ULB UNIVERSITÉ LIBRE DE BRUXELLES

INFO-H-301
Programmation orientée objet

TP4
Librairie javax.Swing

F. Servais & B. Verhaegen

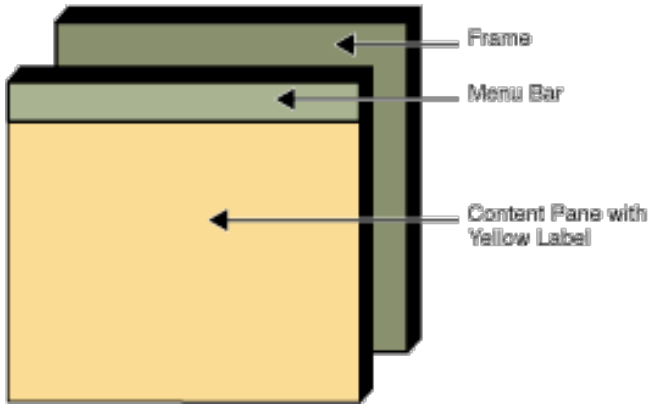
Swing

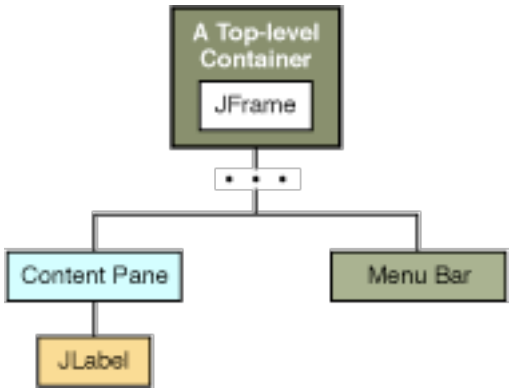
- **Swing** est la librairie standard Java pour créer des interfaces graphiques (GUI).
- http://java.sun.com/docs/books/tutorial/uiswing/components/index.html

- Fait partie des **Java Foundation Classses** (JFC) qui contiennent d'autres outils utiles aux applications graphiques :
  - *Swing GUI Components* : boutons, textes, menus, checkboxes, tableaux, boites de dialogues,etc.
  - *Java 2D API* : librairie d'affichage 2D sur laquelle se base Swing
  - *Synth* : librairie permettant de créer son propre « look and feel »
  - *Data transfert* : gestion du copier/coller, du drag and drop
  - *Internationalization* : gestion des différentes langues et caractères
  - *Accessibility API, Undo Framework API,* etc

Conteneurs Swing

- Un **composant** Swing (label, bouton, etc) ne peut se trouver dans un et un seul conteneur.
- Un **conteneur** est un composant peut contenir d'autres composant. Si le conteneur contient des conteneurs, on parle alors de *containment hierarchy*.
- Une **fenêtre** (un *Top-level Container*) est un conteneur et est implémentée dans la classe `JFrame`. Une `JFrame` est composée d'un titre, d'un `Content Pane` et éventuellement d'un `Menu Bar`.





Composants Swing

- A l'exception des *Top-level containers*, toutes les classes Swing qui commencent par J descendent de la classe `JComponent`. (`JPanel`, `JButton`, `JLabel`, ...)
- `JComponent` descend de `Container` qui descend de `Component`. Un conteneur est donc un composant.

- Tous les `JComponent` ont des méthodes communes :
  - http://java.sun.com/docs/books/tutorial/uiswing/components/jcomponent.html#api
  - Gérer l'apparence
  - Gérer l'état
  - Gérer des évenement
  - Dessiner les composants
  - Placer les composants dans des conteneurs, spécifier leur position
  - …

HelloWorld en Swing

```
import java.awt.*;
import javax.swing.*;
…
//nouvelle fenêtre (top-level container)
JFrame frame = new JFrame(”HelloWorld");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
//nouveau label jaune contenant le texte “HelloWorld”
JLabel label = new JLabel(”HelloWorld");
label.setOpaque(true);
label.setBackground(Color.YELLOW);
//ajout du label dans le ContentPane
frame.getContentPane().add(label);
//dimensionne la frame
frame.pack()
//affiche la fenêtre
frame.setVisible(true);
```

Quelques composants

- `JButton` : un bouton
- `JCheckBox` : un bouton à cocher
- `JRadioButton` : un bouton radio (doit être dans un `ButtonGroup`)
- `JLabel` : un label non éditable
- `JTextField` : un champ texte
- `JTextArea` : une zone de texte (plusieurs lignes)
- `JDialog` : boîtes de dialogue (voir `JOptionPane`)
- `JList` : liste
- `JComboBox` : liste déroulante
- `JProgressBar` : barre de progression
- `JTable`, `JTree` : tableaux et arbres (voir plus loin)
- ...

Panels

- On peut grouper des composants dans des `JPanel` (un conteneur).
- Un `JPanel` peut avoir un layout qui définira comment seront disposés les composants à l'intérieur du *conteneur*.
- Par défaut le *layout* est `FlowLayout`, c'est à dire que les composants seront placés les uns à la suite des autres en fonction de la taille de la fenêtre.
- `BorderLayout` permet d'ajouter des composants dans 5 zones différents :


```
JPanel p = new JPanel(new BorderLayout());
p.add(quelqueChose, BorderLayout.LINE_END)
```

Gestion des événements

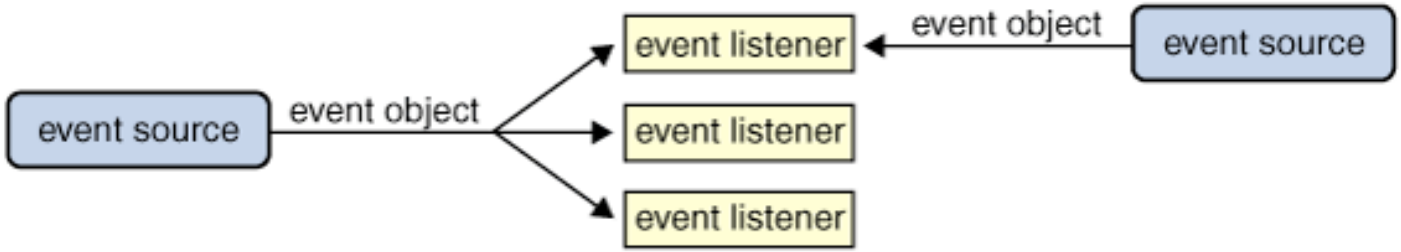
- Quand par exemple on clique sur un bouton, un **événement** est déclenché.
- Un **gestionnaire d'évenement** est donc nécessaire pour y réagir.
- Solution simple : implémenter l'interface `ActionListener` et sa méthode `actionPerformed(ActionEvent e)`
- Exemple : un programme avec un bouton qui fait *beep* quand on clique dessus.

```
public class GUI{
  JButton button;
  public GUI(){
    ...
    button.addActionListener(new BeepListener());
    ...
  }
}
public class BeepListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
     ...
     //Make a beep sound...
    }
}
```

actionPerfomed est appellé à chaque fois que l'on clique sur le bouton

Gestion des événements

- Ce système est **simple** et **flexible**.
- Par exemple, un programme pourrait créer un *listener* par source d'événement ou créer un seul *listener* pour toutes les sources ou encore avoit plus d'un *listener* pour un même événement d'une même source.



- Chaque évément est réprésenté par un objet qui donne de l'information sur l'événement et sur sa source. Les sources sont souvent des composants mais ce n'est pas toujours le cas (ex : un message reçu sur un réseau).

Types d'événements

- Les composants peuvent déclencher des **événements différents**. Le plus simple à utiliser et le plus courant est l'événement `Action`. Par contre, un `JColorChooser` peut déclencher un événement de type `Change` qui est un peu différent (pour lequel il faudra implémenter `ChangeListener`. Idem pour la souris qui peut déclencher des événements `Mouse` ou `MouseMotion`.

- Lors de l'utilisation d'un composant particuler, il est donc utile de consulter sa **documentation** (1) et celle de chaque évenement (2) qu'il peut déclencher :
  1. http://java.sun.com/docs/books/tutorial/uiswing/components/componentlist.html
  2. http://java.sun.com/docs/books/tutorial/uiswing/events/handling.html

Quelques remarques

- Une règle importante est qu'un *listener* doit pouvoir s'exécuter très **rapidement** car la gestion des évenements et du dessin d'une application s'exécutent dans le **même thread** (voir plus loin). Un *listener* lent pourrait donc figer l'application le temps de son exécution.
- Il est plus sûr d'implémenter les *listeners* dans des classes **package-private**.
- Un évenement est un **objet** qui hérite de la classe `EventObject`, par exemple `MouseEvent`.
- `Object getSource()` d'`EventObject` retourne **l'objet qui a déclenché** l'évenement. Il existe des méthodes plus spécifiques comme `getComponent()`, implémentée dans les évenement de type `ComponentEvent`.
- Certaines interfaces *listener* contiennent plus d'une méthode. Par exemple, MouseListener contient 5 méthodes : `mousePressed`, `mouseClicked`, ... Ce n'est pas une mauvaise pratique de laisser certaines méthodes vides.