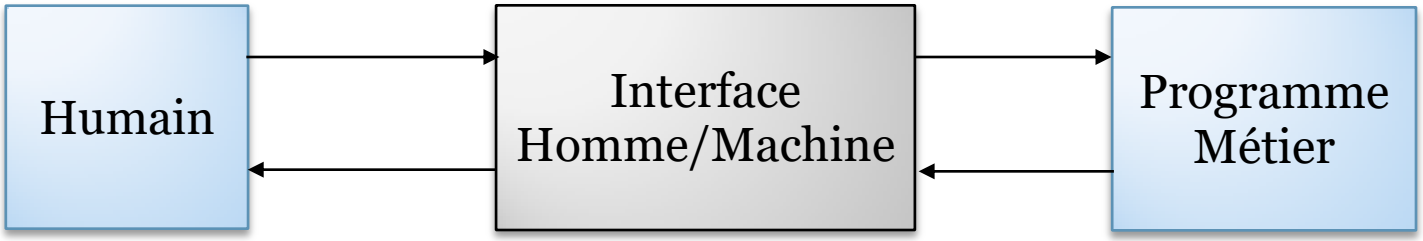
Chapitre :
Les interfaces graphiques en JAVA

Interface avec l'utilisateur

- La quasi-totalité des programmes informatiques nécessitent:
  - l'affichage de questions posées à l'utilisateur
  - l'entrée de données par l'utilisateur
  - l'affichage 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 *Graphical User Interface*.



API utilisées pour les interfaces graphiques en Java

Les API

- bibliothèques :
  - AWT (*Abstract Window Toolkit, JDK 1.1)*
  - Swing (JDK 1.2)
- Swing et AWT font partie de JFC (*Java Foundation Classes) qui offre des facilités pour* construire des interfaces graphiques
- Swing est construit au-dessus de AWT
  - même gestion des événements
  - les classes de *Swing héritent des classes de AWT*

API utilisées pour les interfaces graphiques en Java

Swing ou AWT ?

- AWT (Abstract Window Toolkit)
  - Composants graphiques « lourds »
  - Chaque composant est relié à son équivalent dans l'OS par un « peer »
  - Look & Feel dépendant de l'OS
- SWING
  - Nouveau package
  - Composants graphiques « légers », en pur Java
  - Tous les composants sont détachés de l'OS
  - Look & Feel indépendant de l'OS

API utilisées pour les interfaces graphiques en Java

Swing ou AWT ?

- Tous les composants de AWT ont leur équivalent dans Swing
  - En plus joli
  - avec plus de fonctionnalités
- Swing offre de nombreux composants qui n'existent pas dans AWT
  - Il est fortement conseillé d'utiliser les composants Swing

Swing est plus lourd et plus lent que AWT

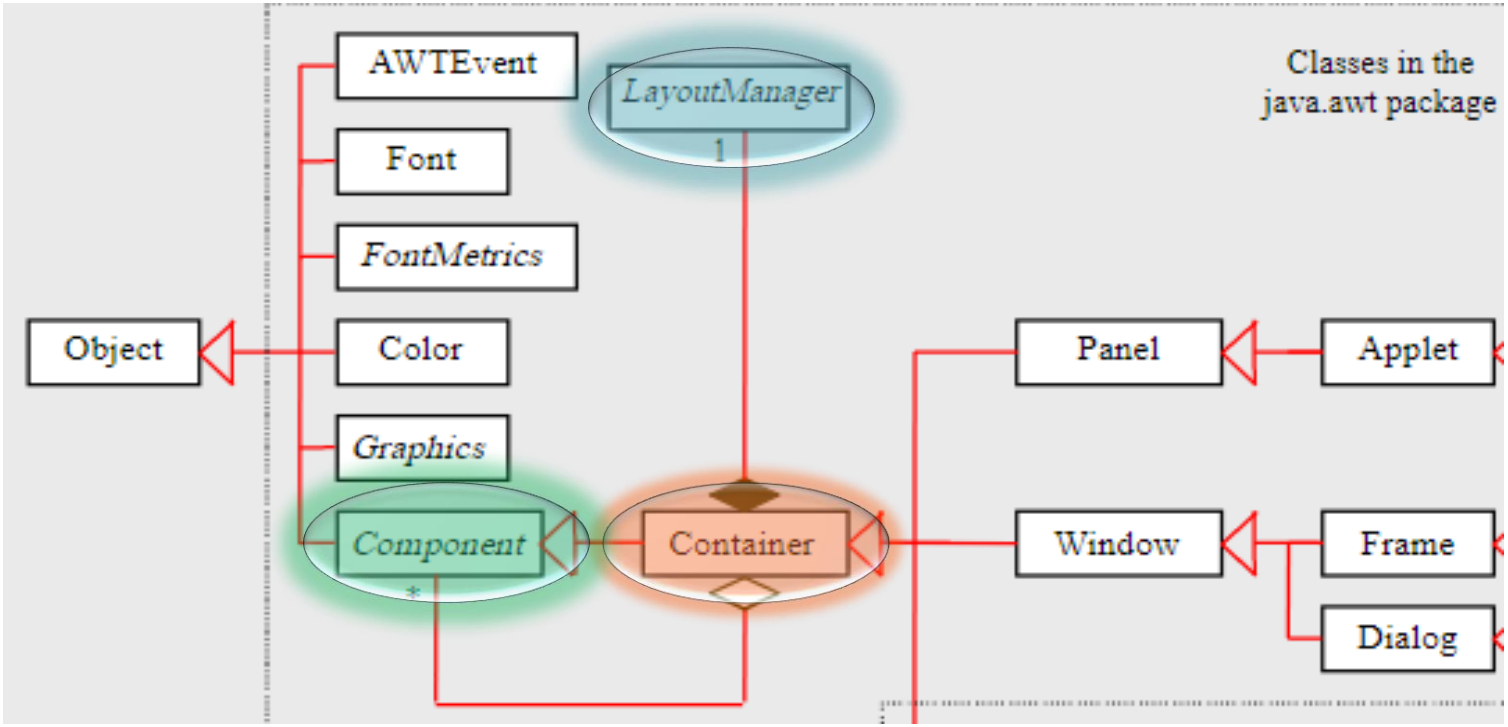
API utilisées pour les interfaces graphiques en Java

- AWT et Swing utilise des classes différentes pour représenter chaque composant graphique.
- Tous les composants ont une classe racine commune (cette classe n'est pas abstraite !!)
  - La classe représentant les composants est :
    - pour l'AWT : `Component`
    - pour Swing : `Jcomponent`
  - Notons que tous les noms des composants graphiques de Swing commence par `J...`

Composant et hiérarchie de composants

Structure de l'AWT

- L'AWT offre trois types d'éléments graphiques
  - Les « Containers » (contenants)
  - Les « Components » (composants ou contenus)
  - Les « LayoutManagers » (disposition des objets d'un contenant)



Composant et hiérarchie de composants

Structure de l'AWT

Les « Containers »

Sont destinés à accueillir des composants

Gèrent l'affichage des composants

Ex: Frame, Panel, Window

Les « Components »

Constituent différents éléments de l'affichage (boutons, barres de menus, etc.)

Ex: Button, Canvas, Label, Scrollbar, Checkbox

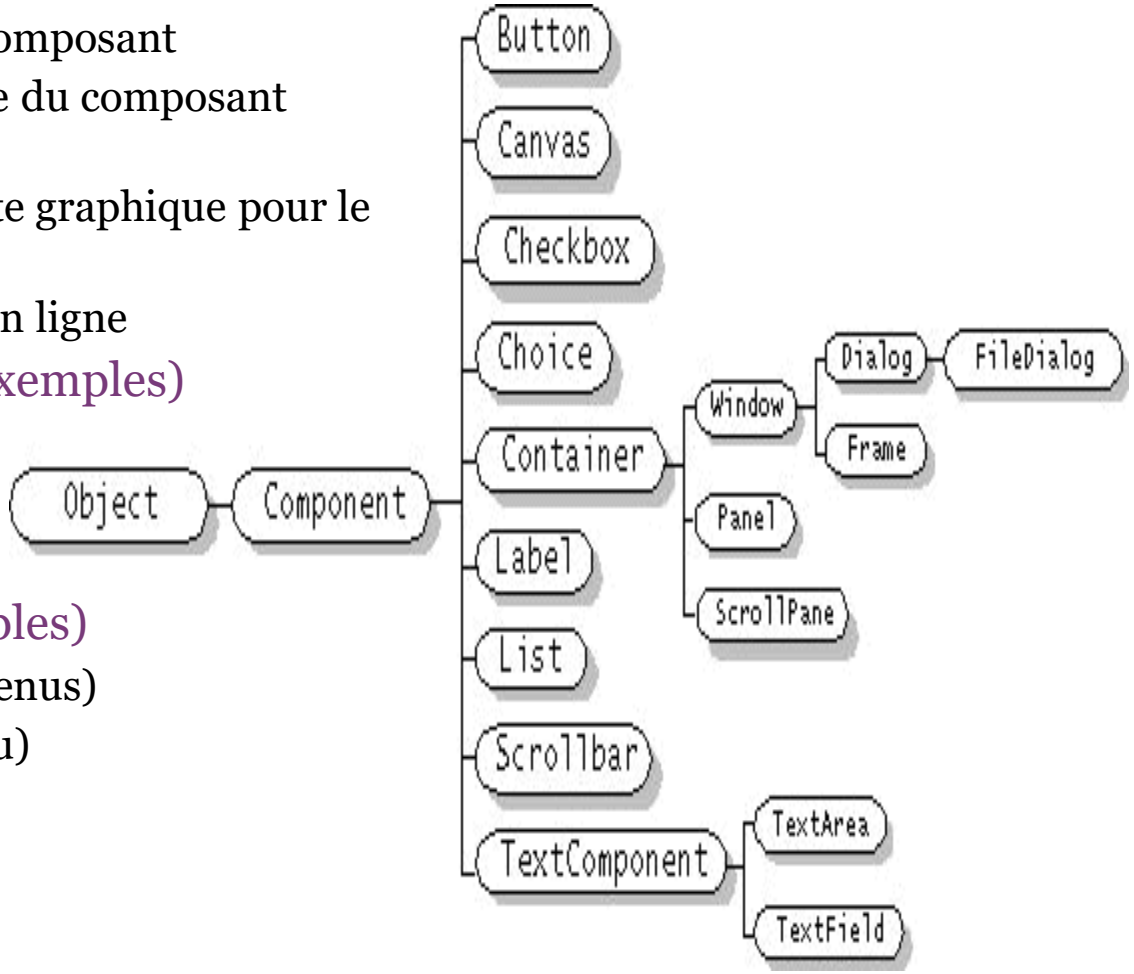
Les « LayoutManagers »

Gèrent la disposition des composants au sein d'un conteneur.

Composant et hiérarchie de composants: AWT

Les « Components »

- Héritage de méthodes:
  - paint(Graphics g) : Affiche le composant
  - repaint() : Rafraîchit l'affichage du composant (rappelle la méthode paint)
  - getGraphics() : Crée un contexte graphique pour le composant
  - Etc. voir documentation Java en ligne
- Composants de formulaires (exemples)
  - Button (bouton)
  - CheckBox (case à cocher)
  - Label (case de texte)
- Composants de fenêtre (exemples)
  - Menu (Menu d'une barre de menus)
  - MenuItem (Elément d'un menu)



Composant et hiérarchie de composants: AWT

Les « Containers »

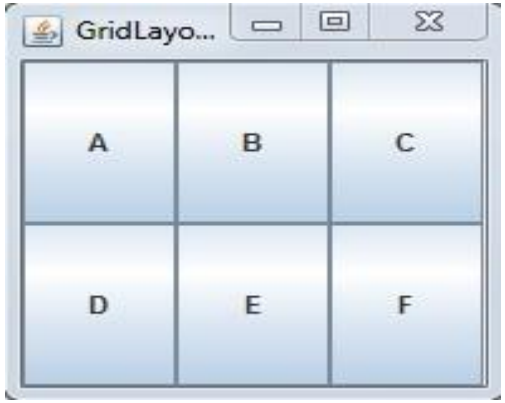
- Héritage de méthodes:
  - add(Component c) : Intègre le composant spécifié à la fin du container
  - setLayout(LayoutManager l) : Configure le LayoutManager du container
  - Etc. voir documentation Java en ligne
- La classe « Frame »
  - Composant du plus haut niveau
  - La fenêtre d'une application est une instance de cette classe
  - Le Frame contient les différents composants graphiques de l'application
  - Ne peut être intégré dans un autre conteneur
- Les classes « Panel », « Window », « ScrollPane », etc.
  - Contenants essentiels
  - Peuvent être intégrés au sein d'un Frame

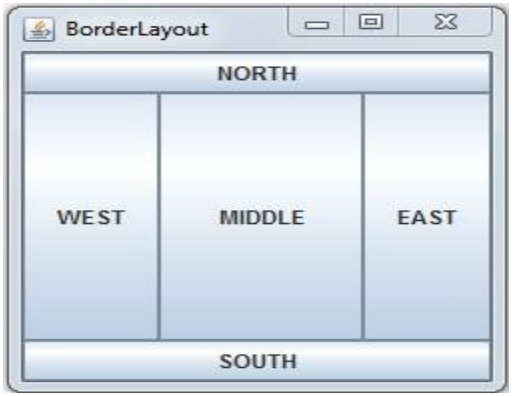
Composant et hiérarchie de composants: AWT

Les « LayoutManagers »

- Rôle
  - Gérer la disposition des composants au sein d'un conteneur

- Types principaux:
  - **BorderLayout:** divise le conteneur en 5 zones
  - **FlowLayout**: rajoute les composants au fur et à mesure
  - **GridLayout**: applique une grille au conteneur pour aligner les composants
  - **CardLayout**: pour un conteneur qui contient plusieurs cartes
  - **GridBagLayout**: grille de cellules élémentaires



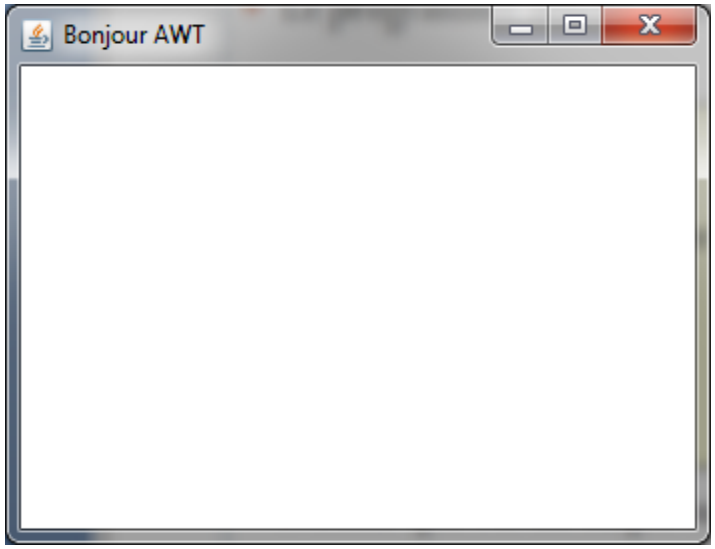


Composant et hiérarchie de composants : AWT

HelloWorld avec AWT

- Le programme affiche une fenêtre ayant pour titre « **`Bonjour AWT`** ».

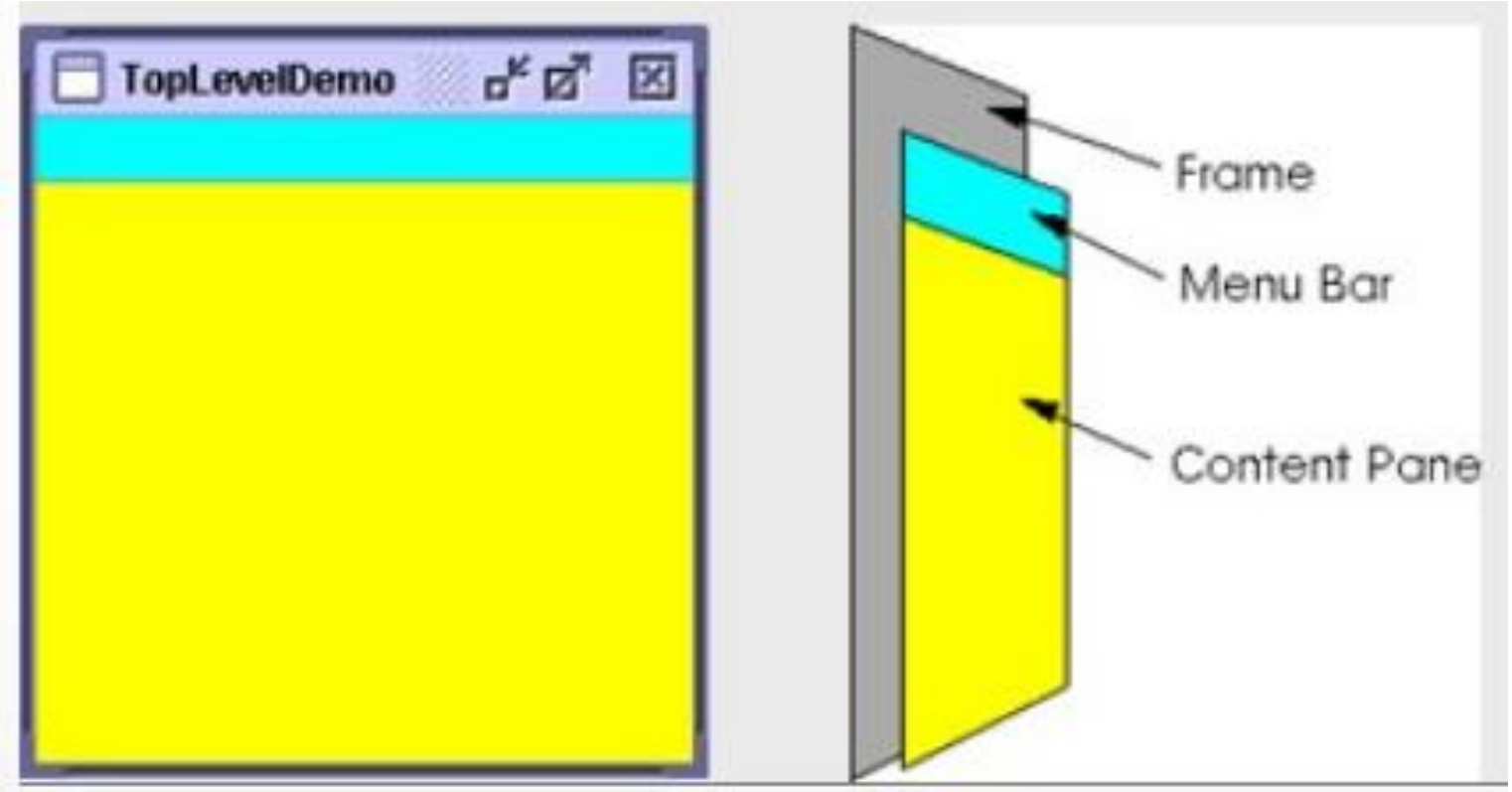
```
import java.awt.Frame;

public class HelloAWT {
  public static void main(String[] args) {
    Frame frame = new Frame();
    frame.setTitle("Bonjour AWT");
    frame.setSize(400,300);
    frame.setVisible(true);
  }
}
```



Notons que s'il on clique sur la croix, la fenêtre devient non visible mais l'application continue.

Hiérarchie de composants

- Les composants sont organisés sous la forme d'un arbre

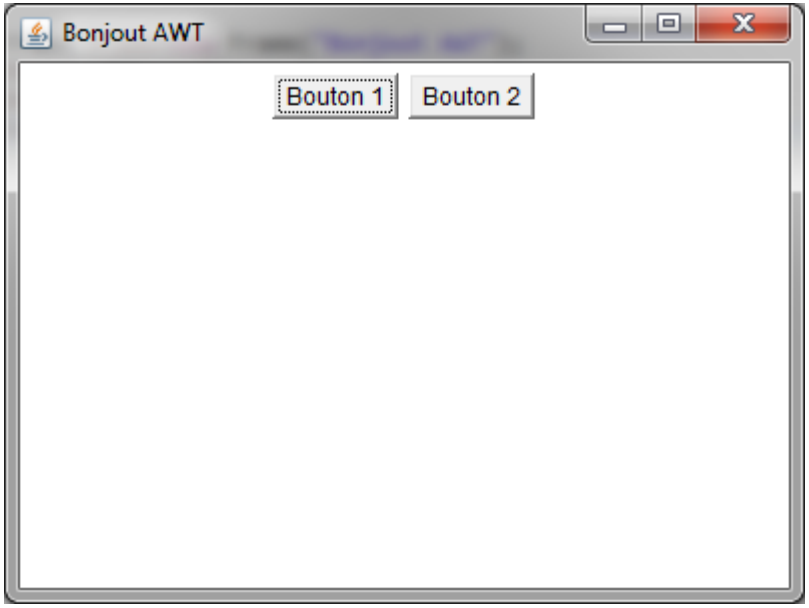


Composant et hiérarchie de composants : AWT

Hiérarchie de composants

- La méthode **`add()`** permet d'ajouter un composant à un container
- **`Panel`** est un sous-type de **`Container`**

```
import java.awt.*;
public class Hierarchy {
  public static void main(String[] args) {
    Button bouton1=new Button("bouton 1");
    Button bouton2=new Button("bouton 2");

    Panel panel=new Panel();
    panel.add(bouton1);
    panel.add(bouton2);

    Frame frame=new Frame("Bonjour AWT");
    frame.add(panel);
    frame.setSize(400,300);
    frame.setVisible(true);
  }
}
```



Hiérarchie de composants

Chaque composant possède un état indiquant s'il est visible ou non (`is/setVisible()`)

Un changement d'état de la visibilité se propage sur l'ensemble des composants fils du composant.

Les composants **`Frame`**, **`Dialog`**, **`Window`**, **`Applet`** possédent aussi la méthode **`setVisible()`**

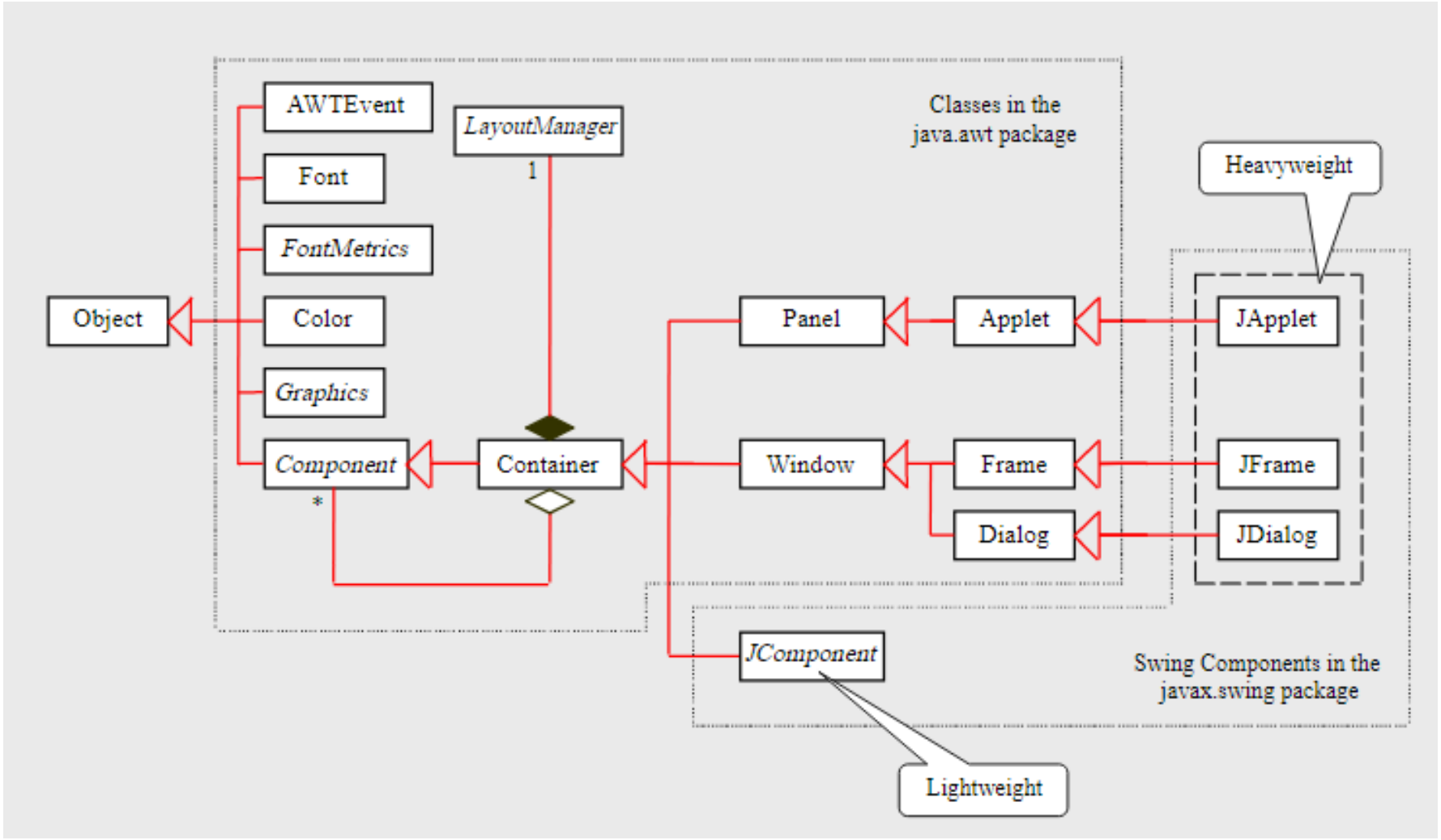
```
import java.awt.*;
public class Hierarchy {
  public static void main(String[] args) {
    Button bouton1=new Button("bouton 1");
    Button bouton2=new Button("bouton 2");

    Panel panel=new Panel();
    panel.add(bouton1);
    panel.add(bouton2);

    Frame frame=new Frame("Bonjour AWT");
    frame.add(panel);
    frame.setSize(400,300);
    frame.setVisible(true);
  }
}
```

Relation entre Swing et AWT

- Swing utilise l'AWT pour ouvrir une fenêtre
- Il y a deux types de composants Swing
  - les **heavyweight** : gérés par l'AWT, correspondent à des fenêtres de la plateforme : `JFrame`, `JDialog`, `JWindow`, `JApplet`
  - les **lightweight** : composants Java qui effectuent le dessin
- Tous les composants AWT sont heavyweights

Composant et hiérarchie de composants

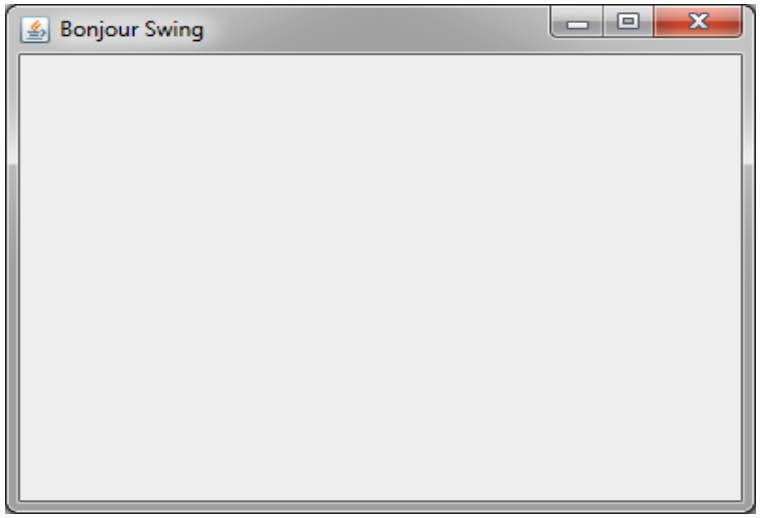
Relation entre Swing et AWT



HelloWorld avec Swing

- Le programme affiche une fenêtre ayant pour titre «Bonjour Swing ».

```
import javax.swing.JFrame;
public class HelloSwing {
    public static void main(String[] args)
    {
        JFrame frame = new JFrame();
        frame.setTitle("Bonjour Swing");
        frame.setSize(400, 300);
        frame.setVisible(true);
    }
}
```

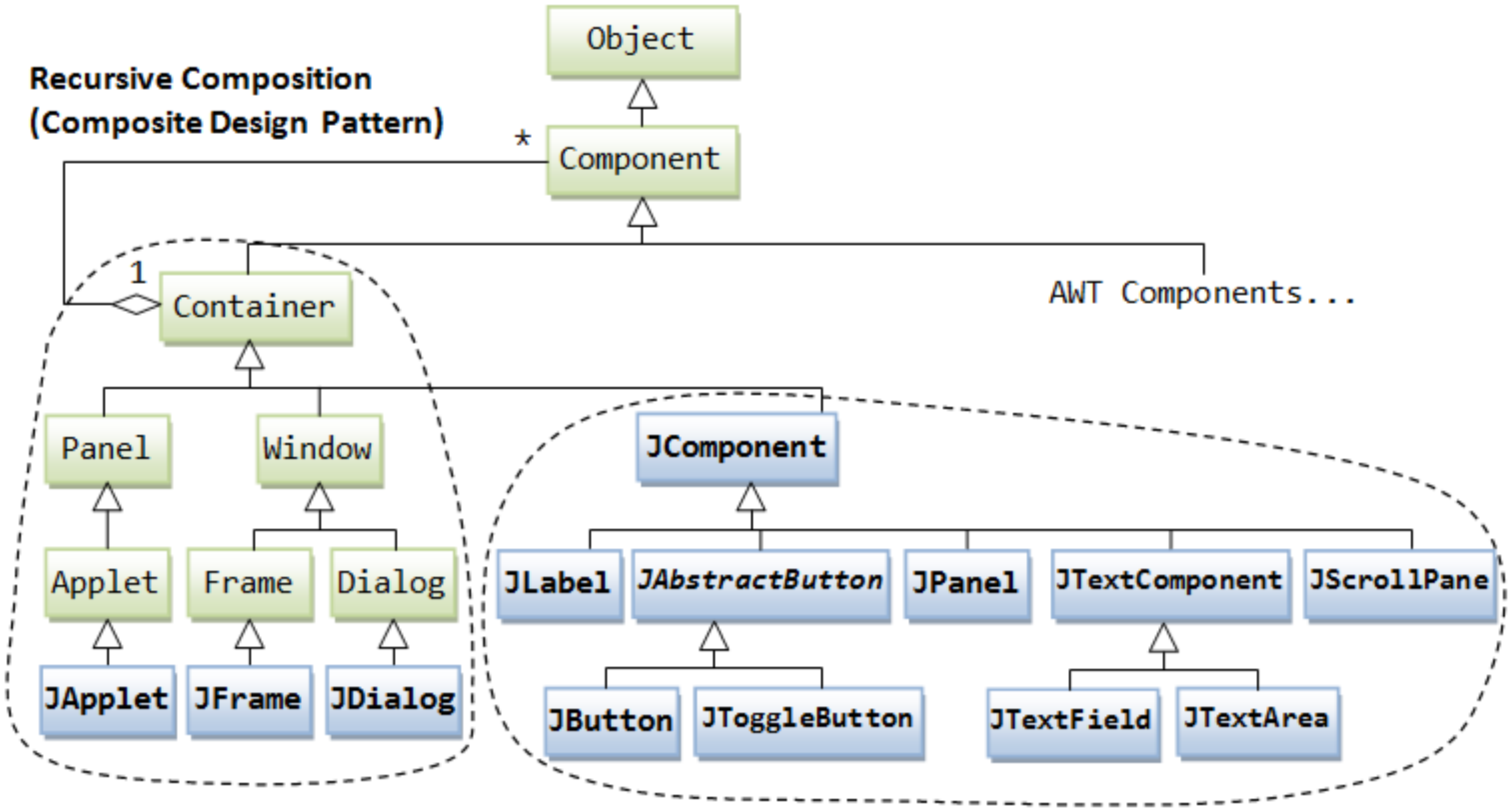


Notons que s'il on clique sur la croix, l'application ne se ferme toujours pas.

Composant et hiérarchie de composants : SWING

Composants de Swing

- Les composants de Swing partagent une partie de leur implantation avec ceux de l'AWT
- Il y a beaucoup plus de composants Swing que de composants AWT



Composant et hiérarchie de composants : SWING

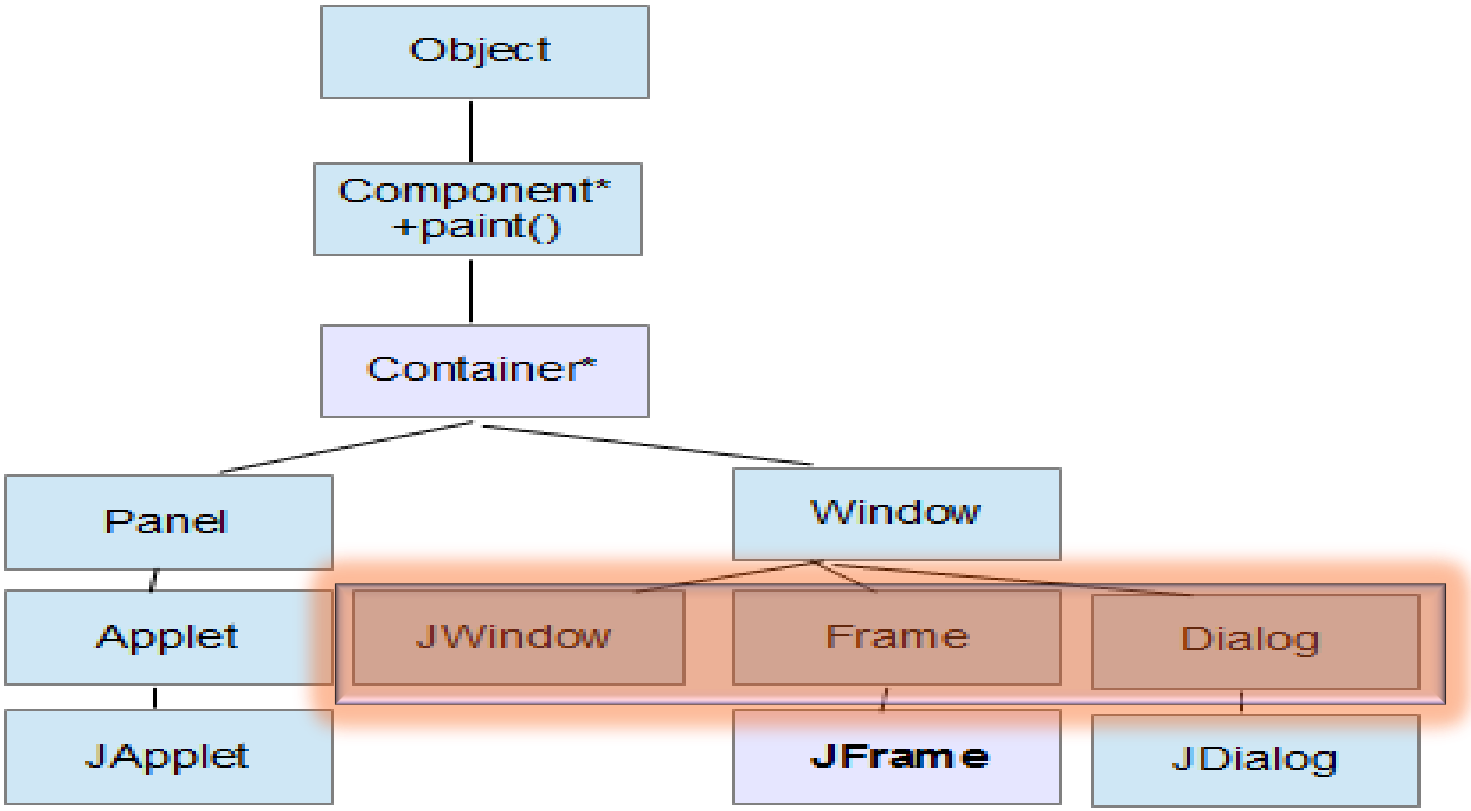
Composants de Swing

- Problème de design n'importe quel **`JComponent`** est un **`Container`**
- Le code suivant est donc possible :

```
import javax.swing.*;

public class PbDesign {
  public static void main(String[] args) {
    JFrame frame=new JFrame();

    JButton bouton1=new JButton("bouton 1");
    JButton bouton2=new JButton("bouton 2");
    bouton1.add(bouton2);

    frame.setContentPane(bouton1);
    frame.setTitle("Bonjour Swing");
    frame.setSize(400,300);
    frame.setVisible(true);
  }
}
```

Les fenêtres

- Il existe plusieurs types de fenêtres dans Swing :



Les fenêtres

- Swing propose deux types de fenêtres : **`JWindow`** et **`JFrame`**.
  - Composants proches et descendent **`Window`** (AWT).
- Constructeur est appelé pour créer une fenêtre
- Par défaut, une fenêtre crée n'est pas affichée
  - La méthode **`setVisible`** permet d'afficher une fenêtre.
- La taille d'une fenêtre dépend des éléments qu'elle contient.
- Pour éviter d'estimer ou de la calculer la taille, Swing propose une méthode (**`pack`**) qui calcule la taille de la fenêtre en fonction de la taille préférée de ses composants internes.
- Lors de l'appel de la méthode **`pack`**, la méthode **`getPreferredSize`** est appelée sur tous les composants pour connaître leurs dimensions.
  - Ces informations sont utilisées pour calculer la dimension de la fenêtre.
- La méthode **`setLocation`** permet de positionner le composant à l'intérieur du conteneur.
- L'écran est le conteneur des **`JFrame`** et des **`JWindow`**

Les fenêtres

JWindow

- 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, 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.

```
import javax.swing.JWindow;

public class TestJWindow {
     public static void main(String[] args) {
          JWindow fenetre = new JWindow();
          fenetre.setSize(300, 300);
          fenetre.setLocation(300, 300);
          fenetre.setVisible(true);
     }
}
```

Les fenêtres

JFrame

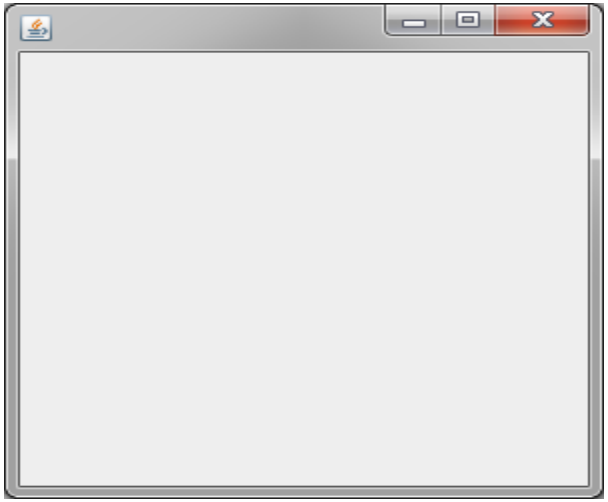
- C'est une fenêtre destinée à être la fenêtre principale de votre application.
- Elle n'est dépendante d'aucune autre fenêtre et ne peut pas être modale.
- La plupart des applications sont construites à partir d'une (ou plusieurs) **`JFrame`**.

  ➔ En effet, **`JFrame`** construit des fenêtres qui comportent une bordure, un titre, des icônes et éventuellement un menu.
- La position des icônes et la police du titre dépend donc du système.
- Par contre, l'icône représentant la fenêtre lorsqu'elle est réduite peut être modifié en utilisant la méthode **`setIconImage`**.

Les fenêtres

JFrame

```
import javax.swing.JFrame;

public class TestJFrame {
     public static void main(String[] args) {
          JFrame fenetre = new JFrame();
          fenetre.setSize(300, 300);
          fenetre.setVisible(true);
          fenetre.setLocation(500, 500);
     }
}
```

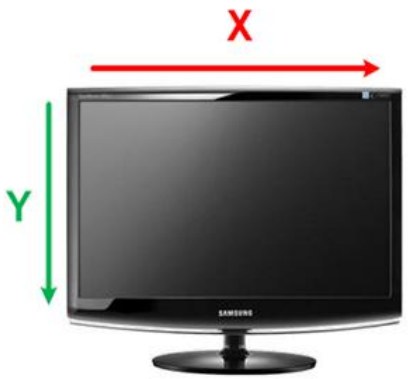


Exemple 1(Jframe) document Exemples-GUI

Les fenêtres

JFrame

- la méthode `setLocation(int x, int y):` spécifier où doit se situer votre fenêtre sur l'écran. Les coordonnées, exprimées en pixels, sont basées sur un repère dont l'origine est représentée par le coin supérieur gauche.

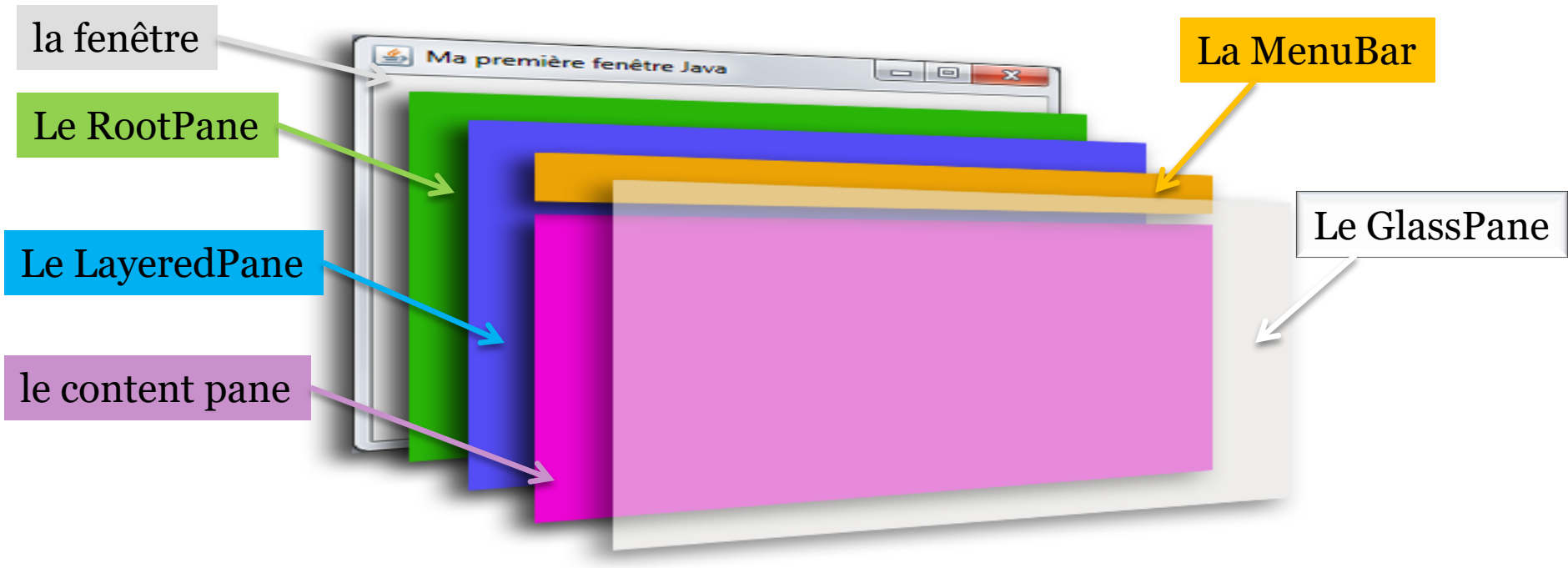


- La méthode `setResizable(boolean b):false` empêche le redimensionnement tandis que `true` l'autorise.
- `setAlwaysOnTop(boolean b):true` laissera la fenêtre au premier plan.

Les fenêtres

JFrame

Une Jframe est découpée en plusieurs parties superposées:



Conteneur principal qui contient les autres composants.

Forme juste un panneau composé du conteneur global et de la barre de menu .

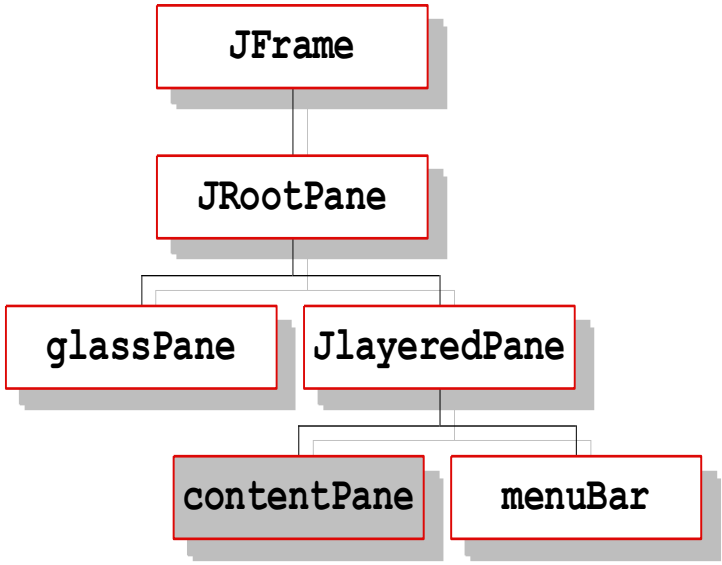
C'est dans celui-ci que nous placerons nos composants ;

Couche utilisée pour intercepter les actions de l'utilisateur avant qu'elles ne parviennent aux composants.

Les fenêtres

JFrame

- Une **`JFrame`** contient une fille unique, de la classe **`JRootPane`**
- Cette fille contient deux fils, **`glassPane`** (**`JPanel`**) et **`layeredPane`** (**`JLayeredPane`**)
- La layeredPane a deux fils, **`contentPane`** (un **`Container`**) et **`menuBar`** (un **`JMenuBar`**)
- On travaille dans **`contentPane`**.
- **`JApplet`**, **`JWindow`** et **`JDialog`** utilisent aussi un **`JRootPane`**.

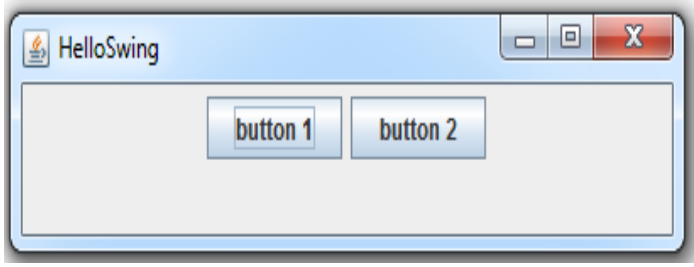


Les fenêtres

Utilisations du contentPane

JFrame

- Il est possible :
  - de demander le `contentPane` (`getContentPane()`)
  - de changer de `contentPane` (`setContentPane()`)

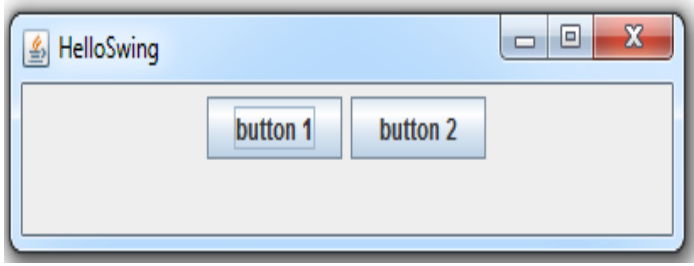



```
import java.awt.*;
import javax.swing.*;

public class HierarchySwing {

  public static void main(String[] args) {

    JFrame frame=new JFrame("HelloSwing");
    Container c=frame.getContentPane();
    c.setLayout(new FlowLayout());

    JButton button1=new JButton("button 1");
    c.add(button1);
    JButton button2=new JButton("button 2");
    c.add(button2);

    frame.setSize(400,100);
    frame.setVisible(true);
  }
}
```

```
import javax.swing.*;

public class HierarchySwing2 {

  public static void main(String[] args) {

    JButton button1 = new JButton("button 1");
    JButton button2 = new JButton("button 2");

    Jpanel panel = new JPanel();
    panel.add(button1);
    panel.add(button2);

    JFrame frame = new JFrame("HelloSwing");
    frame.setContentPane(panel);

    frame.setSize(400, 100);
    frame.setVisible(true);
  }
}
```

Utilisations du contentPane

JFrame



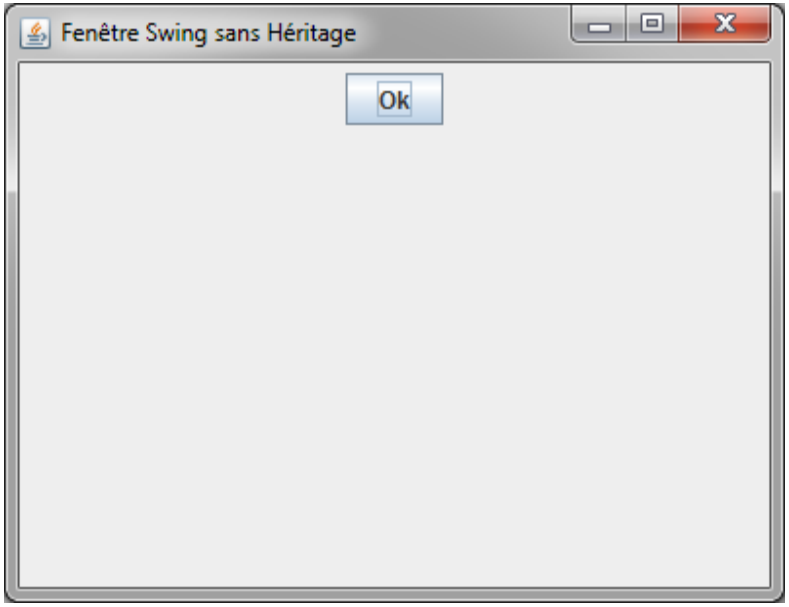
- Normalement, l'ajout de composants se fait sur le `contentPane` et non sur la `JFrame`
- Mais, si l'on effectue un `add()` sur une `JFrame` :
  - En 1.4 et avant, il y a une erreur à l'exécution
  - En 1.5, est équivalent à `getContentPane().add()`
- Ce mécanisme marche pour les méthodes : `add/remove/setLayout`

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

public class HierarchySwing3 {
  public static void main(String[] args) {

    JFrame frame=new JFrame("HelloSwing");
    frame.setLayout(new FlowLayout());

    JButton button1=new JButton("button 1");
    frame.add(button1);
    JButton button2=new JButton("button 2");
    frame.add(button2);

    frame.setSize(400,100);
    frame.setVisible(true);
  }
}
```

JFrame

- Java est un langage Objet, il est donc possible d'utiliser l'héritage.
- Doit-on hériter par exemple de la classe JFrame ?

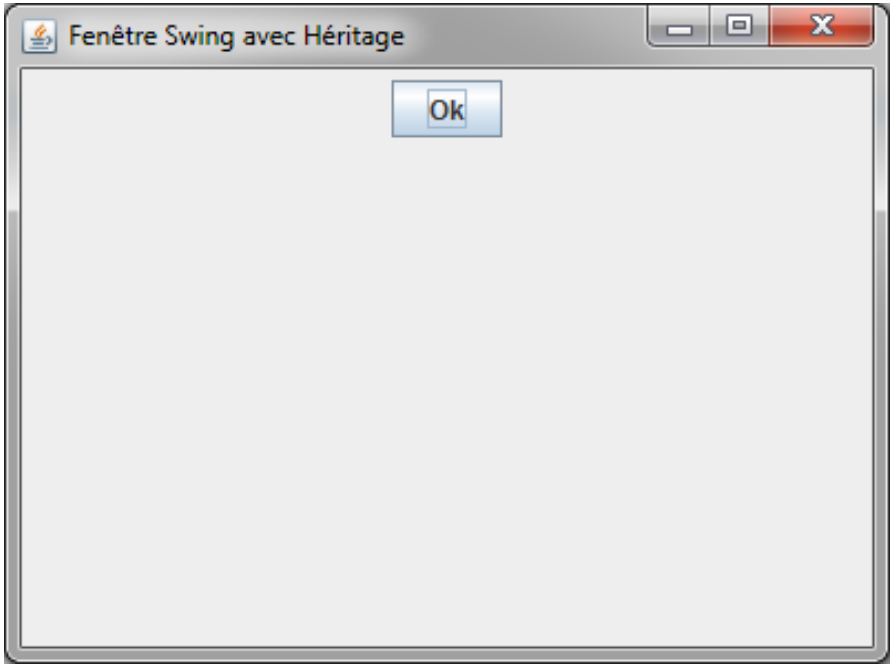
```
import javax.swing.*;

public class MorphSwing1 {
  public static void main(String[] args) {
    JButton button=new JButton("Ok");
    JPanel panel=new JPanel();
    panel.add(button);
    JFrame frame=new JFrame("Fenêtre Swing sans Héritage");
    frame.setContentPane(panel);
    frame.setSize(400,300);
    frame.setVisible(true);
  }
}
```



JFrame

- Java est un langage Objet, il est donc possible d'utiliser l'héritage.
- Doit-on hériter par exemple de la classe JFrame ?
- OUI

```
import javax.swing.*;

public class MorphSwing2 extends JFrame {
public MorphSwing2() {
super("Fenêtre Swing avec Héritage");
JButton button=new JButton("Ok");
    JPanel panel=new JPanel();
    panel.add(button);
    setContentPane(panel);
    setSize(400,300);
    setVisible(true);
    }
    public static void main(String[] args) {
    new MorphSwing2();
    }
}
```

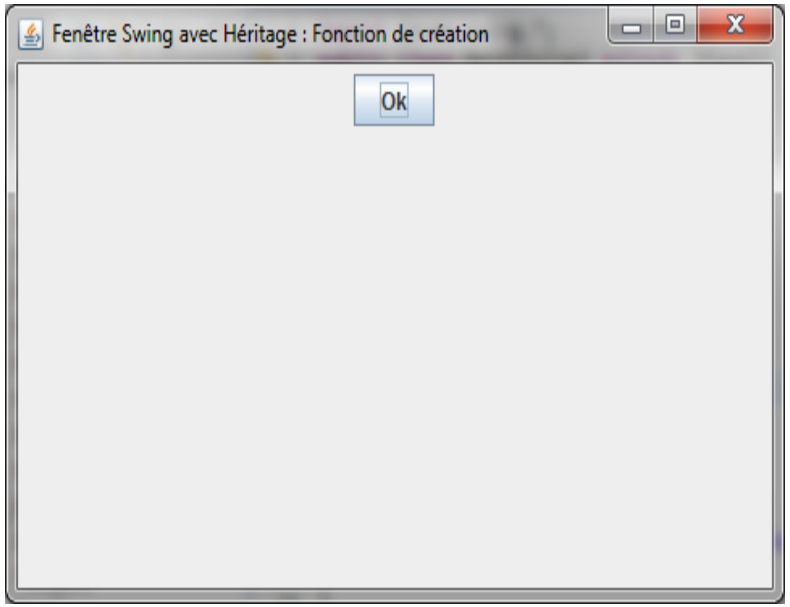


JFrame

- On hérite d'une classe si on veut en changer les fonctionnalités (i.e. redéfinir une méthode)

➔Il est possible de faire des fonctions pour rendre le code plus clair.

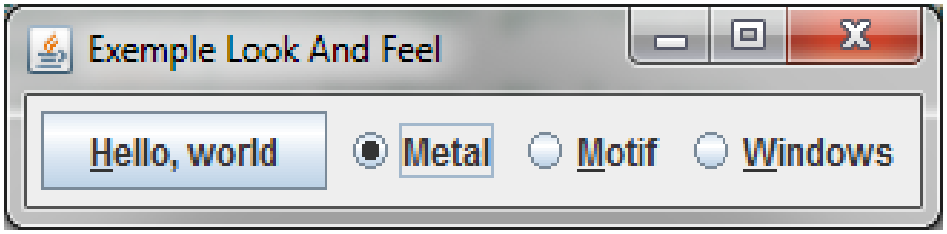
```
import javax.swing.*;
public class MorphSwing3 extends JPanel{
public MorphSwing3() {
     super();
     JButton button=new JButton("Ok");
     this.add(button);
     }
     JFrame créer(String title) {
     JFrame frame = new JFrame(title);
     frame.setContentPane(this);
     frame.setSize(400, 300);
     frame.setVisible(true);
     return frame; }
public static void main(String[] args) {
new MorphSwing3().créer("Fenêtre Swing avec
Héritage : Fonction de création ");
     }
}
```

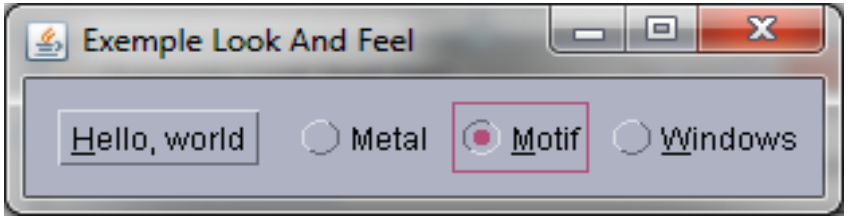


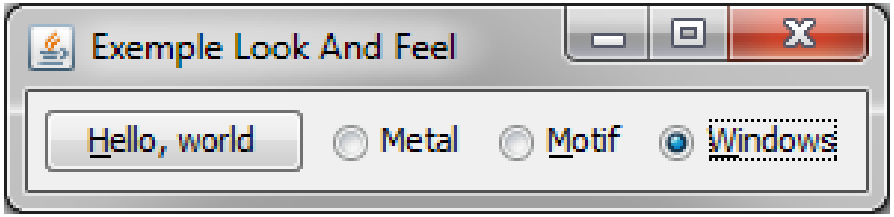
Pluggable Look & Feel

JFrame

- Swing sépare les composants de leur rendu.
- Il est ainsi possible de changer le look d'un ensemble de composants.
  - Windows : com.sun.java.swing.plaf.windows.WindowsLookAndFeel
  - Motif : com.sun.java.swing.plaf.motif.MotifLookAndFeel
  - Metal : javax.swing.plaf.metal.MetalLookAndFeel
  - GTK : com.sun.java.swing.plaf.gtk.GTKLookAndFeel







Pluggable Look & Feel

JFrame

- Swing sépare les composants de leur rendu.
- Il est ainsi possible de changer le look d'un ensemble de composants.
  - Windows : com.sun.java.swing.plaf.windows.WindowsLookAndFeel
  - Motif : com.sun.java.swing.plaf.motif.MotifLookAndFeel
  - Metal : javax.swing.plaf.metal.MetalLookAndFeel
  - GTK : com.sun.java.swing.plaf.gtk.GTKLookAndFeel

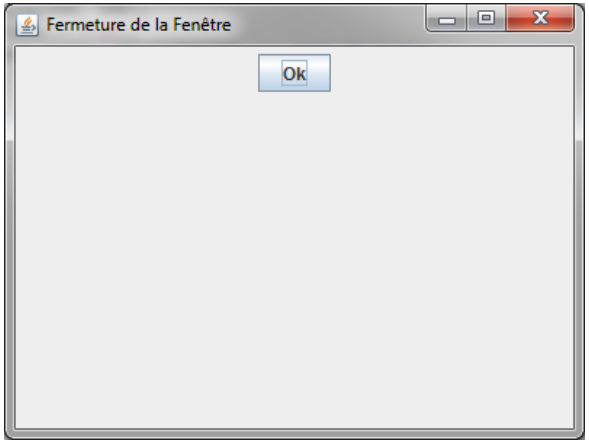
```
try {
  UIManager.setLookAndFeel(className);
} catch (UnsupportedLookAndFeelException e) {
  ...
} catch (Exception e) {
  ...
}
SwingUtilities.updateComponentTreeUI(frame);
frame.pack();
```

```
try {
UIManager.setLookAndFeel("com.sun.java.swing.plaf.motif.MotifLookAndFeel");}
catch(Exception e) { e.printStackTrace(); }
```

Fermeture de fenêtre

JFrame

- Par défaut, en cliquant sur la croix d'une fenêtre, l'application n'est pas arrêtée.
- `setDefaultCloseOperation()` permet de spécifier un comportement
- Comportements :
  - `DO_NOTHING_ON_CLOSE`
  - `HIDE_ON_CLOSE (défaut)`
  - `DISPOSE_ON_CLOSE`
  - `EXIT_ON_CLOSE`



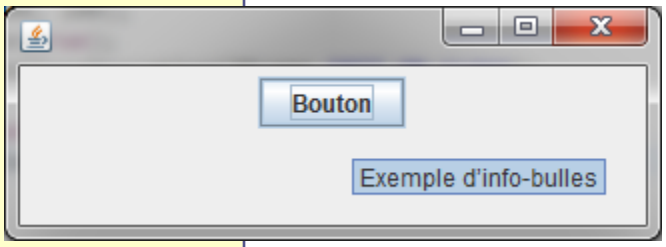

```
import javax.swing.*;
public class FermerFrame extends JFrame {
public static void main(String[] args) {
JButton button = new JButton("Ok");
JPanel panel = new JPanel();
panel.add(button);
JFrame frame = new JFrame("Fermeture de la Fenêtre");
frame.setContentPane(panel);
frame.setSize(400, 300);
frame.setVisible(true);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
} }
```

Les info bulles

JToolTip

- Tous les composants dérivant de **JComponent** peuvent être agrémentés d'info-bulles :
  - ➔petites boites contenant un texte qui présente le rôle du contrôle.
- En pratique, il n'est pas nécessaire d'instancier un objet.
  - ➔La classe **JComponent** propose une méthode qui permet de créer directement le composant **JToolTip** et de l'associer au composant.

```
import javax.swing.*;
public class ExempleJToolTip extends JFrame {
public ExempleJToolTip() {
JButton button = new JButton("Bouton");
button.setToolTipText("Exemple d'info-bulles");
JPanel panel=new JPanel();
panel.add(button);
getContentPane().add(panel);
setSize(300, 100);
setVisible(true);
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); }
public static void main(String args[]) {
new ExempleJToolTip();}}
```



Les bordures

Bordure

- Les composants peuvent être encadrés par une bordure souvent utilisée pour matérialiser des blocs d'informations
- La classe **`BorderFactory`** fournit de nombreuses méthodes statiques pour créer des bordures prédéfinies.

```
import javax.swing.*;
public class Bordure  extends JFrame{
public Bordure() {
JButton button = new JButton("Bouton");
JPanel panel=new JPanel();
panel.add(button);
panel.setBorder(BorderFactory.createTitledBorder("Une bordure"));
setTitle("Exemple Bordure");
setSize(300, 100);
setVisible(true);
setContentPane(panel);
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
  public static void main(String[] args) {
    new Bordure();
  }
}
```

Les boites de dialogues

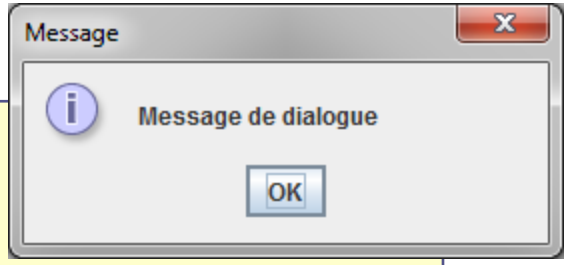
- Swing propose des boites de dialogues afin de communiquer rapidement avec l'utilisateur.
- La classe `JOptionPane` permet de créer des boites de dialogues génériques mais aussi modifiable en fonction des besoins de l'application.
- Les méthodes d'affichage sont statiques

  ➔il n'est donc pas nécessaire d'instancier une classe pour les utiliser.

Les boites de dialogues

Les messages de dialogue

- Informent l'utilisateur en affichant un texte simple et un bouton de confirmation.
- On peut aussi afficher un icône correspondant au message (point d'exclamation, symbole d'erreur,...).
- Pour afficher un message on utilise la méthode **`showMessageDialog`** de la classe **`JOptionPane`**.



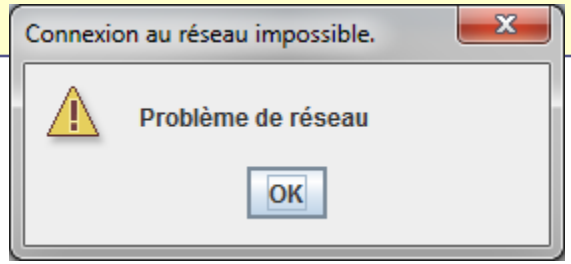

```
import javax.swing.JFrame;
import javax.swing.JOptionPane;

public class MsgDialogue1 {
    public static void main(String[] args) {
    JFrame frame = new JFrame();
    JOptionPane.showMessageDialog(frame, "Message de dialogue");
    System.exit(0);
    }
}
```

Les boites de dialogues

Les messages de dialogue

- On peut modifier l'aspect de la fenêtre en fonction du type de message.
- **Exemple :** Afficher un dialogue d'avertissement .

```
import javax.swing.JFrame;
import javax.swing.JOptionPane;

 public class MsgDialogue2 {
   public static void main(String[] args) {
   JFrame frame = new JFrame();
   JOptionPane.showMessageDialog(frame, "Problème de réseau",
      "Connexion au réseau impossible.", JOptionPane.WARNING_MESSAGE);
   System.exit(0);
   }
   }
```



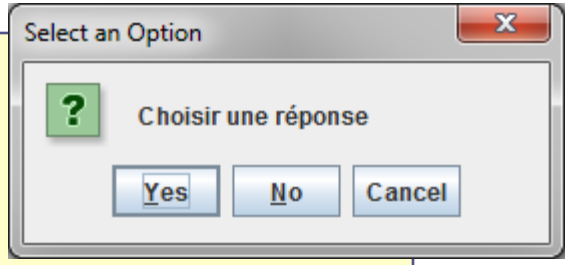
Les boites de dialogues

Les messages de dialogue

Les dialogues de confirmation/question

- Les boites de dialogues peuvent aussi être utilisées pour demander un renseignement à l'utilisateur.
  - Le cas le plus courant est une question fermée.
- La méthode **`showConfirmDialog`** affiche ce type de boite.

```
import javax.swing.JFrame;
import javax.swing.JOptionPane;

public class MsgDialogue3 {
public static void main(String [] args) {
JFrame frame=new JFrame();
JOptionPane.showConfirmDialog(frame, "Choisir une réponse");
System.exit(0);
}
}
```



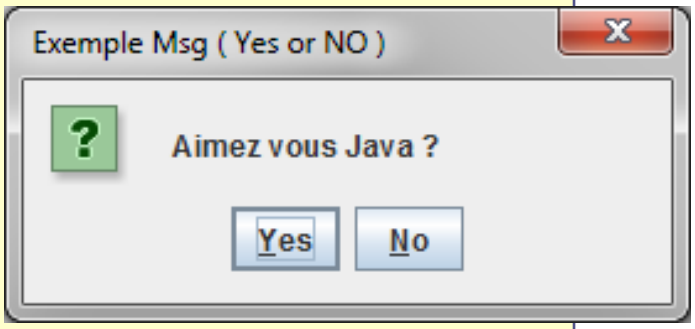
Les boites de dialogues

Les messages de dialogue

Les dialogues de confirmation/question

- La boite de dialogue renvoie un entier en fonction du choix de l'utilisateur et peut être utilisé pour modifier le comportement du programme.
- Cet entier est définit comme une constante de la classe **JOptionPane**.

```
import javax.swing.*;

public class MsgDialogue4 {
     public static void main(String[] args) {
          JFrame frame = new JFrame();
          int rep = JOptionPane.showConfirmDialog(frame, " Aimez vous Java ? ",
          " Exemple Msg (Yes or NO) ", JOptionPane.YES_NO_OPTION);
          if (rep == JOptionPane.YES_OPTION) {
                    // ...
          }
          if (rep == JOptionPane.NO_OPTION) {
                    // ...
          }
          System.exit(0);
     }
}
```



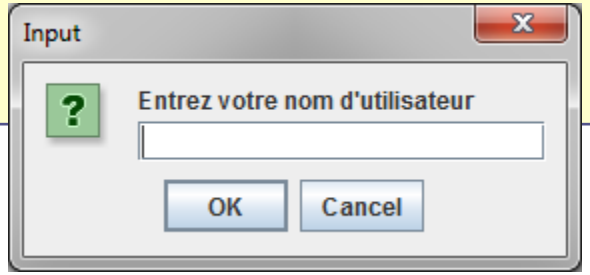
Les boites de dialogues

Les messages de dialogue

Les dialogues de saisie

- La méthode **`showInputDialog`** affiche une boite de dialogue comprenant une entrée de saisie (**`JTextField`**) en plus des boutons de validation.
- Après validation elle retourne une chaîne de caractères (String) si l'utilisateur a cliqué sur OK, sinon elle renvoie **`null`**.

```
import javax.swing.JFrame;
import javax.swing.JOptionPane;

public class MsgDialogue5 {
     public static void main(String[] args) {
          JFrame frame = new Jframe();
          String rep = JOptionPane.showInputDialog(frame, "Entrez votre nom d'utilisateur");
          System.exit(0);
     }
}
```

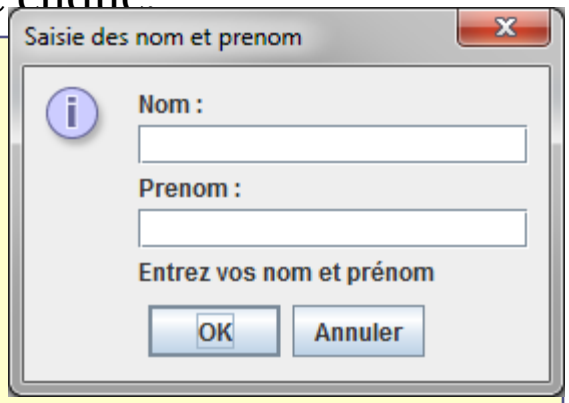


Les boites de dialogues

Les messages de dialogue

Construction de dialogues personnalisés

- Pour construire des boites de dialogue plus complexes on utilise la méthode **`showOptionDialog`**.
- Cette méthode admet 8 paramètres (certains pouvant être à **`null`**).
- Elle renvoie un entier qui correspond au bouton qui a été cliqué.




```
import javax.swing.*;
public class MsgDialogue6{
     public static void main(String[] args) {
          JFrame frame = new Jframe();
          JLabel labelNom = new JLabel("Nom :");
          JLabel labelPrenom = new JLabel("Prenom :");
          JTextField nom = new JTextField();
          JTextField prenom = new JTextField();
          JLabel lab = new JLabel("Entrez vos nom et prénom ");
          Object[] tab = new Object[] { labelNom, nom, labelPrenom, prenom, lab };
          int rep = JOptionPane.showOptionDialog(frame, tab, "Saisie des nom et prenom ",
          JOptionPane.OK_CANCEL_OPTION, JOptionPane.INFORMATION_MESSAGE, null, null, null);
          System.exit(0);
     }
}
```

Les conteneurs secondaires

- Swing propose de nombreux conteneurs secondaires pour créer des interfaces ergonomiques.

  ➔des composants légers.
- Il est possible d'en créer d'autre à l'aide des méthodes de programmation orientée objets.
- La plupart des interfaces possibles dans les systèmes d'exploitation usuels sont présentes comme par exemple les panneaux à onglets, les panneaux défilant,. . .

Les conteneurs secondaires

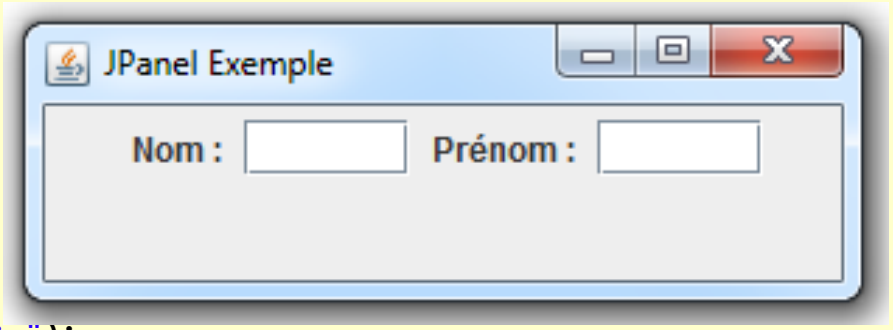
Le panneau : JPanel

- Le conteneur léger le plus simple de Swing et le panneau (`JPanel`), c'est le conteneur par défaut de `JFrame` et de `JWindow`.
- Permet de grouper des composants selon une politique de placement.
- Pour ajouter un composant à un panneau, on utilise la méthode `add` (ou l'une de ses surcharges).
- La méthode réciproque `remove` permet d'enlever un composant.

Les conteneurs secondaires

Le panneau : JPanel

```
import javax.swing.*;
public class FicheIdentite extends JPanel {
      private JTextField nom;
      private JLabel labelNom;
      private JTextField prenom;
      private JLabel labelPrenom;

      public FicheIdentite() {
            super();
            labelNom = new JLabel(" Nom : ");
            labelPrenom = new JLabel(" Prénom : ");
            nom = new JTextField(5);
            prenom = new JTextField(5);
            this.add(labelNom);
            this.add(nom);
            this.add(labelPrenom);
            this.add(prenom);
      }
      public static void main(String[] args) {
            JFrame frame = new JFrame("JPanel Exemple");
            frame.setSize(300, 100);
            frame.setContentPane(new FicheIdentite());
            frame.setVisible(true);
      }
}
```



Les conteneurs secondaires

Le panneau a défilement : JScrollPane

- Les applications traitant du texte ou des images n'affichent souvent qu'une partie du document pour éviter de monopoliser toute la surface d'affichage.
  - Des ascenseurs sont affichées sur les cotés afin de pouvoir se déplacer dans le document.
  - Le composant **`JScrollPane`** permet d'implémenter cette fonction.

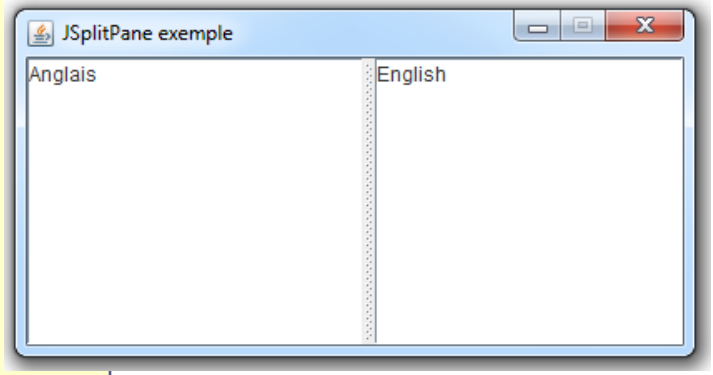
```
import javax.swing.*;
public class TestScrollPane {
     public static void main(String[] args) {
          JFrame frame = new JFrame();
          JLabel monImage = new JLabel(new ImageIcon("Koala.jpg"));
          JScrollPane lePanneau = new JScrollPane(monImage);
          frame.setContentPane(lePanneau);
          frame.setTitle("Image de Koala");
          frame.setSize(400, 400);
          frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
          frame.setResizable(false);
          frame.setVisible(true);
     }
}
```

Les conteneurs secondaires

Le panneau divise : JSplitPane

- Séparer une interface en deux volets est utilisé pour mettre en vis-à-vis deux documents, ou un document et une barre d'outils.
- La séparation peut être horizontale ou verticale selon les interfaces.
- Ce type d'interface fait appel au **`JSplitPane`**.

```
import javax.swing.*;
public class TestSplitPane {
     public static void main(String[] args) {
          JFrame frame = new JFrame("JSplitPane exemple");
          JTextArea source = new JTextArea();
          JTextArea traduction = new JTextArea();
          JSplitPane lePanneau = new
          JSplitPane(JSplitPane.HORIZONTAL_SPLIT,
          source, traduction);
          frame.setContentPane(lePanneau);
          frame.setSize(400, 200);
          frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
          frame.setResizable(false);
          frame.setVisible(true);
     }
}
```

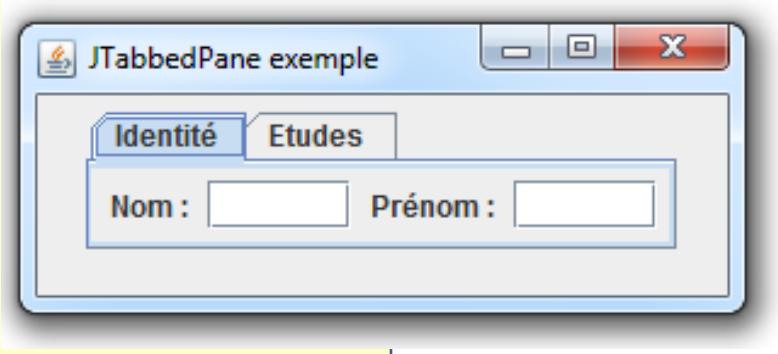


Les conteneurs secondaires

Le panneau a onglets : JTabbedPane

- Le composant **`JTabbedPane`** permet de construire des interfaces en utilisant des onglets.
- Les composants peuvent ainsi être regroupés de manière thématique pour obtenir des interfaces allégées.
- La méthode **`addTab`** permet d'ajouter un composant dans une nouvelle feuille.
- Généralement on utilise le plus souvent **`JPanel`** comme conteneur.

```
public class MenuOnglets extends JTabbedPane {
      private FicheIdentite identite;
      private FicheIdentite etudes;
      public MenuOnglets() {
            identite = new FicheIdentite();
            etudes = new FicheIdentite();
            this.addTab("Identité", identite);
            this.addTab("Etudes ", etudes);
      }
      public static void main(String[] args) {
            JFrame frame = new JFrame("JTabbedPane exemple");
            JPanel panel = new JPanel();
            panel.add(new MenuOnglets());
            frame.setContentPane(panel);
            frame.setSize(300, 120);
            frame.setVisible(true);
      }
}
```

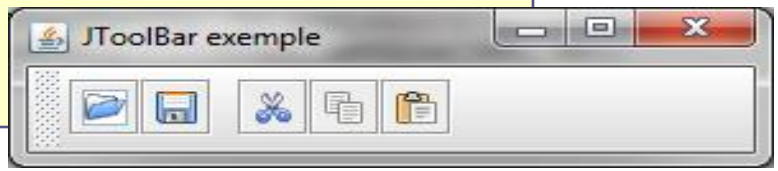


Les conteneurs secondaires

Les barres d'outils : JToolBar

- Les applications complexes font souvent appel à des barres d'outils pour pouvoir accéder facilement aux fonctions les plus utilisées.
- Swing propose un conteneur (`JToolBar`) pour construire des barres d'outils regroupant n'importe quel composant.

```
import javax.swing.*;
public class JToolBarreExemple {
      public static void main(String[] args) {
            JFrame frame = new JFrame("JTabbedPane exemple");
            JButton bOuvrir = new JButton(new ImageIcon("openfile.png"));
            JButton bEnregistrer = new JButton(new ImageIcon("savefile.png"));
            JButton bCouper = new JButton(new ImageIcon("editcut.png"));
            JButton bCopier = new JButton(new ImageIcon("editcopy.png"));
            JButton bColler = new JButton(new ImageIcon("editpaste.png"));
            bOuvrir.setToolTipText("Ouvrir un fichier");
            bEnregistrer.setToolTipText("Enregistrer le fichier");
            bCouper.setToolTipText("Couper vers le presse - papier");
            bCopier.setToolTipText("Copier vers les presse - papier");
            bColler.setToolTipText("Coller depuis le presse - papier");
            JToolBar tb = new JToolBar();  tb.add(bOuvrir);  tb.add(bEnregistrer);
            tb.addSeparator(); tb.add(bCouper);  tb.add(bCopier);       tb.add(bColler);
            frame.setContentPane(tb);     frame.setSize(300, 80);
            frame.setVisible(true);
      }
}
```



Les conteneurs secondaires

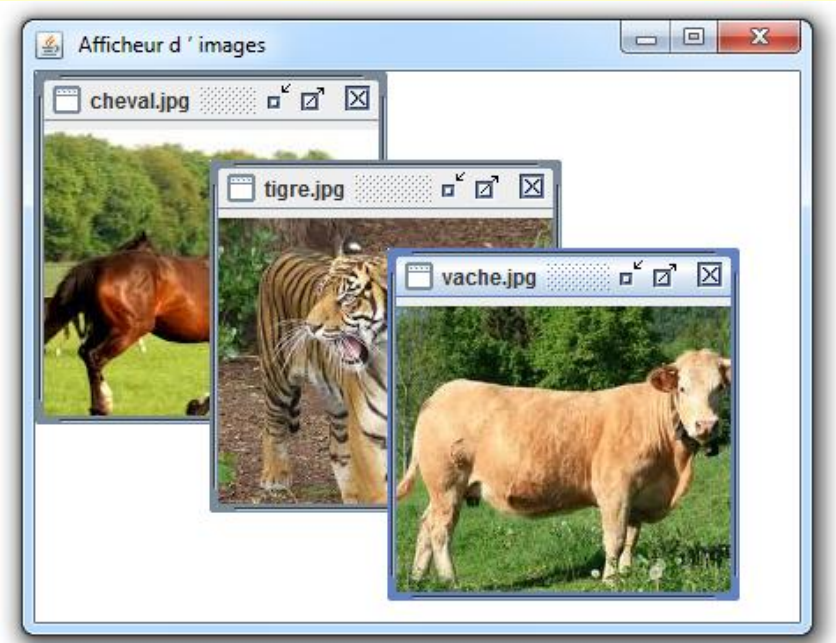
Les bureaux JDesktopPane

- De nombreuses applications autorisent l'ouverture de plusieurs documents simultanément.
  - Ces interfaces sont nommées MDI (Multiple Document Interface).
- Le composant **`JDesktopPane`** propose une implémentation de ce comportement.
  - permet d'afficher des fenêtres (**`JInternalFrame`**) à l'intérieur de l'application.
- Les fenêtres **`JInternalFrame`** proposent des méthodes proches de celles de **`JFrame`** (bien qu'il n'existe pas de relation d'héritage).
- Pour des application MDI, une classe fille est souvent crée à partir de **`JInternalFrame`**.

Les conteneurs secondaires

Les bureaux JDesktopPane

```
import javax.swing.*;

public class FrameMDI extends JInternalFrame {
     public FrameMDI() {
          super("", true, true, true, true);
     }
     public void setImage(String nomImage) {
          JPanel unPanneau = new JPanel();
          JLabel uneImage = new JLabel(new ImageIcon(nomImage));
          unPanneau.add(uneImage);
          this.setContentPane(unPanneau);
          this.setTitle(nomImage);
     }
}
```



Les conteneurs secondaires

Les bureaux JDesktopPane

```
import javax.swing.*;
public class TestDesktop extends JDesktopPane {
     FrameMDI[] animaux;
     String[] nomsFichier = { "cheval.jpg", "tigre.jpg", "vache.jpg" };
     public TestDesktop() {
          animaux = new FrameMDI[3];
          for (int i = 0; i < 3; i++) {
               animaux[i] = new FrameMDI();
               animaux[i].setSize(200, 200);
               animaux[i].setLocation(100 * i, 50 * i);
               animaux[i].setImage(nomsFichier[i]);
               animaux[i].setVisible(true);
               this.add(animaux[i]);
          }
     }
     public static void main(String[] args) {
          JFrame frame = new JFrame();
          frame.setContentPane(new TestDesktop());
          frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
          frame.setVisible(true);
          frame.setSize(450, 350);
          frame.setTitle("Afficheur d'images");
     }}
```