Interface Graphique en Java
Introduction à Swing

Ilhem BOUSSAID

USTHB

5 octobre 2009

Plan

Boîte à outils Swing

- **Swing** est une boîte à outils permettant la création d'interfaces utilisateur en Java.
- **Swing** est une amélioration de **AWT** (*Abstract Window Toolkit*) qui était la boîte à outils originale tout en s'appuyant dessus.
- Swing fournit les classes pour la représentation des différents **éléments d'interfaces graphiques** : *fenêtres*, *boutons*, *menus*, etc., et la gestion des **événements**.
- Ces classes se trouvent dans le paquetage **javax.swing** et ses sous-paquetages.
  - les noms de composants Swing commencent (souvent) par **J**
  - tous héritent du composant graphique **JComponent**
  - **JComponent** hérite du composant graphique AWT **Container** qui permet à un composant d'en contenir d'autres

Look & Feel

- Le Look & Feel permet de d'arranger l'apparence des composants Swing
- Java propose plusieurs look différents dont :
  - StyleWindows
  - Style Motif
  - Style Java Swing
- Une fenêtre avec le style Windows ressemblera aux fenêtres Microsoft Windows

Interfaces Graphiques

Il existe deux principaux types de composants susceptibles d'intervenir dans une interface graphique :

- les **conteneurs** qui sont destinés à contenir d'autres composants, comme par exemple les fenêtres ;
- les **composants atomiques** qui sont des composants qui ne peuvent pas en contenir d'autres, comme par exemple les boutons.

Composants/Conteneur

- Tous les éléments graphiques de Swing sont des **composants**. Ils dérivent de la classe **javax.swing.JComponent**.
- Pour être utilisé, un **composant** doit le plus souvent être placé dans un **conteneur** (**java.awt.Container**).
- Un **conteneur** est un composant qui contient d'autres composants et qui les gouvernent.
- Les objets **conteneurs regroupent** les composants, les **organisent** pour les **afficher** grâce à un **gestionnaire de placement**.
- Les conteneurs les plus utilisés sont **JFrame**, **JDialog** et **JApplet** (il existe également **JWindow** mais il n'est pas utilisé).

Architecture Swing

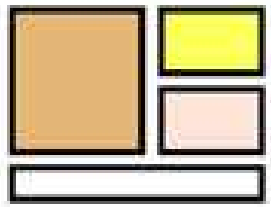
Une application est composée de plusieurs Swing :

- Un composant **top-level**.
  - Swing propose 3 composants top-level : **JFrame**, **JDialog** et **JApplet**
  - Une application graphique doit avoir un composant top-level comme composant racine (composant qui inclus tous les autres composants)
- Plusieurs composants **conteneur intermédiaire**, ils contiennent d'autre composants (JPanel, JScrollPane, JSplitPane, ...)
- Des **composants atomiques** (JButton, JLabel, ...)

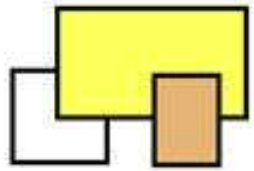
Modèles de fenêtrage

Une application est composée de plusieurs Swing :

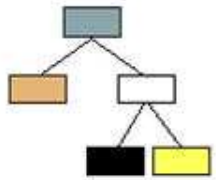
- Sans superposition

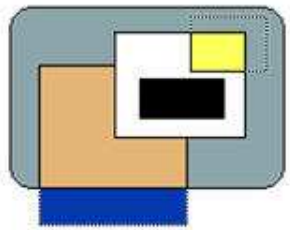


- Avec superposition

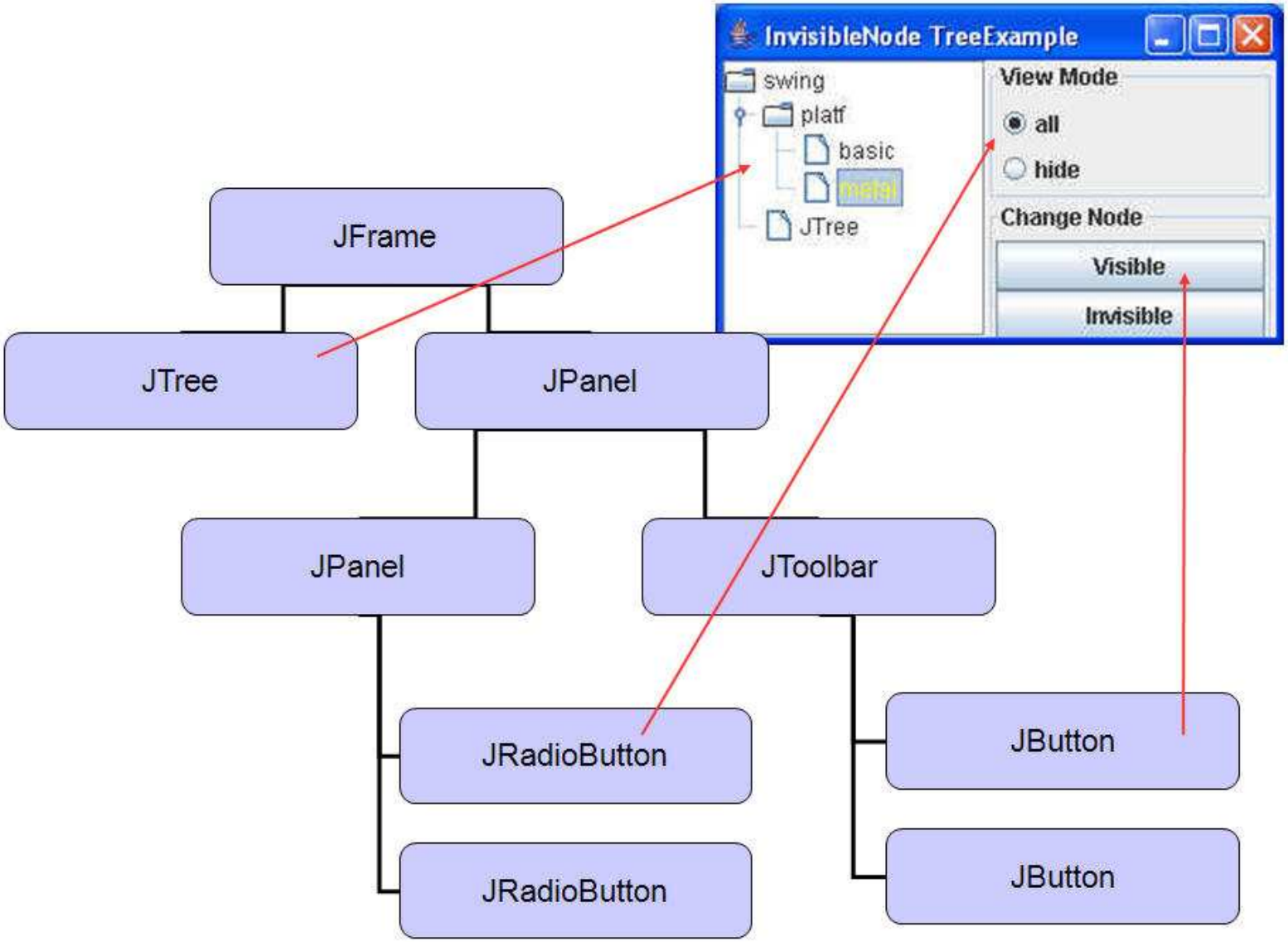


- Hiérarchique

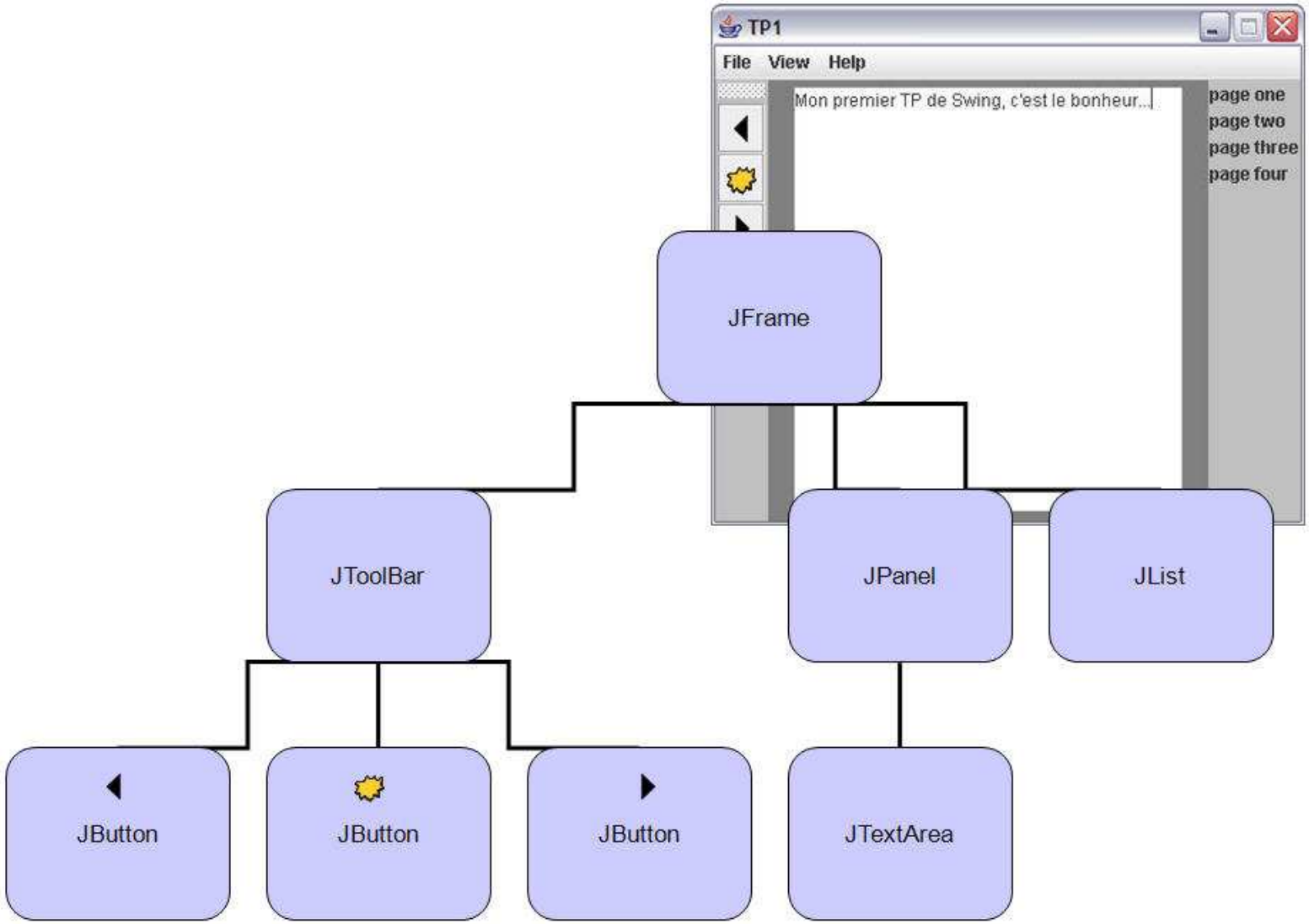




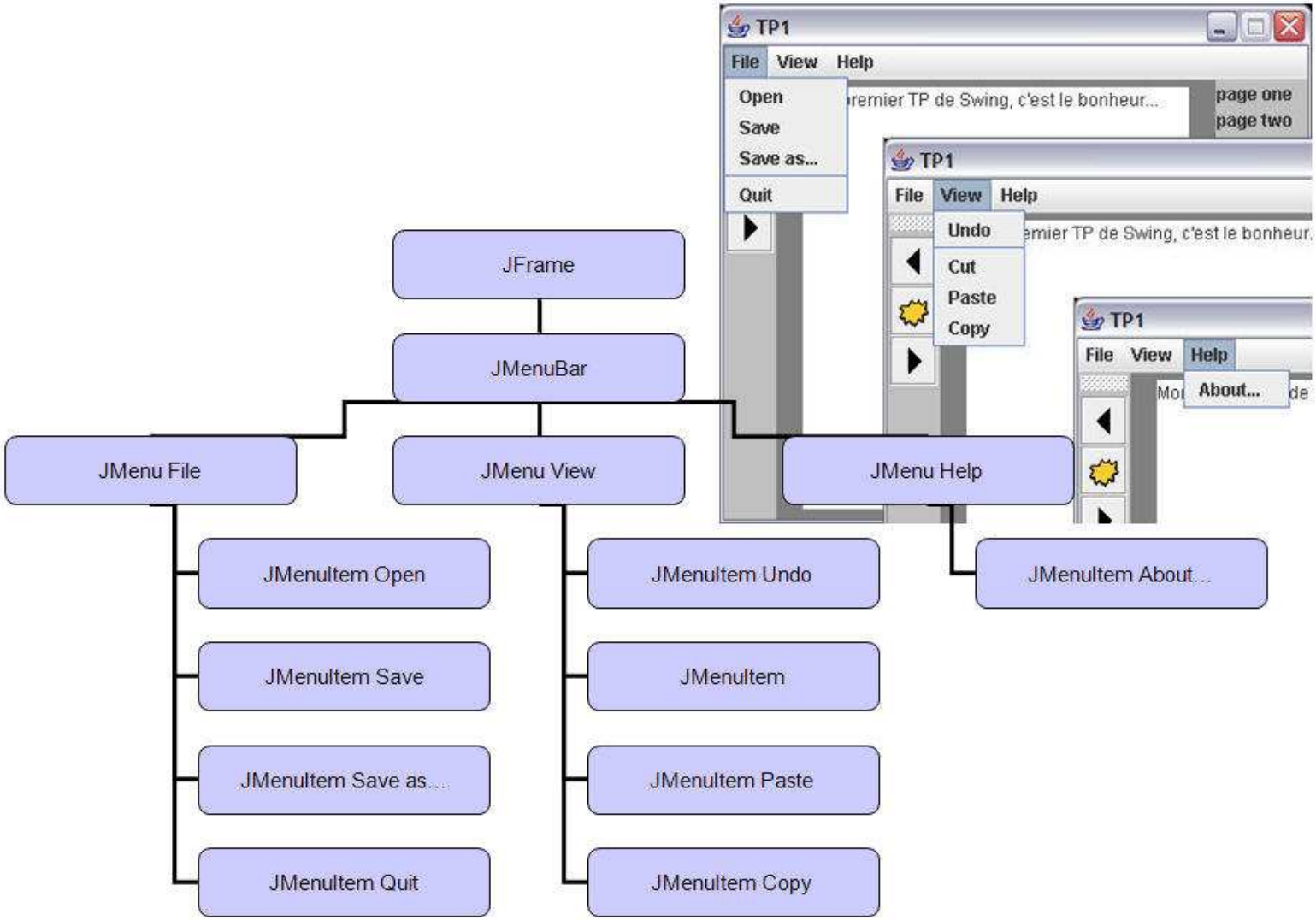
La gestion de l'arbre d'affichage



Arbre Swing


TP1
File View Help
Mon premier TP de Swing, c'est le bonheur...
page one
page two
page three
page four
JFrame
JToolBar
JPanel
JList
JButton
JButton
JButton
JTextArea

Arbre Swing


TP1
File
View
Help
Open
Save
Save as...
Quit
page one
page two
Undo
Cut
Paste
Copy
About...
JFrame
JMenuBar
JMenu File
JMenu View
JMenu Help
JMenuItem Open
JMenuItem Save
JMenuItem Save as...
JMenuItem Quit
JMenuItem Undo
JMenuItem
JMenuItem Paste
JMenuItem Copy
JMenuItem About...

Exemple d'une hiérarchie

```
JFrame frame = new JFrame("Example JFrame");
Container container1 = frame.getContentPane();
JPanel bluePanel = new JPanel();
bluePanel.setBackground(new Color(0, 0, 200));
JTextField component1 = new JTextField("component1");
JCheckBox component2 = new JCheckBox("component2");
JButton component3 = new JButton("component3");
GridLayout layoutBluePanel = new GridLayout(2, 1);
layoutBluePanel.setVgap(3);
bluePanel.setLayout(layoutBluePanel);


container1.add(bluePanel);
container1.add(component3);
bluePanel.add(component1);
bluePanel.add(component2);
```

Le composant JComponent

- La classe de base de tous les objets est **Component** en **AWT** et **JComponent** en **Swing**.
- La classe **JComponent** dérive de la classe **Component**. Ce sont deux classes **abstraites**.
- Tous les composants **Swing** héritent de **JComponent**
- Les composants ont des Tool Tips
- Les composants ont des bordures
- Entité graphique la plus abstraite

Propriétés des JComponent

Montrer et cacher un composant :

```
void setVisible(boolean b)
```

Activer et désactiver un composant (un composant inactif ne reagit plus aux évènements provoqués par l'utilisateur - correspond a un rendu grise) :

```
void setEnabled(boolean b)
```

Connaître l'état (activé ou non) d'un composant :

```
boolean isEnabled()
```

Modifier la couleur de fond d'un composant :

```
void setBackground(Color c)
```

Modifier la couleur du premier plan d'un composant :

```
void setForeground(Color c)
```

Propriétés des JComponent

Modifier la taille d'un composant :

```
void setSize(int largeur, int hauteur)
```

Modifier la position et la taille d'un composant :

```
void setBounds(int x, int y, int largeur, int hauteur)
```

la taille courante :**set/getSize** + 3 autres tailles utilisées, entre autres, par les **LayoutManager** pour calculer la place accordée à chaque composant :

- la préférée : **set/getPreferredSize**
- la minimum : **set/getMinimumSize**
- la maximum : **set/getMaximumSize**

La classe **Dimension** du paquetage **java.awt** contient deux champs publics : un champ height (hauteur) et un champ width (largeur).)

Propriétés des JComponent

- **Bordures** definies avec la méthode **setBorder(Border border)**
- La classe **BorderFactory** fabrique des Borders tout prêt à l'emploi avec les méthodes statiques :
  - **BorderFactory.createRaisedBevelBorder()** - encadrement style biseauté
  - **BorderFactory.createEtchedBorder()** - encadrement style gravé
  - **BorderFactory.createLineBorder(Color couleur, int LargeurTrait)** - tour en ligne
  - **BorderFactory.createTitledBorder(String title)** - encadrement avec un titre apparent
  - etc.
- à utiliser de preference sur les **JPanel** et les **JLabel**
- pour les autres composants, il peut y avoir des problèmes de rendu

Plan

Création d'une JFrame

- Une JFrame est une fenêtre avec un titre et une bordure
- Quelques méthodes :

```
/* constructeur sans argument*/
public JFrame();
/** constructeur avec un argument (titre de la fenêtre)*/
public JFrame(String name);
public Container getContentPane();
public void setJMenuBar(JMenuBar menu);
public void setTitle(String title);
public void setIconImage(Image image);
```

- Une fenêtre est un conteneur destiné à contenir d'autres composants.
- La méthode **getContentPane** de la classe **JFrame** fournit une référence, de type **Container**, au contenu de la fenêtre.
- Les méthodes **add** et **remove** de la classe **Container** permettent respectivement d'ajouter et de supprimer un composant d'une fenêtre.

Création d'une JFrame

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

class Fenetre {
public static void main(String args[]){
JFrame fenetre = new JFrame("Titre");
fenetre.pack();
fenetre.setVisible(true);
fenetre.setBounds(400, 10, 300,150);
}
}
```

- Chaque composant a une dimension préferrée (getPreferredSize())
- La méthode **pack** exécute la mise en page en fonction des layouts managers des conteneurs et des dimensions préferrées des composants feuilles.

Autre approche

```
import java.awt.*;
import javax.swing.*;
public class FrameDemo extends JFrame {
//constructeur
public FrameDemo() {
this.setTitle("MY first Frame"); /* titre de la fenêtre*/
this.setSize (400 , 200) /* dimensions de la fenêtre*/
/* ou this.setSize(new Dimension(400,200));*/
this.setLocation(250,150); // emplacement de la fenêtre
}
public static void main(String[] args) {
new FrameDemo().setVisible(true);
}
}
```

Fermeture d'une JFrame

- La fermeture d'une fenêtre de type JFrame ne met pas fin au programme, mais rend simplement la fenêtre invisible (comme si on appelait la méthode **setVisible(false)**).
- Pour mettre fin au programme, il faut fermer la fenêtre console.
- 4 modes possibles, à définir avec **setDefaultCloseOperation** :
  - **DO_NOTHING_ON_CLOSE**
  - **HIDE_ON_CLOSE** : rend la fenêtre invisible (mode par défaut)
  - **DISPOSE_ON_CLOSE** : cache la fenêtre et libère toutes les ressources systèmes associées ; elles seront réallouées si la fenêtre redevient visible
  - **EXIT_ON_CLOSE** : termine le programme

La classe Toolkit

- La classe **Toolkit** fait le lien entre les classes java indépendantes de toute plate-forme et la plate-forme sur laquelle tourne le programme
- La boîte à outils par défaut permet d'extraire les données comme la taille locale de l'écran de la machine virtuelle Java
- On obtient une instance de **Toolkit** par la méthode de classe de **Toolkit** :**public static Toolkit getDefaultToolkit()**
- **Exemple** :

```
Toolkit aTK= Toolkit.getDefaultToolkit();
Dimension dim = aTK.getScreenSize();
```

- La méthode **getScreenSize()** renvoie un objet **Dimension** qui stocke la hauteur et la largeur de l'écran comme champ publique.
- Notez que les classes **Toolkit** et **Dimension** sont définies dans le paquetage **java.awt.***.

Exemple

Exemple : fenêtre au centre de l'écran en définissant sa taille à 1/4 de l'écran (deux fois moins haute et large).

```
import javax.swing.*;
import java.awt.*;
public class FrameDemo1{
   public static void main(String []arg){
     Dimension screenSize =
                Toolkit.getDefaultToolkit().getScreenSize();
     int w=screenSize.width;
     int h=screenSize.height;
     JFrame frame=new MainFrame(w/2, h/2, w/4, h/4);
     frame.setVisible(true);
   }
}
class MainFrame extends JFrame{
      public MainFrame (int width, int height, int x, int y){
           setTitle("Cadre principal");
           setSize(width, height);
           setLocation(x, y);
       }
}
```

JDialog

- Un **JDialog** est une fenêtre qui a pour but un échange d'information
- Un **JDialog** dépend d'une fenêtre, si celle-ci est détruite, le JDialog l'est aussi
- Un **JDilaog** peut aussi être modal, il bloque tout les inputs sur lui
- Java propose un certain nombre de boîtes de dialogue standard obtenues à l'aide de méthodes de classe de la classe **JOptionPane** :
  - Boîtes de message
  - Boîtes de confirmation
  - Boîtes de saisie
  - Boîtes d'options
  -
- La classe **JDialog** permet de construire des boîtes de dialogue personnalisées.

Les boîtes de message

- Une boîte de message fournit à l'utilisateur un message qui reste affiché tant que l'utilisateur n'agit pas sur le bouton OK.
- Elle est construite à l'aide de la méthode de classe **showMessageDialog** de la classe **JOptionPane**.

Les boîtes de message

```
import java.awt.* ;
import javax.swing.* ;
class MaFenetre extends JFrame {
public MaFenetre () {
super("Une fenetre") ; setSize(300, 200) ; }
}
public class BoiteMess {
public static void main(String args[]) {
JFrame fen = new MaFenetre() ;
fen.setVisible(true) ;
JOptionPane.showMessageDialog(fen, "Bonjour") ;
}
}
```

Le premier argument de la méthode **showMessageDialog** correspond à la fenêtre parent de la boîte de message, c'est à dire la fenêtre dans laquelle la boîte de message va s'afficher. Cet argument peut prendre la valeur **null**.

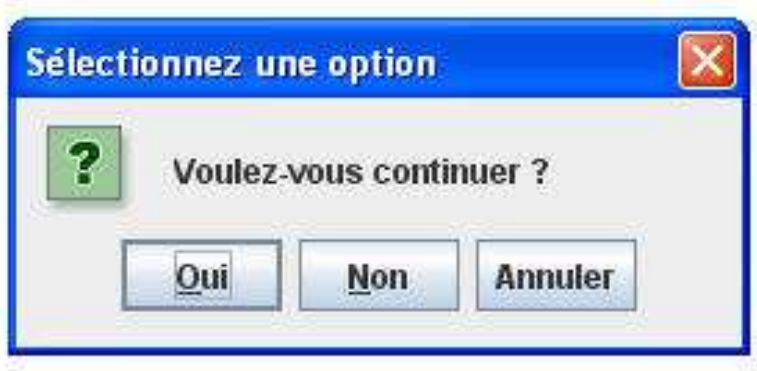
Les boîtes de message

- Il existe une variante de la méthode **showMessageDialog** qui permet aussi de choisir le **titre** de la boîte et le **type d'icône** parmi la liste suivante (les paramètres sont des constantes entières de la classe **JOptionPane**).
  - **Erreur** JOptionPane.ERROR_MESSAGE
  - **Information** JOptionPane.INFORMATION_MESSAGE
  - **Avertissement** JOptionPane.WARNING_MESSAGE
  - **Question** JOptionPane.QUESTION_MESSAGE
  - **Aucune icône** JOptionPane.PLAIN_MESSAGE
- Exemple :

```
JOptionPane.showMessageDialog(fen, "Mauvais choix",
"Message d'avertissement", JOptionPane.ERROR_MESSAGE) ;
```

Les boîtes de confirmation

- Une boîte de confirmation offre à l'utilisateur un choix de type **oui/non**.
- Elle est construite à l'aide de la méthode de classe **showConfirmDialog** de la classe **JOptionPane**.



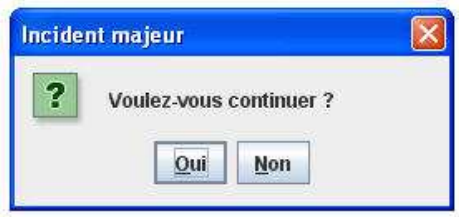
Les boîtes de confirmation

```
import java.awt.* ;
import javax.swing.* ;
class MaFenetre extends JFrame {
public MaFenetre () {
super("Une fenetre") ; setSize(300, 200) ; }
}
public class BoiteConf {
public static void main(String args[]) {
JFrame fen = new MaFenetre() ;
fen.setVisible(true) ;
JOptionPane.showConfirmDialog(fen, "Voulez-vous continuer ?") ;
}
}
```

Les boîtes de confirmation

- Il existe une variante de la méthode **showConfirmDialog** qui permet aussi de choisir le **titre** de la boîte et la **nature des boutons** :
  - **Erreur** JOptionPane.DEFAULT_OPTION (-1)
  - **boutons YES et NO** JOptionPane.YES_NO_OPTION (0)
  - **boutons YES, NO et CANCEL** JOptionPane.YES_NO_CANCEL_OPTION(1)
  - **boutons OK et CANCEL** JOptionPane.OK_CANCEL_OPTION (2)
- Exemple :

```
JOptionPane.showConfirmDialog(fen, "Voulez-vous continuer ?", "In
JOptionPane.YES_NO_OPTION) ;
```



Ajouter un composant à un "top-level" container

- Chaque "**top-level**" container possède une **aire de travail**, une instance de la classe **Container**.
- On n'ajoute pas un composant directement à un top-level container (**JFrame**, **JApplet** ou **JDialog**) mais à son aire de travail
- On **accède** à cette aire de travail en adressant la méthode **getContentPane()** (présente dans chacune des classes **top-level**) à la fenêtre

```
JFrame fen=new JFrame("ajout");
Container c=fen.getContentPane();
c.add(...)
```

Plan

Conteneur Intermédiaire

- Les conteneur intermédiaire sont utilisés pour structurer l'application graphique
- Le composant top-level contient des composants conteneur intermédiaire
- Un conteneur intermédiaire peut contenir d'autre conteneurs intermédiaire
- Swing propose plusieurs conteneurs intermédiaire :
  - JPanel
  - JScrollPane
  - JSplitPane
  - JTabbedPane
  - JToolBar
  - ...

Conteneur Intermédiaire

- Voici les principales méthodes communes à toutes ces classes :
  - **void setLayout(LayoutManager)** qui associe au Conteneur l'objet de placement passé en paramètre.
  - **void add(Component)** qui ajoute ce composant au Conteneur sans désignation de zone lorsque l'objet de placement n'en a pas besoin (**FlowLayout**, **GridLayout** ou **GridBagLayout**)
  - **void add(Component,Object)** qui ajoute ce composant au Conteneur en utilisant une désignation de zone qui dépend de l'objet de placement qui a été associé à ce Conteneur (int pour un **BorderLayout**)
  - **void remove(Component)** qui enlève ce composant du Conteneur
  - **void removeAll()** qui enlève tous les composants du Conteneur
  - **Component locate(int,int)** qui retourne le composant situé aux coordonnées données en paramètre

JPanel

- Le **JPanel** est le conteneur intermédiaire le plus neutre
- On ne peut que choisir la couleur de fond
- Quelques méthodes de JPanel :
  - **JPanel()** qui se contente de créer l'objet.
  - **JPanel(LayoutManager)** qui accepte en paramètre un objet de placement et construit le JPanel associé à cet objet.
- Bien que les composants comme **JLable** puissent toujours être ajoutés directement au contenu du cadre principal, il est généralement préférable de les ajouter à un conteneur **intermédiaire**, qui est lui-même ajouté au panneau de contenu du cadre principal.

Exemple

```
import java.awt.* ;
import javax.swing.* ;
class PanelFrame extends JFrame {
private JPanel panneau ;
public PanelFrame () {
super("Une fenetre avec un panneau jaune") ;
setSize(300, 100) ;
panneau = new JPanel();
panneau.setBackground(Color.yellow) ;

JLabel jl = new JLabel("BONJOUR!");
panneau.add(jl);
getContentPane().add(panneau) ;
}
/*+ code dans main pour créer et afficher la JFrame*/
}
```

JScrollPane

- **JScrollPane** est une version améliorée de JPanel qui possède des ascenseurs de défilement vertical et horizontal.
- Quelques méthodes de JScrollPane :
  - **JScrollPane()** qui crée un JScrollPane vide
  - **JScrollPane(Component)** qui crée un JScrollPane contenant un seul composant (celui passé en paramètre).
  - **JScrollPane(int,int)** qui crée un JScrollPane vide en précisant le comportement des ascenseurs
  - **JScrollPane(Component,int,int)** qui crée un JScrollPane contenant un seul composant (celui passé en paramètre) en précisant le comportement des ascenseurs. Le premier entier définit le comportement de l'ascenseur vertical et le dernier entier définit le comportement de l'ascenseur horizontal

JScrollPane

- Le comportement de l'ascenseur vertical prend les valeurs suivantes :
  - **JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED** (l'ascenseur n'apparaît que s'il est nécessaire),
  - **JScrollPane.VERTICAL_SCROLLBAR_NEVER** (pas d'ascenseur) ou **JScrollPane.VERTICAL_SCROLLBAR_ALWAYS** (l'ascenseur est toujours présent).
- Le comportement de l'ascenseur horizontal prend les valeurs suivantes :
  - **JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED** (l'ascenseur n'apparaît que s'il est nécessaire),
  - **JScrollPane.HORIZONTAL_SCROLLBAR_NEVER** (pas d'ascenseur) ou **JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS** (l'ascenseur est toujours présent).

Conteneur Intermédiaire Spécialisé

Les **conteneur Intermédiaire spécialisé** sont des conteneurs qui offrent des propriétés particulières aux composants qu'ils accueillent

- JRootPane
- JLayeredPane
- JInternalFrame