

PROJET JAVA

Mourad Kmimech

ESIEA

Paris INF3034

Projet Java

2

Remarques générales

- ❑ Le nombre d'étudiants est limité à 2 étudiants par réalisation,
- ❑ Un seul trinôme est autorisé si le nombre d'étudiants d'un groupe est impair
- ❑ Les programmes doivent être réalisés sous Linux en utilisant un éditeur de texte (gedit ou vim).
- ❑ Ne négligez pas la documentation de votre code
- ❑ Le dépôt de fichiers se fera via Moodle.
- ❑ Tout code récupéré sur internet et utilisé par un simple copier/coller causera l'attribution d'un zéro

Projet Java

3

Objectif:

- Le but de ce projet est de créer une application informatique en langage Java pour automatiser la gestion des réservations de supports dans une bibliothèque.
- On vous demande d'écrire un programme offrant les fonctionnalités suivantes :
 - ▣ Ajout d'un support (livre, magazine, support multimédia)
Consultation de l'état des supports
 - ▣ Réservation d'un support pour une durée bien déterminée en fonction de l'utilisateur
 - ▣ Annulation de réservations
 - ▣ Gestion des utilisateurs

Projet Java

4

- Un fichier .tar avec le nom des auteurs dans le nom de l'archive et contenant
 - Vos fichiers java organisés en packages
 - Un diagramme UML contenant les différentes classes utilisées et les relations entre elles. Bonus
 - Un fichier exécutable permettant de lancer le jeu automatiquement

Rappel sur les IHMs en Java

Sommaire

6

- Introduction
- Le composant JFrame
- Les gestionnaire de disposition
- La gestion des evenements
- La connexion à la BD

Introduction

7

- Dès la version Java 1.0, SUN introduit une bibliothèque de classe appelée **A**bstract **W**indow **T**oolkit (**AWT**) pour la programmation de base de l'interface graphique utilisateur (GUI).
- La manière dont **AWT** de base gère les éléments de l'interface utilisateur se fait par la délégation de leur création et de leur comportement à la boîte à outils native du GUI sur chaque plate-forme cible (Windows, Solaris, Macintosh, etc.).
- ⊖ Il est difficile d'écrire une bibliothèque graphique portable de haute qualité qui dépendait des éléments d'une interface utilisateur native (les menus, les barres de défilement et les champs de texte peuvent avoir des différences de comportement par rapport à la plate-forme cible).
- ⊖ Apparition de « bug » dans les interface AWT lors du passage d'un environnement à un autre.
- ⊖ certains environnements graphiques (comme X11/Motif) n'ont pas une large collection de composants d'interface utilisateur comme windows.

Introduction

8

- En 1996, Netscape a créé une bibliothèque de GUI dénommée **IFC** (Internet Foundation Classes).
- Le principe est que les éléments de l'interface utilisateur (boutons, les menus, etc.), étaient dessinés sur des **fenêtres** vierges.
- Ainsi, les éléments IFC de Netscape avaient le même aspect et se comportaient de la même manière, quelle que soit la plateforme.
- Sun a collaboré avec Netscape pour parfaire cette approche, avec pour résultat une bibliothèque d'interfaces utilisateur portant le nom de code "**Swing**".
- Elle représentait une extension à la version Java 1.1 et elle a été intégrée dans la bibliothèque standard de Java SE 1.2.

Introduction

9

- Cependant, les éléments d'interface Swing sont un peu plus lents à s'afficher sur l'écran que les composants lourds employés par AWT.
- En revanche, Swing:
 - ▣ propose un ensemble d'éléments d'interface plus étendu et plus pratique.
 - ▣ dépend peu de la plate-forme d'exécution.
 - ▣ procure une bonne expérience à l'utilisateur qui travaille sur plusieurs plates-formes.

Remarque :

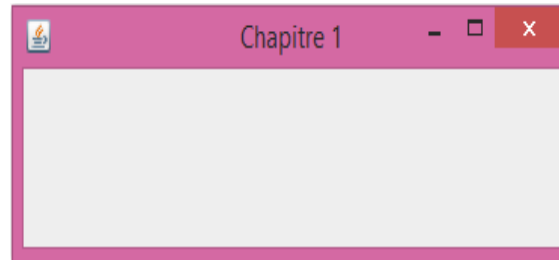
A part AWT et Swing, L'environnement de développement intégré Eclipse utilise une boîte à outils graphique appelée SWT qui est identique à AWT, faisant concorder des composants natifs sur diverses plates-formes.

(<http://www.eclipse.org/articles/>)

Création d'une fenêtre

10

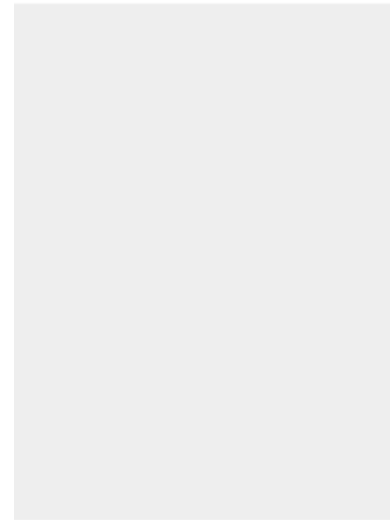
- Une IG SWING est un arbre qui part d'un objet du système. On distingue quatre objets racines :
 - JFrame : une fenêtre normale



Création d'une fenêtre

11

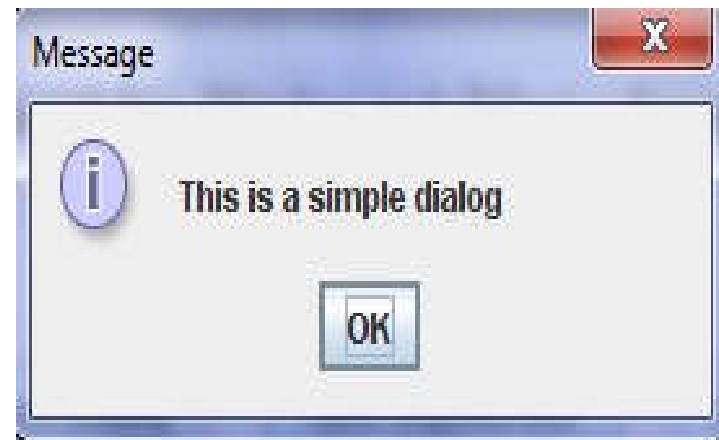
- Une IG SWING est un arbre qui part d'un objet du système (Heavyweight). On distingue quatre objets racines :
 - JFrame : une fenêtre normale
 - JWindow : fenêtre non décoré avec gestion partielle des évènements.



Création d'une fenêtre

12

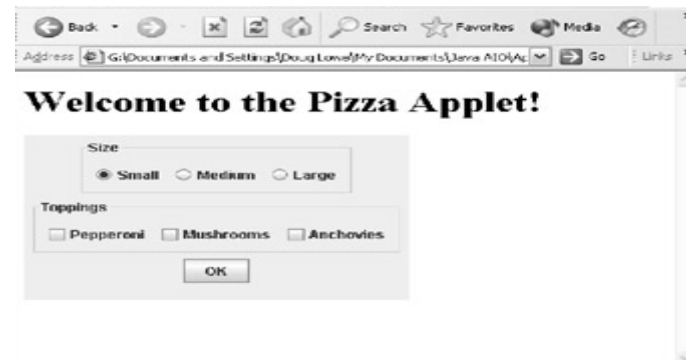
- Une IG SWING est un arbre qui part d'un objet du système (Heavyweight). On distingue quatre objets racines :
 - JFrame : une fenêtre normale
 - JWindow : fenêtre non décoré avec gestion partielle des évènements.
 - JDialog : une boîte de dialogue



Création d'une fenêtre

13

- Une IG SWING est un arbre qui part d'un objet du système (Heavyweight). On distingue quatre objets racines :
 - JFrame : une fenêtre normale
 - JWindow : fenêtre non décoré avec gestion partielle des évènements.
 - JDialog : une boîte de dialogue
 - JApplet : une classe java exécuter/véhiculer via une page web.



Création d'une fenêtre

14

- Une IG SWING est un arbre qui part d'un objet du système (Heavyweight). On distingue quatre objets racines :
 - JFrame : une fenêtre normale
 - JWindow : fenêtre non décoré avec gestion partielle des évènements.
 - JDialog : une boîte de dialogue
 - JApplet : une classe java exécuter/véhiculer via une page web.
- L'objet racine sert comme point de départ pour le dessin de notre interface et contient des objets gérer par java.

Création d'une fenêtre

15

```
import javax.swing.JFrame;
public class MaFenetre extends JFrame {
    public MaFenetre (){
        setTitle("Fenetre N°1");
        setSize(300,200);

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
    }
    public static void main(String[] args){
        new MaFenetre ();
    }
}
```



La barre de titre et les éléments, tels que les boutons à droite, sont dessinés par le système d'exploitation et non par la bibliothèque Swing. Résultat affichée sous windows 8

Création d'une fenêtre

16

```
import javax.swing.JFrame;
public class MaFenetre extends JFrame {
    public MaFenetre (){
        setTitle("Fenetre N°1");
        setSize(200,300);

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
    }
    public static void main(String[] args){
        new MaFenetre ();
    }
}
```

Les classes Swing se trouvent dans le package `javax.swing`.

Le terme `javax` désigne un package d'extension (pour le distinguer d'un package standard).

Pour des raisons historiques, les classes Swing constituent une extension mais elles font partie de chaque implémentation de Java SE depuis la version 1.2.

Création d'une fenêtre

17

```
import javax.swing.JFrame;
public class MaFenetre extends JFrame {
    public MaFenetre (){
        setTitle("Fenetre N°1");
        setSize(200,300);

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
    }
    public static void main(String[]args){
        new MaFenetre ();
    }
}
```

Par défaut, une fenêtre n'a pas de titre.

La méthode `setTitle` permet d'ajouter un titre à notre fenêtre dans la barre de titre.

Création d'une fenêtre

18

```
import javax.swing.JFrame;
public class MaFenetre extends JFrame {
    public MaFenetre (){
        setTitle("Fenetre N°1");
        setSize(300,200);

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
    }
    public static void main(String[] args){
        new MaFenetre ();
    }
}
```

Par défaut, une fenêtre a une taille de 0×0 pixel.

La méthode `setSize` permet d'affecter une taille à notre fenêtre 300×200 pixels.

Création d'une fenêtre

19

```
import javax.swing.JFrame;
public class MaFenetre extends JFrame {
    public MaFenetre (){
        setTitle("Fenetre N°1");
        setSize(300,200);

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
    }
    public static void main(String[]args){
        new MaFenetre ();
    } }
```

Création d'une fenêtre

20

```
import javax.swing.JFrame;
public class MaFenetre extends JFrame {
    public MaFenetre (){
        setTitle("Fenetre N°1");
        setSize(300,200);

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
    }
    public static void main(String[]args){
        new MaFenetre ();
    }
}
```

Par défaut, une fenêtre est invisible (paramètre égale à false).

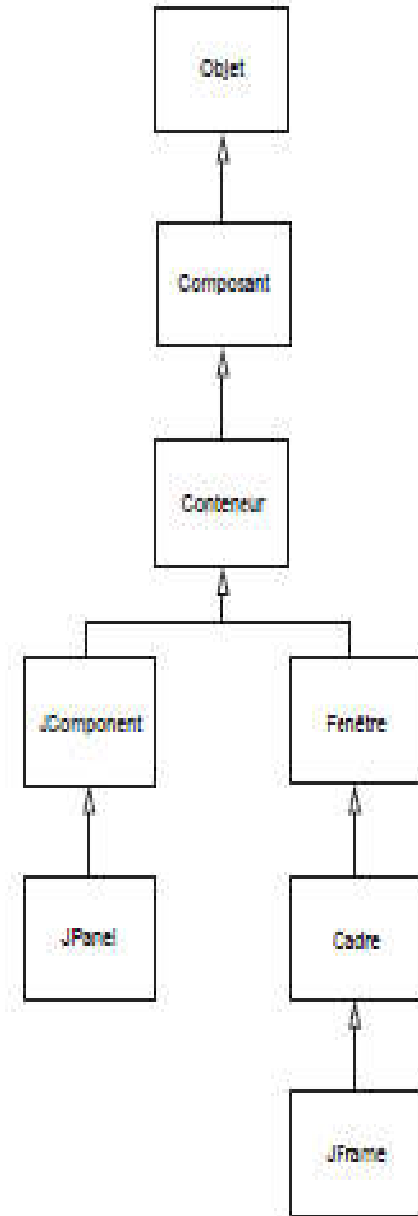
La méthode setVisible permet de modifier l'option de visibilité par défaut.

Elle remplace la méthode show à partir de la version java 1.5 Elle doit être utilisée après le remplissage de votre fenêtre par les composants graphiques.

Arborescence d'une JFrame

21

- Grace à l'héritage, la classe `JFrame` propose différentes méthodes pour la manipulation d'une fenêtre :
 - `setLocation` et `setBounds` définissent respectivement la position et la taille du cadre.
 - La méthode `setIconImage` ajouter une icône dans la barre de titre.
 - La méthode `setResizable` reçoit un paramètre booléen pour modifier ou non la taille d'un cadre.
 - `setTitle` / `getTitle` : permet d'ajouter ou de récupérer le texte d'un cadre.
 - `Pack` : permet de calculer automatiquement la taille idéale pour afficher tous les composants d'une fenetre



Arborescence d'une JFrame

22

□ **java.awt.Component 1.0**

- **boolean isVisible()**
- **void setVisible(boolean b)**
- **void setSize(int width, int height)**
- **void setLocation(int x, int y)**
- **void setBounds(int x, int y, int width, int height)**
- **Dimension getSize()**
- **void setSize(Dimension d)**

□ **java.awt.Window 1.0**

- **void toFront()** : Affiche cette fenêtre par-dessus toutes les autres.
- **void toBack()** : Place cette fenêtre au-dessous de la pile des fenêtres du bureau et réorganise les autres fenêtres visibles.
- **boolean isLocationByPlatform()**
- **void setLocationByPlatform(boolean b)** : Récupèrent ou définissent la propriété locationByPlatform. Lorsque la propriété est définie avant que cette fenêtre ne soit affichée, la plate-forme choisit un emplacement adéquat.

Arborescence d'une JFrame

23

□ **java.awt.Frame 1.0**

- **boolean isResizable()**
- **void setResizable(boolean b)**
- **String getTitle()**
- **void setTitle(String s)**
- **Image getIconImage()**
- **void setIconImage(Image image)**
- **boolean isUndecorated()**
- **void setUndecorated(boolean b):** Lorsque la propriété est définie, le cadre est affiché sans décorations (barre de titre ou un bouton de fermeture). Appeler avant de rendre visible la fenêtre.
- **int getExtendedState()**
- **void setExtendedState(int state):** Récupèrent ou définissent l'état étendu de la fenêtre. L'état est : `Frame.NORMAL` ou `Frame.ICONIFIED` ou `Frame.MAXIMIZED_HORIZ` ou `Frame.MAXIMIZED_VERT` ou `Frame.MAXIMIZED_BOTH`.

Ajouter des composants à une JFrame

24

- Après avoir créer la JFrame, on va insérer les composants graphiques comme suit :

Récupération de la couche contenu :

```
Container contentPane =  
frame.getContentPane();
```

Création du composant graphique :

```
Composant c = ...
```

Ajouter le composant a la couche contenu :

```
contentPane.add(c);
```

- Avant la version Java SE5.0, si on ajoute directement a la fenetre par la méthode add alors une exception sera déclenchée avec le message :

"Do not use JFrame.add(). Use JFrame.getContentPane().add() instead"

- Depuis Java SE 5.0, on peut utiliser l'appel ci-dessous sans faire appel au contenu :

```
frame.add(c);
```


Les conteneurs

25

- Les conteneurs sont des objets graphiques qui peuvent contenir d'autres objets graphiques, incluant éventuellement des conteneurs.
- Ils héritent de la classe Container.
- Un composant graphique doit toujours être incorporé dans un conteneur.
- On distingue deux types de conteneur:
 - Conteneur **lourd** : il peut contenir des conteneurs légers et/ou des composants graphiques.
 - JFrame
 - JWindow
 - JDialog
 - JApplet
 - Conteneur **léger** : il peut contenir d'autres conteneurs légers et/ou des composants graphiques.
 - JPanel
 - JScrollPane
 - JSplitPane
 - ...

La classe JPanel

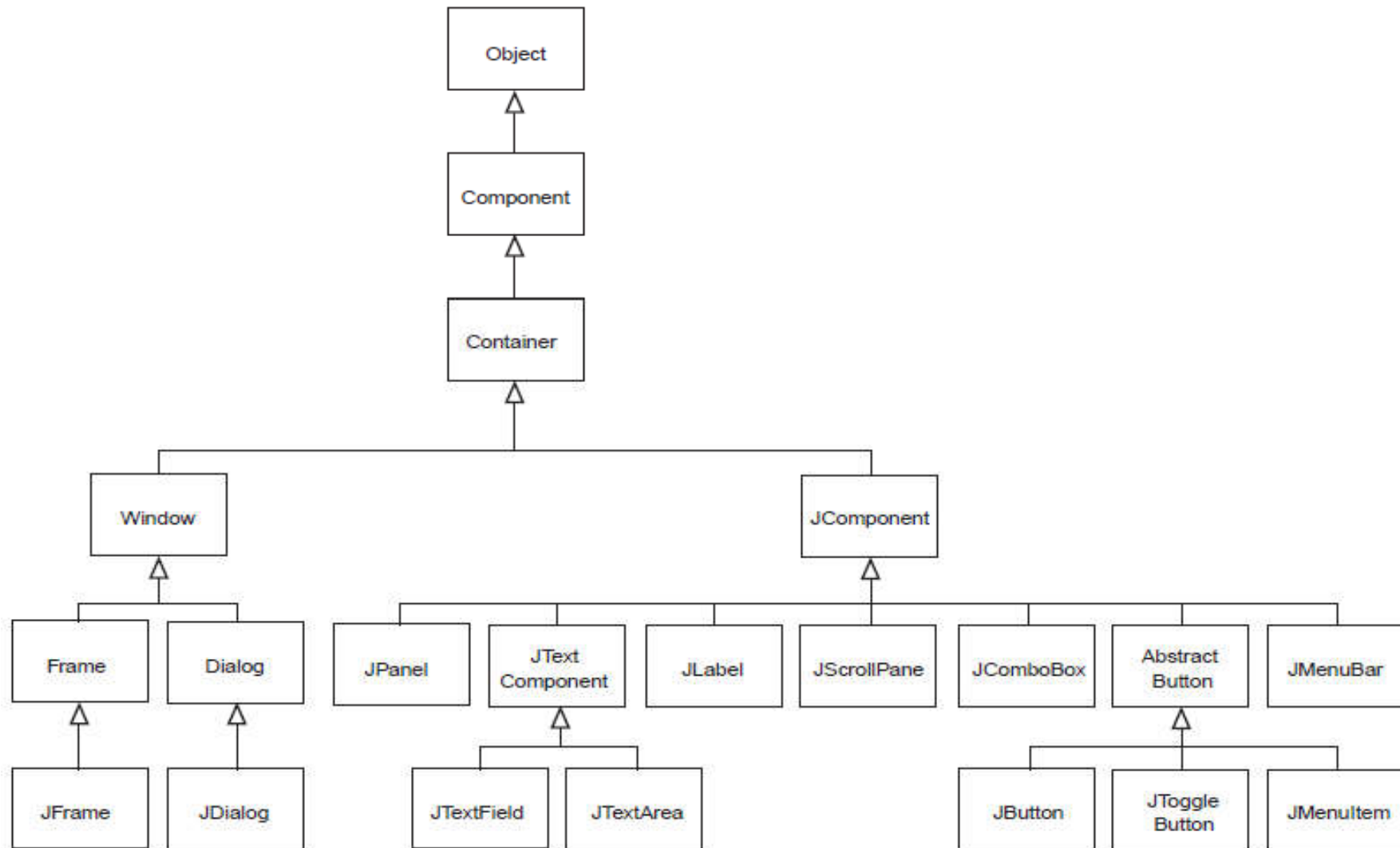
26

```
JPanel p=new JPanel();  
  
JButton b=new JButton("Test");  
  
p.add(b);
```

- C'est essentiellement un objet de rangement pour d'autres composants.
- La classe Panel possède plusieurs constructeurs parmi eux :
 - ▣ Panel() : panel avec un layout par défaut
 - ▣ Panel(LayoutManager l) : panel avec le layout passé en paramètre.
- Un panel seul ne peut être afficher, il doit être insérer dans un composant lourd.

Hiérarchie

27



Gestionnaire de mise en forme

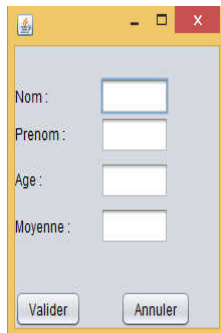
28

- En général, tous les composants sont placés dans des conteneurs. Ces conteneurs peuvent être placés à leur tour dans des conteneurs. Et en final, ils sont insérés dans le conteneur racine (Principal/lourd).
- Comment insérer les composants graphiques dans un conteneur (lourd/léger)?
- On distingue Deux techniques :
 - ▣ Placement sans layout, en indiquant la position et la taille de chaque composant
 - ▣ Placement suivant un layout.

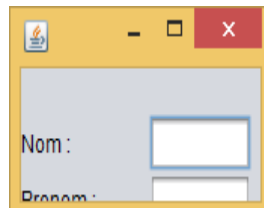
1^{ère} technique : Sans Layout

29

- Les composants sont placés moyennant des coordonnées (abscisse et ordonnée) et en indiquant la taille de chaque composant (largeur et hauteur) par les méthodes
 - `setBound(int x, int y, int l, int L)`
 - `setLocation(int x, int y)`

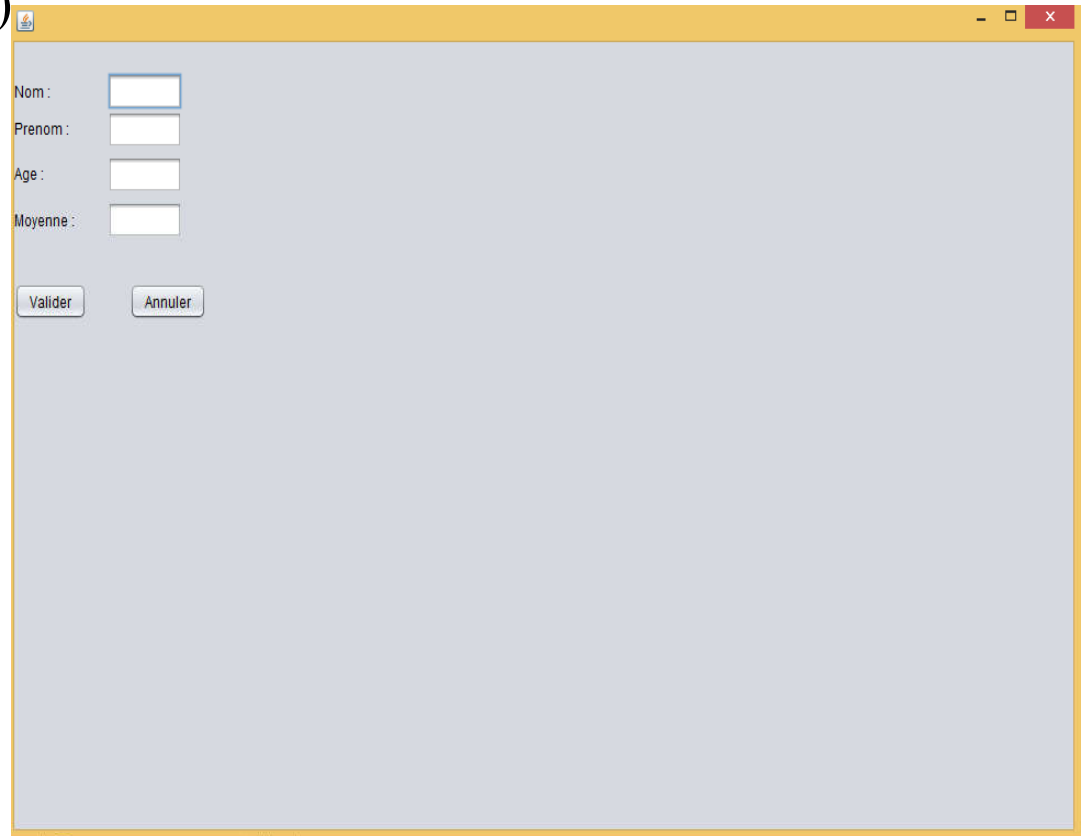
A small Java Swing window with a yellow title bar. It contains four text input fields labeled 'Nom:', 'Prenom:', 'Age:', and 'Moyenne:' stacked vertically. At the bottom are two buttons labeled 'Valider' and 'Annuler'.

**Taille
normale**

A small Java Swing window with a yellow title bar, similar to the first one but scaled down. It contains the same labels and input fields for 'Nom:', 'Prenom:', 'Age:', and 'Moyenne:', along with 'Valider' and 'Annuler' buttons.

**Taille
réduite**

**Taille plus
grande**

A large Java Swing window with a yellow title bar. It contains the same labels and input fields for 'Nom:', 'Prenom:', 'Age:', and 'Moyenne:', along with 'Valider' and 'Annuler' buttons, but scaled to fit the larger window.

1^{ère} technique : Sans Layout manager

30

- On n'écrit **JAMAIS** une application ou l'on fixe les composants, car:
 - ▣ la fenêtre peut être redimensionnée
 - ▣ l'espace occupé par un composant peut changer suivant le Look and Feel.

- Que faire alors?

2^{ème} technique : Avec un Layout manager

31

- un gestionnaire de disposition (layout manager) permet de spécifier comment les composants sont insérés dans le conteneur.
- Chaque layout manager implémente l'interface `java.awt.LayoutManager`.
- Il est possible d'utiliser plusieurs gestionnaires de mise en forme pour définir la présentation des composants
- Les layout manager ont 3 avantages :
 - l'aménagement des composants graphiques est délégué aux layout managers (il est inutile d'utiliser les coordonnées absolues)
 - en cas de redimensionnement de la fenêtre, les contrôles sont automatiquement agrandis ou réduits
 - ils permettent une indépendance vis à vis des plateformes.

2^{ème} technique : Avec un Layout manager

32

□ Le principe est assez simple :

1. Créer le layout manager

```
FlowLayout fl=new  
FlowLayout();
```

- **void setLayout(LayoutManager m)** : Configure le gestionnaire de mise en forme pour ce conteneur.

2. Affecter le layout manager au conteneur

```
JPanel p=new JPanel();  
p.setLayout ( fl );
```

- **Component add(Component c)**
- **Component add(Component c, Object constraints)** : Ajoutent un composant à ce conteneur et renvoient la référence du composant.

3. Ajouter les composants graphique au conteneur en respectant les contraintes du gestionnaire de disposition.

```
JButton b=new JButton("Test");  
p.add(b);
```


2^{ème} technique : Avec un Layout manager

33

- On distingue plusieurs gestionnaires de dispositions
 - ▣ FlowLayout
 - ▣ BorderLayout
 - ▣ GridLayout
 - ▣ BoxLayout
 - ▣ GridBagLayout
 - ▣ CardLayout
 - ▣

Le gestionnaire de disposition : FlowLayout

34

- La classe FlowLayout (mise en page flot) place les composants ligne par ligne de gauche à droite. Chaque ligne est complétée progressivement jusqu'à être remplie, puis passe à la suivante.
- Par défaut, chaque ligne est centrée.
- Le conteneur JPanel possède un FlowLayout par défaut.
- Constructeur de FlowLayout
 - ▣ **FlowLayout()**
 - ▣ **FlowLayout(int align)**
 - ▣ **FlowLayout(int align, int hgap, int vgap)**
- Avec :
 - ▣ **Align** : L'une des constantes d'alignement LEFT, CENTER ou RIGHT.
 - ▣ **Hgap** : L'intervalle horizontal en pixels à utiliser (les valeurs négatives provoquent un chevauchement).
 - ▣ **Vgap** : L'intervalle vertical en pixels à utiliser (les valeurs négatives provoquent un chevauchement).

Le gestionnaire de disposition : FlowLayout

35

```
public class TestFlowLayout extends JFrame{
```

Création des composants graphiques

```
    JButton b1=new JButton("Bouton 1");    JButton b2=new JButton("Bouton 2");  
    JButton b3=new JButton("Bouton 3");    JButton b4=new JButton("Bouton 4");  
    JButton b5=new JButton("Bouton 5");
```

```
public TestFlowLayout(){
```

```
    JPanel contentPane=new JPanel();
```

Création du layout manager

```
    FlowLayout fl=new FlowLayout();
```

```
    contentPane.setLayout(fl);
```

Affectation du layout manager au conteneur

```
    contentPane.add(b1);    contentPane.add(b2);
```

```
    contentPane.add(b3);    contentPane.add(b4);
```

```
    contentPane.add(b5);
```

**Ajout des composants
graphiques au conteneur avec
les contraintes du layout
manager**

```
    this.setContentPane(contentPane);
```

Modification du contentPane de la fenêtre

Le gestionnaire de disposition : FlowLayout

36

**Les opérations de base pour
chaque fenêtre**

```
this.pack();  
this.setTitle("Exemple FlowLayout");
```

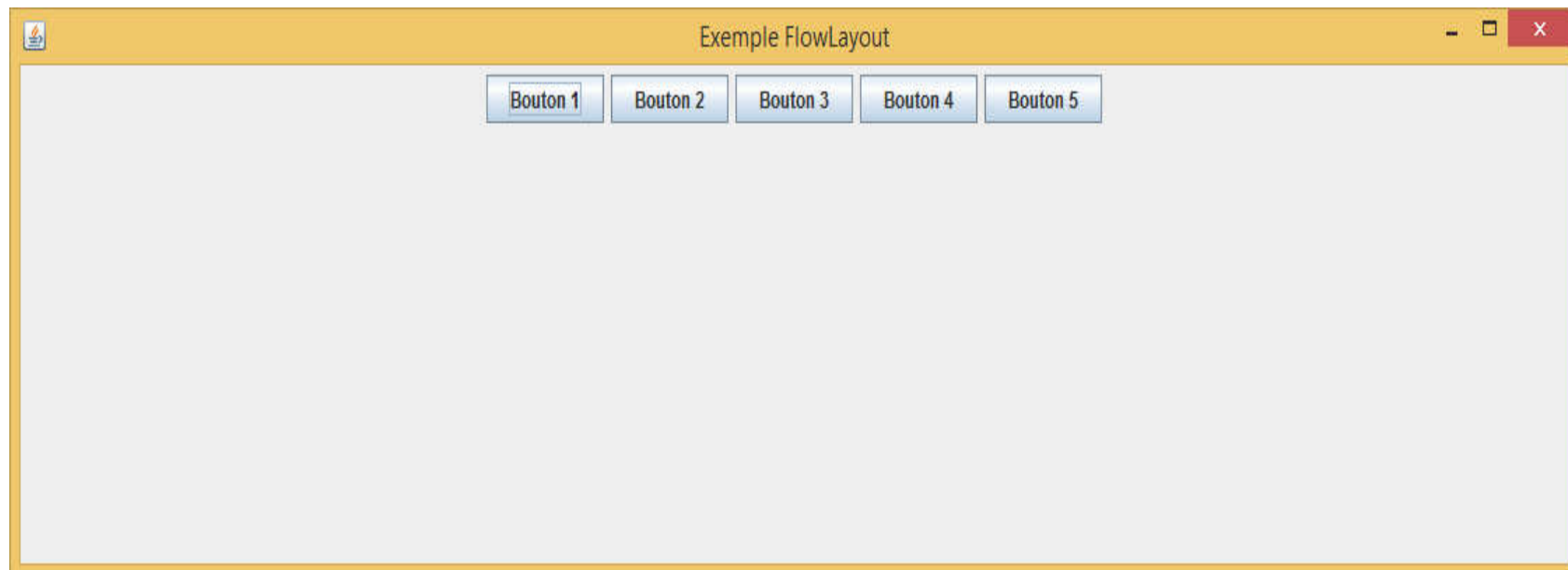
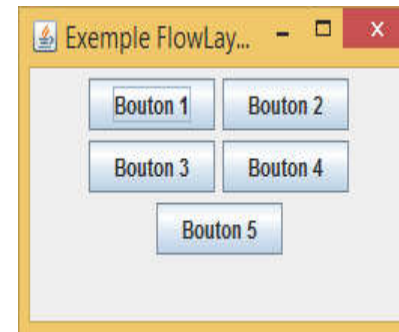
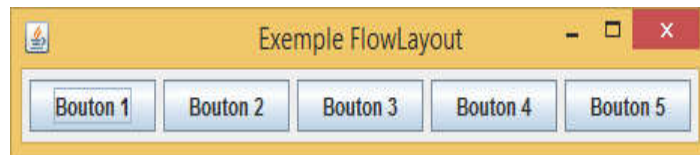
```
this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
this.setVisible(true);  
}
```

```
public static void main(String[] args){  
    new TestFlowLayout();  
}  
}
```

Appel à la fenêtre

Le gestionnaire de disposition : FlowLayout

37



Le gestionnaire de disposition : BorderLayout

38

- Avec ce Layout Manager, la disposition des composants est commandée par une mise en page en bordure qui découpe la surface en cinq zones : North, South, East, West, Center.
- On peut librement utiliser une ou plusieurs zones.
- BorderLayout consacre tout l'espace du conteneur aux composants.
- Le composant du milieu dispose de la place inutilisée par les autres composants.

- Les constructeurs sont :
 - BorderLayout()
 - BorderLayout(int hgap, int vgap)
 - hgap L'intervalle horizontal en pixels à utiliser (les valeurs négatives provoquent un chevauchement).
 - vgap L'intervalle vertical en pixels à utiliser (les valeurs négatives provoquent un chevauchement).

Le gestionnaire de disposition : BorderLayout

39

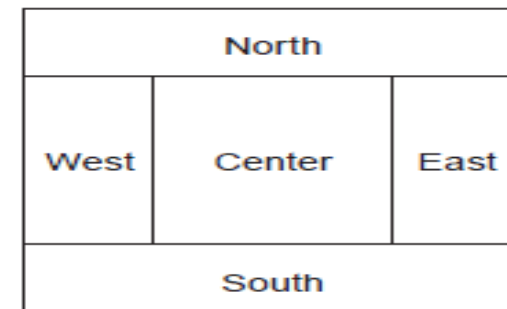
- Avec ce Layout Manager, la disposition des composants est commandée par une mise en page en bordure qui découpe la surface en cinq zones : North, South, East, West, Center.
- On peut librement utiliser une ou plusieurs zones.
- BorderLayout consacre tout l'espace du conteneur aux composants.
- Le composant du milieu dispose de la place inutilisée par les autres composants.

- Les constructeurs sont :

- BorderLayout()

- BorderLayout(int hgap, int vgap)

- hgap L'intervalle horizontal en pixels à utiliser (les valeurs négatives provoquent un chevauchement).
 - vgap L'intervalle vertical en pixels à utiliser (les valeurs négatives provoquent un chevauchement).



Le gestionnaire de disposition : BorderLayout

40

```
public class TestBorderLayout extends JFrame {  
    JButton b1=new JButton("Bouton 1");    JButton b2=new JButton("Bouton 2");  
    JButton b3=new JButton("Bouton 3");    JButton b4=new JButton("Bouton 4");  
    JButton b5=new JButton("Bouton 5");
```

```
    public TestBorderLayout() {  
        JPanel contentPane=new JPanel();  
        BorderLayout bl=new BorderLayout();  
        contentPane.setLayout(bl);
```

Création et affectation du layout manager

Ajout des composants graphiques au conteneur en spécifiant l'emplacement

```
        contentPane.add(b1, BorderLayout.NORTH);  
        contentPane.add(b2, BorderLayout.SOUTH);  
        contentPane.add(b3, BorderLayout.CENTER);  
        contentPane.add(b4, BorderLayout.EAST);  
        contentPane.add(b5, BorderLayout.WEST);
```

On peut utiliser aussi les constante chaine de caractere indiquant l'emplacement comme suit :

```
contentPane.add("South",b2);
```

La première lettre en majuscule :
South, North, East, West, Center

Le gestionnaire de disposition : BorderLayout

41

```
this.setContentPane(contentPane);
```

Modification du contentPane de la fenêtre

```
this.pack();
```

```
this.setTitle("Exemple BorderLayout");
```

```
this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
this.setVisible(true);
```

```
}
```

Les opérations de base pour chaque fenêtre

```
public static void main(String[] args){
```

```
    new TestBorderLayout();
```

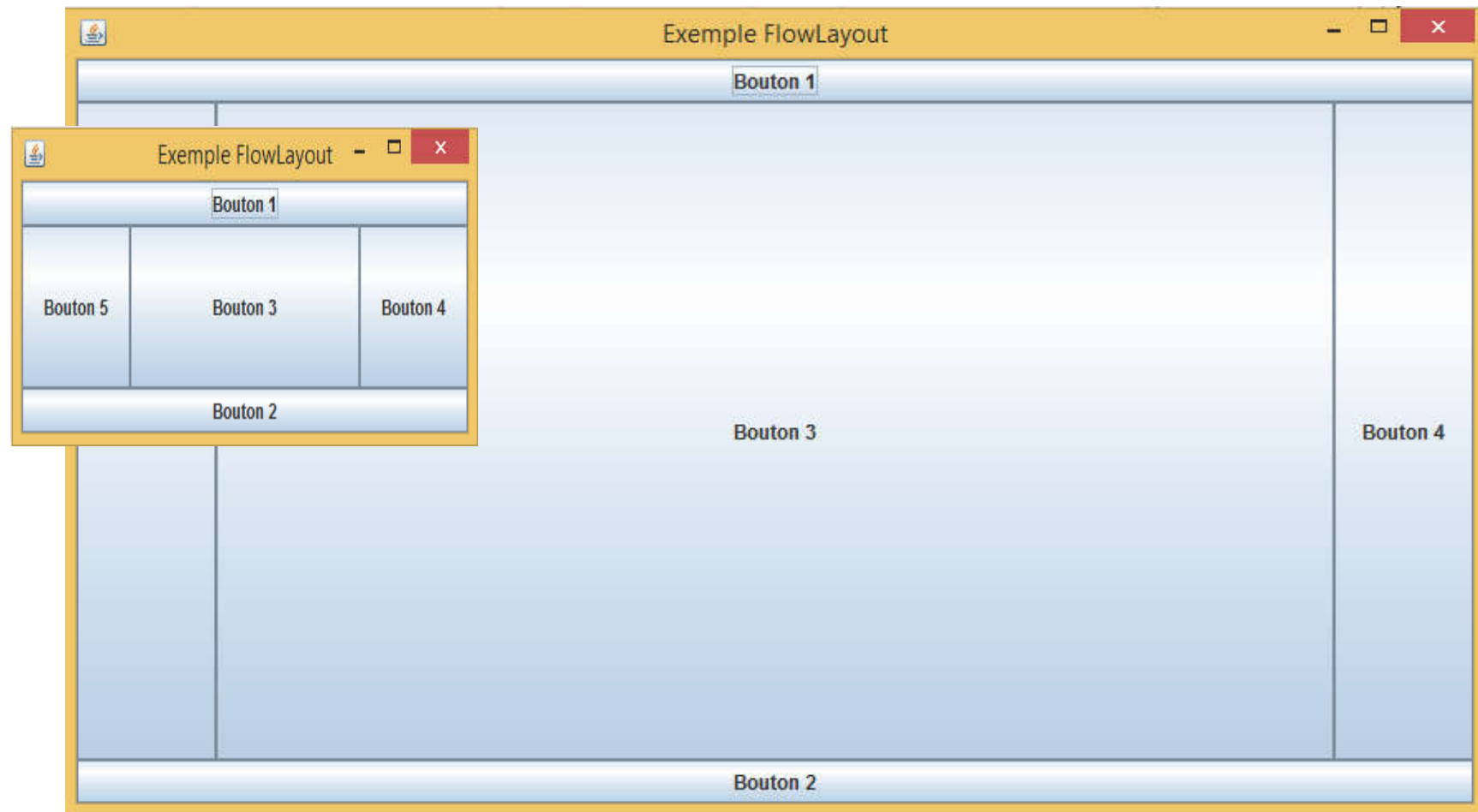
```
}
```

```
}
```

Appel à la fenêtre

Le gestionnaire de disposition : BorderLayout

42

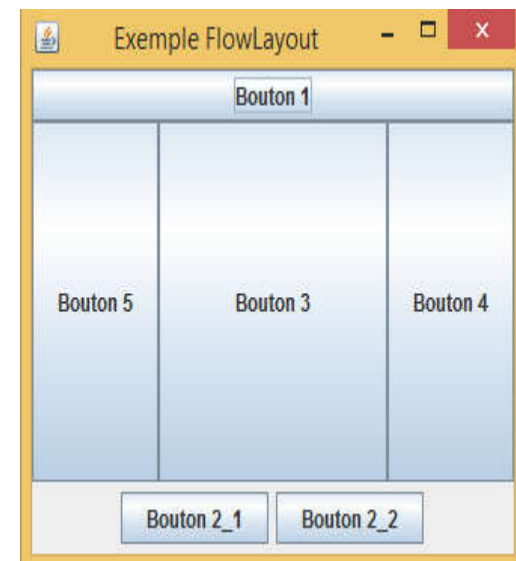


Le gestionnaire de disposition : BorderLayout

43

- Il est préférable d'utiliser des JPanel pour l'insertion de composants graphiques avec le layout en bordure.
- Si on apporte les modifications suivantes sur le code précédent on obtient ce qui suit :

```
JPanel p=new JPanel();  
p.add(b21);  
p.add(b22);  
  
contentPane.add(p, BorderLayout.SOUTH);
```



Le gestionnaire de disposition : GridLayout

44

- La disposition des grilles organise les composants, un peu à la manière des lignes et des colonnes d'un tableur.
- Tous les composants ont une taille *identique*.
- Lorsque vous redimensionnez la fenêtre, les composants s'agrandissent et diminuent tout en conservant des tailles identiques.
- Parmi les constructeurs on site :
 - **GridLayout**(int rows, int columns)
 - **GridLayout**(int rows, int columns, int hgap, int vgap)
- Avec
 - **rows** : Le nombre de lignes dans la grille
 - **columns** : Le nombre de colonnes dans la grille.
 - **hgap** : L'intervalle horizontal en pixels (les valeurs négatives provoquent un chevauchement).
 - **vgap** : L'intervalle vertical en pixels (les valeurs négatives provoquent un chevauchement).

Le gestionnaire de disposition : GridLay

45

```
public class TestGridLayout extends JFrame {  
    JButton b1=new JButton("1");   JButton b2=new JButton("2");  
    JButton b3=new JButton("3");   JButton b4=new JButton("4");  
    JButton b5=new JButton("5");   JButton b6=new JButton("6");  
    JButton b7=new JButton("7");   JButton b8=new JButton("8");   JButton b9=new  
    JButton("9");
```

Création des boutons

```
public TestGridLayout() {  
    JPanel contentPane=new JPanel();  
    GridLayout gl=new GridLayout(3,3);  
    contentPane.setLayout(gl);
```

Création d'une grille de 3 lignes et 3 colonnes

```
    contentPane.add(b1);   contentPane.add(b2);  
    contentPane.add(b3);   contentPane.add(b4);  
    contentPane.add(b5);   contentPane.add(b6);  
    contentPane.add(b7);   contentPane.add(b8);  
                           contentPane.add(b9);
```

**Insertion des boutons dans la grille.
L'ordre des boutons est celui de l'insertion.**

Le gestionnaire de disposition : GridLayout

46

```
this.setContentPane(contentPane);
```

Modification du contentPane de la fenêtre

```
this.pack();
```

```
this.setTitle("Exemple GridLayout");
```

```
this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
this.setVisible(true);
```

Les opérations de base pour chaque fenêtre

```
}
```

```
public static void main(String[] args){
```

```
    new TestGridLayout();
```

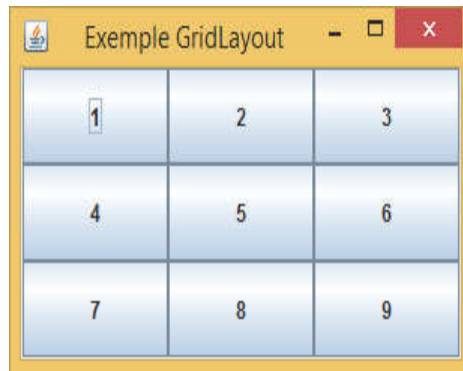
```
}
```

```
}
```

Appel à la fenêtre

Le gestionnaire de disposition : GridLayout

47



Le gestionnaire de disposition : BorderLayout

48

- affiche les composants en ligne ou en colonne, selon leur taille préférée.
- L'ordre d'insertion et l'ordre de placement dans le conteneur
- On doit créer le container AVANT le BorderLayout, car celui-ci en a besoin dans son constructeur
- Le constructeur du BorderLayout est : `public BorderLayout(Container target, int axis)`

Avec

- target : le conteneur cible à qui sera affecté le BorderLayout
- axis : L'alignement des composants. Il peut être l'une des valeurs suivantes :
 - **BorderLayout.X_AXIS** : alignement selon l'axe de X.
 - **BorderLayout.Y_AXIS** : alignement selon l'axe de Y.
 - **BorderLayout.LINE_AXIS** : alignement rangé en ligne.
 - **BorderLayout.PAGE_AXIS** : alignement rangé en colonne.

Le gestionnaire de disposition : BorderLayout

49

```
public class TestBoxLayout extends JFrame{  
    JButton b1=new JButton("Bouton 1");   JButton b2=new JButton("Bouton 2");  
    JButton b3=new JButton("Bouton treeeeeeeeeeeees long 3");  
    JButton b4=new JButton("Bouton treeeeeeeeeeeees long 4");  
    JButton b5=new JButton("B5");   JButton b6=new JButton("B6");  
  
    public TestBoxLayout(){  
        JPanel contentPane=new JPanel();  
  
        BorderLayout bl=new BorderLayout(contentPane, BorderLayout.Y_AXIS);  
        contentPane.setLayout(bl);  
  
        contentPane.add(b1);   contentPane.add(b2);  
        contentPane.add(b3);   contentPane.add(b4);  
        contentPane.add(b5);   contentPane.add(b6);  
    }  
}
```

Création des boutons

Création du
BoxLayout avec
un alignement
selon l'axe de y

Ajout des boutons

Le gestionnaire de disposition : BorderLayout

50

```
this.setContentPane(contentPane);
```

Modification du contentPane de la fenêtre

```
this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
this.setTitle("Exemple BorderLayout");
```

```
this.pack();
```

```
this.setVisible(true);
```

```
}
```

Les opérations de base pour chaque fenêtre

```
public static void main(String[] args){
```

```
    new TestBoxLayout();
```

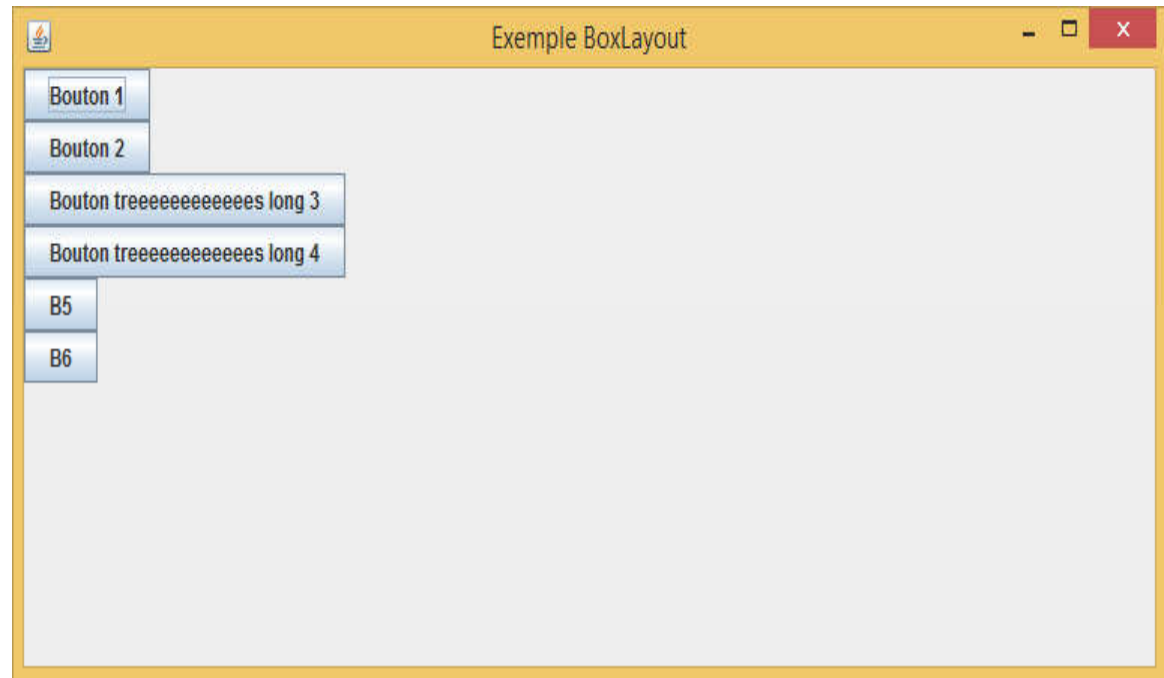
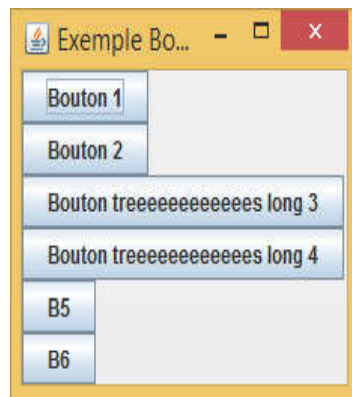
```
}
```

```
}
```

Appel à la fenêtre

Le gestionnaire de disposition : BorderLayout

51



Le gestionnaire de disposition : BorderLayout

52

- Dans un conteneur avec un BorderLayout, on peut ajouter:
 - des glues: ressorts invisibles qui se partagent l'espace disponible

**Création du panel pb1 et affectation
du BorderLayout selon l'axe des X**

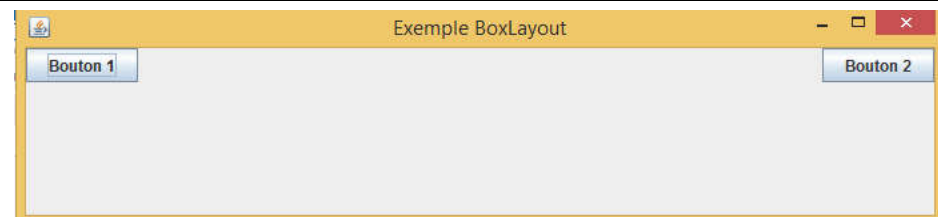
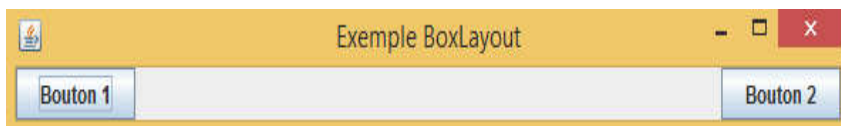
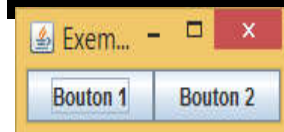
```
JPanel pb1=new JPanel();  
pb1.setLayout(new BorderLayout(pb1, BorderLayout.X_AXIS));
```

```
pb1.add(b1);  
pb1.add(Box.createHorizontalGlue());  
pb1.add(b2);
```

**Création d'une Glue (espace) entre les boutons
b1 et b2**

```
contentPane.setLayout(bl);  
contentPane.add(pb1);
```

Ajout du panel à la fenetre



Le gestionnaire de disposition : BorderLayout

53

- ❑ dans un conteneur avec un BorderLayout, on peut ajouter:
 - des glues: ressorts invisibles qui se partagent l'espace disponible
 - des struts: zones de largeur/hauteur fixe, qui se partagent en hauteur/largeur l'espace disponible

```
JPanel pb1=new JPanel();
```

```
pb1.setLayout(new BorderLayout(pb1, BorderLayout.X_AXIS));
```

Création du panel pb1 et affectation du BorderLayout selon l'axe des X

```
pb1.add(b1);
```

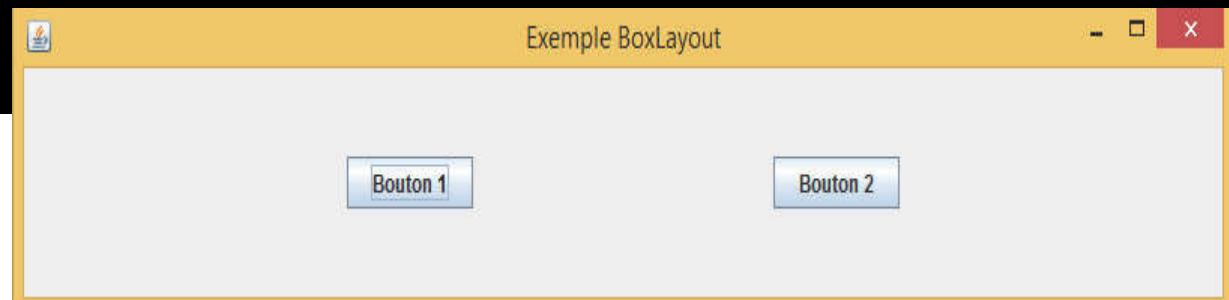
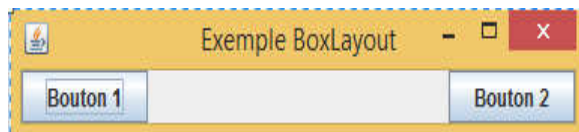
```
pb1.add(Box.createHorizontalStrut(200));
```

Création d'un Struts (espace horizontal fixe) entre les boutons b1 et b2

```
pb1.add(b2);
```

```
contentPane.setLayout(bl);
```

```
contentPane.add(pb1);
```



Le gestionnaire de disposition : BorderLayout

54

□ dans un conteneur avec un BorderLayout, on peut ajouter:

- des glues: ressorts invisibles qui se partagent l'espace disponible
- des struts: zones de largeur/hauteur fixe, qui se partagent en hauteur/largeur l'espace disponible
- des rigid area: zones de tailles fixe

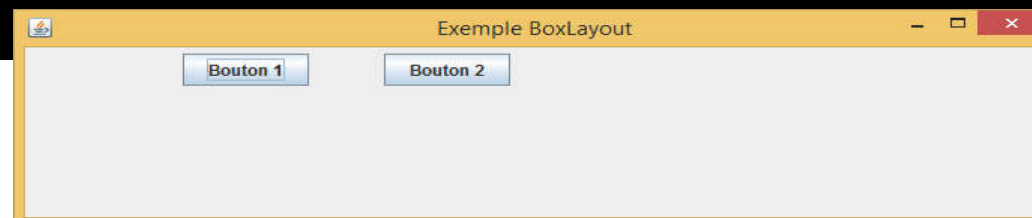
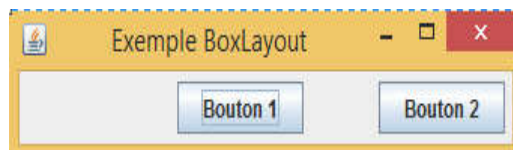
```
JPanel pb1=new JPanel();  
pb1.setLayout(new BorderLayout(pb1, BorderLayout.X_AXIS));
```

```
pb1.add(Box.createRigidArea(new Dimension(100, 10)));  
pb1.add(b1);  
pb1.add(Box.createRigidArea(new Dimension(50, 10)));  
pb1.add(b2);
```

```
contentPane.setLayout(bl);  
contentPane.add(pb1);
```

**Création du panel pb1 et affectation
du BorderLayout selon l'axe des X**

**Création de deux zone
d'espace fixe en
spécifiant la taille sous
forme d'un objet
dimension (longueur et
largeur) avant et après
b1.**

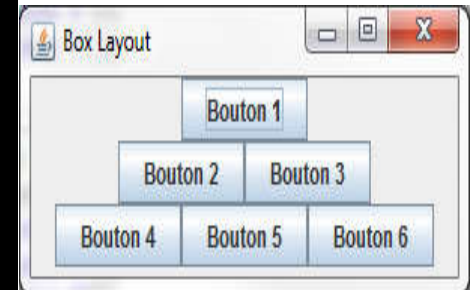


Le gestionnaire de disposition : BoxLay

55

- ❑ Voici un second exemple illustrant l'utilisation d'un BorderLayout.

```
JPanel b1 = new JPanel();  
//On définit le layout en lui indiquant qu'il travaillera en ligne  
b1.setLayout(new BorderLayout(b1, BorderLayout.LINE_AXIS));  
b1.add(new JButton("Bouton 1"));  
JPanel b2 = new JPanel();  
b2.setLayout(new BorderLayout(b2, BorderLayout.LINE_AXIS));  
b2.add(new JButton("Bouton 2"));b2.add(new JButton("Bouton 3"));  
JPanel b3 = new JPanel();  
b3.setLayout(new BorderLayout(b3, BorderLayout.LINE_AXIS));  
b3.add(new JButton("Bouton 4"));b3.add(new JButton("Bouton 5"));  
b3.add(new JButton("Bouton 6"));  
JPanel b4 = new JPanel();  
//On positionne maintenant ces trois lignes en colonne  
b4.setLayout(new BorderLayout(b4, BorderLayout.PAGE_AXIS));  
b4.add(b1); b4.add(b2); b4.add(b3);
```



Le gestionnaire de disposition : BorderLayout

56

- Il y a une façon plus simple de créer le même exemple en utilisant l'objet Box.

- //On crée un conteneur avec gestion horizontale** en BorderLayout.

```
Box b1 = Box.createHorizontalBox();
```

```
b1.add(new JButton("Bouton 1"));
```

```
Box b2 = Box.createHorizontalBox();
```

```
b2.add(new JButton("Bouton 2")); b2.add(new  
JButton("Bouton 3"));
```

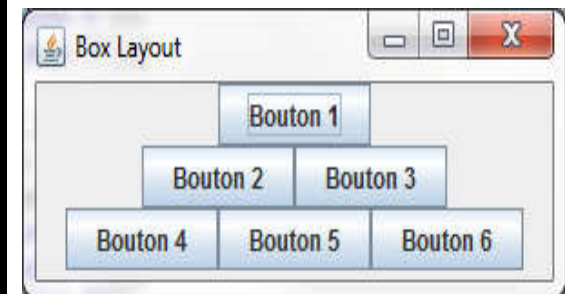
```
Box b3 = Box.createHorizontalBox();
```

```
b3.add(new JButton("Bouton 4")); b3.add(new  
JButton("Bouton 5"));  
b3.add(new JButton("Bouton 6"));
```

//On crée un conteneur avec gestion verticale

```
Box b4 = Box.createVerticalBox();
```

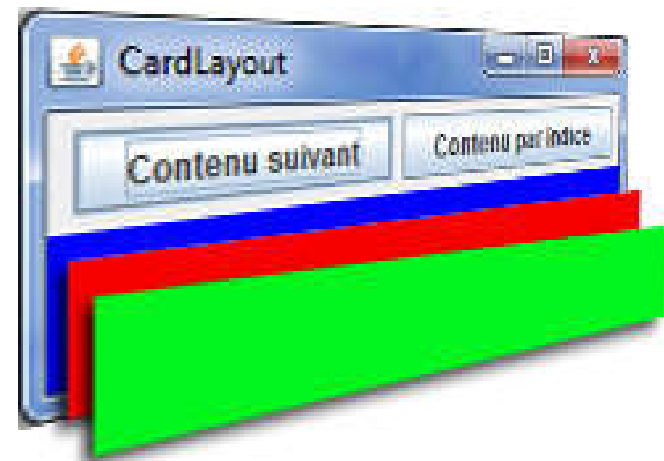
```
b4.add(b1); b4.add(b2); b4.add(b3);
```



Le gestionnaire de disposition : CardLayout

57

- ❑ Le gestionnaire de mise en page CardLayout permet de gérer vos conteneurs comme un tas de cartes (les uns sur les autres).
- ❑ On peut basculer d'un contenu à l'autre.
- ❑ Le principe est d'assigner des conteneurs au layout en leur donnant un nom afin de les retrouver plus facilement, permettant de passer de l'un à l'autre sans effort.
- ❑ Parmi les constructeurs on site :
 - ❑ **CardLayout()**
 - ❑ **CardLayout(int hgap, int vgap)**



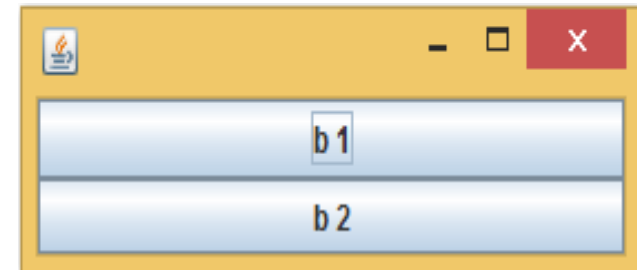
Le gestionnaire de disposition : CardLayout

58

```
public class TestCardLayout extends JFrame{
    JButton b1=new JButton("b 1");   JButton b2=new JButton("b 2");
    JButton b3=new JButton("b 3");   JButton b4=new JButton("b 4");
    public TestCardLayout(){
        CardLayout cl=new CardLayout();
        JPanel contentPane =new JPanel(cl);

        JPanel p=new JPanel(new GridLayout(2,1));
        p.add(b1);      p.add(b2);      contentPane.add("Onglet 1",p);
        JPanel p1=new JPanel();
        p1.add(b3);      p1.add(b4);      contentPane.add("Onglet 2",p1);
        contentPane.add("Texte", new JTextField(" Ma zone de Texte "));

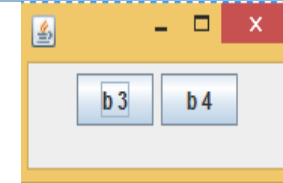
        cl.show(this.getContentPane(), "Onglet 1");
    }
}
```



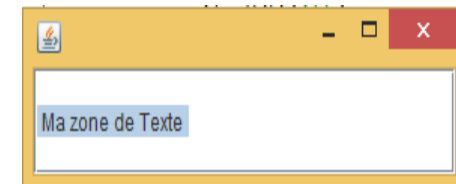
Le gestionnaire de disposition : CardLayout

59

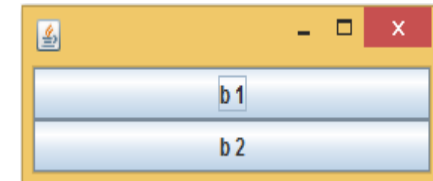
```
//En Changnte le panel à afficher on obtient :  
cl.show(this.getContentPane(), "Onglet 2");
```



```
//En Changnte le panel à afficher on obtient :  
cl.show(this.getContentPane(), "Texte");
```



```
//En mettant un nom d'onglet inexistant on obtient le  
premier panel:  
cl.show(this.getContentPane(), "zzzzzzzzzz");
```



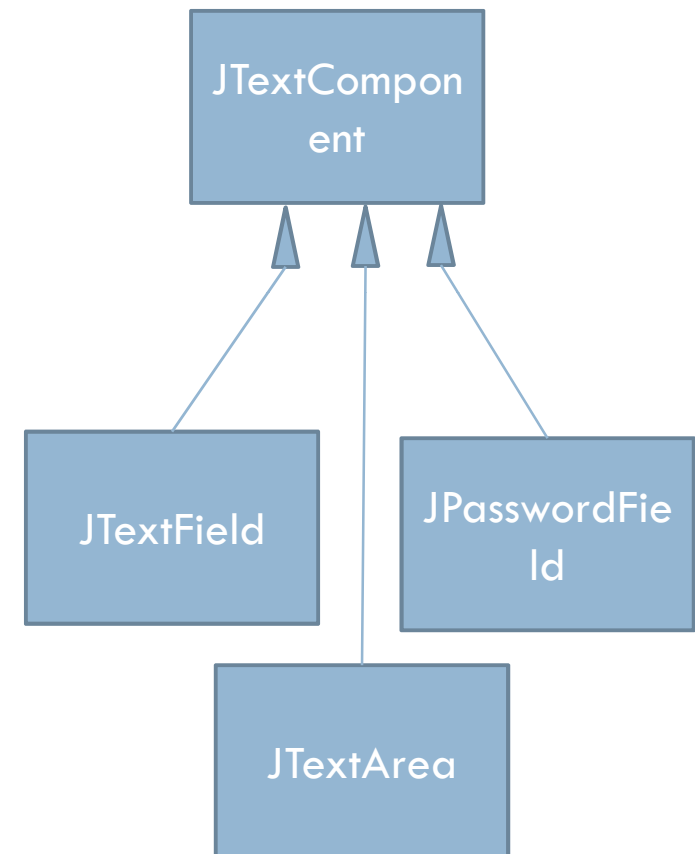
Les composants graphiques

60

□ Les zones de textes

JTextComponent

- *String getText()* : récupération du texte de la zone
- *void setText(String text)* : modification du contenu de la zone de texte.
- *Boolean isEditable()* : true si la zone est modifiable, false sinon
- *void setEditable(boolean b)* : modification de la propriété modifiable.



Les composants graphiques

61

TextField

- *TextField(int cols)* : Construit un `TextField` vide avec un nombre spécifié de colonnes.
- *TextField(String text, int cols)* : Construit un nouveau `TextField` avec une chaîne initiale et le nombre spécifié de colonnes.
- *int getColumns()*
- *void setColumns(int cols)* : Récupèrent ou définissent le nombre de colonnes à utiliser.
- **Méthode de la classe mère : `javax.swing.JComponent`**
 - *void revalidate()* : Entraîne un nouveau calcul de la position et de la taille d'un composant.
 - *void setFont(Font f)* : Définit la police de ce composant.
- **Méthode de la classe mère : `java.awt.Component`**
 - *void validate()* : Recalcule la position et la taille d'un composant. Si le composant est un conteneur, les positions et tailles de ses composants sont recalculées.
 - *Font getFont()* : Récupère la police de ce composant.

Les composants graphiques

62

JPasswordField

Champs de mot de passe (JPasswordField) : Le champ de mot de passe est un type spécial de champ de texte.

- `JPasswordField(String text, int columns)` : Construit un nouveau champ de mot de passe.
- `void setEchoChar(char echo)` : Définit le caractère d'écho pour le champ de mot de passe. Un certain style d'interface peut proposer son propre choix de caractères d'écho. La valeur 0 rétablit le caractère d'écho utilisé par défaut.
- `char[] getPassword()` : Renvoie le texte contenu dans le champ de mot de passe. Pour renforcer la sécurité, vous devez écraser le contenu du tableau renvoyé après utilisation. Le mot de passe n'est pas retourné en tant que `String`, car une chaîne resterait dans la machine virtuelle jusqu'à ce qu'elle soit éliminée par le processus de nettoyage de mémoire (le ramasse-miettes ou *garbage collector*).

Les composants graphiques

63

JTextArea

- Parfois, vous avez besoin de recueillir une entrée d'utilisateur d'une longueur supérieure à une ligne.
 - `JTextArea()` Construisent une nouvelle zone de texte.
 - `JTextArea(int rows, int cols)`
 - `JTextArea(String text, int rows, int cols)`

 - `void setColumns(int cols)` : Indique à la zone de texte le nombre de colonnes préféré à utiliser.
 - `void setRows(int rows)` : Indique à la zone de texte le nombre de lignes préféré à utiliser.

 - `void setLineWrap(boolean wrap)` : Active ou désactive le retour automatique à la ligne.
 - `void setWrapStyleWord(boolean word)` : Si le paramètre `word` vaut `true`, les lignes longues sont scindées au niveau des fins de mot. Si la valeur est `false`, elles sont divisées sans tenir compte des fins de mot.

 - `void append(String newText)` : Ajoute le texte spécifié à la fin du texte déjà présent dans la zone de texte.
 - `void setTabSize(int c)` : Définit les marques de tabulation toutes les `c` colonnes. Notez que les tabulations ne sont pas converties en espaces, mais provoquent l'alignement avec la marque suivante.

Les composants graphiques

64

Étiquettes et composants d'étiquetage (Jlabel)

Construisent une étiquette. Avec :

- *JLabel(String text)* • text Le texte de l'étiquette.
 - *JLabel(Icon icon)* • icon L'icône de l'étiquette.
 - *JLabel(String text, int align)* • align L'une des constantes
 - *JLabel(String text, Icon icon, int align)* SwingConstants LEFT (par défaut), CENTER ou RIGHT.
-
- *String getText()* Récupèrent ou définissent le texte de cette étiquette
 - *void setText(String text)*
-
- *Icon getIcon()* Récupèrent ou définissent l'icône de cette étiquette.
 - *void setIcon(Icon icon)*

Les composants graphiques

65

Volets de défilement

- Dans Swing, une zone de texte ne dispose pas de barres de défilement. Si vous souhaitez en ajouter, prévoyez la zone de texte dans un panneau avec barres de défilement `JScrollPane`
- Exemple :
 - `textArea = new JTextArea(8, 40);`
 - `JScrollPane scrollPane = new JScrollPane(textArea);`
- Des barres de défilement apparaissent automatiquement si le texte entré dépasse la zone d'affichage ; elles disparaissent si, lors d'une suppression, le texte restant tient dans la zone de texte.
- ***JScrollPane(Component c)*** : Crée un panneau avec défilement qui affiche le contenu du composant spécifié. Les barres de défilement apparaissent automatiquement lorsque le contenu du composant est plus grand que la vue.

Les composants graphiques

66

□ Composants du choix : Cases à cocher (JCheckBox)

➤ *JCheckBox(String label)*

➤ *JCheckBox(String label, Icon icon)*

Construisent une case à cocher qui est, à l'origine, non sélectionnée.

➤ *JCheckBox(String label, boolean state)* : Construit une case à cocher avec l'étiquette donnée et l'état initial.

➤ *boolean isSelected()*

Définissent ou récupèrent l'état de sélection de la case à cocher.

➤ *void setSelected(boolean state)*



Les composants graphiques

67

Composants du choix : Bouton Radio (JRadioButton)

- *JRadioButton(String label, Icon icon)* : Construit un bouton radio, non sélectionné au départ.
- *JRadioButton(String label, boolean state)* : Construit un bouton radio avec le libellé donné et l'état initial.
- Les boutons radio doivent être mis en groupe **ButtonGroup**
 - ▣ *void add(AbstractButton b)* : Ajoute le bouton au groupe.
 - ▣ *ButtonModel getSelection()* : Renvoie le modèle du bouton sélectionné.



☐ Small ☐ Medium ☐ Large ☒ Extra large

Les composants graphiques

68

Listes déroulantes : JComboBox

- boolean isEditable()
- void setEditable(boolean b)

Récupèrent ou définissent la propriété editable de cette liste déroulante.

- void addItem(Object item) : Ajoute un élément dans la liste.
- void insertItemAt(Object item, int index) : Insère un élément dans la liste à la position donnée par l'indice.
- void removeItem(Object item) : Supprime un élément de la liste.
- void removeItemAt(int index) : Supprime l'élément dans la liste à la position donnée par l'indice.
- void removeAllItems() : Supprime tous les éléments de la liste.
- Object getSelectedItem() : Renvoie l'élément actuellement sélectionné.

La gestion des événements

69

- Tout environnement d'exploitation qui gère les interfaces utilisateur graphiques doit constamment détecter les **événements** tels que la *pression* sur une touche du clavier ou sur un bouton de la souris.
- L'environnement d'exploitation, en réponse à cet événement, en informe alors les programmes en cours d'exécution.
- Chaque programme détermine ensuite s'il doit répondre à ces événements.
- Comment?

La gestion des événements

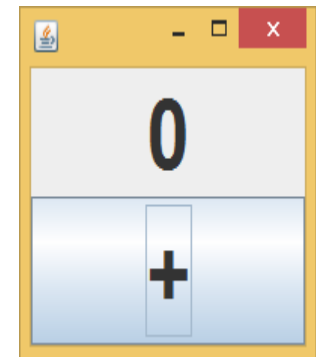
70

- voici en gros comment les événements sont gérés avec AWT/SWING :
 - Un objet écouteur est une instance d'une classe qui implémente une **interface** spéciale appelée *interface écouteur* (*listener interface*).
 - Une source d'événement est un objet qui peut recenser des objets écouteurs et leur envoyer des objets événements.
 - Lorsqu'un événement se produit, la source d'événement envoie l'objet événement à tous les écouteurs recensés.
 - Les objets écouteurs utilisent alors l'information contenue dans l'objet événement pour déterminer leur réponse.

La gestion des événements

71

- L'appui sur le bouton intitulé + permet d'incrémenter la valeur du compteur comme indiqué dans la figure.
- Pour réaliser la gestion des événements sur cette interface on procède comme suit :
 - 1 - Choisir le Type de l'évènement
Dans notre cas : *ActionEvent* (clique sur le bouton, appui sur la touche entrée,...)
 - 2 - Déterminer l'écouteur associé à l'évènement.
Dans notre cas : *ActionListener*
 - 3 - Ajouter l'écouteur sur le bouton.
Dans notre cas appel à la méthode
 - 4 *addActionListener(notre_écouteur_ActionListener)* de notre bouton.
 - Programmer la ou les méthodes de l'écouteur
Dans notre cas : *actionPerformed(ActionEvent e)*

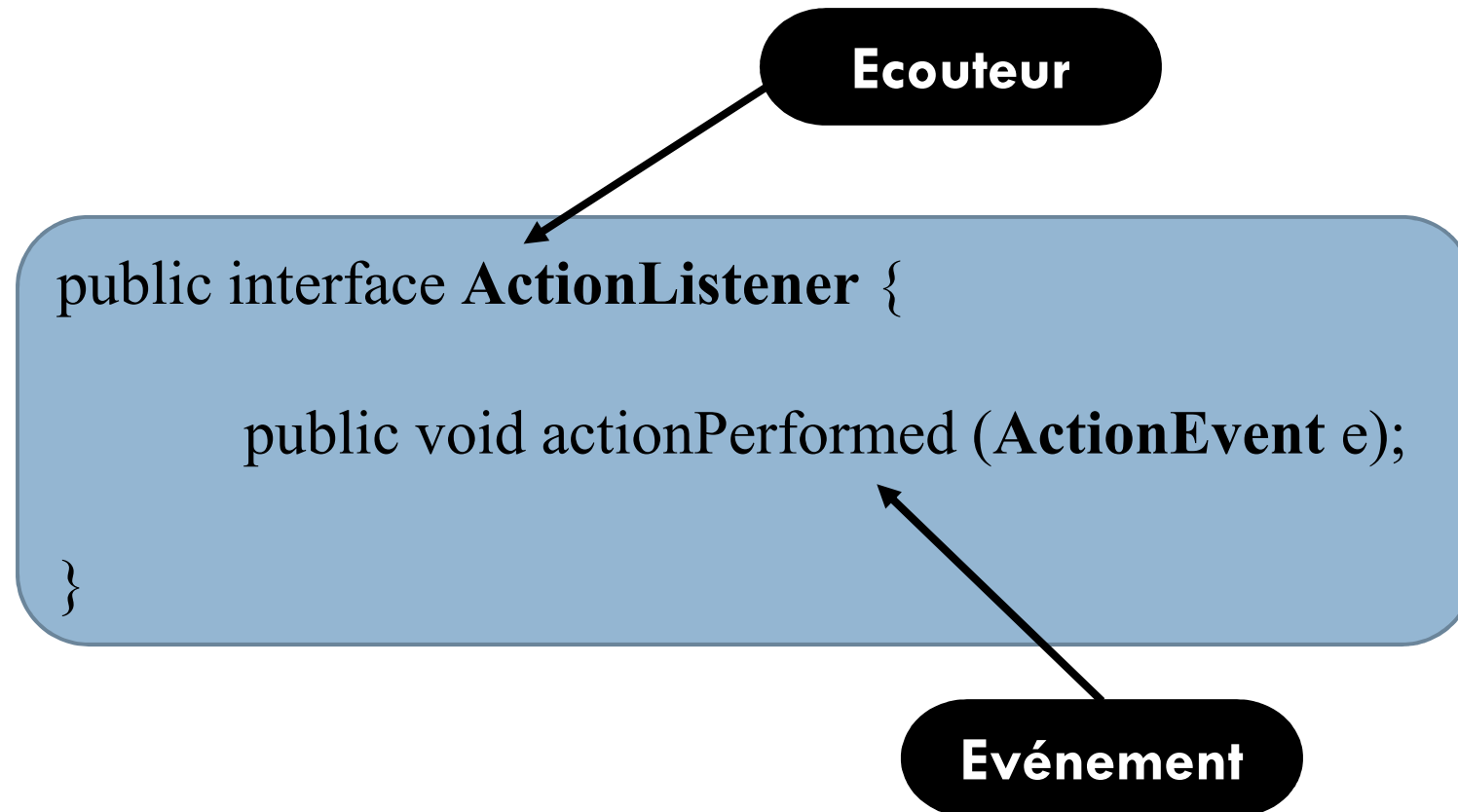


La gestion des événements

72

1 et 2

- L'interface d'écoute est la suivante :



La gestion des événements : exemple I

73

```
public class Ex1 extends JFrame implements  
ActionListener{
```

**La classe implémente
l'interface d'écoute**

```
JLabel lres=new JLabel("0");  
JButton b1=new JButton("+");
```

**Dessin de
l'interface**

```
public Ex1(){
```

```
lres.setHorizontalAlignment(JLabel.CENTER);  
lres.setFont(new Font("", 1, 50));  
b1.setFont(new Font("", 1, 50));  
this.getContentPane().add("North",lres);  
this.getContentPane().add("South",b1);
```

La gestion des événements : exemple I

74

3

```
b1.addActionListener(this);
```

Ajout de l'écouteur sur le bouton
Le paramètre est l'objet écouteur

```
this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    this.pack();  
    this.setVisible(true);  
}
```

4

```
public void actionPerformed(ActionEvent e) {  
    int x=Integer.parseInt(lres.getText());  
    x++;  
    lres.setText(x+"");  
}}
```

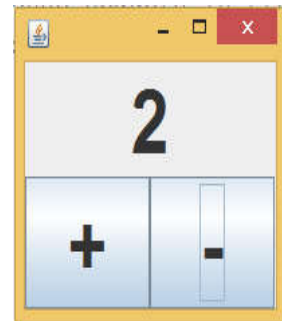
implémentation de la
méthode de l'écouteur

Incrémentation du
compteur et mise à jour de
l'affichage

La gestion des événements : exemple II

75

- On voudrait maintenant créer une seconde interface.
- L'appui sur le bouton + permet d'incrémenter le compteur et l'appui sur le bouton - permet de décrémenter le compteur.
- Un problème se pose : comment différencier entre le bouton + et - ?



- À partir de l'évènement, on doit pouvoir déterminer la source.

Utilisation de la méthode

getSource

à partir de l'évènement.

La gestion des événements : exemple II

76

```
public void actionPerformed(ActionEvent e) {  
    JButton src=(JButton)e.getSource();  
  
    int x=Integer.parseInt(lres.getText());  
    if(src==b1) x++;  
    else x--;  
  
    lres.setText(x+"");  
  
    if(x==0) b2.setEnabled(false);  
    else if (x==10) b1.setEnabled(false);  
    else  
    { b1.setEnabled(true); b2.setEnabled(true); }  
}
```

Utilisation de la méthode getSource pour déterminer le bouton sur lequel on a appuyer.

Base de données et Interface graphique

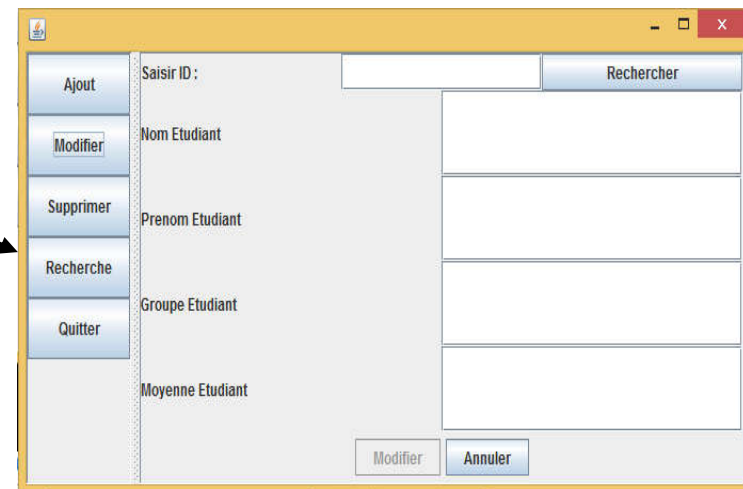
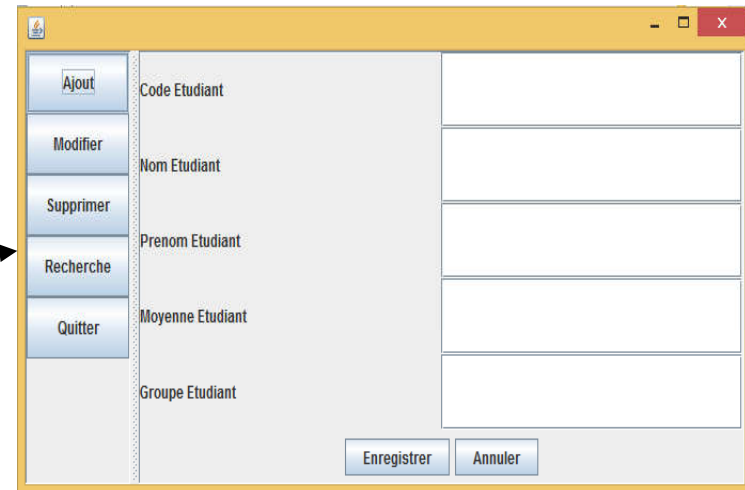
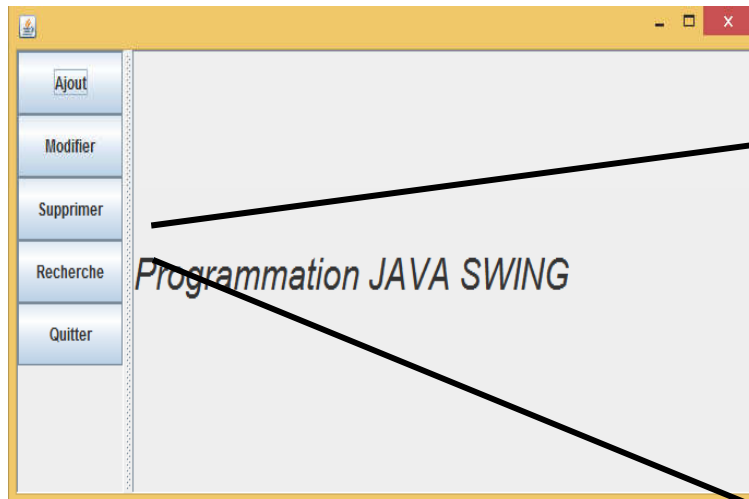
77

- La connexion à la base de données se fait moyennant le même principe que celui énoncé dans le chapitre intitulé *Connexion à la BD*.
- A fin de réaliser l'interface de la figure suivante on va procéder à la définition des classes suivantes :
 1. Une classe permettant de faire le dessin de l'interface graphique
 2. Une classe générant un objet connexion unique.
 3. Une ou des classes manipulant des objets de l'application
 4. Une classe permettant de manipuler les objets de l'application avec les tables de la BD.

Exemple

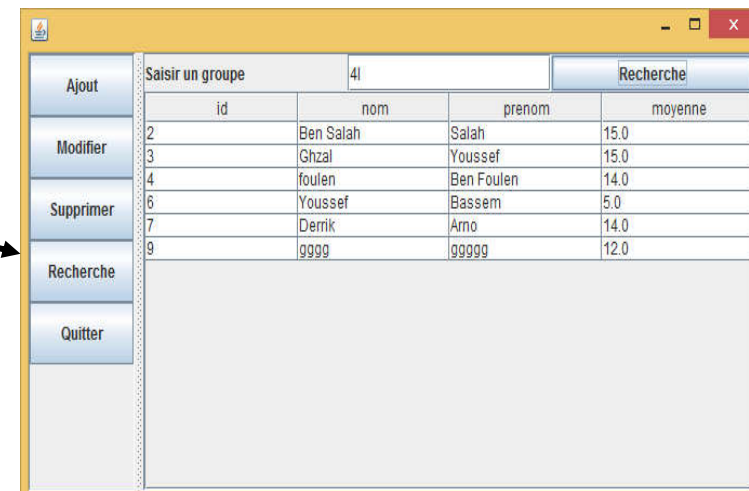
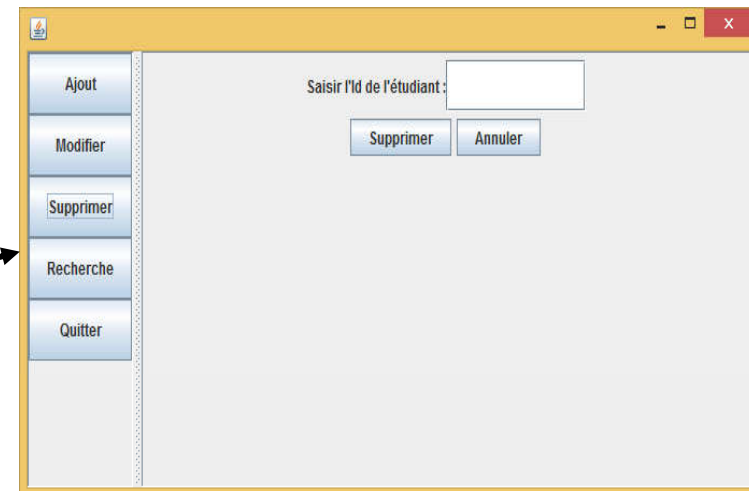
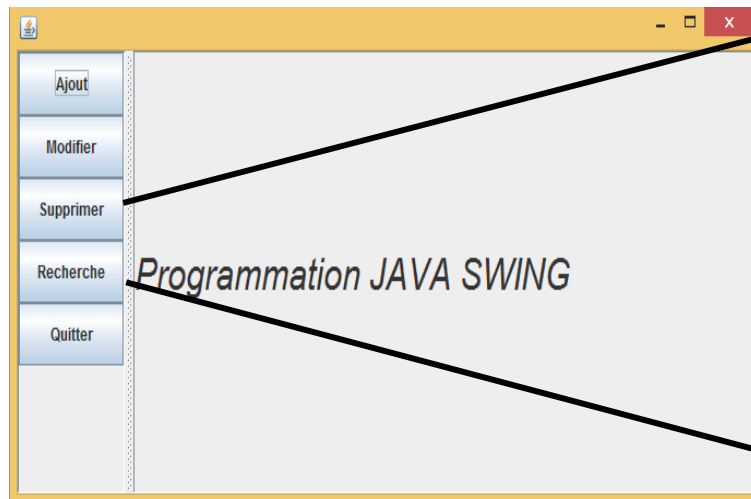
78

- On voudrait réaliser les interfaces suivantes :



Exemple

79

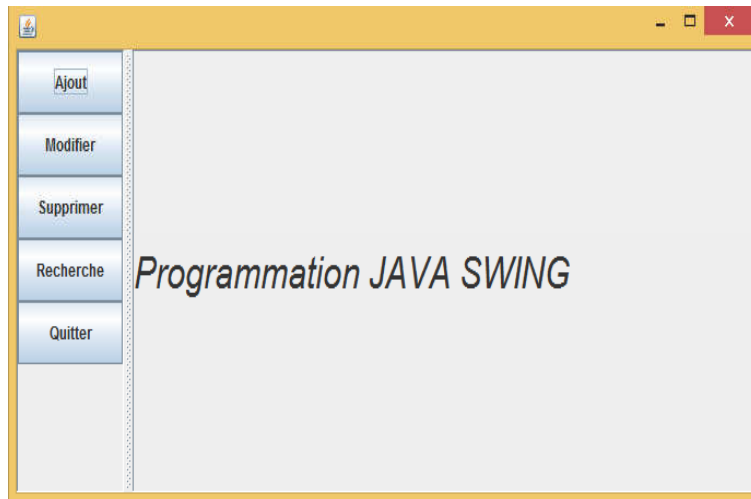


Exemple

80

□ Pour réaliser ces interfaces on procède comme suit :

- Dessiner les interfaces.
- Créer le singleton pour la création de l'interface.
- Créer les objets métiers
- Créer les classes DAO
- Programmer les écouteurs.



Références

81

- Livre

- ▣ Au cœur de java 6

- Site web

- ▣ http://www.jmdoudoux.fr/accueil_java.htm
 - ▣ <http://docs.oracle.com/javase/tutorial/uiswing/>