COURS 6 (semaine du 11/5/2020)

1.1 Les librairies graphiques

Les Interfaces Homme- Machines

- Le package « AWT » (java.awt.*)
- Le package « SWING » (javax.swing.*)

Le dessin

- La classe « Graphics » (java.awt.Graphics)
- La classe « Graphics2D »

La librairie AWT (Abstract Windowing Toolkit)

- Librairie interprétée par le système
- Éléments graphiques : boutons, menus, fenêtres,…
- Bibliothèque pour programmer des GUI depuis Java 1.0
  - ✓ Toujours disponible mais déconseillé
  - ✓ D'un aspect assez médiocre
  - ✓ Implémentation différent pour chaque plateforme (utilisation de composants graphiques natifs)
  - ✓ Bibliothèque de composants très limité
  - ✓ Obligé de l'utiliser pour les installations < Java 1.1
  - ✓ 2 packages :

import java.awt.* ; pour les objets graphiques https://docs.oracle.com/javase/8/docs/api/java/awt/package-summary.html
import java.awt.event.* ; pour les évènements https://docs.oracle.com/javase/8/docs/api/java/awt/event/package-summary.html

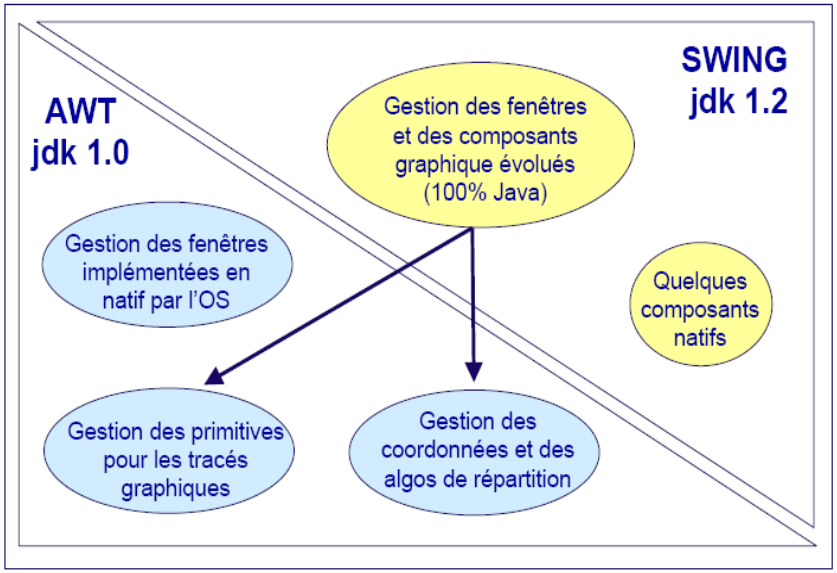
La librairie Swing plus économe en mémoire

- 1 package : **J** devant le nom des objets AWT
- Swing fournit une large et moderne bibliothèque de composants graphiques (**javax.swing**)
  - ✓ Les essentiels: **JButton, JLabel, JTextArea, JMenu,** …
  - ✓ Mais aussi: **Tree, Table, FileChooser, ProgressBar, ScrollPane** …

- Les composants Swing sont légers et entièrement écrit en Java, donc rapide et portable.
- Swing fournit également des fonctionnalités avancées:
  - ✓ Internationalisation
  - ✓ pluggable look and feel.

- 1 package :

import javax.swing.* ; https://docs.oracle.com/javase/8/docs/api/javax/swing/package-summary.html



1.2 Statique et dynamique

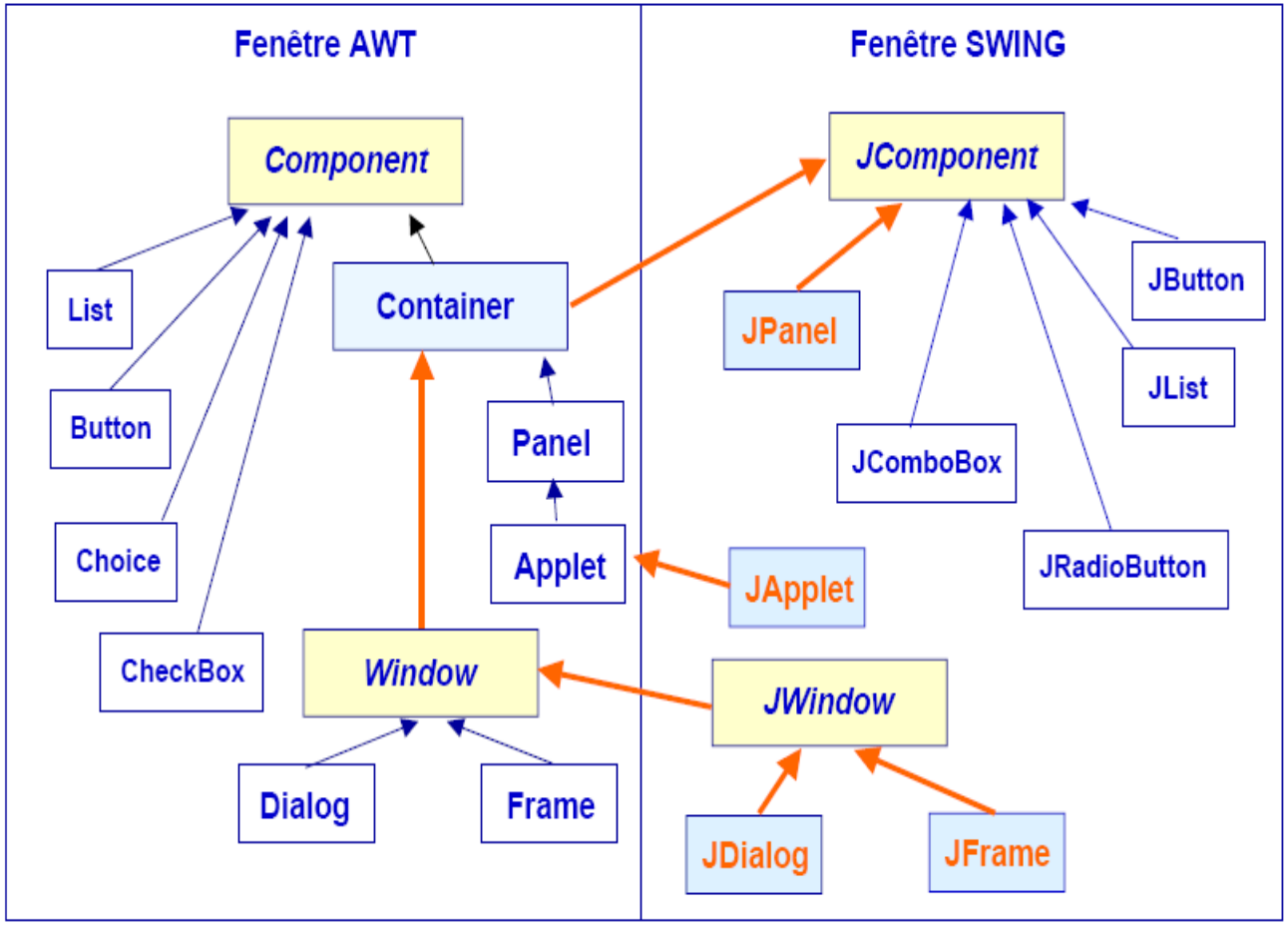
La partie « statique »

- La notion de « conteneurs »
- Les contrôles (composants graphiques)
- La politique de placement des composants dans les « conteneurs »

La partie « dynamique »

- Le système d 'événements
- L'affectation d'un événement à un contrôle

1.3 La hiérarchie des classes AWT et SWING



1.4 Les classes Component et JComponent

Les classes Component et JComponent

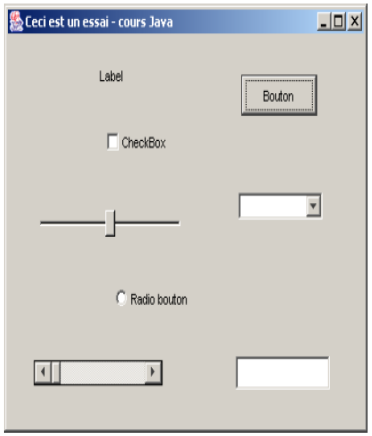
- Héritent de la classe « Object »
- Il est possible d'utiliser des objets dérivant de « Component » et de « JComponent » dans le même programme mais c'est déconseillé
- Préférer les composants « SWING » aux composants « AWT »
  - 100% Java (indépendant du système de fenêtrage de l'OS)
  - Plus riche en terme de fonctionnalités
- Composants « AWT » : Applets devant fonctionner avec certains navigateurs

1.5 Les contrôles (composants graphiques) de SWING

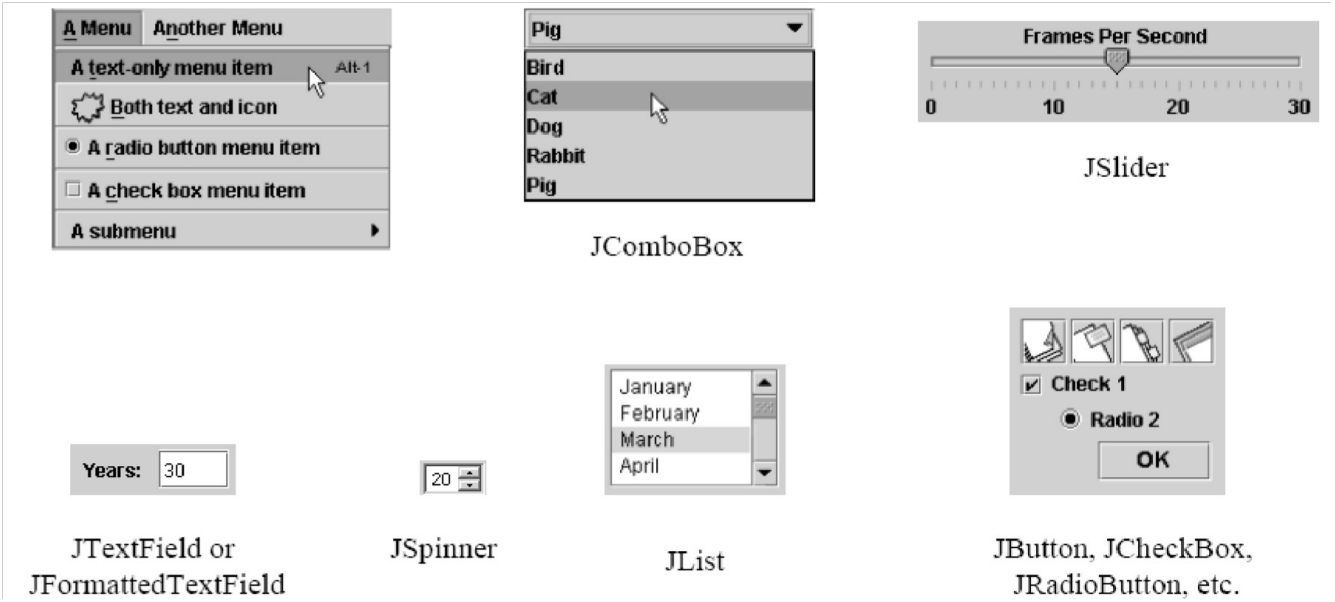
Contrôles (composants graphiques)	Classe SWING	Quelques méthodes
Etiquette	JLabel	setText
Champs de texte	JTextField	getText, setEditable, setColumns
Bouton de commande	JButton	
Cases à cocher	JCheckBox	
Boutons radio	JRadioButton	
Barre de menus	JMenuBar	getMenu
Menus	JMenu	AddMenuListener, add
Menus surgissants	JPopupMenu	
Options de menu	JMenuItem	
Options cases à cocher	JCheckBoxMenuItem	
Options Boutons radio	JRadioButtonMenuItem	
Boîtes de liste	JList	getSelectedValue, getSelectedValues, getSelectedIndex, getSelectedIndices
Boîtes combo	JComboBox	setEditable, getSelectedIndex
Document	JDocument	
Fenêtre	JWindow	
Fenêtre principale	JFrame	setSize, setTitle, setBounds, setVisible, getContentPane
Panneau	Jpanel	
Souris, Clavier, Focus	JComponent	paintComponent

Les « contrôles » de SWING

- JLabel
- JTextField
- JButton
- Canvas ou Canvas3D
- JCheckBox
- JRadioButton
- JScrollBar
- JComboBox

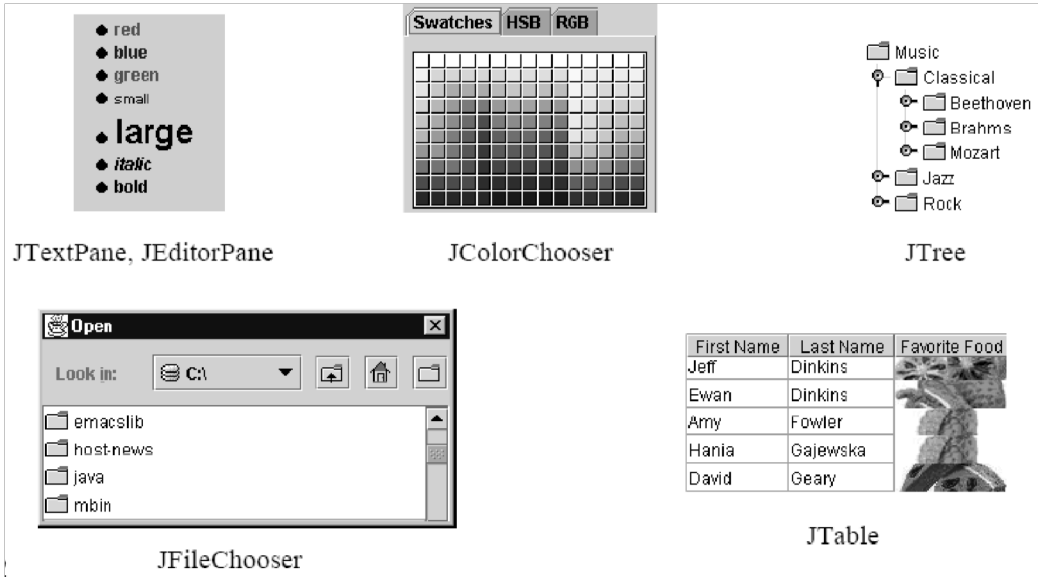


- Composants utilisés pour récupérer une information de l'utilisateur et afficher un simple état



- Composants qui ne servent qu'à afficher une information

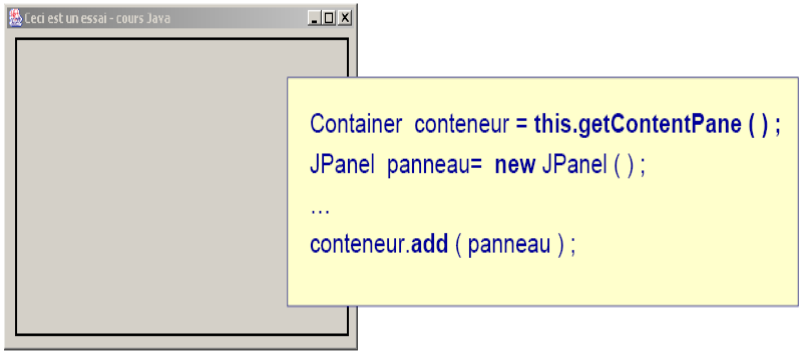
- Composants qui affichent des données formatées, éventuellement modifiables par l'utilisateur



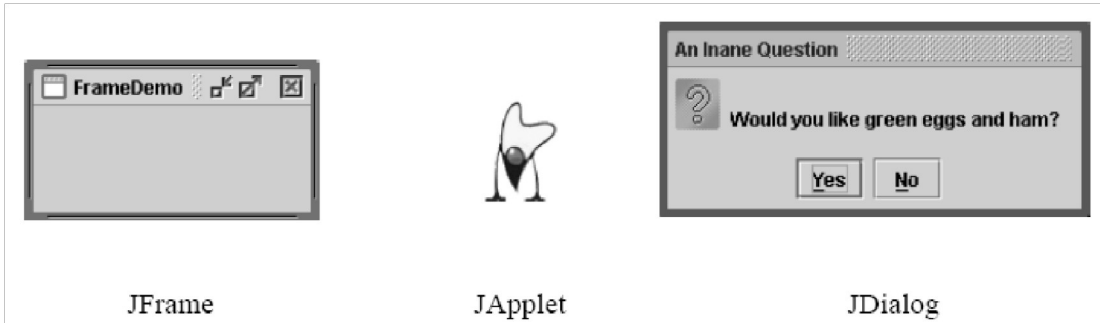
1.6 Les conteneurs

Les « conteneurs »

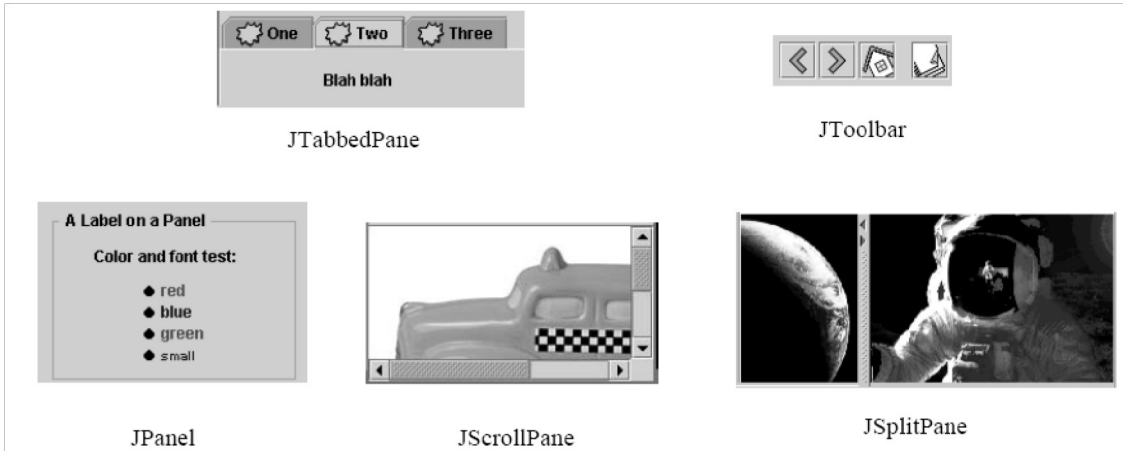
- La classe « Container »
- Les classes « JPanel » , « JScrollPane » et « JSplitPane »



➢ Conteneurs racines de toute hiérarchie de composants Swing



- Conteneurs intermédiaires fréquemment utilisés



JTabbedPane

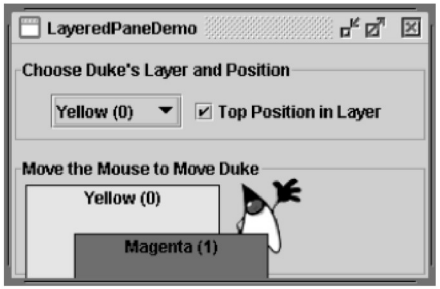
JToolbar

JPanel

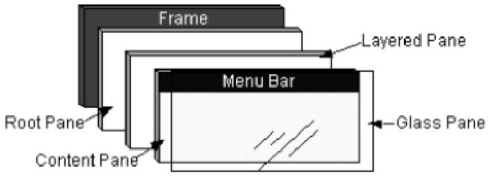
JScrollPane

JSplitPane

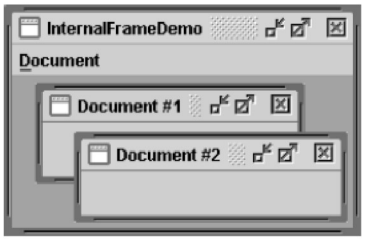
- Conteneurs intermédiaires ayant une signification spéciale dans l'interface utilisateur



JLayeredPane



JRootPane, JGlassPane

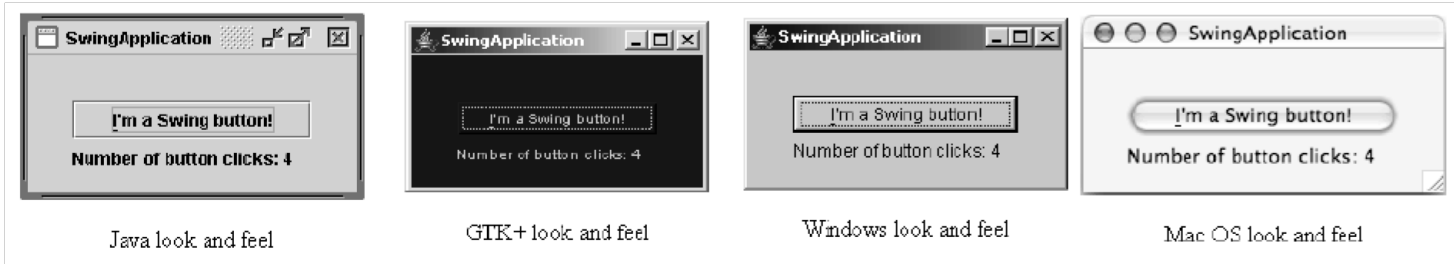


JInternalFrame

VA2

Activer / Choisir un look and feel à la JFrame

- ✓ **JFrame.setDefaultLookAndFeelDecorated(true);** demande la création des fenêtres par le look and feel courant.
- ✓ Swing vous permet de spécifier le look and feel désiré pour votre application avec **UIManager.setLookAndFeel (...)**



Java look and feel | GTK+ look and feel | Windows look and feel | Mac OS look and feel

Créer un top-level container

- ✓ Un application graphique doit avoir au moins un top-level container - un conteneur racine dans lequel un interface peut être construit.
- ✓ Un **JFrame** est une fenêtre d'application principal avec les décorations habituels (bordure, titre, bouton de fermeture, etc.)
- ✓ Un **JApplet** est un espace d'affichage imbriqué dans un page HTML.
- ✓ Également : **JWindow, JDialog**

Configurer un top-level container

- **setDefaultCloseOperation (int)** spécifie l'opération à effectuer par l'application quand l'utilisateur appuie sur le bouton Close de la barre du titre.

Exemples des valeurs possibles:

WindowConstants.HIDE_ON_CLOSE (défaut)
JFrame.EXIT_ON_CLOSE : fait appel à **System.exit()**

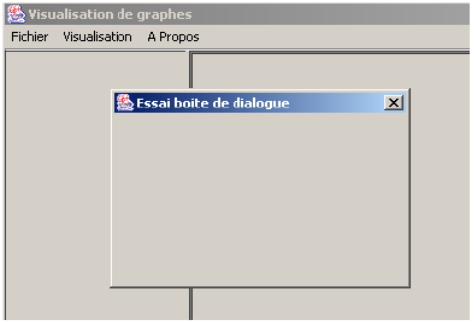
Ajouter les composants graphiques au conteneur

- **JLabel label = new JLabel("Hello World");** Crée un composant Swing qui fait office d'une simple étiquette, ici avec valeur "Hello World".
- **frame.getContentPane().add(label);** Ajoute le nouvel étiquette au content pane de notre **JFrame**.
- Tout conteneur Swing contient un content pane qui contient les composants affiché par le conteneur.
- **frame.pack () ;** Adapte les dimensions du JFrame aux dimensions préférées de ses composants
- **frame.setSize (int width, int height)** Adapte les dimensions du **JFrame** à celles spécifiées
- **frame.setVisible (true) ;** Affiche le **JFrame**

1.7 Les boites de dialogue

Les « boites de dialogue »

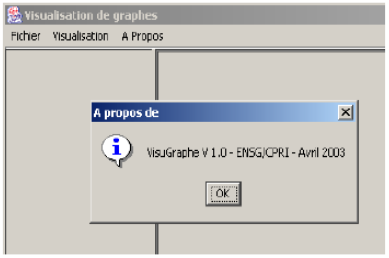
- JDialog
  - Dépend d'une « frame »
  - Modale (doit être fermée)
  - Non modale




```
JDialog dial  =  new JDialog ( frame, "essai" , true ) ;
```

Les « boites de dialogue »

- JOptionPane (toujours modale)




QUESTION_MESSAGE


WARNING_MESSAGE


ERROR_MESSAGE

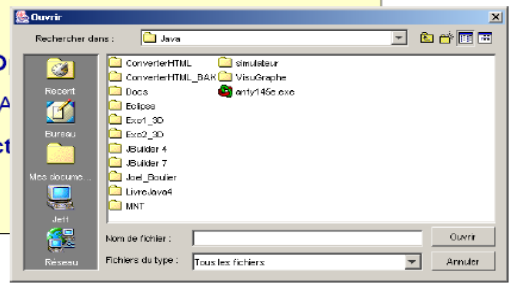
```
JOptionPane.showMessageDialog ( frame, "VisuGraphe V 1.0 - ENSG/CPRI - Avril 2003",
                                "A propos de", JOptionPane.INFORMATION_MESSAGE ) ;
```

Les « boites de dialogue »

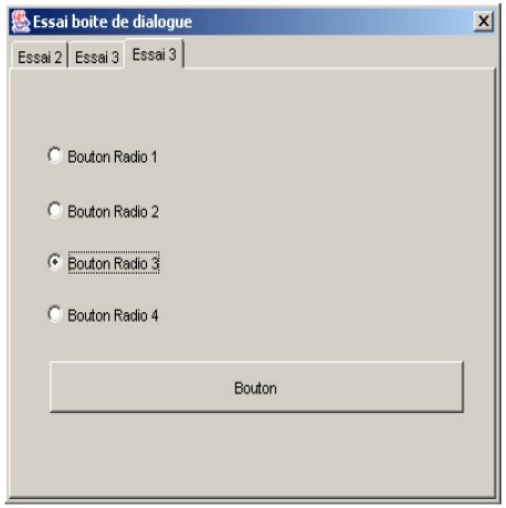
- JFileChooser

```
nomfic = new String ( ) ;
JFileChooser chooser = new JFileChooser ("D:/Java") ;

int returnVal = chooser.showO
if ( returnVal == JFileChooser.A
    nomfic = chooser.getSelect
    ...
}
```



Les « boites de dialogue »



1.8 La disposition des composants dans les conteneurs : Layout

- Java préconise l'utilisation de Layout managers pour contrôler la disposition des composants dans un conteneur, au lieu d'un positionnement absolu.
- Un Layout manager utilise un algorithme interne pour placer les composants, selon l'ordre dans lequel on les ajoute au conteneur avec **add()**.
- L'usage d'un Layout manager est moins fastidieux qu'un positionnement par pixel et plus flexible :
  - ✓ Il adapte (dynamiquement) l'affichage des composants aux dimensions du conteneur.

➢ BorderLayout

Divise l'espace d'affichage dans une zone centrale (**CENTER**) et 4 régions périphériques (**NORTH**, **SOUTH**, **EAST**, **WEST**).

◆ Ajoute au centre :

Container.add (new Label("hi"));
Container.add (BorderLayout.CENTER, new Label("hi"));
Remplir le milieu et étire le composant jusqu'aux autre composants ou jusqu'aux bords Si plusieurs composants sont ajoutés au centre, ils se superposent.

◆ Ajoute à la région périphérique :

Container.add (BorderLayout.NORTH, new Label("hi")); Positionné en haut, remplit l'espace disponible sur l'axe horizontale, et occupe le moins de place possible sur l'axe verticale

Container.add (BorderLayout.WEST, new Label("hi")); Positionné à gauche, remplit l'espace disponible sur l'axe verticale, et occupe le moins de place possible sur l'axe horizontale.

Le « BorderLayout »

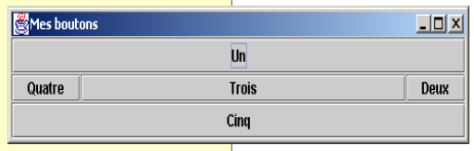
```
panneau.setLayout ( new BorderLayout ( ) ) ;
this.getContentPane ( ).add ( panneau ) ;

Bouton1.setText ("Un") ;
Bouton2.setText ("Deux") ;

...

panneau.add (" North " , Bouton1 ) ;
panneau.add (" East " , Bouton2 ) ;
...

this.setTitle ("Mes boutons") ;
this.setSize (new Dimension ( 496, 78 ) ) ;
this.setVisible ( true ) ;
```



Center, West, South

FlowLayout

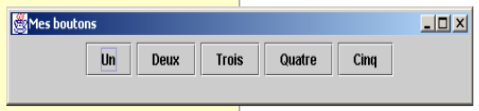
- Aligne les composants sur une ligne de gauche à droite, et ligne par ligne de haut en bas.
- Chaque composant prend son taille minimale; il n'y a pas d'expansion comme dans BorderLayout.

Le « FlowLayout »

```
panneau.setLayout ( new FlowLayout ( ) ) ;
this.getContentPane ( ).add ( panneau ) ;

Bouton1.setText ("Un") ;
Bouton2.setText ("Deux") ;
...

panneau.add ( Bouton1 ) ;
panneau.add (Bouton2 ) ;
...
this.setTitle ("Mes boutons") ;
this.setSize (new Dimension ( 496, 78 ) ) ;
this.setVisible ( true ) ;
```



GridLayout

- Crée un tableau de composants
- Lors de l'ajoute ils sont placés dans les cases du tableau de gauche à droite et de haut en bas.
- Le nombre de rangées et de colonnes dans le tableau sont spécifiés dans le constructeur:

  GridLayout (int rangées, int colonnes**)**

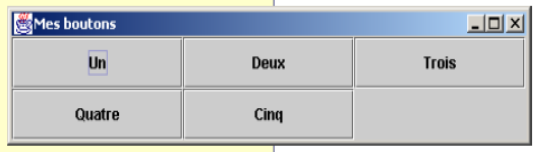
- Les cases ont tous des proportions identiques.
- Chaque composant est étiré pour remplir sa case.

Le « GridLayout »

```
panneau.setLayout ( new GridLayout ( 2, 3 ) ) ;
this.getContentPane ( ).add ( panneau ) ;

Bouton1.setText ("Un") ;
Bouton2.setText ("Deux") ;
...

panneau.add ( Bouton1 ) ;
panneau.add (Bouton2 ) ;
...
this.setTitle ("Mes boutons") ;
this.setSize (new Dimension ( 496, 78 ) ) ;
this.setVisible ( true ) ;
```



GridBagLayout et BoxLayout

GridBagLayout

- Permet un contrôle fin sur la positionnement des régions d'un fenêtre. Très compliqué.
- Destiné principalement à la génération de code automatique par des GUI Builders

BoxLayout

- Fournit la plupart des avantages de **GridBagLayout**, sans la complexité.
- Permet le contrôle du placement des composants soit verticalement soit horizontalement, et le contrôle de l'espace entre eux.

Positionnement absolu

Pour spécifier une position absolue pour les composants :

a) Positionner un layout manager **null** pour le conteneur:
Container :: setLayout (null);

b) Appeler **setBounds()** pour chaque composant :
Component :: setBounds (Rectangle r)

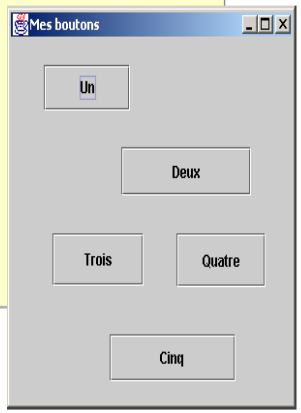
Avec le constructeur de Rectangle (x, y, width, height)

```
panneau.setLayout ( null ) ;
this.getContentPane ( ).add ( panneau ) ;

Bouton1.setBounds ( new Rectangle ( 34, 20, 90, 35 ) ) ;
Bouton1.setText ("Un") ;
Bouton2.setBounds ( new Rectangle ( 115, 82, 134, 36 ) ) ;
Bouton2.setText ("Deux") ;
...

panneau.add ( Bouton1 ) ;
panneau.add ( Bouton2 ) ;
...

this.setTitle ("Mes boutons") ;
this.setSize ( 300,300 ) ;
this.setVisible ( true ) ;
```



1.9 Exemples de code

Exemple 1 : création d'un bouton dans une fenêtre

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

public class ButtonTest extends JFrame {
    JButton MonBouton ;
    public ButtonTest ( String titre ) {
        setTitle ( titre ) ;
        Container contentPane = this.getContentPane ( ) ;
        contentPane.setLayout ( new FlowLayout ( ) ) ;
        MonBouton = new JButton ( "Mon Bouton" ) ;
        ContentPane .add ( MonBouton ) ;
        setSize ( 200, 100 ) ;
        setVisible ( true ) ;
    }
```

```
    public static void main ( String argv [ ] )
    {
        new ButtonTest ( "Mon Bouton" ) ;
    }
}
```

Exemple 2 : création d'un label dans une fenêtre

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

public class HelloWorldSwing {
    public static void main(String[] args) {
        // Demande la création de fenêtres décorés
        JFrame.setDefaultLookAndFeelDecorated(true);

        // Créer et configurer un fenêtre d'application
        JFrame frame = new JFrame("HelloWorldSwing");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.getContentPane().setLayout(new FlowLayout());

        // Ajouter un "label" au contenu du fenêtre
        JLabel label = new JLabel("Hello World");
        frame.getContentPane().add(label);

        // Afficher le fenêtre
        frame.setSize(200,50);
        frame.setVisible(true);
    }
}
```

Decorated == true

Decorated == false

1.10 Ressources de campus

Les interfaces graphiques Swing
Création interface graphique avec Swing

1.11 Dessiner avec l'objet Graphics

Graphiques 2D

- Classes du package AWT
- Tracer en 2D, des lignes, des formes (simples et complexes – courbe de Bézier, composition …) , des caractères (fonte, police …)
- Gérer le remplissage des formes avec ou sans textures
- Gérer les zones de « clipping » (zone de découpe)
- Gérer des transformations (rotation, translation, homothétie)
- Simuler des animations
- Manipuler les images (application de filtres …)

- Les méthodes « draw » et « fill » sont accessibles pour tous les objets des classes graphiques (depuis Java 2 – plus évoluées)
- Il existe des méthodes spécifiques : « drawRect », « drawOval » …

```
g2d.setStroke (dashed) ;
g2d.draw (new RoundRectangle2D.Double (0, 0, 30, 10 ) ) ;
```

```
g2d.setPaint (redtowhite) ;
g2d.fill (new Rectangle2D.Double (0, 0, 30, 10 ) ) ;
```

```
g2d.setPaint (red) ;
g2d.fill (new Arc2D.Double (0, 0, 30, 10, 90, 135,
                            Arc2D.OPEN ) ) ;
```

setColor(Color c) Sets this graphics context's current color to the specified color.
setFont(Font font) Sets this graphics context's font to the specified font.
drawString(String str, int x, int y) Draws the text given by the specified string, using this graphics context's current font and color.
drawLine(int x1, int y1, int x2, int y2) Draws a line, using the current color, between the points (x1, y1) and (x2, y2) in this graphics context's coordinate system.
drawRect(int x, int y, int width, int height) Draws the outline of the specified rectangle.
drawOval(int x, int y, int width, int height) Draws the outline of an oval.

- C'est un objet de la classe « Graphics »
- Il comporte
  - La zone dessin (le composant) pour les méthodes « draw… »
  - Le rectangle de découpe (zone de clipping)
  - La couleur courante
  - La fonte courante
  - Une opération de dessin
- Chaque composant peut accéder à un contexte graphique explicitement ou implicitement

- Le constructeur de la classe « Graphics » est déclaré « protected »
  - Impossible de **construire** un objet de la classe « Graphics »
- Implicitement les méthodes « paint » et « update » construisent un contexte graphique passé en paramètre
- Explicitement :
  - Pour une Application on utilise la méthode « getGraphics » de la classe « Component »
  - Pour une Applet on utilise le « contexte graphique » (définit dans le fichier HTML) passé en paramètre de la méthode « paint »

```
public class  Mon_Application  {
    Graphics g;
    JPanel panneau = new JPanel ( ) ;
    …
    panneau.setVisible ( true ) ;
    …
    g = this.getGraphics ( ) ;
    g.drawRect (20, 20, 60, 60 ) ;
    …
}
```

Application

Applet

```
public class  Mon_Applet extends Applet  {
    public void paint (Graphics g)  {
        g.drawRect ( 20, 20, 60, 60 ) ;
    }
}
```

2.1 Le modèle d'événements

Les événements

- Préviennent le programmeur et les composants de l'interface graphique que quelque chose s'est produit
- Leur détection permet de réagir aux actions de l'utilisateur en adaptant son comportement à ces actions
- Une gestion en trois étapes :
  - Déterminer les événements et les associer à des « écouteurs »
  - Écrire le code des « écouteurs » et de chaque événement
  - Enregistrer les « écouteurs »

La notion « d'écouteur » ou « Listener »

- Classe possédant au moins une méthode, appelée si l'événement attendu apparaît
- Une classe « écouteur » implémente une interface
- Plusieurs types « d'écouteur » :
  - « ActionListener » écoute tous les événements (menus)
  - « MouseListener » écoute les événements souris
  - « WindowListener » écoute les événements sur les fenêtres
  - …

Écriture du code des « écouteurs »

- Exemple : traitement de l'événement « ActionEvent » déclenché par pression d'un bouton souris (ou touche clavier)
- Pour traiter un tel événement on doit implémenter l'interface « ActionListener » qui contient la méthode « actionPerformed »
- La méthode « actionPerformed » doit contenir le code à exécuter à chaque « clic souris »

2.2 Capture d'évènement

Exemple 1 : capture d'évènement

```
public class EventFrame extends JFrame implements ActionListener {
   JButton BoutonOk, BoutonCancel;
   JLabel statusLabel;
   /** Creates a new instance of EventFrame */
   public EventFrame(String titre) {
      setTitle(titre);
      Container contentPane = getContentPane();
      JPanel ButtonPanel = new JPanel();
      ButtonPanel.setLayout (new FlowLayout());
      BoutonOk = new JButton("Ok");
      ButtonPanel.add(BoutonOk);
      BoutonCancel = new JButton("Cancel");
      BoutonOk.addActionListener(this);
      BoutonCancel.addActionListener(this);
      ButtonPanel.add(BoutonCancel);
      contentPane.add(ButtonPanel,BorderLayout.CENTER);
      statusLabel = new JLabel("Zone de texte");
      contentPane.add(statusLabel,BorderLayout.SOUTH);
      setSize(200,100);
      setVisible(true);
   }

   public void actionPerformed (ActionEvent event){
      if (event.getSource() == BoutonOk){
         statusLabel.setText("Ok détecté");
      }
      if (event.getSource() == BoutonCancel) {
         statusLabel.setText("Cancel détecté");
      }
   }
```

```
public class MonTest  {
   public static void main ( String[ ] args )  {
      EventFrame ef = new EventFrame ( "Mon Bouton ") ;
   }
}
```

Exemple 2 : capture - avec classe interne

```
public class CaptureDemo extends JFrame {
    JButton button1 = new JButton("Button X!"), button2 = new JButton("Button Y!") ;
    JTextField txt = new JTextField(10);

    // Définition d'une classe interne
    class BL implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            String name = ((JButton)e.getSource()).getText();
            // une classe interne à accès aux attributs de la classe externe
            txt.setText(name);
        }
    }

    // Instanciation de la classe interne
    BL handler = new BL();

    public CaptureDemo() {
        // enregistrement de l'objet écouteur
        button1.addActionListener(handler);
        button2.addActionListener(handler);
    }
}
```

Appui sur Button X

Appui sur Button Y

2.3 Ressources sur campus

Exemples de gestion evenementielle avec Swing

L'interception des actions de l'utilisateur Auteur : Jean-Michel DOUDOUX

2.4 Les événements et leurs listeners

Evénement	*Interface*	*Générateurs*
ActionEvent	ActionListener	JButton, JList, JTextField, JMenuItem et ses dérivés
AdjustmentEvent	AdjustmentListener	JScrollbar et tous ceux qui implémente l'interface Adjustable
ComponentEvent	ComponentListener	Component et ses dérivés
ContainerEvent	ContainerListener	Container et ses dérivés
FocusEvent	FocusListener	Component et ses dérivés
KeyEvent	KeyListener	Component et ses dérivés
MouseEvent	MouseListener	Component et ses dérivés
MouseEvent	MouseMotionListener	Component et ses dérivés
WindowEvent	WindowListener	Window et ses dérivés
ItemEvent	ItemListener	JCheckBox, JCheckBoxMenuItem,JComboBox, JList et tous ceux qui implémente l'interface ItemSelectable
TextEvent	TextListener	JTextComponent et ses dérivés

Composant	Événement créé	Méthode invoquée
JButton	ActionEvent	actionPerformed
JCheckBox	ItemEvent	itemStateChanged
JList	ListSelectionEvent	valueChanged
JTextField	ActionEvent	actionPerformed
JWindow	WindowEvent	WindowOpen/Close …
JComponent	ComponentEvent	componentMoved/Resized …
JComponent	FocusEvent	focusLost, focusGained
JComponent	KeyEvent	keyPressed/Released/Typed
…	…	…

2.5 Quelques-unes des interfaces Listeners

Listener **[et *Adapter*]**	**Méthodes**
ActionListener	actionPerformed (ActionEvent e)
AdjustmentListener	adjustmentValueChanged(AdjustmentEvent)
ComponentListener ComponenetAdapter	componentHidden(ComponentEvent) componentShown(ComponentEvent) componentMoved(ComponentEvent) componentResized(ComponentEvent)
FocusListener FocusAdapter	focusGained (FocusEvent) focusLost (FocusEvent)
MouseListener MouseAdapter	mouseClicked (MouseEvent) mouseEntered (MouseEvent) mouseExited (MouseEvent) mousePressed (MouseEvent) mouseReleased (MouseEvent)
MouseMotionListener MouseMotionAdapter	mouseDragged (MouseEvent) mouseMoved (MouseEvent)

Listener **[et *Adapter*]**	**Méthodes**
KeyListener KeyAdapter	keyPressed (KeyEvent) keyReleased (KeyEvent) keyTyped (KeyEvent)
WindowListener WindowAdapter	windowOpened (WindowEvent) windowClosing (WindowEvent) windowActivated (WindowEvent) windowDeactivated (WindowEvent) windowIconified (WindowEvent) windowDeiconified (WindowEvent)
ItemListener	itemStateChanged (ItemEvent)

2.6 Utilisation de Listener Adapters

- Comme nous voyons dans le tableau ci-dessus, certaines interfaces Listener ont plusieurs méthodes.
- Rappel sur les interfaces : Quand on implémente une interface, il faut définir toutes les méthodes de l'interface.
- Avec un Listener cela peut devenir pénible, surtout quand on ne s'intéresse qu'à une seule méthode du Listener
- Pour résoudre ce problème, certains Listener sont accompagnés par des classes adapters :
  - ✓ Un adapter est une classe qui fournit des méthodes vides pour chacune des méthodes du Listener.
  - ✓ Ensuite il suffit d'hériter de l'adapter et de redéfinir uniquement les méthodes qui vous intéresse.

Exemple : Pour traiter la fermeture d'une fenêtre.

Par utilisation de WindowListener

```
class MyWindowListener1 implements WindowListener {
    public void windowClosing(WindowEvent e) { System.exit(0);  }
    public void windowOpened(WindowEvent e) {}
    public void windowActivated(WindowEvent e) {}
    public void windowDeactivated(WindowEvent e) {}
    public void windowIconified(WindowEvent e) {}
    public void windowDeiconified(WindowEvent e) {}
}
```

Par utilisation de WindowAdapter

```
public class MyWindowListener2 extends WindowAdapter {
    public void windowClosing(WindowEvent e) { System.exit(0);  }
}
```