Introduction à la programmation événementielle

Module : Programmation Evénementielle

Année : 2008 / 2009

Introduction

- La quasi-totalité des programmes informatiques nécessitent
  - l'affichage de questions posées à l'utilisateur
  - l'entrée de données par l'utilisateur au moment de l'exécution
  - l'affichage d'une partie des résultats obtenus par le traitement informatique
- Cet échange d'informations peut s'effectuer avec une interface utilisateur (UI en anglais) en mode texte (ou console) ou en mode graphique

Interface graphique

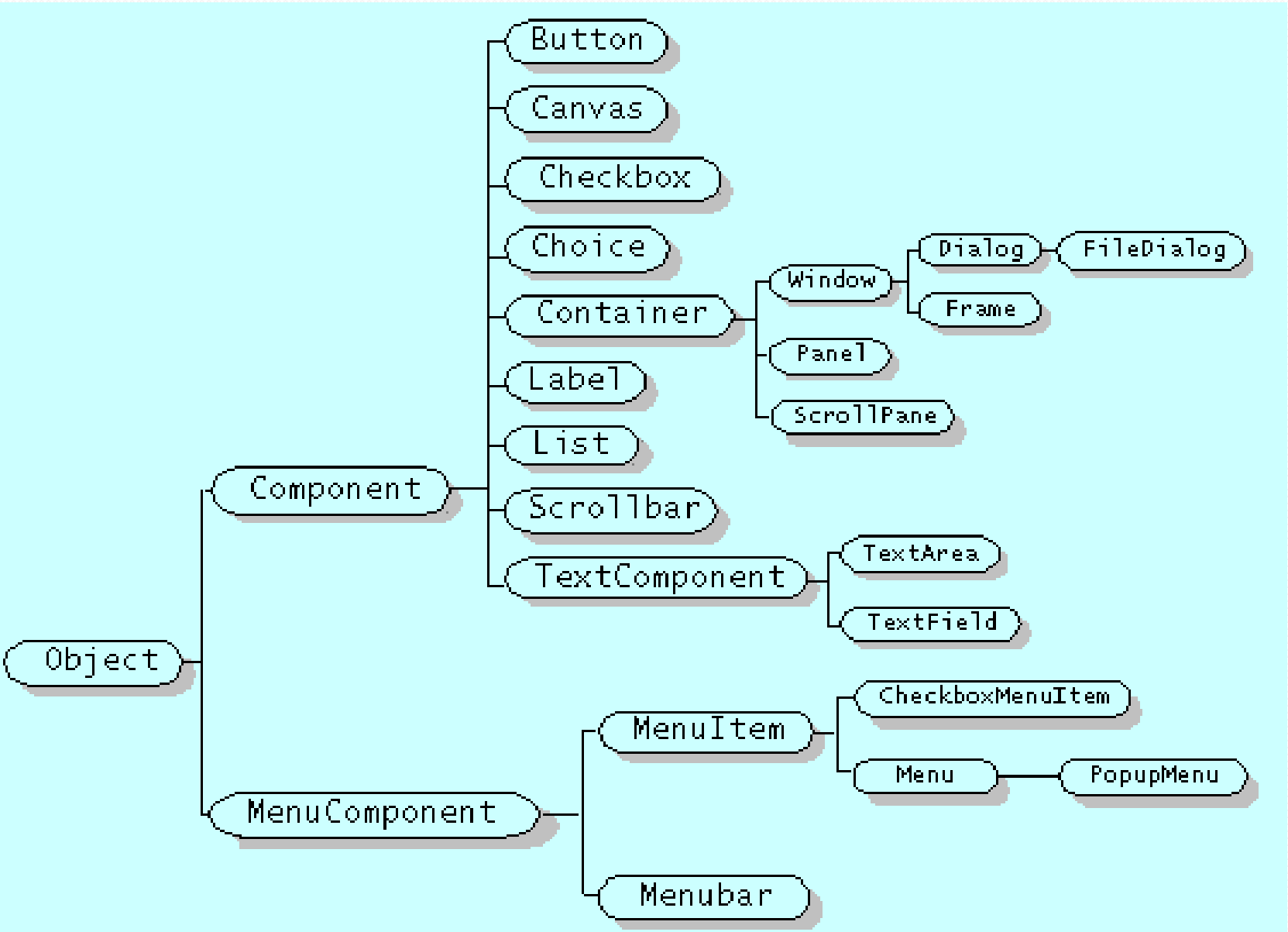
- Une interface graphique est formée d'une ou plusieurs fenêtres qui contiennent divers composants graphiques (widgets) tels que
  - boutons
  - listes déroulantes
  - menus
  - champ texte
  - etc.
- Les interfaces graphiques sont souvent appelés GUI d'après l'anglais GraphicalUser Interface

Les API utilisées pour les interfaces graphiques en Java

- 2 bibliothèques :
  - AWT Abstract WindowToolkit(, JDK 1.1)
  - Swing (JDK/SDK 1.2)
- Swing et AWT font partie de Java JFC(Java Fondation classes)) qui offre des facilités pour construire des interfaces graphiques
- Swing est construit au-dessus de AWT
  - même gestion des événements
  - les classes Swing de héritent des classes de AWT

AWT

- AWT utilise directement les composants du système.
- Cela permet d'être rapide puisque c'est le système qui s'occupe de la création. Néanmoins, Java se voulant portable,
- on est obligé de limiter les composants AWT aux composants qu'on trouve sur tous les systèmes d'exploitation.
- ces composants sont des composants lourds.
- On ne trouvait donc pas de tableau ni d'arbre.

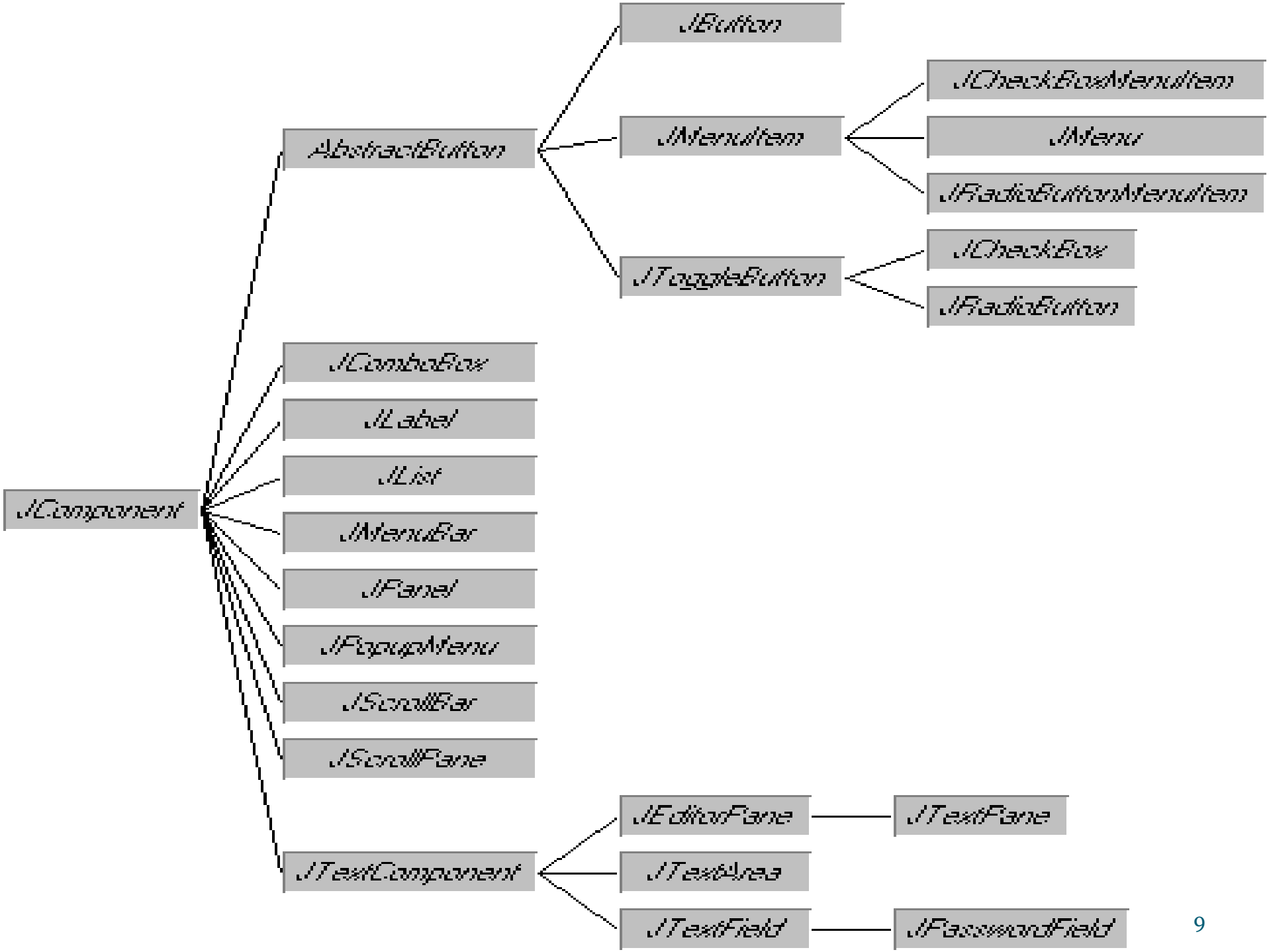

Object
Component
Button
Canvas
Checkbox
Choice
Container
Window
Dialog
FileDialog
Frame
Panel
ScrollPane
Label
List
Scrollbar
TextComponent
TextArea
TextField
MenuComponent
MenuItem
CheckboxMenuItem
Menu
PopupMenu
Menubar

Swing

- 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 mais dont le mode de fonctionnement et d'utilisation est conplètement différent.
- Swing a été intégrée au JDK depuis sa version 1.2.
- Cette bibliothèque existe séparément pour le JDK 1.1.

Swing

- Les composants Swing forment une nouvelle hiérarchie parallèle à celle de l'AWT.
- L'ancêtre de cette hiérarchie est le composant JComponent.
- Presque tous ces composants sont écrits en pur Java.
- Les composants Swing sont des composants légers (lightweight) : non dépendant de l'architecture à l'inverse (heavyweight) AWT (JDK 1.1)


JComponent
AbstractButton
JButton
JMenuItem
JCheckBoxMenuItem
JMenu
JRadioButtonMenuItem
JToggleButton
JCheckBox
JRadioButton
JComboBox
JLabel
JList
JMenuBar
JPanel
JPopupMenu
JScrollBar
JScrollPane
JTextComponent
JEditorPane
JTextPane
JTextArea
JTextField
JPasswordField

Structure d’une interface graphique :

- conteneurs: Japplet, JFrame, JPanel...
- composants « atomiques »: JButton, Jlist, JPopupMenu
- gestionnaire de disposition : LayoutManager
- interaction avec l’utilisateur : gestionnaire d’évènements

Les conteneurs

Les fenêtres

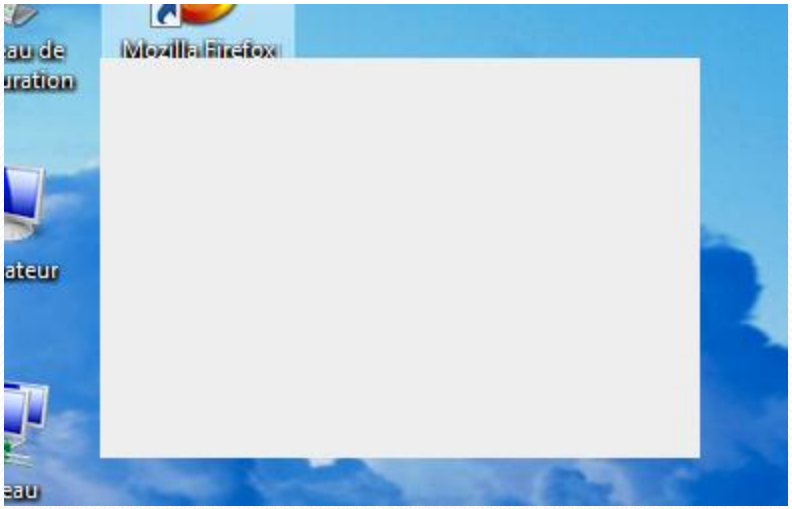
- Il existe plusieurs types de fenêtres dans Swing :
  - **JWindow**
  - **JDialog**
  - **JFrame**

JWindow

- C'est la fenêtre la plus basique. C'est juste un conteneur que vous pouvez afficher sur votre écran. Il 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.*;
public class testW {
   public static void main(String[] args) {
    //On crée une nouvelle instance de notre JWindow
    JWindow window = new JWindow();
    window.setSize(300, 200);//On lui donne une taille pour qu'on puisse la voir
    window.setVisible(true);//On la rend visible
    window.setLocationRelativeTo(null); //on centre la fenetre
  }
}
```

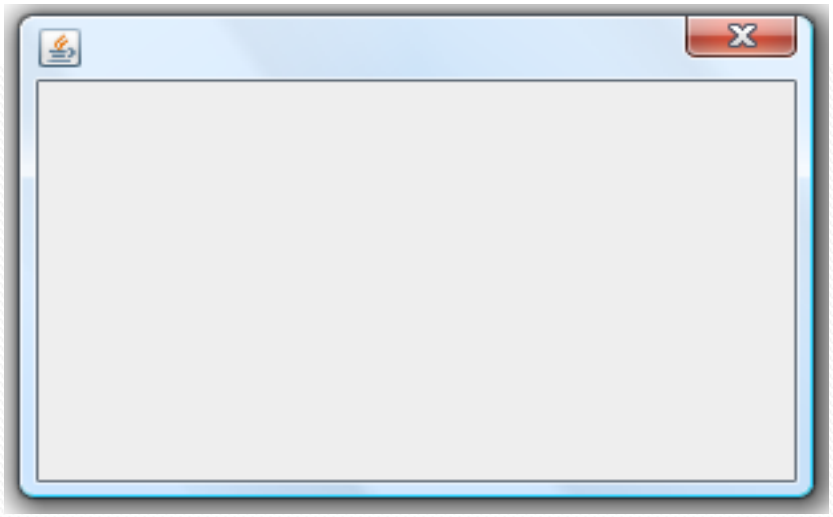


JDialog

- C'est une fenêtre destinée aux boîtes de dialogue.
- Peut être modal, c'est-à-dire qu'elle bloque une autre fenêtre tant qu'elle est ouverte.
- Elles sont destinées à travailler de paire avec la fenêtre principale(JFrame).

Exemple

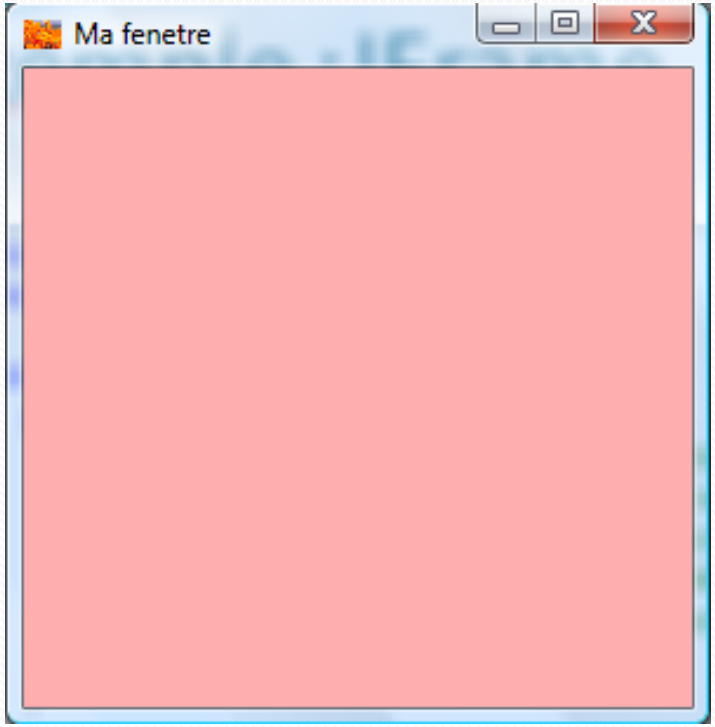
```
import javax.swing.*;
public class testW {
   public static void main(String[] args) {
    //On crée une nouvelle instance de notre JDialog
    JDialog dialog = new JDialog();
    dialog.setSize(300, 200);//On lui donne une taille pour qu'on puisse la voir
    dialog.setVisible(true);//On la rend visible
    dialog.setLocationRelativeTo(null); //on centre la fenetre
 }
}
```



JFrame

- C'est une fenêtre destinée à être la fenêtre principale de votre application.
- Elle n'est dépendante d'aucune autre fenêtre et ne peut pas être modale.
- Elle a une barre de titre et peut accueillir une barre de menu.
- Elle possède un bouton de fermeture, un bouton de redimensionnement et un bouton pour l'iconifier.

```java
import java.awt.*;
import javax.swing.JFrame;
public class Fenetre extends JFrame {
   public Fenetre()
{   // titre de la fenetre
    setTitle("Ma fenetre");
    setSize(300,300);
    setLocationRelativeTo(null);
 //On dit à l'application de se fermer lors du
 //clic sur la croix
     setDefaultCloseOperation(Jframe.EXIT_ON_CLOSE);
 // donner une couleur pour le font de la fenetre
    getContentPane().setBackground(Color.PINK);
// changer l'image de la barre de titre
    ImageIcon image = new ImageIcon("Garden.jpg");
     f.setIconImage(image.getImage());
     setVisible(true);
    }
    public static void main(String[] args) {
       Fenetre f=new Fenetre();
    }
}
```



JPanel

- Un JPanel est un conteneur élémentaire destiné à contenir d'autres composants. Il est muni d'un gestionnaire de placement.
- On l'utilise pour regrouper les different composants (voir son utilisation après, dans les exemples)

```
JPanel pan=new JPanel();
```

Les onglets : JTabbedPane

- Ils servent à présenter un ensemble de documents dans une même application.
- Cette approche utilise une seule fenêtre et chaque document peut être placer dans un panneau dédié qui est alors accessible aux travers d'onglets.
- Un panneau à onglets est représenté par la classe **JTabbedPane**.
- Ce composant est un conteneur acceptant un nombre quelconque d'enfants et affiche un onglet pour chacun.
- L'utilisateur peut alors cliquer sur un onglet pour afficher l'enfant correspondant.

JTabbedPane : Constructeurs

- **JTabbedPane()** : Création d'un panneau à onglets vierge. Si le nombre d'onglets devient important ces derniers se placent automatiquement sur plusieurs lignes.
- **JTabbedPane(int placement)** : Création d'un panneau à onglets vierge. Les onglets, se place dans l'une des quatre orientations prévues suivant la constante spécifiée en argument : JTabbedPane.TOP, JTabbedPane.BOTTOM, JTabbedPane.LEFT ou JTabbedPane.RIGHT.

JTabbedPane : Constructeurs

- **JTabbedPane(int position, int règle)** : vous avez la possibilité de faire en sorte que les onglets, quel que soit leur nombre, reste systématiquement sur une seule ligne. Dans ce cas là, des flèches supplémentaires apparaissent afin que vous puissiez naviguer dans l'ensemble de vos onglets. Voici les deux constantes que vous pouvez alors utiliser : JTabbedPane.WRAP_TAB_LAYOUT (par défaut) ou JTabbedPane.SCROLL_TAB_LAYOUT.

Ajout de nouveaux onglets :

- **void addTab(String titre, Component contenu)**: Ajout d'un nouvel onglet avec son titre, en spécifiant le composant enfant qui propose le contenu désiré.

- **void addTab(String titre, Icon icône, Component contenu)** : Par rapport à la méthode précédente, nous rajoutons une icône associée au titre de l'onglet.

Ajout de nouveaux onglets :

- **void addTab(String titre, Icon icône, Component contenu, String aide) :** Cette fois-ci, nous rajoutons également une bulle d'aide qui s'activera lors du passage du curseur de la souris au dessus de l'onglet.

- **void insertTab(String titre, Icon icône, Component contenu, String aide, int emplacement)** : Il est possible également de rajouter un onglet et de l'insérer à un emplacement spécifique. Par rapport au précédentes méthodes, il est alors nécessaire de spécifier la position désirée dans l'ordre des onglets déjà établi.

Navigation et gestion des onglets :

- **void remove(Component contenu)** : Suppression de l'onglet et de son contenu correspondant au composant choisi.
- **void remove(int index)** : Suppression de l'onglet spécifié et de son contenu.
- **void removeAll()** : Suppression de tous les onglets et de leur contenu.
- **void setBackgroundAt(int index, Color fond)** : Proposer une couleur de fond pour l'onglet sélectionné.
- **void setComponentAt(int index, Component contenu)** : Proposer un autre contenu pour l'onglet sélectionné.

Navigation et gestion des onglets :

- **void setEnabledAt(int index, boolean actif)** : Permet d'activer ou de désactiver un onglet spécifique.
- **void setForegroundAt(int index, Color fond)** : Proposer une couleur du titre pour l'onglet sélectionné.
- **void setIconAt(int index, Icon icône)** : Proposer une nouvelle icône pour l'onglet sélectionné.
- **void setSelectedIndex(int index)** : Sélectionne l'onglet.
- **void setTitleAt(int index, String titre)** : Spécifie le titre de l'onglet sélectionné.
- **void setToolTipTextAt(int index, String aide)** : Spécifie la bulle d'aide associé à l'onglet sélectionné.

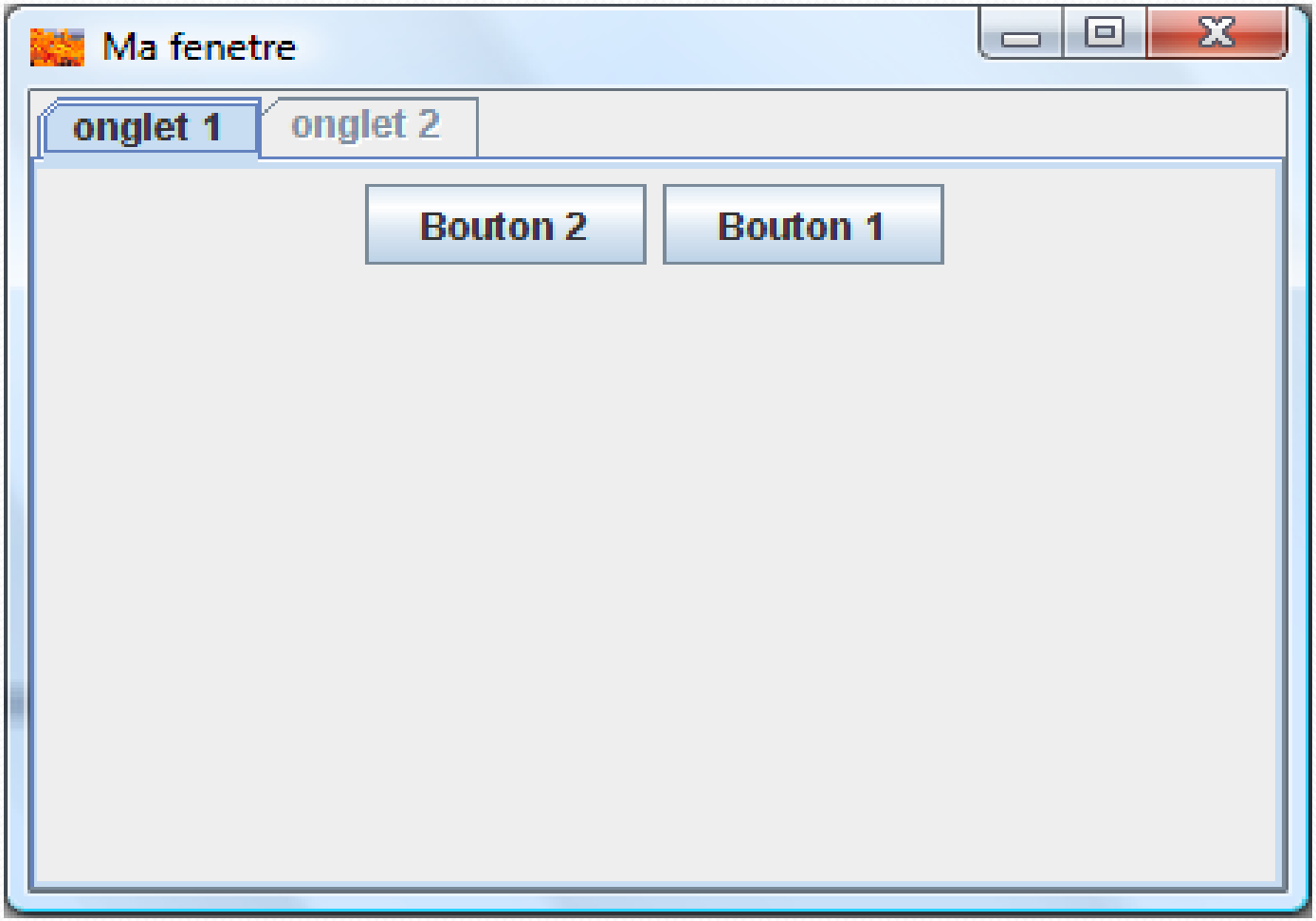
Exemple

```
import java.awt.*;
import javax.swing.*;
public class Fenetre extends JFrame {
   private JPanel pan=new JPanel();
   private JLabel lab=new JLabel("bonjour");
   private JButton b1=new JButton("Bouton 1");
   private JButton b2=new JButton("Bouton 2");
   private JTabbedPane ongl=new
    JTabbedPane(JTabbedPane.TOP,JTabbedPane.SCROLL_TA
    B_LAYOUT);
```

Exemple

```
public Fenetre() {
   pan.add(b2);
   pan.add(b1);
   ongl.addTab("onglet 1",pan);
   ongl.addTab("onglet 2",lab);
   ongl.setEnabledAt(1,false);
   add(ongl);
   setTitle("Ma fenetre");
   setSize(500,500);
   setLocationRelativeTo(null);
   setDefaultCloseOperation(EXIT_ON_CLOSE);
   ImageIcon image = new ImageIcon("Garden.jpg");
    setIconImage(image.getImage());
   setVisible(true);}
  public static void main(String[] args) {
     Fenetre f=new Fenetre();        }}
```

Exemple



Le gestionnaire de placement

Architecture de Layout

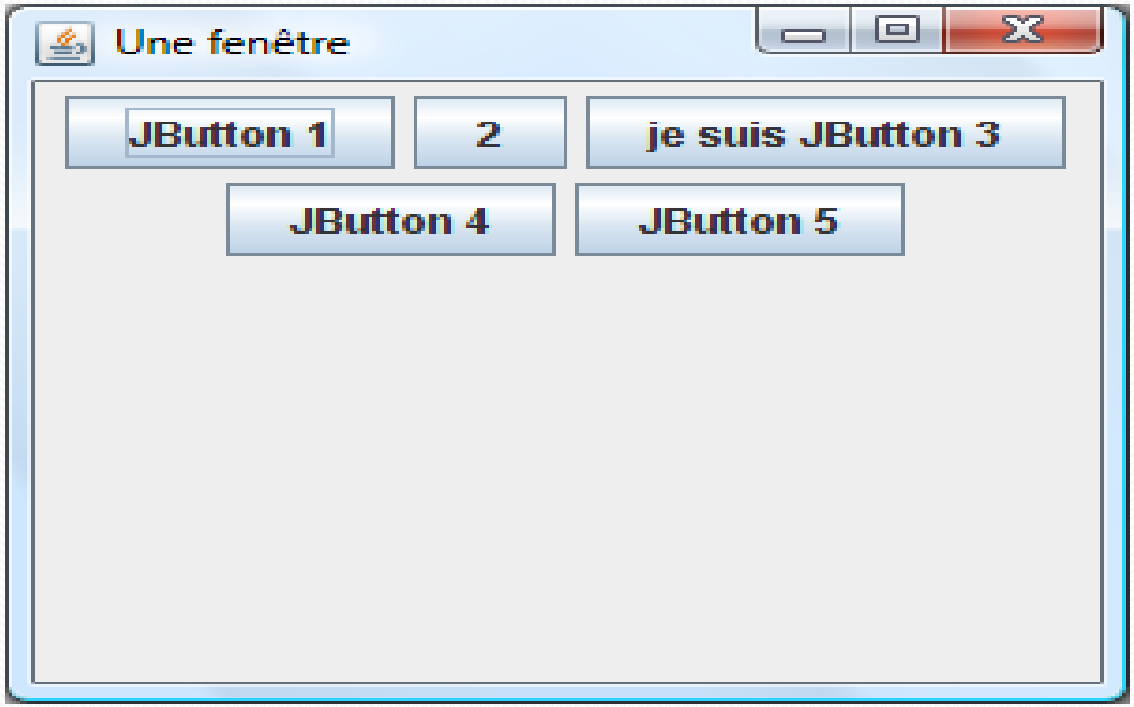
- Pour placer des composants dans un container, Java propose une technique de Layout.
- Un layout est une entité Java qui place les composants les uns par rapport aux autres.
- Le layout s'occupe aussi de réorganiser les composants lorsque la taille du container varie.
- Il y a plusieurs layout : BorderLayout, BoxLayout, CardLayout, FlowLayout, GridLayout, GridBagLayout.
- Un layout n'est pas contenu dans un container, il gère le positionnement.

Gestionnaire de disposition

Classe	Description
`java.awt.FlowLayout`	Dispose les composants d'un container les uns derrière les autres en ligne et à leur taille préférée, en retournant à la ligne si le container n'est pas assez large.
`java.awt.GridLayout`	Dispose les composants d'un container dans une grille dont toutes les cellules ont les mêmes dimensions.
`java.awt.BorderLayout`	Dispose cinq composants maximum dans un container, deux au bords supérieur et inférieur à leur hauteur préférée, deux au bords gauche et droit à leur largeur préférée, et un au centre qui occupe le reste de l'espace.
`java.awt.CardLayout`	Affiche un composant à la fois parmi l'ensemble des composants d'un container (pratique pour créer des panneaux comme ceux de la boîte de dialogue de *Preferences* d'Eclipse).
`java.awt.GridBagLayout`	Dispose les composants d'un container dans une grille dont les cellules peuvent avoir des dimensions variables. La position et les dimensions de la cellule d'un composant varient en fonction de sa taille préférée et des contraintes de classe `java.awt.GridBagConstraints` qui lui sont associées.

FlowLayout

- Un FlowLayout permet de ranger les composants dans une ligne. Si l'espace est trop petit, une autre ligne est créée.
- Le FlowLayout est le layout par défaut des JPanel



Exemple : FlowLayout

```
import java.awt.*;
public class DemoFlowLayout extends JFrame {
    private JButton nord = new JButton("JButton 1");
    private JButton ouest = new JButton("2");
    private JButton sud = new JButton("je suis JButton 3");
    private JButton centre = new JButton("JButton 4");
    private JButton est = new JButton("JButton 5");
    private JPanel panneau = new JPanel();

    public DemoFlowLayout() {
        setTitle("Une fenêtre");
        setSize(300, 250);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        panneau.setLayout(new FlowLayout());
        panneau.add(nord);
        panneau.add(ouest);
        panneau.add(sud);
        panneau.add(centre);
        panneau.add(est);
        add(panneau);
        setVisible(true);
    }
    public static void main(String[] args) {
        new DemoFlowLayout();
    }
}
```

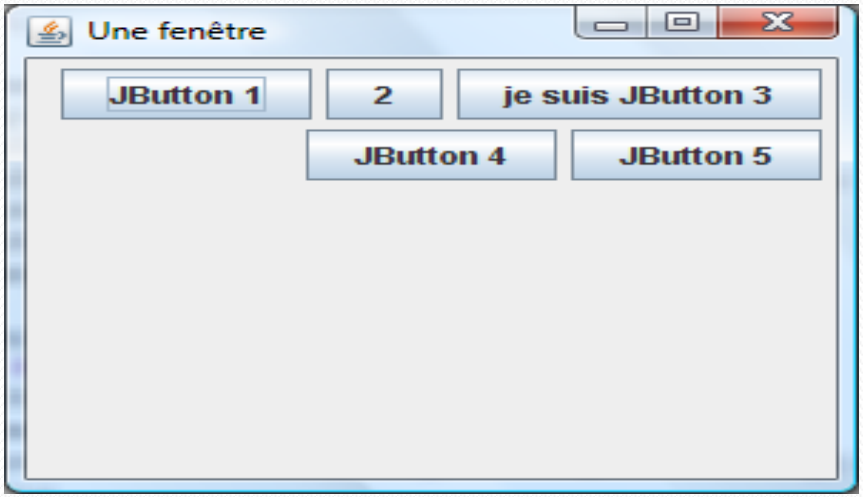
Exemple : FlowLayout

- Lors de la construction d'un gestionnaire FlowLayout, nous pouvons spécifier un paramètre d'alignement d'une ligne de composants par rapport aux bords verticaux de la fenêtre. Pour cela, nous utilisons l'une des composantes entières suivantes :

Constante symbolique	Valeur	Alignement de la ligne de composants
FlowLayout.LEFT	"Left"	A gauche
FlowLayout.RIGHT	"Right"	A droite
FlowLayout.CENTER	"Center"	Au centre (valeur par défaut)

Exemple : FlowLayout

- setLayout(new FlowLayout. RIGHT);



GridLayout

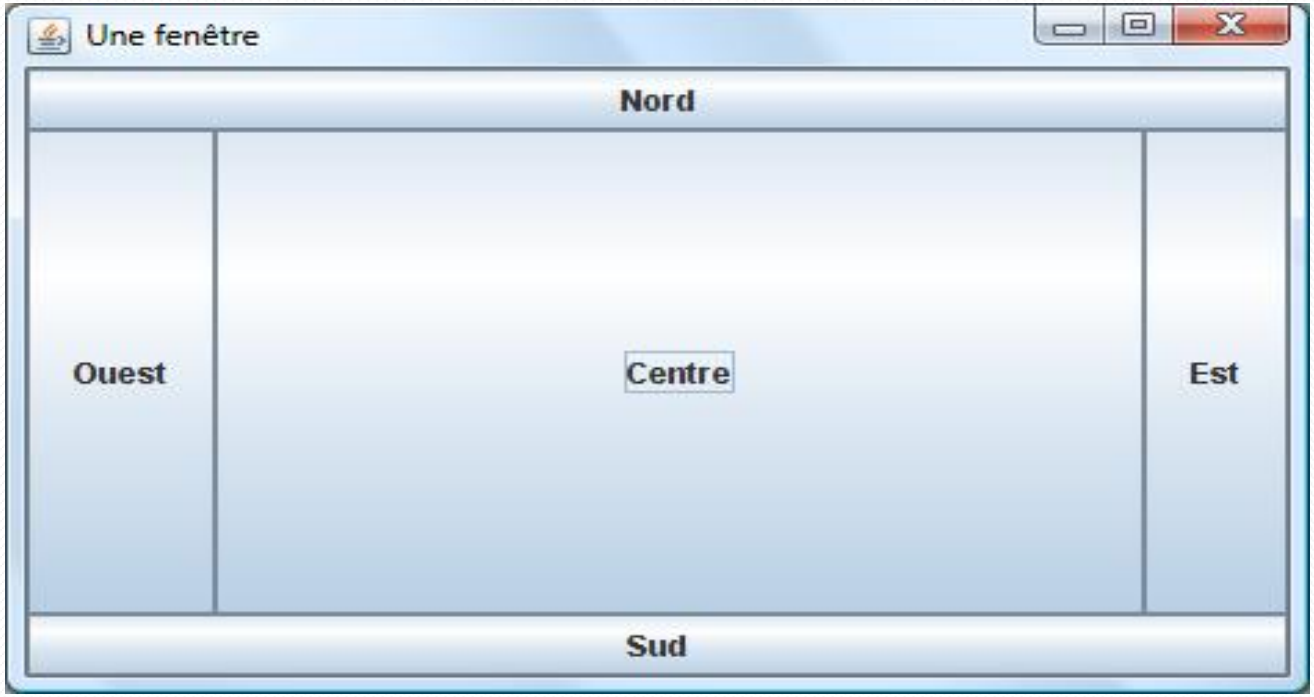
- Un GridLayout permet de positionner les composants sur une grille.

Exemple : GridLayout

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

public class DemoGridLayout extends JFrame {
    private JButton b1 = new JButton("JButton1");
    private JButton b2 = new JButton("JButton2");
    private JButton b3 = new JButton("JButton3");
    private JButton b4 = new JButton("JButton4");
    private JButton b5 = new JButton("JButton5");

    public DemoGridLayout() {
        setTitle("Une fenêtre");
        setSize(300, 250);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setLayout(new GridLayout(3, 2));
        add(b1);
        add(b2);
        add(b3);
        add(b4);
        add(b5);
        setVisible(true);
    }
    public static void main(String[] args) {
        new DemoGridLayout();
    }
}
```

BorderLayout

- Le BorderLayout sépare un container en cinq zones: NORTH, SOUTH, EAST, WEST et CENTER
- Lorsque l'on agrandit le container, le centre s'agrandit. Les autres zone prennent uniquement l'espace qui leur est nécessaire.

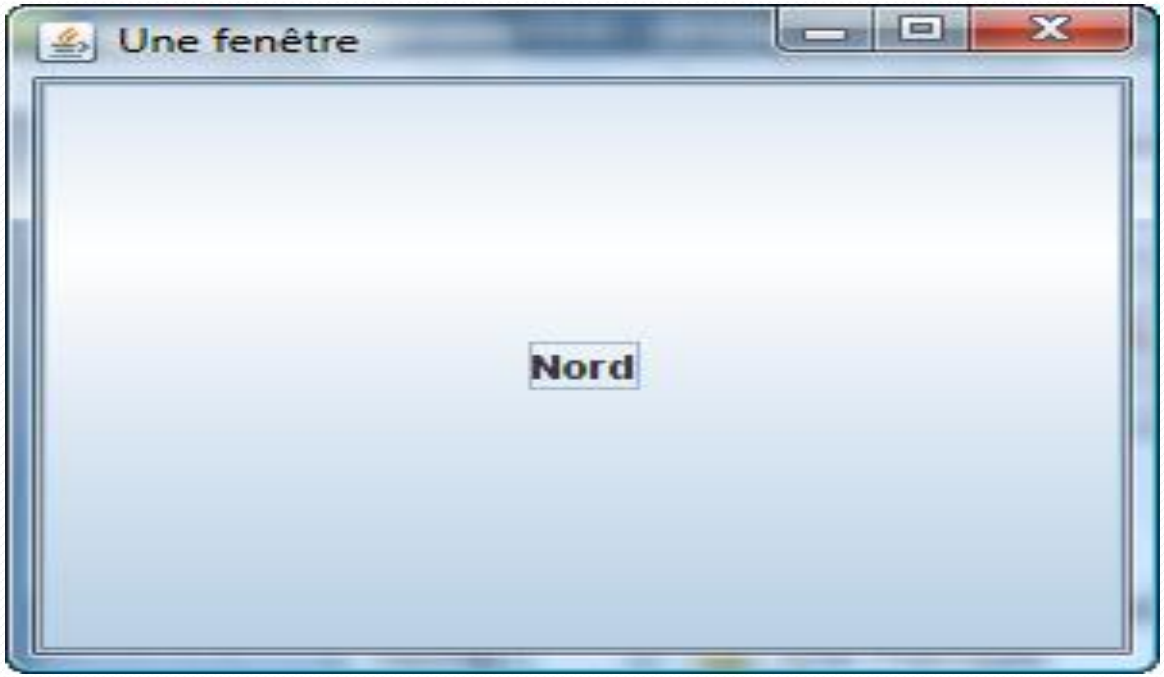


Exemple : BorderLayout

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

public class DemoBorderLayout extends JFrame {
    private JButton nord = new JButton("Nord");
    private JButton ouest = new JButton("Ouest");
    private JButton sud = new JButton("Sud");
    private JButton centre = new JButton("Centre");
    private JButton est = new JButton("Est");

    public DemoBorderLayout() {
        setTitle("Une fenêtre");
        setSize(300, 250);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        add(nord, BorderLayout.NORTH);
        add(ouest, BorderLayout.WEST);
        add(sud, BorderLayout.SOUTH);
        add(centre, BorderLayout.CENTER);
        add(est, BorderLayout.EAST);
        setVisible(true);
    }
    public static void main(String[] args) {
        new DemoBorderLayout();
    }
}
```

CardLayout

- Un CardLayout permet d'avoir plusieurs conteneurs ; les uns au dessus des autres (comme un jeux de cartes).



Exemple : CardLayout

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

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

    public DemoCardLayout() {
        setTitle("Une fenêtre");
        setSize(300, 250);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setLayout(new CardLayout());
        add(b1, "Nord");
        add(b2, "Ouest");
        add(b3, "Sud");
        add(b4, "Centre");
        add(b5, "Est");
        setVisible(true);
    }
    public static void main(String[] args) {
        new DemoCardLayout();
    }
}
```

Les composants atomiques

Les composants atomique

- Jlabel
- JTooltip
- JTextfield
- JTextArea
- JButton
- JCheckBox
- JRadioButton
- Jlist
- JComboBox

JLabel

- **JLabel()** : Construit une étiquette vierge, sans texte ni icône.
- **JLabel(String libellé)** : Construit une étiquette avec le texte aligné à gauche.
- **JLabel(Icon icône)** : Construit une étiquette sans texte, avec une icône alignée à gauche.
- **JLabel(String libellé, int alignement)** : Construit une étiquette en choisissant l'alignement du texte au moyen d'une des constantes LEFT, CENTER ou RIGHT.

JLabel

- **JLabel(Icon icône, int alignement)** : Construit une étiquette sans texte en choisissant l'alignement de l'icône au moyen d'une des constantes LEFT, CENTER ou RIGHT.
- **JLabel(String libellé, Icon icône, int alignementHorizontal)** : Construit une étiquette avec un texte et une icône placée à gauche du texte en choisissant l'alignement du couple suivant l'une des constantes LEFT, CENTER ou RIGHT.

Exemlpe

```
import java.awt.*;

import javax.swing.*;
public class Fenetre extends JFrame {

   private JLabel lab=new JLabel("bonjour",new ImageIcon("c:/home_bebe[1].jpg"),JLabel.CENTER);

   public Fenetre() {

    lab.setFont(new Font("Aial",Font.BOLD,20));
    add(lab);

    setTitle("Ma fenetre");
    setSize(550,550);
    setLocationRelativeTo(null); setDefaultCloseOperation(EXIT_ON_CLOSE);
    Toolkit tk=Toolkit.getDefaultToolkit();
    setIconImage(tk.getImage("c:/Garden.jpg"));
    setVisible(true);
    }

   public static void main(String[] args) {
       Fenetre f=new Fenetre();
    }
}
```

Exemple

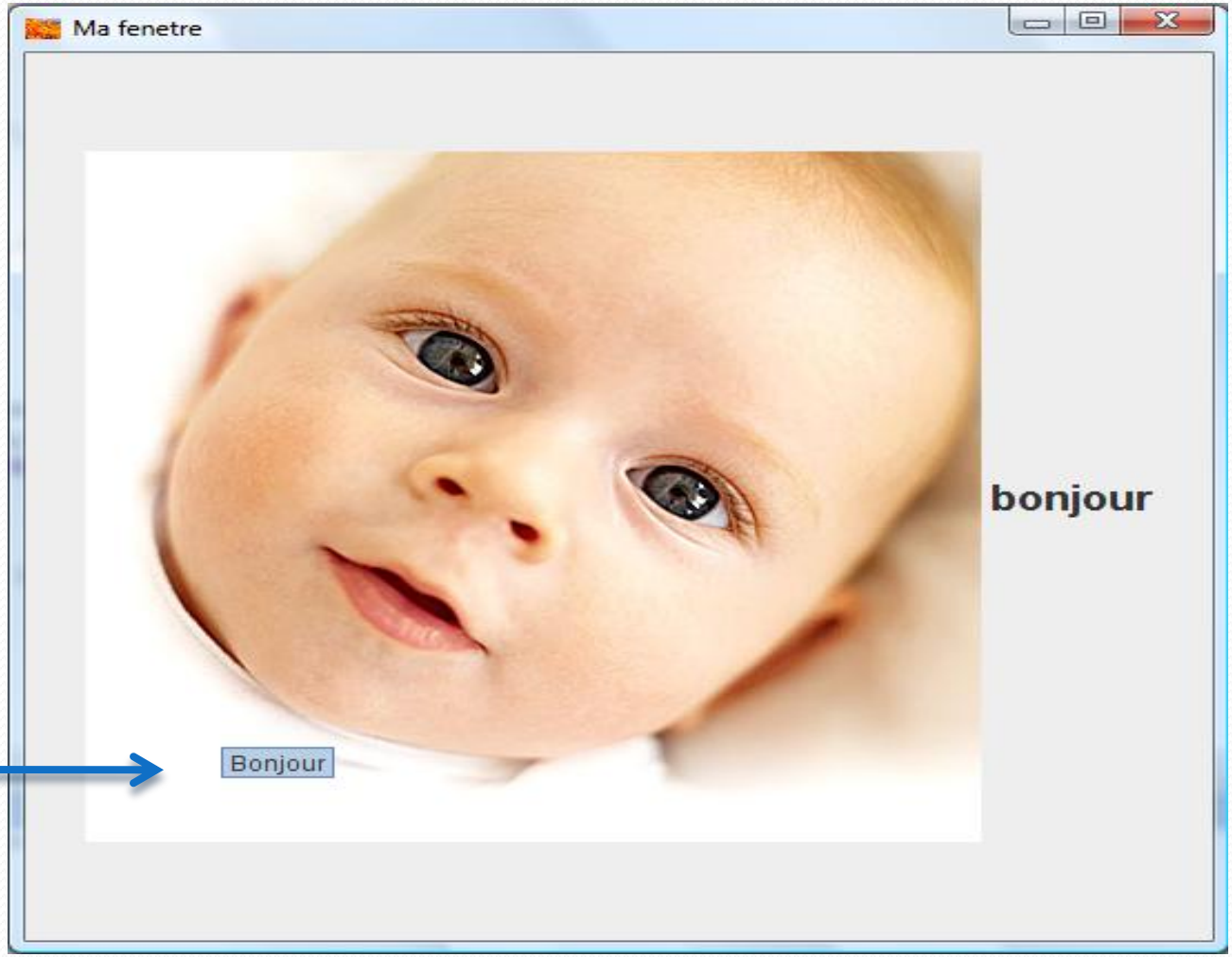
JToolTip

- Il est possible, lorsque la souris passe au dessus d'un composant, de faire apparaître un petit message sous le curseur et qui précise le rôle du composant, ce que nous appelons une bulle d'aide.
- Cette bulle d'aide est assurée par la classe JToolTip.
- Tous les composants graphiques, qui héritent donc de JComponent, peuvent accéder à cette fonctionnalité au travers de la méthode setToolTipText(String).
- Il suffit juste de préciser le texte désiré, et la bulle d'aide s'affichera alors automatiquement à chaque passage du curseur de la souris au dessus de ce composant.

Exemple

```
import java.awt.*;

import javax.swing.*;
public class Fenetre extends JFrame {

   private JLabel lab=new JLabel("bonjour",new ImageIcon("c:/home_bebe[1].jpg"),JLabel.CENTER);

   public Fenetre() {

    lab.setFont(new Font("Aial",Font.BOLD,20));
    lab.setToolTipText("Bonjour");
    add(lab);

    setTitle("Ma fenetre");
    setSize(550,550);
    setLocationRelativeTo(null); setDefaultCloseOperation(EXIT_ON_CLOSE);
    Toolkit tk=Toolkit.getDefaultToolkit();
    setIconImage(tk.getImage("c:/Garden.jpg"));
    setVisible(true);
    }

   public static void main(String[] args) {
       Fenetre f=new Fenetre();
    }
}
```

Exemple



JScrollPane

- C’est la classe qui permet de définir les barres de défilement horizontale et verticale.
- Lorsque vous créez un JScrollPane, vous pouvez préciser les conditions d'affichage de ses barres de défilement.
- Cela s'appelle une politique d'affichage d'une barre de défilement.

JScrollPane : constructeurs

- **JScrollPane()** : Crée un panneau sans composant interne avec des barres de défilement qui apparaissent suivant le besoin. Vous devez précisez par la suite le composant qui bénificiera de cette fonctionnalité au travers de la méthode setViewportView(Component).
- **JScrollPane(Component composantEnveloppé)** : Crée un panneau qui intègre le composant enveloppé et qui propose des barres de défilement horizontale et verticale qui apparaissent automatiquement suivant les capacités de ce composant interne et suivant les besoins du moment.

JScrollPane : constructeurs

- **JScrollPane(Component composantEnveloppé, int politiqueVerticale, int politiqueHorizontale)** : Crée un panneau qui intègre le composant enveloppé et qui propose éventuellement des barres de défilement horizontale et verticale suivant la politique choisie.

JScrollPane

- **HORIZONTAL_SCROLLBAR_AS_NEEDED, VERTICAL_SCROLLBAR_AS_NEEDED** : n'affiche une barre de défilement que si le composant enveloppé ne tient pas entièrement.
- **HORIZONTAL_SCROLLBAR_ALWAYS, VERTICAL_SCROLLBAR_ALWAYS** : affiche toujours une barre de défilement, quelle que soit la taille du composant enveloppée.
- **HORIZONTAL_SCROLLBAR_NEVER, VERTICAL_SCROLLBAR_NEVER** : n'affiche jamais de barre, même lorsque le composant enveloppé ne tient pas entièrement. Si vous utilisez cette politique pour les deux barres, vous devez proposer un autre moyen de manipuler le JScrollPane.

Exemple

```
import java.awt.*;

import javax.swing.*;
public class Fenetre extends JFrame {

   private JLabel lab=new JLabel("bonjour",new ImageIcon("c:/home_bebe[1].jpg"),JLabel.CENTER);
   private JPanel pan= new JPanel();
   private JScrollPane scroll=new
    JScrollPane(pan,ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED ,ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED);
   public Fenetre() {

    lab.setFont(new Font("Aial",Font.BOLD,20));
    lab.setToolTipText("Bonjour");
    pan.add(lab);
    add(scroll);

    setTitle("Ma fenetre");
    setSize(550,550);
    setLocationRelativeTo(null); setDefaultCloseOperation(EXIT_ON_CLOSE);
    Toolkit tk=Toolkit.getDefaultToolkit();
    setIconImage(tk.getImage("c:/Garden.jpg"));
    setVisible(true);
    }

   public static void main(String[] args) {
       Fenetre f=new Fenetre();
    }
}
```

Exemple

JTexteField

- **JTexteField** : collecte une entrée de texte sur une seule ligne.
- nous pouvons lire et écrire le texte avec les méthode
  - getText()
  - setText()
- Ces deux méthodes sont héritées de la super-classe JTextComponent.
- **Constructeurs :**
  - JTextField(String chaine, int taille)
  - JTextField(String chaine)
  - JTextField(int taille)

JTexteField

- D'autres méthodes propres à la classe JTextField :

  - **setFont()** : permet de spécifier la fonte dans lequel le texte est affiché.
  - **setColumns()** : donne le nombre de caractères dans le champ.

JTextArea

- JTextArea : collecte une entrée de texte sur plusieurs lignes.
- Constructeurs:
  - **JTextArea(int lignes, int colonnes):** spécifie le nombre de lignes et de colonnes pour la zone
  - **JTextArea() :** zone de texte vide .
- Si on a pas spécifie le nombre de lignes et de colonnes pour la zone on utilise les méthodes **setColumns() et setRows()** .

JTextArea

- Retour à la ligne automatique : si le texte entré dépasse la capacité d'affichage de la zone de texte, le reste du texte est coupé. Pour éviter que des longues lignes ne soient tronquées, vous pouvez activer le retour automatique à la ligne avec la méthode **setLineWrap()** :

JTextArea

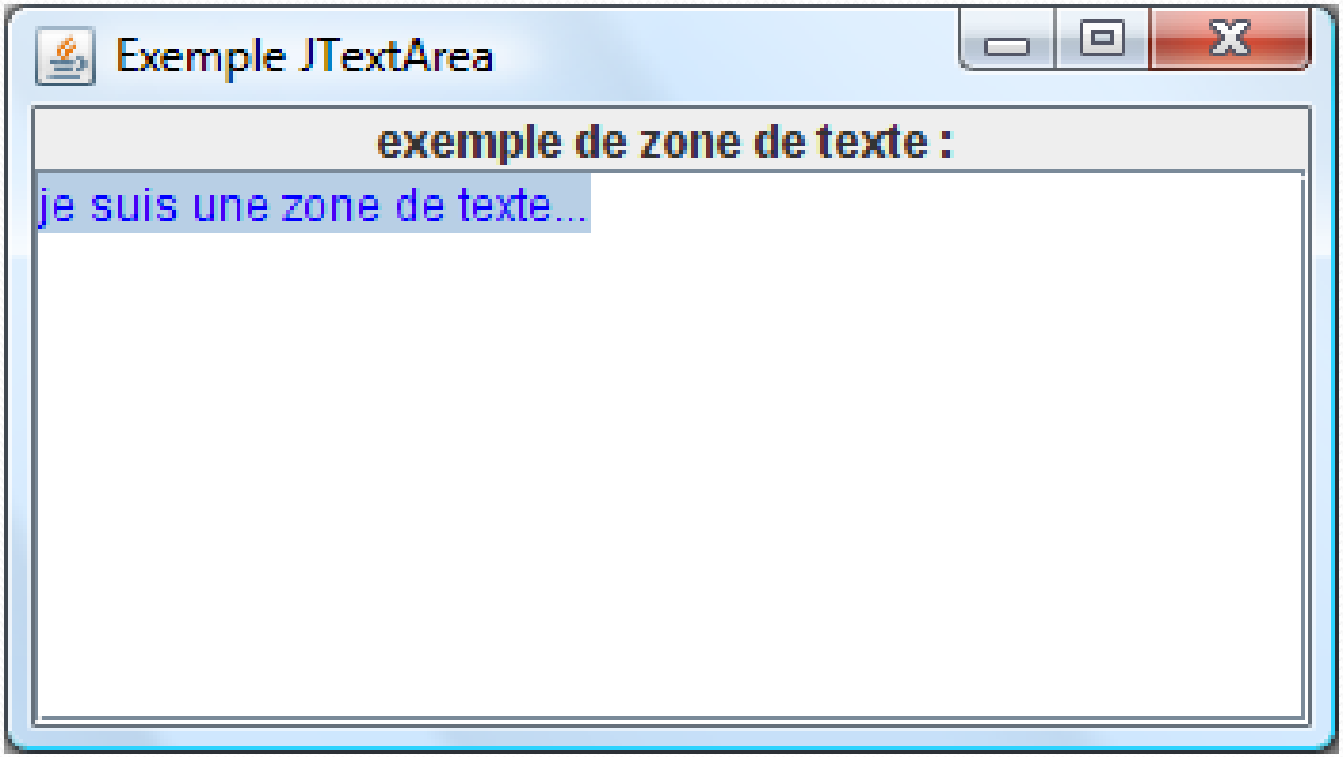
- setEditable() :Permettre l'édition ou provoquer la lecture seule
- setMargin() :Proposer des marges
- setSelectedTextColor() et setSelectionColor() : Changer les couleurs de la sélection
- setCaretColor() : Changer la couleur du curseur de texte
- getSelectedText() : Récupérer la sélection
- getText() : Récupérer le texte

JTextArea

DemoJTextArea.java

```
import javax.swing.*;
import java.awt.*;
public class DemoJTextArea extends JFrame{

private JLabel etiquette=new JLabel("exemple de zone de texte : ",JLabel.CENTER);
private JTextArea texte= new JTextArea("je suis une zone de texte...");
private JScrollPane ascenceur = new JScrollPane(texte);

public DemoJTextArea(){
     super("Exemple JTextArea ") ;
     this.setLayout(new BorderLayout());
     texte.setSelectedTextColor(Color.BLUE);
     texte.setCaretColor(Color.magenta);
     this.add(etiquette,BorderLayout.NORTH);
     this.add(ascenceur,BorderLayout.CENTER);
     String s=texte.getText();
     System.out.println(s);
     this.setBounds(100,100,200,300);
     this.setDefaultCloseOperation(EXIT_ON_CLOSE);
     this.setVisible(true);
}
public static void main(String[] args){
     new DemoJTextArea();
}
}
```

JTextArea


Exemple JTextArea
exemple de zone de texte :
je suis une zone de texte...

Composants de choix

- Jbutton
- JCheckBox
- JRadioButton
- Jlist
- JComboBox

JButton

- Cette classe JButton implémente un bouton poussoir.
- Les boutons Swing peuvent comporter une image en plus d'un label.
- La classe **JButton** possède des constructeurs acceptant un objet Icon capable de se dessiner lui-même.
- Vous pouvez créer des boutons dotés de légendes, d'images ou des deux.

JButton

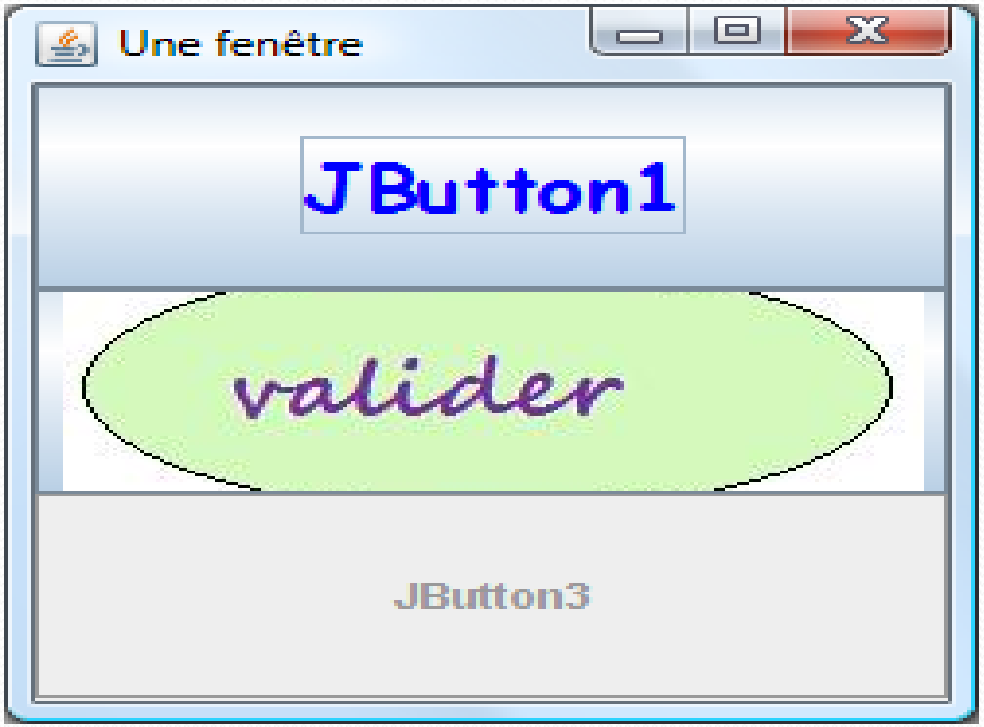
- Constructeurs :
  - JButton()
  - JButton(String libellé)
  - JButton(Icon icône)
  - JButton(String libellé, Icon icône)

JButton

DemoJButton.java

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

public class DemoJButton extends JFrame {
    private JButton b1 = new JButton("JButton1");
    private JButton b2 = new JButton(new ImageIcon("valider.jpg"));
    private JButton b3 = new JButton("JButton3");
    public DemoJButton() {
        setTitle("Une fenêtre");
        b1.setFont(new Font("Comic Sans MS",Font.BOLD,22));
        b1.setForeground(Color.BLUE);
        b3. setEnabled(false);
        setSize(250, 250);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setLayout(new GridLayout(3,1));
        add(b1);
        add(b2);
        add(b3);

        setVisible(true);
    }
    public static void main(String[] args) {
        new DemoJButton();
    }
}
```

JButton



Les cases à cocher : JCheckBox

- Si vous souhaitez recueillir une réponse qui se limite à une entrée "oui" ou "non", utilisez plutôt une case à cocher. Ce type de composant, qui est représenté par la classe JCheckBox, s'accompagne d'un intitulé qui permet de les identifier.

- L'utilisateur clique à l'intérieur de la case pour la cocher, et fait de même pour la désactiver.

Les cases à cocher : JCheckBox

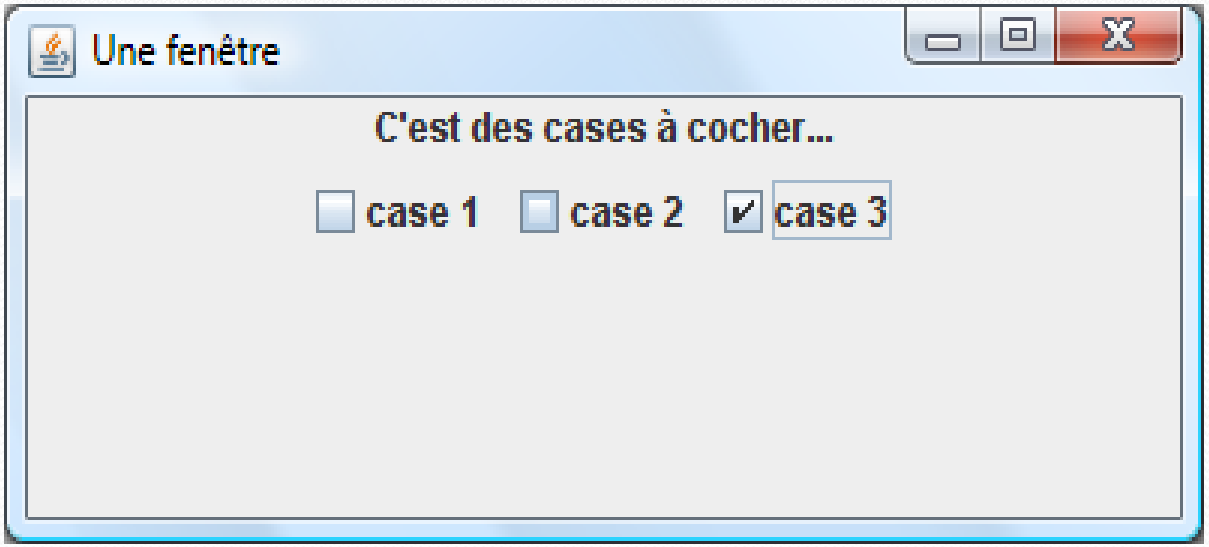
- Constructeurs :
  - JCheckBox()
  - JCheckBox(String libellé)
  - JCheckBox(Icon icône)
  - JCheckBox(String libellé, Icon icône)
  - JCheckBox(String libellé, boolean sélectionné)
  - JCheckBox(Icon icône, boolean sélectionné)
  - JCheckBox(String libellé, Icon icône, boolean sélectionné)
- La méthode isSelected():permet de voir si une case à cocher est selectionnée ou non .

Les cases à cocher : JCheckBox

DemoJCheckBox.java

```
import java.awt.*;
import javax.swing.*;
public class DemoJCheckBox extends JFrame{
   private JLabel etiquette = new JLabel("C'est des cases à cocher...",JLabel.CENTER);
   private JCheckBox r1 = new JCheckBox("case 1");
   private JCheckBox r2 = new JCheckBox("case 2");
   private JCheckBox r3 = new JCheckBox("case 3");
   private JPanel panneau = new JPanel();

    public DemoJCheckBox() {
        setTitle("Une fenêtre");
        setSize(400, 160);
        setLayout(new BorderLayout());
        panneau.add(r1);
        panneau.add(r2);
        panneau.add(r3);
        add(etiquette,BorderLayout.NORTH);
        add(panneau,BorderLayout.CENTER);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setVisible(true);
   }
   public static void main(String[] args) {
      new DemoJCheckBox();
   }}
```

Les cases à cocher : JCheckBox



Les boutons radio: JRadioButton

- De nombreuses situations exigent que l'utilisateur ne puissent choisir qu'une seule option parmi un ensemble de choix. Lorqu'une seconde option est sélectionnée, la première est désactivée.

- Cet ensemble d'options est souvent mis en oeuvre au moyen d'un groupe de boutons radio, représentés par la classe JRadioButton.

Les boutons radio: JRadioButton

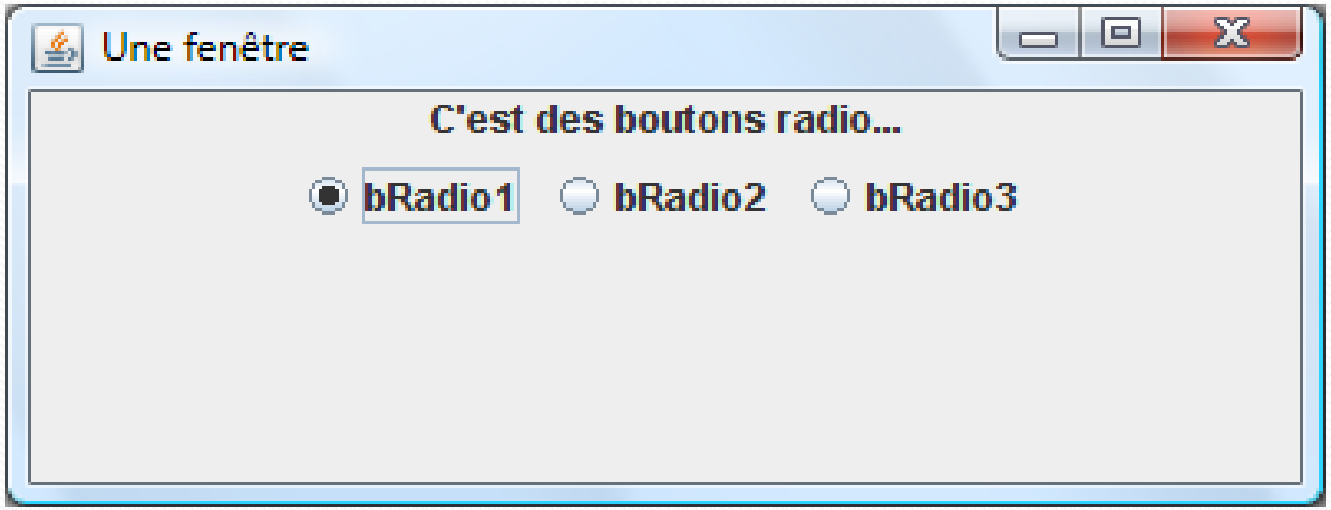
- Constructeurs :
  - **JRadioButton()**
  - **JRadioButton(String libellé)**
  - **JRadioButton(Icon icône)**
  - **JRadioButton(String libellé, Icon icône)**
  - **JRadioButton(String libellé, boolean sélectionné)**
  - **JRadioButton(Icon icône, boolean sélectionné)**
  - **JRadioButton(String libellé, Icon icône, boolean sélectionné)**

Les boutons radio: JRadioButton

DemoJRadioButton.java

```
import java.awt.*;
import javax.swing.*;
public class DemoJRadioButton extends JFrame{
    private JLabel etiquette = new JLabel("C'est des boutons radio...",JLabel.CENTER);
    private ButtonGroup groupe = new ButtonGroup();
    private JRadioButton r1 = new JRadioButton("bRadio1");
    private JRadioButton r2 = new JRadioButton("bRadio2");
    private JRadioButton r3 = new JRadioButton("bRadio3");
    private JPanel panneau = new JPanel();

     public DemoJRadioButton() {
          setTitle("Une fenêtre");
          setSize(400, 160);
          setLayout(new BorderLayout());
          groupe.add(r1);
          groupe.add(r2);
          groupe.add(r3);
          panneau.add(r1);
          panneau.add(r2);
          panneau.add(r3);
          add(etiquette,BorderLayout.NORTH);
          add(panneau,BorderLayout.CENTER);
          setDefaultCloseOperation(EXIT_ON_CLOSE);
          setVisible(true);
    }
    public static void main(String[] args) {
       new DemoJRadioButton();
    }}
```

Les boutons radio: JRadioButton



Liste de Choix : JList

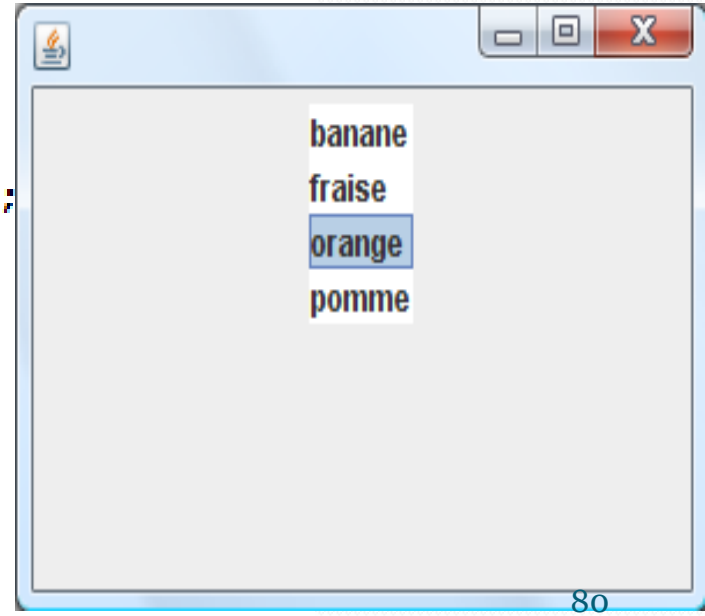
- JList est un composant qui permet de choisir une ou plusieurs valeurs dans une liste prédéfinie.
- Constructeurs :
  - **JList()** : Préparer une liste vide. Elle sera rempli après par la méthode setListData().
  - **JList(Object[] liste)** : Construire la boîte au moyen d'un tableau d'objet, comme par exemple un tableau de chaînes de caractères : String[]. Dans ce cas de figure, le nombre d'élément est prédéterminé.
  - **JList(Vector liste)** : Nous pouvons aussi construire la boîte de liste au moyen d'un tableau dynamique représenté par la classe **Vector.**

Liste de Choix : JList

- Il existe trois sortes de boîtes de liste, correspondant chacune à la façon de sélectionner un élément ou un ensemble d'éléments.
- Il est donc nécessaire, une fois que l'objet JList est construit, de choisir le mode de sélection au travers de la méthode **setSelectionMode()** en proposant le type approprié :
  - **SINGLE_SELECTION** : Sélection d'une seule valeur.
  - **SINGLE_INTERVAL_SELECTION** : Sélection d'une seule plage de valeur (contiguës).
  - **MULTIPLE_INTERVAL_SELECTION** : Sélection d'un nombre quelconque de plages de valeurs.

Exemple

```
import javax.swing.*;
import java.awt.*;
public class TestJListe extends JFrame{
    private String[] choix={"banane","fraise","orange","pomme"};
    JList liste=new JList(choix);
    JPanel p=new JPanel();

    public TestJListe()
    {   liste.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
        liste.setSelectedIndex(2);

        p.add(liste);
        add(p);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(300,200);
        setVisible(true);
    }
    public static void main(String[] args) {
        new TestJListe();
    }
}
```



Liste déroulante (liste combinée) : JComboBox

- Dès que le nombre d'options augmente, les boutons radio ne conviennent plus, car ils occupent trop d'espace à l'écran. Vous pouvez, dans ce cas, utiliser une liste déroulante. Lorsque l'utilisateur clique sur le champ, une liste de choix se déroule
- Constructeurs :
  - **ComboBox()** : Préparer une liste déroulante vide qui sera complété plus tard à l'aide de la méthode **addItem().**
  - **JComboBox(Object[] liste)** : Construire la liste déroulante au moyen d'un tableau d'objet, comme par exemple un tableau de chaînes de caractères : String[]. Dans ce cas de figure, le nombre d'élément est prédéterminé.
  - **JComboBox(Vector liste)** : Nous pouvons aussi construire la liste déroulante au moyen d'un tableau dynamique représenté par la classe Vector.

Liste déroulante (liste combinée) : JComboBox

- D'autres méthodes :
- **addItem(Object rubrique):** Ajouter une nouvelle rubrique à la fin de votre liste déroulante.
- **insertItemAt(Object rubrique, int position) :** Insérer une nouvelle rubrique à un endroit spécifique
- **removeAllItems() :** Supprimer toutes les rubriques
- **removeItem(Object rubrique) :** Supprimer une rubrique spécifique
- **removeItemAt(int position) :** Supprimer une rubrique suivant la position spécifiée
- **getItemCount()** : Connaître le nombre de rubriques déjà présentes

Liste déroulante (liste combinée) : JComboBox

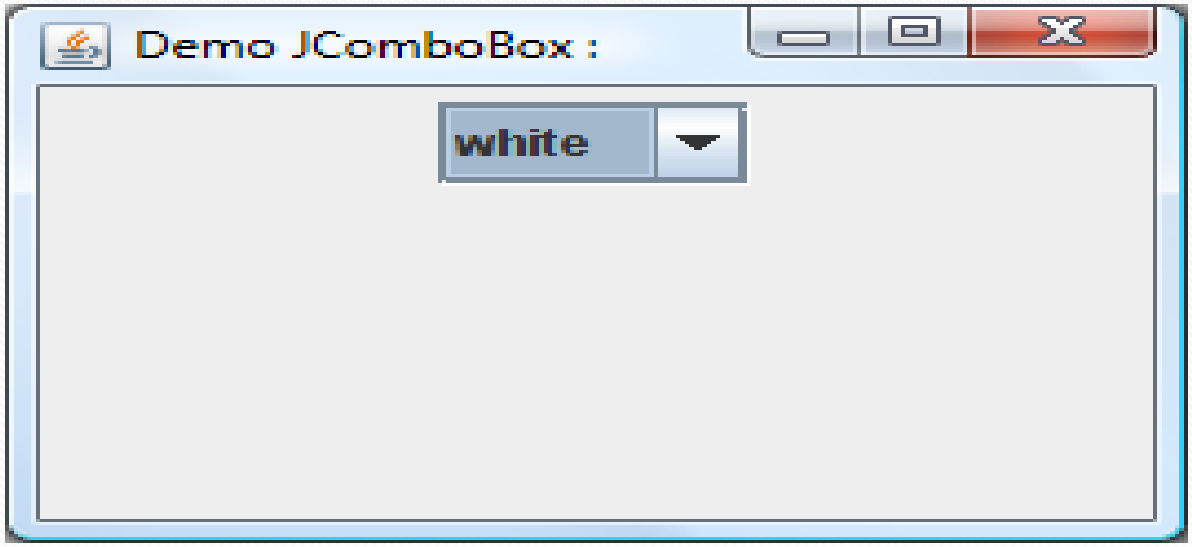
- **getSelectedItem()** fournit la valeur sélectionnée, qu'il s'agisse d'une valeur provenant de la liste prédéfinie ou d'une valeur saisie dans le champ de texte associé.
- **getSelectedIndex()** fournit aussi le rang de la valeur sélectionnée.

Liste déroulante (liste combinée) : JComboBox

DemoJComboBox.java

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

public class DemoJComboBox extends JFrame {

    String[] listColorNames = { "black", "blue", "green", "yellow","white" };
    JComboBox list= new JComboBox(listColorNames);
    public DemoJComboBox() {
        super("Demo JComboBox : ");
        setLayout(new FlowLayout());
        list.setSelectedIndex(4);
        list.setMaximumRowCount(20);
        list.addItem("cyon");
        list.addItem("orange");
        list.insertItemAt("couleur", 3);
        add(new JScrollPane(list));
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setSize(200, 200);
        setVisible(true);
    }
    public static void main(String[] args) {
        DemoJComboBox test = new DemoJComboBox();
    }
}
```

Liste déroulante (liste combinée) : JComboBox



Les menus

- Java nous permet de doter une fenêtre de menus déroulants. Comme dans la plupart des applications du commerce, nous disposons de deux possibilités complémentaires :
  - Créer une barre de menus qui s'affiche en haut de la fenêtre, et dans laquelle chaque menu pourra faire apparaître une liste d'options.
  - Faire apparaître à un moment donné ce que nous nommons un menu surgissant (ou menu contextuel), formé quant à lui d'une seule liste d'options spécifiques correspondantes à l'endroit où se situe la demande.

Les menus

- Ces menus déroulants usuels font donc intervenir trois sortes d'objets :
  - **JMenuBar :** Un objet barre de menus.
  - **Jmenu** : Différents objets menu qui seront visibles dans la barre de menus.
  - **JMenuItem** : les options qui constituent un menu.

Création d'une barre de menus

- La création de menus est une opération assez simple. Voici la séquence à respecter avec ses différentes phases :

  - Création de la barre de menus
  - Placement de la barre de menu sur le composant désiré
  - Création des différents menus associés à la barre
  - Mise en place des différents options associées à chaque menu

Création d'une barre de menus

- Créer une barre de menus

```
JMenuBar barre = new JMenuBar();
```

- Placement de la barre de menu sur le composant désiré :

```
JFrame fenêtre = new JFrame();
fenêtre.setJMenuBar(barre);
```

- Créer et ajouter des menus

```
JMenu menu1 = new JMenu("Menu 1");
barre.add(menu1);
```

- Créer et ajouter des éléments dans les menus

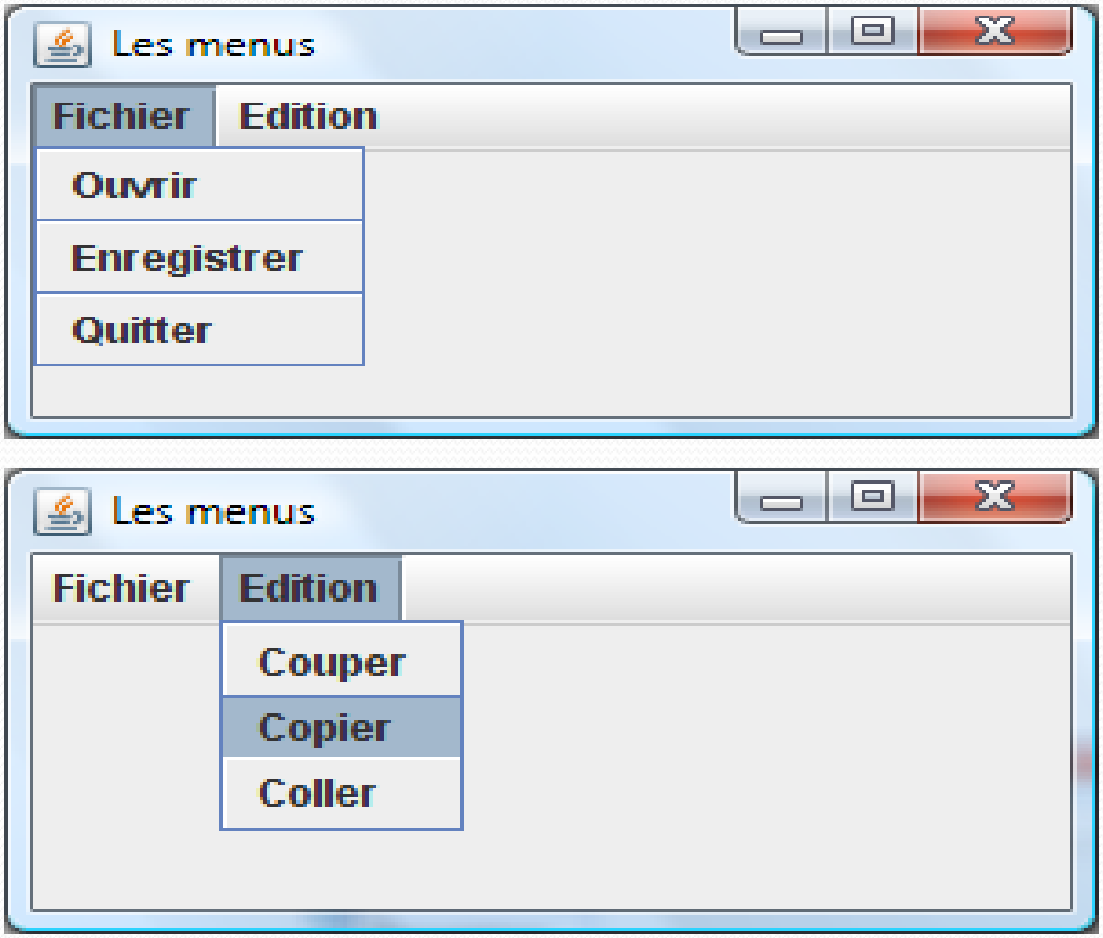
```
JMenuItem choix1 = new JMenuItem("Choix 1");
menu1.add(choix1);
```

Création d'une barre de menus

DemoJMenus.java

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

public class DemoJMenus extends JFrame {
    private JMenuBar barre = new JMenuBar();
    private JMenu fichier = new JMenu("Fichier");
    private JMenu edition = new JMenu("Edition");
    private JMenuItem ouvrir = new JMenuItem("Ouvrir");
    private JMenuItem enregistrer = new JMenuItem("Enregistrer");
    private JMenuItem quitter = new JMenuItem("Quitter");
    private JMenuItem couper = new JMenuItem("Couper");
    private JMenuItem copier = new JMenuItem("Copier");
    private JMenuItem coller = new JMenuItem("Coller");

    public DemoJMenus() {
        super("Les menus");
        setJMenuBar(barre);
        barre.add(fichier);
        barre.add(edition);
        fichier.add(ouvrir);
        fichier.addSeparator();
        fichier.add(enregistrer);
        fichier.addSeparator();
        fichier.add(quitter);
        edition.add(couper);
        edition.add(copier);
        edition.add(coller);
        setSize(300, 150);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setVisible(true);
    }
    public static void main(String[] args) { new DemoJMenus(); }
}
```

Création d'une barre de menus



Les boites de dialogue

Les boîtes de dialogue

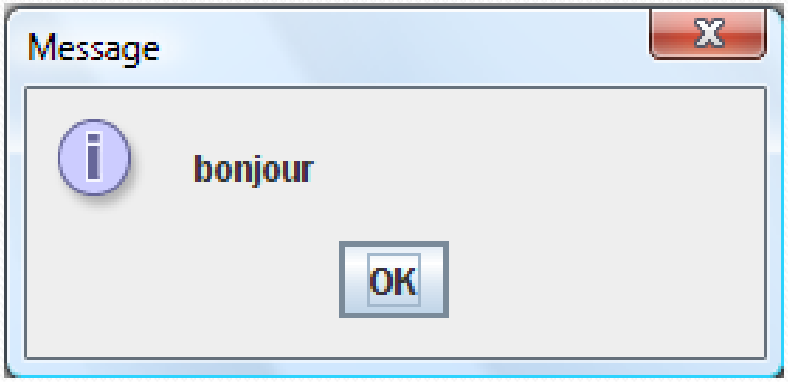
- Dans les composants Swing, il existe un composant très intéressant, le JOptionPane qui possède plusieurs méthodes statiques permettant d'ouvrir d'afficher diverses boîtes de dialogue :
  - **Les boîtes de messages:** permettent d'afficher un message pour l'utilisateur.
  - **Les boîtes de saisie :** on va pouvoir demander une chaîne de caractères à l'utilisateur
  - **Les boites de confirmation :** on peut demander une confirmation à l'utilisateur.
  - **Les boîtes de choix :** qui permettent de donner le choix entre plusieurs choses, une liste déroulante en quelque sorte

Boîtes de dialogue d'options - JOptionPane

- Dialogue de message :
  - **showMessageDialog()** : Affiche un message et attend que l'utilisateur clique sur OK (validation de lecture du message).

Exemple

```
import javax.swing.*;
public class TestDialog{
  public static void main(String[] args)
   {
    JOptionPane.showMessageDialog(null," bonjour");
    }
}
```

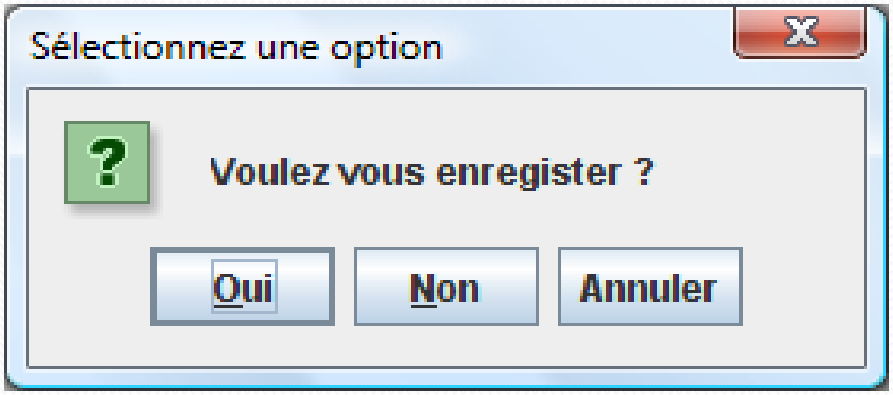


Boîtes de dialogue d'options - JOptionPane

- Dialogue de confirmation :
  - **showConfirmDialog() :** Affiche un message et attend une confirmation de la part de l'utilisateur. Les boutons de réponse sont généralement Oui, Non et Annuler.

Exemple

```
import javax.swing.*;
public class TestDialog{
  public static void main(String[] args)
   {
JOptionPane.showConfirmDialog(null," Voulez vous enregistrer");
    }
}
```

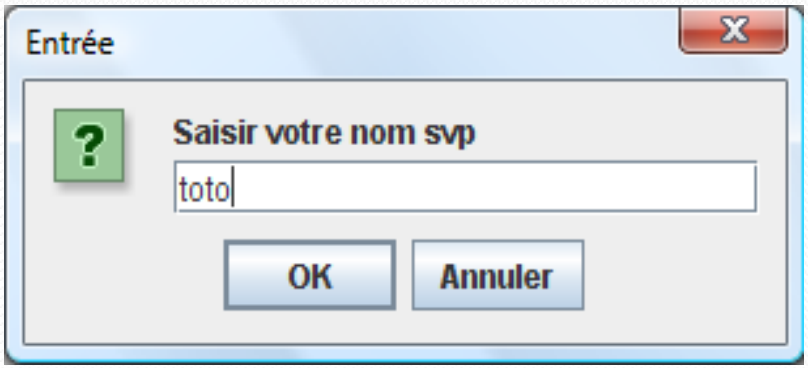


Boîtes de dialogue d'options - JOptionPane

- Dialoque de saisie :
  - **showInputDialog()** : Affiche un message et récupère une valeur saisie par l'utilisateur.

Exemple

```
import javax.swing.*;
public class TestDialog{
  public static void main(String[] args)
   {
JOptionPane.showInputDialog(null, " Saisir votre nom svp");
    }
}
```

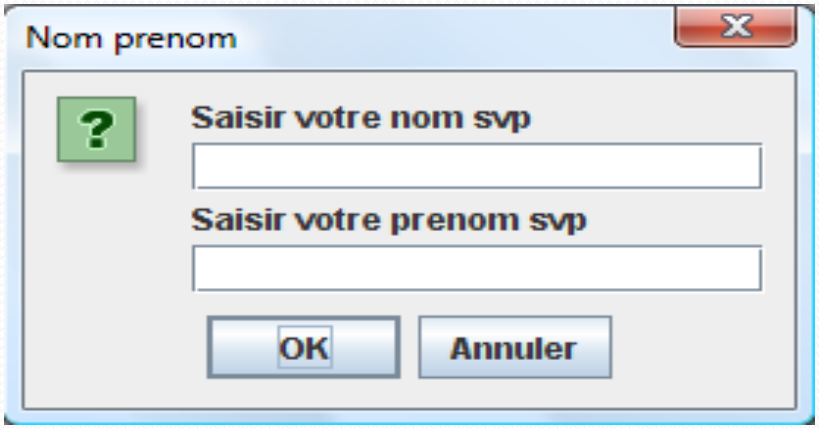


Boîtes de dialogue d'options - JOptionPane

- Dialogue d'option :
  - **showOptionDialog()** : Le type le plus général, vous lui passez vos propres composants, qui se retrouvent affichés dans le dialogue, comme des boîtes combo, un ensemble de boutons, un ensemble d'objets hétérogènes.

Exemple

```
import javax.swing.*;
public class TestDialog{
  public static void main(String[] args)
   {JTextField nom=new JTextField();
    JTextField prenom=new JTextField();
    JOptionPane.showInputDialog (null, new Object[] {"Saisir votre nom
    svp ",nom,"Saisir votre prenom svp ",prenom},"Nom prenom",
    JOptionPane.OK_CANCEL_OPTION,JOptionPane.QUESTION_MESSAGE
    ,null,null,null );
    }
}
```



Les types d'icône de boite de dialogue

JOptionPane.INFORMATION_MESSAGE : Icône d'information

JOptionPane.ERROR _MESSAGE : Icône d'erreur

JOptionPane.QUESTION_MESSAGE : Icône de question

JOptionPane.WARNING_MESSAGE : Icône d'avertissement

Création d'une instance de boîte de dialogue

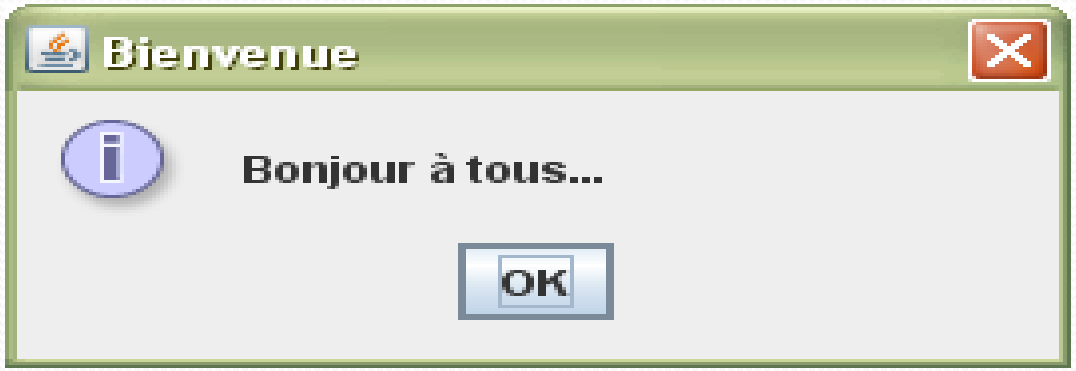
- **JOptionPane()** : Création d'une boîte d'option avec un message de test "JOptionPane message" avec un bouton OK et sans icône. Boîte de dialogue de type message.
- **JOptionPane(Object message)** : Par rapport au précédent, vous choisissez votre message. Boîte de dialogue de type message.
- **JOptionPane(Object message, int typeMessage)** : Cette fois-ci vous précisez en plus le type de message (icône correspondante). Boîte de dialogue de type message.
- **JOptionPane(Object message, int typeMessage, int typeOption)** : Par rapport au précédent, vous choisissez en plus les boutons qui feront parti de votre boîte de dialogue. Boîte de dialogue de type confirmation.

Création d'une instance de boîte de dialogue

- **JOptionPane(Object message, int typeMessage, int typeOption, Icon icône)** : Par rapport au précédent, vous choisissez en plus votre icône personnalisée. Boîte de dialogue de type confirmation.
- **JOptionPane(Object message, int typeMessage, int typeOption, Icon icône, Object[] options)** : Par rapport au précédent, vous placez en plus l'ensemble des objets qui feront parti de votre boîte de dialogue. Boîte de dialogue de type saisie ou option.

Création d'une instance de boîte de dialogue

```
import javax.swing.*;

public class Dialogue {
  public static void main(String[] args) {
    JOptionPane dialogue = new JOptionPane("Bonjour à tous", JOptionPane.INFORMATION_MESSAGE);
    JDialog boîte = dialogue.createDialog("Bienvenue");
    boîte.setVisible(true);
  }
}
```



Les boîtes de message

- **static void showMessageDialog(Component composantParent, Object message)** : Affiche automatiquement une boîte de dialogue de type message, dont le titre est intitulé "Message", avec une icône de type information et avec un bouton OK. Vous spécifiez éventuellement en argument la fenêtre parente et surtout le contenu de votre message.

- **static void showMessageDialog(Component composantParent, Object message, String titre, int typeMessage)** : Même boîte de message que précédemment, mais cette fois-ci, vous pouvez choisir votre titre et votre icône.

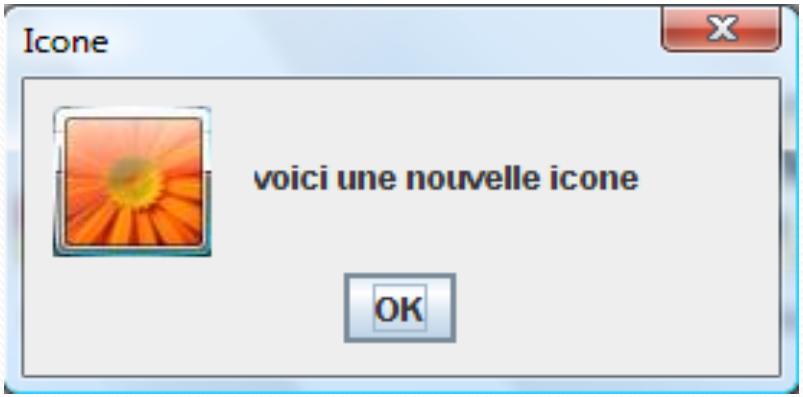
- **static void showMessageDialog(Component composantParent, Object message, String titre, int typeMessage, Icon icône)** : Par rapport à la précédente, vous pouvez placer en plus une icône personnalisée.

Les boîtes de message

- L'argument composantParent spécifie le composant servant de parent au dialogue.
- L'argument titre spécifie une chaîne apparaissant dans la barre de tittre.
- L'argument message est déclaré comme un Object. Nous lui passons généralement une valeur de type String, qui est automatiquement affichée dans un JLabel. Nous pouvons cependant spécifier aussi une icône type Icon qui est également affichée par le JLabel, ou un composant graphique quelconque de JComponent, qui est affiché tel quel. De plus, au lieu de donner un seul objet message, nous pouvons spécifier un tableau d'objets contenant n'importe quelle combinaison de chaînes, d'icônes et de composants.
- L'argument typeMessage doit être choisi parmi les constantes
- Pour surcharger l'icône par défaut du dialogue, nous pouvons également spécifier une icône personnalisée au paramètre icône.

Exemple

```
import javax.swing.*;
public class TestDialog1 {

    public static void main(String[] args) {
        JOptionPane.showMessageDialog(null,"voici une nouvelle icone",
        "Icone",JOptionPane.INFORMATION_MESSAGE,new  ImageIcon("C:/img1.jpg"));
    }
}
```



Les boîtes de confirmation

- Java permet d'afficher des boîtes dites "de confirmation" offrant à l'utilisateur un choix de type Oui/Non. Il suffit pour cela de recourir à l'une des variantes de la méthode statique showConfirmDialog() de la classe JOptionPane .

Les boîtes de confirmation

- L'argument typeOption indique quel bouton doit apparaître dans le dialogue et vous permet de choisir parmi les quatres constantes suivantes :
  - **JOptionPane.DEFAULT_OPTION** : fournit un unique bouton OK.
  - **JOptionPane.YES_NO_OPTION** : fournit les boutons Oui et Non
  - **JOptionPane.YES_NO_CANCEL_OPTION** : fournit les boutons Oui, Non et Annuler (valeur par défaut).
  - **JOptionPane.OK_CANCEL_OPTION :** fournit les boutons OK et Annuler.

Les boîtes de confirmation

- Comme les dialogues de confirmation présente des choix à l'utilisateur, ils donnent une valeur de retour reflétant le choix de l'utilisateur. Cette valeur est l'une des constantes suivantes :

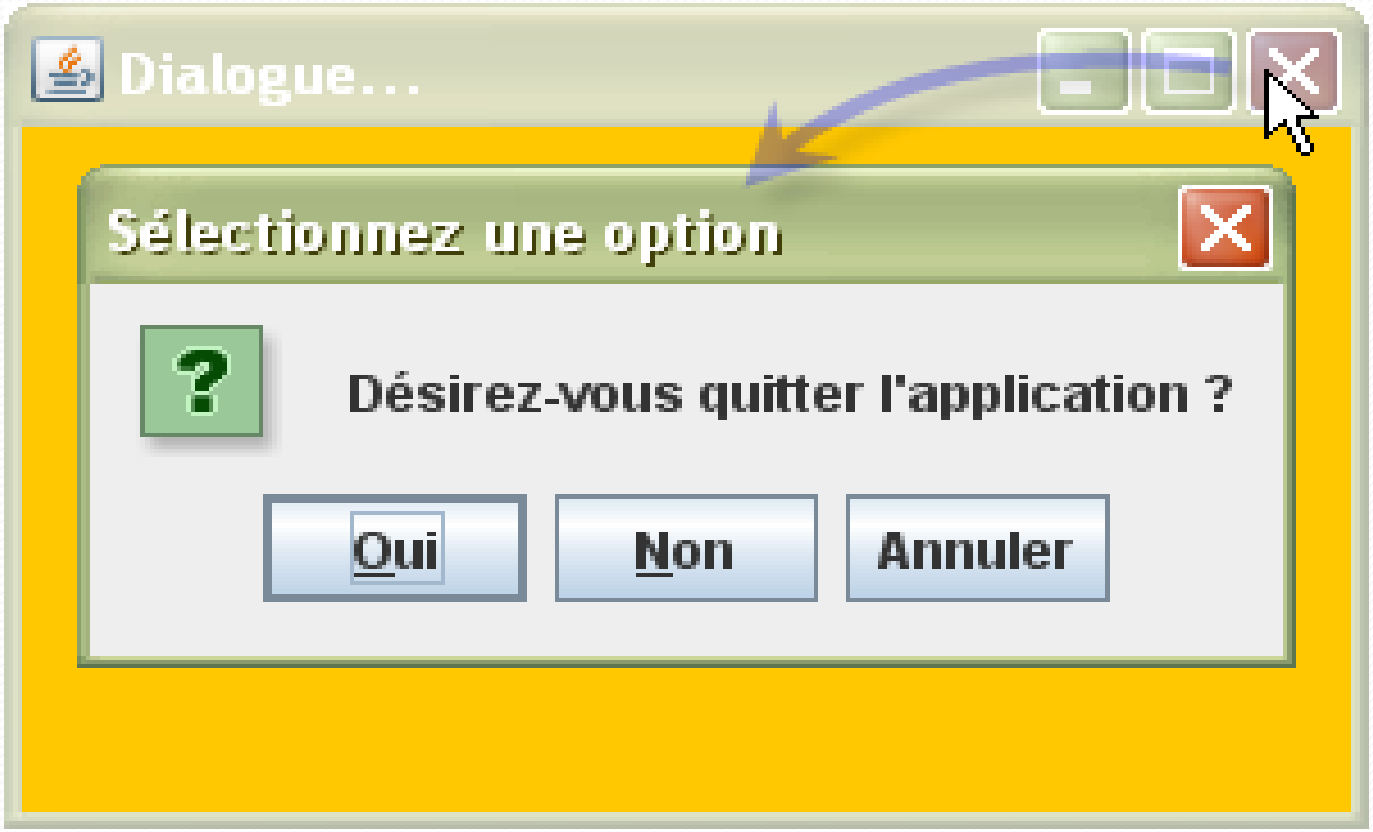
  - **JOptionPane.OK_OPTION.**
  - **JOptionPane.CANCEL_OPTION.**
  - **JOptionPane.YES_OPTION.**
  - **JOptionPane.NO_OPTION.**
  - **JOptionPane.CLOSED_OPTION.**

Les boîtes de confirmation

Start Page | Dialogue.java

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

public class Dialogue extends JFrame {
  public Dialogue() {
    super("Dialogue...");
    getContentPane().setBackground(Color.ORANGE);
    setSize(300, 200);
    addWindowListener(new WindowAdapter() {

        public void windowClosing(WindowEvent e) {
          if (JOptionPane.showConfirmDialog(Dialogue.this, "Désirez-vous quitter l'application ?")
             == JOptionPane.YES_OPTION) System.exit(0);
        }
      });
    setDefaultCloseOperation(DO_NOTHING_ON_CLOSE);
    setVisible(true);
  }

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

Les boîtes de confirmation



Les boîtes de saisie

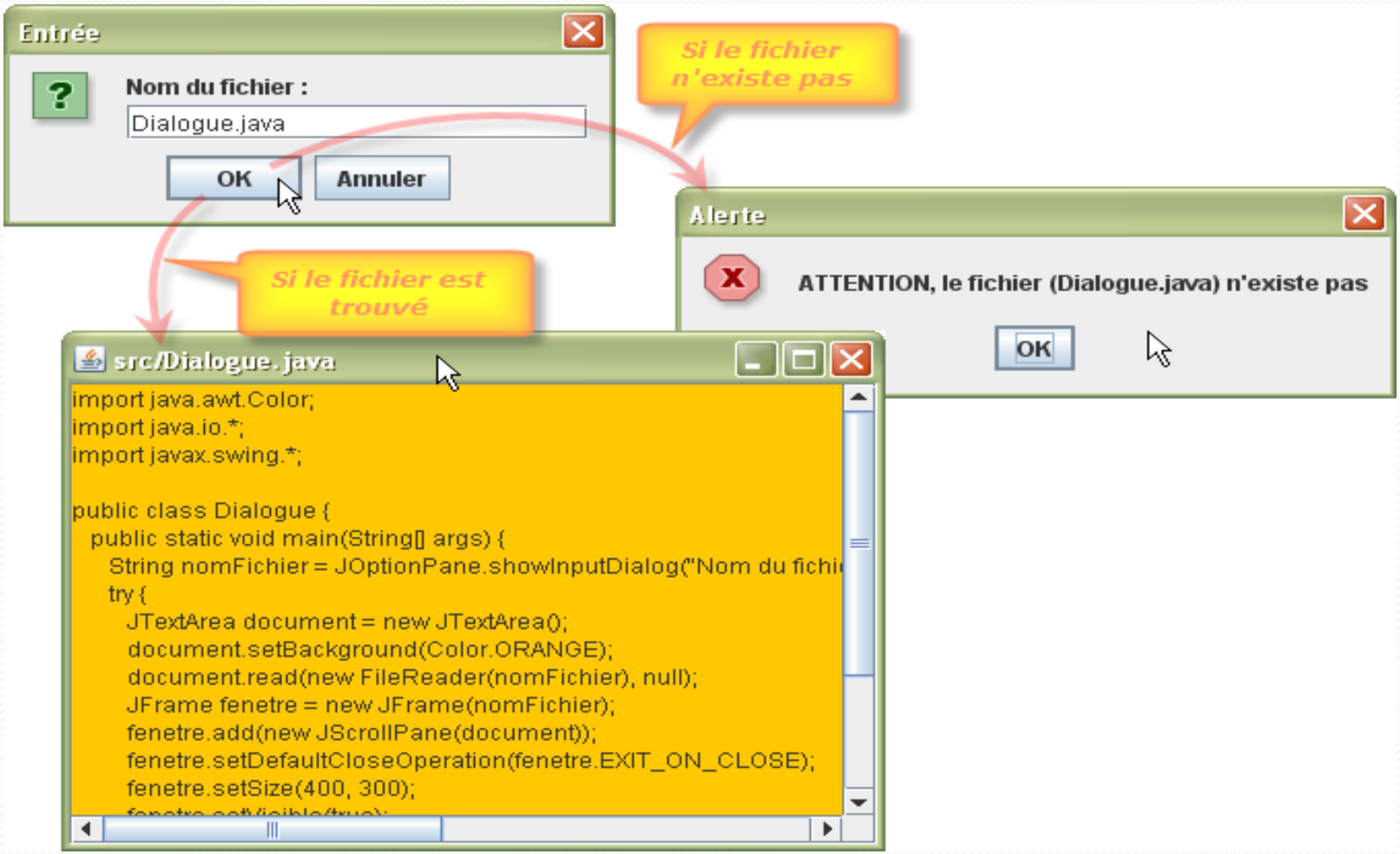
- La boîte de saisie permet à l'utilisateur de fournir une information sous la forme d'une chaîne de caractères. La méthode statique showInputDialog() de la classe JOptionPane vous permet de gérer automatiquement le dialogue avec l'utilisateur.
- Ces dialogues sont modaux et les méthodes qui les affichent bloquent jusqu'à ce que l'utilisateur ferme le dialogue.
  - S'il le ferme avec le bouton OK, les méthodes renvoient ce que l'utilisateur a saisie sous forme de String.
  - Si l'utilisateur n'a pas confirmé sa saisie par OK, autrement dit s'il a agit sur Annuler ou s'il a fermé la boîte de saisie (et ce même s'il a commencé à entrer une information dans le champ de texte), ces méthodes renvoient la valeur null.

Les boîtes de saisie

Dialogue1.java *

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

public class Dialogue1 {
  public static void main(String[] args) {
    String nomFichier = JOptionPane.showInputDialog("Nom du fichier :");
    try {
      JTextArea document = new JTextArea();
      document.setBackground(Color.PINK);
      document.read(new FileReader(nomFichier), null);
      JFrame fenetre = new JFrame(nomFichier);
      fenetre.add(new JScrollPane(document));
      fenetre.setDefaultCloseOperation(fenetre.EXIT_ON_CLOSE);
      fenetre.setSize(400, 300);
      fenetre.setVisible(true);
    }
    catch (IOException ex) {
      JOptionPane.showMessageDialog(null,
        "ATTENTION, le fichier ("+nomFichier+") n'existe pas",
        "Alerte",
        JOptionPane.ERROR_MESSAGE);
    }
  }
}
```

Les boîtes de saisie



Les boîtes d'options

- Cette boîte d'options, représentée par la méthode statique **showOptionDialog()**, généralise en fait la boîte de dialogue de confirmation. Il existe effectivement un argument supplémentaire, appelé options, qui indique quels boutons afficher dans la boîte de dialogue.
- **static Object showOptionDialog(Component composantParent, Object message, String titre, int typeOption, int typeMessage, Icon icône, Object[] options, Object valeurInitiale)** : Boîte de dialogue de type option. c'est l'argument options au travers duquel vous passez l'ensemble des éléments qui constituerons les boutons de la boîte de dialogue, sous forme de tableaux d'objets.

Les boîtes d'options

Dialogue2.java

```
import javax.swing.*;

public class Dialogue2 {
  public static void main(String[] args) {
    JTextField utilisateur = new JTextField();
    JPasswordField motDePasse = new JPasswordField();
    int choix = JOptionPane.showOptionDialog(null,
      new Object[] {"Votre nom :", utilisateur, "Mot de passe :", motDePasse},
      "Connexion",
      JOptionPane.OK_CANCEL_OPTION,
      JOptionPane.QUESTION_MESSAGE, null,  null, null);

    if (choix == JOptionPane.OK_OPTION)
      if (utilisateur.getText().equalsIgnoreCase("fatima JEBBAR") && motDePasse.getText().equals("fatimajebbar"))
         JOptionPane.showMessageDialog(null, "Vous êtes connecté");
      else
         JOptionPane.showMessageDialog(null,
           new String[] {"Utilisateur inconnu", "ou mot de passe incorrecte"},
           "Connexion refusée",
           JOptionPane.ERROR_MESSAGE);
    else
      JOptionPane.showMessageDialog(null, "Non connecté...", "ATTENTION", JOptionPane.ERROR_MESSAGE);
  }
}
```

Les boîtes d'options