

Java Swing

Plan

- Introduction
- Architecture du framework Swing
- Composants Swing
- Gestion des layouts
- Gestion des évènements
- Composants et principes avancés
- Déploiement
- Présentation des concepts avancés de Swing



Introduction

Introduction

- Historique de AWT et de Swing
- Présentation des plates-formes RCP java
- JDK
- Eclipse
- Windows Builder



AWT & Swing

Historique

- Swing

- Intégré à Java

- Drag'n'Drop, I18N, Java 2D, accessibilité...

- AWT

- Abstract Window Toolkit

- Avec les composants natifs

- Problèmes de portabilité

- Plus bas niveau que Swing

Architecture

**Window, main container
(JFrame)**

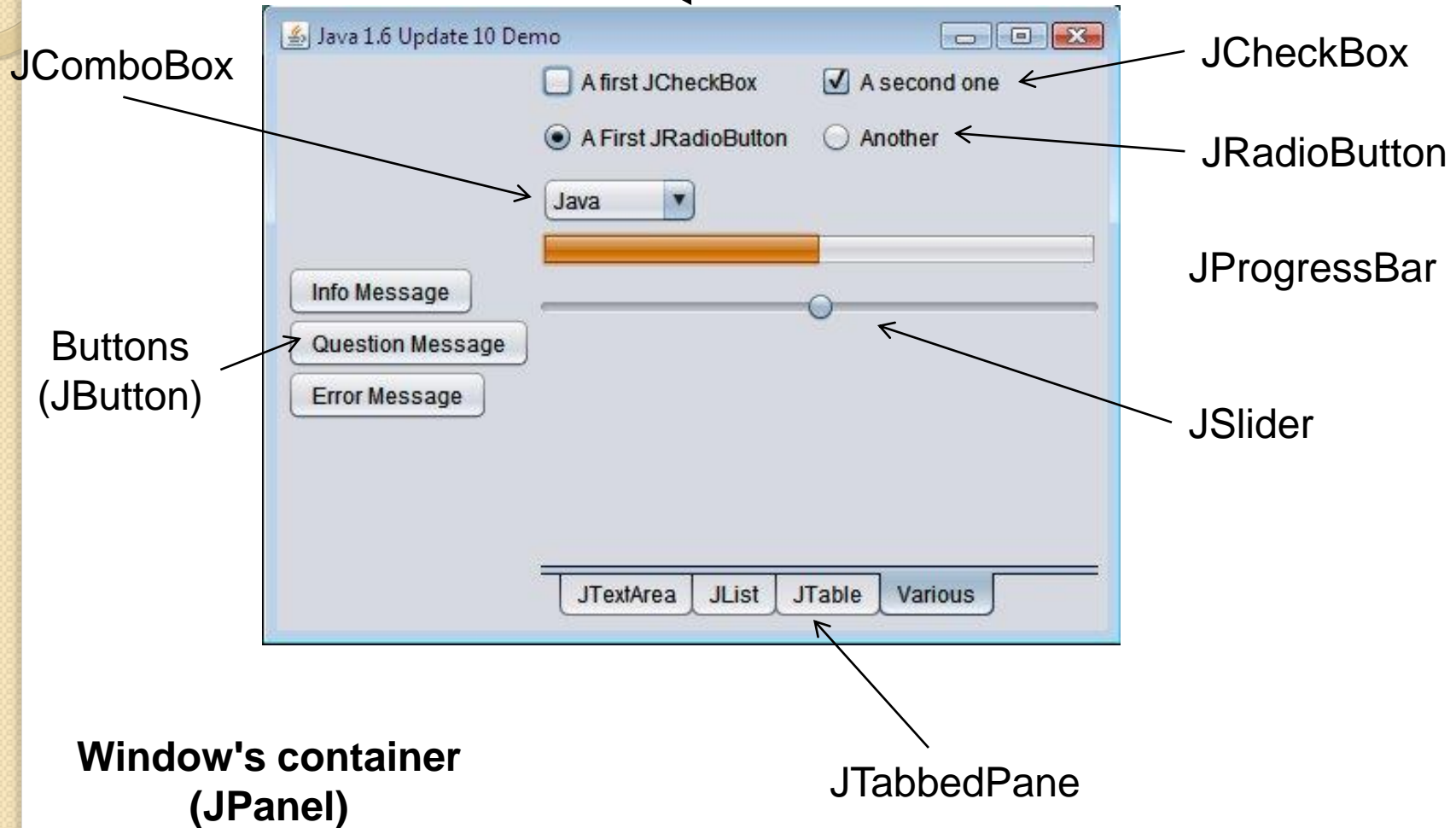




Plate-formes RCP JAVA

RCP Java

- Rich Client Platform
- Jusqu'en 2000 : client lourd/serveur
- A partir de 2000 : client léger (pas d'installation sur le client)
- A partir de 2002 : Rich Internet Application (application généralement lourde mais en client léger)
- Aujourd'hui : client riche
 - Environnement d'exécution comprenant des composants de base : framework.
 - Noyau exécutif générique
 - Interface Utilisateur
 - Eclipse RCP, Netbeans RCP



JDK

JDK installation

- Télécharger et installer la dernière du JDK (1.7)
<http://java.sun.com/javase/downloads/index.jsp>
- Créer une variable d'environnement JAVA_HOME :
 - <http://www.oracle.com/technetwork/java/javase/downloads/index.html>
 - Ajouter la variable d'environnement JAVA_HOME
 - Mettez les Java binaries dans le PATH
 - Les binaires correspondent au répertoire bin/
- Mac Users : déjà installé ! ☺
- Linux : `sudo apt-get install openjdk` et choisir la dernière version disponible



Eclipse

Eclipse installation

- Télécharger <http://www.eclipse.org/downloads/>
- Désarchiver dans un répertoire de travail
- Vous pouvez lancer Eclipse! 😊



The screenshot shows the 'Eclipse Downloads' page. It features a purple header with the title 'Eclipse Downloads'. Below the header, there are tabs for 'Packages' and 'Projects'. Under the 'Packages' tab, there are sub-tabs: 'Compare Packages', 'Older Versions', and 'Eclipse Hellos (3.6.1) Packages'. The main content area lists four download options, each with an icon, the package name, size, download count, and a 'Details' link.

Eclipse Downloads	
Packages	Projects
Eclipse IDE for Java Developers , 98 MB Downloaded 701,573 Times Details	
Eclipse Classic 3.6.1 , 169 MB Downloaded 520,504 Times Details Other Downloads	
Eclipse IDE for Java EE Developers , 204 MB Downloaded 449,253 Times Details	
Eclipse IDE for C/C++ Developers , 87 MB Downloaded 200,157 Times Details	



Windows Builder

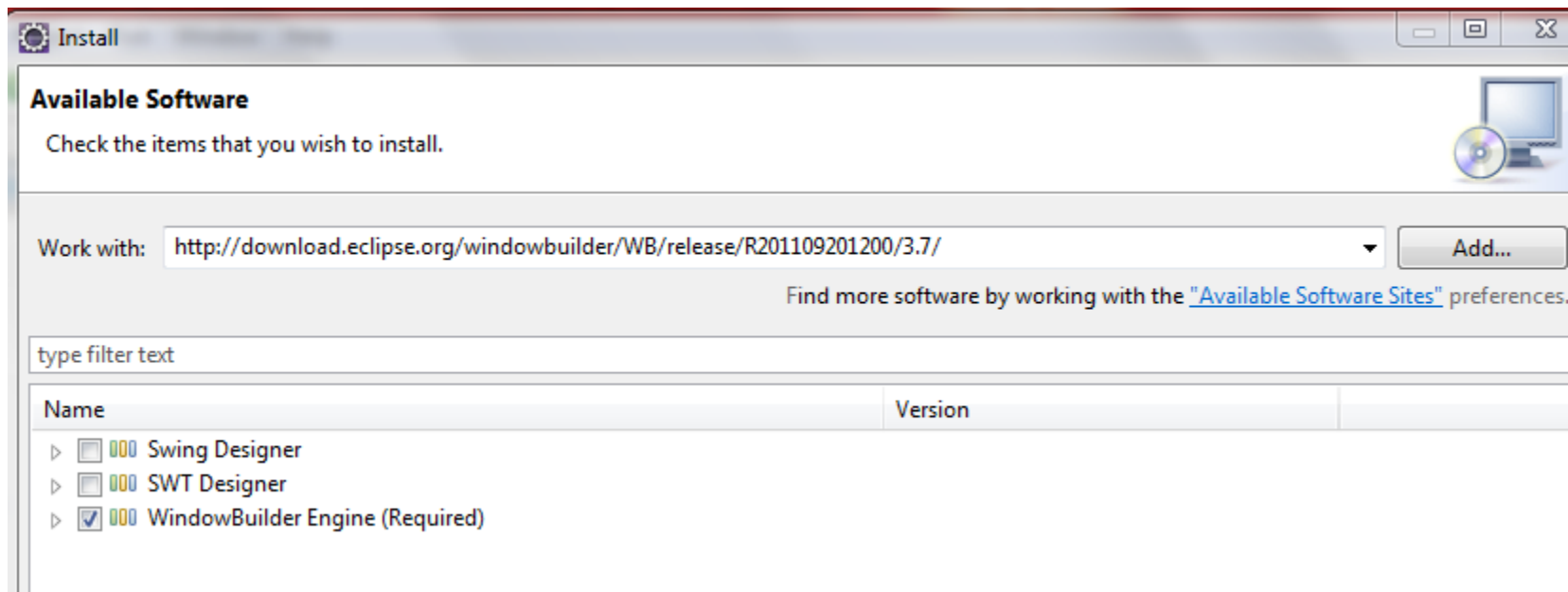
Création facile d'interface graphique

- **W**hat **Y**ou **S**ee **I**s **W**hat **Y**ou **G**et WYSIWY
- Eclipse propose un constructeur d'interface graphique mais ne l'intègre pas à la distribution
- Disponible à l'adresse

<http://www.eclipse.org/windowbuilder/>

Installation

- Help/Install new software
 - <http://download.eclipse.org/windowbuilder/WB/release/R201109201200/3.7/>



■ Des questions?





Architecture du framework Swing

Architecture du framework Swing

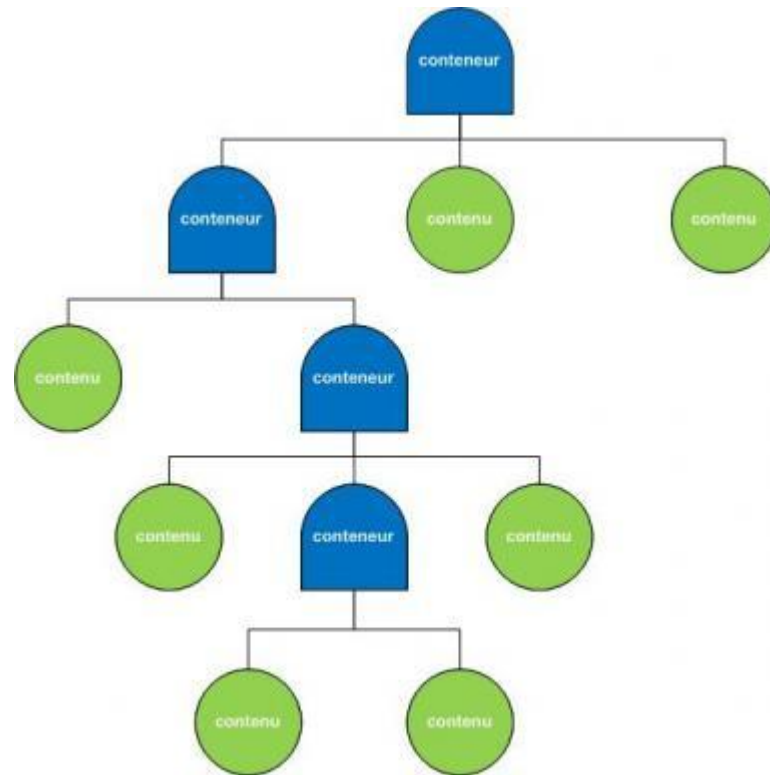
- Arbre des composants
- Catégories de composants
- Variante du modèle MVC
- Fenêtre et conteneurs principaux
- Menus
- Look And Feel



Design Pattern Composite

Design Pattern Composite

- Bonne pratique.
- Structurer les données en hiérarchie selon des relations de compositions.



■ Des questions?





Catégories des composants

Catégories des composants

- Conteneurs

- Par exemple les fenêtres, assure la liaison avec le système de fenêtrage

- Widget

- Composant de l'IHM

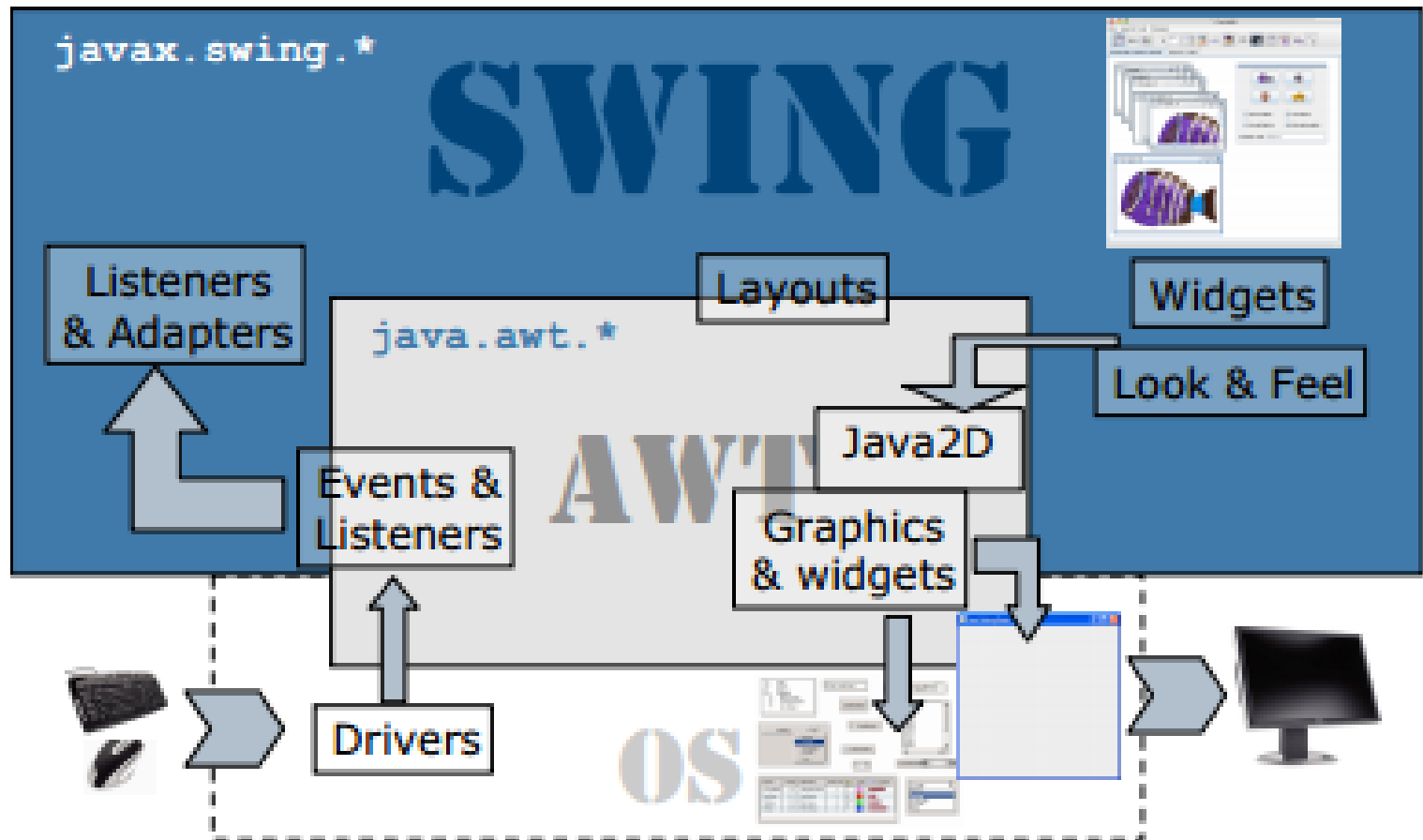
- Layout

- Positionnement des widgets

- Listener

- Gestion des évènements

Catégories des composants



■ Des questions?

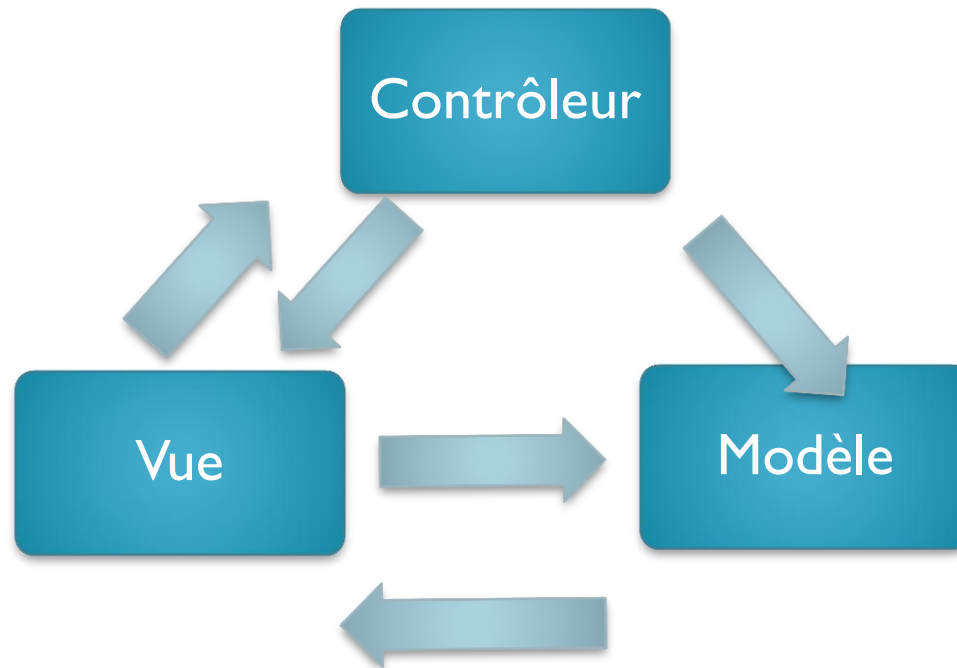




MVC et variantes

MVC

- Model-View-Controller
- Organiser les interfaces graphiques.
- Sépare les données, des visuels, des traitements.



MVC

- Modèle :

- Comportement de l'application.
- Gestion, accès et intégrité des données.

- Vue :

- IHM
- Réceptionne les actions.

- Contrôleur :

- Gestion et synchronisation des évènements.

Variantes

- MVC2 : un seul contrôleur.
- Plusieurs frameworks reprennent le pattern MVC
 - Swing
 - JavaServer Faces
 - Struts
 - Stripes
 - SWT...

■ Des questions?

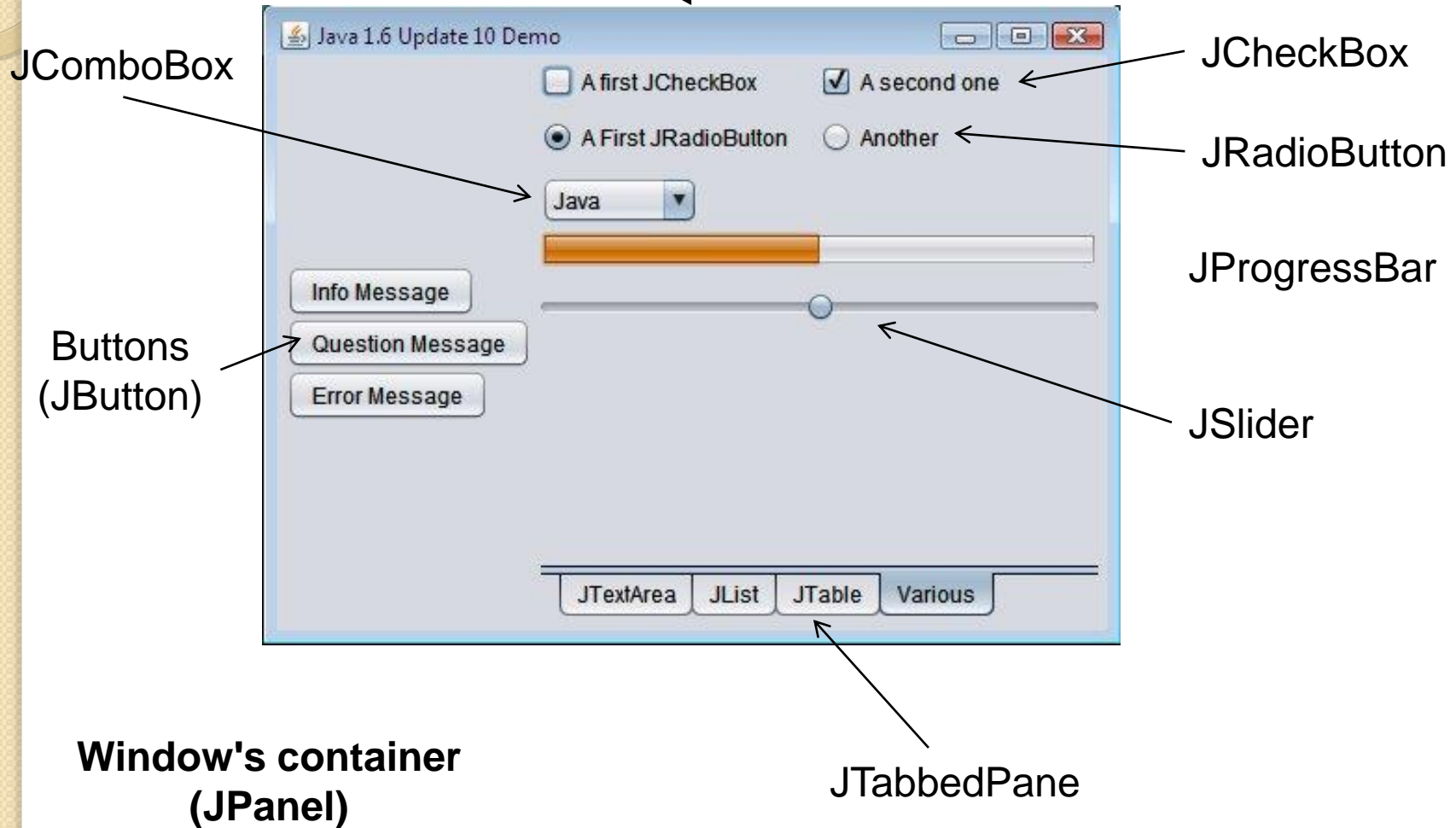




Fenêtre et conteneurs principaux

Architecture

**Window, main container
(JFrame)**



JFrame: description

- Le conteneur le plus haut niveau.
- Représente la fenêtre de l'application.
- Peut avoir :
 - Une taille
 - Un titre
 - Des composants

JFrame: méthodes

- *setSize(int width, int height)* ou *setSize(Dimension dim)* :
 - Taille de fenêtre.
- *setTitle(String title)* :
 - Titre de la fenêtre.
- *setVisible(boolean visible)* :
 - Fenêtre visible ou non.
- *setDefaultCloseOperation(int op)* :
 - Opération de fermeture.
 - Par défaut : rien.
- *setResizable(boolean resizable)* :
 - Redéfinition de la taille possible ou non.

JFrame: Example

```
public class MyFrame extends JFrame {  
    ...  
    public MyFrame() {  
        this.setSize(300, 200);  
  
        // define the default operation when  
        // the frame is closed  
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
  
        // define the title of the frame  
        this.setTitle("My wonderful frame");  
    }  
    ...  
}
```

JPanel: description

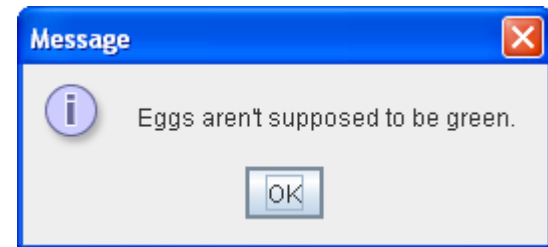
- Agrégation de composants.
- Généralement le conteneur principal de la fenêtre.
- Possède un layout
 - A voir plus tard

JPanel: Example

```
public class MyFrame extends JFrame {  
    ...  
    public MyFrame() {  
        ...  
        JPanel panel = new JPanel();  
        panel.add(new JButton("My button"));  
  
        // add more elements if you like  
  
        this.setContentPane(panel);  
    }  
    ...  
}
```

JDialog

- Boîte de dialogue modal ou non sur une JFrame
- JDialog(Frame frame, boolean modal, String title)
- S'appelle grâce à la méthode showXXXDialog(new Jdialog)
 - PLAIN_MESSAGE
 - ERROR_MESSAGE
 - INFORMATION_MESSAGE
 - WARNING_MESSAGE
 - QUESTION_MESSAGE





- Des questions?

- Une application!



Application

- Ouvrir avec Eclipse le projet Librairy qui se trouve dans le répertoire workspace.
- Lancer le programme principal.
- Ajouter au projet une fenêtre qui aura pour taille 400x600 et comme titre Lybrairy.
- Questions : que se passe-t-il lorsque vous fermer la fenêtre?



Menus

JMenuBar

- Bar de menu contenant les JMenu.
- Relié à une JFrame :
 - *JFrame.setJMenuBar(MenuBar menuBar)*
- Ajout de menu :
 - *MenuBar.add(Menu menu)*

JMenu

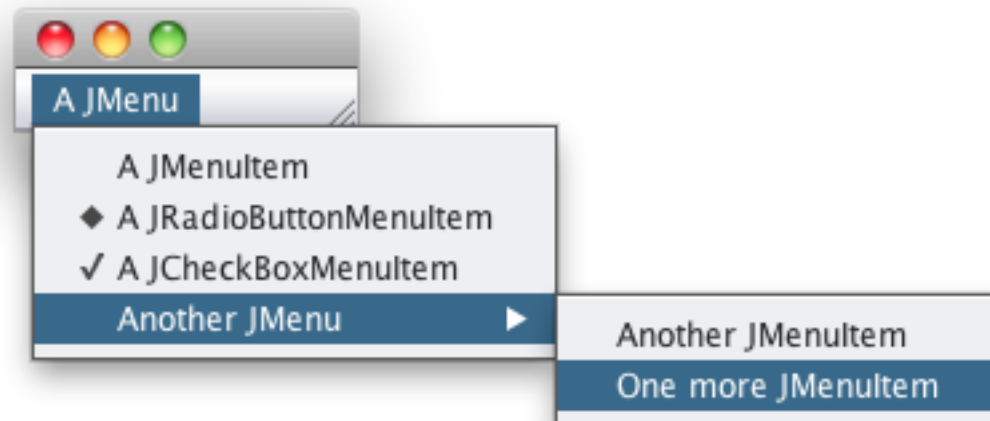
- Contient des *JMenuItems*. Add item to a menu :
 - *Menu.add(Menuitem item)*
- Peut avoir un raccourci clavier
- Exemple :

```
JMenu menu = new JMenu("File");  
// keyboard shortcut: CTRL+F  
menu.setMnemonic(KeyEvent.VK_F);
```

JMenuItem

- Item d'un menu
- Fonctionne comme un JButton :
 - Peut avoir un `ActionCommand`.
 - Utilise un `ActionListener` pour le click.
- Plusieurs types :
 - *JRadioButtonMenuItem* :
 - `JRadioButton`.
 - *JCheckBoxMenuItem* :
 - `JCheckBox`.

Example



JPopupMenu

- Menu contextuel.
- Click droit.
- Comme un JMenu.



- Des questions?

- Une application 😊



Application

- Rajouter le menu suivant à votre fenêtre principal :
 - File : chaque item ouvre une nouvelle fenêtre.
 - List of books
 - List of author
 - Add book
 - Add author
 - Help : affiche une JDialog qui indique le créateur et la date de développement.
 - Quit : demander la confirmation de l'utilisateur avant de quitter l'application.



Look & Feel

Introduction

- Nous pouvons changer l'apparence des applications JSE.
- Quelques styles sont disponibles :

<https://substance.dev.java.net/>

- Comme **Nimbus** introduit depuis Java 1.6.

Classes

■ **UIManager :**

- Permet de changer les apparences.

- *LookAndFeelInfo[] UIManager.getInstalledLookAndFeels()*

- *UIManager.setLookAndFeel(String className)*
LookAndFeel UIManager.getLookAndFeel()

- Récupère l'apparence courante.

■ **LookAndFeelInfo :**

- Récupère les informations de l'apparence.

- *String getName()*

- *String getClassName()*



- Des questions?

- Une application 😊



Application

- Appliquer le style Nimbus à votre projet.



Composants Swing

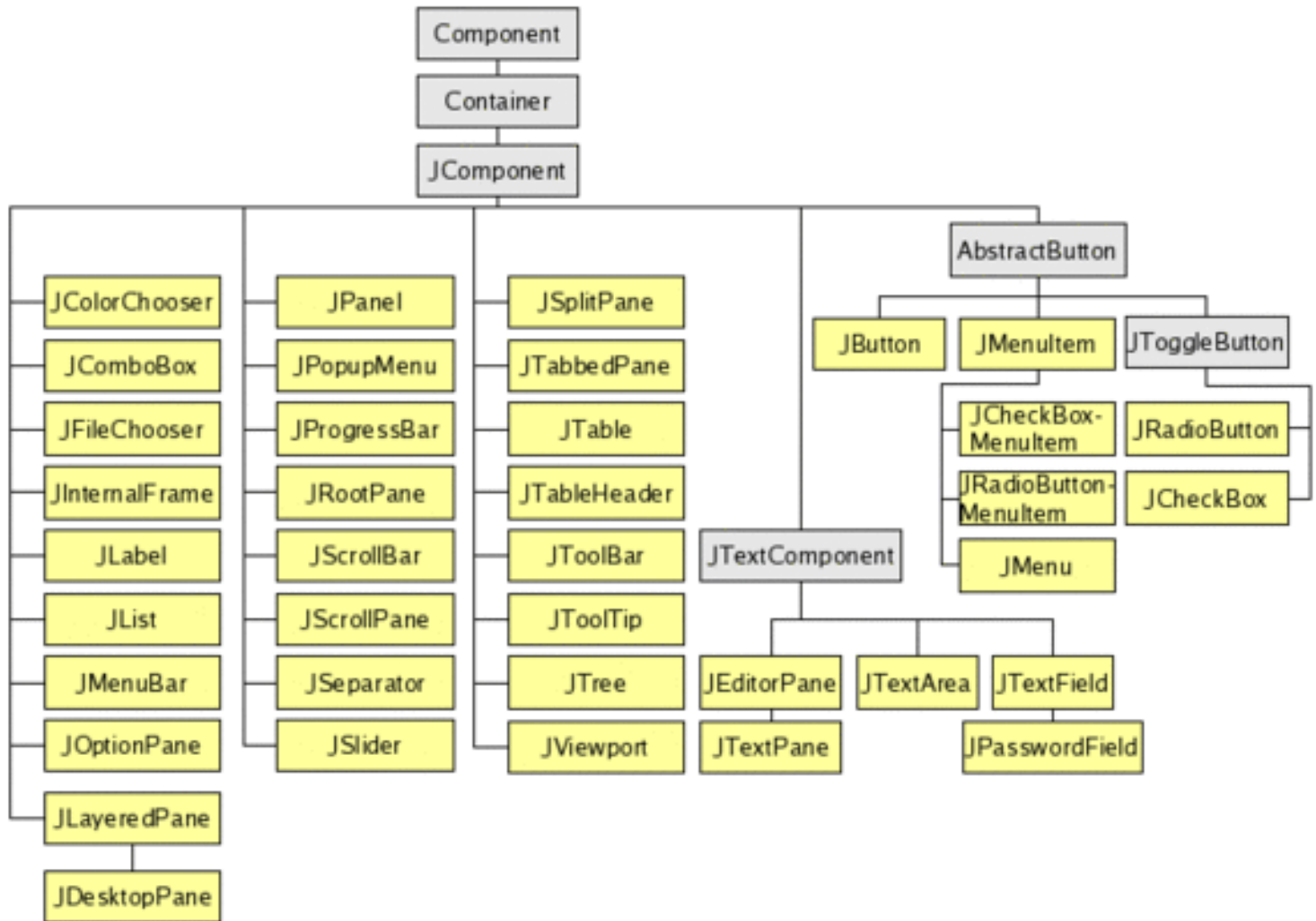
Composants Swing

- Hiérarchie
- Composants simples
- Composants texte



Hiérarchie

Hiérarchie





Composants simples

JLabel

- Un texte et/ou une image.
 - `setText(String text)`
 - `setIcon(Icon icon)`
- Constructeurs :
 - `JLabel()`
 - `JLabel(String text)`
 - `JLabel(Icon icon)`

JButton

I love Java

- Bouton classique.
- Définition de l'action du bouton:
 - ***addActionListener(ActionListener event)***
- Constructeurs :
 - *JButton()*
 - *JButton(String text)*
 - *JButton(Icon image)*

JList

- Liste simple.
- Multiple sélection possible.
- Modèle possible.
- Constructeurs:
 - *`JList(Object[] items)`*
 - *`JList(Vector<?> items)`*
- Méthodes utiles:
 - *`int getSelectedIndex()`*
 - *`int[] getSelectedIndices()`*
 - *`Object getSelectedValue()`*
 - *`Object[] getSelectedValues()`*



JCheckBox

- Check box.

☒ Java is soooo cool

- Constructeurs :

- *JCheckBox(String text)*

- *JCheckBox(String text, boolean selected)*

- Méthodes utiles :

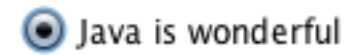
- *boolean isSelected()*

- *void setSelected(boolean selected)*

- *String getText() / void setText(String label)*

JRadioButton

- Radio button.



- Constructeurs:

- *JRadioButton(String text)*

- *JRadioButton(String text, boolean selected)*

- Méthode utile:

- *boolean isSelected() / void setSelected(boolean)*

ButtonGroup

- Lier des radios boutons:

- Sélection exclusive.

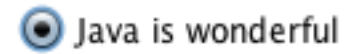
- Méthodes utiles:

- *void add(AbstractButton btn) :*

- Ajout des boutons dans le groupe.

- *void remove(AbstractButton btn) :*

- Enlève un bouton.



JComboBox

- JComboBox

- Constructeurs:

- *JComboBox(Object[] items)*

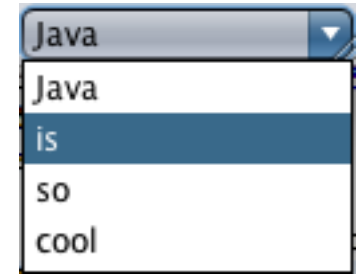
- *JComboBox(Vector<?> items)*

- Méthodes utiles:

- *int getSelectedIndex()*

- *Object getSelectedItem()*

- Remarque: le texte affiché est celui de la méthode **toString()**.



ActionCommand

- Disponible sur beaucoup de composants.
- Une String.
- **Définit l'action du composant:**
 - Click, focus, select ...
- Set it by: ***void setActionCommand(String command)***.
- Get it by: ***String getActionCommand()***.



Composants texte

TextField

- Zone de texte
- Possède un nombre de colonnes.
- Constructeurs :
 - *TextField(int columns)*
 - *TextField(String defaultText)*
 - *TextField(String defaultText, int columns)*
- Méthodes utiles:
 - *void setText(String text)*
 - *String getText()*



I really love Java

JPasswordField

- Zone de texte secret
- Se gère comme un JTextField sauf pour récupérer le password:
 - `char[] getPassword()`

JFormattedTextField

- Formate le contenu d'un JTextField
- Retourne la valeur uniquement si elle correspond au format
- Le format se déclare avec le constructeur
 - `JFormattedTextField(NumberFormat.getIntegerInstance())`
 - NumberFormat, DateFormat, MessageFormat



JTextArea

- Zone de texte avec des lignes et des colonnes.
- Constructeurs :
 - *JTextArea(int rows, int columns)*
 - *JTextArea(String defaultText)*
 - *JTextArea(String defaultText, int rows, int columns)*
- Méthodes utiles:
 - *String getText() / void setText(String text)*
 - *int getRows() / void setRows(int nbOfRows)*
 - *int getColumns() / void setColumns(int nbOfCol)*



Java rocks so much



- Des questions?

- Une application! 😊



Application

- Ajouter dans AddBookFrame une zone de texte avec un label et un bouton « ok » et « cancel »
- Faire les mêmes manipulations pour AddAuthorFrame
- Ajouter une comboBox dans AddBookFrame pour sélectionner l'auteur du livre.



Gestion des Layouts

Gestion des Layouts

- Layout
- Positionnement absolu
- Layouts de Swing



Layout

Layout

- Gestion de la position des composants.
 - Dans un conteneur :
 - Class extending **Container** class comme JPanel.
- Nombreux choix.
- On peut les mettre:
 - Pour tous les coneneurs:
 - *setLayout(LayoutManager layout).*
 - Seulement sur les JPanel :
 - *JPanel(LayoutManager layout).*



- Des questions?



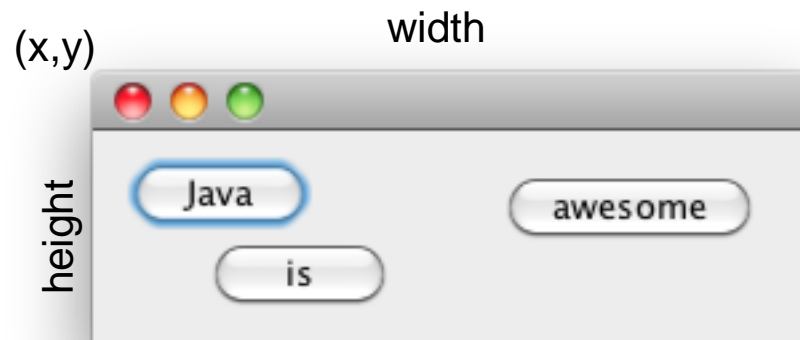
- Une application complète ! Afin de finir 😊



Positionnement absolu

Absolute positioning

- You can place your components wherever you want.
- Set it up :
 - Set the Layout container to **null**.
 - Add your components to the container.
 - Specify the **bounds** for each added component.



Position absolue

- Selon un X et Y en relation avec la largeur et la hauteur du composant :

- *setBounds(int x, int y, int width, int height)*

- Exemple :

```
Dimension dim = myLabel.getPreferredSize();  
myLabel.setBounds(10, 10, dim.width, dim.height);
```

Absolute positioning

■ Exemple :

```
// ...
JPanel panel = new JPanel();
panel.setLayout(null);

JButton playButton = new JButton("Play");
JButton exitButton = new JButton("Exit");

Dimension playSize = playButton.getPreferredSize();
Dimension exitSize = exitButton.getPreferredSize();

playButton.setBounds(10, 10, playSize.width, playSize.height);
exitButton.setBounds(10, 50, exitSize.width, exitSize.height);

panel.add(playButton);
panel.add(exitButton);
// ...
```

■ Des questions?





Layouts standards

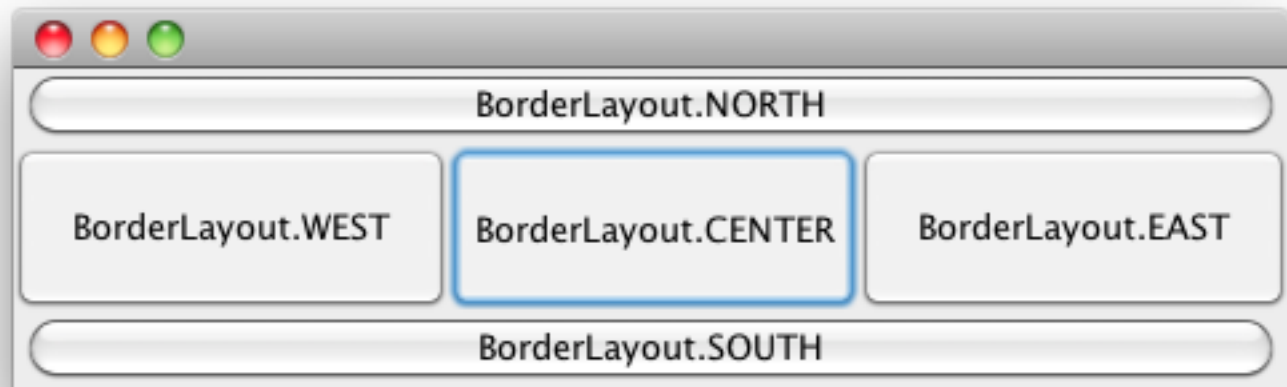
FlowLayout

- Layout par défaut des JPanel
- Composant de gauche à droite:
 - Sur la même ligne.
 - Création d'une nouvelle ligne si nécessaire.
 - Chaque ligne est centrée.



BorderLayout

- Séparation en 5 régions.
- Région par défaut : *BorderLayout.CENTER*.
- Layout par défaut des JFrame.

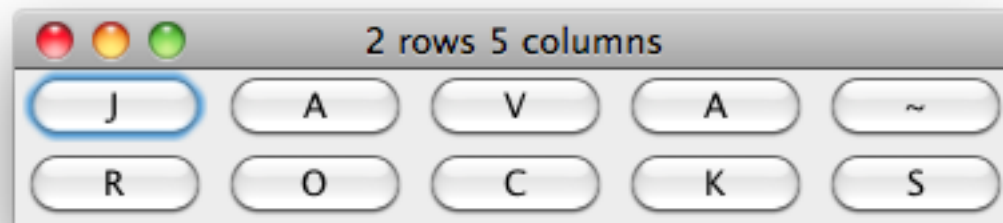


- Exemple :

```
JPanel panel = new JPanel(new BorderLayout());  
panel.add(new JButton("My Button"), BorderLayout.EAST);
```

GridLayout

- Sous forme de grille
- Même taille pour toutes les cellules.
- Le composant prend toutes la place possible de la cellule.

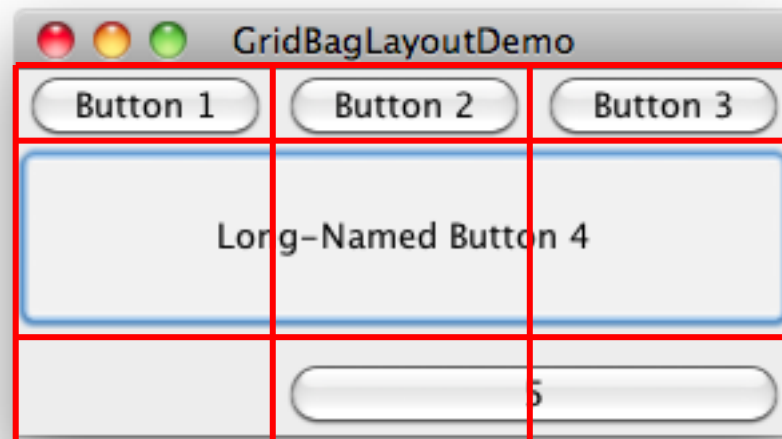


- Exemple :

```
JPanel panel = new JPanel(new GridLayout(2, 5));  
panel.add(new JButton("J"));  
panel.add(new JButton("A"));  
// ...
```


GridBagLayout

- Un des plus flexibles et complexes.
- Comme un GridLayout sauf que:
 - Les colonnes et les lignes peuvent être multiples,
 - Et de taille différentes



GridBagLayout

- L'objet *GridBagConstraints* permet d'en définir les caractéristiques.
- Variables utiles:
 - ***int gridx, gridy*** : position du composant dans la grille
 - ***int gridwidth, gridheight*** : nombre de cellules utilisées par le composants
 - ***int fill*** : comment redéfinir la taille quand la zone d'affichage des composant est plus large que la taille des composants. Les valeurs valides sont définies par les constantes *GridBagConstraints*.
 - ***int anchor*** : où sont placés les composants dans la zone d'affichage. Les valeurs valides sont définies par les constantes *GridBagConstraints*.

GridBagLayout

Complete Exemple :

```
// ...
pane.setLayout(new GridBagLayout());
GridBagConstraints c = new GridBagConstraints();
c.fill = GridBagConstraints.HORIZONTAL;

JButton button1 = new JButton("Button 1");
pane.add(button1, c);

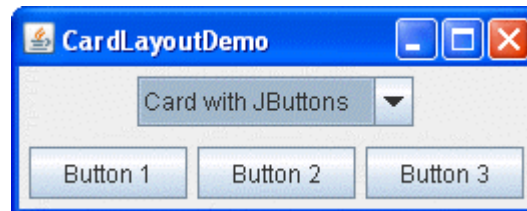
JButton button2 = new JButton("Button 2");
c.weightx = 0.5;
pane.add(button2, c);

JButton button3 = new JButton("Button 3");
pane.add(button3, c);

JButton button4 = new JButton("Long-Named Button 4");
c.ipady = 40; c.weightx = 0.0; c.gridwidth = 3; c.gridy = 1;
pane.add(button4, c);
// ...
```

CardLayout

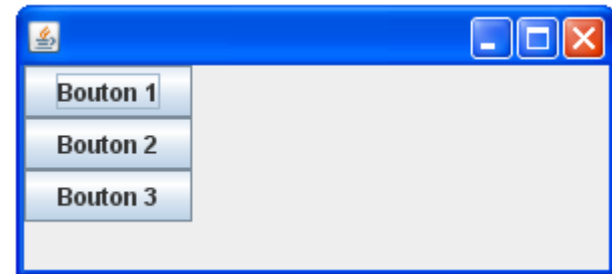
- Disposition selon une pile
- `CardLayout(int hgap, int hvgap)`
- Par défaut, le composant visible est le premier empilé.
 - Sinon méthodes `next`, `previous`, `first`, `last`



GroupLayout

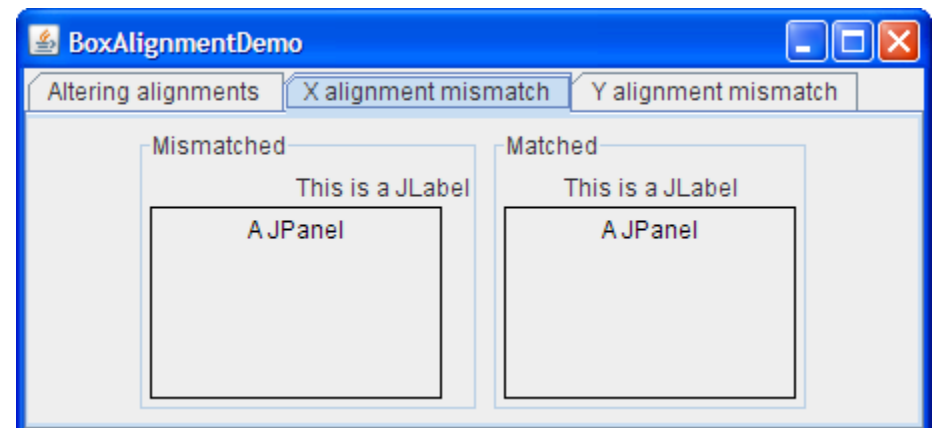
- Regroupe les composants graphique par groupe en les collant les uns aux autres.
- Placer les composants par rapport à l'axe horizontale et vertical (séquentiellement, parallèle)

```
GroupLayout gl = new GroupLayout();  
GroupLayout.SequentialGroup hg = gl.createSequentialGroup();  
Gl.setSequantialGroup(hg);
```



BoxLayout

- Disposition selon une seule ligne ou une seule colonne :
GridLayout à une seule ligne ou colonne
- `BoxLayout bl = Box.CreateHorizontalBox(); //ou vertical`
- Donner une marge de séparation :
 - `Box.createVerticalStrut(10)`



■ Des questions?

■ Une application! 😊



Application

- Appliquer les layouts suivants à la fenêtre `AddBookFrame` :
 - Un `FlowLayout` pour les boutons « ok » et « cancel »
 - Un `GridBagLayout` pour le label, le zone de texte et la combobox.



Gestion des évènements

Gestion des évènements

- Listeners et JavaBean
- Types de notifications
- Hiérarchie des évènements
- Classes internes

Règle de la souscription

- Les éléments graphiques ...
 - Buttons, windows, ...
- ... doivent souscrire à des listeners!
- Listeners :
 - Interfaces gérant les évènements:
 - Click, redimensionnement...
 - **Appelé automatique.**

Souscription

- Très flexible:
 - Presque tout ce que nous voulons faire.
- Facile à mettre en oeuvre.
- Facile à utiliser.

ActionListener

- **ActionListener** : interface.
- Evènement : click sur un bouton.
- On doit définir la méthode **actionPerformed(ActionEvent e)** :
 - Méthode appelée automatique au click du bouton.
- La classe JButton contient la méthode :
addActionListener(ActionListener listener)

Définir un Listener

- Trois possibilités:
 - La classe courantes l'implémente.
 - Créer une nouvelle classe.
 - Créer une classe anonyme.

Définir un Listener

- Classe implémentant:

```
public class MyFrame implements ActionListener {  
  
    private JButton button;  
  
    public MyFrame() {  
        button = new JButton("My button");  
        button.addActionListener(this);  
    }  
  
    @Override  
    public void actionPerformed(ActionEvent ae) {  
        System.out.println("I performed a click ! It works !!");  
    }  
}
```

Définir un Listener

■ Nouvelle classe

```
public class MyFrame {  
  
    private JButton button;  
  
    public MyFrame() {  
        button = new JButton("My button");  
        button.addActionListener(new MyListener());  
    }  
}
```

```
public class MyListener implements ActionListener {  
  
    @Override  
    public void actionPerformed(ActionEvent ae) {  
        System.out.println("I performed a click ! It works !!");  
    }  
}
```


Définir un Listener

■ Classe anonyme

```
public class MyFrame {  
  
    private JButton button;  
  
    public MyFrame() {  
        button = new JButton("My button");  
        button.addActionListener(new ActionListener() {  
  
            @Override  
            public void actionPerformed(ActionEvent e) {  
                System.out.println("I performed a click ! It works !!");  
            }  
  
        });  
        add(button);  
    }  
}
```

Autres Listeners

- *WindowListener* :
 - Evènements de la fenêtre
- *MouseListener* :
 - Click, pression
- *KeyListener* :
 - Utilisation du clavier
- *FocusListener* :
 - Focus: perte...
- ...

ActionEvent

- Package **java.awt.event**.
- Informations d'un évènement.
- Méthodes utiles:
 - ***String getActionCommand()*** :
 - Récupère l'action command du bouton.
 - ***Object getSource()*** :
 - Composant qui a provoqué l'action



- Des questions?

- Une application! 😊



Application

- Donner les évènements correspondants pour les boutons développés dans vos fenêtres.



Composants et principes avancés

Composants et principes avancés

- JProgressBar
- Gestion de tables
- Gestion des arbres



JProgressBar

JProgressBar

- Barre de progression.
- Constructeurs:
 - *JProgressBar(int min**Value**, int max**Value**).*
- Méthodes utiles:
 - *int getValue() / void setValue()*
 - *void setString(String text)*
 - *void setStringPainted(boolean painted) :*
 - Texte visible, faux par défaut.





- Des questions?

- Une application! 😊



ProgressBar

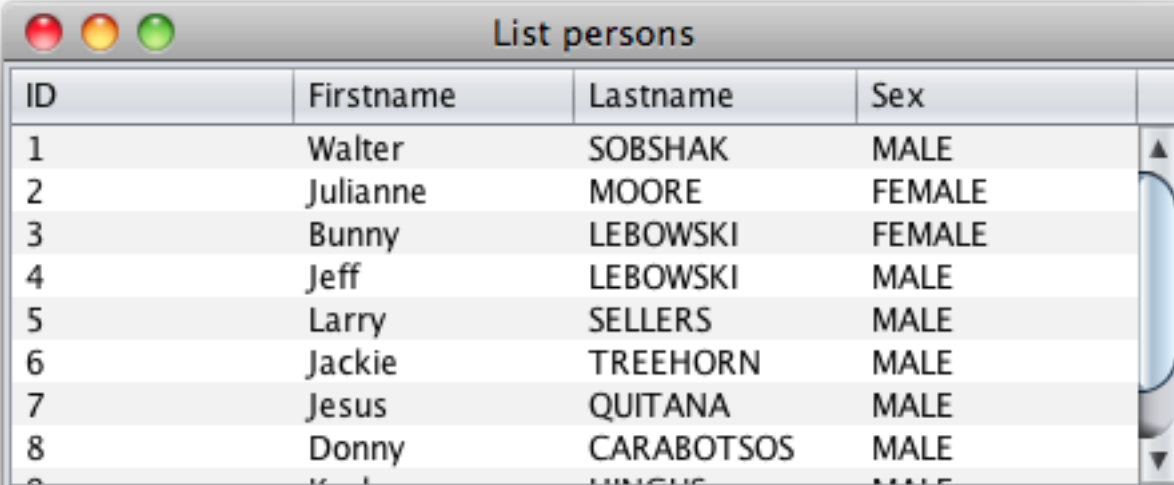
- Au lancement de la fenêtre principale, une barre de progression doit s'afficher pendant un minimum de 5 secondes.



Table

JTable

- Listing.
- Placé en général dans un JScrollPane.



ID	Firstname	Lastname	Sex
1	Walter	SOBSHAK	MALE
2	Julianne	MOORE	FEMALE
3	Bunny	LEBOWSKI	FEMALE
4	Jeff	LEBOWSKI	MALE
5	Larry	SELLERS	MALE
6	Jackie	TREEHORN	MALE
7	Jesus	QUITANA	MALE
8	Donny	CARABOTSOS	MALE

Table model

- Main interface : ***TableModel***.
- Main abstract class : ***AbstractTableModel***.
- Main concrete class : ***DefaultTableModel***.
- Gestion de chaque cellule
 - Editable ou pas.
- Vue gérée automatiquement
- On définit son propre modèle en étendant une classe abstraite ou en implémentant une interface :
 - Nombre de colonnes?
 - Nombres de lignes?
 - Entêtes?
 - ...

Table model

- Réécriture de quelques méthodes :
 - *int getColumnCount()*
 - Nombres de colonnes à afficher.
 - *int getRowCount()*
 - Nombre de lignes à afficher.
 - *String getColumnName(int col)*
 - Header des colonnes
 - *Object getValueAt(int row, int col)*
 - Valeur de la cellule

Table model

- *boolean isCellEditable(int row, int col)*
 - Cellule éditable ou non.
- *void setValueAt(Object value, int row, int col)*
 - Mets une nouvelle valeur à la cellule.
 - **Doit** être redéfinie si la cellule est éditable

Table model

■ Exemple :

```
public class MyModel extends AbstractTableModel {  
    private List<Person> persons;  
    // ...  
    @Override  
    public int getColumnCount() {  
        return 4;  
    }  
    @Override  
    public int getRowCount() {  
        return getPersons().size();  
    }  
    @Override  
    public boolean isCellEditable(int row, int col) {  
        return false;  
    } // ...  
}
```

Table model

■ Exemple :

```
// ...
@Override
public String getColumnName(int col) {
    if(col == 0) return "ID";
    else if(col == 1) return "Firstname";
    // ...
}
@Override
public Object getValueAt(int row, int col) {
    Person p = getPersons(row);
    if(col == 0) return p.getId();
    else if(col == 1) return p.getFirstname();
    // ...
}
}
```

Table model

- Mettre à jour la vue...
 - Ajout/suppression d'une ligne.
 - Mise à jour d'une cellule.
 - ...
- ... peut appeler la méthode **fireTableDataChanged()**
- La vue se rafraichit toute seule!



- Des questions?

- Une application! 😊



Application

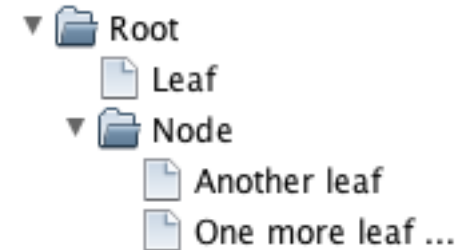
- Dans la fenêtre ListBookFrame, afficher les livres sous forme d'une table éditable.



Arbre

JTree

- Arborescence
- Composé de noeuds et de feuilles.
- Contient **MutableTreeNode** :
 - Pour les noeuds
 - Pour les feuilles.



```
DefaultMutableTreeNode root = new DefaultMutableTreeNode("Root");  
root.add(new DefaultMutableTreeNode("Leaf"));
```

```
DefaultMutableTreeNode node = new DefaultMutableTreeNode("Node");  
node.add(new DefaultMutableTreeNode("Another leaf"));  
node.add(new DefaultMutableTreeNode("One more leaf..."));
```

```
root.add(node);
```

JTree

- Récupérer la sélection:

```
DefaultMutableTreeNode node =  
    (DefaultMutableTreeNode) getLastSelectedPathComponent();
```

- Méthodes sur un *DefaultMutableTreeNode* :

- *boolean isLeaf()*

- Noeud ou feuilles?

- *boolean isRoot()*

- Racine de l'arbre?

- *TreeNode[] getPath()*

- Chemin du root jusqu'au noeud.



- Des questions?

- Une application! 😊



Application

- Dans la fenêtre ListAuthorFrame, afficher les auteurs sous forme d'un arbre où chaque auteur est un noeud possédant comme fils le titre de ses livres.



Déploiement

Déploiement

- Création de jar exécutables
- Signature



Création de jar exécutable

Jar exécutable

- Java est multi-plateforme : pas de .exe ou .sh ou .app
 - Uniquement un jar exécutable sur tout OS supportant le java ! 😊
- Soit par ligne de commande, soit par Eclipse (Créer un jar exécutable)

Faire un jar

- Grâce au JDK.
- Ligne de commande:

```
jar [options] [manifest] destination.jar input-file [input-file]
```

Option de la ligne de commande

Options	Description
c	Créer une nouvelle archive
f	Spécifie le nom de l'archive
v	Mode verbeux
t	Liste le contenu du jar
x file	Extrait le contenu du jar
u	Mise à jour de l'archive
m	Spécifie le fichier manifest

Le fichier manifest

- Requis pour rendre le jar exécutable.
- Nommé **manifest.mf**.

Manifest-version: 1.0

Main-Class: class.containing.the.main.method.to.Launch

- Création d'un jar exécutable :

```
jar cmf manifest.mf MyCoolJar.jar MyClass.class java.jpg
```



Signature

Jar signé

- Deux outils sont requis :
 - keytool :
 - Créer une clé dans un keystore en indiquant les informations sur l'application demandées.
 - jarsigner :
 - Signer le jar avec une clé du keystore.
- En ligne de commande :

```
keytool -genkey -keystore myKeys -alias aKey  
jarsigner -keystore myKeys MyCoolJar.jar aKey
```



- Des questions?



- Une application! 😊

- Faire un jar exécutable de votre application.



Présentation des concepts avancés de Swing

Concepts avancés de Swing

- Swing application framework
- Internationalisation
- Technologies de Binding
- Framework de validation de JGoodies



JSR 296

JSR 296

- Ensemble de classes qui vise à simplifier le développement avec Swing.
- Définit un socle commun :
 - Gestion du cycle de vie
 - Support pour la gestion des ressources
 - Support la définition, la gestion et l'enregistrement des Swing Actions
 - Persistance de la session et préférences

Gestion du cycle de vie

- Classe Application
- Centralise la gestion du cycle de vie de l'application
- Méthodes : `launch()`, `stratup()`, `shutdown()`, `exit()`

Les ressources

- ResourceBundle : fichier plat de propriétés.
- Apporte la possibilité de stocker des images, des valeurs numériques et des codes.
- Héritage des ressources.
- Classe ResourceMap : décorateur de la classe ResourceBundle qui convertit et met en cache les propriétés. label).

Gestion des actions

- Évènement déclenché par l'utilisateur sur l'interface.
- Annotation `@Action` : une méthode est une action

■ Des questions?





Internationnnalisation

Fichier de propriétés

- Syntaxe : key=value
- Suffixe du fichier :
 - **lang_<lang>_<country>.properties**
 - Ex : lang_fr_CA.properties
 - **_<lang>.properties**
 - Ex : lang_fr.properties
 - Attention : sans suffixe = langue par défaut

cancel=Annuler
validate=Valider
personList=Liste des personnes

ResourceBundle

- Gestion des fichiers de propriétés
- *ResourceBundle.getBundle(String basename)*

```
ResourceBundle.getBundle("com.sun.myapp.lang.myLang");
```

- Récupérer une valeur :

```
myBundle.getString("cancel");
```

Classe Locale

- Situation géographique.
- Méthodes
 - *static Locale getDefault()*
 - Situation par défaut de la JVM.
 - *static void setDefault(Locale l)*
 - Setter de la situation par défaut.
 - *static Locale[] getAvailableLocales()*
 - Toutes les situations supportées par la JVM.
 - *String getCountry()*
 - *String getLanguage()*

■ Des questions?





Binding

Binding

■ Lier un objet à un composant graphique

```
Person person = new Person("djo");  
/* ----- Start Swing Binding ----- */  
SwingRealm.createDefault(); Realm realm =  
    SwingObservables.getRealm();  
DataBindingContext context = new DataBindingContext(realm);  
// Bind JavaBean User name getter/setter (java.lang.String) with Swing  
// JTextField.  
context.bindValue(  
    SwingObservables.observeText(  
        jTextField, SwingEventConstants.Modify),  
        BeansObservables.observeValue(person, "name"),  
        null, null);  
/* ----- End Swing Binding ----- */
```

■ Des questions?





Présentation de JGoodies

JGoodies

- Conception facile d'une interface en Java.
- 5 librairies openSource
- 8 applications et démonstration
- 1 suite payante

Librairies

- Animation
 - Animation 2D en temps réel
- Binding
 - MVC
- Forms
 - Création de formulaire
- Looks
 - Styles prédéfinis pour le rendu graphique
- Validation
 - Contraintes sur les champs

- Des questions?
- Merci pour votre attention.
- Passer un bon week-end !

