# Interface Homme-Machine (IHM)

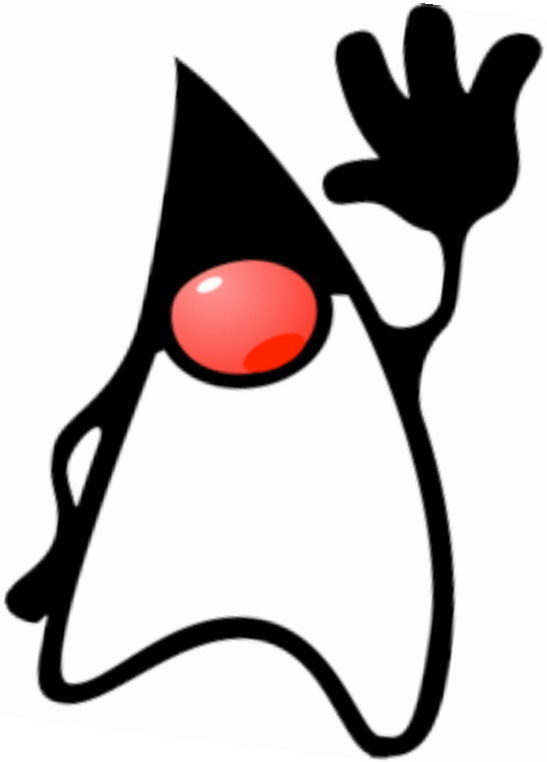
*Partie SWING*

**INFO0202**

**janvier 2019**

*Contact : Jean-Charles.Boisson@univ-reims.fr*

# JAVA Swing



# Qu'est ce que SWING ?

- Une bibliothèque :
  - incluse dans la distribution JAVA
  - Servant à produire des IHM portables
  - Étendant la librairie AWT

    ⇔ *Abstract Window Toolkit*

# AWT

- Bibliothèque fournie originellement avec JAVA pour produire des IHM qui :
  - Dessine en se servant des ressources graphiques du système
  - Est rapide
  - Peut être complexe à utiliser

# AWT vs SWING

- SWING :
  - Plus récent (qu'AWT)
  - Dessine en se servant de la machine virtuelle JAVA
  - Est lente (comparée à AWT)
  - Est plus souple d'utilisation (qu'AWT)

# Pourquoi utiliser Swing ?

- Tout programme JAVA est portable.

➔ Toute IHM programmée en JAVA est portable.

- Rendu graphique :
  - Sous AWT : peut varier fortement d'un système à un autre.
  - Sous SWING : est globalement stable du fait que c'est la machine virtuelle qui s'occupe du graphisme.

# LES COMPOSANTS DE BASE

# Les composants

- Tout composant Swing :
  - Est situé dans le paquetage **javax.swing**
  - Hérite de composants AWT (pour la plupart)
  - Est facile à utiliser
  - Est capable de répondre à un grand nombre d'appels de méthodes (*lesquelles ?*)

# Les plus courants

- Le JPanel :
  - Composant servant de conteneur générique à d'autres composants.
  - Est souvent le conteneur de plus haut niveau (après la fenêtre).
  - Création avec le constructeur par défaut :
    - **JPanel** panel = new **JPanel**();

# Les plus courants

- Le JLabel :
  - Composant servant à l'affichage de texte et/ou d'une image.
  - Le contenu est initié à la création où via la méthode « setText » :
    - **Jlabel** label = new **Jlabel**(« Hello World »);
    - label.**setText**(« Coucou tout le monde »);

# Les plus courants

- Le JTextField :
  - Composant servant à la saisie de texte
  - Peut être utilisé pour les mots de passe (contenu caché)
  - L'obtention des autres types se fait par l'utilisation des objets associés (Integer, Float, …)
  - On peut donner un contenu par défaut lors de la création de l'objet et paramétriser le JTextField en fonction :
    - **JTextField** textField = new **JTextField**(″Nom Joueur 1″);
    - textField.**setColumns**(textField.**getText()**.length());

# Les plus courants

- Le JButton :
  - Composant permettant d'afficher un bouton
  - Peut contenir du texte et/ou une image
  - Peut changer d'état suivant le passage de la souris au-dessus de lui.
  - Ne devient utile que lorsqu'un évènement lui est associé (*cf. les évènements*).
  - Ex :
    - **JButton** button = new **JButton**("OK"):
    - button.**setMnemonic**('k')// raccourci clavier alt+'k'

# Les plus courants

- Le JRadioButton :
  - Permet le choix d'un élément parmi plusieurs.
  - Son équivalent en case à cocher existe (JCheckBox).
  - Est généralement associé à un ButtonGroup pour permettre la désélection des autres dès qu'un choix est fait.
  - On peut préciser lequel est activé par défaut.

# Les plus courants

- Le JRadioButton :
  - **ButtonGroup** groupe = new **ButtonGroup**();
  - **JRadioButton** jRadioButton1 = new **JRadioButton**(″Choix 1″);
  - **JRadioButton** jRadioButton2 = new **JRadioButton**(″Choix 2″);
  - **JRadioButton** jRadioButton3 = new **JRadioButton**(″Choix 3″);
  - jRadioButton2.**setSelected**(true);
  - groupe.**add**(jRadioButton1);
  - groupe.**add**(jRadioButton2);
  - groupe.**add**(jRadioButton3);

# Les plus courants

- Le JComboBox :
  - Permet l'affichage de (petites) listes déroulantes.
  - Propose à l'utilisateur de faire un choix.
  - Peut permettre de saisir une nouvelle valeur.
  - A deux états : ouvert ou fermé.

# Les plus courants

- Le JComboBox :
  - **Vector** couleurs = new **Vector**();
  - couleurs.**add**("Jaune");
  - couleurs.**add**("Rouge");
  - couleurs.**add**("Vert");
  - **JComboBox** jCB = new **JComboBox**(couleurs)

# Paramétrer les composants

- Tout composant est :
  - Visible ou non :
    - **setVisible**(true/false)
    - boolean **isVisible**()
  - Activé ou non :
    - **setEnabled**(true/false)
    - boolean **isEnabled**()

# Paramétrer les composants

- Taille des composants :
  - Taille réelle et taille souhaitée :
    - Dépendent d'un objet Dimension :
      - Dimension **getSize**();
      - Dimension **getSize**(Dimension dim);
      - Accesseur **getHeight()** et **getWidth()** sur Dimension
    - La taille peut être imposée en dur

      ➔ **setSize**(Dimension)
    - La taille peut être calculée en fonction des contenus correspondant.

# Paramétrer les composants

- Modifier le curseur suivant les composants :
  - Croix, sablier, …
- Associer un curseur prédéfini à un composant :
  - **JButton** jbutton = new **JButton**(″ok″);
  - **Cursor** cursor = **Cursor.getPredefinedCursor**(**Cursor.CROSSHAIR_CURSOR**);
  - Jbutton.**setCursor**(cursor);

# Paramétrer les composants

- Utiliser des bulles d'aide (JToolTip):
  - Permet d'améliorer l'utilisation de l'interface
  - Est compatible avec tous les composants
  - Méthode générique : setToolTipText("De l'aide")
  - Méthode plus spécifique pour certains composants (panneau à onglets par exemple)

# Paramétrer les composants

- On peut jouer sur les couleurs des composants :
  - **setBackground**(Color)
  - **setForeground**(Color)

- On peut donner des bordures particulières aux composantes (pour grouper différentes éléments) :
  - **setBorder**(Border)

# LES LAYOUTS

# Disposition des composants

- Par défaut :
  - un composant s'ajoute à celui qui le contient : méthode add
  - l'ordre d'ajout impacte directement sur l'ordre d'affichage.
  - Tous les composants suivent un même layout

# Qui contient tout ?

- Le composant qui contient tous les autres est (dans bien des cas) la JFrame.

- Une JFrame :
  - Est une fenêtre.
  - Possède :
    - Un titre.
    - Une barre de menu (JMenuBar).
    - Un comportement par défaut lors de  sa fermeture.

# Un layout c'est quoi ?

- C'est une stratégie de positionnement des composants.
- JAVA fourni des layout par défaut.
- On associe un layout à un composant avant d'inclure des composants à l'intérieur.
- Le layout par défaut est le FlowLayout

# Le FlowLayout

- Layout par défaut des composants (awt).

- Tout composant qui est ajouté l'est de :
  - Gauche à droite
  - De haut en bas

- La taille du composant impacte sur l'agencement des composants internes.

# Le GridLayout

- Layout disponible sous JAVA (awt)
- Correspond à une grille de positionnement
- L'ordre des éléments se fait comme dans le FlowLayout
- Un redimensionnement de la fenêtre conserve l'ordonnancement des composants.

# Le GridLayout

- La déclaration requiert les dimensions de la grille.

- Il faut fixer le layout avant d'ajouter des composants.

- Exemple :
  - **GridLayout** layout = new **GridLayout**(3,2);
  - monComposant.**setLayout**(layout);

# Le BorderLayout

- Layout disponible sous JAVA (awt)

- Respecte ce schéma de positionnement :

| | | |
|---|---|---|
| **North** | | |
| **West** | **Center** | **East** |
| **South** | | |

# Le BorderLayout

- L'ajout d'un composant se fait par défaut au centre ou à autre une position à préciser.
- Comme le GridLayout, l'agencement des composants est préservées en cas de redimensionnement.

- Exemple :
  - monComposant.setLayout(new **BorderLayout()**);
  - monComposant.add(unComposant)
  - monComposant.**add**(autreComposant,**BorderLayout.NORTH**);

# Le Boxlayout

- Layout disponible sous JAVA (swing)
- Ressemble à un FlowLayout dans lequel on pourrait imposer l'alignement (vertical ou horizontal).
- La création du BoxLayout doit contenir une référence vers le composant qui respectera l'agencement.
- Exemple :
  - **BoxLayout** layout = new **BoxLayout**(monComposant,**BoxLayout.X_AXIS**);
  - monComposant.setLayout(layout);

# Les autres layout

- GridBagLayout : il est entièrement paramétrable et donc par conséquent moins aisé d'utilisation.

- FormLayout :
  - Non inclus (directement) dans JAVA
  - Exemple de layout mis en place pour un type d'agencement en particuliers : les formulaires.

# LES ÉVÈNEMENTS

# Ecouter et agir

- Toute action repose sur le principe d'écoute et de réaction.

- Un composant est écouté et si l'écouteur entends un évènement, il essaye de le gérer s'il en est capable.

- Les ActionListener et les ActionEvent.

# Gérer des évènements

- Il faut implémenter le listener capable de gérer l'évènement concerné (clic de souris, touche au clavier, déplacement souris, …)

- Contenu dans java.awt.event (en général)

- Exemple de listener :
  - MouseListener : clic, entrée, passage ou sortie de la souris d'une zone, bouton maintenu, bouton relâché.

# Gérer des évènements

- Les Listener sont des interfaces, il faut :
  - Créer une classe les implémentant
  - Implémenter toutes les méthodes associées (même si une seule nous intéresse ☹)
  - Associer la nouvelle classe au composant concerné

➔ et hop c'est prêt ☺

# Gérer un évènement

- Écrire la classe qui gère l'évènement :
  - Exemple : quitter une application
  - Classe Exit.
  - Si un composant a cette classe comme action associée ➔ un évènement sur ce composant provoquera la fin du programme.

# Exit.java

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

class Exit implements ActionListener
 {
   public void actionPerformed(ActionEvent ev)
        {
              System.exit(0);
        }
 }
```

# Utilisation de Exit

- Association au choix Quitter d'une IHM.
- Utilisation de la méthode addActionListener

```
jmi_Quitter.addActionListener(new Exit ());
```

# Gérer un évènement

- Écrire une classe à part :
  - Utile quand l'action est utilisée par plusieurs classes
  - Contraignant si l'action n'est utilisée que dans une classe plusieurs fois :-(
  - Encore plus contraignant si l'action n'est utilisée qu'une seule fois dans une seule classe :-/

# Gérer un évènement

- Action utilisée plusieurs fois dans une classe :
  - Seule cette classe a besoin de cette action
  - Aucune nécessité que cette action soit accessible en dehors.

➔ classe interne

# Les classes internes

- Non spécifiques au traitement des évènements
- Déclaration à l'intérieur d'une classe au même niveau qu'une méthode
- Classe interne ⇔ type spécifique pour la classe englobante.

# Gérer un évènement

- Action utilisée une seule fois dans une classe :
  - Seule cette classe a besoin de cette action
  - Aucune nécessité que cette action soit accessible autre part même au sein de la classe.
  - Utilité d'un nom ???

    ➔ classe anonyme

# Une classe anonyme

- Elle est :
  - Interne à une classe
  - Définie seulement par son contenu (pas de nom)
  - Dans le cas d'évènements, elle permet d'implémenter une interface en la citant.

# Exemple

- Exemple d'une action liée à un bouton de validation : boutonValider.

- C'est une action liée uniquement à ce bouton.

```
boutonValider.addActionListener(new ActionListener()   {
        public void actionPerformed(ActionEvent ev) {
                //traitement
        }
});
```

# Résultats de compilation

- Classe « maClasse » ➔ *maClasse.class*

- Classe « maClasse » + classe interne « interne »
  - ➔ *maClasse.class*
  - ➔ *maClasse$interne.class*

- Classe « maClasse » + une classe interne anonyme
  - ➔ *maClasse.class*
  - ➔ *maClasse$1.class*