le JDialog

Les fenêtres de dialogue sont utilisées quand on veut demander à l'utilisateur de saisir une information, afficher une erreur, lui demander une confirmation, etc.

Les classes de base pour travailler avec des fenêtres de dialogue sont les classes `JDialog` et `JOptionPane`. La classe `JOptionPane` permet de créer des fenêtres de dialogue de modèles prédéfinis. On n'utilise alors directement la classe `JDialog` si on veut utiliser une fenêtre de dialogue personnalisée. Cette fenêtre est généralement temporaire et modale ou non. Elle peut bloquer l'interaction avec l'application tant qu'elle est ouverte et impose à l'utilisateur de répondre pour débloquer la situation. C'est la méthode `setModal(boolean val)` qui permet de décider si la fenêtre de dialogue est modale ou non en fonction de la valeur de `val` : `true` ou `false`.

L'exemple suivant montre comment créer une fenêtre `JDialog`

```
import javax.swing.*;
public class JDialog_cont {
     public static void main(String[] args){
            JDialog dialogue = new JDialog();//Pour créer une fenêtre JDialog
            dialogue.setSize(200, 100);//Pour lui donner une taille
            dialogue.setTitle("Fenêtre JDialog"); //Pour lui donner un titre
            dialogue.setVisible(true);//Pour la rendre visible

     }
}
```

A la différence de la `JFrame`, la fermeture d'une `JDialog` se fait par la méthode `dispose()`. C'est à dire, quand on ferme la fenêtre de dialogue depuis l'icône de fermeture (x), le processus ne s'arrête pas.

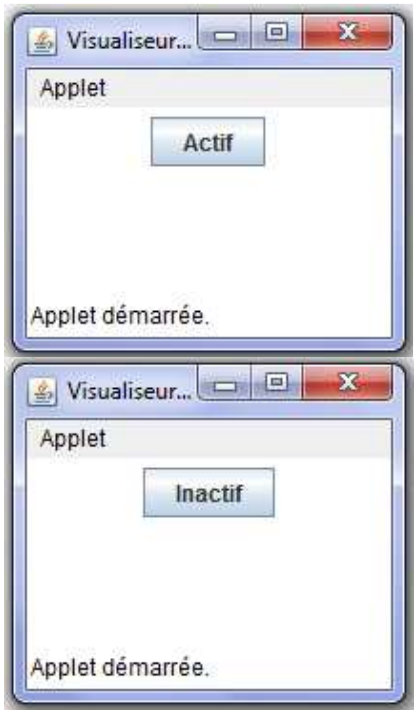
2.1.3 La JApplet

Elle est utilisée pour construire un programme devant être incorporé dans une page HTML et exécuté dans un navigateur HTML ou dans un visualiseur d'applet. `JApplet` étant une sous-classe de `Panel`, elle peut contenir des composants, mais elle ne possède ni bordure, ni titre. L'exemple suivant est une l'applet contenant un bouton étiqueté Actif. Quand on clique sur le bouton, il serait étiqueté inactif.

```
import java.awt.*;
import java.awt.event.*;
import java.applet.Applet;

import javax.swing.JButton;

public class AppletExemple extends Applet
implements ActionListener {
      JButton btn;
      public void init() {
            btn = new JButton("Actif");
            btn.addActionListener(this);
            add(btn);
      }
      public void actionPerformed(ActionEvent
e) {
            if (btn.getLabel().equals("Actif"))
                  btn.setLabel("Inactif");
            else
                  btn.setLabel("Actif");
      }
}
```



2.1.4 La JWindow

`JWindow` n'est rien d'autre qu'un plein écran d'affichage graphique. C'est la fenêtre la plus basique. Il s'agit d'un conteneur que nous pouvons afficher sur notre écran. `JWindow` n'a pas de barre de titre, pas de boutons de fermeture/redimensionnement et n'est pas redimensionnable par défaut. Vous pouvez bien sûr lui ajouter toutes ces fonctionnalités. On utilise surtout les `JWindow` pour faire des `SplashScreen`, c'est-à-dire des interfaces d'attente qui se ferment automatiquement.

Exemple

```
import javax.swing.JFrame;
import javax.swing.JWindow;

public class JWindow_cont {
  public static void main(String[] args) {
    JWindow w = new JWindow();
    w.setSize(300, 300);
    w.setLocation(500, 100);
    w.setVisible(true);

  }
}
```

2.2 Les conteneurs de niveaux intermédiaires

Les conteneurs intermédiaires sont utilisés pour structurer l'application graphique. Ils sont contenus dans un composant top-level. Un conteneur intermédiaire peut contenir d'autres conteneurs intermédiaires. Swing propose une variété de conteneurs intermédiaires.

2.2.1 Le JPanel

C'est le conteneur intermédiaire le plus neutre. La seule manipulation que l'on peut effectuer dans ce conteneur est de choisir la couleur de fond. Les composants sont placés dans ce conteneur les uns après les autres de gauche à droite. En effet, ce conteneur utilise par défaut le gestionnaire de placement `FlowLayout` (Chapitre 4). Il est possible de lui attribuer un autre gestionnaire de position en utilisant la méthode `setLayout()`.

Pour créer un `JPanel`, on utilise le constructeur `JPanel()`. Et pour ajouter des composants dans le `JPanel`, on utilise la méthode `add()`.

Exemple

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

public class JPanel_cont {
  public static void main(String[] args) {
    JFrame f = new JFrame("JPanel contenant
                      un bouton et un label");
    JPanel jp = new JPanel();
    Container c = f.getContentPane();
    c.add(jp);
    jp.add(new JButton("bouton"));
    jp.add(new JLabel("Label"));
    f.setSize(200, 100);
    f.setVisible(true);
  }
}
```

Figure 8 Un JPanel contenant un bouton suivi d'un label

La Figure 8 montre un exemple de composants (un bouton et un label) contenus dans un `JPanel`, lui-même contenu dans une `JFrame`. En fait, un `JPanel` ne peut pas être utilisé de façon isolée comme les conteneurs de haut niveau (fenêtres, frames, …etc).

2.2.2 Le JScrollPane

C'est un conteneur général. Il ne peut pas être utilisé de façon autonome. Il permet de manipuler une zone de composant (texte, image, etc) plus large en utilisant des barres de défilement (Figure 9). Ces barres de défilement sont apparues automatiquement dès que la taille du composant est plus grande que le scrollpane. Donc, rien à coder pour que ces barres apparaissent.

Remarque

A la différence des autres conteneurs, le `JScrollPane` n'utilise pas la méthode `add()` pour ajouter des composants. L'exemple du code java suivant montre comment ajouter un composant à ce conteneur. Il s'agit d'une liste de

noms présents dans un `JList`. La variable c représente le contentpane du conteneur de haut niveau auquel on ajoute le scrollpane.

Figure 9 Un scrollpane contenant une grande image

```
JScrollPane scrollpane;
String categories[] = { "Ahmed", "Aboubakr", "Meriem",
        "Kamel", "Aicha", "Badr", "Hassine",
        "Rahma", "Faouzi", "Faiza", "Tasnim",
        "Alia" };
JList list = new JList(categories);
scrollpane = new JScrollPane(list);
c.add(scrollpane, BorderLayout.CENTER);
```

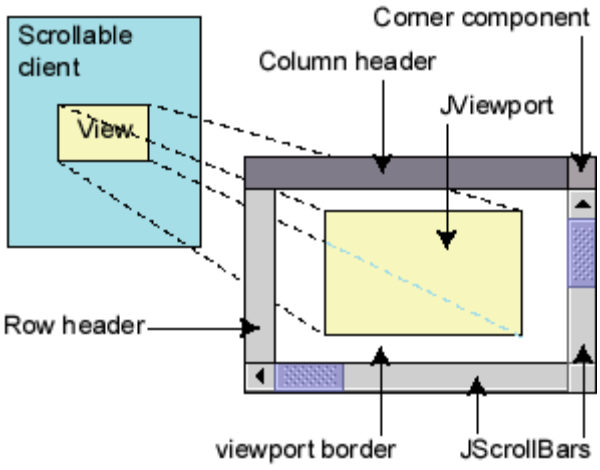


Figure 10 Structure d'un JScrollPane

En effet, un `JScollPane` contient deux composants principaux `JScrollBars` et `JViewport` (Figure 10). Ce dernier propose une fenêtre servant à contenir le composant à présenter. Si ce dernier n'est pas utilisé comme paramètre du

JScrollPane, alors il peut être ajouté en utilisant la méthode setViewportView() ou encore scrollPane.getViewPort().add().

2.2.3 Le JSplitPane

C'est un panneau à deux compartiments. La séparation de ces derniers, qui peut être horizontale ou verticale, se fait par une barre. Chaque compartiment est ajustable. Une seule classe de look-and-feel est nécessaire pour les deux compartiments. Un JSplitPane accueille deux composants.

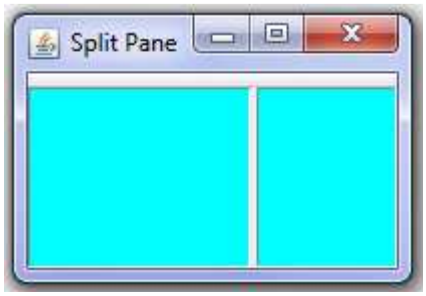


Figure 11 Un Split Pane avec deux compartiments vides

Plusieurs constructeurs peuvent être utilisés pour créer un JSplitPane :

```
JSplitPane(int orientation, Component comp1, Component comp2);
JSplitPane(int orientation)
JSplitPane()  //horizontal par défaut
```

Le paramètre orientation permet de déterminer si la séparation est horizontale ou verticale. Les deux paramètres comp1 et comp2 sont les composants à ajouter respectivement à gauche et à droite (dans le cas d'une séparation verticale) ou en haut et en bas (dans le cas d'une séparation horizontale). Quand aucun paramètre n'est spécifié, l'orientation est horizontale par défaut et aucun composant n'est ajouté.

2.3 Les Conteneurs Intermédiaires Spécialisés

Tout conteneur de haut niveau contient un conteneur intermédiaire spécialisé. C'est le conteneur intermédiaire principal. Il contient typiquement un ou plusieurs panneaux. Les conteneurs intermédiaires spécialisés sont des conteneurs qui offrent des propriétés particulières aux composants qu'ils accueillent. En effet, ils sont destinés à des utilisations plus particulières.

2.3.1 Le JRootPane

Pour bien comprendre les conteneurs de haut niveau et en particulier le `JWindow`, le `JFrame` et `JDialog` ou encore `JInternalFrame`, il est essentiel d'avoir une compréhension du `JRootPane`. En effet, ces conteneurs contiennent un objet appelé un contentpane (panneau de contenu), qui est le vrai référentiel pour les composants qui sont ajoutés au conteneur. Ce volet de contenu est un objet de la classe `JRootPane`.

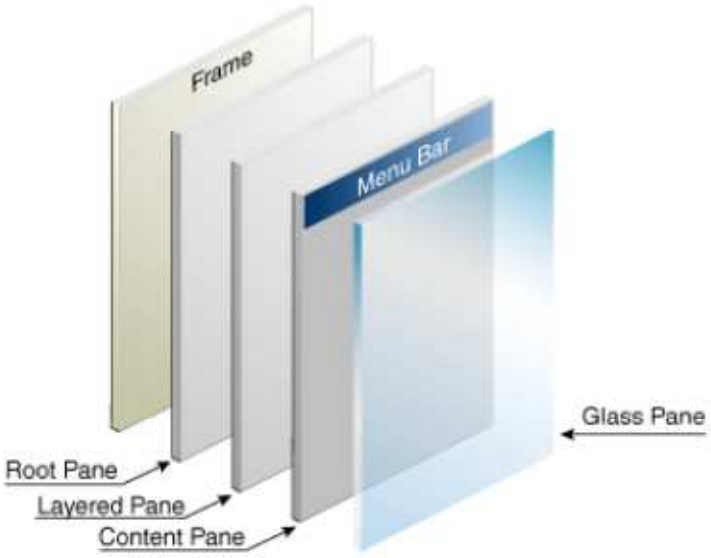


Figure 12 Vue d'ensemble d'un JRootPane

Java utilise cette structure en couches pour animer certains composants comme les barres de menus. Le `JRootPane` comporte quatre parties : un panneau

de verre (glass pane), un panneau de contenu (content pane), un premier panneau en couche (layered pane), et une barre de menu (Menu Bar) qui est optionnelle.

2.3.2 Le JLayeredPane

Un `JLayeredPane` (ou les panneaux superposés) permet de positionner les composants par couches. Chaque couche est codée par un entier. Ces couches sont transparentes, ordonnées suivant leur profondeur. Un composant situé sur une couche donnée masquera les composants des couches les plus profondes qu'il recouvre. La profondeur d'une couche est représentée par un objet `Integer`. Six profondeurs sont prédéfinies :

- `FRAME_CONTENT_LAYER (-30000)` : le contentPane est de ce niveau
- `DEFAULT_LAYER (0)` : c'est la couche (niveau) "par défaut"
- `PALETTE_LAYER (100)` : cette couche se trouve sur la couche par défaut. Elle est utile pour les barres d'outils et les palettes flottantes, afin qu'elles puissent être déplacés au-dessus d'autres composants.
- `MODAL_LAYER (200)` : cette couche est utilisée pour les dialogues modaux. Elle apparait en haut des barres d'outils, des palettes, ou des composants standards dans le conteneur
- `POPUP_LAYER (300)` : cette couche est utilisée pour les menus glissants, les tooltips. Elle apparait au dessus des boites de dialogue. De cette façon, les fenêtres contextuelles associées à des combos, des info-bulles, ou tout autre texte d'aide apparaissent au dessus du composant, de la palette ou le dialogue qui les a générés.
- `DRAG_LAYER (400)` : pour glisser/déposer les composants d'une couche vers une autre.

L'affichage est évidemment dans l'ordre croissant.

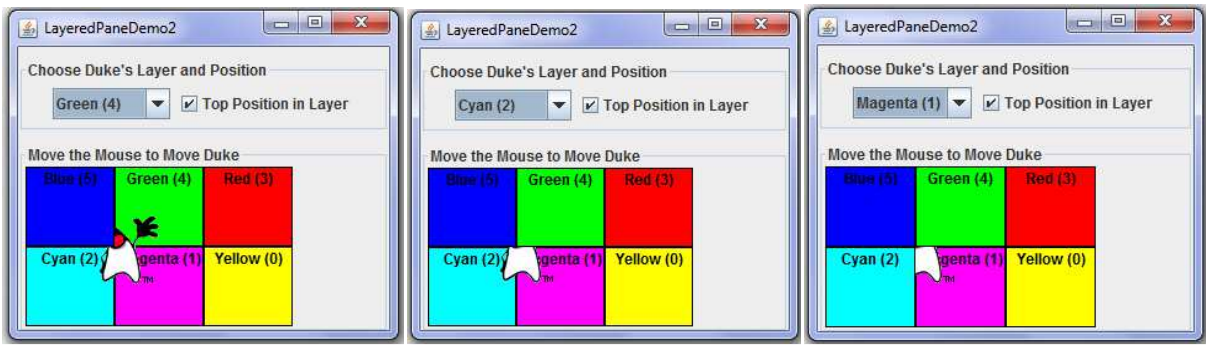


Figure 13 Les couches d'un JLayeredPane et leurs pronfondeurs

L'exemple de la Figure 13 (pris de la doc oracle) représente six rectangle avec des couleurs ; trois aux dessus et trois en dessous. Le duke Wave Red peut être déplacé par la souri. Son apparence dépond de son niveau de profondeur de couche. Ainsi, il s'affiche au dessus ou en dessous des rectangle coloriés. Le niveau de profondeur est déterminé par une valeur choisie dans la liste déroulante. Par exemple, quand on choisit la valeur `Cyan`(2), le Duke s'affiche au dessus du rectangle de cette couleur mais en dessous du rectangle de la couleur verte associé à la valeur `Green`(4). Quand on choisit cette dernière, le Duke s'affiche au dessus du rectangle en question.

`JLayeredPane` dispose d'un certain nombre de méthodes. `moveToFront` (Component), `moveToBack`(composant) et `setPosition()` peuvent être utilisés pour repositionner un composant dans sa couche. La méthode `SetLayer`() peut également être utilisé pour modifier la couche actuelle du composant. Il est essentiel de savoir qu'il n'y a pas de gestionnaire de disposition par défaut pour déterminer la taille des composants contenus dans le `JLayeredPane`.

2.3.3 La JInternalFrame

Une `JInternaleFrame` permet de gérer plusieurs fenêtres dans une seule fenêtre comme dans un desktop sur un ordinateur. Ces fenêtres peuvent être déplacées, redimensionnées, fermées, etc. Une `JInternaleFrame` peut

ressembler à une JFrame sauf qu'elle n'est pas un conteneur de haut niveau. C'est-à-dire qu'elle affiche une barre de titre et des poignées de modification de taille, mais n'est pas une fenêtre indépendante. Par ailleurs, ce conteneur dispose d'un sous-conteneur de type JRootPane comme unique enfant. L'ajout d'un menu et/ou de composants dans un tel conteneur s'effectue donc de la même manière que pour un conteneur de haut niveau, c'est-à-dire en utilisant le ContentPane.

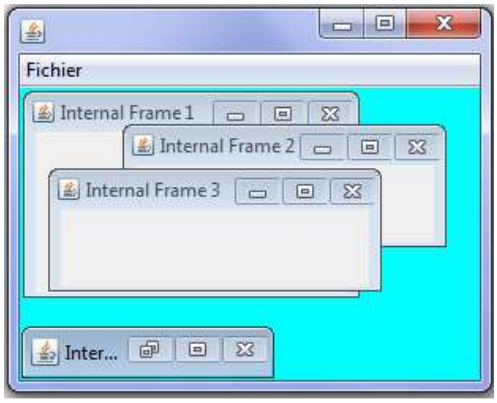


Figure 14 une interface graphique avec des internal frames

Remarque

Il faut noter que les internalframes ne sont pas ajoutés directement au conteneur de haut niveau (JFrame dans notre exemple). En effet, il faut les mettre dans un JDesktopPane et ajouter ce dernier au conteneur de haut niveau.

```
JFrame f = new JFrame(); // définition du conteneur de haut niveau
...
JDesktopPane desktop = new JDesktopPane(); // definition du JDesktopPane qui
...                                        // gérerait les JInternalFrames
...
f.add(desktop, BorderLayout.CENTER);  // l'ajout du desktop au JFrame
...
JInternalFrame jif = new JInternalFrame(); // creation d'un JInternalFrame
...
desktop.add(jif); // l'ajout du JInternalFrame au desktop
```

Chapitre 3

3 Les composants Swing

3.1 Introduction

Comme nous l'avons introduit dans le premier chapitre, nous aborderons dans cet ouvrage le développement d'interfaces graphiques et, par extension, la programmation événementielle. Les entrées/sorties ne seront, pratiquement, plus des saisies au clavier mais plutôt via des événements provenant de composants graphiques : des boutons, des listes, etc. Nous allons voir la librairie Swing qui fonctionne en collaboration avec la librairie AWT. Les composants Swing forment une nouvelle hiérarchie parallèle à celle d'AWT. Nous n'utiliserons pas de composants AWT ayant un équivalent dans Swing. En revanche, nous ferrons éventuellement appel à des objets issus du package AWT (n'ayant pas d'équivalent dans Swing) pour interagir et communiquer avec les composants Swing.

Nous découvrons, dans ce chapitre, les composants de base Swing et comment les intégrer à une interface graphique. L'ancêtre de la hiérarchie Swing est le composant `JComponent`. Ils ne possèdent aucune partie native sauf ceux qui assurent l'interface avec le système d'exploitation : `JApplet`, `JDialog`, `JFrame`, et `JWindow`.

Figure 15 - Exemple d'une interface graphique

L'interface de la Figure 15 a été crée par le code suivant :

```
import javax.swing.*;
public class Hello {
     public static void main(String[] args) {
         JOptionPane.showMessageDialog(null, "Hello World");
         System.exit(0);
     }
}
```

Un composant Swing possède les caractéristiques suivantes :

- C'est un beans
- Ses bords, sa taille et sa couleur peuvent être changés
- Peut être activé ou désactivé
- Peut être visible ou non
- Peut être personnalisé en apparence et en comportement

On distingue trois types de composant de base ; les composants de présentation, de choix et de saisie.

3.2 Les composants de présentation

3.2.1 Le label

Un label ou une étiquette est l'élément le plus fondamental dans la bibliothèque Swing. Il sert à afficher un texte et/ou une icône (image) non modifiable par l'utilisateur. Il est très utile et pratiquement toujours présent dans une interface graphique. Il peut servir à étiqueter d'autres composants comme les `JComboBox` et les `JList`. Dans ce cas, il faut associer l'étiquette au composant pour pouvoir définir un raccourci clavier pour accéder à ce composant. Le texte du label est accessible en lecture par la méthode `getText()` et en écriture par la méthode `setText()`. La Figure 16 montre à quoi ressemble une étiquette.

Figure 16 - Exemple d'une étiquette

La classe permettant de créer une étiquette est `JLabel`. Différents constructeurs existe pour cette classe. Nous pouvons en citer par exemple :

- **`JLabel()`** Création d'une étiquette sans texte ni image
- **`JLabel(String)`** Création d'une étiquette avec le texte uniquement
- **`JLabel(Icon)`** Création d'une étiquette avec une image uniquement

D'autres constructeurs, pouvant combiner le texte et l'image, existent.

L'étiquette de la Figure 16 est composée d'un texte et une icône. La position du texte par rapport à l'image peut être indiquée par l'intermédiaire des méthodes `setHorizontalAlignment` et `setVerticalAlignment` en spécifiant une des constantes prédéfinies suivantes :

`SwingConstants.RIGHT` ou `JLabel.RIGHT`: alignement sur la droite

`SwingConstants.CENTER` ou `JLabel.CENTER`: alignement sur le centre

`SwingConstants.LEFT` ou `JLabel.LEFT`: alignement sur la gauche

`SwingConstants.TOP` ou `JLabel.TOP`: alignement sur le haut

`SwingConstants.BOTTOM` ou `JLabel.BOTTOM`: alignement sur le bas

D'autres méthodes peuvent exister pour ce composant :

- `getIcon()` et `setIcon()` permettent de manipuler l'image du label.
- `get/setDisplayedMnemonic()`: permettent de manipuler le mnémonique (caractère souligné) pour l'étiquette.
- `get/setLabelFor()`: gèrent l'association de l'étiquette à un composant. Si une mnémonique a été définie pour l'étiquette, elle permet d'accéder au composant en la combinant avec la touche Alt.

3.2.2 Le ToolTip

C'est une bulle d'aide qui s'affiche momentanément pour décrire un composant (Figure 17). Elle s'affiche lorsque le pointeur de la souris s'attarde sur le composant. Pour créer un tooltip, le programmeur n'a pas à utiliser directement la classe `JToolTip`. Il suffit qu'il utilise la méthode `setToolTipText`. Par exemple, pour associer une bulle d'aide à l'étiquette de la Figure 16, il suffit d'ajouter la ligne de code java suivante :

```
label.setToolTipText("Ceci est un label");
```

Figure 17 - Exemple d'un tooltip

3.2.3 Le séparateur

C'est un composant simple qui divise visuellement l'interface en sections en utilisant des lignes horizontales ou verticales (Figure 18). Il est très utilisé dans les menus et les barres d'outils. La création de séparateurs se fait en instanciant un objet de la classe `JSeparator` et l'ajoutant à l'interface.

Figure 18 - Exemple de séparateur

Il faut noter qu'il n'est pas toujours obligatoire d'utiliser la classe `JSeparator` pour créer un séparateur puisque les menus et les barres d'outils proposent des méthodes qui créent et ajoutent leurs propres séparateurs.

3.3 Les composants de choix

La programmation événementielle est une fonctionnalité principale dans une interface graphique. Ce n'est plus le programme qui dirige les actions de l'utilisateur à sa guise. En effet, l'utilisateur déclenche des évènements et réagit à ce qui se passe dans la fenêtre. Plusieurs composants permettent de déclencher des évènements comme par exemple, les boutons.

3.3.1 Le Bouton

Un bouton est le composant d'action de base dans Swing. C'est tout simplement un élément graphique sur lequel l'utilisateur peut cliquer pour déclencher une action. Le bouton ne fait rien tant que l'utilisateur n'a pas cliqué dessus. Ainsi, un simple click sur le bouton et l'évènement est déclenché et une exécution d'une action s'en suit. Qu'est-ce qui se passe exactement? Eh bien, vous devez définir ceci (voir chapitre 5). Un bouton dans l'action ressemble à ceci:

Figure 19 - Un exemple de boutons

Les boutons de la Figure 19 ont été placés l'un au dessus de l'autre. Ce placement n'est pas aléatoire mais plutôt fait par un gestionnaire de placement (voir chapitre 4).

La classe pour créer un bouton est le **`JButton`**. Les méthodes de base pour modifier les propriétés d'un bouton consistent à établir le texte (`get/setText`), les images (`get/setIcon`), et l'orientation (`get/setHorizontalAlignment` et `get/setVerticalAlignment`). Ces méthodes sont pratiquement similaires pour la plupart des composants Swing puisqu'ils héritent tous de la même classe, `JComponent`.

D'autres méthodes existent pour gérer les différents états d'un bouton. Un état est une propriété qui décrit un composant, généralement dans un paramètre avec des valeurs vrai ou faux. Ces états sont accessibles par des accesseurs

(méthodes) get et set. Un bouton peut avoir les états suivants : actif / inactif, sélectionnés / non sélectionné, etc.

3.3.2 Les cases à côcher et les boutons radio

Ces deux composants offrent à l'utilisateur des options (ou des réponses à une question) généralement sous forme de choix multiples. Ceux sont des objets graphiques à côcher/sélectionner. Donc ils ne présentent pas une grande différence dans leurs comportements. Seulement, dans le cas des cases à côcher, l'utilisateur peut sélectionner ou désélectionner au hasard plusieurs réponses à la même question. Par contre, les boutons radio sont regroupés et présentent à l'utilisateur les réponses possibles à une question sauf que le choix de la réponse est obligatoire et unique. Par exemple, si on demande à un étudiant quels sont les langages de programmation qu'il préfère apprendre ; là il peut en choisir plusieurs. Dans ce cas, il faut utiliser les cases à côcher. Par contre, l'étudiant ne peut être à la fois masculin et féminin (Figure 20).

Figure 20 - Exemple de cases à côcher et boutons radio

Les classes permettant de créer des cases à côcher et des boutons radio sont respectivement `JCheckBox` et `JRadioButton`. La classe qui met en groupe ces objets est `ButtonGroup`. Elle permet, par exemple, de récupérer les réponses de l'utilisateur ou d'imposer un choix obligatoire et unique dans le cas des boutons radio, c'est-à-dire, quand un bouton est sélectionné l'autre est automatiquement désélectionné.

3.3.3 Le combo box

C'est un composant sous la forme d'une liste déroulante préétablie non modifiable. Il permet à l'utilisateur de choisir une valeur parmi plusieurs. Ce composant dispose d'un bouton et une liste déroulante de valeurs. Il fait la même chose qu'un groupe de boutons radio qui est pratiquement plus facile à comprendre pour les utilisateurs. Par contre, un combo box peut être plus approprié lorsque l'espace dans l'interface graphique est trop limité et/ou le nombre de choix disponible est important. Par exemple, une liste déroulante représentant les jours de la semaine est mieux approprié qu'un groupe de sept boutons radio (Figure 21). Les zones de listes déroulantes nécessitent peu d'espace à l'écran.

Figure 21 - Un combo box avant et après avoir clicker sur le bouton

Pour créer un combo box, on utilise la classe `JComboBox`. Le constructeur utilisé requiert forcément un argument de type tableau de chaine de caractères (String). La liste déroulante contient alors les chaines de caractères du tableau. Pour ajouter le composant de la Figure 21 à notre interface, il suffit d'ajouter au programme les lignes de code suivantes:

```
String [] noms={"dimanche", "lundi", "mardi",
                "mercredi", "jeudi", "vendredi", "samedi"};
JComboBox cb= new JComboBox(noms);
c.add(cb);
```

3.3.4 La liste de valeurs

Les listes de valeurs ne sont pas très attrayantes, mais elles sont plus appropriées que les combos box lorsque le nombre de choix est important (par exemple, les mois de l'année) ou éventuellement lors de la sélection de multiples choix (Figure 22).

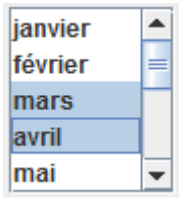


Figure 22 - Exemple d'une liste de valeurs

Pour créer une liste de valeurs il faut utiliser la classe `JList`. Le constructeur de cette classe requiert un argument de type tableau de chaines de caractères contenant les valeurs à mettre dans la liste. Si le nombre de valeurs est important, alors l'affichage ne serait pas très élégant. Il serait judicieux de réduire le nombre de valeurs de la liste à afficher et d'utiliser un panneau de défilement par l'intermédiaire de la classe `JScrollPane`. Pour déterminer le nombre de valeurs à afficher, il faut utiliser la méthode `setVisibleRowCount`. Le constructeur de la classe `JScrollPane` requiert un argument de type `JList`. Les lignes de code ayant permis d'ajouter le composant de la Figure 22 à notre interface graphique sont les suivant :

```
String [] noms={"janvier", "février", "mars",
                "avril", "mai", "juin", "juillet", "août",
                "septembre", "octobre", "novembre", "décembre"};

JList list= new JList(noms);
list.setVisibleRowCount(5);
JScrollPane sp = new JScrollPane(list);
c.add(sp);
```

3.4 Les composants de saisie

Tous les composants permettant de saisir du texte héritent de la super classe `JTextComponent`. C'est une classe qui contient toutes les caractéristiques pour la création d'un éditeur, une zone ou un champ de texte. En revanche, pour créer ces objets, on ne peut pas instancier cette classe puisqu'elle est abstraite. En effet, il faut utiliser ses sous classes : `JTextField`, `JTextArea` et `JEditorPane`. `JPasswordField` et `JTextPane` sont également deux sous-classes particulières permettant la saisie du texte.

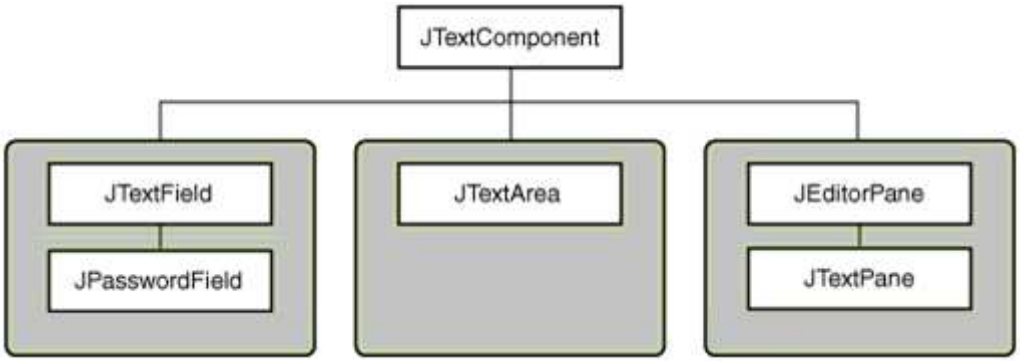


Figure 23 - Hiérarchie des composants de saisie de texte

Si le texte dépasse l'espace mis à disposition dans la zone de saisie, il faut placer le composant texte à l'intérieur d'un `JScrollPane`. Ce dernier permet de visualiser tout le texte à l'aide de barres de défilement. Dans ce cours, nous ne développons que les classes `JTextField`, `JTextArea` et `JPasswordField`.

3.4.1 Un champ de texte

Pour afficher, saisir et éditer du texte simple sur une seule ligne, il suffit d'utiliser la classe `JTextField` et la sous classe `JPasswordField` (Figure 23). Il est toutefois possible de lire ou modifier le texte à l'aide des méthodes `getText()` et `setText()` de la super classe `JTextComponent`. La classe `JTextField` dispose également de ses propres méthodes comme par exemple `setFont()` pour spécifier la police du texte et `setColumns()` pour définir le

nombre de caractères dans le champ de texte. Plusieurs constructeurs existent pour la classe `JTextField` :

- `JTextField()` (constructeur par défaut), champ de saisie vide de longueur zéro
- `JTextField (String str)` un champ de saisie avec `str` comme texte par défaut et une longueur qui est celle de `str`
- `JTextField(int nbr)`. champ de saisie vide de longueur `nbr`.

La propriété `horizontalAligment` permet l'alignement du texte à gauche, à droite ou au centre du champ de saisie selon les trois valeurs respectives : `JTextField.LEFT, JTextField.RIGHT` et `JTextField.CENTER`.

Lorsque le texte à saisir est confidentiel, comme par exemple pour un mot de passe, il faut utiliser la classe `JPasswordField`. Dans ce cas, un caractère écho * (par défaut) est affiché à la place des caractères saisis par l'utilisateur.

La Figure 24 regroupe des exemples instanciant ces deux classes.

Figure 24 - Champ de texte simple

3.4.2 Zone de texte

Dans le cas de la saisie et l'édition du texte simple sur plusieurs lignes (

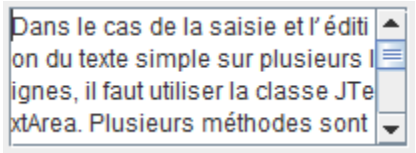


Figure 25), il faut utiliser la classe `JTextArea`. La taille de la zone de texte est celle du texte défini par défaut. Elle augmente au fur et à mesure de la saisie du nouveau texte. Pour que la taille de la zone ne change pas, il faut la mettre dans un `JScrollPane`.

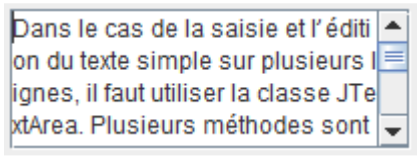


Figure 25 - Exemple d'une zone de texte

Plusieurs méthodes sont définies ou héritées par cette classe pour manipuler le texte. La méthode `setText()` est utilisée pour initialiser le texte dans la zone de texte. Dans le cas où on veut préserver le texte existant et en ajouter un autre à la fin, on utilise la fonction `append()`. La méthode `insert()` permet d'insérer du texte dans celui existant à une position donnée.

Chapitre 4

4 Les gestionnaires de placement

Dans ce chapitre, nous présentons les composants utilisés pour positionner d'autres composants. Ils sont appelés des gestionnaires de placement (ou layout). Ils sont définis dans la librairie AWT. Ils gèrent la position des composants les uns par rapport aux autres dans une fenêtre. Ils redéfinissent les dimensions des composants et les espaces entre eux pendant l'agrandissement de la fenêtre. Les tailles dépendent de la stratégie utilisée par le layout.

Chaque composant a deux tailles: la taille effective (`size`) et la taille idéale (`preferredSize`). Le layout rend la taille réelle la plus proche possible de la taille préférée compte tenu des contraintes dues à sa stratégie et dues à la taille réelle du conteneur. Pour désigner le layout qui va gérer la position des composants, on utilise la méthode `setLayout()`.

4.1 Les layouts classiques

4.1.1 Le FlowLayout

C'est le gestionnaire par défaut pour les `JPanel`. Il place les composants les uns à la suite des autres de la gauche vers la droite, sur une même ligne, en passant à la ligne suivante s'il n'y a plus de place (**Erreur ! Source du renvoi introuvable.**). Contrairement aux autres gestionnaires de placement, ce gestionnaire respecte la taille des composants. On peut aussi imposer une taille à un composant en utilisant la méthode `setPreferredSize`.

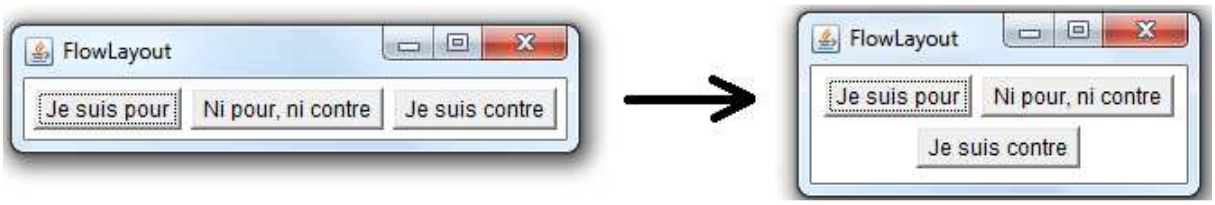


Figure 26 - Fenêtre avec un FlowLayout

La ligne contenant les composants peut être ajustée à gauche par défaut en utilisant le constructeur `conteneur.setLayout (new FlowLayout())`. Par ailleurs, on peut préciser l'ajustement de la ligne contenant les composants à droite (`RIGHT`), au centre (`CENTER`) ou à gauche (`LEFT`). Par exemple, l'instruction `conteneur.setLayout (new FlowLayout(FlowLayout.CENTER, 10, 15)` permet de centrer les composants sur les différentes lignes. Les valeurs 10 et 15 servent à espacer les composants en hauteur (15) et en largeur (10).

Remarque : l'emploi de la méthode `pack()` sur la fenêtre lui permet de prendre la plus petite taille possible en fonction des composants qu'elle contient.

4.1.2 Le BorderLayout

Ce gestionnaire divise son espace de travail en cinq zones : Centre, Nord, Sud, Est et Ouest (Figure 27). Si on veut mettre un composant dans une zone

donnée, il faut préciser cette zone, pendant l'ajout du composant, dans la méthode `add()`. Il n'est pas obligatoire d'occuper toutes les zones. Si l'ajout se fait par défaut, le composant occupe la zone centrale. Une zone ne peut contenir qu'un seul composant qui occupe le maximum d'espace disponible dans la zone. Dans le cas où on veut placer plusieurs composants dans une zone, on doit d'abord y placer un conteneur intermédiaire dans lequel on met ces composants.

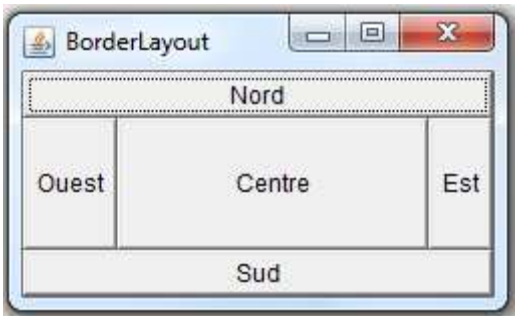


Figure 27 - Une fenêtre avec un BorderLayout

Dans cet exemple (code de l'exercice 6.1.8), nous n'avons pas utilisé la fonction `setLayout` pour désigner le gestionnaire de placement car `BorderLayout` est le gestionnaire par défaut des containers de type `Window` (`JFrame` dans notre cas).

4.1.3 Le GridLayout

Ce gestionnaire place les composants dans une grille où chaque composant occupe le même espace. On peut créer un `GridLayout` en précisant les nombre de lignes et de colonnes dans le constructeur (par exemple new `GridLayout`(3,3)). Toutes les lignes et les colonnes prennent la même dimension de façon à ce que les composants soient alignés. Pour le placement des composants, le gestionnaire remplit toutes les cases de la première ligne puis il passe à la ligne suivante et ainsi de suite.

Figure 28 - Exemple d'un GridLayout

Un espace vertical et horizontal peut être inséré entre les cases de la grille en spécifiant deux arguments supplémentaires lors de l'appel du constructeur. Ces deux arguments indiquent respectivement l'espacement horizontal et l'espacement vertical. Ces espacements sont de 5 points par défaut. Ils peuvent également être modifiés en utilisant les méthodes `setHgap` et `setVgap`. Un des deux paramètres donnant le nombre de lignes ou de colonnes peut être égal à zéro. Auquel cas, il n'est pas pris en compte et le nombre de lignes ou de colonnes s'adapte au nombre de composants.

Si le nombre de composants à insérer est plus grand que le nombre de cases dans le `GridLayout`, des cases supplémentaires sont créées en augmentant le nombre de colonnes et en respectant le nombre de lignes d'origine.

4.2 Les layouts spécialisés

4.2.1 Le GridbagLayout

C'est un gestionnaire d'une très grande utilité, très souple mais aussi très complexe. Les composants dans ce gestionnaire sont disposés dans une grille comme le `GridLayout` ; seulement chaque composant occupe un nombre de lignes et/ou de colonne variables. Ainsi les tailles des lignes et des colonnes ne sont pas toutes les mêmes. Le `GridBagLayout` fonctionne à la base comme un `GridLayout` en plaçant les composants à l'intérieur d'une grille sauf que

- Les tailles des lignes et des colonnes sont variables,
- Un composant, dont la taille dépasse celle d'une cellule, occupe une cellule issue de la fusion de cellules adjacentes.
- Un composant peut n'occuper qu'une partie de la surface de sa cellule
- Les positions des composants à l'intérieur de la cellule peuvent être modifiées.

Comme tout autre gestionnaire de disposition, Le `GridBagLayout` est attribué à un conteneur en utilisant la fonction setLayout().La particularité de ce gestionnaire réside principalement dans la méthode d'ajout des composants. Cette méthode a deux arguments : le composant à ajouter et un objet de la classe `GridBagConstraints` (contraintes de la grille). Ces contraintes ne sont rien d'autre que des attributs de la cellule, devant être positionnés pour chaque cellule séparément. Par exemple, pour ajouter *composant* à *panel* qui dispose d'un `GridBagLayout` on procède comme suit :

```
GridBagConstraints gbc = new GridBagConstraints();
composant.add(composant2, gbc);
```

La classe `GridBagConstraints` dispose d'un ensemble d'attributs permettant à `GridBagLayout` de positionner les composants.

- `GRIDX, GRIDY` : ces deux attributs servent à définir la position du composant dans la grille. Par exemple, si nous voulons mettre le composant dans la première case (en haut à gauche), nous le mettons à la première colonne (`gridx=0`) et la première ligne (`gridy=0`).
- `GRIDWIDTH, GRIDHEIGHT` : ces deux attributs permettent de définir la dimension de la grille. `gridwidth` (respectivement `gridheight`) pour définir le nombre de colonnes (respectivement lignes).
- `FILL` : Cet attribut est utilisé lorsque l'espace alloué au composant est supérieur à celui de ses désidératas et on veut le redimensionner. Quatre valeurs sont possibles en fonction du redimensionnement
  - `NONE : valeur par défaut`

- HORIZONTAL : le composant parent est rempli horizontalement
- VERTICAL : le composant parent est rempli verticalement
- BOTH : toute la zone contenant le composant est remplie

- IPADX, IPADY : Ces attributs sont utilisés pour définir les marges internes au composant
- INSETS : Cet attribut sert à préciser les marges en pixels autour du composant
- ANCHOR : Cet attribut permet de spécifier un point d'ancrage à un composant à l'intérieur de sa (ses) cellule(s).

Exemple

La Figure 29 représente un IHM contenant des boutons placés dans une grille. Le premier bouton occupe la première ligne. Le deuxième, occupe le tiers de la deuxième ligne. Le troisième occupe le reste de la deuxième ligne. Les trois derniers boutons partagent la troisième et dernière ligne.

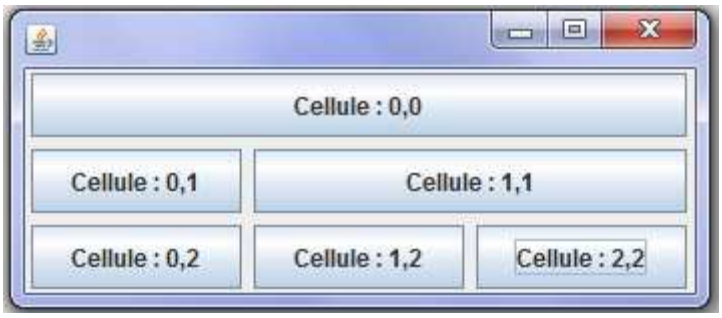


Figure 29 - Exemple d'un GridBagLayout

4.2.2 Le CardLayout

Ce gestionnaire permet de placer des composants d'une IHM sous la forme d'une pile ; Comme un jeu de cartes (Card = Carte). Le composant visible est toujours le premier ajouté dans la fenêtre. Le fait de ne faire figurer qu'un seul composant a en effet plusieurs intérêts comme par exemple ne pas encombrer l'interface graphique. De ce fait, il est possible de choisir un composant, plutôt

qu'un autre en fonction de la (ou les) action(s) à effectuer tout en ayant la possibilité d'accéder aux autres composants non visible. Pour pouvoir accéder à ces composants

- On conserve la référence de l'objet instanciant `CardLayout`. C'est à dire au lieu d'utiliser l'instruction suivante pour attribuer le layout au conteneur

```
conteneur.setLayout(new CardLayout());
```

On utilise plutôt les deux instructions suivantes

```
 CardLayout carte = new CardLayout();
conteneur.setLayout(carte);
```

- On utilise des méthodes associées à la classe `CardLayout` permettant de parcourir par la suite les divers composants et/ou de rendre visible tel composant plutôt qu'un autre.

La classe CardLayout dispose d'un certain nombre de méthodes permettant d'accéder aux différentes cartes de l'interface. Avant de parler de ces méthodes et afin d'accéder directement à un composant non visible dans l'interface graphique, il faut associer ce composant à une variable de type String. Ceci se fait lors de l'ajout du composant au conteneur en utilisant la méthode `add()`. Par exemple, au lieu d'écrire `conteneur.add(bouton);` on écrit `conteneur.add(bouton, "btn");` "btn" est en fait la chaine de caractères qui sert à identifier le composant.

Les méthodes permettant d'accéder à ces composants sont :

`public void show(Container parent, String name)`: Affiche le composant ou le sous-conteneur associé à la chaîne de caractères `name`.

`public void next(Container parent)` : Affiche la carte suivante.

`public void previous(Container parent)` : Affiche la carte précédente.

`public void first(Container parent)` : Aller à la première carte.

`public void last(Container parent)` : Aller à la dernière carte.

`parent` doit être égal au conteneur auquel est associé ce layout.

Exemple

L'exemple de la Figure 30 représente une interface graphique avec comme gestionnaire de placement un CardLayout.

Figure 30 - Exemple d'un CardLayout

4.2.3 Le BoxLayout

Ce gestionnaire est utilisé dans le cas où on veut placer les composants soit sur une ligne ou une colonne. C'est le gestionnaire par défaut du conteneur `Box`. A la différence d'un `GridLayout` avec une seule ligne ou une seule colonne, le `BoxLayout` ne met pas de contraintes sur la taille des composants ; pas forcément la même taille pour tous les composants. Il est de même pour le `FlowLayout` qui force les composants à aller à la ligne s'il n'y a pas assez de place horizontalement. L'attribution de ce gestionnaire à un conteneur est un peu différente par rapport aux autres. Le conteneur doit être créé avant le BoxLayout, car celui-ci en a besoin de son constructeur.

Nous montrons dans les deux lignes de code java suivantes comment attribuer un `BoxLayout` à un `JPanel`, par exemple, avec un alignement horizontal.

```
JPanel pan = new JPanel();
Pan.setLayout(new BoxLayout(pan, BoxLayout.X_AXIS));
```

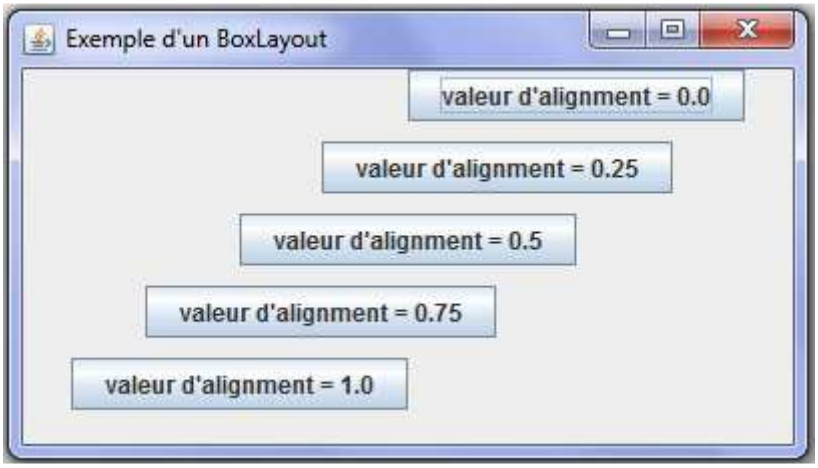


Figure 31 - Exemple d'un BoxLayout

Dans un conteneur dont le placement est géré par un `BoxLayout`, on peut gérer l'alignement des composants (Figure 31) placés verticalement en utilisant la méthode `setAlignment(`*value*`)`. Le paramètre *value* prend une valeur entre 0f et 1f. Pour aligner les composants à gauche (resp. à droite) *value* prend la valeur 0f (resp. 1f). La valeur 0.5f est utilisée pour aligner les composants au milieu. Il est également possible de conserver la taille des composants lorsqu'on redimensionne la fenêtre en ajoutant des ressorts (glue). Cette fonctionnalité est possible en utilisant les méthodes `createGlue()`, `createHorizontalGlue()` et `createVerticalGlue()` de la classe `Box`. Une autre fonctionnalité consiste à ajouter des séparateurs verticaux (resp. horizontaux) entre les composants en utilisant les méthodes `createVerticalStrut(`*value*`)` (resp. `createHorizontalStrut(`*value*`)`). *value* représente la hauteur (resp. l'épaisseur) du séparateur.

4.3 Gestion de placement personnalisée

4.3.1 Créer son propre gestionnaire de placement

Comme nous l'avons vu plus haut, un gestionnaire de placement gère le placement des composants dans le conteneur. Puisque chaque conteneur dispose déjà d'un gestionnaire de placement par défaut, ce dernier doit être désactivé avant de créer son propre gestionnaire.

La création d'un gestionnaire de placement personnalisée, nécessite la création d'une classe qui implémente l'interface `LayoutManager`. Soit on met en œuvre directement cette interface, soit sa sous-interface, `LayoutManager2`. Par conséquent, le gestionnaire de placement doit mettre en œuvre au moins les cinq méthodes suivantes, qui sont requises par l'interface `LayoutManager`

```
void addLayoutComponent(String, Component)

void removeLayoutComponent(Component)

Dimension preferredLayoutSize(Container)

Dimension minimumLayoutSize(Container)

void layoutContainer(Container)
```

Ces méthodes sont associées aux différents comportements (évènements) des composants dans le conteneur (voir chapitre 5)

4.3.2 Positionnement absolu

Les gestionnaires de placements prennent en charge le positionnement et le dimensionnement automatique des composants graphiques dans les conteneurs. Par ailleurs, et dans des situations bien précises, on a besoin de fixer manuellement la position et les dimensions des composants.

Le positionnement absolu permet de contrôler la position de chaque composant par rapport aux bords de la fenêtre ainsi que de définir la taille de ces composants. Pour gérer soi-même le positionnement (absolu) des composants, il faut tout d'abord mettre le gestionnaire de placement du conteneur à null en utilisant la méthode `setLayout`(null). Ensuite et afin de définir la taille et la position de chaque composant, on utilise la méthode `setBounds`(). Il s'agit de définir la taille et la position de chaque composant. L'origine étant le coin supérieur gauche du conteneur. Finalement, pour tout mettre en œuvre, on appelle la méthode `repaint`().

Chapitre 5

5 Gestion des événements

Une interface graphique impose une façon particulière de programmer. C'est de la "programmation à bas d'événements". L'utilisateur effectue des actions sur les composants l'interface en utilisant le clavier et/ou la sourie (frappe d'une touche de clavier, clic de souris, …). Ces actions génèrent des événements. Ces derniers sont récupérés et traités un à la suite de l'autre par le programme.

Cette gestion diffère d'un langage à un autre. Java utilise une architecture de type "émetteur - récepteur". Les composants graphiques (boutons, listes, menus,…) sont les émetteurs. Chacun a son récepteur (ou écouteurs ou encore listeners) qui s'enregistrent auprès de lui pour déterminer les traitements nécessaires. Ainsi, Java se charge de déterminer la source de l'événement et établir les liens avec les écouteurs. Les codes des méthodes, qui vont faire les traitements des événements, sont écrits par le développeur.

5.1 Les événements

Un événement est une information émise par un composant appelé émetteur (ou source d'événement) et qui pourrait déclencher des réactions sur d'autres éléments appelés récepteurs. Si on clique, par exemple, sur une touche de clavier, le système découvre l'événement et détermine quelle application contrôle cette touche de clavier, le cas échéant, l'événement est ignoré. Il passe les informations à cette application. C'est-à-dire qu'il faut que les écouteurs soient prévenus pour gérer les événements que va leur transmettre la source d'événement. Pour ce faire, à chaque type d'événement est prévue une interface écouteur que doit implémenter un objet qui désire intercepter cet événement. Les sources et les écouteurs d'événements sont des objets. Généralement, les événements et les interfaces écouteurs sont définis soit dans la librairie java.awt.event ou dans javax.swing.event. Les événements sont catégorisés en événements de bas niveau et événements de haut niveau ou sémantiques.

5.1.1 Les événements de bas niveau

Ce type d'événements est généré directement par des actions élémentaires de l'utilisateur. Par exemple, appuyer sur un bouton de la souris est un événement élémentaire. Relâcher ce bouton étant un autre événement élémentaire. Par ailleurs, un clic de souri est également un événement de bas niveau. Pour capturer ces d'événements, des classes prédéfinies permettent de définir l'événement qu'on veut intercepter. Par exemple, la classe KeyEvent permet de définir un événement clavier. Bien évidemment, il existe d'autres classes, nous en citons quelques unes :

- WindowEvent: fenêtre activée, désactivée, réduite, fermée, ...
- ContainerEvent: ajout, suppression, etc. d'un composant dans un conteneur

- ComponentEvent : déplacement, affichage, masquage ou modification de taille d'un composant.
- FocusEvent: obtention ou perte du focus du clavier sur un composant.
- MouseEvent: appui, relâchements, clics de souris, et les entrée /sortie de la souri dans un composant.

5.1.2 Les événements de haut niveau

Ces événements appelés aussi sémantiques sont engendrés par plusieurs actions élémentaires. L'ensemble des ces actions correspond à une action complète de l'utilisateur. Par exemple, choisir un élément dans une liste, modifier le texte d'une zone de texte, écrire un A majuscule, etc. Cette dernière action, par exemple, engendre quatre événements :

- appui sur la touche Majuscule
- appui sur la touche A
- relâchement de la touche A
- relâchement de la touche Majuscule

Comme pour les événements de bat niveau, il existe des classes pour définir les événements sémantiques. Nous en citons quelques-unes:

- ActionEvent : sélection ou clic d'un élément
- ItemEvent : sélection dans une liste ou dans un groupe de cases à cocher
- TextEvent : modification du texte d'un composant texte

5.2 Interception des événements

5.2.1 Les écouteurs

Comme nous l'avons précédemment dit, les sources d'événements et les écouteurs sont des objets. L'objet écouteur peut se déclarer comme étant

l'objet qui va écouter un certain événement survenant sur un objet source s'il s'en intéresse. Un écouteur est donc un objet qui intercepte des événements qui se produisent sur un ou plusieurs autres objets. Il définit ainsi la ou les méthodes contenant les traitements associés à ces événements. Ces méthodes prennent un événement en argument et se déclenchent chaque fois que cet événement survient sur l'objet source. Par exemple, l'écouteur du bouton "**Annuler**" dans un formulaire déclenche la méthode associée qui annule les informations saisies dans le formulaire et le ferme.

Il faut noter qu'un écouteur peut écouter plusieurs composants. Par exemple, deux boutons dans une IHM peuvent intéresser le même écouteur (ActionListener). Ce dernier doit identifier le bouton qui a agi pour déclencher le traitement qui lui a été dédié. Il est aussi possible que deux écouteurs s'intéressent au même composant. Par exemple, un bouton dans une interface pourrait avoir deux écouteurs. Un écouteur (MouseListener) pour surveiller les clics de souri sur le bouton et écouteur (MouseMotionListener) pour surveiller la souri quand entre et sort du bouton.

Nous décrivons ci-dessous quelques écouteurs (les plus utilisés). Pour chacun d'eux, nous citons les sources des événements en question et la méthode permettant de relier ces événements aux écouteurs. Nous citons également les méthodes qui devraient être déclenchées en réaction à ces événements.

- WindowListener : activer, désactiver, réduire, fermer, etc. une fenêtre
  - Evénements générés par : Window avec la méthode addWindowListener
  - Méthodes : windowActivated(WindowEvent e)
    windowDeactivated(WindowEvent e)
    windowClosing(WindowEvent e)

windowClosed(WindowEvent e)

windowDeiconified(WindowEvent e)

windowIconified(WindowEvent e)

windowOpened(WindowEvent e)

- ContainerListener : ajout ou suppression d'un composant dans un conteneur
  - Evénements générés par : Container avec la méthode addContainerListener
  - Méthodes : componentAdded(ContainerEvent e)

    componentRemoved(ContainerEvent e)
- ComponentListener : masquage, déplacement, affichage ou modification de taille de composants
  - Evénements générés par : Component avec la méthode addComponentListener
  - Méthodes : componentHidden(ComponentEvent e)

    componentMoved(ComponentEvent e)

    componentShown(ComponentEvent e)

    componentResized(ComponentEvent e)
- ActionListener : sélection ou clic d'un élément
  - Evénements générés par : JButton, JComboBox, JTextField, JList, etc. avec la méthode addActionListener
  - Méthode : actionPerformed(ActionEvent e)
- KeyListener : presser, relâcher ou taper une touche du clavier
  - Evénements générés par : Component avec la méthode addKeyListener
  - Méthodes : keyPressed(KeyEvent e)

    keyReleased(KeyEvent e)

    keyTyped(KeyEvent e)
- MouseListener : presser, relâcher, cliquer, etc sur un bouton de souri, ou mouvement de son pointeur

- Evénements générés par : Component avec la méthode addMouseListener
- Méthodes : mousePressed(MouseEvent e)
  mouseReleased(MouseEvent e)
  mouseClicked(MouseEvent e)
  mouseExited(MouseEvent e)
  mouseEntered(MouseEvent e)

- ItemListener : sélection dans une liste ou dans un groupe de cases à cocher
  - Evénements générés par : AbstractButton, JList, Choice, JComboBox, JCheckbox, CheckboxMenuItem, etc.
  - Méthodes : itemStateChanged(ItemEvent e)
- TextListener : modification du texte d'un composant texte
  - Evénements générés par : TextComponent avec la méthode addTextListener
  - Méthodes : textValueChanged(TextEvent e)

5.2.2 Les adaptateurs

Comme nous l'avons vu ci-dessus, la gestion des événements en utilisant un écouteur impose l'implémentation de toutes les méthodes de l'interface correspondante. Prenant, par exemple, le cas où la source de l'événement est la souri et que l'interface auditrice (MouseListener) s'intéresse uniquement aux clics de la souri. Dans ce cas, il faut définir le traitement associé uniquement dans la méthode mouseClicked(). Par ailleurs, l'interface MouseListener comportent bien quatre autres méthodes : mousePressed(), mouseReleased(), mouseEntered, et mouseExited. Ces méthodes doivent apparaître dans le code même si elles restent vides. Pour remédier à ce problème, on utilise les classes adaptatrices. Ceux sont des classes qui implémentent les interfaces écouteurs dont les méthodes ne sont pas toutes à développer. En conséquence, si la source dévénement ***étend*** une classe adaptatrice, on ne redéfinit que les méthodes que

l'on souhaite utiliser. Donc l'adaptateur évite à citer les méthodes non utilisées dans le programme.

Il faut noter que seules les interfaces écouteurs, ayant plusieurs méthodes, sont accompagnées de classes adaptatrices. Nous en citons ci-dessous quelques-unes.

- WindowAdapter
- ContainerAdapter
- ComponentAdapter
- KeyAdapter
- FocusAdapter
- MouseAdapter
- MouseMotionAdapter

5.3 Gestion des événements

Le composant doit prévenir ses écouteurs de l'arrivée de l'événement qu'elles désirent recevoir. A cet effet, la classe écouteur doit implémenter les interfaces correspondantes. Elle se s'enregistre auprès de la source d'événement, et quand elle reçoit les événements souhaités, elle les traite grâce aux méthodes de l'interface écouteur. Ces méthodes doivent être définies dans l'objet (classe) qui implémente l'interface écouteur. Par ailleurs, pour un type d'événement, il implémenter la bonne interface et développer la bonne méthode. Par exemple, pour un événement de type ActionEvent on implémente l'interface ActionListener et développer la méthode actionPerformed (ActionEvent e).

Hélas, tout ça n'est pas suffisant. Le fait d'implémenter l'interface écouteur ne suffit pas pour déclencher les méthodes sur les événements. Il faut, en fait, relier la source de l'événement à l'écouteur. Pour ce faire, il faut

recenser l'écouteur dans la source en utilisant une méthode addXxxListener. Cette méthode prend en argument un objet qui implémente l'interface XxxListener.

La relation entre la source d'événement et l'écouteur peut être définie différentes façons :

5.3.1 La source implémente elle même l'écouteur

Dans cet exemple, la source d'événement est la classe CoulF. Cette classe implémente elle-même l'écouteur de l'événement qui est l'interface ActionListener. Ainsi, tout se passe dans la même classe ; la source et l'écouteur de d'événement, le lien entre eux et le traitement associé.

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

class CoulFInd extends JFrame implements ActionListener{
      private JButton boutoncouleur;
      private Color couleur;
      private Container c;
      public CoulFInd(){
            setTitle("Color Mem");
            setSize(200,150);
            boutoncouleur=new JButton("Rouge");
            boutoncouleur.setBackground(Color.white);
            c=getContentPane();
            c.setLayout(new FlowLayout());
            c.setBackground(Color.red);
            add(boutoncouleur);
            boutoncouleur.addActionListener(this);
            setLocationRelativeTo(this.getParent());
            setDefaultCloseOperation(3);
      }
      public void actionPerformed(ActionEvent evt) {
            if (boutoncouleur.getText()=="Rouge") {
                  couleur=Color.yellow;
                  boutoncouleur.setText("Jaune");}
            else {
                  couleur=Color.red;
                  boutoncouleur.setText("Rouge");}
            c.setBackground(couleur);
      }
}

public class EcouteurMeme{
```

```
	public static void main(String [] argv){
		CoulFInd cf=new CoulFInd();
		cf.setVisible(true);
	}
}
```

5.3.2 La source est indépendante de l'écouteur

Dans cette deuxième façon de définir le listener, nous définition la source d'événement et le lien avec l'écouteur dans une classe (la classe CoulF ici). L'écouteur d'événement et les traitements associés sont définis dans une autre classe (la classe EcouteBouton).

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class CoulF extends JFrame{
	private JButton boutoncouleur;
	private Color couleur;
	private Container c;
	public CoulF(){
		setTitle("Color Ind");
		setSize(200,150);
		boutoncouleur=new JButton("Rouge");
		boutoncouleur.setBackground(Color.white);
		c=getContentPane();
		c.setLayout(new FlowLayout());
		c.setBackground(Color.red);
		add(boutoncouleur);
		EcouteBouton bouton = new EcouteBouton (this, boutoncouleur);
		boutoncouleur.addActionListener(bouton);
		setLocationRelativeTo(this.getParent());
		setDefaultCloseOperation(3);
	}
}
class EcouteBouton implements ActionListener{
	private JFrame frame; JButton boutoncouleur; Color couleur;
	public EcouteBouton(CoulF frame, JButton boutoncouleur) {
		this.frame = frame;
		this.boutoncouleur = boutoncouleur;
	}
	public void actionPerformed(ActionEvent evt) {
		if (boutoncouleur.getText()=="Rouge") {
			couleur=Color.yellow;
			boutoncouleur.setText("Jaune");}
		else {
			couleur=Color.red;
			boutoncouleur.setText("Rouge");}
		frame.setBackground(couleur);
	}
}
```

```
public class EcouteurIndependant{
	public static void main(String [] argv){
		CoulF cf=new CoulF();
		cf.setVisible(true);
	}
}
```

5.3.3 L'écouteur est dans la source

Dans ce cas, l'écouteur est un objet qui instancie une classe implémentant une interface. Le plus important est que cette classe est définie dans la source. L'objet écouteur est passé comme paramètre de la fonction addXxxListener pour le relier à la source d'événement.

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

public class EcouteurInterne extends JFrame{
	private JButton boutoncouleur;
	private Color couleur;
	private Container c;
	class EcouteBouton implements ActionListener{
		public void actionPerformed(ActionEvent evt) {
			if (boutoncouleur.getText()=="Rouge") {
				couleur=Color.yellow;
				boutoncouleur.setText("Jaune");}
			else {
				couleur=Color.red;
				boutoncouleur.setText("Rouge");}
			c.setBackground(couleur);
		}
	}
	private EcouteBouton eb = new EcouteurInterne.EcouteBouton();
	public EcouteurInterne(){
		setTitle("Color Int");
		setSize(200,150);
		boutoncouleur=new JButton("Rouge");
		boutoncouleur.setBackground(Color.white);
		c=getContentPane();
		c.setLayout(new FlowLayout());
		c.setBackground(Color.red);
		add(boutoncouleur);
		boutoncouleur.addActionListener(eb);
		setLocationRelativeTo(this.getParent());
		setDefaultCloseOperation(3);
	}
	public static void main(String [] argv){
		EcouteurInterne cf=new EcouteurInterne();
		cf.setVisible(true);
	}
}
```

5.3.4 L'écouteur est dans la source en tant qu'une classe anonyme

Dans ce dernier cas, l'écouteur est défini dans le paramètre de la méthode addXxxListener. Là on instancie directement l'interface (sans référencier l'objet résultant) avec les traitements nécessaires en réaction de l'événement.

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

public class EcouteurAnonym extends JFrame{
     private JButton boutoncouleur;
     private Color couleur;
     private Container c;
     public EcouteurAnonym(){
            setTitle("Color Ano");
            setSize(200,150);
            boutoncouleur=new JButton("Rouge");
            boutoncouleur.setBackground(Color.white);
            c=getContentPane();
            c.setLayout(new FlowLayout());
            c.setBackground(Color.red);
            add(boutoncouleur);
            boutoncouleur.addActionListener(new ActionListener(){
                  public void actionPerformed(ActionEvent evt) {
                        if (boutoncouleur.getText()=="Rouge") {
                              couleur=Color.yellow;
                              boutoncouleur.setText("Jaune");}
                        else {
                              couleur=Color.red;
                              boutoncouleur.setText("Rouge");}
                        c.setBackground(couleur);
                  }
            });
            setLocationRelativeTo(this.getParent());
            setDefaultCloseOperation(3);
     }
     public static void main(String [] argv){
            EcouteurAnonym cf=new EcouteurAnonym();
            cf.setVisible(true);
     }
}
```

Chapitre 6

6 Exercices avec solutions

6.1 Codes des exemples du polycopié

6.1.1 Code de la Figure 16 - Exemple d'une étiquette (§ 3.2.1)

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

public class Fenetre
{
     public static void main(String [] args)
     {
         // création d'une fenêtre d'application
          JFrame f = new JFrame("DIG Tuto");
          f.setSize(150, 150);
          Container c = f.getContentPane();
          c.setLayout(new FlowLayout());
          // création de l'image
          Icon icon = new ImageIcon("Victoire.png");
          // création du label
          JLabel label = new JLabel("Victoire");
          // atribuer une image à l'étiquette
          label.setIcon(icon);
          // position du label par rapport à l'image
          label.setVerticalTextPosition(SwingConstants.BOTTOM);
          label.setHorizontalTextPosition(SwingConstants.CENTER);
          // ajout du label au conteneur de la fenêtre
```

```
            c.add(label);
            f.setVisible(true);
            f.setDefaultCloseOperation(3);
        }
    }
```

6.1.2 Code de la Figure 18 - Exemple de séparateur (§ 3.2.3)

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

    public class Fenetre
    {
        public static void main(String [] args)
        {
            JFrame f = new JFrame("DIG Tuto");
            f.setSize(150, 150);
            Container c = f.getContentPane();
            c.setLayout(new FlowLayout());
            JPanel b = new JPanel();
            b.setLayout(new BoxLayout(b, BoxLayout.LINE_AXIS));
            JLabel label1=new JLabel("LABEL1");
            b.add(label1);
            b.add(Box.createHorizontalStrut(5));
            b.add(new JSeparator(SwingConstants.VERTICAL));
            b.add(Box.createHorizontalStrut(5));
            JLabel label2=new JLabel("LABEL2");
            b.add(label2);
            c.add(b);
            f.setVisible(true);
            f.setDefaultCloseOperation(3);
        }
    }
```

6.1.3 Code de la Figure 19 - Un exemple de boutons (§ 3.3.1)

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

    public class Fenetre
        public static void main(String [] args)
        {
            // création d'une fenêtre d'application
             JFrame f = new JFrame("DIG Tuto");
             f.setSize(200, 100);
             Container c = f.getContentPane();
             c.setLayout(new FlowLayout());
             // création d'un bouton
            JButton bouton1 = new JButton("Bouton N°1");
            JButton bouton2 = new JButton("Bouton N°2");
            // ajout du bouton au panneau de la fenêtre d'application
            c.add(bouton1);
```

```
            c.add(bouton2);
            f.setVisible(true);
            f.setDefaultCloseOperation(3);
        }
    }
```

6.1.4 Code de la Figure 20 - Exemple de cases à côcher et boutons radio (§ 3.3.2)

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

    public class Fenetre
    {
         public static void main(String [] args)
         {
            JFrame f = new JFrame("DIG Tuto");
             f.setSize(150, 150);
             Container c = f.getContentPane();
             c.setLayout(new GridLayout(0,2,3,3));
             JPanel pan1 = new JPanel();
             JPanel pan2 = new JPanel();

             JCheckBox cb1= new JCheckBox("Pascal");
             JCheckBox cb2= new JCheckBox("Java   ");
             JCheckBox cb3= new JCheckBox("C++    ");
             pan1.add(cb1);
             pan1.add(cb2);
             pan1.add(cb3);

             JRadioButton Sexe1 = new JRadioButton("Masculin");
             JRadioButton Sexe2 = new JRadioButton("Féminin  ");
             ButtonGroup bg = new ButtonGroup();
             bg.add(Sexe1);
             pan2.add(Sexe1);
             bg.add(Sexe2);
             pan2.add(Sexe2);

             c.add(pan1);
             c.add(pan2);
             f.setVisible(true);
             f.setDefaultCloseOperation(3);
         }
    }
```

6.1.5 Code de la Figure 24 - Champ de texte simple (§ 3.4.1)

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

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

public class Saisie {
     public static void main(String [] args){
         JFrame f = new JFrame();
         f.setSize(300, 90);
         Container c = f.getContentPane();
         c.setLayout(new GridLayout(0,2,1,1));
         JTextField tf = new JTextField("Saisissez votre texte");
         c.add(new JLabel(" JTextField"));
         c.add(tf);
         JPasswordField pf = new JPasswordField(10);
         c.add(new JLabel(" JPasswordField"));
         c.add(pf);
         f.setVisible(true);
         f.setDefaultCloseOperation(3);
     }
}
```

6.1.6 Code de la Figure 25 - Exemple d'une zone de texte (§ 3.2.1)

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

public class Saisie {
     public static void main(String [] args){
            JFrame f = new JFrame();
            f.setSize(300, 90);
         Container c = f.getContentPane();
         //c.setLayout(new GridLayout(0,2,1,1));
         JTextArea ta = new JTextArea("Dans le cas de la saisie et l'édition du
 texte simple sur plusieurs lignes, il faut utiliser la classe
JTextArea. Plusieurs méthodes sont définies ou héritées par cette
classe pour manipuler le texte.");
         ta.setLineWrap(true);
         JScrollPane sp = new JScrollPane(ta);
         sp.setPreferredSize(new Dimension(200,70));
         JPanel pan = new JPanel();
         pan.add(sp);
         c.add(pan);
         f.setVisible(true);
         f.setDefaultCloseOperation(3);
     }
}
```

6.1.7 Code de la Erreur ! Source du renvoi introuvable. (§ 4.1.1)

```
import java.awt.* ;
public class FLayout extends Frame{
	protected Component[] tab ;
	public FLayout(Component[] composants) {
		super("FlowLayout") ;
		setLayout(new FlowLayout()) ;
		tab = composants ;
		for ( int i = 0 ; i < tab.length; i++) {
			add(tab[i]);
		}
	}

	public static void main(String[] args) {
		Button[] boutons = { new Button ("Je suis pour"),
				new Button ("Ni pour, ni contre"),
				new Button ("Je suis contre"),
		} ;
		FLayout fenêtre = new FLayout(boutons) ;
		fenêtre.pack() ;
		fenêtre.show();
	}
}
```

6.1.8 Code de la Figure 27 - Une fenêtre avec un BorderLayout (§ 4.1.2)

```
import java.awt.*;
import javax.swing.* ;
public class BLayout {
	private JFrame fenetre;
	private JButton zone_nord, zone_sud, zone_est, zone_ouest, zone_centre ;
	public BLayout() {
		fenetre = new JFrame("BorderLayout") ;
		zone_nord = new JButton("Nord") ;
		zone_sud = new JButton("Sud") ;
		zone_est = new JButton("Est") ;
		zone_ouest = new JButton("Ouest") ;
		zone_centre = new JButton("Centre") ;
		// dispositions
		fenetre.add(zone_nord, BorderLayout.NORTH) ;
		fenetre.add(zone_sud, BorderLayout.SOUTH) ;
		fenetre.add(zone_est, BorderLayout.EAST) ;
		fenetre.add(zone_ouest, BorderLayout.WEST) ;
		fenetre.add(zone_centre, BorderLayout.CENTER) ;
		fenetre.setSize(250,150);
		fenetre.show();
	}
	public static void main(String[] args) {
		BLayout bl = new BLayout() ;
	}
}
```

6.1.9 Code de la Figure 28 - Exemple d'un GridLayout (§ 4.1.3)

```
import java.awt.* ;
import javax.swing.*;
public class TestButtonPanel {
     public static void main(String[] args) {
           JPanel grille = new JPanel ();
           grille.setLayout(new GridLayout(3, 3,5,5)) ;
           String[] cases = new String[] { "1", "2", "3", "4", "5",
                                           "6", "7", "8", "9" };
           for (int ix = 0 ; ix < cases.length; ix ++)
                 grille.add(new Button(cases[ix])) ;
           JFrame fenêtre = new JFrame("GridLayout") ;
           fenêtre.add(grille) ;
           fenêtre.pack() ;
           fenêtre.show() ;
     }
}
```

6.1.10 Code de la Figure 29 - Exemple d'un GridBagLayout (§ 4.2.1)

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

     public class GBLayout {

           public static void main(String[] args) {

                 JPanel pan = new JPanel();
                 pan.setLayout(new GridBagLayout());
                 GridBagConstraints gbc = new GridBagConstraints();
                 gbc.fill = GridBagConstraints.BOTH;

                 /* Marges entre les composant new Insets(margeSupérieure,
                  * margeGauche, margeInférieur, margeDroite) */
                 gbc.insets = new Insets(3, 3, 3, 3);

                 /* Si le composant n'occupe pas la totalité de l'espace
                  * disponible on le place au centre*/
                 gbc.ipady = gbc.anchor = GridBagConstraints.CENTER;
                 gbc.weightx = 3;  // nombre de cases en abscisse
                 gbc.weighty = 3;  // nombre de cases en ordonnée

                 /* ajout du premier composant en position (0, 0) */
                 gbc.gridx = 0;
                 gbc.gridy = 0;
                 gbc.gridwidth = 3;  // composant occupant 3 cases

                 /* Ajout du composant au pan en précisant
                  * le GridBagConstraints */
                 pan.add(new JButton("Cellule : 0,0"), gbc);
                 gbc.gridx = 0;
                 gbc.gridy = 1;
                 gbc.gridwidth = 1;
```

```
                pan.add(new JButton("Cellule : 0,1"), gbc);
                gbc.gridx = 1;
                gbc.gridy = 1;
                gbc.gridwidth = 2;
                pan.add(new JButton("Cellule : 1,1"), gbc);

                gbc.gridwidth = 1;
                gbc.gridy = 2;
                for (int i=0; i<3; i++){
                      gbc.gridx = i;
                      pan.add(new JButton("Cellule : "+i+",2"), gbc);
                }

                JFrame frame = new JFrame();
                frame.setSize(350, 150);
                frame.add(pan);
                frame.setLocationRelativeTo(null);
                frame.setVisible(true);
                frame.setDefaultCloseOperation(3);
          }
     }
```

6.1.11 Code de la Figure 30 - Exemple d'un CardLayout (§ 4.2.2)

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

public class CdLayout extends Frame {
  final static String st1 = "carte 1";
  final static String st2 = "carte 2";
  JPanel Cards, Command, pan1, pan2;
  Button bac1, bac2, bs, bp ;
  TextField textcarte1 ;
  TextField textcarte2 ;

  public CdLayout() {
     this.setLayout(new BorderLayout());
     this.setTitle("fenetre avec CardLaout");
     bac1 = new Button("Afficher Carte 2");
     bac2 = new Button("Afficher Carte 1");
     bs = new Button("Suivant");
     bp = new Button("Précédent");
     textcarte1 = new TextField("Carte 1");
     textcarte2 = new TextField("Carte 2");
     Cards = new JPanel();
     Command = new JPanel();
     pan1 = new JPanel();
     pan2 = new JPanel();
     Cards.setLayout(new CardLayout());
     Command.add(bp);
     Command.add(bs);
     pan1.add(bac1);
     pan1.add(textcarte1);
     Cards.add(st1,pan1);
     pan2.add(bac2);
     pan2.add(textcarte2);
     Cards.add(st2,pan2);
```

```
      add(Cards, "Center");
      add(Command,"North");
    }
  public boolean handleEvent(Event e)  {
     if (e.id == Event.WINDOW_DESTROY)  {
         System.exit(0) ;
        }
      return super.handleEvent(e) ;
  }

  public boolean action(Event e, Object arg)
  {
       if (e.target == bac1)   {
             ((CardLayout) Cards.getLayout()).show(Cards,st2);
       };
       if (e.target == bs)   {
             ((CardLayout) Cards.getLayout()).next(Cards);
       };

       if (e.target == bp)   {
             ((CardLayout) Cards.getLayout()).previous(Cards);
       };
       if (e.target == bac2)   {
             ((CardLayout) Cards.getLayout()).show(Cards,st1);
       };
       return true;
  }

 static public void main(String[] args)  {
    CdLayout f = new CdLayout();
    f.setSize(300,150);
    f.show();
  }
}
```

6.1.12 Code de la Figure 31 - Exemple d'un BoxLayout (§ 4.2.3)

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

public class BxLayout {
  public static void main(String[] args) {
    JFrame frame = new JFrame("Exemple d'un BoxLayout");
    Container pan = frame.getContentPane();
    pan.setLayout(new BoxLayout(pan, BoxLayout.Y_AXIS));
    for (float alignement = 0.0f; alignement <= 1.0f; alignement += 0.25f) {
      JButton btn = new JButton("valeur d'alignment = " + alignement);
      btn.setAlignmentX(alignement);
      pan.add(btn);
      pan.add(Box.createVerticalStrut(10));
    }
    frame.setSize(400, 300);
    frame.setVisible(true);
    frame.setDefaultCloseOperation(3);
  }
}
```

6.2 Pour mieux vous entrainer

6.2.1 Exercice 1

Enoncé

Soit l'interface suivante (Figure 32). Quand on click sur le bouton VALIDATION, il est affiché sur la console la case qui est cochée et celle qui ne l'est pas.

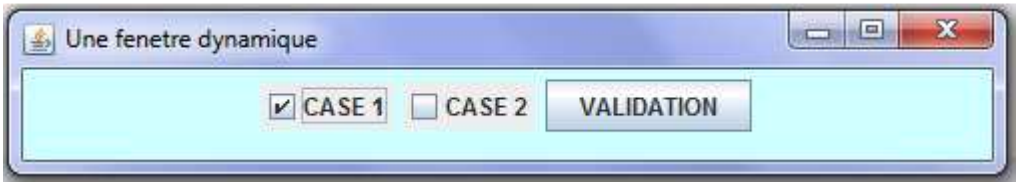


Figure 32 - Cases cochées

Solution

```
import javax.swing.*;
     import java.awt.*;
     import java.awt.event.*;
     class Fenetrecase4 extends JFrame {
         private JCheckBox cocher1, cocher2;
         private JButton bouton;
          public Fenetrecase4() {
              setTitle("Une fenetre dynamique");
              Container c = getContentPane();
              c.setBackground(new Color(208, 255, 255));
              setSize(400, 100);
              c.setLayout(new FlowLayout());
              cocher1 = new JCheckBox("CASE 1");
              c.add(cocher1);
              cocher2 = new JCheckBox("CASE 2");
              c.add(cocher2);
              setLocationRelativeTo(this.getParent());
              bouton = new JButton("VALIDATION");
              c.add(bouton);
              EcouteurDeBoutons eb = new EcouteurDeBoutons(cocher1, cocher2);
              bouton.addActionListener(eb);
              setDefaultCloseOperation(3);
          }
    }
    class EcouteurDeBoutons implements ActionListener {
         private JCheckBox cocher1, cocher2;
         public EcouteurDeBoutons(JCheckBox cocher1, JCheckBox cocher2) {
              this.cocher1 = cocher1;
              this.cocher2 = cocher2;
         }
```

```
        public void actionPerformed(ActionEvent a) {
             if(cocher1.isSelected())
                  System.out.println("la case 1 est cochée");
             else System.out.println("la case 1 n'est pas cochée");
             if(cocher2.isSelected())
                  System.out.println("la case 2 est cochée");
             else System.out.println("la case 2 n'est pas cochée");

         }
    }
    public class TesterClicCase4 {
          public static void main(String [] args) {
               Fenetrecase4 f = new Fenetrecase4();
               f.setVisible(true);
          }
    }
```

6.2.2 Exercice 2

Soit l'interface graphique suivante (Figure 33). Elle est constituée de deux boutons et deux labels étiquetés respectivement "Selected" et "UnSelected". Quand on click sur le BOUTON1, la valeur du label associé devient "Selected" et celle du label associé à BOUTON2 devient "UnSelected" et ainsi de suite.

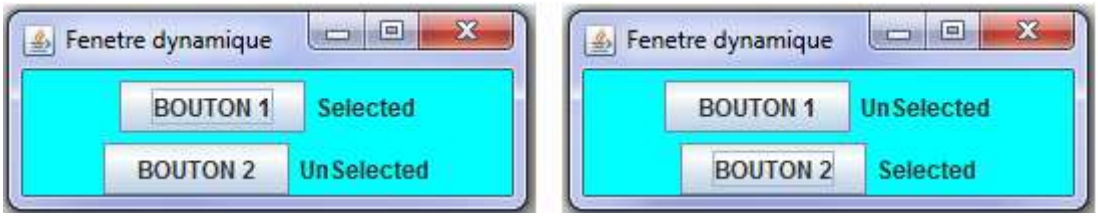


Figure 33 - La valeur du label dépond du bouton appuié

Solution

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

     class Fenetre2 extends JFrame      {
         private JButton bouton1, bouton2;
         private JLabel l1,l2;
          public Fenetre2()  {
              setTitle("Fenetre dynamique");
              Container c = getContentPane();
              c.setBackground(Color.cyan);
              setSize(260, 100);
              c.setLayout(new FlowLayout());
              l1=new JLabel("Selected");
```

```
            l2=new JLabel("UnSelected");
            bouton1 = new JButton("BOUTON 1");
            bouton2 = new JButton("BOUTON 2");
            c.add(bouton1); c.add(l1);
            c.add(bouton2); c.add(l2);
            setLocationRelativeTo(this.getParent());
            //EcouteurBouton1 eb= new EcouteurBouton1();
            bouton1.addActionListener(new EcouteurBouton1(l1,l2) );
            bouton2.addActionListener(new EcouteurBouton2(l1,l2) );
        }
    }
    class EcouteurBouton1  implements ActionListener {
     JLabel l1,l2;
     public EcouteurBouton1(JLabel l1, JLabel l2){
           this.l1=l1;  this.l2=l2;
     }
          public void actionPerformed(ActionEvent a1) {
              l1.setText("Selected");
              l2.setText("UnSelected");
          }
    }
    class EcouteurBouton2 implements ActionListener    {
     JLabel l1,l2;
     public EcouteurBouton2(JLabel l1, JLabel l2){
           this.l1=l1;  this.l2=l2;
     }
          public void actionPerformed(ActionEvent a1)    {
              l2.setText("Selected");
              l1.setText("UnSelected");
          }
    }
    public class BoutonLabel  {
          public static void main(String [] args)  {
               JFrame f = new Fenetre2();
               f.setVisible(true);
          }
   }
```

6.2.3 Exercice 3

Enoncé

Soit l'interface graphique suivante (Figure 34). Elle est constituée d'un label, d'une liste déroulante et d'un bouton. On choisit une valeur dans la liste déroulante, on valide en cliquant sur le bouton Validate et le label prend cette valeur.

Figure 34 - un label prend sa valeur d'une liste déroulante

Solution

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

class JListFrame extends JFrame{
      private JButton bouton;
      JLabel lab1, lab2;
      public JListFrame(){
            setSize(300,200);
            setLocationRelativeTo(this.getParent());
            setLayout (new FlowLayout());
            bouton=new JButton("Validate");
            String [] jours={"sa","di","lu","ma", "me","je","ve"};
            JList jl=new JList(jours);
            JScrollPane sp=new JScrollPane(jl);
            jl.setVisibleRowCount(3);
            jl.setSelectedIndex(0);
            lab1=new JLabel("Jour : ");
            lab2=new JLabel("sa");
            add (lab1);  add(lab2);
            add(new JLabel("          "));
            add(sp);
            add(bouton);
            bouton.addActionListener(new EcouteurJList(jl,lab2));
            setVisible(true);
            setDefaultCloseOperation(3);
      }
}
class EcouteurJList implements ActionListener{
      private JList jl;
      private JLabel lab2;
      public EcouteurJList(JList jl, JLabel lab2){
            this.jl=jl;  this.lab2=lab2;
      }
      public void actionPerformed(ActionEvent arg0) {
            lab2.setText((String) jl.getSelectedValue());
      }
}
public class TestJList {
      public static void main(String argv[]){
            JFrame jlf=new JListFrame();
      }
}
```

6.2.4 Exercice 4

Enoncé

Soit l'interface graphique suivante (Figure 35). Cette interface contient une liste déroulante et une liste de choix contenant chacune les mêmes valeurs de couleurs. Quand on choisit une couleur dans la liste déroulante, elle est sélectionnée dans la liste de choix et l'arrière plan prend cette couleur. Il est de même quand on choisit une couleur dans la liste de choix.

Figure 35 - couleur de l'arrière plan

Solution

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

class JListFrame extends JFrame{
     private JButton bouton;
     JLabel lab1, lab2;
     public JListFrame(){
            setSize(300,200);
            setLocationRelativeTo(this.getParent());
            setLayout (new FlowLayout());
            bouton=new JButton("Validate");
            String [] jours={"sa","di","lu","ma", "me","je","ve"};
            JList jl=new JList(jours);
            JScrollPane sp=new JScrollPane(jl);
            jl.setVisibleRowCount(3);
            jl.setSelectedIndex(0);
            lab1=new JLabel("Jour : ");
            lab2=new JLabel("sa");

            add (lab1);
            add(lab2);
            add(new JLabel("          "));
```

```
            add(sp);
            add(bouton);
            bouton.addActionListener(new EcouteurJList(jl,lab2));
            setVisible(true);
            setDefaultCloseOperation(3);
      }
}
class EcouteurJList implements ActionListener{
      private JList jl;
      private JLabel lab2;
      public EcouteurJList(JList jl, JLabel lab2){
            this.jl=jl;
            this.lab2=lab2;
      }
      public void actionPerformed(ActionEvent arg0) {
            lab2.setText((String) jl.getSelectedValue());
      }
}
public class TestJList {
      public static void main(String argv[]){
            JFrame jlf=new JListFrame();
      }
}
```

6.2.5 Exercice 5

Enoncé

Soit l'interface graphique suivante (Figure 36). Cette interface permet de convertir les dinars en euros ou l'inverse. Le choix du sens de la conversion se fait en utilisant le combo Box. Si on fait, par exemple, la conversion de l'euro en dinar, le champ de saisie euro doit être actif pour saisir les valeurs en euro à convertir et le champ dinar doit être inactif, mais il affiche la valeur de conversion.

Figure 36 - Un convertisseur de devis

Solution

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

class FenConvert extends JFrame implements DocumentListener, ItemListener{
     private JTextField champ1, champ2;
     private JLabel lab1,lab2;
     String []sens={"Dinars --> Euros","Euros --> Dinars"};
     private JComboBox sensbox;
     public FenConvert(){
            setTitle("[SITW] Convertisseur de devis");
            setSize(300,150);
            Container c =null;
            Box box =new Box(BoxLayout.Y_AXIS);
            sensbox= new JComboBox(sens);
            JPanel p3= new JPanel();
            p3.add(sensbox);
            p3.add(new JButton("conversion"));
            box.add(p3);
            sensbox.addItemListener(this);
            lab1=new JLabel("Dinars");
            champ1=new JTextField(15);
            JPanel p1=new JPanel();
            p1.add(lab1);
            p1.add(champ1);
            box.add(p1);
            champ1.getDocument().addDocumentListener(this);
            lab2=new JLabel("Euros");
            champ2=new JTextField(15);
            champ2.setEditable(false);
            JPanel p2=new JPanel();
            p2.add(lab2);
            p2.add(champ2);
            box.add(p2);
            add(box);
            setDefaultCloseOperation(3);
     }
     public void changedUpdate(DocumentEvent arg0) {
     }
     public void insertUpdate(DocumentEvent arg0) {
            String st=champ1.getText();
            double d=Double.parseDouble(st);
            double e=(d/145);
            champ2.setText((""+e).substring(0,     5));
     }
     public void removeUpdate(DocumentEvent arg0) {
            String st=champ1.getText();
            double d=Double.parseDouble(st);
            double e=(d/145);
            champ2.setText((""+e).substring(0,     5));
     }
     public void itemStateChanged(ItemEvent ebox) {
            switch (sensbox.getSelectedIndex()){
            case 0:
                  champ1.setEditable(true);
```

```
                champ2.setEditable(false);
                break;
            case 1:
                champ2.setEditable(true);
                champ1.setEditable(false);
                break;
            }
      }
}
public class Converstisseur {
      public static void main(String [] argv){
            FenConvert fc= new FenConvert();
            fc.setVisible(true);
      }
}
```

6.2.6 Exercice 6

Enoncé

Soit l'IHM suivante (Figure 37). Cette interface contient deux boutons étiquetés respectivement "Plus" et "Moins" et un label initialisé à la valeur 1. Quand on click sur le bouton Plus, la valeur du label est incrémentée de 1 et quand on click sur le bouton Moins, la valeur du label est décrémentée de 1.

Figure 37 - Calculette Plus-ou-Moins

Solution

```
import java.awt.*;
//import java.awt.*;
import java.awt.event.*;

class MonBouton extends Button {
  int incr;
  MonBouton(String titre, int incr) {
    super(titre);
    this.incr = incr;
  }
  int getIncr() { return incr; }
```

```
}
class ListenerLabel extends Label implements ActionListener {
       ListenerLabel() { super("0", Label.CENTER);}
       public void actionPerformed(ActionEvent e) {
         MonBouton b = (MonBouton)e.getSource();
         int c = Integer.parseInt(getText());
         c += b.getIncr();
         setText(Integer.toString(c));
       }
     }
class WindowCloser extends WindowAdapter {
       public void windowClosing(WindowEvent e) {
         System.exit(0);
       }
     }
class PlusouMoins extends Frame {
       public PlusouMoins() {
         super("Plus ou moins");
         Button oui = new MonBouton("Plus !", +1);
         Button non = new MonBouton("Moins !", -1);
         Label diff = new ListenerLabel();
         add(oui, "North");
         add(diff,"Center");
         add(non,"South");
         oui.addActionListener((ActionListener) diff);
         non.addActionListener((ActionListener) diff);
       };
       public static void main (String[] argv) {
         Frame r = new PlusouMoins();
         r.pack();
         r.setVisible(true);
         r.addWindowListener(new WindowCloser());
       }
}
```

6.2.7 Exercice 7

Enoncé

Soit l'IHM suivante (Figure 38). Cette interface contient un bouton au milieu étiqueté d'une valeur numérique, trois boutons sur la gauche étiquetés respectivement des valeurs 1, 2 et 3. Ces boutons servent à choisir une valeur à ajouter ou à enlever de la valeur du bouton du milieu en fonction des boutons de haut (+ ou -).

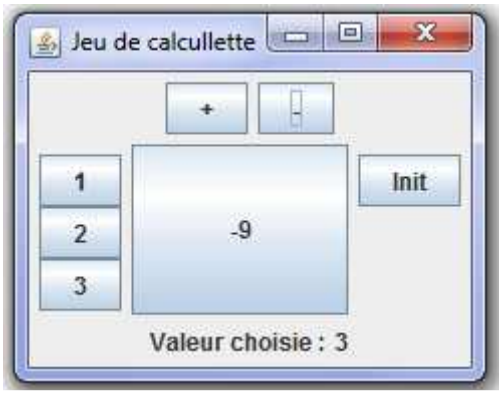


Figure 38 - Calculette 2

Solution

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

class MyFrame extends JFrame implements ActionListener{
     private JButton plus, moins;
     private JLabel valeur_choisie, valeurx;
     private JButton valeur1, valeur2, valeur3;
     private JButton initial;
     private JButton resultat;
     private JPanel pan1, pan2, pan3, pan4;
     public MyFrame(){
           setTitle("Jeu de calcullette");
        Container c = getContentPane();
        c.setBackground(Color.cyan);
        setSize(350, 300);
        c.setLayout(new BorderLayout());
        setLocationRelativeTo(this.getParent());
        plus = new JButton("+");
        moins = new JButton("-");
        pan1 = new JPanel();
        pan1.add(plus);
        pan1.add(moins);
        c.add(pan1, BorderLayout.NORTH);
        plus.addActionListener(this);
        moins.addActionListener(this);
        valeur_choisie=new JLabel("Valeur choisie :");
        valeurx =new JLabel("1");
        pan2=new JPanel();
        pan2.add(valeur_choisie);
        pan2.add(valeurx);
        c.add(pan2,"South");
        valeur1 = new JButton("1");
        valeur2 = new JButton("2");
        valeur3 = new JButton("3");
        pan3 = new JPanel();
        Box b = new Box(BoxLayout.Y_AXIS);
        b.add(valeur1);
        b.add(valeur2);
        b.add(valeur3);
```

```
        pan3.add(b);
        c.add(pan3,"West");
        valeur1.addActionListener(this);
        valeur2.addActionListener(this);
        valeur3.addActionListener(this);
        initial = new JButton("Init");
        pan4 = new JPanel();
        pan4.add(initial);
        c.add(pan4, BorderLayout.EAST);
        initial.addActionListener(this);
        resultat =new JButton("0");
        c.add(resultat, BorderLayout.CENTER);
     }
     public void actionPerformed(ActionEvent e) {
           if (e.getSource() == plus) {
                 int x = Integer.parseInt(resultat.getText()) +
                             Integer.parseInt(valeurx.getText());
                             resultat.setText(""+x);
           }

           if (e.getSource() == moins) {
                 int x = Integer.parseInt(resultat.getText()) -
                             Integer.parseInt(valeurx.getText());
                             resultat.setText(""+x);
           }
           if (e.getSource() == valeur1) valeurx.setText("1");
           if (e.getSource() == valeur2) valeurx.setText("2");
           if (e.getSource() == valeur3) valeurx.setText("3");
           if (e.getSource() == initial) resultat.setText("0");
     }
}
public class Jeu_de_calcul {
      public static void main(String [] args)
     {
            MyFrame f = new MyFrame();
         f.setVisible(true);
         f.setDefaultCloseOperation(3);
     }
}
```

6.2.8 Exercice 8

Enoncé

Soit l'interface graphique suivante (Figure 39). Cette IHM ressemble à celle de l'exercice 7 sauf qu'on inverse les boutons numériques et les boutons des opérations arithmétiques.

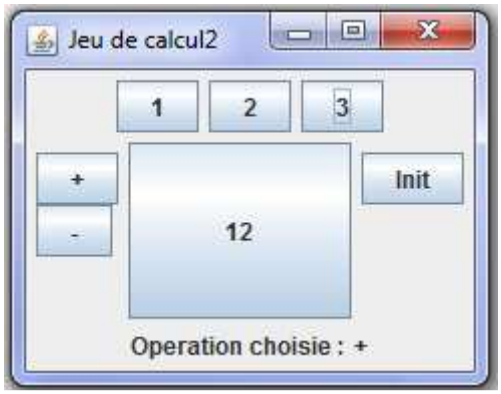


Figure 39 - Calculette 3

Solution

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

class MyFrame2 extends JFrame implements ActionListener{
     private JButton plus, moins;
     private JLabel Operation_choisie, Operationx;
     private JButton valeur1, valeur2, valeur3;
     private JButton initial;
     private JButton resultat;
     private JPanel pan1, pan2, pan3, pan4;
     public MyFrame2(){
           setTitle("Jeu de calcul2");
        Container c = getContentPane();
        c.setBackground(Color.cyan);
        setSize(450, 300);
        c.setLayout(new BorderLayout());
        setLocationRelativeTo(this.getParent());
        valeur1 = new JButton("1");
        valeur2 = new JButton("2");
        valeur3 = new JButton("3");
        pan1 = new JPanel();
        pan1.add(valeur1);
        pan1.add(valeur2);
        pan1.add(valeur3);
        c.add(pan1, BorderLayout.NORTH);
        valeur1.addActionListener(this);
        valeur2.addActionListener(this);
        valeur3.addActionListener(this);
        Operation_choisie=new JLabel("Operation choisie :");
        Operationx =new JLabel("+");
        pan2=new JPanel();
        pan2.add(Operation_choisie);
        pan2.add(Operationx);
        c.add(pan2,"South");
        plus = new JButton("+");
        moins = new JButton("-");
        pan3 = new JPanel();
        Box b = new Box(BoxLayout.Y_AXIS);
        b.add(plus);
```

```
        b.add(moins);
        pan3.add(b);
        c.add(pan3,"West");
        plus.addActionListener(this);
        moins.addActionListener(this);
        initial = new JButton("Init");
        pan4 = new JPanel();
        pan4.add(initial);
        c.add(pan4, BorderLayout.EAST);
        initial.addActionListener(this);
        resultat =new JButton("0");
        c.add(resultat, BorderLayout.CENTER);
     }
     public void actionPerformed(ActionEvent e) {
           int x ;
           if (e.getSource() == valeur1 || e.getSource() == valeur2 ||
e.getSource() == valeur3) {
                 if (Operationx.getText() == "+"){
                       x = Integer.parseInt(resultat.getText()) +

                             Integer.parseInt(((AbstractButton)
e.getSource()).getText());
                 }else {
                       x = Integer.parseInt(resultat.getText()) -

                             Integer.parseInt(((AbstractButton)
e.getSource()).getText());
                 }
                 resultat.setText(""+x);
           }
           if (e.getSource() == plus) Operationx.setText("+");
           if (e.getSource() == moins) Operationx.setText("-");
           if (e.getSource() == initial) resultat.setText("0");
     }
}
public class Jeu_de_calcul2 {
      public static void main(String [] args)
     {
            MyFrame2 f = new MyFrame2();
         f.setVisible(true);
         f.setDefaultCloseOperation(3);
     }
}
```

7 Références bibliographiques

1 http://docs.oracle.com/javase/tutorial/uiswing/

2 The Definitive Guide to Java Swing, John Zukowski, 2005 Apress

3 Support de cours de Jean Berstel