Modalité de l'examen : Projet

1. Introduction Java
2. **Introduction IHM: Swing(composants, conteneurs, layouts),[réflexion projet]**
3. Les évènements, [Cahier des charges]
4. Boites de dialogues, [Projet]
5. MVC, [Projet]
6. Composants complexes, [Projet]
7. Présentations

Swing

Internet

- https://docs.oracle.com/javase/tutorial/uiswing/TOC.html

- https://docs.oracle.com/javase/tutorial/uiswing/examples/components/index.html#MenuLookDemo

IHM

- Interface homme – machine
  - Dispositif matériel et logiciel lié à une application
- Interaction homme – machine
  - Relations entre l'humain et la machine par l'interface

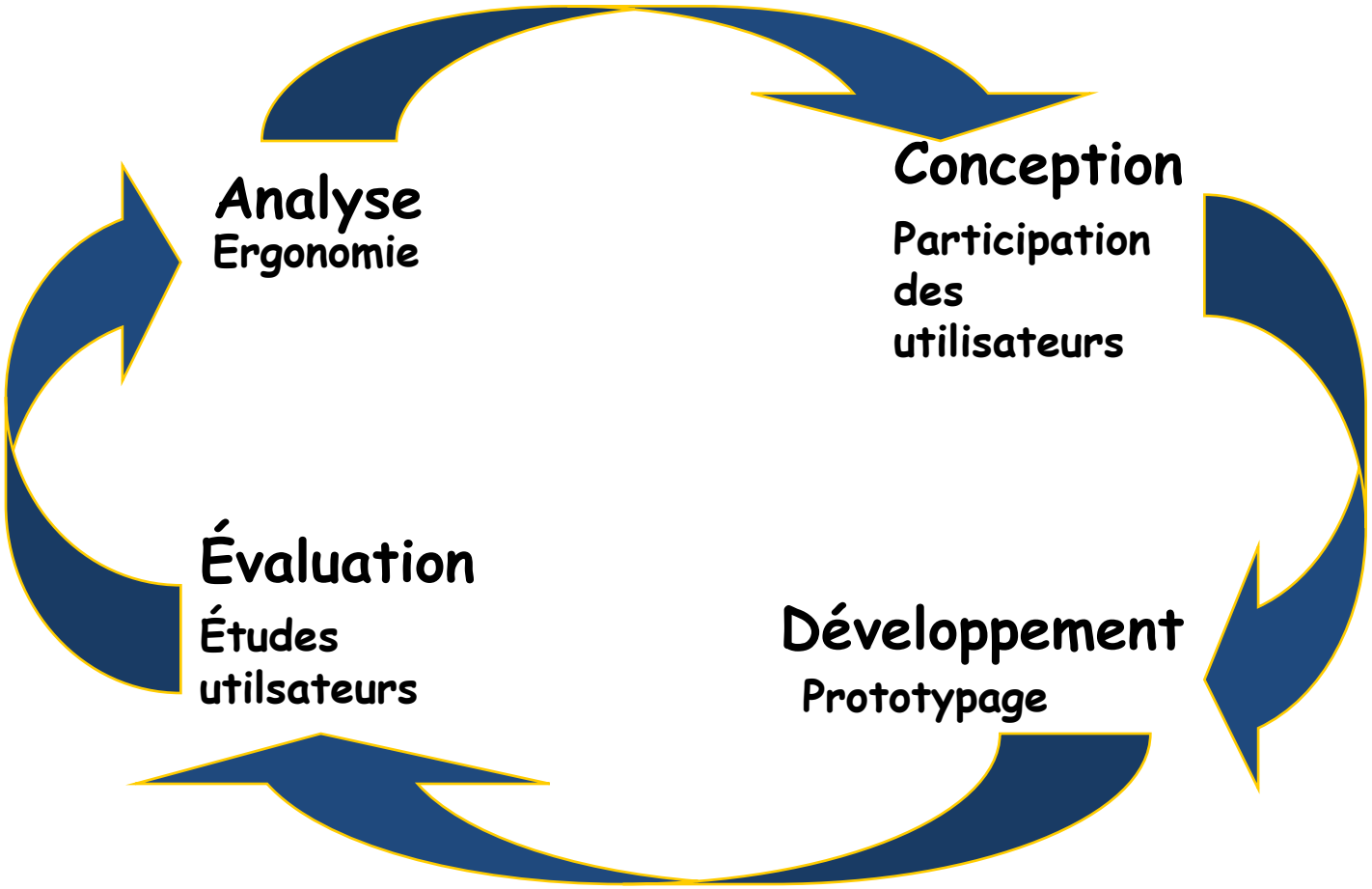
Interface Utilisateur (User Interfaces)

- J. Nanard , J. Landay
  - pour l'usager, c'est le lieu où s'opère la communication avec le système informatique utilisé pour l'aider dans son activité
- J. Caelen
  - l'interface participe autant de l'homme que de la machine puisqu'elle est à leur frontière

Les clés du succès

- Cycle de conception
- Conception centrée usager
- L'analyse des tâches et les analyses en contexte de travail usuel
- Le prototypage rapide
- L'évaluation constante
- Conception itérative
- La qualité de la programmation

Cycle de conception



Conception centrée usager

Connais tes usagers !

- Classes d'utilisateurs
  - Créer des personnages
- Leurs compétences
  - Visuelles, tactiles, manipulatrices
  - Métiers
  - Connaissances du domaine
- Leur contexte
- Maintenir des usagers dans l'équipe de conception tout au long du projet

Comment ?

- Analyse de tâches, analyses en contexte
  - Observer les pratiques usuelles de travail
    - Routinières
    - Exceptionnelles
    - Situations de stress
  - Créer des scénarios d'utilisation réelle
  - Tester les idées nouvelles avant de développer un logiciel

Évaluation

- Tester avec des usagers réels
- Construire des modèles
- Pour les modèles et les maquettes, utiliser des techniques d'évaluation « légères »
  - Évaluation par des experts
  - Inspections
  - Tests informels
- Pour les prototypes avancés, utiliser des techniques d'évaluation plus « lourdes » et rigoureuses
  - Expérimentations contrôlées
  - Expérimentations en contexte

Event-Driven

- qu'est ce que ce type d'application a de particulier ?
  - son architecture
    - c'est l'utilisateur qui décide...
      - ...ou plutôt ce sont les évènements qui décident
- event-driven programming, event-driven design

Event-Driven

- Event-driven programming ?

  – une boucle principale :

    - (event detection) retirer un évènement
    - (event handling) distribuer l'évènement à la partie logicielle concernée

Guide de conception

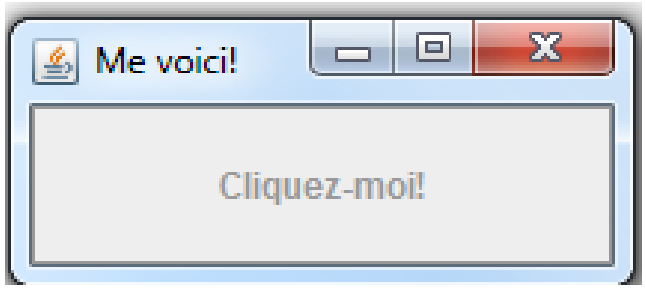
- suivre les règles permet d'obtenir une application qui se comporte comme attendu

- les API permettant en grande partie de garantir la conformité vis-à-vis de ces règles

- il n'y a donc qu'à se focaliser sur les aspects abstraits ou logiques

Guide de conception

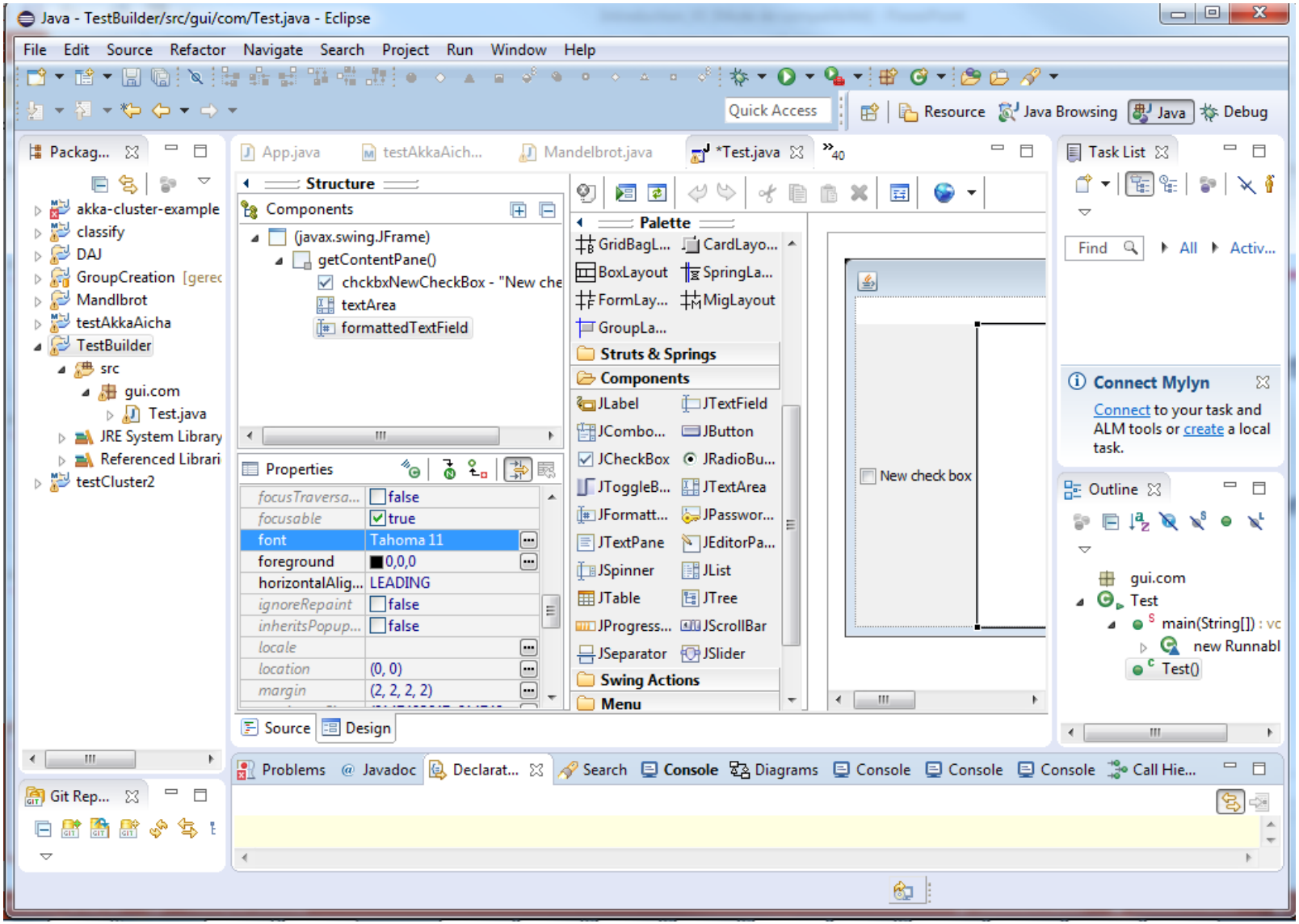
- Un programme utilisant une interface graphique compliquée n'est pas nécessairement complexe
- les API offrent en général des objets tout faits permettant de rendre la plupart des services généraux attendus...
- afficher un message dans une fenêtre, par exemple...



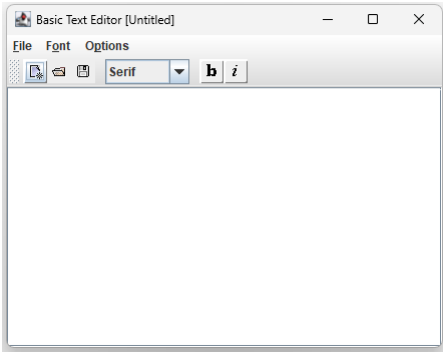
Objets d'un GUI

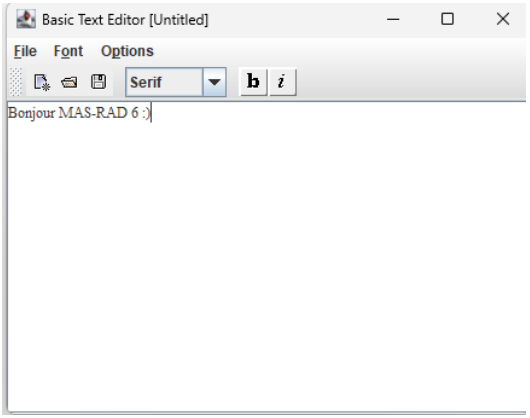
- Que sont/font tous ces objets ?
- Comment interfèrent-ils les uns avec les autres ?
  - C'est le sujet central de ce cours
    - Comprendre la structure d'une API d'une interface graphique
    - nous étudierons celle de Java : Swing

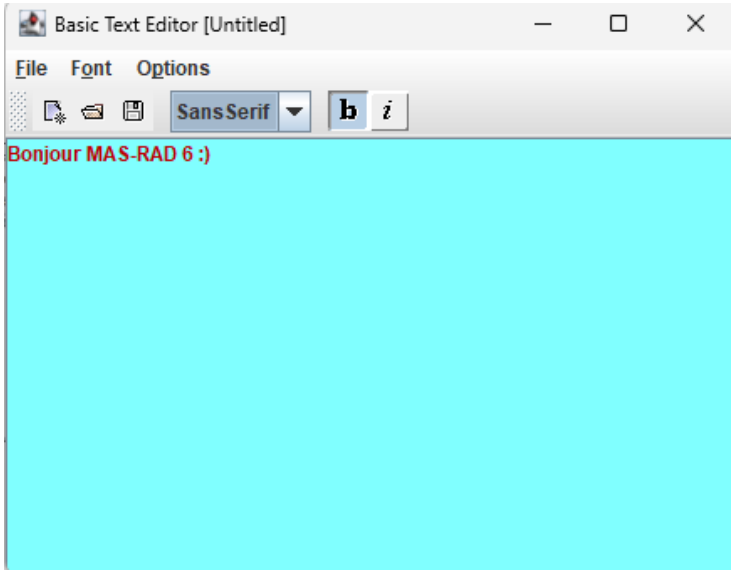
Exemple_01


Java - TestBuilder/src/gui/com/Test.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Resource
Java Browsing
Java
Debug
Packag...
akka-cluster-example
classify
DAJ
GroupCreation
Mandlbrot
testAkkaAicha
TestBuilder
src
gui.com
Test.java
JRE System Library
Referenced Librari
testCluster2
App.java
testAkkaAich...
Mandelbrot.java
*Test.java
Structure
Components
(javax.swing.JFrame)
getContentPane()
chckbxNewCheckBox - "New che
textArea
formattedTextField
Palette
GridBagL...
CardLayo...
BoxLayout
SpringLa...
FormLay...
MigLayout
GroupLa...
Struts & Springs
Components
JLabel
JTextField
JCombo...
JButton
JCheckBox
JRadioBu...
JToggleB...
JTextArea
JFormatt...
JPasswor...
JTextPane
JEditorPa...
JSpinner
JList
JTable
JTree
JProgress...
JScrollBar
JSeparator
JSlider
Swing Actions
Menu
New check box
Properties
focusTraversa... false
focusable true
font Tahoma 11
foreground 0,0,0
horizontalAlig... LEADING
ignoreRepaint false
inheritsPopup... false
locale
location (0, 0)
margin (2, 2, 2, 2)
Source
Design
Task List
Find
All
Activ...
Connect Mylyn
Connect to your task and ALM tools or create a local task.
Outline
gui.com
Test
main(String[]) : vc
new Runnabl
Test()
Problems
Javadoc
Declarat...
Search
Console
Diagrams
Console
Console
Console
Call Hie...
Git Rep...

Exemple_02


Basic Text Editor [Untitled]
File Font Options
Serif
b i


Basic Text Editor [Untitled]
File Font Options
Serif
b i
Bonjour MAS-RAD 6 :)


Basic Text Editor [Untitled]
File Font Options
SansSerif
b i
Bonjour MAS-RAD 6 :)

Swing(Evolution historique)

- Basé sur AWT : Swing est une bibliothèque d'interfaces graphiques qui s'appuie sur AWT mais offre une plus grande flexibilité et un meilleur contrôle sur l'apparence des composants. Swing est basé sur l'architecture de composants légers (contrairement aux composants lourds d'AWT), ce qui signifie qu'il dessine les composants sur la fenêtre indépendamment du système d'exploitation.

- Avantages :

  - Look-and-feel indépendant : Avec Swing, l'apparence des composants est plus uniforme, peu importe le système d'exploitation.
  - Flexibilité : On peut personnaliser les composants à l'aide de renders et d'autres outils.
  - Plus riche : Swing propose une gamme plus large de composants, tels que des tables, des arbres, des boîtes de dialogue avancées, etc.
  - Modularité : Swing offre un meilleur modèle de gestion des événements avec son architecture basée sur les listeners.

Swing

- SWING est en sus de AWT, mais en réutilise quelques parties. Il est inclu dans le SDK (software Development Kit) de JAVA depuis la version 1.2.
- SWING est contenu dans le package :**javax.swing.**

```
1   import javax.swing.*;  // Importation des classes Swing
2   import java.awt.*;     // Importation des classes AWT
3   import java.awt.event.*;  // Importation des événements AWT
```

SWING est indépendant du système d'exploitation (Mac, Windows, Linux, …)

→on garde donc une portabilité des programmes.

Swing

- Les applications SWING se développent de la même manière que toutes les applications JAVA en utilisant les classes de la bibliothèque SWING:

  objets, classes, héritages

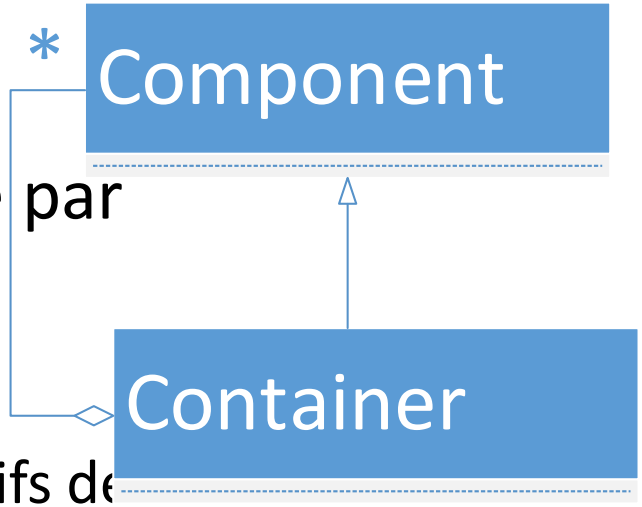
- Programmer une interface n'est donc pas différent que programmer un ensemble de classe

Architecture Swing

- Une application est composée de plusieurs Swing :
  - Un composant **top-level**
  - Plusieurs composants **conteneur intermédiaire**, ils contiennent d'autre composants
  - Des **composants atomiques**

Composants/Containers

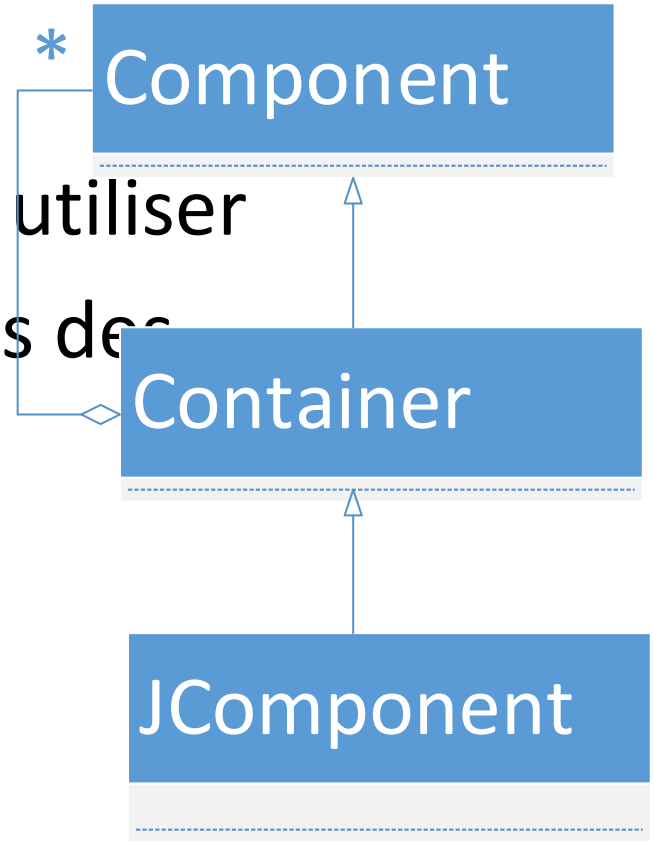
- Interface
  - Une interface est obtenue par agrégation de composants



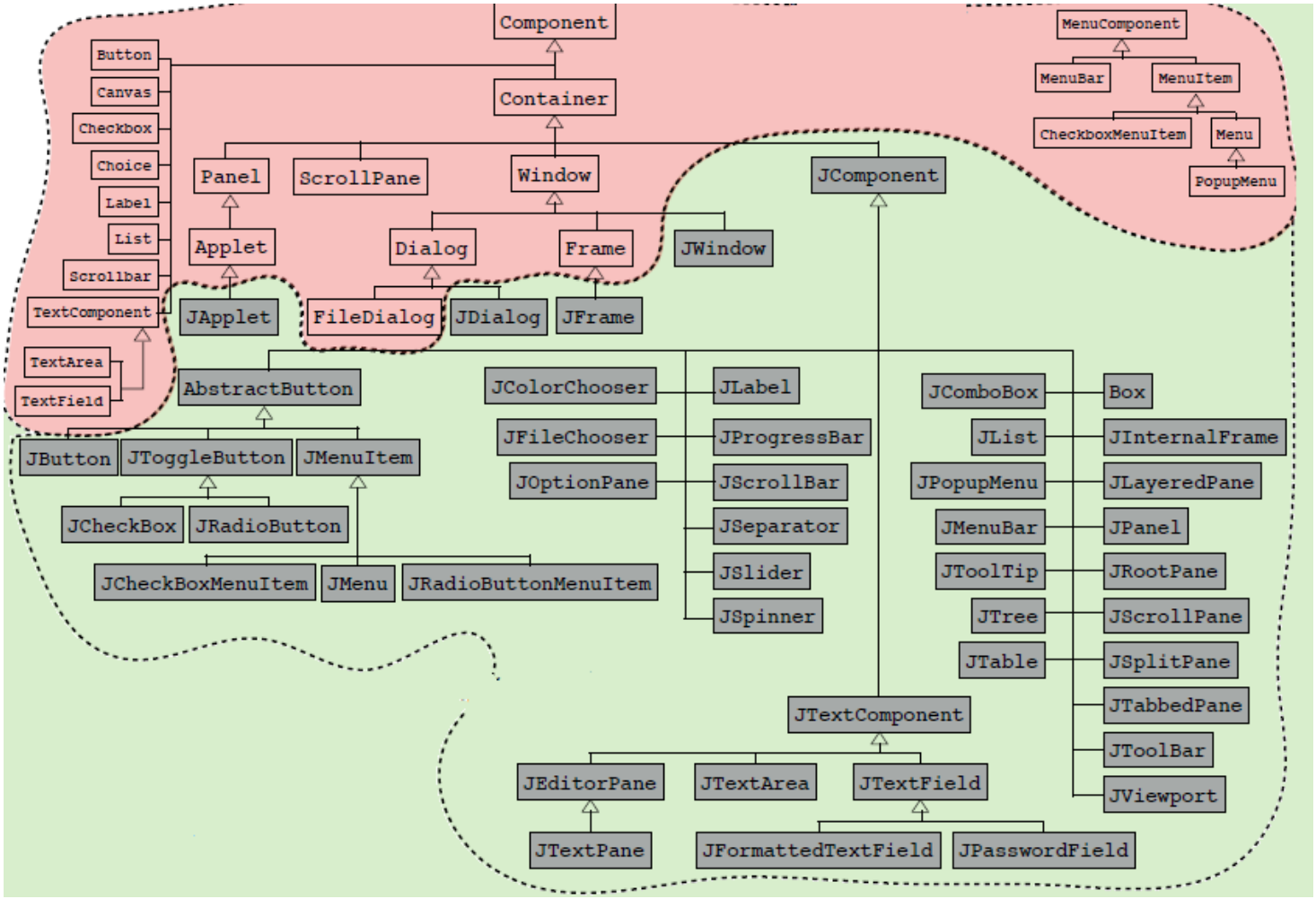
    - des emboîtements successifs de
      - boîtes (containers)
      - objets (composants simples)

Composants/Containers

- Les JComponents Swing sont des Container AWT…
- il n'est pas conseillé de les utiliser comme tels… sauf dans le cas des *containers* Swing…



Composants/Containers



Component

- Les composants (awt.Component) :
  - un nom (name)
  - une taille et position (x/y/width/height - size/location)
  - visible ou non (visible)
  - réceptacle de divers événements
- Il est **très important de rendre visible** les composants sous peine d'invisibilité…

Container

- Les containers (awt.Container) :
  - des méthodes de gestion de la relation d'agrégation Container/Composant
    - add/remove/getComponent/getComponentCount/ getComponentAt/getComponents…
- des méthodes de gestion de la disposition (layout)
- une police par défaut (font)

JComponent

- Les composants Swing (swing.JComponent) :
  - support pour une apparence dynamique (pluggable look-and-feel)
  - amélioration de la gestion du clavier
  - support pour info-bulles
  - support pour l'accessibilité
  - support pour stockage de propriétés spécifiques
  - support amélioré pour le dessin (double-buffering,bords)

JComponent

- Quelques propriétés des JComponents :
  - opaque (boolean)
  - background/foreground (Color)
  - font (Font)
  - toolTip (String)
- illustration avec un JLabel (JComponentExemple.java)
  - précaution : rendre le JLabel opaque car par défaut son fond est transparent…

Containers

- Les Containers
  - des boîtes (2D)
  - ont pour rôle de contenir d'autres composants

Containers racine

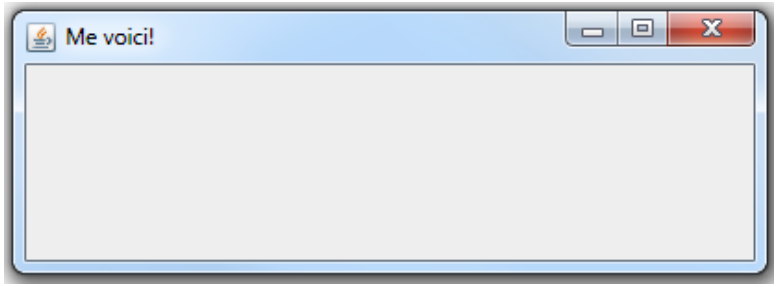
- Les containers racine (JDialog, JFrame, JWindow) :
  - Sont les points d'entrée de toute interface graphique Swing
  - Peuvent contenir d'autres composants Swing
  - Sont gérés directement par le système de fenêtres (window manager)
- Caractéristiques :
- Chaque application Swing utilise au moins un container racine
- Ce sont les seules fenêtres "top-level" visibles indépendamment

Containers racine

- JFrame
  - Fenêtre principale avec décoration (barre de titre, icônes, bouton fermer...)
  - Utilisée pour les applications autonomes
  - Peut contenir une JMenuBar
- Jwindow
  - Fenêtre sans décoration (pas de barre de titre)
  - Idéal pour des splash screens ou des info-bulles personnalisées
  - Toujours en relation avec une autre fenêtre (owner)
- Jdialog
  - Fenêtre secondaire liée à une autre (modale ou non)
  - Sert pour les boîtes de dialogue : confirmation, saisie, paramètres...
  - Comportement configurable (modalité, parent...)

Containers racine

- JFrame(String title)
  - contient un unique JRootPane
    - peut être remplacé
    - ne peut être enlevé
- peut être associée à une JMenuBar



Containers racine

- JWindow() / JWindow(Frame owner) / JWindow(Window owner)!
  - contient un unique JRootPane
    - peut être remplacé
    - ne peut être absent
  - pas de barre de menu…

Containers racine

- Rappel : les containers racines peuvent être visibles ou non :
  - setVisible(boolean)
- il **ne faut pas oublier** des les rendre visibles sous peine d'invisibilité…
- il **ne faut pas** les rendre visible trop tôt! Pour éviter des effets désagréables de construction visible de l'interface et de performances…

Containers racine

- JWindow, JFrame
- des capsules pour un container *utilisateur*
- le container principal (JRootPane) est accessible *via*
  - Container getContentPane()
  - setContentPane(Container)

Containers

- Les containers ordinaires
- JPanel
- JScrollPane
- JSplitPane
- JTappbedPane
- JToolBar
- Permettent la division d'un espace existant
  - Ne sont utilisables que dans d'autres containers

Containers

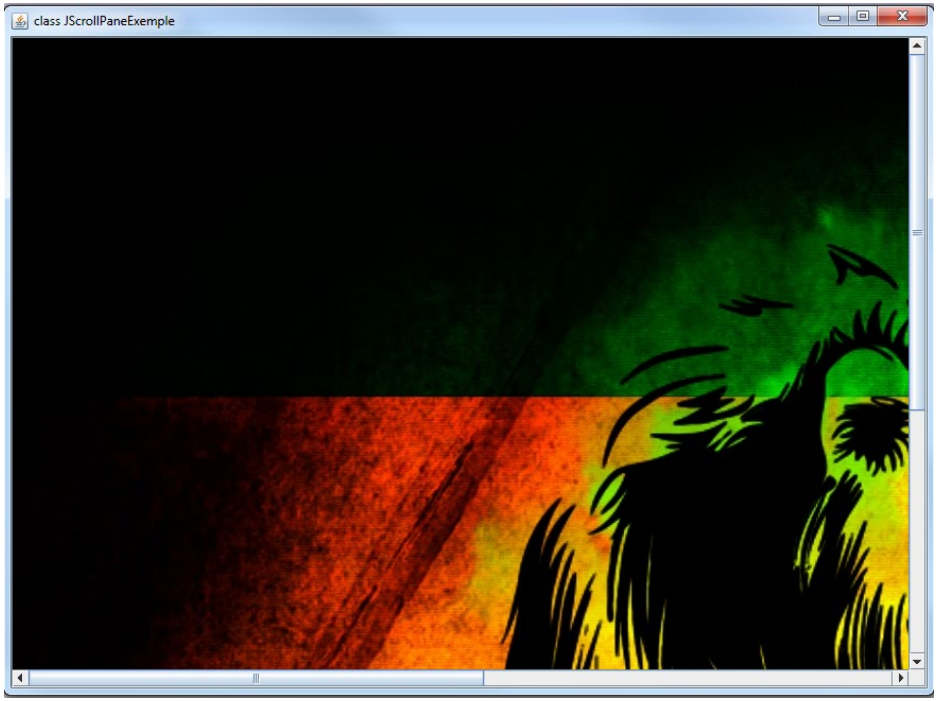
- Les containers ordinaires
  - un espace de rangement d'autres composants, l'agrégation vit grâce à :
    - add(Component) / remove(Component)
  - comment les composants sont-ils rangés/placés ?
- comme on veut… on le verra plus tard…

JPanel

- JPanel
  - le plus simple, un espace rectangulaire
  - un simple panneau d'affichage

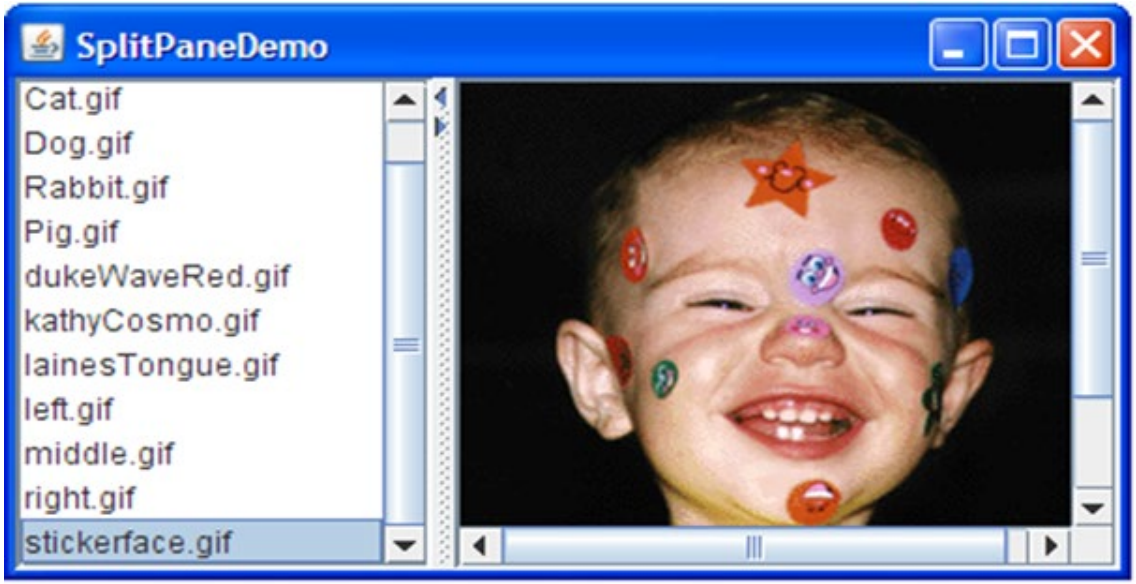
JScrollpane

- JScrollPane
  - une fenêtre sur un espace rectangulaire déplaçable



JSplitPane

- divise un espace verticalement ou horizontalement en deux parties dont la somme est l'espace entier

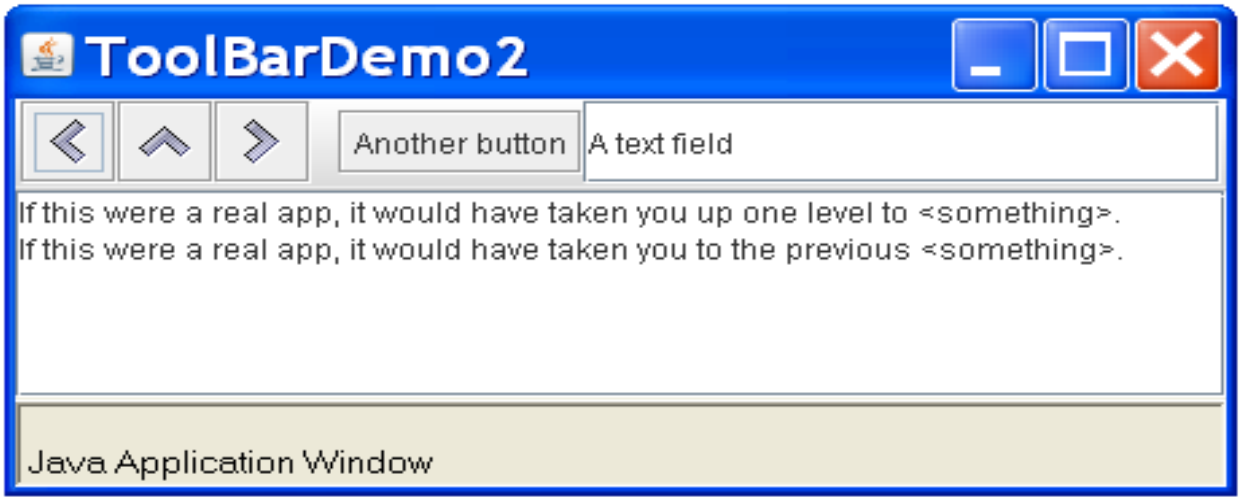


JTabbedPane

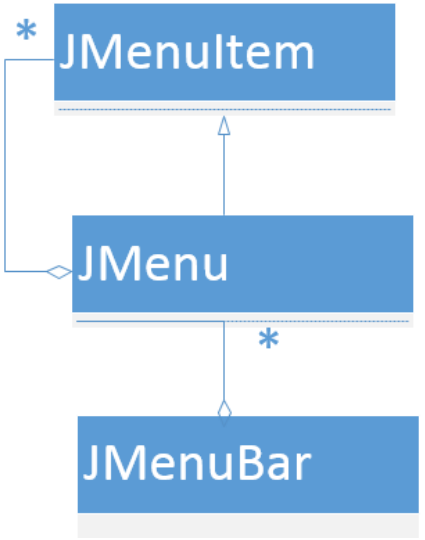
- Un JTabbedPane permet d'avoir des onglets

JToolBar

- Une JToolBar est une barre de menu



Menu



- Les menus
  - JMenuItem
    - Un choix dans un menu
  - JMenu
    - Un container dédié qui hérite de JMenuItem
  - JMenuBar
    - Un container dédié à l'accueil de menu

Les menus

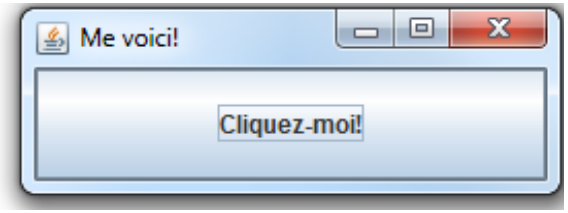
- Si on a une barre de menu JMenuBar, on ajoute des JMenu dans la barre.Les JMenu et le JPopupMenu ont le même mode de fonctionnement, créer des JMenuItem et les ajouter.

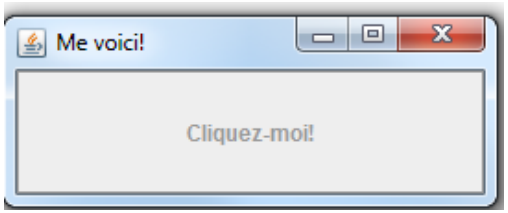
- Ex:


```
menuBar = new JMenuBar();
setJMenuBar(menuBar);
menu = new JMenu("A Menu");
menuBar.add(menu);
menuItem = new JMenuItem("menu item");
menu.add(menuItem);
```

Composants

- Les composants :
- peuvent être actifs ou non, *i.e.* autorisent l'interactivité
  - setEnabled(boolean)
  - boolean getEnabled()
- l'effet obtenu est en général un grisé





Composants

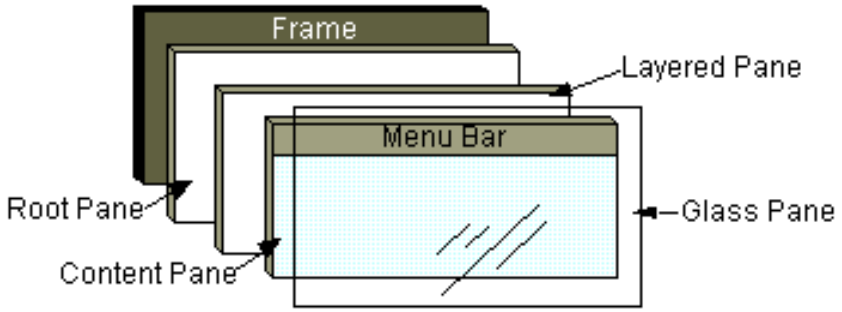
- Si le container est visible
  - validate() permet d'obtenir un replacement correct de tous les composants après ajout de nouveaux composants
- Container racine
  - pack() permet d'obtenir un rangement *optimal*

Conteneur Intermédiaire

- Les conteneur Intermédiaire spécialisé sont des conteneurs qui offrent des propriétés particulières aux composants qu'ils accueillent
  - JRootPane
  - JLayeredPane:
  - JInternalFrame
  - JDesktopPane

JRootPane

- En principe, un JRootPane est obtenu à partir d'un top-level ou d'une JInternalFrame



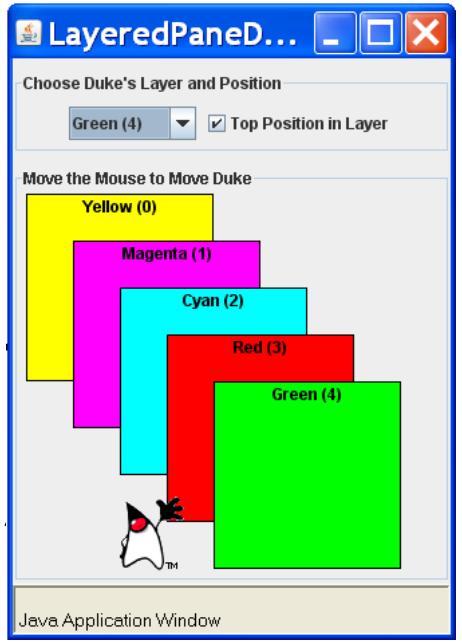
- Un JRootPane es
  - glass pane
  - layered pane
  - content pane
  - menu bar

JLayeredPane

- Un JLayeredPane permet de positionner les composants dans un espace à trois dimensions

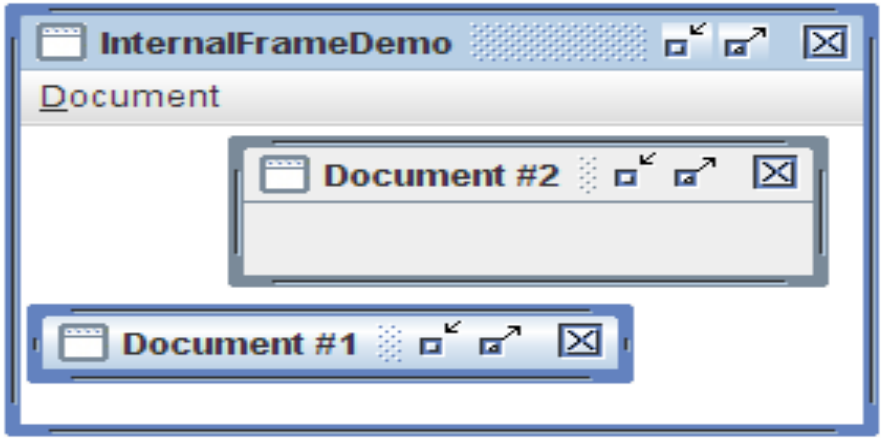
- Pour ajouter un Composant:

  add(Component c, Integer i)



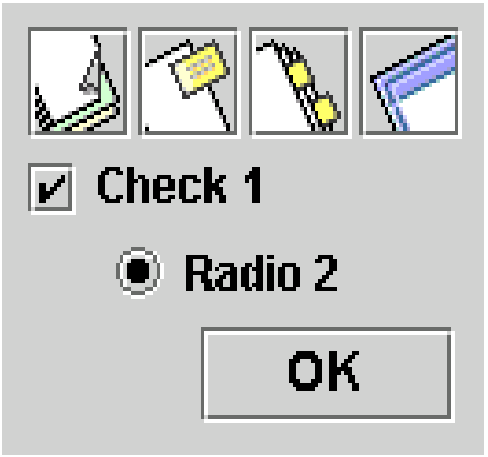
JInternaleFrame

- Un JInternaleFrame permet d'avoir des petites fenêtres dans une fenêtre.
- Une JInternaleFrame ressemble très fortement à une JFrame mais ce n'est pas un container Top-Level



Composants atomiques

- Un composant atomique est considéré comme étant une entité unique.
- Java propose beaucoup de composants atomiques:
  - boutons, CheckBox, Radio
  - Combo box
  - List, menu
  - TextField, TextArea, Label
  - FileChooser, ColorChooser,
  - ...

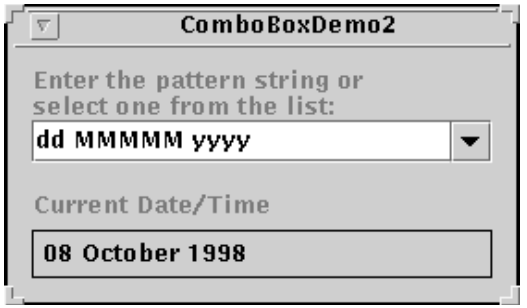


Boutons

- Les boutons (AbstractButton) peuvent être regroupés logiquement *via* des ButtonGroup
  - cela n'a vraiment de sens que pour les boutons qui ont un état de sélection
    - donc ni JButton, ni JMenuItem
    - le plus souvent utilisé avec les RadioButton
- Les ButtonGroup permettent de contrôler l'exclusion mutuelle lors de sélection

JComboBox

- Un JComboBox est un composant permettant de sélectionner un élément parmi plusieurs propositions.

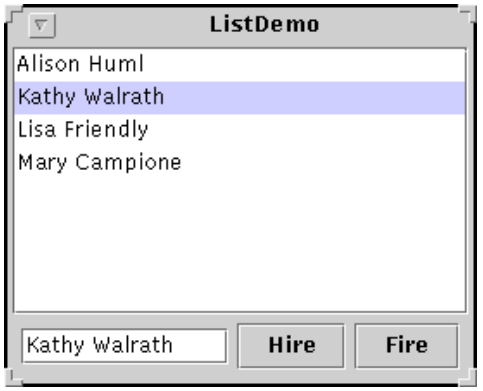


- Quelques méthodes:

```
public JComboBox(Vector v);
public JComboBox(ComboBoxModel c);
Object getSelectedItem();
void addItem(Object o);
```

JList

- Une JList propose plusieurs éléments rangés en colonne.Une JList peut proposer une sélection simple ou multiple Les JList sont souvent contenues dans un srolledPane



- Quelques méthodes:

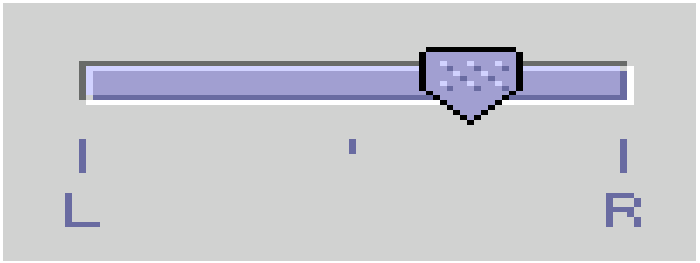
public JList(Vector v);

public JList(ListModel l);

Object[] getSelectedValues();

JSlider

- Les JSlider permettent la saisie graphique d'un nombre Un JSlider doit contenir les bornes max et min



- Quelques méthodes:

public JSlider(int min ,int max, int value);

public void setLabelTable(Dictionary d);

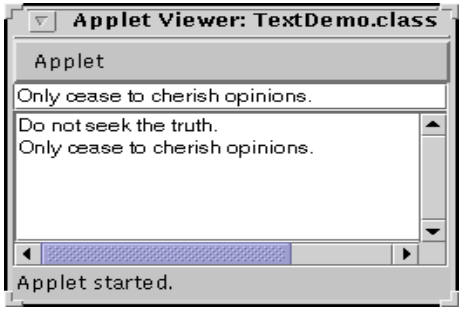
JTextField

- Un JTextField est un composant qui permet d'écrire du texte.Un JTextField a une seule ligne contrairement au JTextArea Le JPasswordField permet de cacher ce qui est écrit

- Quelques méthodes:

public JTextField(String s);

public String getText();

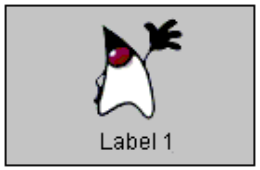


JLabel

- Un JLabel permet d'afficher du texte ou une image.
- Un JLabel peut contenir plusieurs lignes et il comprend les tag HTML.

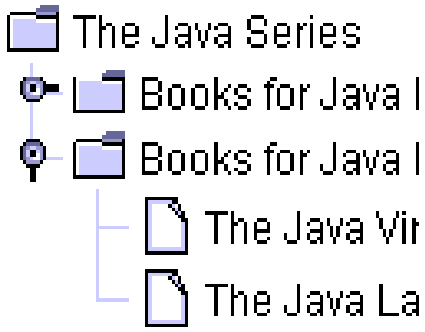
- Quelques méthodes:

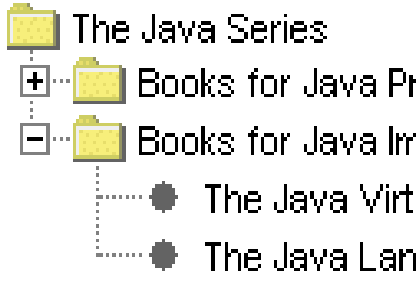
```
public JLabel(String s);
public JLabel(Icon i);
```

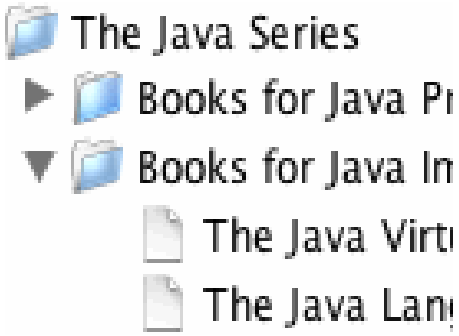


JTree

- Un JTree permet d'afficher des informations sous forme d'arbre. Les nœuds de l'arbre sont des objets qui doivent implanter l'interface MutableTreeNode. Une classe de base est proposée pour les nœuds : DefaultMutableTreeNode.Pour construire un arbre il est conseillé de passer par la classe TreeModel qui est la représentation abstraite de l'arbre.





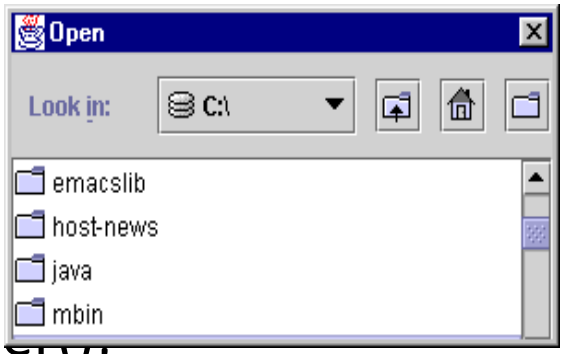


- Pour construire un arbre:

```
rootNode = new DefaultMutableTreeNode("Root");
treeModel = new DefaultTreeModel(rootNode);
tree = new JTree(treeModel);
childNode = new DefaultMutableTreeNode ("Child");
rootNode.add(childNode);
```

JFileChooser

- Un JFileChooser permet de sélectionner un fichier en parcourant l'arborescence du système de fichier.



- Ex :
- JFileChooser fc = new JFileChooser();

```
int returnVal = fc.showOpenDialog(aFrame);
if (returnVal == JFileChooser.APPROVE_OPTION) {    File file =
    fc.getSelectedFile();
```

JColorChooser

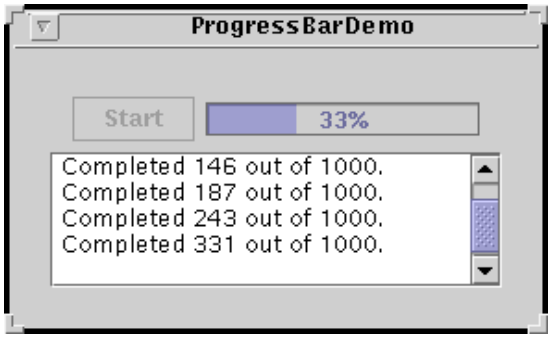
- Un JColoChooser permet de choisir une couleur

- Une méthode :

public static Color showDialog(Component c , String title , Color initialColor);

JProgressBar

- Un JProgressBar permet d'afficher une barre de progression.



- Quelques méthodes :

```
public JProgressBar();
public JProgressBar(int min, int max);
public void setValue(int i);
```

Positionnement des composants

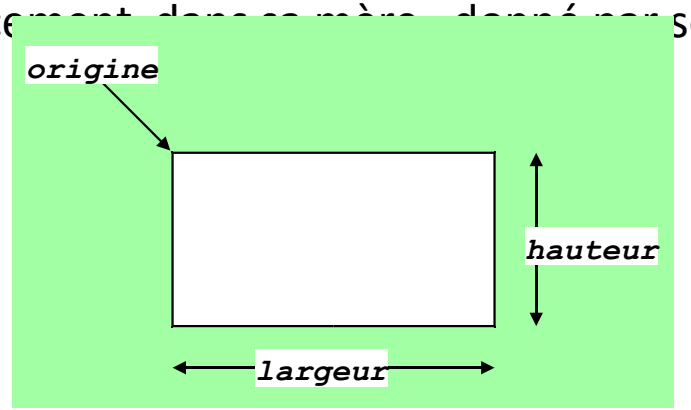
- Layouts de base
- AWT
  - BorderLayout
  - CardLayout
  - GridLayout!
  - GridBagLayout
  - FlowLayout

➢ Swing

- BoxLayout
- GroupLayout
- OverlayLayout
- ScrollPaneLayout
- SpringLayout
- ViewportLayout

La géométrie

- Tout composant a un placement dans sa mère donné par son origine et ses dimensions



- Le rectangle est géré par **Rectangle getBounds()** resp. **setBounds(...)**
- La position (origine) est gérée par **Point getLocation()** resp. **setLocation(..)**
- La taille (une **Dimension)** gérée par **getSize()** resp. **setSize(...)**
- Les tailles "idéale", maximale et minimale sont retournées par **getPreferredSize() getMaximalSize() getMinimalSize()**
- Les *gestionnaires de géométrie* utilisent ces informations pour placer les filles dans un conteneur.

Architecture de layout

- Pour placer des composants dans un container, Java propose une technique de Layout.
- Un layout est une entité Java qui place les composants les uns par rapport aux autres.
- Le layout s'occupe aussi de réorganiser les composants lorsque la taille du container varie.
- Un layout n'est pas un contenu dans un container, il gère le positionnement.

Layouts

- on peut toujours essayer de ranger les éléments soi-même mais c'est généralement non-portable…
- on peut choisir la politique de placement associée à un container donné
  - méthode setLayout(LayoutManager)

FlowLayout

- Un FlowLayout permet de ranger les composants dans une ligne. Si l'espace est trop petit, une autre ligne est créée.
- Le FlowLayout est le layout par défaut des JPanel

- Ex :
- Container contentPane = getContentPane();
- contentPane.setLayout(new FlowLayout());

- contentPane.add(new JButton("Button 1"));
- contentPane.add(new JButton("2"));
- contentPane.add(new JButton("Button 3"));
- contentPane.add(new JButton("Long-Named Button 4"));
- contentPane.add(new JButton("Button 5"));

FlowLayout

- après retaille, la disposition est recalculée.
- **FlowLayout(int align, int hgap, int vgap)**
- **FlowLayout(int align)**
- **FlowLayout()**
- **align** vaut **LEFT, CENTER** (défaut) ou **RIGHT** , LEADING , TRAILING (par rapport à l'orientation) et indique comment chaque ligne est remplie.
- **hgap** (= 5) et **vgap** (=5) sont les espaces entre composants.
- les marges sont régies par le membre **insets** du conteneur.
- Les paramètres s'obtiennent et se modifient par des méthodes **set** et **get** .
- Le changement ne prend effet
  - qu'après retaille,
  - ou en forçant par **layoutContainer().**
  - ou en demandant une validation par **revalidate()** .

BorderLayout

- Le BorderLayout sépare un container en cinq zones: NORTH, SOUTH, EAST, WEST et CENTER
- Lorsque l'on agrandit le container, le centre s'agrandit. Les autres zone prennent uniquement l'espace qui leur est nécessaire.

- Ex :

```
Container contentPane = getContentPane();
 contentPane.setLayout(new BorderLayout());
contentPane.add(new JButton("Button 1 (NORTH)"),
 BorderLayout.NORTH);
```

BorderLayout

- **pack()** : les conteneurs prennent la taille *minimale* pour contenir leur composants, en fonction de la taille *préférée* des composants.
- Pour suggérer ou imposer une autre taille, on retaille par
  - **setSize(largeur, hauteur)**
  - **setPreferredSize(dimension)**
- Ou encore, on réécrit la méthode **getPreferredSize(),** par exemple

```
public Dimension getPreferredSize() { return new
    Dimension(100,150);
}
```

- Nécessaire pour un **Canvas**, dont la taille est *nulle* par défaut.

BorderLayout

- Les marges sont entre les bords et les filles.
- L'ensemble des marges est de la classe **Insets** :
  - quatre champs
  - constructeur

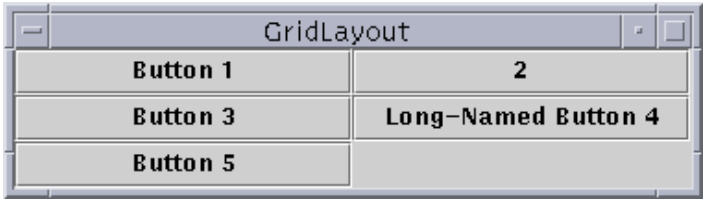
    Insets(top, left, bottom, right)

- Les marges comprennent le bord, et pour **Frame**, aussi la barre de titre !
- Pour changer de marges, on redéfinit la méthode

```
public Insets getInsets () {
        return new Insets(10, 10, 10, 10);
}
```

GridLayout

- Un GridLayout permet de positionner les composants sur une grill



- Ex:

```
Container contentPane = getContentPane();
contentPane.setLayout(new GridLayout(0,2));
contentPane.add(new JButton("Button 1"));
contentPane.add(new JButton("2"));
contentPane.add(new JButton("Button 3"));
contentPane.add(new JButton("Long-Named Button 4"));
contentPane.add(new JButton("Button 5"));
```

GridLayout

- **GridLayout(int lignes, int cols, int hgap, int vgap)**
- **GridLayout(int lignes, int cols)**

- composants sur une ***grille*** , ligne par ligne (dans l'or
- les cellules ont la ***même taille*** ,
- à la retaille, les cellules se taillent,
- **hgap** et **vgap** sont nuls par défaut.
- On fixe le nombre de lignes ***ou*** de colonnes,
- ***si*** **lignes>0** , *alors* **cols** est ignoré ,
- ***si*** **lignes = 0** , *alors* **lignes** *est ignoré.*

BoxLayout

- Un boxLayout permet de positionner en une ligne ou une colonne.
- A ne pas confondre avec le GridLayout dans le sens que les sous-composants ne sont pas obligés d'avoir tous la même largeur et la même hauteur.

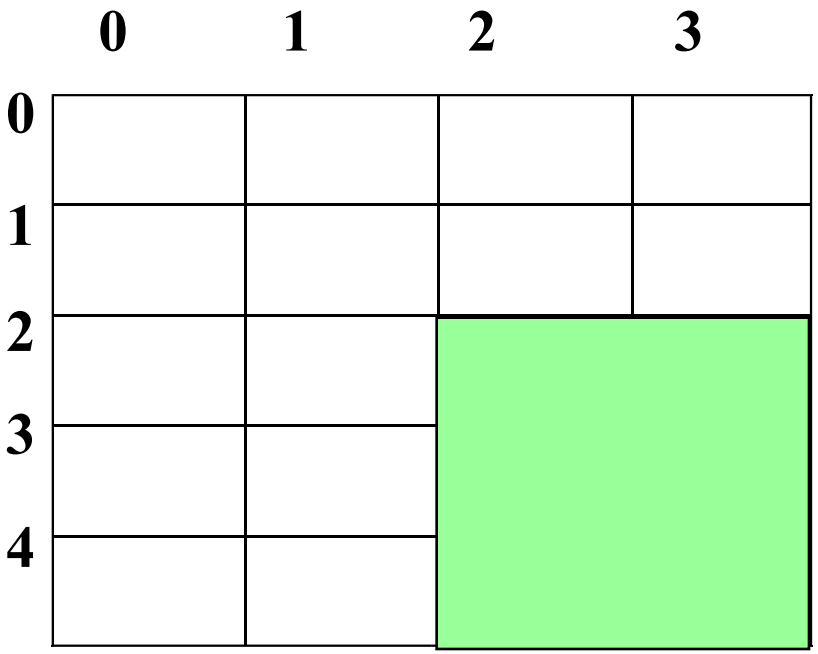
- On peut ajouter
  - des "*struts* ", blocs de taille fixe
  - de la *glue* qui fait du remplissage
  - des *aires* rigides
- Le composant **Box** est un conteneur
  - dont le gestionnaire est **BoxLayout**
  - qui est enfant de **Container** , donc n'est pas dans Swing.

GridBagLayout

- **GridBagLayout** répartit ses filles *dans une grille* , selon les désidératas exprimés par chaque fille.
- Chaque fille s'exprime dans un objet **GridBagConstraints**.
- Les contraintes sont ajoutées avec la fille.
- Les "contraintes" d'une fille concernent
  - la ***zone*** réservée dans la grille
  - la ***place*** occupée dans cette zone

GridBagConstraints

- Pour regrouper l'ensemble des composants dans un GridBagLayout, on utilise le composant java.awt.GridBagConstraints
- Chaque fille réserve une ***zone*** dans la grille par son origine et sa dimension.
- **gridx = 2, gridy = 2, gridwidth = 2, gridheight = 3**
- Défaut : **width = height = 1** .
- Si **gridx = RELATIVE** , l'origine est à droite du précédent.
- Si **gridwith = REMAINDER** , la zone qui reste dans la ligne

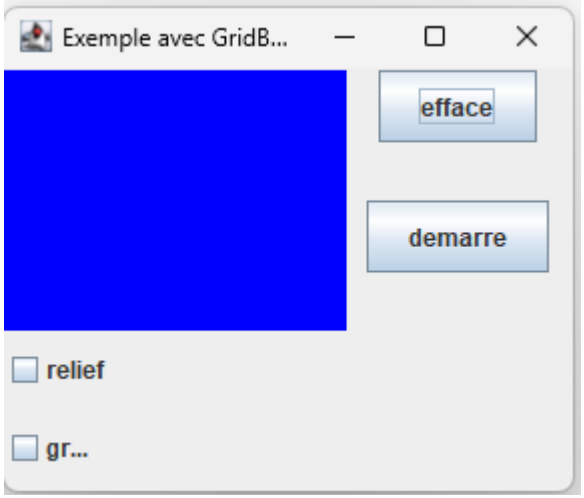


GridBagLayout

- gridx et gridy qui donnent l'abscisse et l'ordonnée du sous-composant dant la grille. Leurs valeurs par défaut sont GridBagConstraints.RELATIVE ; si gridx (resp. gridy) possède cette valeur par défaut, le composant est ajouté juste à la droite (resp. juste en-dessous) du dernier composant ajouté.
- gridwidth et gridheight qui donnent la largeur et la hauteur du sous-composant dans la grille, donc en nombre de lignes et colonnes dans la grille virtuelle. Leurs valeurs par défaut est de 1. Si on donne la valeur GridBagConstraints.REMAINDER à gridwidth (resp. gridheight), le sous-composant devra être le dernier de sa ligne (resp. colonne). Si on donne la valeur GridBagConstraints.RELATIVE, le sous-composant devra occupé tout le reste de sa ligne (resp. colonne).
- fill qui indique dans quelle direction doit éventuellement augmenter le sous-composant (quatre possibilités : GridBagConstraints.NONE (valeur par défaut), GridBagConstraints.BOTH, GridBagConstraints.HORIZONTAL, GridBagConstraints.VERTICAL.
- anchor qui donne la position du sous-composant dans sa cellule de la grille lorsqu'il y a de l'espace autour de lui (neuf constantes CENTER (valeur par défaut), NORTH, NORTHEAST, EAST, SOUTHEAST, SOUTH, SOUTHWEST, WEST, NORTHWEST sont prévues à cet effet dans la classe GridBagConstraints).

GridBagLayout

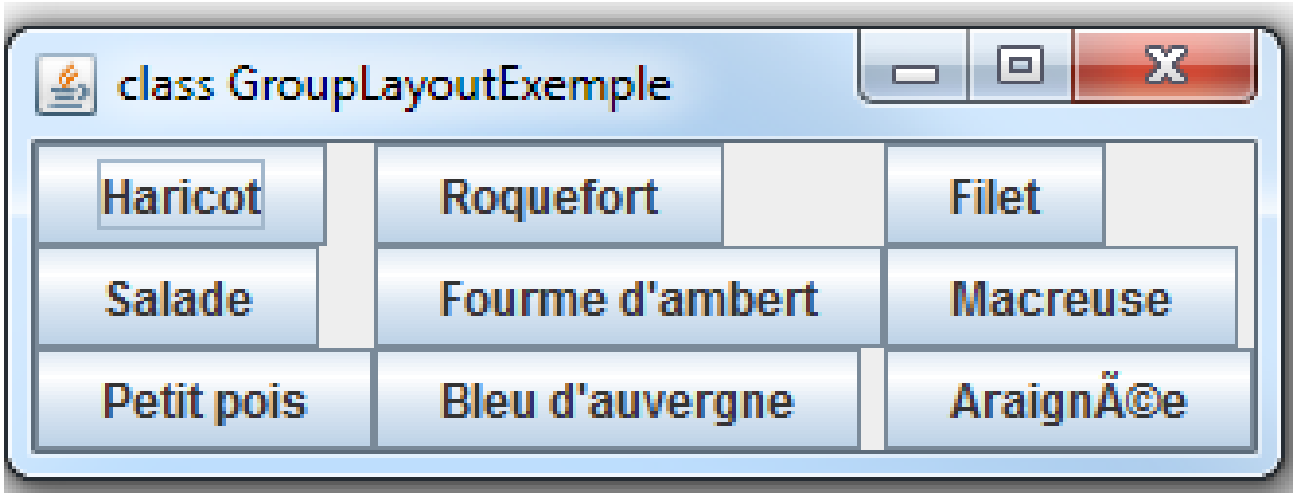
- ipadx et ipady qui indiquent le nombre de pixels dont on doit accroître le composant en plus de sa taille minimum (valeur par défaut 0).



- weightx et weighty sont deux paramètres de type **double** qui indiquent de distribuer l'espace supplémentaire selon les directions horizontales et verticales de façon prioritaire aux sous-composants qui ont les plus grandes valeurs de ces paramètres. Les valeurs données sont souvent 0 ou 1 ; la valeur par défaut est 0.

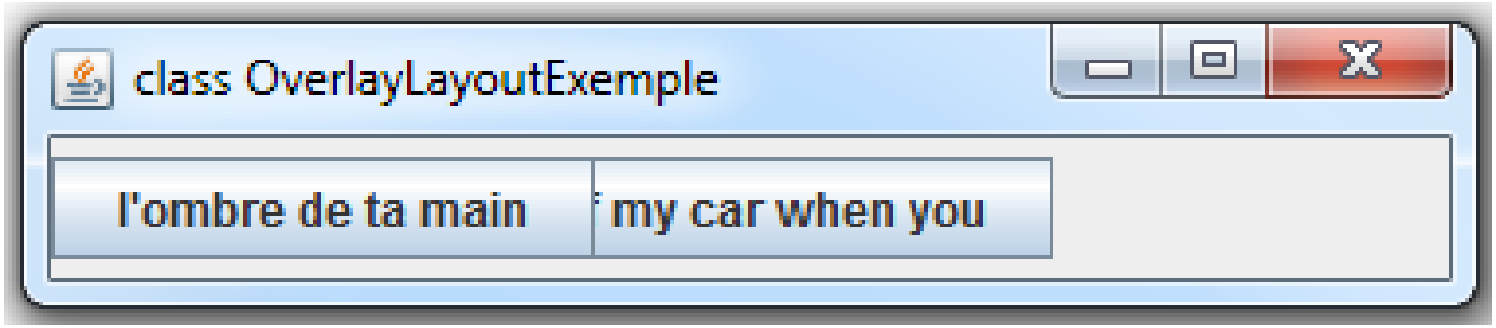
GroupLayout

- permet d'obtenir des effets d'alignement
  - ne retaille pas les composants
  - un poil complexe à utiliser



OverlayLayout

- superpose des composants comme CardLayout
- mais autorise la visualisation/manipulation par transparence…



SpringLayout

- exprime des contraintes entre composants
- simple en apparence……