Les Interfaces Graphiques

Bibliothèques (API)

- API Graphiques
  - Bibliothèques de composants graphiques permettant de réaliser des IHM complexes
  - Exemple de composants : fenêtres, conteneurs, zones de texte, listes, boutons, boites de dialogue, …

- Structure d'une interface
  - (Une arborescence de composant)
  - Conteneur de 1er niveau
    - Interface (programme) avec l'OS
    - Contient en général des sous conteneurs
  - Conteneur
    - Contient des composants
    - Mettant généralement en œuvre gestionnaires de « mise en page » pour organiser les composants contenus
    - NB : les conteneurs sont des composants…
  - Widgets
    - Composants graphiques qui constituent les « feuilles » de l'arbre
    - Ce sont les
    - zones de texte, listes, box, boutons, …

Les APIs

- **Les principales :**
  - **AWT, Swing, SWT, Jface, JavaFX**

- **AWT**
  - **1ere API disponible en Java**
  - **Nombre limité de composants disponibles**
  - **Faible possibilité de gestion du rendu**
- **Swing**
  - **Construite sur AWT (le concepteur doit d'ailleurs directement utiliser certains des composants d'AWT)**
  - **Grand nombre de composants, du plus simple à des beaucoup plus complexes**

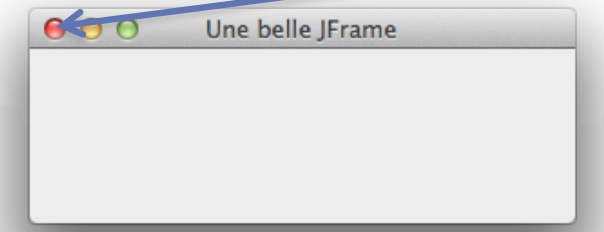
- **SWT**
  - **API graphique non standard (initiée par IBM)**
  - **Encapsulation directe des composants graphiques des divers OS**
  - **(base de l'IHM d'Eclipse)**
- **JFace**
  - **API pour des composants de haut niveau bâtie sur SWT**

- **JavaFX**
  - **Bibliothèque graphique officielle standard de Java 8**

Mise en œuvre de Swing

- Conteneurs de 1er niveau
  - JApplet
    - Fenêtre graphique destinée à être incluse dans une page HTML
    - Prise en charge par ma machine virtuelle d'un navigateur Internet
  - JWindow
    - Fenêtre graphique très rudimentaire : un simple rectangle à l'écran
    - Rarement utilisée
  - JFrame
    - Le fenêtre d'application autonome classique
    - Ex :

```
JFrame f = new JFrame("Une belle JFrame") ;
f.setVisible(true) ;
```



Attention : les boutons ne sont pas directement totalement fonctionnels

Une fenêtre fonctionnelle

```
import java.awt.Rectangle;
import javax.swing.JFrame;

public class GestionnaireContacts extends JFrame {

   public GestionnaireContacts() {
      super("Gestionnaire de contacts") ; // constructeur de la classe mère

      setBounds(new Rectangle(500, 300)) ; // fixe la taille de la fenêtre
      setDefaultCloseOperation(EXIT_ON_CLOSE) ; // arrête le programme lorsqu'on ferme la fenêtre

      setVisible(true) ; // affichage de la fenêtre
   }

   public static void main(String[] args) {
      GestionnaireContacts appli = new GestionnaireContacts() ; // instanciation
   }

}
```

Ajout de composants

```
import java.awt.Rectangle;

import javax.swing.JFrame;
Import javax.swing.JButton ;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;

public class GestionnaireContacts extends JFrame {

        public GestionnaireContacts() {
                super("Gestionnaire de contacts") ;

                setBounds(new Rectangle(500, 300)) ;
                setDefaultCloseOperation(EXIT_ON_CLOSE) ;

                JLabel textLabel = new Jlabel("Nom : ") ; // texte non editable
                JTextField nomField = new JTextField() ; // zone de texte sur 1 ligne
                JButton helloButton = new JButton("Dire bonjour") ; // un bouton

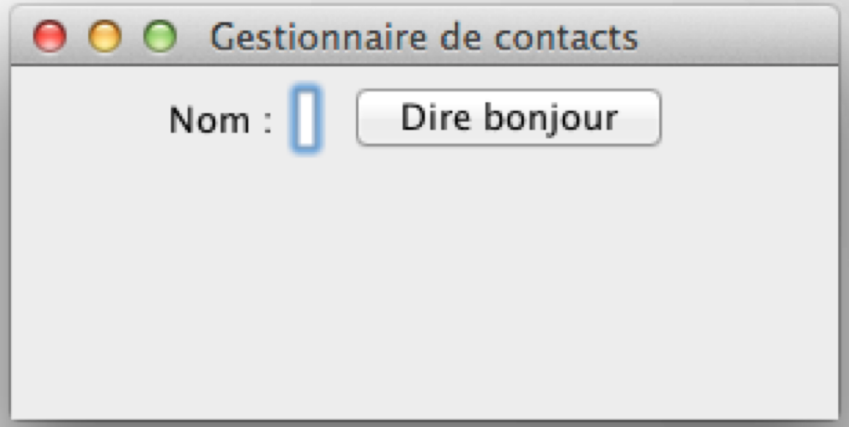
                JPanel saisieNom = new JPanel() ; // Un sous conteneur dans lequel on groupe les widgets
                saisieNom.add(textLabel) ;
                saisieNom.add(nomField) ;
                saisieNom.add(helloButton) ;

                this.getContentPane().add(saisieNom) ; // ajout à la fenêtre

                setVisible(true) ;
        }

        public static void main(String[] args) {
                GestionnaireContacts appli = new GestionnaireContacts() ;
        }

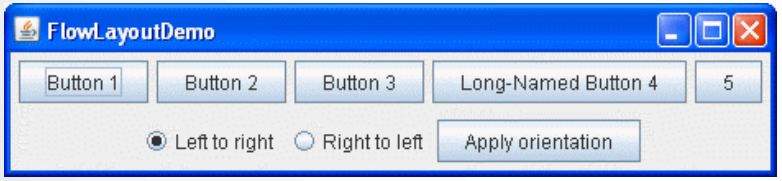
}
```

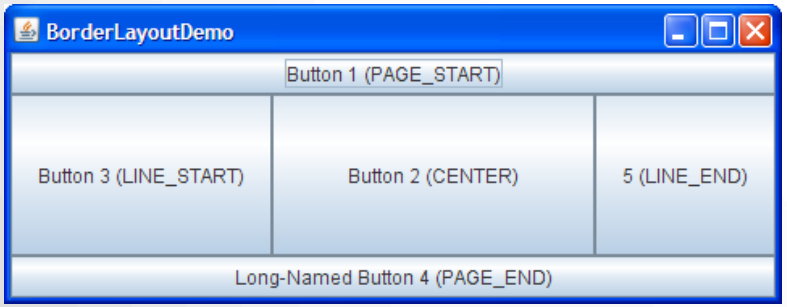


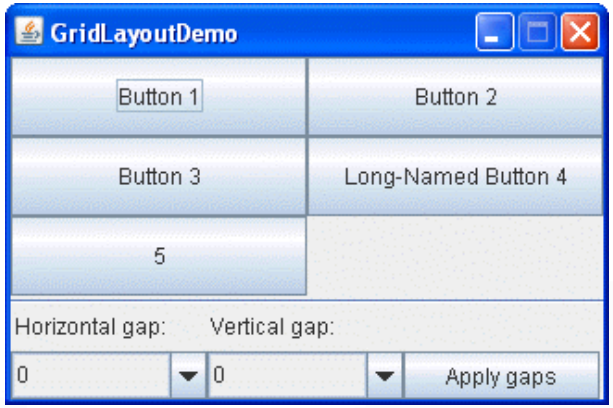
Positionnement

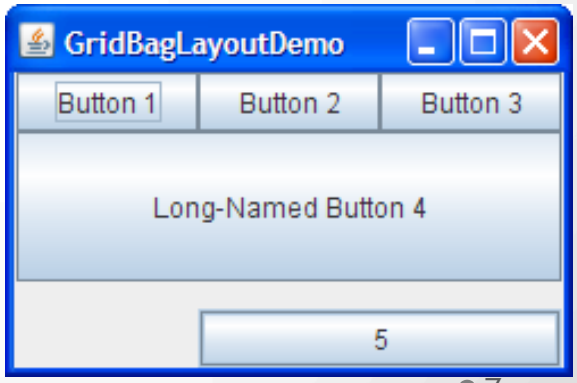
- Les **LayoutManager**
  - Permettent d'indiquer le positionnement des composants au sein d'un conteneur
  - Positions non « absolues » ce qui facilite la gestion en cas de redimensionnement

- Principaux Managers
  - FlowLayout (par défaut)
  - BorderLayout
  - GridLayout
  - BoxLayout (plus malléable que GridLayout, mais plus complexe aussi)
  - GridBagLayout (le plus flexible/complexe des managers)









Exemple avec **BorderLayout**

```
import java.awt.BorderLayout;
...

public class GestionnaireContacts extends JFrame {

        public GestionnaireContacts() {
                super("Gestionnaire de contacts") ;

                setBounds(new Rectangle(300, 150)) ;
                setDefaultCloseOperation(EXIT_ON_CLOSE) ;

                JLabel textLabel = new JLabel("Nom :") ;
                JTextField nomField = new JTextField() ;
                JButton helloButton = new JButton("Dire bonjour") ;

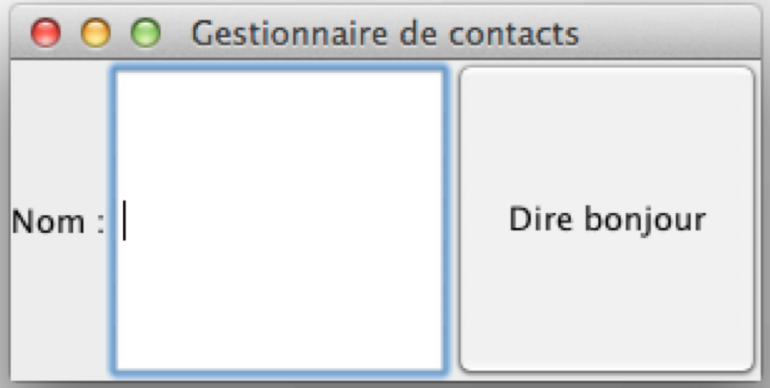
                JPanel saisieNom = new JPanel() ;
                saisieNom.setLayout(new BorderLayout()) ;
                saisieNom.add(textLabel, BorderLayout.WEST) ;
                saisieNom.add(nomField, BorderLayout.CENTER) ;
                saisieNom.add(helloButton, BorderLayout.EAST) ;

                this.getContentPane().add(saisieNom) ;

                setVisible(true) ;
        }

        public static void main(String[] args) {
                GestionnaireContacts appli = new GestionnaireContacts() ;
        }

}
```



Exemple avec **GridBagLayout**
(les contraintes sont gérées par **GridBagConstraints**)

```
...

JPanel saisieNom = new JPanel() ;

GridBagLayout gl = new GridBagLayout() ;
GridBagConstraints gc = new GridBagConstraints() ;
saisieNom.setLayout(gl) ;

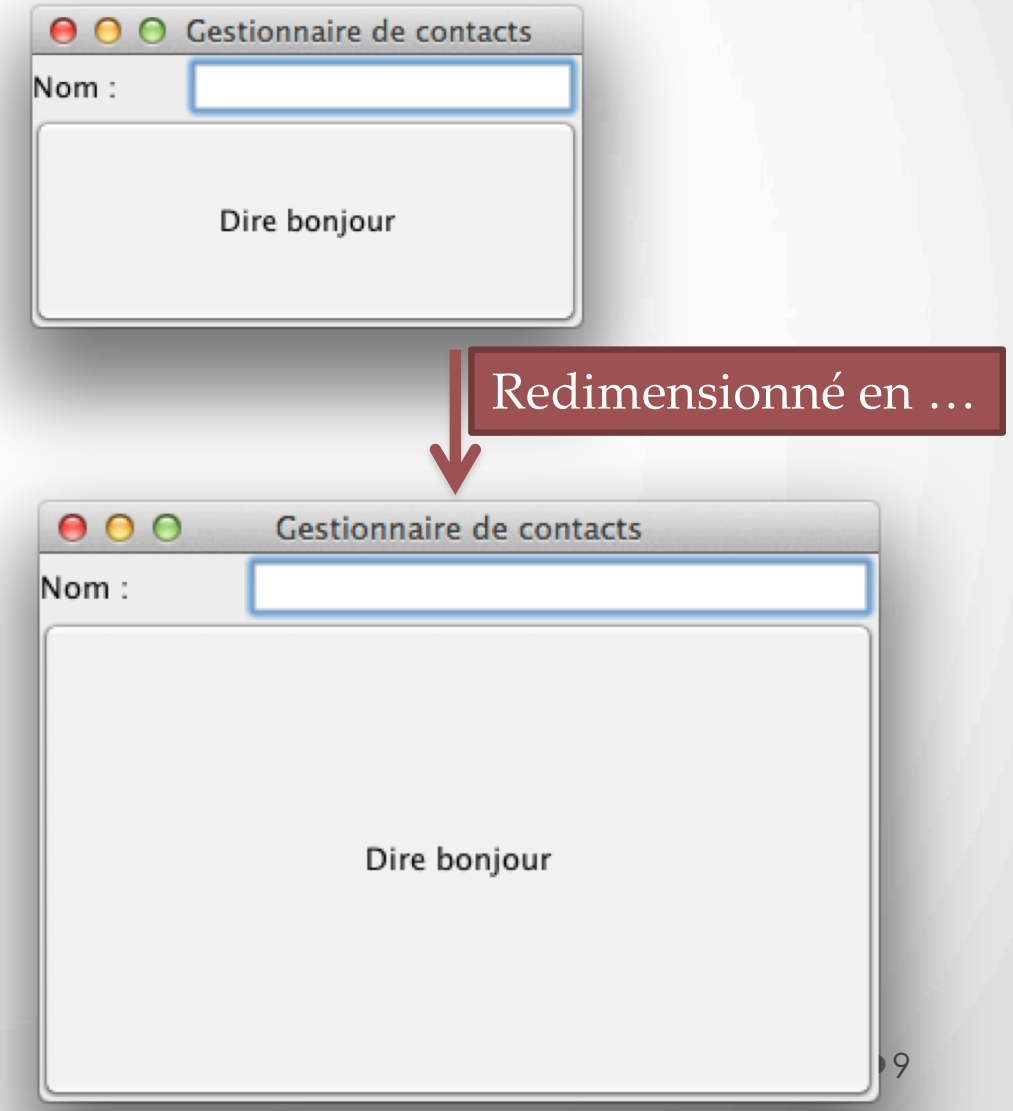
gc.gridx = 0 ; gc.gridy = 0 ; // coordonnées de la cellule
gc.fill = GridBagConstraints.BOTH ; // remplissage
gc.weightx = 0.2 ; // poids sur la ligne
gc.weighty = 0 ; // poids sur la colonne
gl.setConstraints(textLabel, gc) ; // application des contraintes
saisieNom.add(textLabel) ; // ajout au conteneur

gc.gridx = 1 ; // changement de cellule (même ligne)
gc.weightx = 1 ; // poids sur la ligne
gl.setConstraints(nomField, gc) ;
saisieNom.add(nomField) ;

gc.gridx = 0 ; gc.gridy = 1 ; // changement de cellule
gc.gridwidth = 2 ; // nombre de cellules horizontales occupées
gc.weighty = 1 ; // poids sur la (les) colonne(s)
gl.setConstraints(helloButton, gc) ;
saisieNom.add(helloButton) ;

this.getContentPane().add(saisieNom) ;

...
```



Gestion des événements

- Fonctionnement sur un mode « publier / souscrire »
  - Des **sources** proposent des canaux d'**événement**s dans lesquels des **écouteurs** peuvent s'enregistrer
  - NB : en anglais écouteur : listener
- Pour pouvoir s'enregistrer dans un canal d'événement, un écouteur doit implémenter l'interface correspondante
  - Notion de contrat entre la source et l'écouteur
    - La source s'engage à transmettre l'événement à (appeler une méthode spécifique sur) l'écouteur.
    - L'écouter se charge de traiter cet événement au sein de la méthode appelée
- Fonctionnement
  - Chaque source gère une liste d'écouteurs (qui peut être vide)
  - Lors d'un événement, la méthode correspondante (à l'événement) est appelée sur tous les écouteurs de la liste

Exemple : action sur un JButton

OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP — Java™ Platform Standard Ed. 8

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

javax.swing

Class JButton

Method Summary

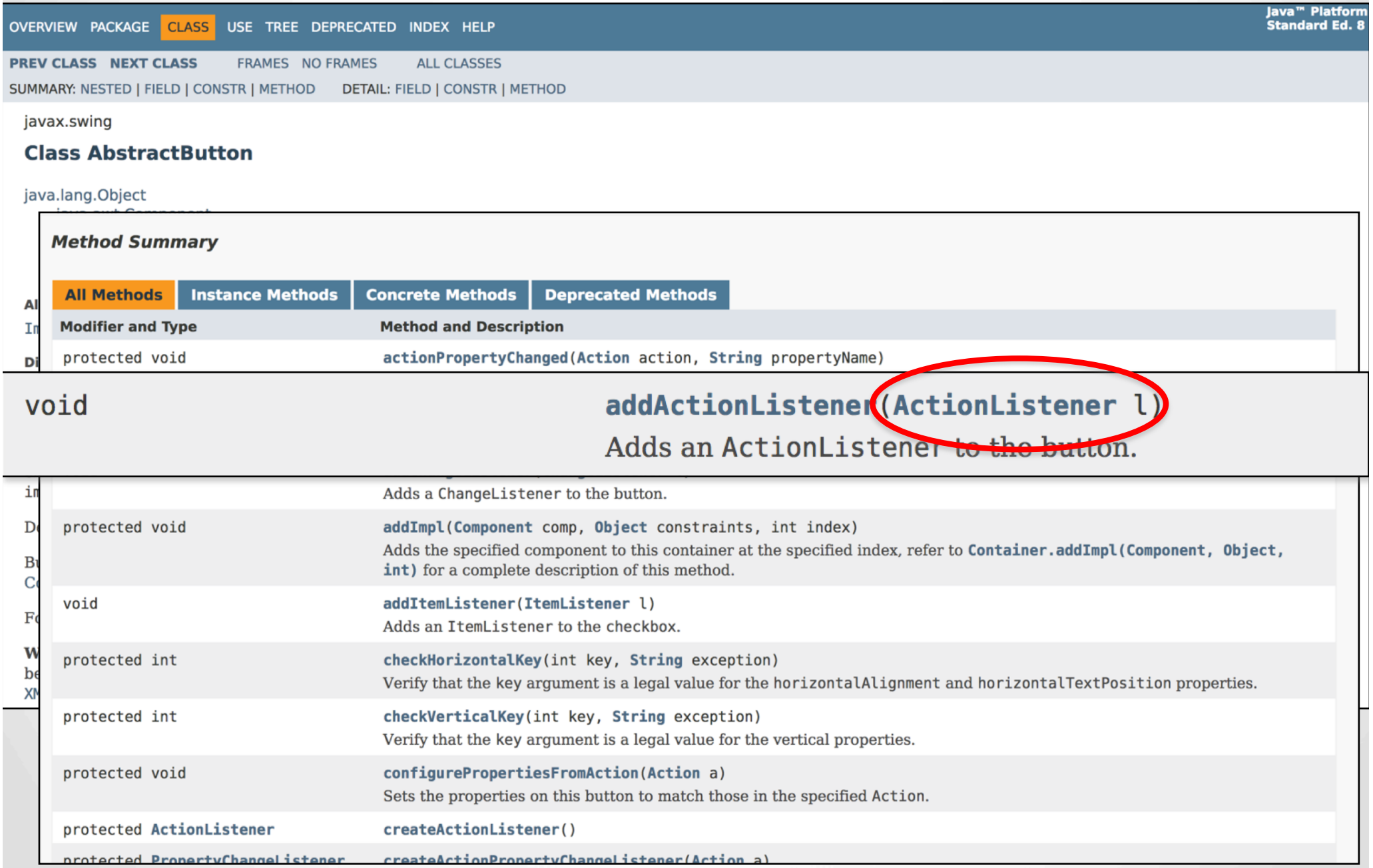
All Methods | Instance Methods | Concrete Methods

Modifier and Type	Method and Description
AccessibleContext	**getAccessibleContext**() Gets the AccessibleContext associated with this JButton.
String	**getUIClassID**() Returns a string that specifies the name of the L&F class that renders this component.
boolean	**isDefaultButton**() Gets the value of the defaultButton property, which if true means that this button is the current default button for its JRootPane.
boolean	**isDefaultCapable**() Gets the value of the defaultCapable property.
protected String	**paramString**() Returns a string representation of this JButton.
void	**removeNotify**() Overrides JComponent.removeNotify to check if this button is currently set as the default button on the RootPane, and if so, sets the RootPane's default button to null to ensure the RootPane doesn't hold onto an invalid button reference.
void	**setDefaultCapable**(boolean defaultCapable) Sets the defaultCapable property, which determines whether this button can be made the default button for its root pane.
void	**updateUI**() Resets the UI property to a value from the current look and feel.

Methods inherited from class javax.swing.AbstractButton

actionPropertyChanged, addActionListener, addChangeListener, addImpl, addItemListener, checkHorizontalKey, checkVerticalKey, configurePropertiesFromAction, createActionListener, createActionPropertyChangeListener, createChangeListener, createItemListener, doClick, doClick, fireActionPerformed, fireItemStateChanged, fireStateChanged, getAction, getActionCommand, getActionListeners, getChangeListeners,

Exemple : action sur un JButton



OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP — Java™ Platform Standard Ed. 8

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

javax.swing

Class AbstractButton

java.lang.Object

Method Summary

All Methods | Instance Methods | Concrete Methods | Deprecated Methods

Modifier and Type	Method and Description
protected void	actionPropertyChanged(Action action, String propertyName)
void	addActionListener(ActionListener l) Adds an ActionListener to the button.
	Adds a ChangeListener to the button.
protected void	addImpl(Component comp, Object constraints, int index) Adds the specified component to this container at the specified index, refer to Container.addImpl(Component, Object, int) for a complete description of this method.
void	addItemListener(ItemListener l) Adds an ItemListener to the checkbox.
protected int	checkHorizontalKey(int key, String exception) Verify that the key argument is a legal value for the horizontalAlignment and horizontalTextPosition properties.
protected int	checkVerticalKey(int key, String exception) Verify that the key argument is a legal value for the vertical properties.
protected void	configurePropertiesFromAction(Action a) Sets the properties on this button to match those in the specified Action.
protected ActionListener	createActionListener()
protected PropertyChangeListener	createActionPropertyChangeListener(Action a)

Exemple : action sur un JButton

OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP — Java™ Platform Standard Ed. 8

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

java.awt.event

Interface ActionListener

```
public interface ActionListener
extends EventListener
```

The listener interface for receiving action events. The class that is interested in processing an action event implements this interface, and the object created with that class is registered with a component, using the component's `addActionListener` method. When the action event occurs, that object's `actionPerformed` method is invoked.

Since:

1.1

See Also:

`ActionEvent`, How to Write an Action Listener

Method Summary

All Methods | Instance Methods | Abstract Methods

Modifier and Type	Method and Description
void	**actionPerformed**(ActionEvent e) Invoked when an action occurs.

Method Detail

actionPerformed

```
void actionPerformed(ActionEvent e)
```

Invoked when an action occurs.

Exemple : action sur un JButton

- Le **JButton** est la source d'événements de type **ActionEvent**
  - … car le **JButton** propose la méthode :
    public void addActionListener(ActionListener l)
  - Un écouteur (intéressé par ce type d'événement) doit donc implémenter l'interface **ActionListener**
  - … ce qui signifie implémenter toutes les méthodes définies dans **ActionListener**
- L'interface **ActionListener** possède une seule méthode :
  void actionPerformed(ActionEvent e)
  - L'écouteur doit implémenter cette méthode. Elle sera appelée par le bouton lorsque l'utilisateur l'aura utilisé.
  - Le corps de la méthode correspond au traitement de cet événement dans l'application.

Différentes Implémentations

- Sous forme de classe anonyme :

```
JButton helloButton = new JButton("Dire bonjour") ;
helloButton.addActionListener(new ActionListener() { // implémentation en classe anonyme
        @Override
        public void actionPerformed(ActionEvent e) { // traitement spécifique à l'application
                System.out.println("Traitement de l'événement : " + e);
                System.out.println("Source : " + e.getSource());
        }
}) ;
```

- Dans une classe dédiée :

```
class TraitementBoutonBonjour implements ActionListener{
        @Override
        public void actionPerformed(ActionEvent e) {
                System.out.println("Traitement de l'événement : " + e);
                System.out.println("Source : " + e.getSource());
        }
}
....
helloButton.addActionListener(new TraitementBoutonBonjour()) ;
```

- Au niveau de l'application elle-même :

```
public class GestionnaireContacts extends JFrame implements ActionListener {
    ...
    @Override
    public void actionPerformed(ActionEvent e) {
            System.out.println("Traitement de l'événement : " + e);
            System.out.println("Source : " + e.getSource());
    }
    ...
            helloButton.addActionListener(this) ; // (mis par exemple dans le constructeur)
```

Exemple avec une interface plus complexe : **WindowListener**

- Exemple / problème / solution :
  - on veut arrêter l'application lorsque l'utilisateur ferme la fenêtre principale, et sans utiliser **setDefaultCloseOperation(*EXIT_ON_CLOSE)***
  - Pour écouter les événements de la fenêtre, **JFrame** (qui hérite de **Window**) propose la méthode **addWindowListener(WindowListener l)**
  - On enregistre donc un objet qui implémente **WindowListener** dans les écouteurs de la **JFrame** (NB : l'action qui nous intéresse appelle **windowClosing(WindowEvent e)**)

- Implémentation (dans **GestionnaireContacts**) :

```
...
this.addWindowListener(new WindowListener() { // ajout  d'un écouteur anonyme à la JFrame
        public void windowActivated(WindowEvent e) {} // implémentation vide
        public void windowClosed(WindowEvent e) {} // implémentation vide
        public void windowClosing(WindowEvent e) { // LA méthode qui nous intéresse !
                System.out.println("Fermeture de la fenêtre !");
                System.exit(0) ; // arret de l'application
        }
        public void windowDeactivated(WindowEvent e) {} // implémentation vide
        public void windowDeiconified(WindowEvent e) {} // implémentation vide
        public void windowIconified(WindowEvent e) {} // implémentation vide
        public void windowOpened(WindowEvent e) {} // implémentation vide
}) ;
```

Simplification : la surcharge d'Adapter

- Un Adapter :
  - est une **classe** de l'API Java
  - implémente de manière vide toutes les méthodes d'**interface**s complexes
  - S'utilise en créant une sous-classe qui ne surcharge que la ou les méthodes utiles dans le cadre de l'application

- Exemple : **WindowAdapter**
  - la classe **WindowAdapter** fournit une implémentation vide de l'interface **WindowListener**
  - Utilisation :

```
// instanciation et ajout d'une sous-classe anonyme de WindowAdapter
this.addWindowListener(new WindowAdapter() {

        public void windowClosing(WindowEvent e) {
                System.out.println("Fermeture de la fenêtre !");
                System.exit(0) ; // arret de l'application
        }

}) ;
```

Composants Swing

(http://docs.oracle.com/javase/tutorial/ui/features/components.html)

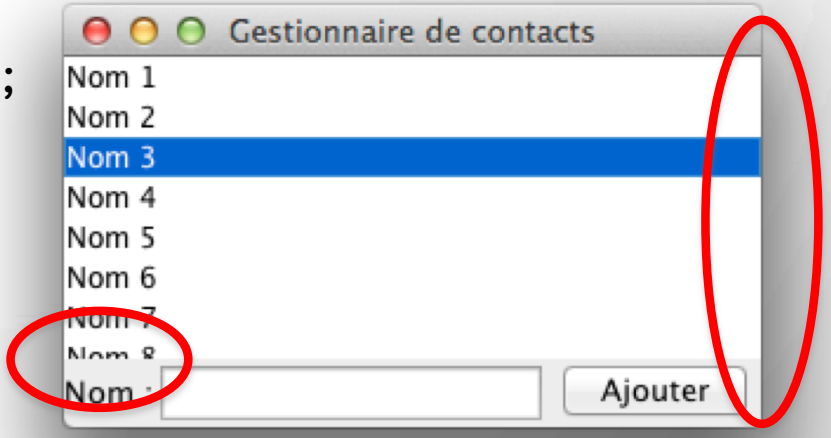
- JApplet
- JDialog
- JFrame
- JLayeredPane
- Jpanel
- JScrollPane
- JSplitPane
- JTabbedPane
- JInternalFrame
- JButton
- JLabel
- JCheckBox
- JComboBox
- JTextField
- JPasswordField
- JRadioButton
- JSlider
- JSpinner
- JTextArea
- JProgressBar
- Jseparator
- JList
- JTable
- JMenuBar
- JMenu
- JColorChooser
- JEditorPane
- JFileChooser
- JToolTip
- JToolBar

Exemple : **JList**

- **JList** permet de créer une représentation graphique (sous forme d'une « simple » liste) d'un ensemble de données.
- **JList** founit la représentation mais ne « contient » pas les données à afficher
  - **JList** est lié à un ensemble de données que l'on appelle « modèle »
  - (NB : le fonctionnelemnt est identique avec **JTable**)
  - … il s'agit d'une approche de type MVC (Modèle Vue Contrôleur)

- Exemple de création d'une liste de 20 éléments :

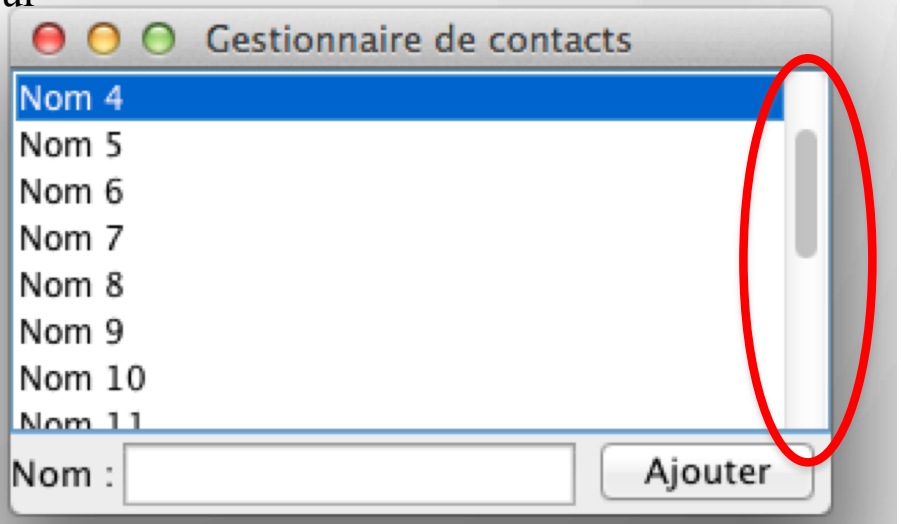
```
…
DefaultListModel model = new DefaultListModel() ;
for (int i=1 ; i <= 20 ; i++) {
    model.addElement("Nom " + i) ;
}
JList list = new JList(model) ;
…
```



Le JScrollPane

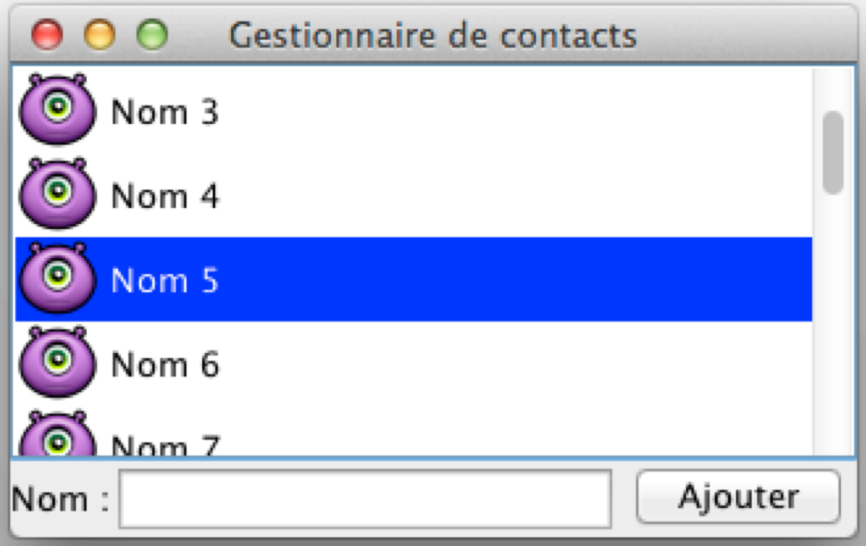
- Le **JScrollPane** permet de gérer le problème de composants trop grands pour l'espace d'affichage
- Ajoute des barres de défilement
  - Horizontale
  - Verticale
  - Les 2 !
- Utilisation
  - On n'ajoute plus de composant au conteneur, mais on l'intègre dans un **JScrollPane**
  - C'est le **JScrollPane** qu'on ajoute au conteneur
  - Exemple :

```
…
JList list = new JList(model) ;
JScrollPane scroll = new JScrollPane(list) ;
gc.gridx = 0 ;
gc.weightx = 1  ; gc.weighty = 1 ;
gc.gridwidth = 3 ;
gl.setConstraints(scroll, gc) ;
contactsPanel.add(scroll) ;
```



Les **CellRenderer**

- JList affiche les données au moyen d'un CellRenderer
- Par défaut, le CellRenderer de JList utilise la méthode toString() pour créer la représentation d'une donnée
- Il est possible de customiser cette représentation en créant son propre CellRenderer
- Pour JList, on implémente une sous-classe de ListCellRenderer
- Exemple en se basant sur un JLabel :



```
class AlienRenderer extends JLabel implements ListCellRenderer{

   public AlienRenderer() {
                // création d'une ImageIcon à partir d'un fichier sur le disque
                URL anUrl = ClassLoader.getSystemResource("test/1352456448_Alien.png");
                ImageIcon icon = new ImageIcon(anUrl);

                // on retaille l'image pour qu'elle ne soit pas trop grande
                int iconSize = 32 ;
                setIcon(new ImageIcon(icon.getImage().getScaledInstance(iconSize, iconSize,
Image.SCALE_SMOOTH))) ;

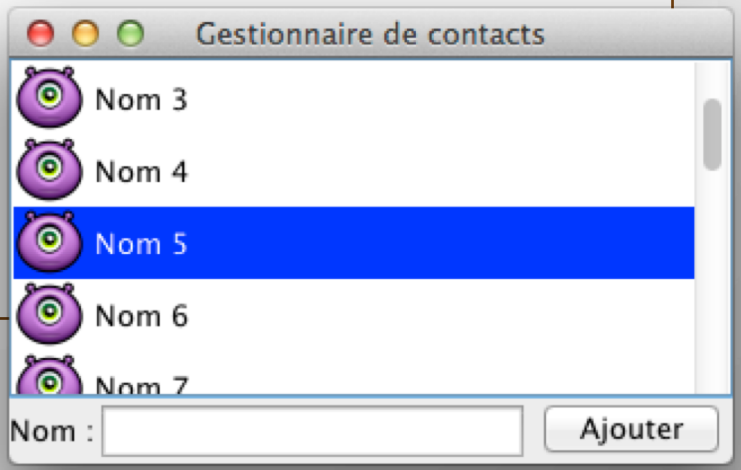
                setOpaque(true) ; // … le composant est opaque
   }

   @Override
   public Component getListCellRendererComponent(JList list, Object value,
                                                 int index, boolean isSelected, boolean cellHasFocus) {

                // Description de la représentation de l'objet en fonction de son "état" dans la liste
                setText(value.toString());
                setBackground(isSelected ? Color.blue : Color.white);
                setForeground(isSelected ? Color.white : Color.black);

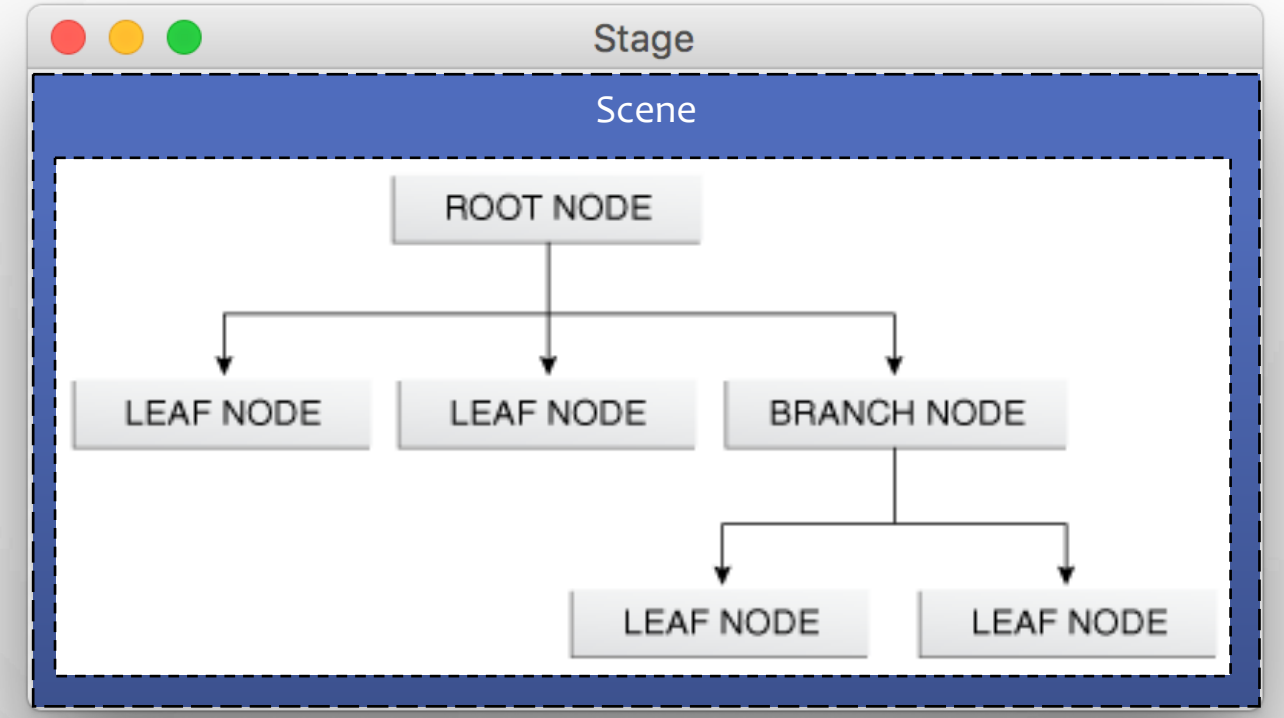
                return this ;
   }
}
```

```
JList list = new JList(model) ;
list.setCellRenderer(new AlienRenderer()) ;
```



JavaFX

- Bibliothèque graphique standard officielle depuis Java 8
- De très nombreuses possibilités :
  - **Demo applications**
  - **Demo Widgets**
- Architecture hiérarchique : Stage(s) / Scene(s) / Nodes



JavaFX - HelloWorld

```
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

public class HelloWorld extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception {

        primaryStage.setTitle("Hello World");

        Button btn = new Button() ;
        btn.setText("Say 'Hello World'");
        btn.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) { System.out.println("Hello World"); }
        });

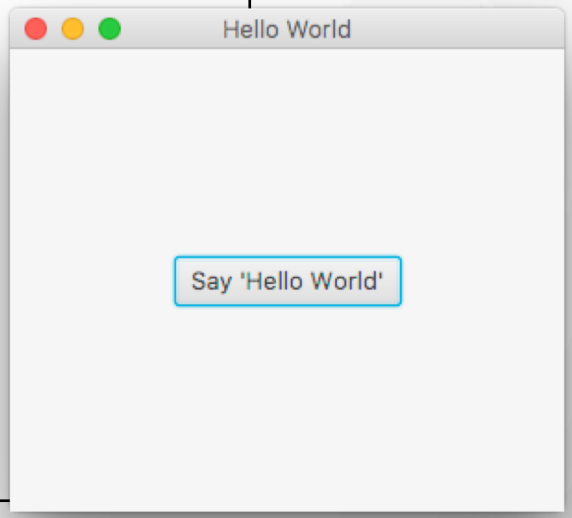
        StackPane root = new StackPane() ;
        root.getChildren().add(btn) ;

        primaryStage.setScene(new Scene(root, 300, 250));
        primaryStage.show();

    }

    public static void main(String[] args){
        launch(args);
    }

}
```



http://docs.oracle.com/javafx/2/get_started/hello_world.htm

JavaFX - HelloWorld V2

```
public class HelloWorldV2 extends Application {

    Button btn ;

    @Override
    public void start(Stage primaryStage) throws Exception {
        buildUI(primaryStage) ;
        addListeners();
        primaryStage.show();
    }

    private void buildUI(Stage stage){
        stage.setTitle("Hello World");

        btn = new Button() ;
        btn.setText("Say 'Hello World'");

        StackPane root = new StackPane() ;
        root.getChildren().add(btn) ;

        stage.setScene(new Scene(root, 300, 250));
    }

    private void addListeners(){
        btn.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) { System.out.println("Hello World"); }
        });
    }

    public static void main(String[] args) { launch(args); }

}
```

JavaFX - LoginForm (1)

```
public class LoginForm extends Application {

    Button btn ;
    Text actionTarget ;

    @Override
    public void start(Stage primaryStage) throws Exception {
        buildUI(primaryStage);
        addListeners();
        primaryStage.show();
    }

    private void buildUI(Stage stage){...}

    private void addListeners(){...}

    public static void main(String[] args){
        launch(args);
    }

}
```

JavaFX - LoginForm (2)

```
private void buildUI(Stage stage){
    stage.setTitle("JavaFX Welcome");

    GridPane grid = new GridPane() ;
    grid.setAlignment(Pos.CENTER);
    grid.setHgap(10);
    grid.setVgap(10);
    grid.setPadding(new Insets(25,25,25,25));

    Text sceneTitle = new Text("Welcome") ;
    sceneTitle.setFont(Font.font("Tahoma", FontWeight.NORMAL, 20));
    grid.add(sceneTitle, 0, 0, 2, 1);

    Label userName = new Label("User Name:") ;
    grid.add(userName, 0, 1);

    TextField userTextField = new TextField() ;
    grid.add(userTextField, 1,1);

    Label pwd = new Label("Password:") ;
    grid.add(pwd, 0, 2);

    PasswordField pwdField = new PasswordField() ;
    grid.add(pwdField, 1, 2);

    HBox hbBtn = new HBox(10) ;
    hbBtn.setAlignment(Pos.BOTTOM_RIGHT);
    btn = new Button("Sign in") ;
    hbBtn.getChildren().add(btn) ;
    grid.add(hbBtn, 1, 4);

    actionTarget = new Text() ;
    grid.add(actionTarget, 1,6);

    Scene scene = new Scene(grid, 300, 275) ;
    stage.setScene(scene);
}
```




```
private void addListeners(){
    btn.setOnAction(new EventHandler<ActionEvent>() {
        @Override
        public void handle(ActionEvent event) {
            actionTarget.setFill(Color.FIREBRICK);
            actionTarget.setText("Sign in button pressed");
        }
    });
    btn.addEventHandler(MouseEvent.MOUSE_ENTERED, (event)->{
        System.out.println("!!! MOUSE IN THE BUTTON : " + event);
    });
}
```

JavaFX - LoginForm + CSS

```
private void buildUI(Stage stage){
    stage.setTitle("JavaFX Welcome");

    GridPane grid = new GridPane() ;
    grid.setAlignment(Pos.CENTER);
    grid.setHgap(10);
    grid.setVgap(10);
    grid.setPadding(new Insets(25,25,25,25));

    Text sceneTitle = new Text("Welcome") ;
    sceneTitle.setId("welcome-text");
    grid.add(sceneTitle, 0, 0, 2, 1);

    Label userName = new Label("User Name:") ;
    grid.add(userName, 0, 1);

    TextField userTextField = new TextField() ;
    grid.add(userTextField, 1,1);

    Label pwd = new Label("Password:") ;
    grid.add(pwd, 0, 2);

    PasswordField pwdField = new PasswordField() ;
    grid.add(pwdField, 1, 2);

    btn = new Button("Sign in") ;
    HBox hbBtn = new HBox(10) ;
    hbBtn.setAlignment(Pos.BOTTOM_RIGHT);
    hbBtn.getChildren().add(btn) ;
    grid.add(hbBtn, 1, 4);

    actionTarget = new Text() ;
    actionTarget.setId("action-target");
    grid.add(actionTarget, 1,6);

    Scene scene = new Scene(grid, 300, 275) ;
    stage.setScene(scene);
    scene.getStylesheets().add(LoginFormCSS.class.getResource("css/LoginFormCSS.css").toExternalForm());
}
```

./CSS/LoginFormCSS.css

```
.button {
    -fx-text-fill: white;
    -fx-font-family: "Arial Narrow";
    -fx-font-weight: bold;
    -fx-background-color: linear-gradient(#61a2b1, #2A5058);
    -fx-effect: dropshadow( three-pass-box , rgba(0,0,0,0.6), 5, 0.0, 0, 1 );
}
.button:hover {
    -fx-background-color: linear-gradient(#2A5058, #61a2b1);
}
.root {
    -fx-background-image: url("../images/background.jpg");
}
.label {
    -fx-font-size: 12px;
    -fx-font-family: "Tahoma";
    -fx-font-weight: bold;
    -fx-text-fill: #333333;
    -fx-effect: dropshadow( gaussian , rgba(255,255,255,0.5), 0,0,0,1 );
}
#welcome-text {
    -fx-font-size: 32px;
    -fx-font-family: "Arial Black";
    -fx-fill: #818181;
    -fx-effect: innershadow( three-pass-box, rgba(0,0,0,0.7), 6, 0.0, 0, 2 );
}
#action-target {
    -fx-fill: FIREBRICK;
    -fx-font-weight: bold;
}
```



JavaFX - LoginForm - FXML (1)

```
package fxmlexample;
import ...

public class FXMLExample extends Application {
    @Override
    public void start(Stage stage) throws Exception{
        Parent root = FXMLLoader
                .load(getClass()
                .getResource("fxml_example.fxml"));
        Scene scene = new Scene(root, 300, 275);
        stage.setTitle("FXML Welcome");
        stage.setScene(scene);
        stage.show();

    }

    public static void main(String[] args){
        launch(args);
    }

}
```

```
package fxmlexample;

import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.text.Text;

public class FXMLExampleController {

    @FXML
    private Text actionTarget;

    @FXML protected void handleSubmitButtonAction(ActionEvent event) {
        actionTarget.setText("Sign in button pressed");
    }
```

```
<?xml version="1.0" encoding="UTF-8"?>

<?import java.net.*?>
<?import javafx.geometry.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>
<?import javafx.scene.text.*?>

<GridPane fx:controller="fxmlexample.FXMLExampleController"
          xmlns:fx="http://javafx.com/fxml"
          alignment="center" hgap="10" vgap="10"
          styleClass="root"
>
    <padding>
        <Insets top="25" right="25" bottom="10" left="25"/>
    </padding>

    <Text id="welcome-text"
          text="Welcome"
          GridPane.columnIndex="0" GridPane.rowIndex="0"
          GridPane.columnSpan="2"/>

    <Label text="User Name:" GridPane.columnIndex="0" GridPane.rowIndex="1"/>

    <TextField GridPane.columnIndex="1" GridPane.rowIndex="1"/>

    <Label text="Password:" GridPane.columnIndex="0" GridPane.rowIndex="2"/>

    <PasswordField fx:id="passwordField" GridPane.columnIndex="1" GridPane.rowIndex="2"/>

    <HBox spacing="10" alignment="bottom_right"
          GridPane.columnIndex="1" GridPane.rowIndex="4">
        <Button text="Sign In" onAction="#handleSubmitButtonAction"/>
    </HBox>

    <Text fx:id="actionTarget" GridPane.columnIndex="1" GridPane.rowIndex="6"/>

    <stylesheets>
        <URL value="@Login.css"/>
    </stylesheets>

</GridPane>
```

JavaFX - LoginForm - FXML (2)

```
<?xml version="1.0" encoding="UTF-8"?>

<?import java.net.*?>
<?import javafx.geometry.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>
<?import javafx.scene.text.*?>

<GridPane fx:controller="fxmlexample.FXMLExampleController"
          xmlns:fx="http://javafx.com/fxml"
          alignment="center" hgap="10" vgap="10"
          styleClass="root"
>
    <padding>
        <Insets top="25" right="25" bottom="10" left="25"/>
    </padding>

    <Text id="welcome-text"
          text="Welcome"
          GridPane.columnIndex="0" GridPane.rowIndex="0"
          GridPane.columnSpan="2"/>

    <Label text="User Name:" GridPane.columnIndex="0" GridPane.rowIndex="1

    <TextField GridPane.columnIndex="1" GridPane.rowIndex="1"/>

    <Label text="Password:" GridPane.columnIndex="0" GridPane.rowIndex="2"

    <PasswordField fx:id="passwordField" GridPane.columnIndex="1" GridPane

    <HBox spacing="10" alignment="bottom_right"
          GridPane.columnIndex="1" GridPane.rowIndex="4">
        <Button text="Sign In" onAction="#handleSubmitButtonAction"/>
    </HBox>

    <Text fx:id="actionTarget" GridPane.columnIndex="1" GridPane.rowIndex=

    <stylesheets>
        <URL value="@Login.css"/>
    </stylesheets>

</GridPane>
```

./Login.css

```
.button {
    -fx-text-fill: white;
    -fx-font-family: "Arial Narrow";
    -fx-font-weight: bold;
    -fx-background-color: linear-gradient(#61a2b1, #2A5058);
    -fx-effect: dropshadow( three-pass-box ,
                rgba(0,0,0,0.6), 5, 0.0, 0, 1 );
}
.button:hover {
    -fx-background-color: linear-gradient(#2A5058, #61a2b1);
}
.root {
    -fx-background-image: url("../images/background.jpg");
}
.label {
    -fx-font-size: 12px;
    -fx-font-family: "Tahoma";
    -fx-font-weight: bold;
    -fx-text-fill: #333333;
    -fx-effect: dropshadow( gaussian ,
                rgba(255,255,255,0.5), 0,0,0,1 );
}
#welcome-text {
    -fx-font-size: 32px;
    -fx-font-family: "Arial Black";
    -fx-fill: #818181;
    -fx-effect: innershadow( three-pass-box,
                rgba(0,0,0,0.7), 6, 0.0, 0, 2 );
}
#actionTarget {
    -fx-fill: FIREBRICK;
    -fx-font-weight: bold;
}
```