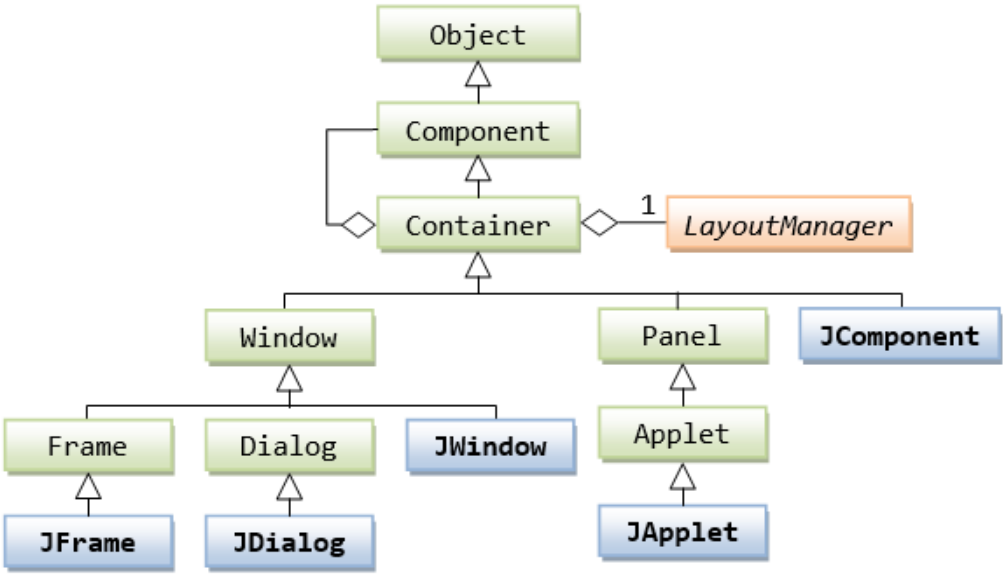
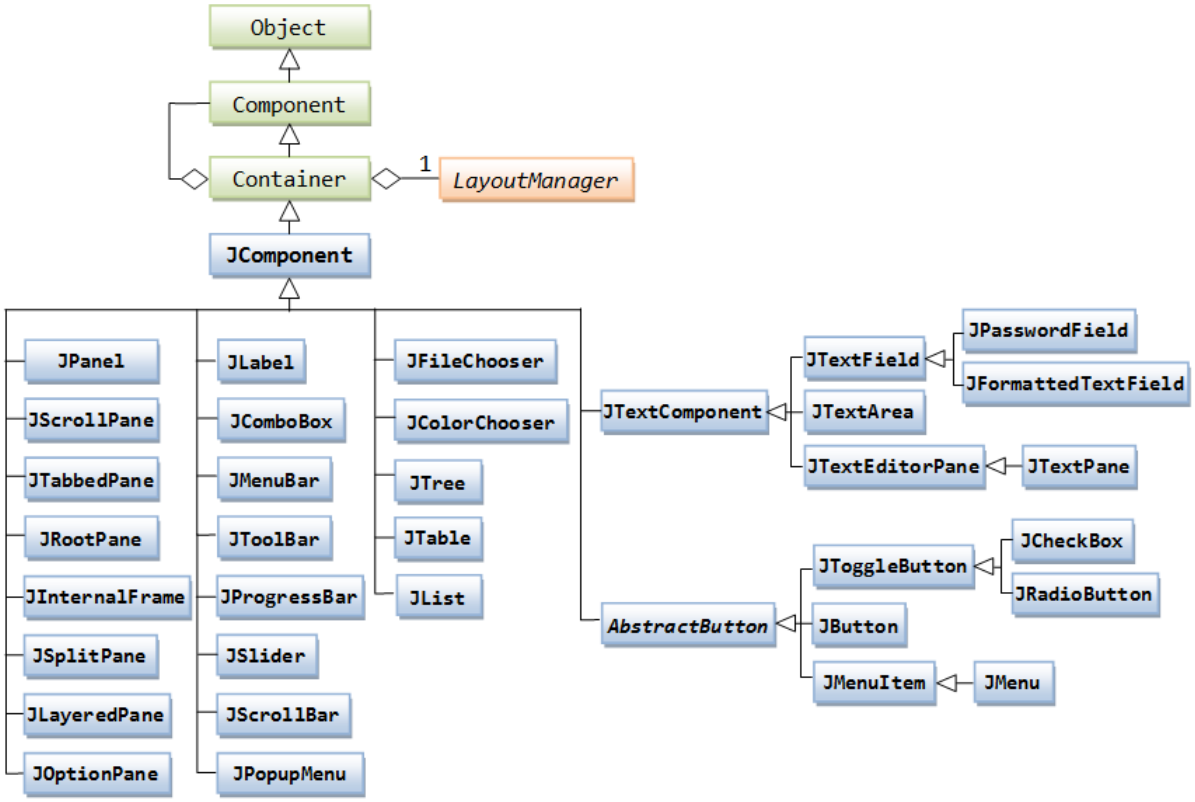
Swing

Installation des plugins Eclipse nécessaires

- Aller dans *Help|Install new software*
- Cliquer sur le bouton *Add*
- Dans *nom*, mettre WindowBuilder et dans *url*, http://archive.eclipse.org/windowbuilder/WB/integration/4.7/
- Dans la liste, cocher *WindowBuilder*
- Cliquer sur *Next*
- Encore *Next*
- Cliquer sur *I accept …* puis *Finish*
- L'installation se met en route (Ca peut être long)
- Cliquer sur *Restart now*
- Il se peut que Eclipse demande d'autres installations : suivez les instructions

Arborescence Swing





Créer un projet sous Eclipse

On crée d'abord un projet Java Standard

Puis un package pour mettre les éléments d'interface dedans (disons qu'il s'appelle *ui*).

Ensuite, click droit et *New|Other|WindowBuilder|Swing Designer|Application Window*. Cela vous créera une fenêtre JFrame (vide) avec un main() pour la lancer.

Listener et Renderer

Les Listener et Renderer sont les deux points clés d'un système de GUI sous Java.

- Le Listener sert à associer un handler (une fonction) à un évènement donné. Typiquement, on associe un Listener à un bouton pour faire quelque chose quand on appuie dessus.
- Le Renderer sert à changer le comportement graphique par défaut d'un composant. Par exemple, on peut décider qu'une JList affiche des images (au lieu/en plus du texte comme c'est par défaut). Le Renderer est propre à l'objet dont on veut changer le comportement graphique.

On effectue souvent le codage des Listener *inline*.

Exemple dans le cas d'un bouton :

```
btnDelete.addActionListener(new ActionListener() {
	public void actionPerformed(ActionEvent arg0) {
		… do something
	}
});
```

Les Panels

Ce qu'il faut bien comprendre, c'est que les éléments sont positionnés non pas directement dans la fenêtre elle-même, mais dans un Panel, qui en gros doit être vu comme le contenant. La Window/JFrame (ou la Dialog) désigne en gros le pourtour (les borders, la zone de titre, etc ..) et le Panel comme la surface de la fenêtre, le contenant, donc.

Il existe donc toujours un Panel de plus haut niveau dans une JFrame (ou une JDialog). On l'appelle le contentPane, et on y accède via la fonction (de la fenêtre) getContentPane().

Evidemment, les Panels peuvent contenir d'autre Panels, etc, jusqu'à arriver au point où on positionne les effectivement les élements d'interface.

Il existe évidemment des Panel spécialisés comme le JScrollPane (pour scroller) ou le JSplitPane (pour couper la fenêtre en deux).

Du fait de ce qu'on a dit, les Layout s'appliquent aux Panels et non pas aux fenêtres.

Les Layouts

C'est un des points les plus complexes. Le layout désigne la façon dont les éléments vont s'afficher à l'écran ou plus exactement dans le container considéré.

Par défaut, on utilise plutôt l'Absolute Layout, avec x et y.

Les autres layouts couramment utilisés sont :

- Le FlowLayout : les élements se mettent les uns à la suite des autres (horizontalement ou verticalement). Il sert souvent pour définir les layout de sous-component, par exemple le layout d'une JList si on utilise un custom Renderer..
- Le BorderLayout : les élements remplissent le component à partir des bords. Il est souvent utilisé pour remplir tout un component avec un seul élément.
- Ils existent d'autres layouts (BoxLayout, GridLayout, etc) mais ils sont complexes et peu utilisés.

Quelques éléments de base

Le JLabel

C'est simplement un texte qu'on ne peut pas éditer. Pour mettre ou récupérer un texte, on fait setText()/setText(). On peut aussi positionner le texte via le constructeur : JLabel label = new JLabel("Ceci un est label")

Le JTextField

C'est pareil, mais le champ est éditable

Le JButton

C'est un bouton.

```
JButton b = new JButton("Ceci un est label") ;
```

Le Listener associé est un ActionListener

La JList

C'est une liste. A noter qu'une liste doit être paramétrisée :

```
JList<Person> list = new JList<Person>();
```

Ici on a une liste de Person.
Une liste fonctionne avec ce qu'on appelle un Model qui indique sur quelles données travaille la JList. Typiquement, une JList<Person> travaille avec une List<Person>.
Mais ce n'est pas aussi simple : il faut créer un objet Model qui encapsule la List de données. Il faut donc créer un Model qui dérive de AbstractListModel (paramétrisé, donc AbstractListModel<Person> dans l'exemple précédent).
Ce Model doit implémenter 2 méthodes :

- **public int** getSize() : renvoie la taille du Model (donc de la JList). Si on emploie une List, on renvoie simplement la taille de la List ;
- **public** Object getElementAt(**int** index) : renvoie l'élément à l'index *index*. Dans l'exemple précédent, Object est évidemment une Person.

A noter que ce qui s'affiche dans la liste est simplement le toString() de l'objet renvoyé par getElementAt(). Si on veut afficher autre chose, il faut créer un Renderer.
Pour associer un Model à une JList, on fait : jlist.setModel(model) ;
Si on ajoute dynamiquement des éléments dans le Model, pour qu'il apparaissent dans la JList, il faut faire : jlist.updateUI() ;

A noter aussi que si on veut que la JList puisse scroller , il faut l'inclure dans un JScrollPane (et préciser comment scrolle ce JScrollPane).

Les Listener associés sont :

- ListSelectionListener : Appelé lorsqu'on sélectionne quelque chose dans la liste.
- MouseAdapter : Appelé par addMouseListener. Pour faire des choses un peu plus complexes que précédemment (par exemple pour détecter un double-click).

Pour accéder aux éléments sélectionnés :

- int getSelectedIndex (); (int[] getSelectedIndices() ; en cas de sélection multiple)
- Object getSelectedValue() ; (Object[] getSelectedValues() ; en cas de sélection multiple)
- boolean isSelectedIndex(int index) ;

Pour afficher une liste dans un ScrollPane, il faut faire :

```
scrollPane.setViewportView(list);
```

JOptionPane

Permet d'afficher des popups facilement. Ex :

```
JOptionPane.showConfirmDialog(null, "Etes-vous sur de vouloir détruire cet élément ?",
"Suppression", JOptionPane.YES_NO_OPTION);
```

Affiche une popup de confirmation avec deux boutons, Yes/No.

Les fonctions classiques sont :

- void showMessageDialog(fenêtre appelante,message[,titre]) ; // alerte simple
- int showConfirmDialog(fenêtre appelante,message[,titre],
  options de bouton) ; // confirmation
- String showInputDialog(fenêtre appelante, message[,titre],
  valeur initiale) ; // récupération d'une valeur dans un champ texte

On passe en général null pour *fenêtre appelante.* A noter que ces fonctions existent avec de nombreuses signatures différentes permettant de passer tout ou partie des paramètres possibles.