SHAKE THE FUTURE

CENTRALE NANTES

Programmation Orientée Objet - Annexe 1 Interface Graphiques avec SWING

Jean-Marie Normand
jean-marie.normand@ec-nantes.fr
Bâtiment E - Bureau 211

CENTRALE NANTES

Plan du cours I

CENTRALE NANTES

Plan

Interface graphique

Qu'est ce qu'une interface graphique ?

- GUI (*Graphical User Interface* en anglais)
- Façade visuelle qui lie une application avec l'extérieur et qui facilite l'interaction avec le(s) utilisateur(s) de celle-ci
- Une interface graphique nécessite d'être programmée et possède trois niveaux principaux :
  - les `Widgets` : quels composants graphiques ?
  - les `Layouts` : comment disposer les widgets les uns par rapport aux autres ?
  - les évènements (`Events`) : comment gérer les actions de l'utilisateur

Comment programmer une interface graphique ?

Soit :

- on part de “zéro” :
  - tout le code est écrit par le programmeur
  - nécessite une très bonne compréhension de l’API graphique de Java
- on s’appuie sur des IDEs spécifiques :
  - le code est partiellement généré automatiquement (approche “WYSIWYG” : *What You See Is What You Get*)
  - permet de se concentrer sur le design et l’ergonomie des interfaces
  - peut être un peu brouillon et difficile à “customiser” (le code étant généré automatiquement à la volée, on ne peut généralement pas le modifier)

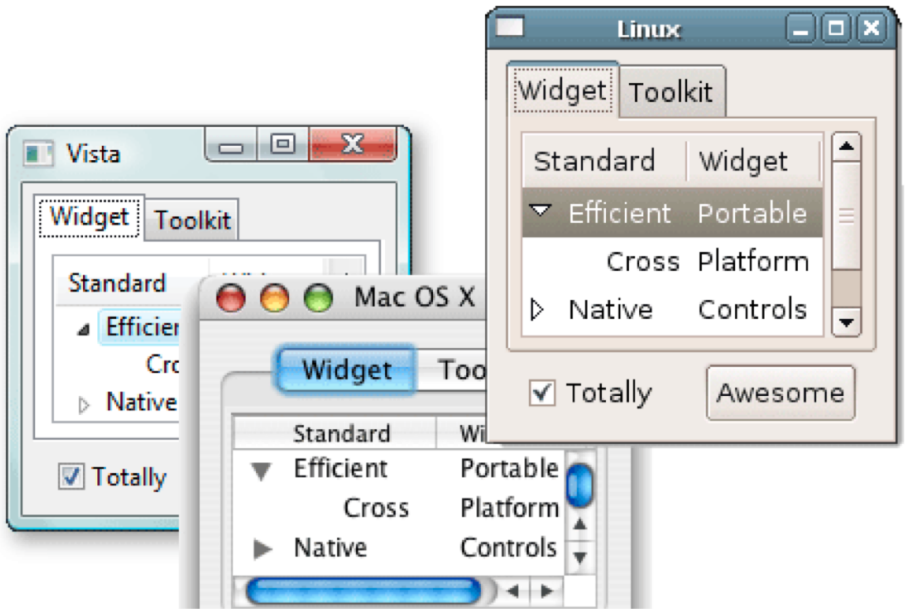
CENTRALE NANTES

GUI en Java

- On va faire appel à l'API Java pour développer des interfaces graphiques
- Paquetages spécifiques à importer :
  - `import java.awt.*;`
  - `import java.awt.event.*;`
  - `import javax.swing.*;`
- On va utiliser les deux paquetages `AWT` et `Swing`
  - AWT (*Abstract Window Toolkit*) : première bibliothèque graphique de Java (dès Java 1.0) permettant de créer des interfaces graphiques. Comme Java est multi-plateformes, AWT se limite aux composants disponibles sur tous les systèmes d'exploitations, ses possibilités sont donc limitées.
  - Swing : depuis Java 1.2 la bibliothèque de fenêtrage officielle de Java est SWING. Toutefois AWT sert encore de fondement à SWING, dans la mesure où de nombreuses classes de SWING héritent de classes de AWT
  - Principale différence : AWT utilise les composants natifs de la plateforme (**composants "lourds"**) alors que Swing utilise (à quelques exceptions près) des composants écrits en Java (**composants "légers"**)

Look'n'Feel adaptatif

- Rappel : Java est indépendant de la plateforme cible (i.e. du système d'exploitation ou OS)
- Conséquence : l'apparence des GUI s'adapte au gestionnaire du système de fenêtrage de la plateforme sur laquelle le programme est exécuté



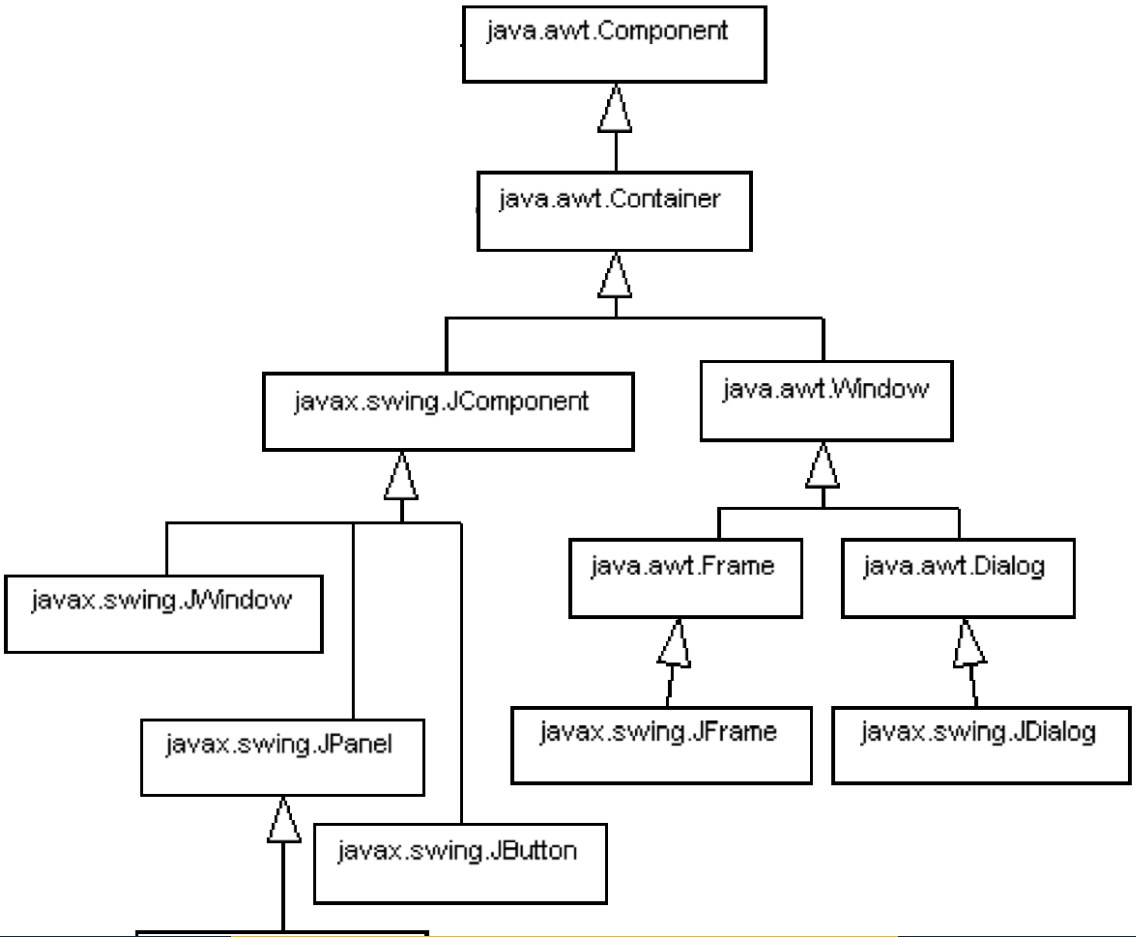
Swing

- Les composants graphiques (aussi appelés *widgets*) de Swing commencent le plus souvent par un `J`
- Tous ces composants héritent du composant graphique `JComponent`
- `JComponent` hérite du composant graphique AWT `Container` qui permet à un composant d'en contenir d'autres
- Principaux *widgets* de Swing :

- `JButton` (bouton)
- `JCheckBox` (case à cocher)
- `JRadioButton` (boutons radios)
- `JComboBox` (liste déroulante)
- `JLabel` (étiquette de texte)
- `JTextField` (champ de saisie)
- `JTable` (tableau)
- `JTree` (arborescence)
- `JMenuBar, JMenuItem, JMenu` (barre de menu et choix dans les menus)
- `JFrame` (fenêtre)
- `JOptionPane` (boîte de dialogue)
- etc.

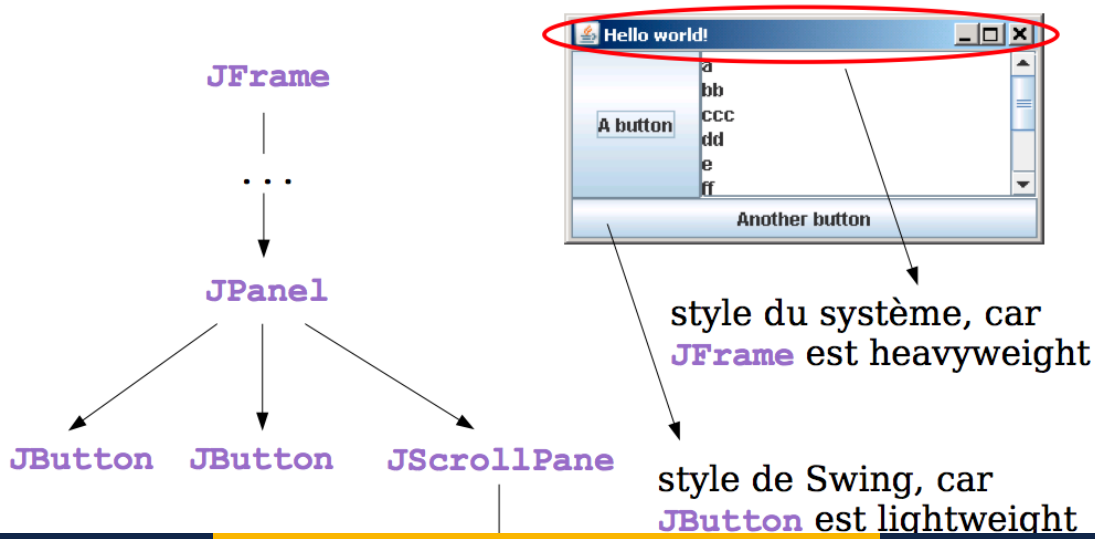
Allez voir la documentation !

Hiérarchie d'héritage des composants graphiques Swing/AWT



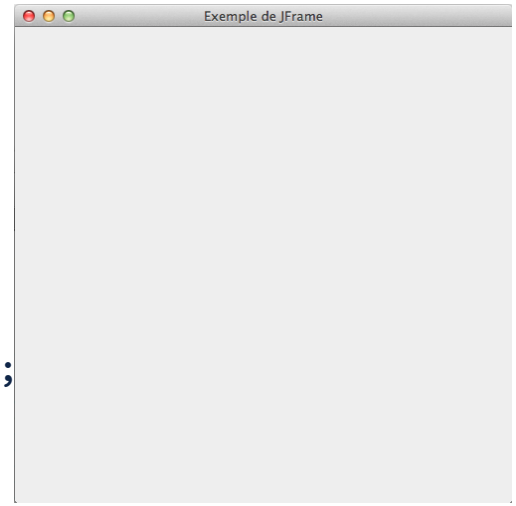
Les GUI sont arborescentes

- une GUI Swing possède une représentation arborescente qui a comme racine d'un composant lourd (*heavyweight*) du système :
  - `JFrame` : fenêtre "classique"
  - `JWindow` : fenêtre non décorée (sans barre de nom) avec gestion partielle des évènements
  - `JDialog` : boîte de dialogue
- la racine contient des objets Swing (composants légers, *lightweight*)



Affichage fenêtre vide JFrame

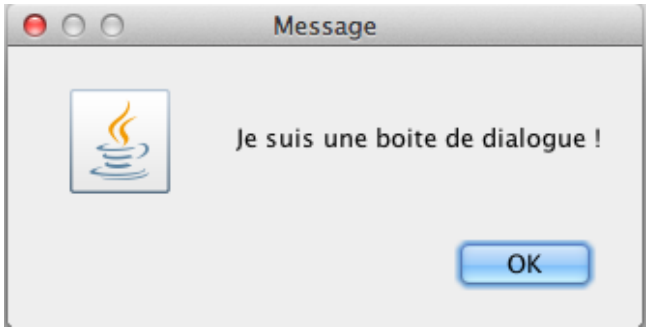
```
import javax.swing.*;

public class ExempleJFrame {
    public static void main(String[] args) {
      JFrame maFrame = new JFrame();
      maFrame.setTitle("Exemple de JFrame");
      maFrame.setSize(500,500);
      maFrame.setVisible(true);

      maFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```



Affichage boîtes de dialogue JOptionPane

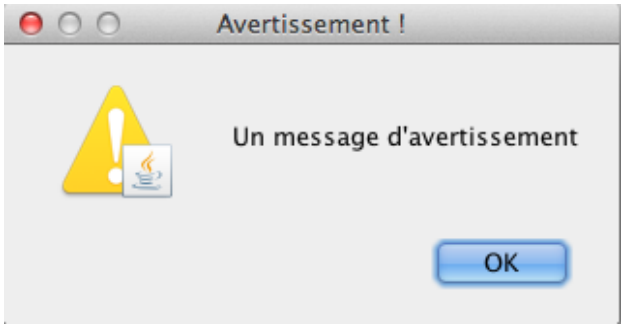
Boîte de dialogue simple

```
// a la suite du main de ExempleJFrame
JOptionPane.showMessageDialog(maFrame,
    "Je suis une boite de dialogue !");
```



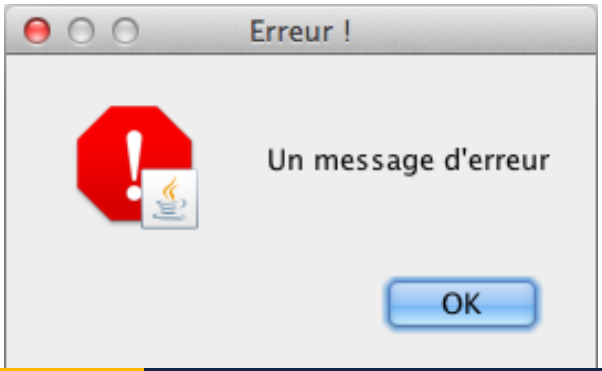
Boîte de dialogue d'avertissement

```
// a la suite du main de ExempleJFrame
JOptionPane.showMessageDialog(maFrame,
"Un message d'avertissement",
    "Avertissement !",
JOptionPane.WARNING_MESSAGE);
```



Boîte de dialogue d'erreur

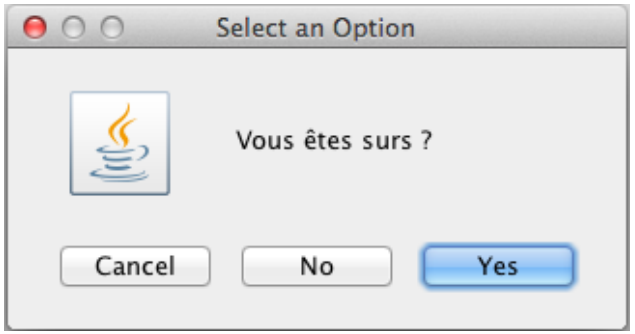
```
// a la suite du main de ExempleJFrame
JOptionPane.showMessageDialog(maFrame,
"Un message d'erreur", "Erreur !",
JOptionPane.ERROR_MESSAGE);
```



Affichage boîtes de dialogue II JOptionPane

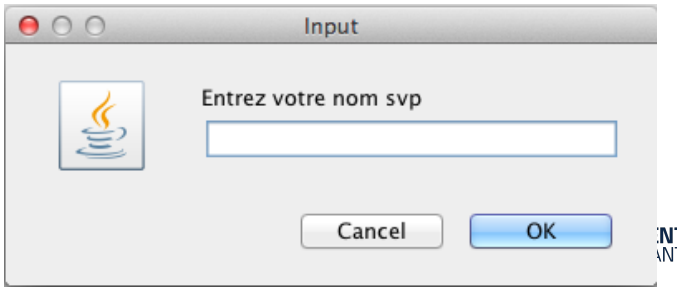
Boîte de dialogue de confirmation

```
// a la suite du main de ExempleJFrame
int confirm =
     JOptionPane.showConfirmDialog(maFrame,
     "Vous etes surs ?");
if(confirm == JOptionPane.YES_OPTION) {
  // on a dit oui
}
else if(confirm == JOptionPane.NO_OPTION) {
  // on a dit non
}
else if(confirm == JOptionPane.CANCEL_OPTION){
  // on a annule
}
```

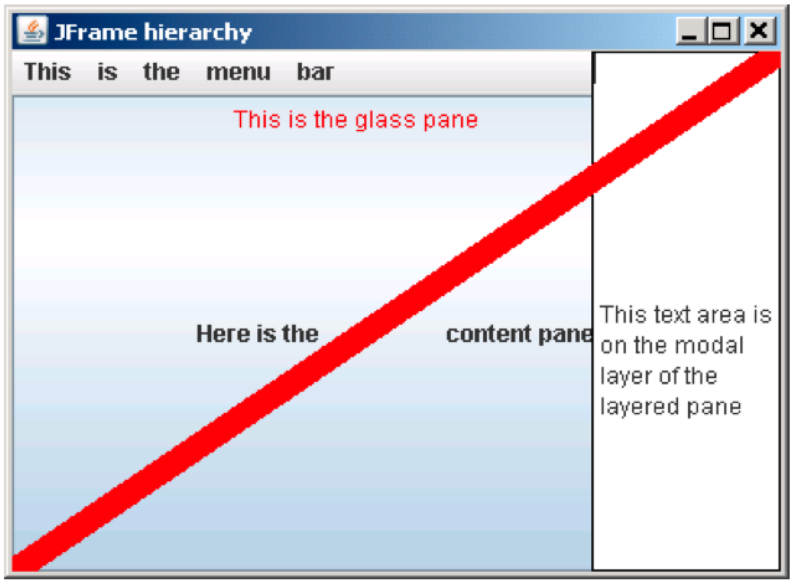


Boîte de dialogue demande d'informations

```
// a la suite du main de ExempleJFrame
String str =
     JOptionPane.showInputDialog(maFrame,"Entrez
```



Exemple : structure JFrame



JFrame

JRootPane (un seul par **JFrame**)

glassPane (un **JPanel** transparent par-dessus tout l'intérieur de la **JFrame**)

JLayeredPane

JMenuBar

contentPane (un **JPanel**)

Le contentPane

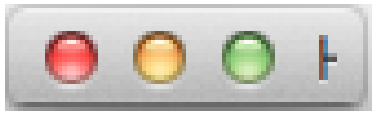
- par défaut c'est un JPanel (un conteneur générique) opaque
- peut être récupéré ou modifié : getContentPane() ou setContentPane(...)

Exemple manipulation contentPane

```
public static void main(String[] args) {
  JFrame f = new JFrame("Hello World!");
  f.getContentPane().add(new JButton("Un bouton"),BorderLayout.WEST);
  f.getContentPane().add(new JButton("Autre bouton",BorderLayout.SOUTH);
  String[] s = {"a","bb","ccc","dd","e","ff","ggg","hhh","i","jj"};
  f.getContentPane().add(new JScrollPane(new JList(s)));
  f.setSize(320,240);
  f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  f.setVisible(true);
}
```

Création d'une `JFrame`

- on doit préciser une taille (`setSize(...);`) sinon :



- **OU BIEN** utiliser la méthode `pack()` qui redimensionnera la fenêtre de manière à ce qu'elle contienne au plus près tous les composants qu'elle contient !!
- on doit la rendre visible (`setVisible(true);`)
- on évite de "toucher à la fenêtre" (i.e. d'appeler des méthodes modifiant l'objet représentant la fenêtre) après l'appel à `setVisible(true);` sinon on peut avoir des surprises

Fermeture d'une `JFrame`

- On doit définir le mode de fermeture souhaité (i.e. lorsque l'on clique sur le bouton de fermeture de la fenêtre) via l'appel à la méthode `setDefaultCloseOperation(...);`
  - `DO_NOTHING_ON_CLOSE` : comme son nom l'indique
  - `HIDE_ON_CLOSE` : rend la fenêtre invisible (mode par défaut)
  - `DISPOSE_ON_CLOSE` : cache la fenêtre et libère toutes les ressources systèmes associées ; elles seront ré-allouées si la fenêtre redevient visible
  - `EXIT_ON_CLOSE` : termine le programme (c'est généralement ce que l'on souhaite faire)

CENTRALE NANTES

Propriétés d'une `JFrame` 

- on peut bloquer le redimensionnement de la fenêtre avec `setResizable(true/false);`
- on peut enlever les décorations (barre de titre, boutons de redimensionnement, fermeture, etc.) avec `setUndecorated(true/false);` mais seulement quand la fenêtre n'est pas associée à des ressources système
- attention toutefois : une `JFrame` sans décorations ne pourra plus être redimensionnée

Positionnement des Widgets les uns par rapport aux autres

- Les composants graphiques (`JComponent`) se positionnent dans des panneaux (`JPanel`)
- Une fenêtre est composée d'un panneau : le `contentPane` que l'on remplit avec :
  - `setContentPane(JPanel panel)` : voir plus haut
- La position des différents composants graphiques dans un `JPanel` dépend du système de placement" (appelé **layout**) choisi pour le panneau considéré
- Ce positionnement des différents composants est réalisé par un gestionnaire de placement, ou `LayoutManager`
- Notez qu'un panneau (`JPanel`) peut contenir d'autres panneaux (`JPanel`)

CENTRALE NANTES

Les gestionnaires de placement I

- `FlowLayout` : gestionnaire par défaut en SWING
  - les composants sont ajoutés à la suite les uns des autres, de gauche à droite et de haut en bas
- `BorderLayout` :
  - le panneau est découpé en 5 zones : NORTH, SOUTH, CENTER, WEST, EAST
  - utilise également 5 constantes de positionnement relatives qui dépendent de l'orientation du `JComponent` (de la gauche vers la droite ou inversement) : PAGE_START, PAGE_END, LINE_START et LINE_END
  - si le composant possède l'orientation par défaut (gauche vers la droite) alors ces constantes de positionnement relatives sont équivalentes (dans l'ordre) à NORTH, SOUTH, WEST et EAST

CENTRALE NANTES

Les gestionnaires de placement II

- lors de l'ajout du composant, on précise la zone où l'on souhaite placer le composant

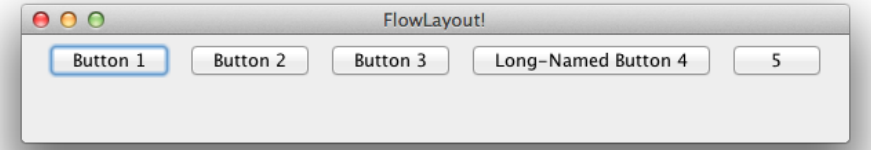
Les gestionnaires de placement II I

- `GridLayout(rows, columns)` :
  - le panneau est découpé en une grille de cellules de `rows` lignes et `columns` colonnes
  - les composants sont ajoutés successivement dans chacune des cellules par ligne en partant du haut (cf. exemple plus loin)
- `GridBagLayout(rows, columns)` :
  - c'est un gestionnaire très puissant mais également complexe à utiliser
  - le panneau est découpé en une grille de cellules de `rows` lignes et `columns` colonnes
  - les composants peuvent être placés dans plusieurs cellules (i.e. sur une même ligne ou colonne)
  - les lignes et les colonnes ne font pas forcément la même hauteur
- plus d'informations et d'exemples sur : `http://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html`

Exemple de FlowLayout

Exemple FlowLayout

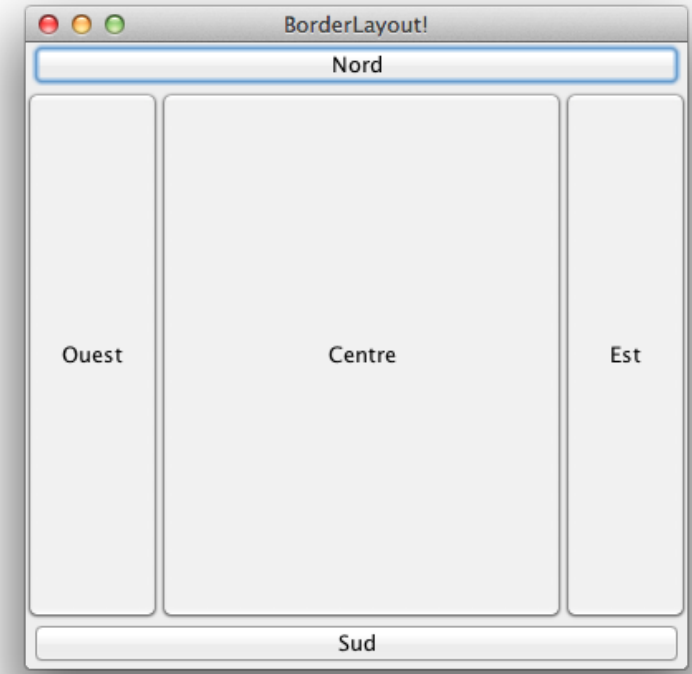
```
public static void main(String[] args) {
  JFrame maFrame = new JFrame("FlowLayout!");

  JPanel panneau = new JPanel();
  FlowLayout gestionnaire = new FlowLayout();
  // applique le gestionnaire de placement au panneau
  panneau.setLayout(gestionnaire);

  // Ajout d'elements au panneau
  panneau.add(new JButton("Button 1"));
  panneau.add(new JButton("Button 2"));
  panneau.add(new JButton("Button 3"));
  panneau.add(new JButton("Long-Named Button 4"));
  panneau.add(new JButton("5"));

  // affecte le panneau a la fenetre
  maFrame.setContentPane(panneau);
  maFrame.pack();
  // rend la fenetre visible
  maFrame.setVisible(true);

      maFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```



Exemple de BorderLayout

Exemple BorderLayout

```
public static void main(String[] args) {
  JFrame maFrame = new JFrame("BorderLayout!");

  JPanel panneau = new JPanel();
  BorderLayout gestionnaire = new BorderLayout();
  // applique le gestionnaire de placement au panneau
  panneau.setLayout(gestionnaire);

  // Ajout d'elements au panneau
  panneau.add(new JButton("Centre"), BorderLayout.CENTER);
  panneau.add(new JButton("Nord"), BorderLayout.NORTH);
  panneau.add(new JButton("Sud"), BorderLayout.SOUTH);
  panneau.add(new JButton("Est"), BorderLayout.EAST);
  panneau.add(new JButton("Ouest"), BorderLayout.WEST);

  // affecte le panneau a la fenetre
  maFrame.setContentPane(panneau);
  maFrame.pack();
  // rend la fenetre visible
  maFrame.setVisible(true);
  maFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
```



Exemple de GridLayout

Exemple GridLayout

```
public static void main(String[] args) {
  JFrame maFrame = new JFrame("GridLayout!");

  JPanel panneau = new JPanel();
  // 3 colonnes et 2 lignes
  GridLayout gestionnaire = new GridLayout(3,2);
  // applique le gestionnaire de placement au panneau
  panneau.setLayout(gestionnaire);

  // Ajout d'elements au panneau
  panneau.add(new JButton("Bouton 1"));
  panneau.add(new JButton("Bouton 2"));
  panneau.add(new JButton("Bouton 3"));
  panneau.add(new JButton("Bouton 4"));
  panneau.add(new JButton("Bouton 5"));
  panneau.add(new JButton("Bouton 6"));

  // affecte le panneau a la fenetre
  maFrame.setContentPane(panneau);
  maFrame.pack();
  // rend la fenetre visible
  maFrame.setVisible(true);
```

Exemple de GridBagLayout

Exemple GridBagLayout

```
public static void main(String[] args) {
  JFrame maFrame = new JFrame("GridBagLayout!");
  JPanel panneau = new JPanel();
  GridBagLayout gestionnaire = new GridBagLayout();
  // applique le gestionnaire de placement au panneau
  panneau.setLayout(gestionnaire);

  JButton button;
  GridBagConstraints c = new GridBagConstraints();
  // natural height, maximum width
  c.fill = GridBagConstraints.HORIZONTAL;
  button = new JButton("Button 1");
  c.weightx = 0.5;
  c.fill = GridBagConstraints.HORIZONTAL;
  c.gridx = 0;
  c.gridy = 0;
  panneau.add(button, c);

  button = new JButton("Button 2");
  c.fill = GridBagConstraints.HORIZONTAL;
  c.weightx = 0.5;
  c.gridx = 1;
```

Exemple de GridBagLayout II

Exemple GridBagLayout

```
button = new JButton("Button 3");
c.fill = GridBagConstraints.HORIZONTAL;
c.weightx = 0.5;
c.gridx = 2;
c.gridy = 0;
panneau.add(button, c);

button = new JButton("Long-Named Button 4");
c.fill = GridBagConstraints.HORIZONTAL;
c.ipady = 40;      //make this component tall
c.weightx = 0.0;
c.gridwidth = 3;
c.gridx = 0;
c.gridy = 1;
panneau.add(button, c);
```

Exemple de GridBagLayout III

Exemple GridBagLayout

```
button = new JButton("5");
c.fill = GridBagConstraints.HORIZONTAL;
c.ipady = 0;       //reset to default
c.weighty = 1.0;   //request any extra vertical space
c.anchor = GridBagConstraints.PAGE_END; //bottom of space
c.insets = new Insets(10,0,0,0);  //top padding
c.gridx = 1;       //aligned with button 2
c.gridwidth = 2;   //2 columns wide
c.gridy = 2;       //third row
panneau.add(button, c);

// affecte le panneau a la fenetre
maFrame.setContentPane(panneau);
maFrame.pack();

// rend la fenetre visible
maFrame.setVisible(true);
maFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

Propriétés des widgets (`JComponent`) : visibilité

Est-ce que le composant est visible ou non ?

- méthodes `setVisible` et `isVisible`
- un widget invisible :
  - n'est plus pris en compte par le `LayoutManager`
  - ne capte plus les évènements souris et clavier

Propriétés des widgets (`JComponent`) : taille

Gestion de la taille des widgets :

- la taille courante : `setSize` et `getSize`
- trois autres tailles gérées par SWING :
  - taille préférée : `setPreferredSize` et `getPreferredSize`
  - taille minimum : `setMinimumSize` et `getMinimumSize`
  - taille maximum : `setMaximumSize` et `getMaximumSize`
- ces tailles sont utilisées (entre autres) par le `LayoutManager` pour calculer la place accordée à chaque widget

CENTRALE NANTES

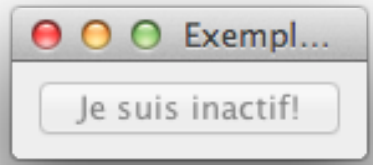
Propriétés des widgets (JComponent) : activation

Gestion de si un widget est actif ou non :

- actif/inactif : `setEnabled` et `isEnabled`
- un composant inactif ne réagit plus aux évènements provoqués par l'utilisateur (souris, clavier)
- le widget est affiché de manière "grisée"

Exemple activation

```
public static void main(String[] args) {
  JFrame maFrame = new JFrame();
  maFrame.setTitle("Exemple de JFrame");
  JPanel panneau = new JPanel();
  JButton but = new JButton("Je suis inactif!");
  but.setEnabled(false);
  panneau.add(but);
  maFrame.setContentPane(panneau);
  maFrame.setVisible(true);
  maFrame.pack();
  maFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
```



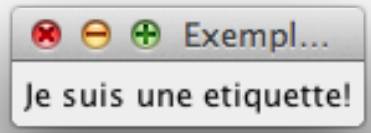
Le widget JLabel 

Un **label** (ou étiquette) représenté par le composant `JLabel` correspond à un texte non éditable sur une ligne

- il peut être accompagné d'une icône
- il est possible de modifier :
  - la couleur du texte et du fond
  - la fonte du texte
  - la position du texte et de l'icône

Exemple JLabel

```
public static void main(String[] args) {
  JFrame maFrame = new JFrame();
  maFrame.setTitle("Exemple de JFrame");
  JPanel panneau = new JPanel();
  JLabel lab = new JLabel("Je suis une etiquette!");
  panneau.add(lab);
  maFrame.setContentPane(panneau);
  maFrame.setVisible(true);
  maFrame.pack();
```



Les widgets boutons

Il existe plusieurs sortes de boutons en SWING :

- `JButton` : bouton "classique"
- `JCheckBox` : bouton ayant deux états (sélectionné/de-sélectionné) utilisé principalement pour représenter des cases à cocher
- `JRadioButton` : par convention les boutons radios forment un groupe dans lequel seul un bouton peut être sélectionné (pour représenter un choix multiple à une seule possibilité, contrairement aux `JCheckBox` qui permettent des choix multiples)
- `JMenuItem` : un bouton faisant partie d'un menu (par exemple File/...)
- `JToggleButton` : un bouton qui reste enfoncé lorsque l'on clique dessus
- pour plus de détails allez voir la page suivante : `http://docs.oracle.com/javase/tutorial/uiswing/components/button.html`

CENTRALE NANTES

Le widget JButton

Le widget JButton possède les caractéristiques suivantes (entre autres) :

- il a un texte affiché, voir les méthodes setText/getText
- il peut posséder plusieurs icônes, pouvant être différentes, en fonction de son état (sélectionné, appuyé, désactivé, lorsqu'il est sous la souris, etc.), voir les méthodes setIcon/getIcon ; setDisabledIcon/getDisabledIcon, etc.
- peut être associé à un raccourci clavier (méthode setMnemonic)
- pour plus de détails allez voir la page suivante : http://docs.oracle.com/javase/7/docs/api/javax/swing/JButton.html

Les widgets de zones de texte 

Il existe plusieurs widgets permettant de gérer du texte :

- une seule ligne de texte :
  - `JTextField` : zone d'édition de texte utilisant principalement les méthodes `setText` et `getText`
  - `JFormattedTextField` : zone d'édition permettant des saisies de texte formatté (méthodes `setValue` et `getValue` mais attention c'est à utiliser en combinaison avec la classe `AbstractFormatter`)
  - `JPasswordField` : zone d'édition de texte où les caractères tapés sont cachés. On utilise les méthodes `getPassword` et `getEchoChar`/`setEchoChar` (gestion du caractère "cachant")
- plusieurs lignes de texte : la classe `JTextArea` qui permet de gérer du texte de façon simple. On utilise les méthodes :
  - `setText`, `getText`, `getSelectedText`
  - `append`, `insert`, `replaceRange`, `replaceSelection`
  - etc.

CENTRALE NANTES

Le widget JToolBar  

Le widget JToolBar permet d'intégrer simplement une barre d'outils dans votre application Java. De manière générale une barre d'outils regroupe sous forme de boutons "icônes" des raccourcis vers des fonctionnalités également offertes dans des menus.
Les principales méthodes de la classe JToolBar sont :

- add(Component) : ajoute le composant passé en paramètre à la barre d'outils
- addSeparator : ajoute un séparateur à la barre d'outils
- isFloatable/setFloatable : permet de savoir si la barre d'outils peut être décrochée de la fenêtre de l'application et de modifier cette propriété
- pour plus de détails allez voir la page suivante : http://docs.oracle.com/javase/7/docs/api/javax/swing/JToolBar.html

Les widgets de gestion d'un menu : JMenuBar, JMenu, JMenuItem, etc.

Le widget JMenu permet d'intégrer simplement un menu dans votre application Java.

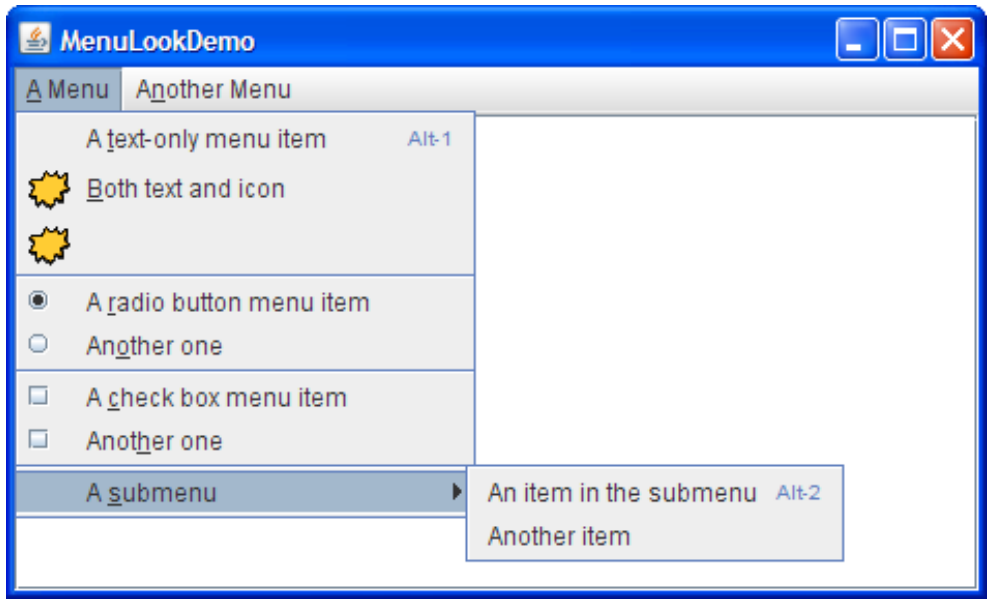
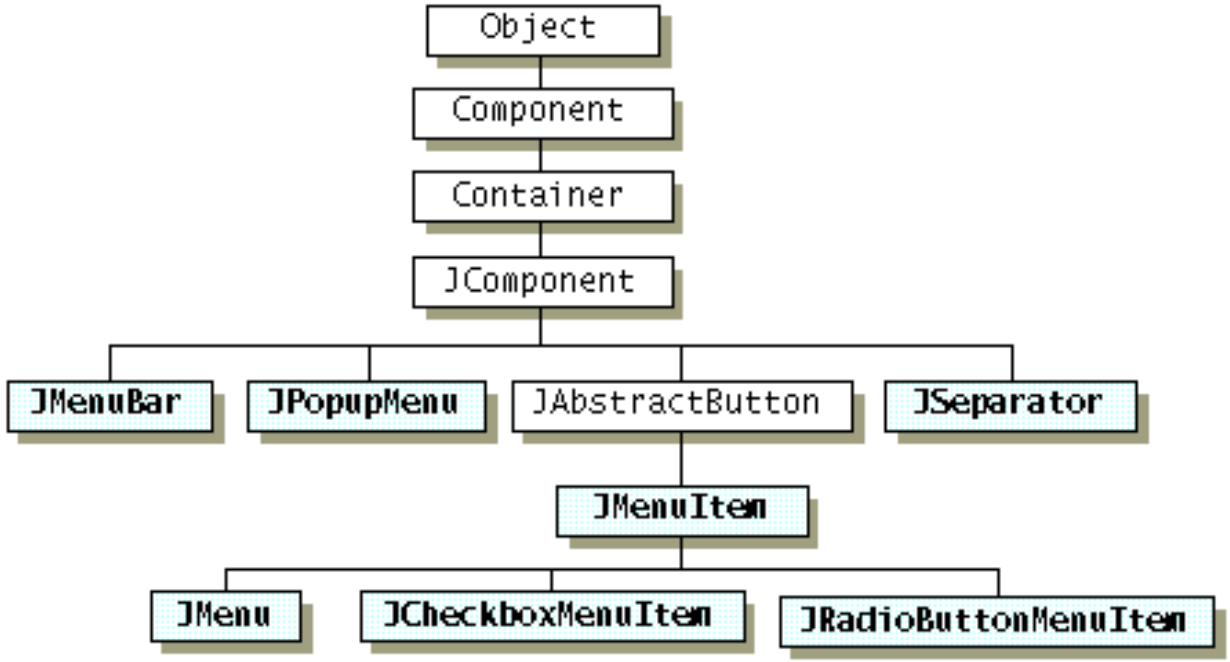


Figure: Illustration de l'utilisation dans une application de plusieurs widgets relatifs aux menus (JMenuBar, JMenu, JMenu, JRadioButtonMenuItem, JCheckboxMenuItem, etc.). Nous pouvons également remarquer qu'un JMenuItem peut être représenté soit par une image, soit par du texte

Les widgets de gestion d'un menu : JMenuBar, JMenu, JMenuItem, etc. II

Voici la hiérarchie des classes relatives aux menus en SWING :



Le widget JMenu est quelque peu à part dans le sens où il est forcément inclus dans un widget JMenuBar qui représente une barre de menus dans

Les widgets de gestion d'un menu : JMenuBar, JMenu, JMenuItem, etc. III

Un widget JMenuBar contient un ou plusieurs JMenu et est positionnée dans l'application en fonction des préférences du système (généralement en haut de la fenêtre).
La principale méthode des classes JMenuBar et JMenu est :

- add(Component) : ajoute le composant passé en paramètre à la barre de menu (ou au menu)

Pour un JMenuItem, on a les méthodes suivantes:

- les méthodes héritées de AbstractButton
- pour plus de détails allez voir les pages suivantes :
  - http://docs.oracle.com/javase/7/docs/api/javax/swing/JToolBar.html
  - http://docs.oracle.com/javase/7/docs/api/javax/swing/AbstractButton.html

CENTRALE NANTES

Les évènements SWING

- Les composants SWING (widgets) créent des évènements, soit **directement**, soit par **une action de l'utilisateur sur le widget**. Ces évènements peuvent **déclencher une action** exécutée par d'autre(s) composant(s).
- On parle de "programmation évènementielle" pour faire référence à ce paradigme par actions/réactions
- L'API Java nous fournit un système de "**sources**" et d'"**auditeurs**" (ou "**écouteurs**")
  - un widget qui créé un évènement est appelé **source**. Le composant source délègue le traitement de l'évènement au composant **auditeur** (*listener* en anglais)
  - un widget qui traite un évènement est appelé **auditeur**
- Un widget "**auditeur**" doit s'inscrire auprès du composant **source** des évènements qu'il veut traiter

Gestion des évènements

Les évènements permettent de faire réagir notre interface en fonction des actions de l'utilisateur :

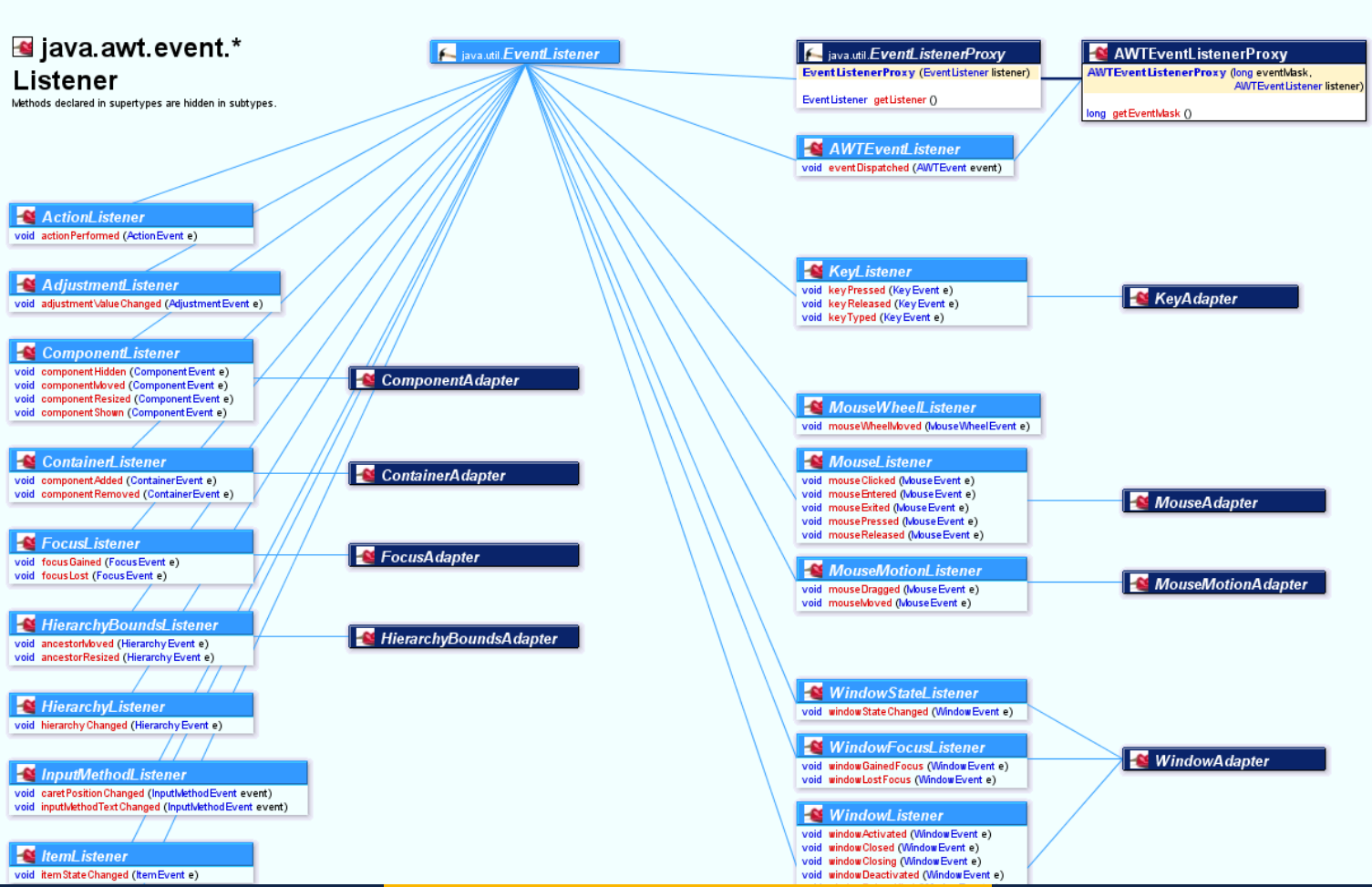
- "Si l'utilisateur clique sur tel bouton, alors il faut afficher tel message dans telle zone de texte"
- un objet **écouteur d'évènements** : est une instance d'une classe qui **implémente une interface d'"écoute"** (dont le nom se termine par `Listener`)
- un objet **source d'évènements** : permet de recenser les objets qui vont écouter ce que va faire cet objet, et de leur envoyer à tous un **objet évènement** (`*Event`) correspondant aux actions utilisateur
- donc lorsqu'un évènement se produit (par exemple grâce à une action utilisateur) : la source d'évènement envoie à tous ses écouteurs recensés l'évènement en question

Les écouteurs : principes

- Approche écouteur/écouté :
  - tout widget est susceptible d'émettre un évènement lorsque celui-ci est sollicité par l'utilisateur de différentes manières (clics, saisie clavier, etc.)
  - vous allez alors créer des objets spécifiquement à l'"écoute" de ces types d'évènements, qui contiendront le code a exécuter lorsqu'ils surviennent
- Un même widget peut avoir plusieurs objets écouteurs, qui sont autant de façons différentes de réagir à l'évènement qui s'est produit
- La mise en œuvre de ce système est très encadré :
  - vos classes "écouteuses" doivent implémenter des **interfaces pré-définies** pour s'assurer contractuellement qu'elles contiennent bien du code à exécuter dans le cas où certains types d'évènements surgissent
  - le corollaire c'est qu'un widget n'accepte d'être écouté que par des objets qui remplissent ce contrat
- Á tout moment dans le programme, on peut décider de créer une

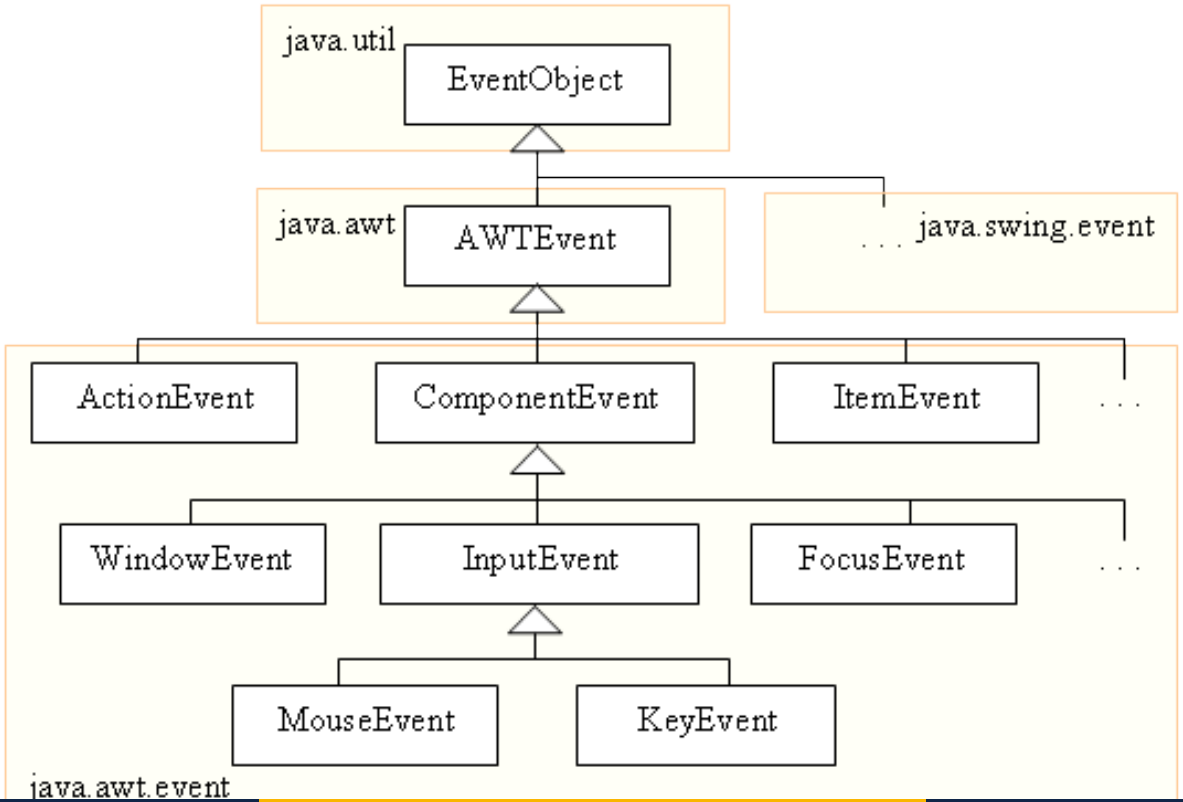
Les interfaces écouteurs : *Listener

Il existe une hiérarchie des interfaces d'écoute en Java :



Les évènements : `*Event`

Chaque interface `Listener` dispose d'une classe évènement `Event` associée qui fournit une description de l'évènement capturé et fournit un moyen de le traiter (via une méthode) :



L'interface ActionListener et l'évènement : ActionEvent

ActionListener

```
public interface ActionListener {
  void actionPerformed(ActionEvent e);
}
```

ActionEvent

```
public class ActionEvent {
  Object getSource();
  int getModifiers();
  long getWhen();
  String getActionCommand();
  String paramString();
}
```

L'interface MouseListener et l'évènement : MouseEvent

MouseListener

```
public interface ActionListener {
  void mouseClicked(MouseEvent e);
  void mouseEntered(MouseEvent e);
  void mouseExited(MouseEvent e);
  void mousePressed(MouseEvent e);
  void mouseReleased(MouseEvent e);
}
```

MouseEvent

```
public class ActionEvent {
  Object getSource();
  int getModifiers();
  Point getPoint();
  int getX();
  int getY();
  int getModifiers();
  long getWhen();
  int getButton();
  int getClickCount();
}
```

CENTRALE NANTES

L'interface KeyListener et l'évènement : KeyEvent


MouseListener

```
public interface KeyListener {
void keyTyped(KeyEvent e);
   void keyPressed(KeyEvent e);
   void KeyReleased(KeyEvent e);
}
```

MouseEvent

```
public class KeyEvent {
  int getExtendedKeyCode();
  static int getExtendedKeyCodeForChar(int
     c);
  char getKeyChar();
  int getKeyCode();
  int getKeyLocation();
  static String getKeyModifiersText(int
     modifiers);
  static String getKeyText(int keyCode);
  boolean isActionKey();
  String paramString();
  void setKeyChar(char keyChar);
  void setKeyCode(int keyCode);
}
```

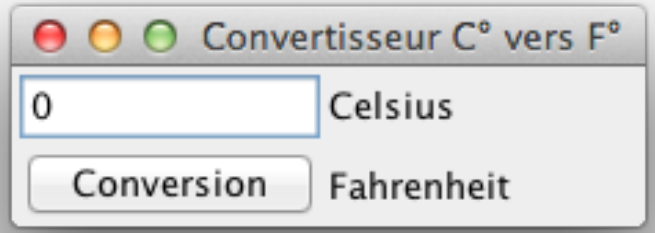
Un mini exemple complet mais simple

Afin de vous présenter un petit exemple complet, nous allons réaliser une application de conversion d'une température en degrés Celsius vers des degrés Fahrenheit.
Ce mini exemple peut-être fait directement en ligne et en utilisant l'IDE NetBeans pour créer les interfaces (nous allons le faire "à la main") :
`http://docs.oracle.com/javase/tutorial/uiswing/learn/index.html`
Nous allons voir les concepts suivants :

- la création d'interface en ligne de code
- l'utilisation de layout simple
- la création d'une classe implémentant l'interface `ActionListener`
- la manière de relier une interface et une classe gérant une interaction utilisateur simple

Création de l'interface utilisateur

L'interface que nous cherchons à réaliser est la suivante :



Elle est composée des widgets suivants :

- une `JFrame` associée à un `JPanel`
- le positionnement des différents widgets est assuré par un `GridLayout`
- deux `JLabel` : un qui affiche le texte "Celsius" et un affichant "Fahrenheit" au début, puis le résultat suivi de "Fahrenheit"
- un champ de texte formaté spécialement pour les nombres : `JFormattedTextField` associé à un `NumberFormat`

Code de notre classe principale ConvertisseurTemp

```
/*
 * Mini exemple de manipulation des widgets SWING et des listeners/events
 * Version issue d'un tutorial en ligne disponible sur le site de Java
 * plus de details ici :
 * http://docs.oracle.com/javase/tutorial/uiswing/learn/index.html
 */

import java.awt.Dimension;
import java.awt.GridLayout;
import java.text.NumberFormat;
import javax.swing.JButton;
import javax.swing.JFormattedTextField;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
```

Code de notre classe principale ConvertisseurTemp II

```
/**
 *
 * @author Jean-Marie Normand <jean-marie.normand@ec-nantes.fr>
 */

public class ConvertisseurTemp {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // Declaration d'une JFrame
        JFrame maFenetre = new JFrame("Convertisseur C vers F");

        // Creation d'un panneau
        JPanel monPanneau = new JPanel();
```

Code de notre classe principale ConvertisseurTemp III

```
// Creation d'un grid layout de 2x2
GridLayout layout = new GridLayout(2,2);
// on applique le layout au panneau
monPanneau.setLayout(layout);

// Classe necessaire pour formatter notre champ de texte
NumberFormat celsiusFormat = NumberFormat.getNumberInstance();
celsiusFormat.setMaximumFractionDigits(8);

// Creation d'un champ de texte formate !
JFormattedTextField celsius = new JFormattedTextField(celsiusFormat);
celsius.setSize(100,20);
celsius.setMinimumSize(new Dimension(100,20));
celsius.setPreferredSize(new Dimension(100,20));
celsius.setValue(new Double(0.0));
```

Code de notre classe principale ConvertisseurTemp IV

```
    // Creation d'un label
    JLabel labelCelsius = new JLabel("Celsius");

    // Creation d'un bouton
    JButton convert = new JButton("Conversion");

    // Creation d'un deuxieme label
    JLabel labelFahr = new JLabel("Fahrenheit");

    // On cree un objet special qui a pour but de gerer le clic
    // sur le bouton et de realiser la conversion
    // en plus il devra modifier le texte du label Fahrenheit pour afficher
    // le resultat de la conversion
    ReactionBoutonConversion reacConv = new
ReactionBoutonConversion(celsius,labelFahr);
```

Code de notre classe principale ConvertisseurTempV

```
// On ajoute le listener au bouton !
convert.addActionListener(reacConv);

// On ajoute les composants au panneau
monPanneau.add(celsius);
monPanneau.add(labelCelsius);
monPanneau.add(convert);
monPanneau.add(labelFahr);

// On finalise la fenetre
maFenetre.setContentPane(monPanneau);
maFenetre.pack();
// on la rend visible
maFenetre.setVisible(true);
maFenetre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

Code de notre classe principale ConvertisseurTemp VI

```
    }
}
```

Code de notre classe “écouteur”

ReactionBoutonConversion

```
/*
 * Classe permettant d'intercepter et de gerer la reaction
 * a l'interaction de l'utilisateur sur le bouton de conversion
 * i.e. quand l'utilsateur clique sur le bouton de conversion
 * on doit calculer le resultat de la conversion et afficher
 * le resultat dans le JLabel cree a cet effet.
 */

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JFormattedTextField;
import javax.swing.JLabel;

/**
 *
 * @author Jean-Marie Normand <jean-marie.normand@ec-nantes.fr>
```

Code de notre classe “écouteur”

ReactionBoutonConversion II

```
 */


public class ReactionBoutonConversion implements ActionListener {
    // Attributs

    // Le champ de texte contenant la valeur de la temperature
    private JFormattedTextField celsiusValue;

    // La reference vers le Label dont on veut modifier le nom
    private JLabel affichageFahr;

    // Constructeur
    public ReactionBoutonConversion(JFormattedTextField celsiusText, JLabel
     lab) {
        this.celsiusValue = celsiusText;
```

Code de notre classe “écouteur”

ReactionBoutonConversion III

```
        this.affichageFahr = lab;
    }


    // On implemente ActionListener --> on doit redefinir actionPerformed !
    @Override
    public void actionPerformed(ActionEvent e) {
        // la methode 'getValue' de JFormattedTextField
        // retourne un objet de type 'Number'
        Number valeurFormattedText = (Number)this.celsiusValue.getValue();
        // mais nous savons ici que c'est un double on peut donc
        // utiliser la methode 'doubleValue' pour recuperer le doule
     correspondant
        double tempFahr = (valeurFormattedText.doubleValue() * 1.8 + 32);
```

Code de notre classe “écouteur” ReactionBoutonConversion IV

```
        // on arrondit le resultat a deux chiffres apres la virgule
        tempFahr = (double)Math.round(tempFahr * 100) / 100;
        affichageFahr.setText(tempFahr + " Fahrenheit");
    }
}
```

Plan

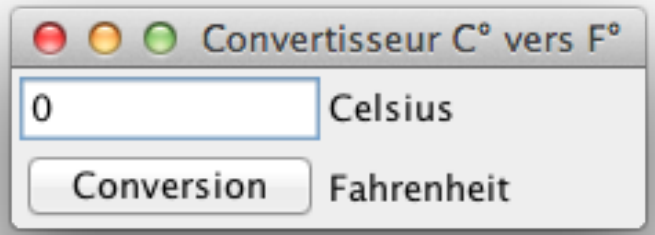
Un mini exemple complet mais simple

Afin de vous présenter un petit exemple complet, nous allons réaliser une application de conversion d'une température en degrés Celsius vers des degrés Fahrenheit.
Ce mini exemple peut-être fait directement en ligne et en utilisant l'IDE NetBeans pour créer les interfaces (nous allons le faire "à la main") :
`http://docs.oracle.com/javase/tutorial/uiswing/learn/index.html`
Nous allons voir les concepts suivants :

- la création d'interface en ligne de code
- l'utilisation de layout simple
- la gestion des évènements (en particulier `ActionListener`)
- la manière de relier une interface et une classe gérant une interaction utilisateur simple

Création de l'interface utilisateur

L'interface que nous cherchons à réaliser est la suivante :



Elle est composée des widgets suivants :

- une `JFrame` associée à un `JPanel`
- le positionnement des différents widgets est assuré par un `GridLayout`
- deux `JLabel` : un qui affiche le texte "Celsius" et un affichant "Fahrenheit" au début, puis le résultat suivi de "Fahrenheit"
- un champ de texte formaté spécialement pour les nombres : `JFormattedTextField` associé à un `NumberFormat`

Code de notre classe principale ConvertisseurTemp

```
/*
 * Mini exemple de manipulation des widgets SWING et des listeners/events
 * Version issue d'un tutorial en ligne disponible sur le site de Java
 * plus de details ici :
 * http://docs.oracle.com/javase/tutorial/uiswing/learn/index.html
 */

import java.awt.Dimension;
import java.awt.GridLayout;
import java.text.NumberFormat;
import javax.swing.JButton;
import javax.swing.JFormattedTextField;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
```

Code de notre classe principale ConvertisseurTemp II

```
/**
 *
 * @author Jean-Marie Normand <jean-marie.normand@ec-nantes.fr>
 */

public class ConvertisseurTemp {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // Declaration d'une JFrame
        JFrame maFenetre = new JFrame("Convertisseur C vers F");

        // Creation d'un panneau
        JPanel monPanneau = new JPanel();
```

Code de notre classe principale ConvertisseurTemp III

```
// Creation d'un grid layout de 2x2
GridLayout layout = new GridLayout(2,2);
// on applique le layout au panneau
monPanneau.setLayout(layout);

// Classe necessaire pour formatter notre champ de texte
NumberFormat celsiusFormat = NumberFormat.getNumberInstance();
celsiusFormat.setMaximumFractionDigits(8);

// Creation d'un champ de texte formate !
JFormattedTextField celsius = new JFormattedTextField(celsiusFormat);
celsius.setSize(100,20);
celsius.setMinimumSize(new Dimension(100,20));
celsius.setPreferredSize(new Dimension(100,20));
celsius.setValue(new Double(0.0));
```

Code de notre classe principale ConvertisseurTemp IV

```
// Creation d'un label
JLabel labelCelsius = new JLabel("Celsius");

// Creation d'un bouton
JButton convert = new JButton("Conversion");

// Creation d'un deuxieme label
JLabel labelFahr = new JLabel("Fahrenheit");

// On ajoute les composants au panneau
monPanneau.add(celsius);
monPanneau.add(labelCelsius);
monPanneau.add(convert);
monPanneau.add(labelFahr);
```

Code de notre classe principale ConvertisseurTempV

```
        // On finalise la fenetre
        maFenetre.setContentPane(monPanneau);
        maFenetre.pack();
        // on la rend visible
        maFenetre.setVisible(true);
        maFenetre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

Pour gérer le clic : 2 solutions I

Afin d'intercepter le clic sur le bouton `Conversion`, il faut capturer l'évènement associé, à savoir un `ActionEvent`.
Pour ce faire, trois options s'offrent à nous :

1. Créer une nouvelle classe `ConvertisseurTempAL` afin qu'elle implémente l'interface `ActionListener` et créer une classe qui utilise une instance de `ConvertisseurTemp`
   - Rajouter `implements ActionListener`
   - Ajouter une méthode `actionPerformed(ActionEvent e)` prenant en compte la gestion du clic
   - Dans le constructeur de cette classe, créer la GUI
   - Ajouter une méthode `showGUI` qui pourra être appelée dans le `main` de la classe de test

Pour gérer le clic : 2 solutions I

Afin d'intercepter le clic sur le bouton `Conversion`, il faut capturer l'évènement associé, à savoir un `ActionEvent`.
Pour ce faire, trois options s'offrent à nous :

2. Créer une nouvelle classe `ReactionBoutonConversion` (vous pouvez choisir un autre nom :) qui implémente l'interface `ActionListener` :
   - Créer la nouvelle classe avec : `implements ActionListener`
   - Ajouter une méthode `actionPerformed(ActionEvent e)` prenant en compte la gestion du clic
   - Garder à l'esprit que cette classe va devoir accéder :
     - au widget contenant le texte correspondant à la température en Celsius
     - au widget affichant le texte correspondant à la température en Farenheit
   - Solution :
     - ajouter deux attributs à notre nouvelle classe
     - écrire le constructeur correspondant

CENTRALE NANTES

Pour gérer le clic : 2 solutions I

Afin d'intercepter le clic sur le bouton `Conversion`, il faut capturer l'évènement associé, à savoir un `ActionEvent`.
Pour ce faire, trois options s'offrent à nous :

3. Utiliser le mécanisme des classes internes (ou *inner classes*) qui est principalement utilisé pour la gestion des évènements dans les GUI
   - Ajouter un écouteur à notre bouton, en utilisant la méthode `addActionListener`
   - Cet écouteur va en fait être une nouvelle classe générée directement dans le code de la méthode `addActionListener` !
   - Comme cette nouvelle classe doit être un `ActionListener`, elle doit implémenter la méthode `actionPerformed` !
   - Cette méthode est elle aussi déclarée directement à l'intérieur de la classe interne

Pour gérer le clic : 2 solutions II

- Comme tout cela se passe dans le `main` $\Rightarrow$ on doit passer tous les widgets en static ! (cf. cours sur les `static` et sur la méthode `main`)

Solution 1 ConvertisseurTempAL I

Fichier ConvertisseurTempAL :

```
import java.awt.Dimension;
import java.awt.GridLayout;
import java.text.NumberFormat;
import javax.swing.JButton;
import javax.swing.JFormattedTextField;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

public class ConvertisseurTempAL implements ActionListener {

   // Attributs
   public JFrame maFenetre;
   public JPanel monPanneau;
```

Solution 1 ConvertisseurTempAL II

```
public GridLayout layout;
public NumberFormat celsiusFormat;
public JFormattedTextField celsius;
public JLabel labelCelsius;
public JLabel labelFahr;

ConvertisseurTempAL(){
   // Declaration d'une JFrame
   maFenetre = new JFrame("Convertisseur C vers F");

   // Creation d'un panneau
   monPanneau = new JPanel();

   // Creation d'un grid layout de 2x2
   layout = new GridLayout(2,2);
   // on applique le layout au panneau
   monPanneau.setLayout(layout);
```

CENTRALE NANTES

Solution 1 ConvertisseurTempAL III

```
// Classe necessaire pour formatter notre champ de texte
celsiusFormat = NumberFormat.getNumberInstance();
celsiusFormat.setMaximumFractionDigits(8);

// Creation d'un champ de texte formate !
celsius = new JFormattedTextField(celsiusFormat);
celsius.setSize(100,20);
celsius.setMinimumSize(new Dimension(100,20));
celsius.setPreferredSize(new Dimension(100,20));
celsius.setValue(new Double(0.0));

// Creation d'un label
labelCelsius = new JLabel("Celsius");

// Creation d'un deuxieme label
labelFahr = new JLabel("Fahrenheit");

// Creation d'un bouton
```

CENTRALE NANTES

Solution 1 ConvertisseurTempAL IV

```
        JButton convert = new JButton("Conversion");
        convert.addActionListener(this);

        // On ajoute les composants au panneau
        monPanneau.add(celsius);
        monPanneau.add(labelCelsius);
        monPanneau.add(convert);
        monPanneau.add(labelFahr);

        // On finalise la fenetre
        maFenetre.setContentPane(monPanneau);
        maFenetre.pack();
        maFenetre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    public void showGUI() {
        maFenetre.setVisible(true);
    }
```

Solution 1 ConvertisseurTempAL V

```
    @Override
    public void actionPerformed(ActionEvent e) {
        Number valeurFormattedText = (Number)this.celsius.getValue();
        double tempFahr = (valeurFormattedText.doubleValue() * 1.8 + 32);
        // on arrondit le resultat a deux chiffres apres la virgule
        tempFahr = (double)Math.round(tempFahr * 100) / 100;
        labelFahr.setText(tempFahr + " Fahrenheit");
    }

}
```

Fichier MainConvertisseurAL.java

Solution 1 ConvertisseurTempAL VI

```
public class MainConvertisseurAL {
   public static void main(String[] args) {
      // on la rend visible
      ConvertisseurTempAL conv = new ConvertisseurTempAL();
      conv.showGUI();
   }
}
```

Solution 2 : Classe ConvertisseurTemp et classe "écouteur" ReactionBoutonConversion I

Fichier ConvertisseurTemp.java :

```
import java.awt.Dimension;
import java.awt.GridLayout;
import java.text.NumberFormat;
import javax.swing.JButton;
import javax.swing.JFormattedTextField;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;

public class ConvertisseurTemp {

    public static void main(String[] args) {
        // Declaration d'une JFrame
        JFrame maFenetre = new JFrame("Convertisseur C vers F");
```

Solution 2 : Classe ConvertisseurTemp et classe "écouteur" ReactionBoutonConversion II

```
// Creation d'un panneau
JPanel monPanneau = new JPanel();

// Creation d'un grid layout de 2x2
GridLayout layout = new GridLayout(2,2);
// on applique le layout au panneau
monPanneau.setLayout(layout);

// Classe necessaire pour formatter notre champ de texte
NumberFormat celsiusFormat = NumberFormat.getNumberInstance();
celsiusFormat.setMaximumFractionDigits(8);

// Creation d'un champ de texte formate !
JFormattedTextField celsius = new JFormattedTextField(celsiusFormat);
celsius.setSize(100,20);
```

Solution 2 : Classe ConvertisseurTemp et classe "écouteur" ReactionBoutonConversion III

```
celsius.setMinimumSize(new Dimension(100,20));
celsius.setPreferredSize(new Dimension(100,20));
celsius.setValue(new Double(0.0));

// Creation d'un label
JLabel labelCelsius = new JLabel("Celsius");

// Creation d'un bouton
JButton convert = new JButton("Conversion");

// Creation d'un deuxieme label
JLabel labelFahr = new JLabel("Fahrenheit");

// On cree un objet special qui a pour but de gerer le clic
// sur le bouton et de realiser la conversion
// en plus il devra modifier le texte du label Fahrenheit pour afficher
```

Solution 2 : Classe ConvertisseurTemp et classe "écouteur" ReactionBoutonConversion IV

```
    // le resultat de la conversion
    ReactionBoutonConversion reacConv = new
ReactionBoutonConversion(celsius,labelFahr);

    // On ajoute le listener au bouton !
    convert.addActionListener(reacConv);

    // On ajoute les composants au panneau
    monPanneau.add(celsius);
    monPanneau.add(labelCelsius);
    monPanneau.add(convert);
    monPanneau.add(labelFahr);

    // On finalise la fenetre
    maFenetre.setContentPane(monPanneau);
    maFenetre.pack();
```

CENTRALE NANTES

Solution 2 : Classe ConvertisseurTemp et classe "écouteur" ReactionBoutonConversion V

```
        // on la rend visible
        maFenetre.setVisible(true);
        maFenetre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

Fichier ReactionBoutonConversion.java :

Solution 2 : Classe ConvertisseurTemp et classe "écouteur" ReactionBoutonConversion VI

```
/*
 * Classe permettant d'intercepter et de gerer la reaction
 * a l'interaction de l'utilisateur sur le bouton de conversion
 * i.e. quand l'utilisateur clique sur le bouton de conversion
 * on doit calculer le resultat de la conversion et afficher
 * le resultat dans le JLabel cree a cet effet.
 */

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JFormattedTextField;
import javax.swing.JLabel;

public class ReactionBoutonConversion implements ActionListener {
    // Attributs
```

Solution 2 : Classe ConvertisseurTemp et classe "écouteur" ReactionBoutonConversion VII

```
// Le champ de texte contenant la valeur de la temperature
private JFormattedTextField celsiusValue;

// La reference vers le Label dont on veut modifier le nom
private JLabel affichageFahr;

// Constructeur
public ReactionBoutonConversion(JFormattedTextField celsiusText, JLabel
 lab) {
    this.celsiusValue = celsiusText;
    this.affichageFahr = lab;
}

// On implemente ActionListener --> on doit redefinir actionPerformed !
@Override
public void actionPerformed(ActionEvent e) {
```

Solution 2 : Classe ConvertisseurTemp et classe "écouteur" ReactionBoutonConversion VIII

```
        Number valeurFormattedText = (Number)this.celsiusValue.getValue();
        double tempFahr = (valeurFormattedText.doubleValue() * 1.8 + 32);
        // on arrondit le resultat a deux chiffres apres la virgule
        tempFahr = (double)Math.round(tempFahr * 100) / 100;
        affichageFahr.setText(tempFahr + " Fahrenheit");
    }
}
```

Solution 3 ConvertisseurTempStat  I

Fichier ConvertisseurTempStat :

```
import java.awt.Dimension;
import java.awt.GridLayout;
import java.text.NumberFormat;
import javax.swing.JButton;
import javax.swing.JFormattedTextField;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

public class ConvertisseurTempStat {
   // Attributs
   public static JFrame maFenetre;
   public static JPanel monPanneau;
   public static GridLayout layout;
```

Solution 3 ConvertisseurTempStat II

```
public static NumberFormat celsiusFormat;
public static JFormattedTextField celsius;
public static JLabel labelCelsius;
public static JLabel labelFahr;

public static void main(String[] args) {
   // Declaration d'une JFrame
   maFenetre = new JFrame("Convertisseur C vers F");

   // Creation d'un panneau
   monPanneau = new JPanel();

   // Creation d'un grid layout de 2x2
   layout = new GridLayout(2,2);
   // on applique le layout au panneau
   monPanneau.setLayout(layout);

   // Classe necessaire pour formatter notre champ de texte
```

Solution 3 ConvertisseurTempStat III

```
celsiusFormat = NumberFormat.getNumberInstance();
celsiusFormat.setMaximumFractionDigits(8);

// Creation d'un champ de texte formate !
celsius = new JFormattedTextField(celsiusFormat);
celsius.setSize(100,20);
celsius.setMinimumSize(new Dimension(100,20));
celsius.setPreferredSize(new Dimension(100,20));
celsius.setValue(new Double(0.0));

// Creation d'un label
labelCelsius = new JLabel("Celsius");

// Creation d'un deuxieme label
labelFahr = new JLabel("Fahrenheit");

// Creation d'un bouton
JButton convert = new JButton("Conversion");
```

Solution 3 ConvertisseurTempStat IV

```
convert.addActionListener(new ActionListener() {
   @Override
   public void actionPerformed(ActionEvent e) {
      Number valeurFormattedText = (Number)celsius.getValue();
      double tempFahr = (valeurFormattedText.doubleValue() * 1.8 + 32);
      // on arrondit le resultat a deux chiffres apres la virgule
      tempFahr = (double)Math.round(tempFahr * 100) / 100;
      labelFahr.setText(tempFahr + " Fahrenheit");
   }
});

// On ajoute les composants au panneau
monPanneau.add(celsius);
monPanneau.add(labelCelsius);
monPanneau.add(convert);
monPanneau.add(labelFahr);

// On finalise la fenetre
```

Solution 3 ConvertisseurTempStat V

```
        maFenetre.setContentPane(monPanneau);
        maFenetre.pack();
        maFenetre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        maFenetre.setVisible(true);
    }
}
```