Composants graphiques de Java

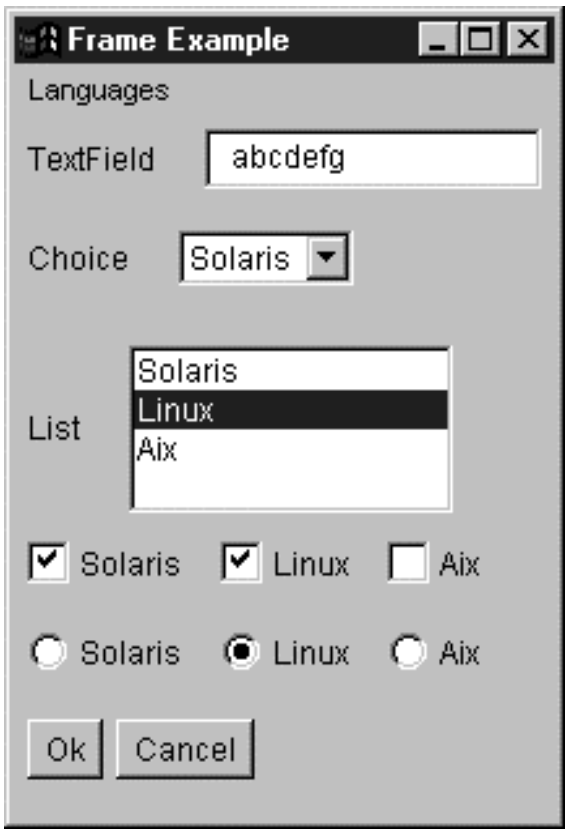
- Aperçu général
- “Bonjour, monde”
- Classes de composants
- Gestion des événements
- Modèle - vue - contrôleur
- Applettes
- Images
- Un exemple détaillé
- Swing
- le même exemple, en Swing
- Pluggable Look and feel
- Actions

Sources des exemples...

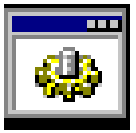
- La plupart des exemples de programmes sont inspirés des exemples figurant dans les livres suivants:
  - R. Eckstein, M. Loy, D. Wood, “Java Swing”, O’Reilly 1998.
  - C. Horstmann, G. Cornell, “Au cœur de Java 2”, Campus Press, vol. 1 1999, vol. 2 2000.
  - M. Robinson, P. Vorobiev, “Swing”, Manning Publ. Co. 2000.
  - J. Knudsen, “Java 2D Graphics”, O’Reilly 1999.
- Pour des compléments Java, voir:
  - G. Roussel, E. Duris, “Java et Internet”, Vuibert 2000.

Aperçu général

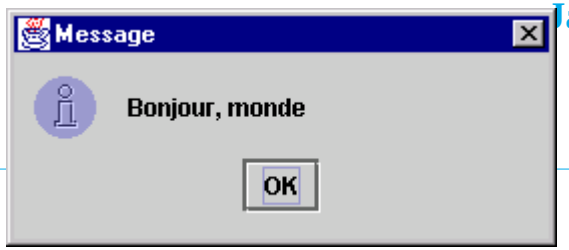
- Les programmes à interfaces graphiques font usage des classes *awt* (*abstract windowing toolkit*) et/ou *swing*.
- Ils sont dirigés par évènements.
- Classe de base des *awt* : la classe abstraite **`Component`**.
- Classe de base des composants *swing* : **`JComponent`**.
- On distingue, par service
  - les classes conteneur
  - les classes d'interaction
  - les menus et dialogues
- *Swing* offre une palette bien plus large.



“Bonjour, monde”



Bjm.bat



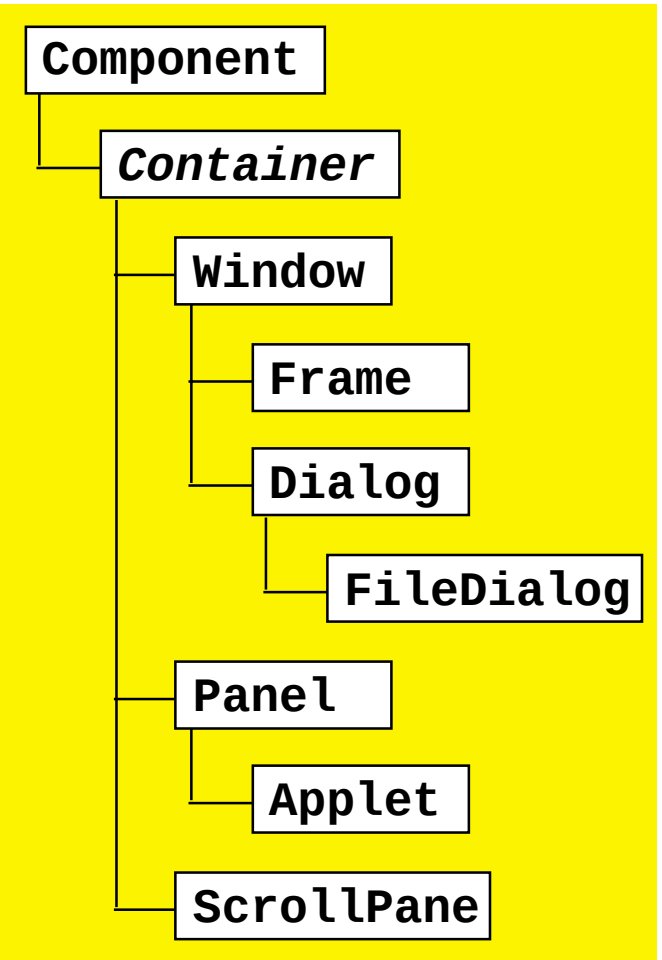

```
import javax.swing.*;

class bjm {
  public static void main(String[] args) {
    JOptionPane.showMessageDialog(null, "Bonjour, monde");
    System.exit(0);
  }
}
```

- Le programme est *dirigé par événements*
  - un thread dédié: **`EventDispatchThread`** gère la distribution des événements
  - le programme ne termine pas implicitement, d’où le **`System.exit(0)`**

Les conteneurs

- **`Container`** classe abstraite, responsable du layout
- **`Window`** pour interaction avec le système
- **`Frame`** fenêtre principale d'application
- **`Panel`** contient des composants
- **`Applet`**
- **`ScrollPane`** enrobe un conteneur d'ascenseurs

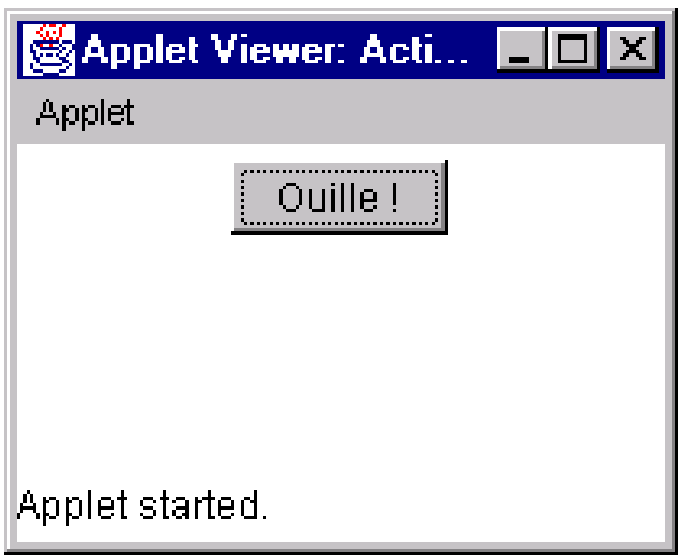
- **un *programme* étend `Frame`**
- **une *applette* étend `Applet`**



Exemple

ActionExemple.bat

```
import java.awt.*;
import java.awt.event.*;
import java.applet.Applet;

public class ActionExemple extends Applet
   implements ActionListener
{
   Button b;
   public void init() {
      b = new Button("En avant !");
      b.addActionListener(this);
      add(b);
   }
   public void actionPerformed(ActionEvent e) {
      if (b.getLabel().equals("En avant !"))
         b.setLabel("Ouille !");
      else
         b.setLabel("En avant !");
   }
}
```

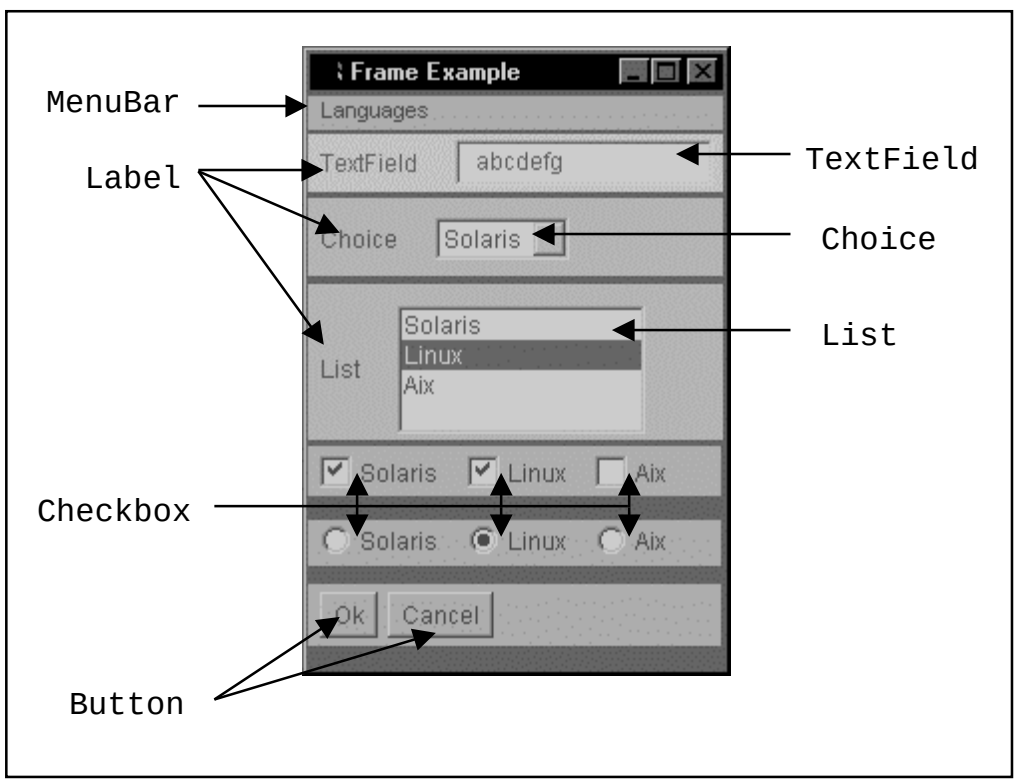


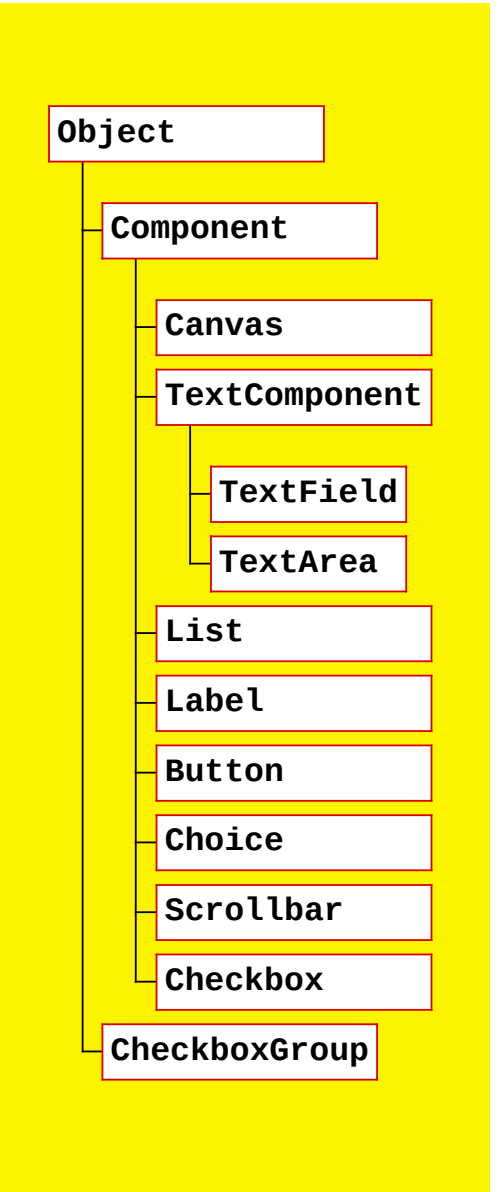
Classes d'interaction

`Canvas` pour le dessin

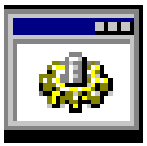
`Choices` est une sorte de combobox

`CheckboxGroup` composant logique





Menus

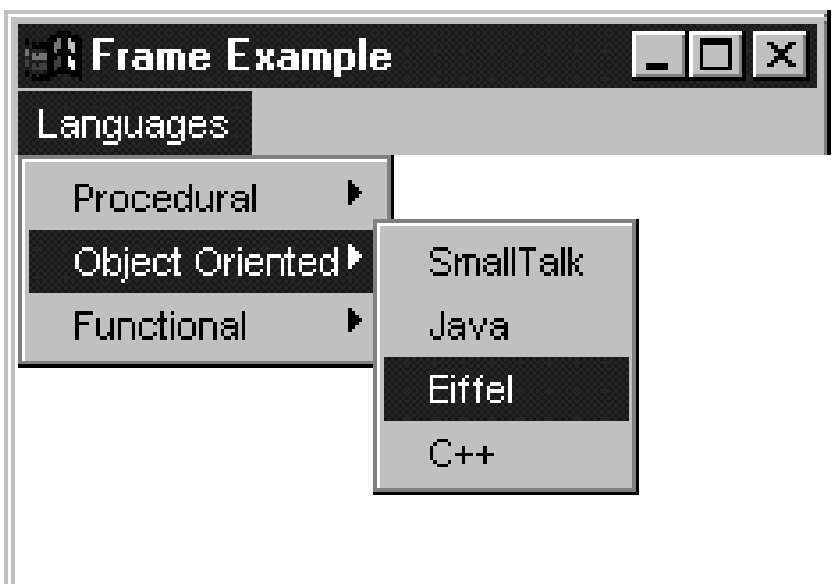


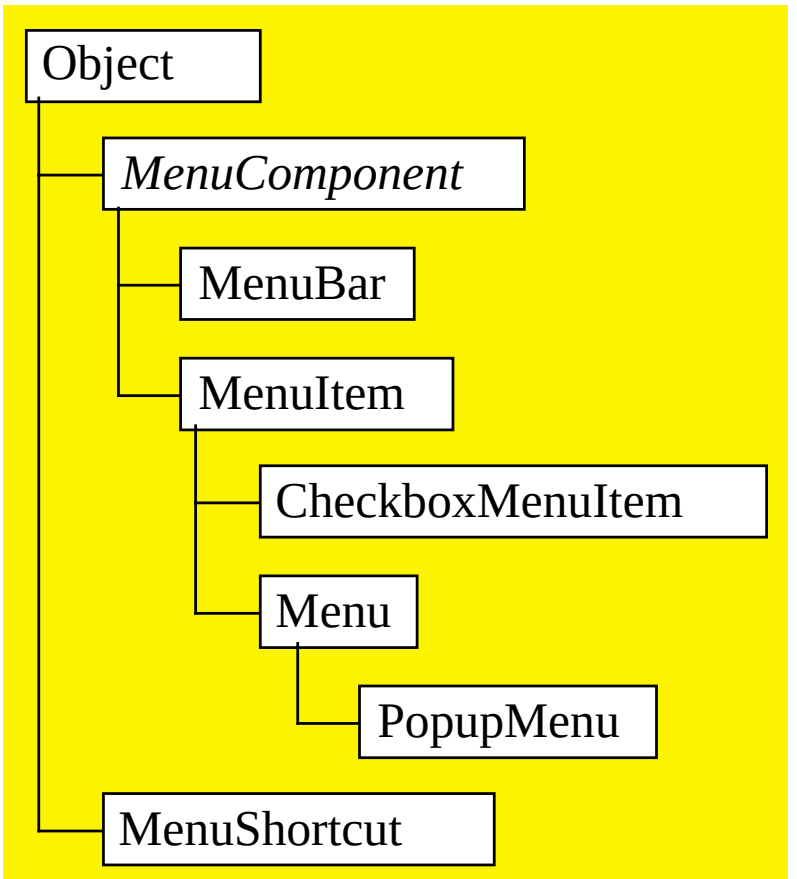
Tout.bat

MenuComponent est abstraite

Menu est une sous-classe de **MenuItem**
par le *design pattern* des conteneurs

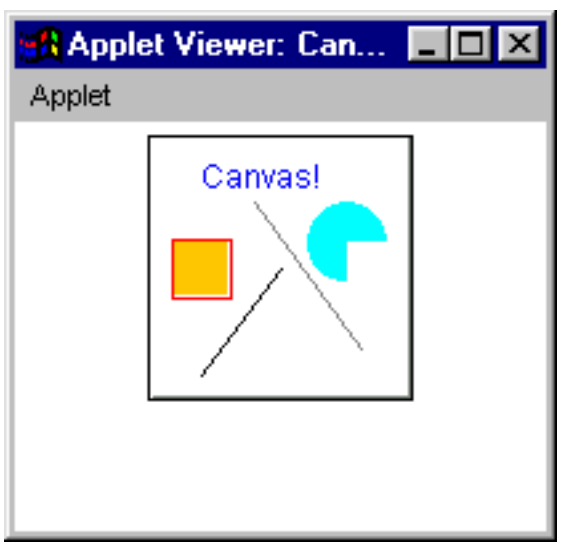
Les raccourcis sont adaptés à la plate-forme





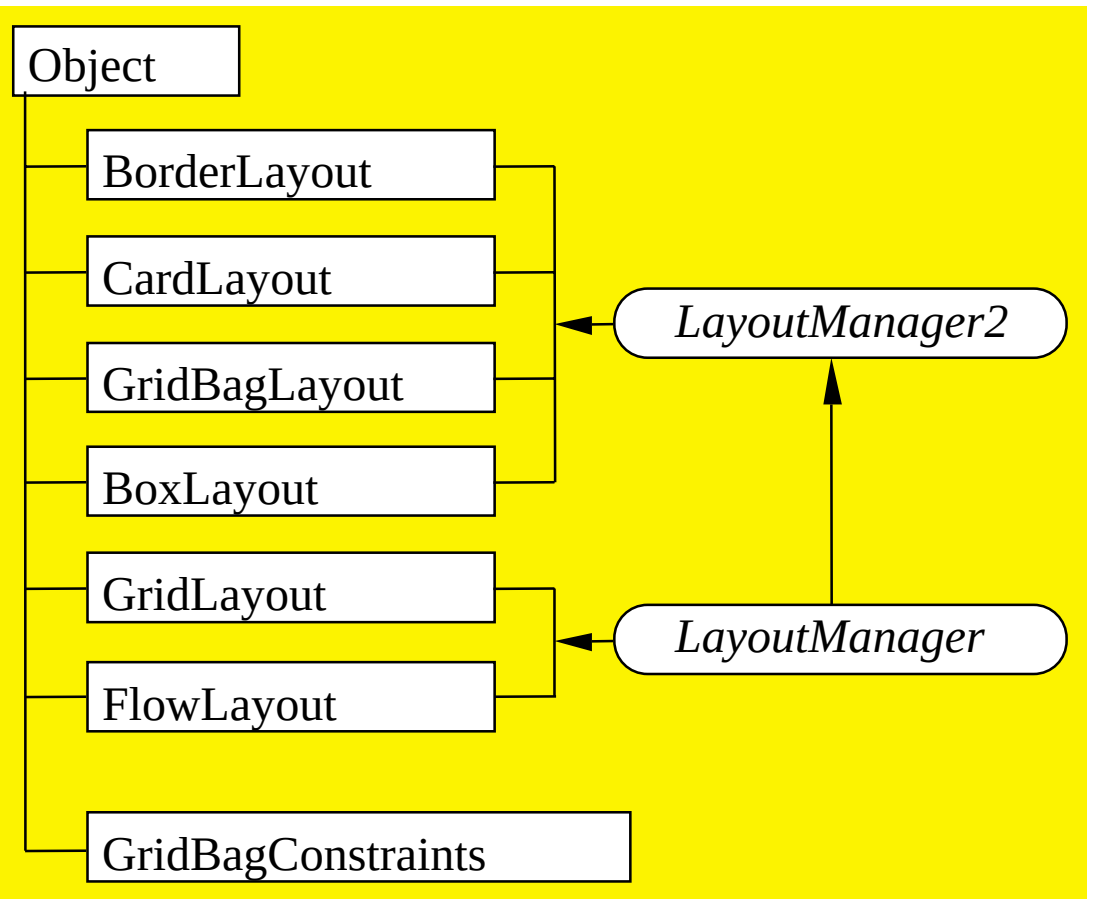
Graphique

- **`Graphics`** fournit à la fois le canal d'affichage et les outils de dessin
- **`Image`** pour les images
- **`Point`**, **`Polygon`**, **`Rectangle`**
- **`Font`**, **`FontMetrics`** pour les polices
- **`Color`**
- ***`Java2D`*** a beaucoup de possibilités



Layouts : gestionnaires de géométrie

- Gère la disposition des composantes filles dans un conteneur
- Les gestionnaires par défaut sont
  - **`BorderLayout`** pour
    - `Window`
    - `Frame`
    - `Dialog`
  - **`FlowLayout`** pour
    - `Panel`
    - `Applet`

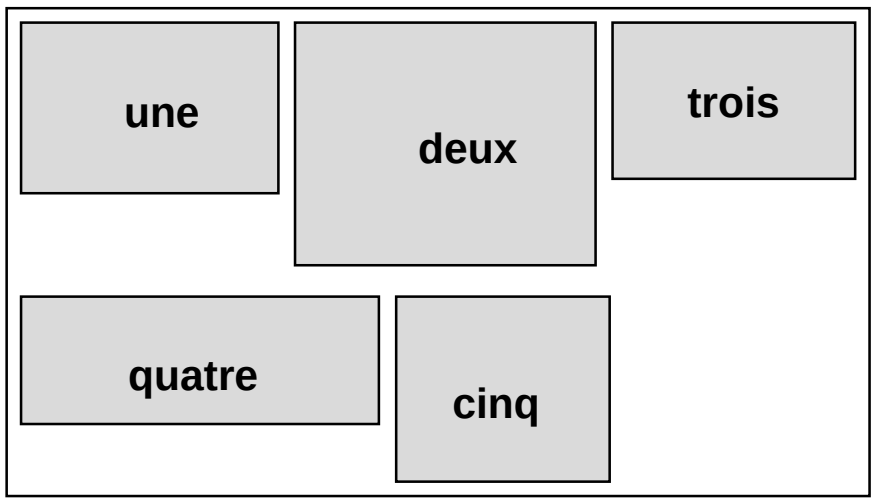


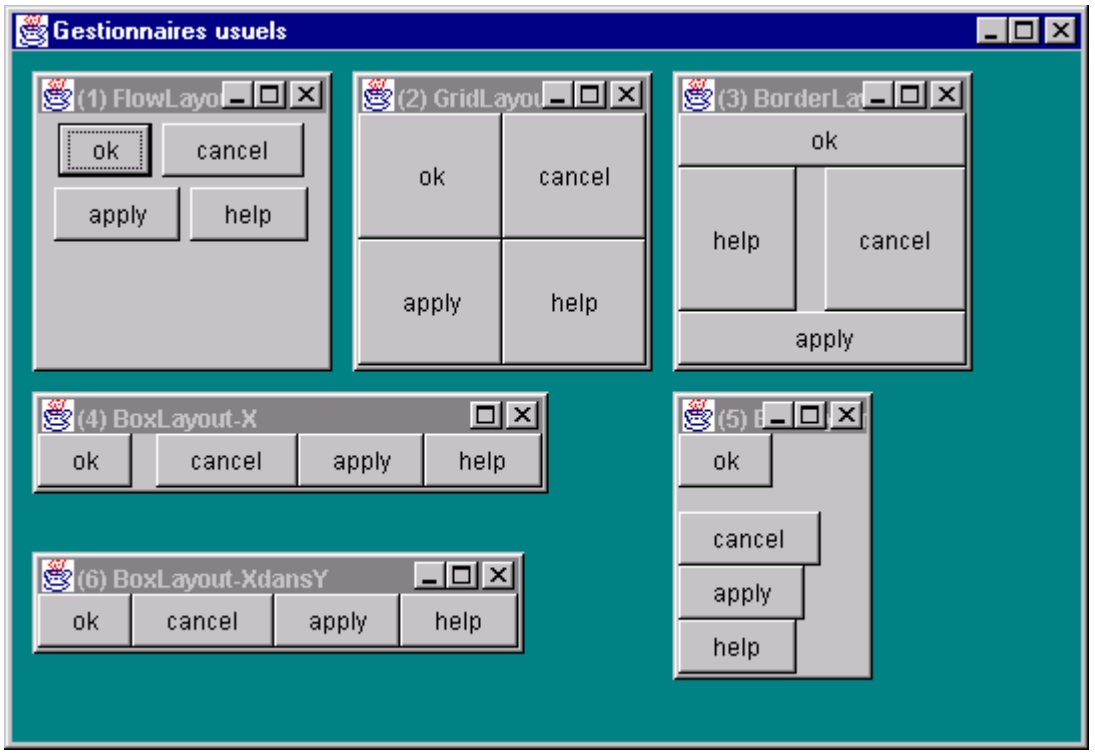
- **`BoxLayout`** est nouveau et utile
- **`LayoutManager`** et **`LayoutManager2`** sont des interfaces

Exemples

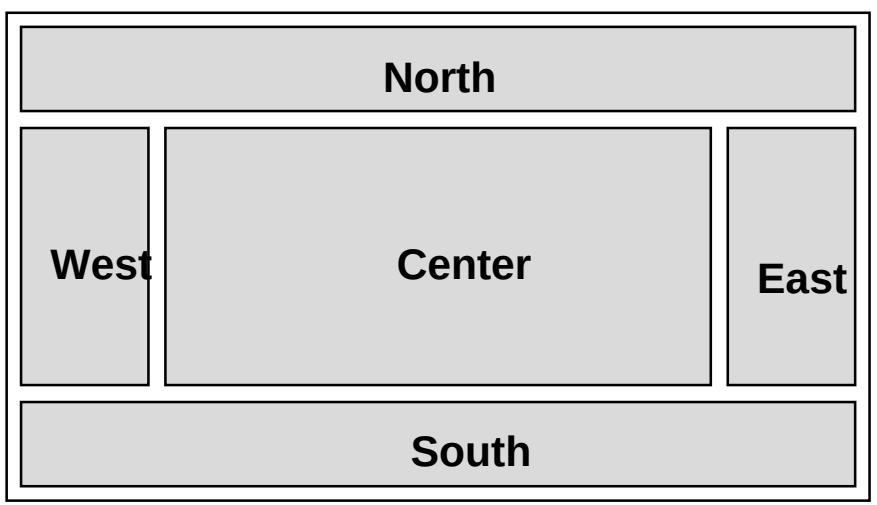
CommonLayouts.bat

- FlowLayout





- BorderLayout

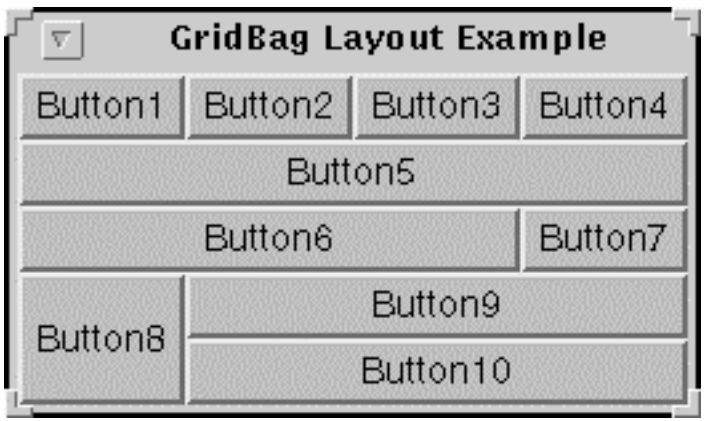


Exemples

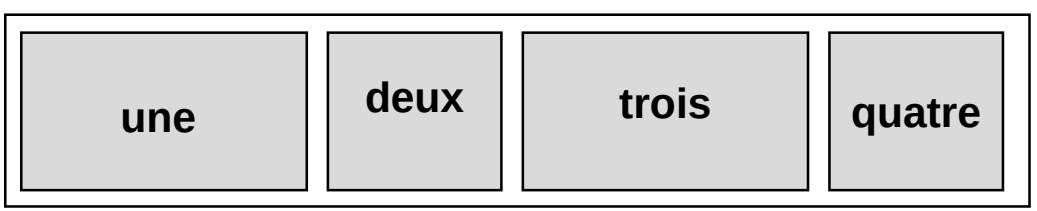
- **`GridLayout`** : chaque cellule de même taille

une	deux	trois
quatre	cinq	six

- **`GridbagLayout`** : grille, lignes et colonnes de taille variable



- **`BoxLayout`** (horizontal) : cellules de même hauteur, de largeur variable



Gestion des évènements

PlusouMoins.bat

- Un composant enregistre des *auditeurs d'évènements* (**Listeners**).
- Lorsqu'un événement se produit dans un composant, il est *envoyés* aux auditeurs enregistrés.
- Chaque auditeur définit les actions à entreprendre, dans des méthodes aux noms prédéfinis.
- Exemple:
- Un **`Button`** enregistre des **`ActionListener`**
- Lors d'un clic, un **`ActionEvent`** est envoyé aux **`ActionListener`** enregistrés.
- Ceci provoque l'exécution de la méthode **`actionPerformed`** de chaque **`ActionListener`**

Exemple

- Une classe de boutons

```
class MonBouton extends Button {
  int incr;

  MonBouton(String titre, int incr) {
    super(titre);
    this.incr = incr;
  }

  int getIncr() { return incr; }
}
```

- Une classe d'étiquettes auditrices

```
class ListenerLabel extends Label implements ActionListener {

  ListenerLabel() { super("0", Label.CENTER);}

  public void actionPerformed(ActionEvent e) {
    MonBouton b = (MonBouton)e.getSource();
    int c = Integer.parseInt(getText());
    c += b.getIncr();
    setText(Integer.toString(c));
  }
}
```

Exemple (fin)

- Le cadre

```
class PlusouMoins extends Frame {
  public PlusouMoins() {
    super("Plus ou moins");
    Button oui = new MonBouton("Plus !", +1);
    Button non = new MonBouton("Moins !", -1);
    Label diff = new ListenerLabel();
    add(oui, "North");
    add(diff,"Center");
    add(non,"South");
    oui.addActionListener((ActionListener) diff);
    non.addActionListener((ActionListener) diff);
  };

  public static void main (String[] argv) {
    Frame r = new PlusouMoins();
    r.pack();
    r.setVisible(true);
    r.addWindowListener(new WindowCloser());
  }
}
```

- Et pour fermer

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

Qui est à la manœvre ?

- Le modèle MVC (*model -view -controller*)
  - le modèle gère les données
  - la vue affiche les données
  - le contrôleur gère la communication et les mises-à-jour
- Origine : Smalltalk


```
class PlusouMoinsMVC {

   Model model;
   View view;
   Controller control;

   public PlusouMoinsMVC() {
     model = new Model();
     control = new Controller();
     view = new View(control);

     control.setModel(model);
     control.setView(view);
     view.setVisible(true);
     view.addWindowListener(new WindowCloser());
   }

   public static void main (String[] argv) {
     PlusouMoinsMVC r = new PlusouMoinsMVC();
   }
}
```

Echange de messages

```
Vue a controlleur : bouton Up !
Controlleur a modele : changer compteur de 1
Controlleur a modele : que vaut compteur ? reponse = 1
Controlleur a vue : afficher compteur dans etiquette
Vue a controlleur : bouton Up !
Controlleur a modele : changer compteur de 1
Controlleur a modele : que vaut compteur ? reponse = 2
Controlleur a vue : afficher compteur dans etiquette
Vue a controlleur : bouton Down !
Controlleur a modele : changer compteur de -1
Controlleur a modele : que vaut compteur ? reponse = 1
Controlleur a vue : afficher compteur dans etiquette
...
```

Modèle

- Le *modèle* contient la donnée
- La *mise-à-jour* par **`update()`**
- Le *renseignement* par **`getValue()`**

```
class Model {
   int compteur;
   Model() {
      compteur = 0;
   }
   void update(int incr) {
      compteur += incr;
   }
   int getValue() {
      return compteur;
   }
}
```

Vue

- La *vue* affiche les composants et les données
- La *mise-à-jour* (du texte de l'étiquette est faite par **`update()`**
- La *notification* de modifications au *contrôleur*. C'est le contrôleur qui écoute !
- Les *renseignements* sont pris par **`getValue()`**. La vue se débrouille pour les obtenir.

```
class View extends Frame {
  Button oui = new Button("Up !");
  Button non = new Button("Down !");
  Label diff = new Label("0",
    Label.CENTER);

  public View(Controller c) {
    super("Plus ou moins");
    add(oui,BorderLayout.NORTH);
    add(non,BorderLayout.SOUTH);
    add(diff,BorderLayout.CENTER);
    oui.addActionListener(c);
    non.addActionListener(c);
    pack();
  }
  void update(int compte) {
    diff.setText(Integer.toString(compte));
  }
  int getValue(ActionEvent e) {
    Button b = (Button)e.getSource();
    return (b == oui) ? 1 : -1;
  }
}
```

Contrôleur

- Réveillé par les actions produites dans la vue
- Récupère des information dans la vue
- Met à jour dans le modèle
- Récupère la nouvelle valeur dans le modèle
- La transmet pour affichage à la vue

```
class Controller implements ActionListener {
   private Model m;
   private View v;

   public void actionPerformed(ActionEvent e) {
      m.update(v.getValue(e));
      v.update(m.getValue());
   }
}
```

Applettes : généralités

- Une ***applette*** est une **`Panel`** spécialisée. Elle est incorporé dans un conteneur pour exécution.
- Une applette n'est donc pas une application autonome.
- Un *"applet context"* est un objet contenant des informations sur l'environnement dans lequel s'exécute l'applette.
- Une applette est *active* ou *inactive:*
  - au chargement, inactive
  - au premier affichage, devient active
  - selon le contexte, devient inactive
    - si elle sort de la partie visible de la page
    - si une autre page est activée
- Méthodes de gestion
  - **`init`**`()` création, inactive
  - **`start`**`()` mise en route, activation
  - **`stop`**`()` arrêt, désactivation
  - **`destroy`**`()`quand on quitte le programme.

Balise d'applette

- Dans une page HTML, balises **<applet>** et **<param>.**
- Attributs de la balise **<applet>:**

  name : donne un nom à une applette; permet de distinguer deux occurrences de la même applette;
  codebase : adresse URL où se trouve le binaire, par défaut "."
  code : la classe de l'applette; le nom local, sans répertoire.
- Attributs de la balise **<param>:**

  name : nom d'un paramètre
  value : la valeur associée au paramètre.

- ***Balisage minimal*** :
  code, largeur, hauteur

```
<applet code=Bonjour
        width=200 height=200>
</applet>
```

```
<APPLET
   [CODEBASE = url du répertoire]
   CODE = fichier de l'applette
   WIDTH = largeur du conteneur
   HEIGTH = sa hauteur
   [ALT = texte de remplacement]
   [ARCHIVE = fichiers archives]
   [NAME = nom de l'instance de l'applette ]
   [ALIGN = top, middle, left (dans la page)]
   [VSPACE = marges haute et basse]
   [HSPACE = marges gauche et droite]
>
[<PARAM NAME = nom  VALUE = sa valeur>]
[<            ...  >]
["Remplaçant-html" si balise APPLET inconnue]
</APPLET>
```

Méthodes d'applettes

- **`URL getCodeBase()`** : le répertoire contenant le fichier suffixé **`class`** de l'*applette*.

- **`URL getDocumentBase()`** : le répertoire contenant le *fichier html* contenant la balise `<applet>`.

- **`String getParameter(String)`** : retourne la valeur associée au paramètre:

```
Image i = getImage(getDocumentBase(),getParameter("image"));
```

Obtenir une image

- La classe abstraite **`java.awt.Image`** étend **`Object`**.
- Les images sont obtenus, dans les applettes, par la méthode **`java.applet.Applet.getImage()`**
- Un producteur d'images (***`ImageProducer`***) les fournit,
- Un observateur d'images (***`ImageObserver`***) les récupère.
- L'interface ***`ImageObserver`*** est implémentée par **`Component`**, et tout composant graphique peut donc "observer" des images.
- Le chargement d'images est ***asynchrone*** (sauf en Swing).

Afficher une image

- C'est un **`Graphics`** qui affiche l'image, par:

```
g.drawImage(image, ...,  observateur)
```

- L'observateur est le composant où s'affiche l'image.
- Il y a 6 variantes de **`drawImage`**, la plus simple indique l'origine de la zone d'affichage, la plus sophistiquée permet d'afficher une partie de l'image dans un rectangle spécifiée.

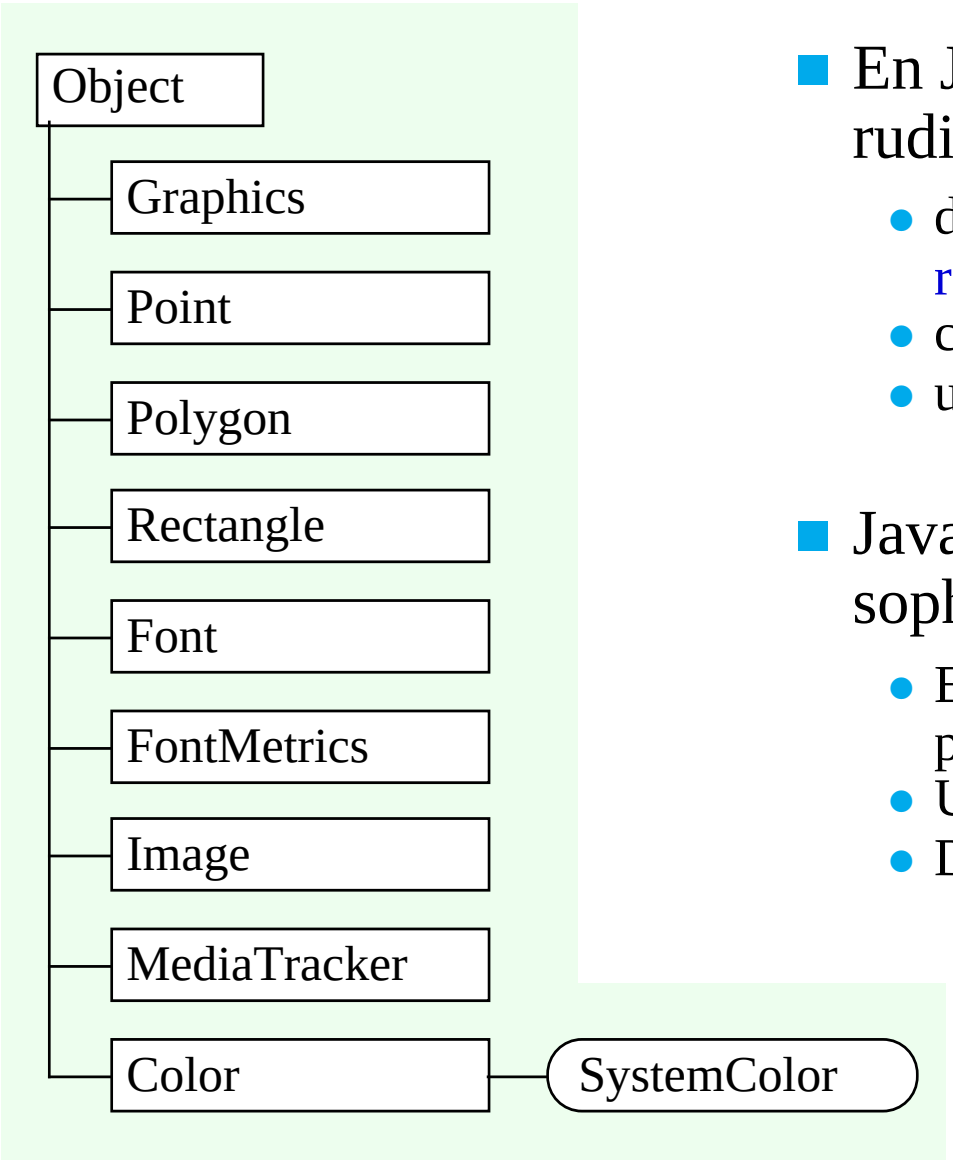
```
public void paint(Graphics g) {
   g.drawImage(im,0,0,this);
}
```

Un exemple (en applette)

- Package **net** pour les URL.
- Les dimensions de l'applette sont indiquées dans le fichier **html**
- L'image est prise où elle se trouve.

```
import java.net.URL;
import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Image;

public class ImageTestAppletSimple extends Applet {
   private Image im;
   public void init() {
       URL cb = getCodeBase();
       im = getImage(cb, "saint.gif");
   }
   public void paint(Graphics g) {
     g.drawImage(im,0,0,this);
   }
}
```

Le dessin



- En Java 1.1, les outils de dessin sont assez rudimentaires .
  - des méthodes **`draw`**`*()` et **`fill`**`*()` pour lignes, rectangles, ovales, polygone;
  - choix de deux modes de dessin : **`direct`** ou **`xor`**;
  - une zone de découpe (clipping) rectangulaire.
- Java 2 propose des possibilités très sophistiquées.
  - En particulier, le “double buffering” est automatique par défaut dans les classes Swing.
  - Une hiérarchie de classes **`Shape`**
  - Des possibilités similaires à PostScript (path’s)

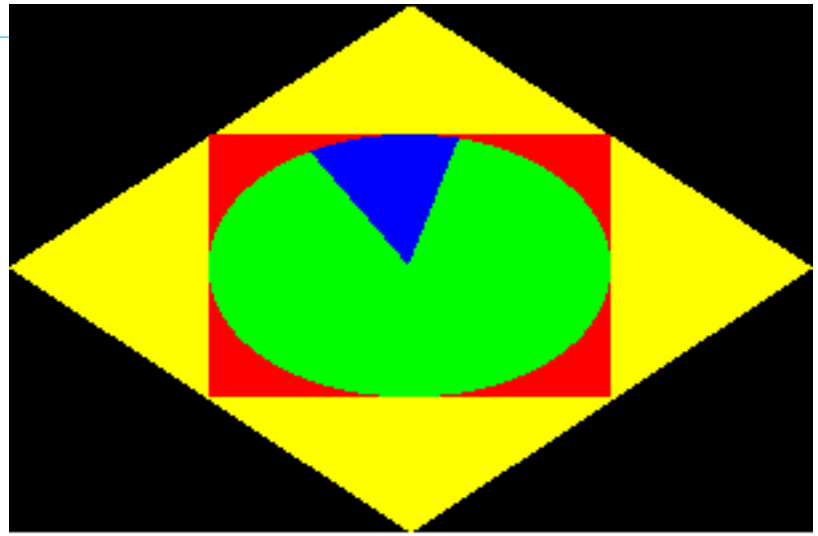
Contexte graphique

- L'outil de dessin est le ***contexte graphique***, objet de la classe **`Graphics`**. Il encapsule l'information nécessaire, sous forme d'***état graphique***.

  Celui-ci comporte
  - la zone de dessin (le ***composant***), pour les méthodes **`draw*()`** et **`fill*()`**
  - une éventuelle translation d'origine
  - le rectangle de découpe (***clipping***)
  - la couleur courante
  - la fonte courante
  - l'opération de dessin (simple ou xor)
  - la couleur du xor, s'il y a lieu.
- Chaque composant peut accéder implicitement et explicitement à un contexte graphique.
- Java 2D utilise la classe **`Graphics2D`** qui dérive de **`Graphics`**.

Exemple

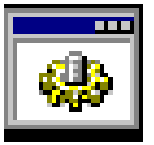

Losange.bat



```
public void paint(Graphics g) {
  int largeur = getSize().width;
  int hauteur = getSize().height;
  int dl = largeur/2, dh = hauteur/2;
  int [] polx = { 0, dl, largeur, dl};
  int [] poly = {dh, 0, dh, hauteur};
  Polygon pol = new Polygon(polx,poly,4);

  g.setColor(Color.black);
  g.fillRect(0,0,largeur,hauteur);
  g.setColor( Color.yellow);
  g.fillPolygon(pol);
  g.setColor( Color.red);
  g.fillRect(dl/2, dh/2, dl,dh);
  g.setColor( Color.green);
  g.fillOval(dl/2, dh/2, dl,dh);
  g.setColor( Color.blue);
  g.fillArc(dl/2, dh/2, dl, dh, th, del);
}
```

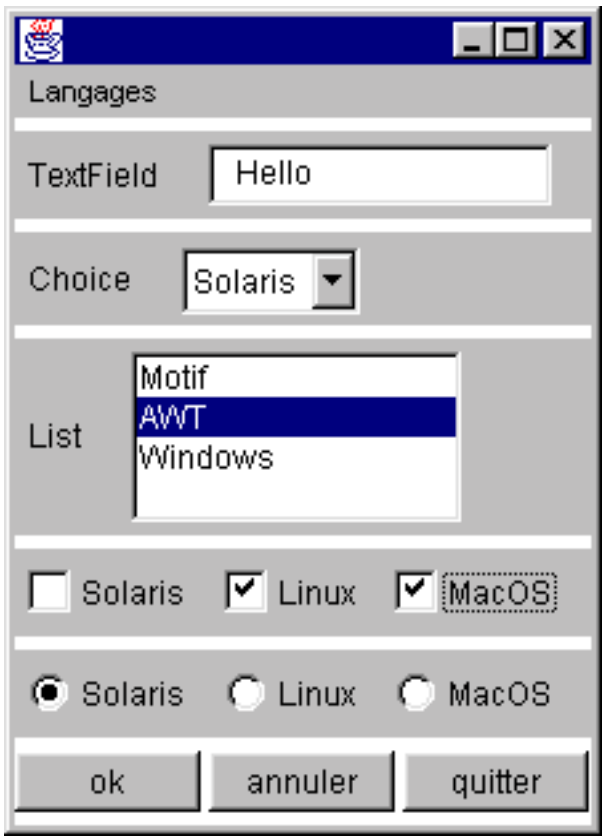
```
public class Losange extends Applet{
  int th = 45, del =45;
  public void init(){
    addMouseListener(new MouseAdapter() {
      public void mousePressed(MouseEvent e){
        th = (th +10)%360;
        repaint();
      }
    });
  }
  public void paint(Graphics g) {...}
}
```

Un exemple détaillé



Tout.bat

- L'exemple comporte
  - un menu
  - des panneaux (**`Panel`**)
- Les panneaux contiennent
  - des étiquettes (**`Label`**)
  - un champ de texte (**`TextField`**)
  - une liste (**`List`**)
  - des boutons à cocher (**`CheckBox`**)
  - des boutons radio
  - des boutons ordinaires
- Des auditeurs (**`ActionListener`**) pour réagir



Structure globale

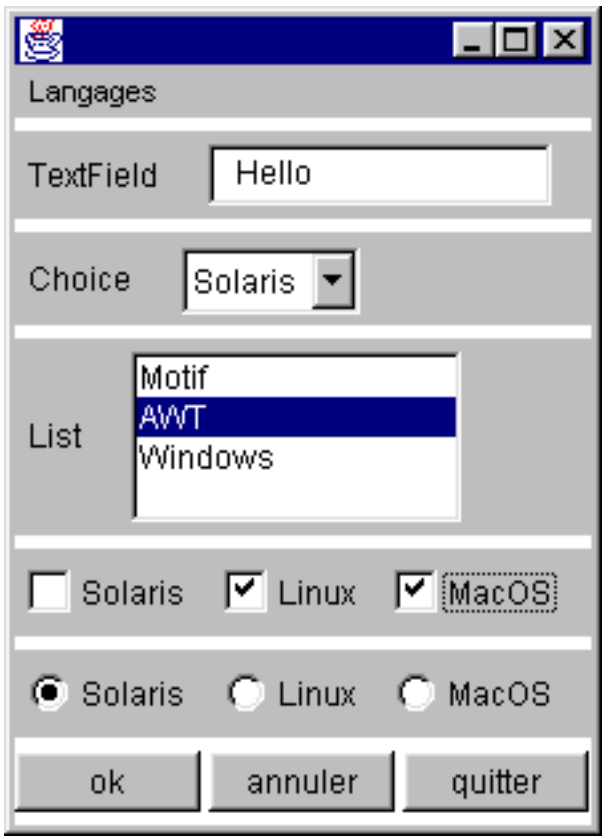
- Le constructeur **`Tout()`** met en place les composants
- Les variables servent dans l'audition
- Tout est son propre auditeur pour le bouton **`ok`**
- Le **`Fermeur`** permet de fermer par le système
- Les actions sur le menu sont auditionnées séparément

```
class Tout extends Frame
  implements ActionListener
{
  TextField tf;
  Choice ch;
  List ls;
  Checkbox cbSolaris, cbLinux, cbMacOs;
  CheckboxGroup rbGroup;
  Tout() {
    // TextField
    // Choice
    // List
    // Checkbox
    // Radiobox
    // Buttons
    // Menu
    // Layout
   }

  public static void main(String[] args) {...}
  public void actionPerformed(ActionEvent e) {...}
}

class Fermeur extends WindowAdapter {...}
class MenuActionListener
  implements ActionListener {...}
}
```

Texte et options

```
Tout() {
  // TextField
  Panel tfPanel = new Panel(
    new FlowLayout(FlowLayout.LEFT));
  tfPanel.setBackground(Color.lightGray);
  tfPanel.add(new Label("TextField"));
  tf = new TextField(15);
  tfPanel.add(tf);
```

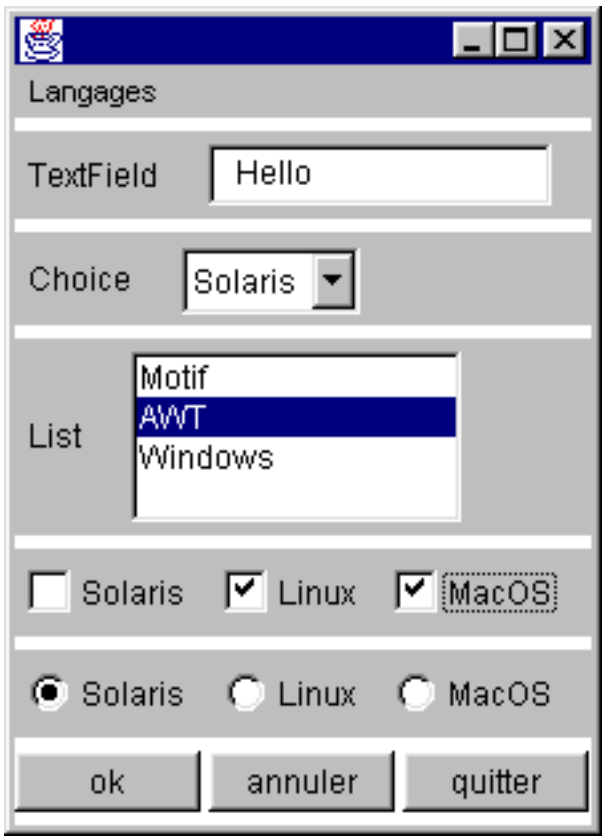
```
// Choice
Panel chPanel = new Panel(
  new FlowLayout(FlowLayout.LEFT));
chPanel.setBackground(Color.lightGray);
chPanel.add(new Label("Choice"));
ch = new Choice();
ch.addItem("Solaris");
ch.addItem("Linux");
ch.addItem("MacOS");
chPanel.add(ch);
```



Liste et boutons à cocher

```
// List
Panel lsPanel = new Panel(
  new FlowLayout(FlowLayout.LEFT));
lsPanel.setBackground(Color.lightGray);
ls = new List();
ls.add("Motif");
ls.add("AWT");
ls.add("Windows");
lsPanel.add(new Label("List"));
lsPanel.add(ls);
```

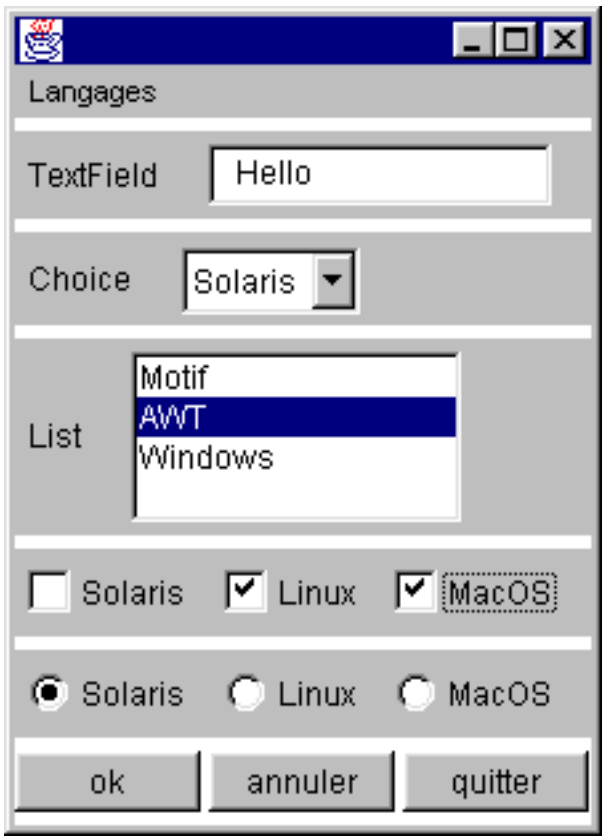
```
// Checkbox
Panel cbPanel = new Panel(
  new FlowLayout(FlowLayout.LEFT));
cbPanel.setBackground(Color.lightGray);
cbPanel.add(cbSolaris = new Checkbox("Solaris"));
cbPanel.add(cbLinux = new Checkbox("Linux"));
cbPanel.add(cbMacOs = new Checkbox("MacOS"));
```



Boutons radio et boutons simples

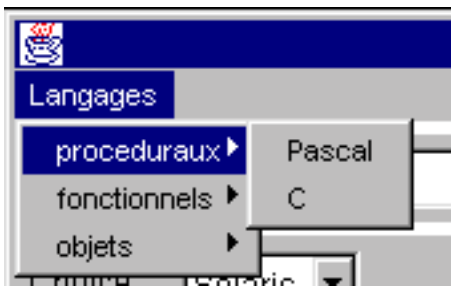
```
// Radiobox
Panel rbPanel = new Panel(
  new FlowLayout(FlowLayout.LEFT));
rbPanel.setBackground(Color.lightGray);
//CheckboxGroup (logique)
rbGroup = new CheckboxGroup();
rbPanel.add(new Checkbox("Solaris",rbGroup,true));
rbPanel.add(new Checkbox("Linux",rbGroup,false));
rbPanel.add(new Checkbox("MacOS",rbGroup,false));
```

```
// Buttons
Panel btPanel = new Panel(
  new GridLayout(1,0,3,3));
Button ok = new Button("ok");
Button cancel = new Button("annuler");
Button quitter = new Button("quitter");
btPanel.add(ok);
btPanel.add(cancel);
btPanel.add(quitter);
```



Menu

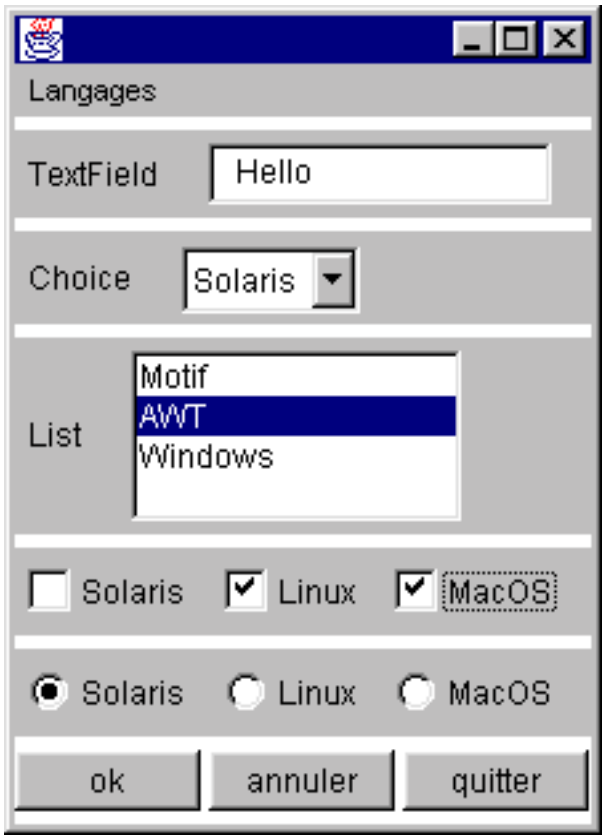
```
ActionListener doIt = new MenuActionListener();
MenuItem m;

Menu procMenu = new Menu("proceduraux");
procMenu.add(m = new MenuItem("Pascal"));
m.addActionListener(doIt);
procMenu.add(m = new MenuItem("C"));
m.addActionListener(doIt);

Menu functMenu = new Menu("fonctionnels");
functMenu.add(m = new MenuItem("Lisp"));
m.addActionListener(doIt);
functMenu.add(m = new MenuItem("ML"));
m.addActionListener(doIt);

Menu objMenu = new Menu("objets");
objMenu.add(m = new MenuItem("C++"));
m.addActionListener(doIt);
objMenu.add(m = new MenuItem("Smalltalk"));
m.addActionListener(doIt);

Menu langagesMenu = new Menu("Langages");
langagesMenu.add(procMenu);
langagesMenu.add(functMenu);
langagesMenu.add(objMenu);

MenuBar bMenu = new MenuBar();
bMenu.add(langagesMenu);
setMenuBar(bMenu);
```




```
class MenuActionListener
  implements ActionListener
{
  public void actionPerformed(ActionEvent e) {
    System.out.println("Menu : "
     + e.getActionCommand());
  }
}
```

Assemblage des panneaux

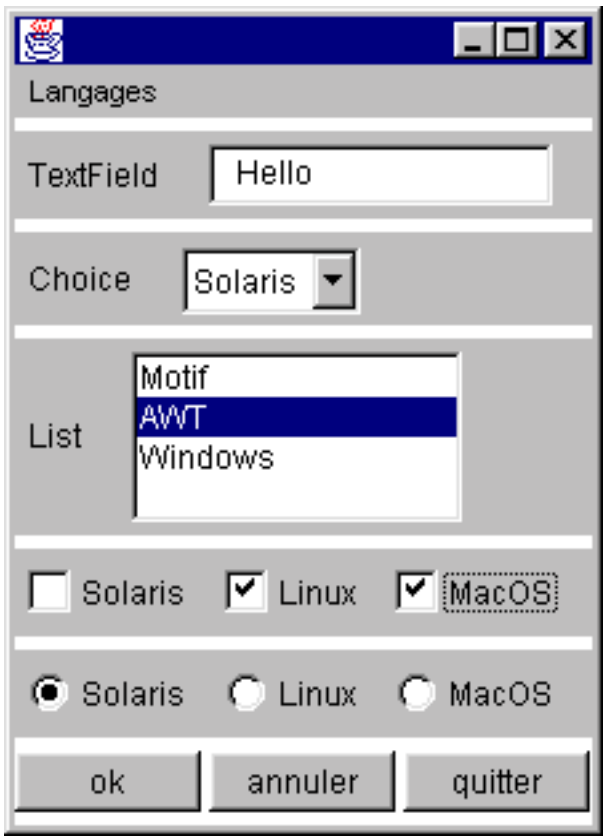
```
Tout() {
  ...
  // Layout
  setLayout(new GridBagLayout());
  GridBagConstraints gbc =new GridBagConstraints();
  gbc.fill = GridBagConstraints.BOTH;
  gbc.gridwidth = GridBagConstraints.REMAINDER;
  gbc.insets = new Insets(5,0,0,0); // top
  add(tfPanel,gbc);
  add(chPanel,gbc);
  add(lsPanel,gbc);
  add(cbPanel,gbc);
  add(rbPanel,gbc);
  gbc.insets = new Insets(5,0,5,0); // top-bottom
  add(btPanel,gbc);
}
```



- Un objet **`GridBagConstraints`** dit *comment* disposer
- NB un **`GridLayout`** produit des blocs de même taille

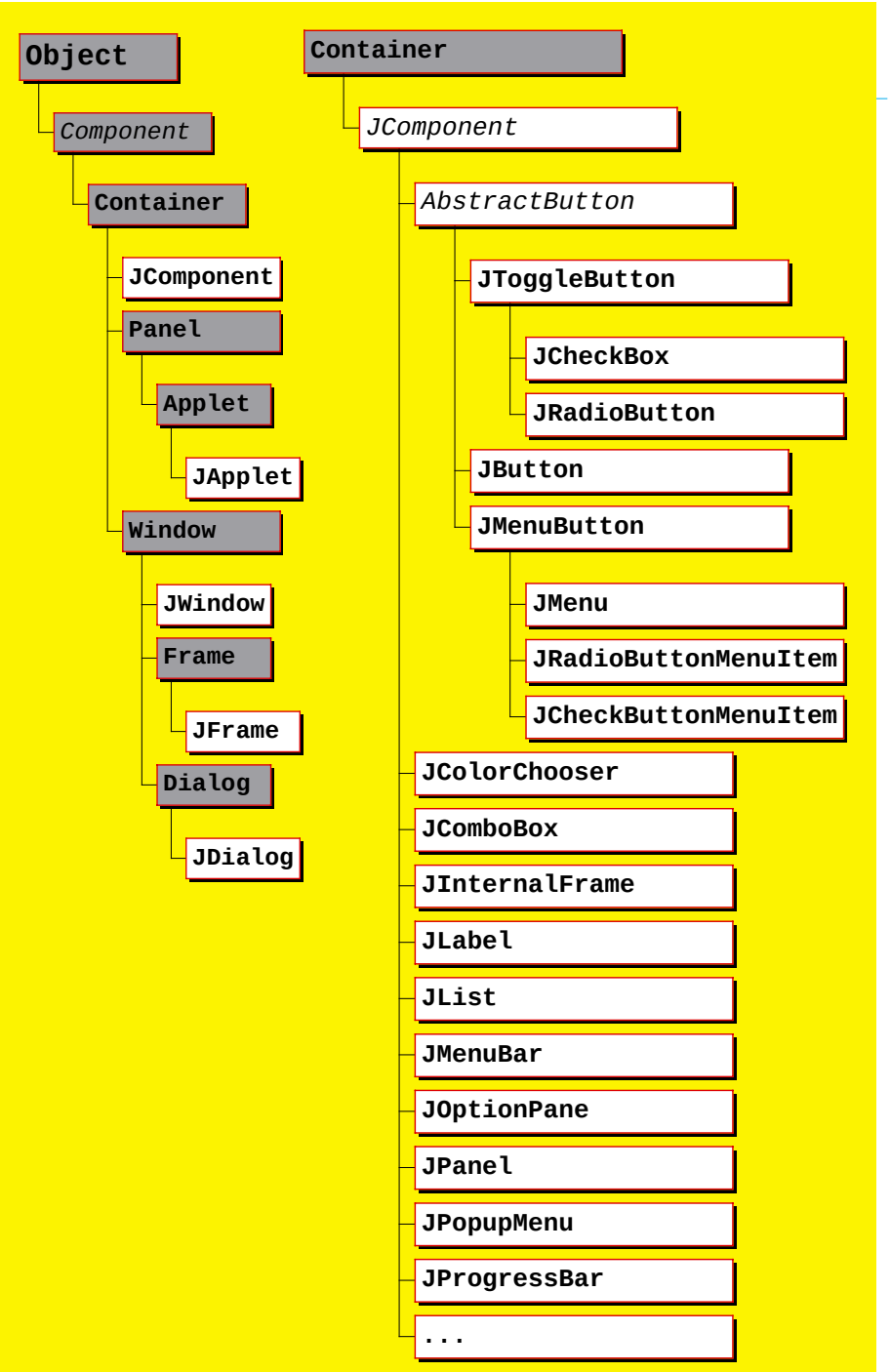
Les actions

```
public static void main(String[] args) {
  Tout t = new Tout();
  t.pack();
  t.setVisible(true);
  t.addWindowListener(new Fermeur());
}

public void actionPerformed(ActionEvent e) {
  if (e.getActionCommand().equals("quitter"))
      System.exit(0);
  System.out.println("textField = "+tf.getText());
  System.out.println("choice = "+ch.getSelectedItem());
  System.out.println("list = "+ls.getSelectedItem());
  System.out.println("checkbox = "+cbSolaris.getState()
     +" "+cbLinux.getState()+" "+cbMacOs.getState());
  System.out.println("radiobox = "
     +rbGroup.getSelectedCheckbox().getLabel());
}
```



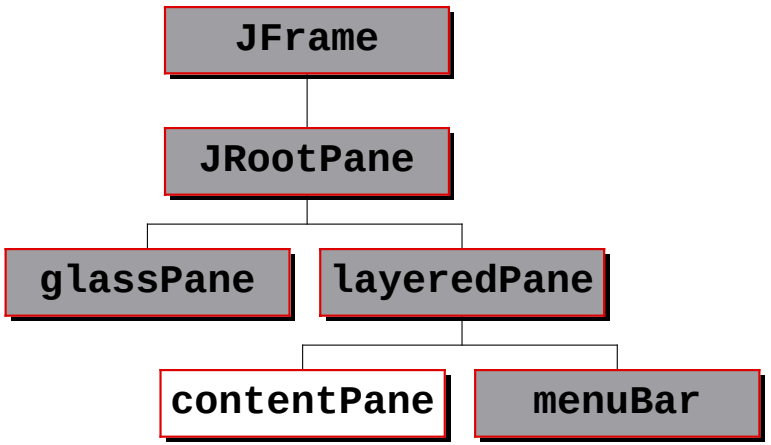
Swing

- *Swing* est une extension des AWT
  - nombreux nouveaux composants
  - nombreuses facilités
  - séparation entre
    - modèle (données)
    - aspect visuel (UI)
    - contrôle
- Les composants sont *légers*, sauf **`JApplet, JWindow, JFrame, JDialog`**
- Ces derniers ont une structure spéciale



JFrame

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

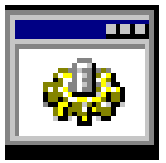



```
class Tout extends JFrame {
  Tout() {
    JPanel panel;
    getContentPane().add(panel);
    ...
  }
...
}
```

Modèles et vues

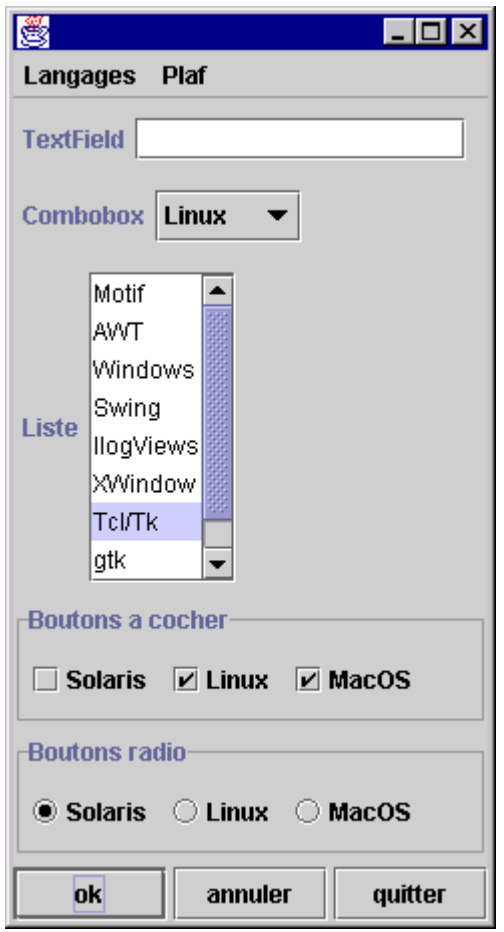
- Les composants (sauf les conteneurs) ont un *modèle* qui contient les données associées
  - **`ButtonModel`** pour les boutons
  - **`ListModel`** pour les données d'un **`JList`**
  - **`TableModel`** pour les **`JTable`**
  - **`TreeModel`** pour les **`JTree`**
  - **`Document`** pour tous les composants de texte
- La *vue* d'un composant sont aggrégés en un *délégué* **`UI`** (User Interface)
  - détermine le ***look-and-feel*** du composant (bord, couleur, ombre, forme des coches)
  - peut-être changé
  - est parfois spécifié par un dessinateur (***renderer***) à un niveau plus élevé
  - un changement global, pour tous les composants, est le ***pluggable look and feel*** (**`plaf`**)
  - trois implémentations (quatre ave le Mac) existent : Windows, CDE/Motif, Metal (défaut).
- Le *contrôle* est assuré par le modèle.

L'exemple en Swing

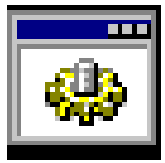


JTout.bat

- Comporte
  - deux menus (**`JMenu`**)
  - des panneaux (**`JPanel`**) organisés dans un **`BoxLayout`** vertical
- Les panneaux contiennent
  - des étiquettes (**`JLabel`**)
  - un champ de texte (**`JTextField`**)
  - une liste (**`JList`**) et une combobox (**`JComboBox`**)
  - des boutons à cocher (**`JCheckBox`**)
  - des bordures (**`TitledBorder`** ici)
  - des boutons radio (**`JRadioButton`**)
  - des boutons ordinaires (**`JButton`**)
- Trois auditeurs (**`ActionListener`**)
  - pour les boutons du bas
  - pour le menu des langages
  - pour le menu plaf

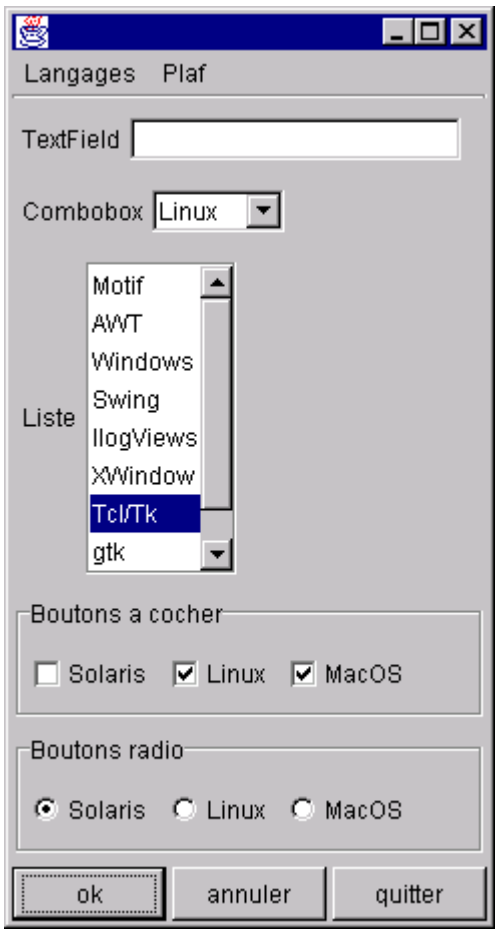


Structure d'ensemble



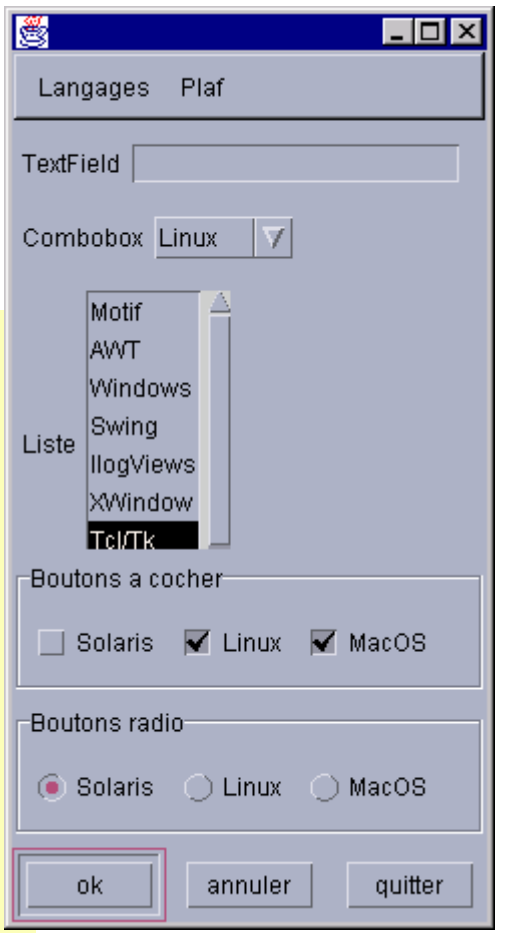
JTout.bat

```
class JTout extends JFrame
  implements ActionListener {
  JTextField tf;
  JComboBox ch;
  JList ls;
  JCheckBox cbSolaris, cbLinux, cbMacOs;
  ButtonGroup rbGroup;
  JTout() {
  // JTextField
  // JComboBox
  // JList
  // CheckBox
  // RadioBox et ButtonGroup
  // Buttons
  // Menu langages
  // Menu plaf
}
public static void main(String[] args) {
  JTout t = new JTout();
  t.pack();
  t.setVisible(true);
  t.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
```



Texte et ComboBox

- Le texte ne change pas, le composant **Choice** s'appelle **JComboBox**

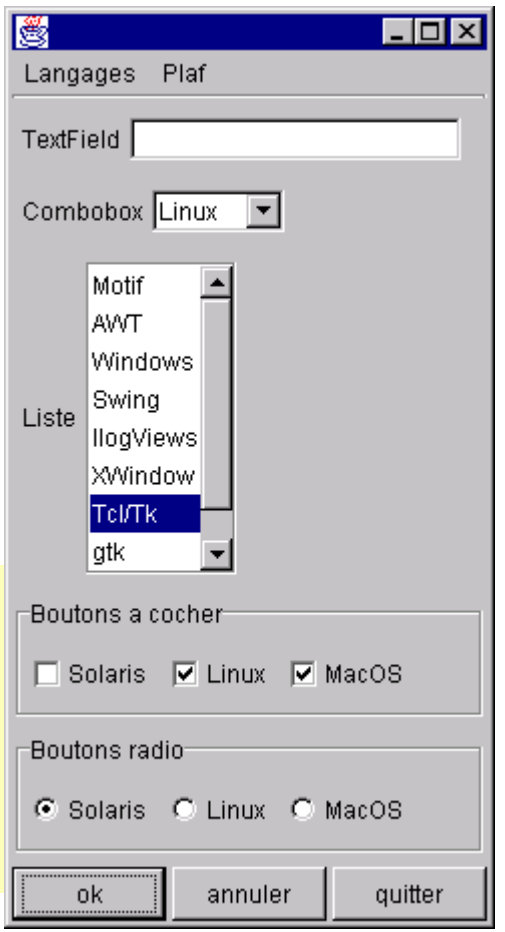



```
// JTextField
JPanel tfPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));
tfPanel.add(new JLabel("TextField"));
tf = new JTextField(15);
tfPanel.add(tf);

// JComboBox
JPanel chPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));
chPanel.add(new JLabel("Combobox"));
ch = new JComboBox();
ch.addItem("Solaris");
ch.addItem("Linux");
ch.addItem("MacOS");
chPanel.add(ch);
```

Listes

- **`JList`** prend ses données dans un **`ListModel.`** Un vecteur en est un, mais tout tableau d'objets (**`Object[]`**) est accepté dans un constructeur et converti.
- Sélection régie par un **`ListSelectionModel`**
- Affichage personnalisé par un **`ListCellRenderer`**

```
JPanel lsPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));
String[] donnees = { "Motif", "AWT", "Windows", "Swing" ,
   "IlogViews", "XWindow", "Tcl/Tk", "gtk", "gnome"};
ls = new JList(donnees);
lsPanel.add(new JLabel("Liste"));
lsPanel.add(new JScrollPane(ls)); // ascenseur
```



Ajouter et supprimer dans une liste

- On ajoute au modèle :

```
JList ls ...
ls.getModel().addElement(string); // à la fin
ls.getModel().add(position, string);
```

- On retranche du modèle :

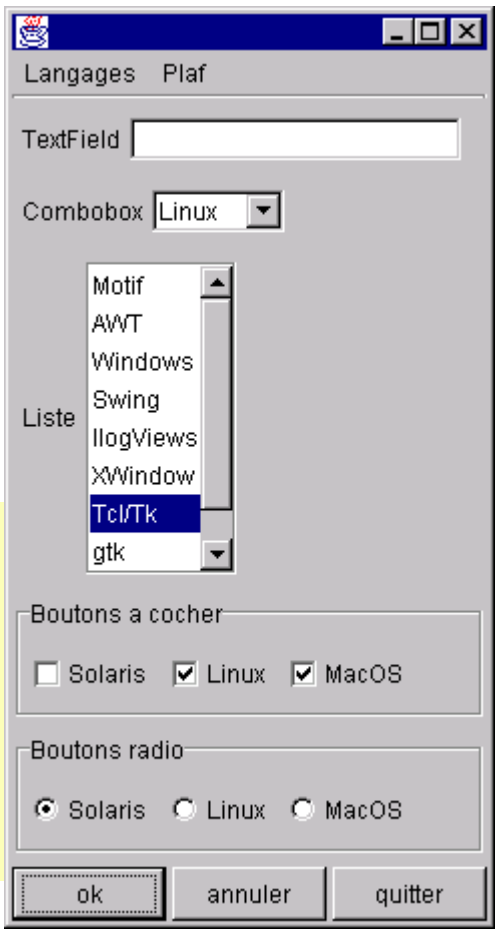
```
JList ls ...
ls.getModel().remove(position);
```

- On récupère les positions sélectionnées par:

```
int[] positions = ls.getSelectedIndices();
int position = ls.getSelectedIndex(); //premier
Object[] values = ls.getSelectedValues();
Object value = ls.getSelectedValue();
```

Boutons à cocher

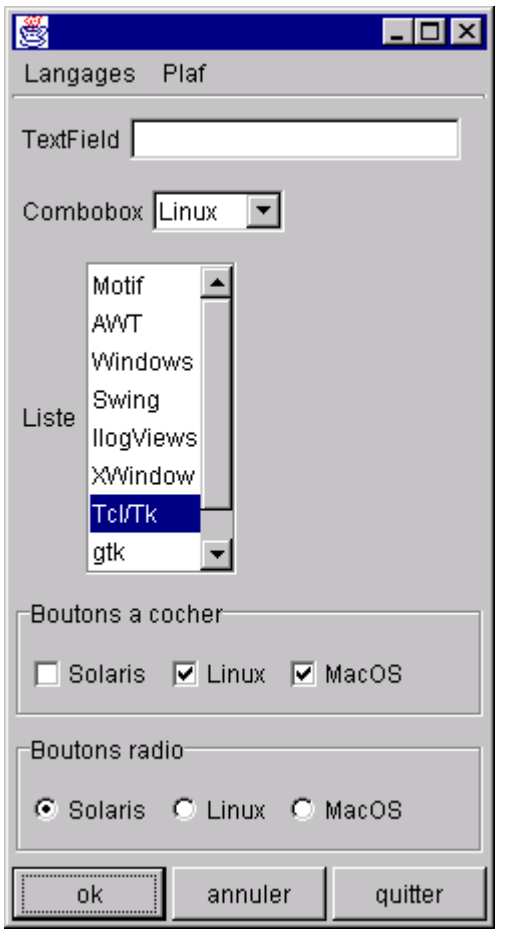
- **Checkbox** en **JCheckBox**
- De nombreuses bordures existent (importer **`javax.swing.border.*`**)




```
JPanel cbPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));

cbPanel.setBorder(new TitledBorder("Boutons a cocher"));

cbPanel.add(cbSolaris = new JCheckBox("Solaris"));
cbPanel.add(cbLinux = new JCheckBox("Linux"));
cbPanel.add(cbMacOs = new JCheckBox("MacOS"));
```

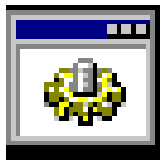
Boutons radio

- Un **`JRadioButton`** est ajouté
  - au conteneur *physique*
  - au conteneur *logique* (un **`ButtonGroup`**)




```
JPanel rbPanel =
  new JPanel(new FlowLayout(FlowLayout.LEFT));
rbPanel.setBorder(new TitledBorder("Boutons radio"));

rbGroup = new ButtonGroup();
JRadioButton rb;
rb = new JRadioButton("Solaris",true);
rb.setActionCommand("Solaris");
rbPanel.add(rb); rbGroup.add(rb);

rb = new JRadioButton("Linux",false);
rb.setActionCommand("Linux");
rbPanel.add(rb); rbGroup.add(rb);

rb = new JRadioButton("MacOS",false);
rb.setActionCommand("MacOS");
rbPanel.add(rb); rbGroup.add(rb);
```

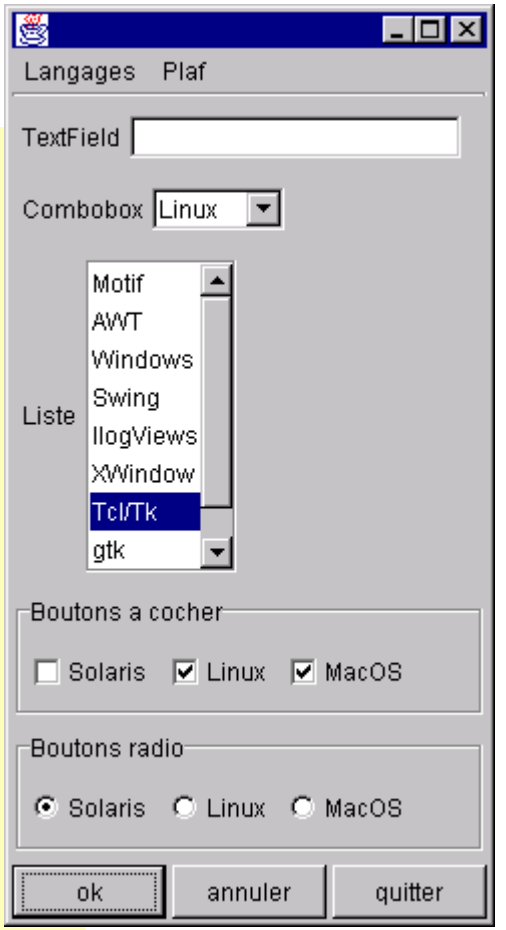
La lecture des valeurs



JTout.bat

- Le nom de la méthode dépend (encore) du composant:

```
public void actionPerformed(ActionEvent e) { // boutons
  if (e.getActionCommand().equals("quitter"))
    System.exit(0);
  System.out.println();
// Texte : getText()
  System.out.println( "textfield = " + tf.getText());
// ComboBox : getSelectedItem()
  System.out.println( "combobox = " + ch.getSelectedItem());
// Liste : getSelectedValue()
  System.out.println( "list = " + ls.getSelectedValue());
// CheckBox : isSelected()
  System.out.println( "checkbox = " + cbSolaris.isSelected()
    + " " + cbLinux.isSelected() + " " + cbMacOs.isSelected());
// Radio : group.getSelection().getActionCommand()
  System.out.println("radiobox = "
    + rbGroup.getSelection().getActionCommand() );
}
```

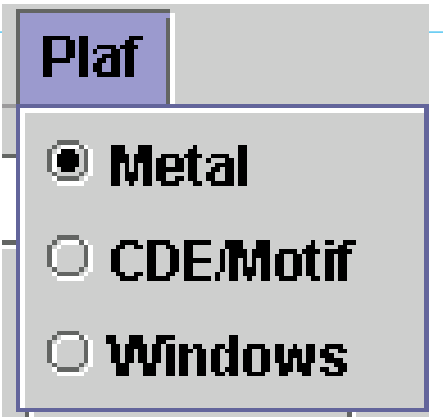


Assemblage

- Le *conteneur* est le champ **getContentPane()** de **JFrame**
- sauf pour le *menu*, que l'on ajoute illogiquement à **getRootPane()**
- Un "Strut" est un composant réduit à un espacement rigide.
- Sa création est une méthode statique de la fabrique **Box**.

```
getRootPane().setJMenuBar(bMenu);

JPanel panneau = (JPanel)getContentPane();
panneau.setLayout(
  new BoxLayout(panneau, BoxLayout.Y_AXIS));
panneau.add(Box.createVerticalStrut(5));
panneau.add(tfPanel);
panneau.add(Box.createVerticalStrut(5));
panneau.add(chPanel);
panneau.add(Box.createVerticalStrut(5));
panneau.add(lsPanel);
panneau.add(Box.createVerticalStrut(5));
panneau.add(cbPanel);
panneau.add(Box.createVerticalStrut(5));
panneau.add(rbPanel);
panneau.add(Box.createVerticalStrut(5));
panneau.add(btPanel);
```

Look and feel



- Trois “look and feel” existent, de noms

```
"com.sun.java.swing.plaf.windows.WindowsLookAndFeel"
"com.sun.java.swing.plaf.motif.MotifLookAndFeel"
"javax.swing.plaf.metal.MetalLookAndFeel”
```

- On essaye de l’utiliser par

```
UIManager.setLookAndFeel(lf);
```

- et de l’appliquer à la racine de l’arbre par

```
SwingUtilities.updateComponentTreeUI(
        SwingUtilities.getRoot(this));
```

Plaf : le menu

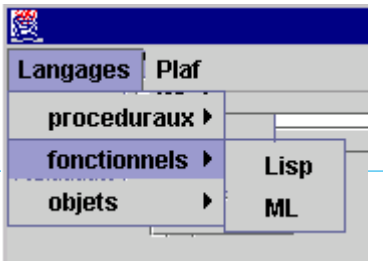
- Le menu s'écoute lui-même...

```
public class PlafMenu extends JMenu implements ActionListener {
  UIManager.LookAndFeelInfo[] lfi = UIManager.getInstalledLookAndFeels();
  public PlafMenu () {
    super("Plaf");
    ButtonGroup rbGroup = new ButtonGroup();
    JRadioButtonMenuItem rb;
    String lfs = UIManager.getCrossPlatformLookAndFeelClassName();
    String lfName;
    for (int i = 0; i < lfi.length; i++) {
      rb = new JRadioButtonMenuItem(lfi[i].getName());
      rb.addActionListener(this);
      rbGroup.add(rb);
      add(rb);
      lfName = lfi[i].getClassName();
      if (lfs.equals(lfName))
        rb.setSelected(true); // par defaut
      else
       try {
          LookAndFeel lf = (LookAndFeel) Class.forName(lfName).newInstance();
          if (!lf.isSupportedLookAndFeel()) // n'est pas "supporté" par la plate-forme
            rb.setEnabled(false);
       }
       catch ( Exception ex ) {
           rb.setEnabled(false);
       }
    }
  }...
```

Plaf : changement

```
...
public void actionPerformed(ActionEvent e) {
  String lfName = null;
  String comm = e.getActionCommand();
  for (int i = 0; i < lfi.length; i++)
    if (comm.equals(lfi[i].getName())) {
      lfName = lfi[i].getClassName();
      break;
    }
  try {
    UIManager.setLookAndFeel(lfName);
    SwingUtilities.updateComponentTreeUI(SwingUtilities.getRoot(this));
  }
  catch ( Exception ex ) {
    System.err.println( "Could not load " + lfName );
  }
}
```

Actions



- Moyen commode pour définir une entrée (dans un menu) et *simultanément* y attacher un auditeur
- Entrée simultanée dans **`Toolbar`** possible
- Tout changement dans l'un se reflète sur l'autre (grisé etc.)

```
// Menu langages
...
JMenu functMenu = new JMenu("fonctionnels");
functMenu.add(new MenuAction("Lisp"));
functMenu.add(new MenuAction("ML"));
...
langagesMenu.add(procMenu);
langagesMenu.add(functMenu);
langagesMenu.add(objMenu);
```

```
class MenuAction extends AbstractAction {
  public MenuAction(String libelle) {
    super(libelle);
  }
  public void actionPerformed(ActionEvent e) {
    System.out.println("Menu : ”
      + e.getActionCommand());
  }
}
```

AbstractAction


Texte.bat

- **`AbstractAction`** est une classe abstraite
  - elle implémente l'interface **`Action`**
  - **`Action`** étend **`ActionListener`**
  - la seule méthode à écrire est **`actionPerformed()`**
- Les conteneurs **`JMenu`**, **`JPopupMenu`** et **`JToolBar`** honorent les actions:
  - un même objet d'une classe implémentant **`AbstractAction`** peut être "ajouté" à plusieurs de ces conteneurs.
  - les diverses instances ***opèrent de concert***.
  - par exemple, un objet ajouté à un menu et à une barre d'outils est activé ou désactivé simultanément dans les deux.
- Les classes dérivées de **`AbstractAction`** sont utiles quand une ***même*** action peut être déclenchée de ***plusieurs*** manières.

Emploi d'AbstractAction

- Création d'une classe qui étend **`AbstractAction`**

```
class MonAction extends AbstractAction {
   public void actionPerformed( ActionEvent e ) {
      ...
   }
}
```

- Utilisation comme **`ActionListener`**

```
Action monAction = new MonAction();
JButton b = new JButton("Hello");
b.addActionListener(monAction);
```

- Utilisation dans un menu *et* dans une barre d'outils

```
Action copyAction = new MonAction("Copy");
JMenu menu = new  JMenu("Edit");
JToolBar tools = new JToolBar();
JMenuItem copyItem = menu.add(copyAction);
JButton copyBtn = tools.add(copyAction);
```