ircam Centre Pompidou

UPMC SORBONNE UNIVERSITÉS

JAVA
- Interface Graphique-

Ninon Devis: ninon.devis@ircam.fr

Philippe Esling: esling@ircam.fr

License 3 Professionnelle - Multimédia

Plan du cours

I. Introduction

II. Base de la programmation graphique avec Swing

III. Programmation Événementielle

IV. Architecture d’un projet avec GUI

Introduction

Abstractions graphiques

- Chaque OS possède sa propre API (Application Programming Interface) pour créer des GUI (graphical user interface).
- Librairies Java:
  - `java.awt` est une première abstraction graphique consistant principalement en des wrappers des composants de systèmes.
  - `java.swing` étend `awt` pour proposer plus de fonctionnalités.
  - `awt` contient des appels spécifiques vers la JVM pour accéder aux opérations graphiques.
  - `swing` est écrit en Java pur.
- *Remarques*:
  - `swing` étend les classes de `awt`.
  - Les classes de `swing` commencent par 'J' (ex: `JFrame`) contrairement à celles de awt (`Frame`).

Introduction à Swing

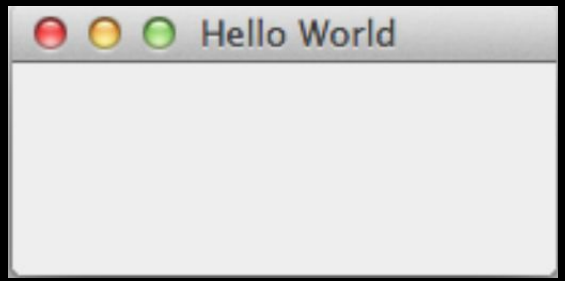
JFrame

Sélectionner JavaSE1.8 lors de la création d'un nouveau projet !

```java
import javax.swing.*;

public class LE380
{
    public static void main(String[] args)
    {
        JFrame window = new JFrame("Hello World");
        window.setSize(200, 100);
        window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        window.setVisible(true);
    }
}
```

Quelle sortie ?



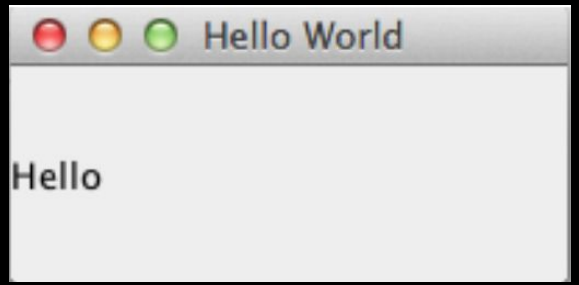
La fenêtre utilise automatiquement le style du système d'exploitation sous-jacent

Introduction à Swing

JLabel

```java
import javax.swing.*;

public class LE380
{
    public static void main(String[] args)
    {
        JFrame window = new JFrame("Hello World");
        window.setSize(200, 100);
        window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        window.add(new JLabel("Hello"));
        window.setVisible(true);
    }
}
```



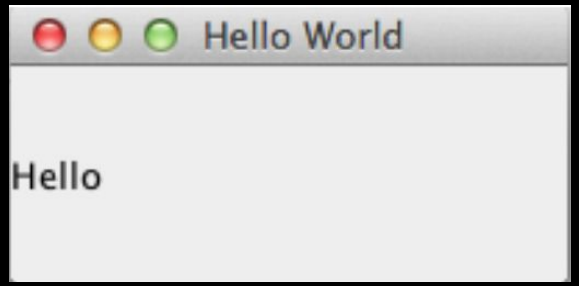
Il est possible d'ajouter des labels c'est à dire du texte dans la fenêtre

Introduction à Swing

JLabel

```java
public static void main(String[] args)
{
    // ...
    window.add(new JLabel("Hello"));
    window.add(new JLabel("World"));
    window.setVisible(true);
}
```

Que se passe t-il dans ce cas?



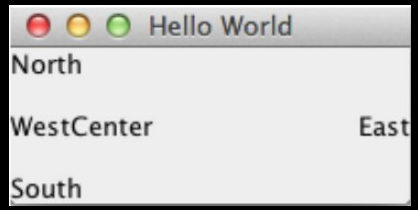
- Un seul label affiché: en réalité les deux sont affichés l'un au dessus de l'autre.
- Lorsqu'on appelle `window.add(label)` successivement il faut que le système choisisse où les afficher.
- Pour éviter de préciser les coordonnées, on utilise un gestionnaire qui arrange les composants automatiquement d'une certaine manière.

Introduction à Swing

Gestionnaire de mise en forme (Layout Manager)

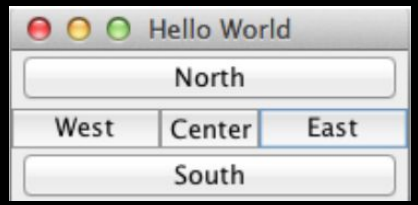
- Le gestionnaire `BorderLayout` est celui par défaut de `JFrame`.
- Les éléments sont organisés selon 5 directions.

```
1 window.add(new JLabel("North"), BorderLayout.NORTH);
2 window.add(new JLabel("South"), BorderLayout.SOUTH);
3 window.add(new JLabel("Center"), BorderLayout.CENTER);
4 window.add(new JLabel("West"), BorderLayout.WEST);
5 window.add(new JLabel("East"), BorderLayout.EAST);
6 window.setVisible(true);
```



- Les labels sont alignés à gauche ou à droite.
- Utiliser `JButton` pour un affichage plus clair: les boutons prennent par défaut un maximum de place.

```
1 window.add(new JButton("North"), BorderLayout.NORTH);
2 window.add(new JButton("South"), BorderLayout.SOUTH);
3 window.add(new JButton("Center"), BorderLayout.CENTER);
4 window.add(new JButton("West"), BorderLayout.WEST);
5 window.add(new JButton("East"), BorderLayout.EAST);
6 window.setVisible(true);
```



Introduction à Swing

Gestionnaire de mise en forme (Layout Manager)

- `FlowLayout`: ajoute des éléments comme du texte, les uns après les autres.

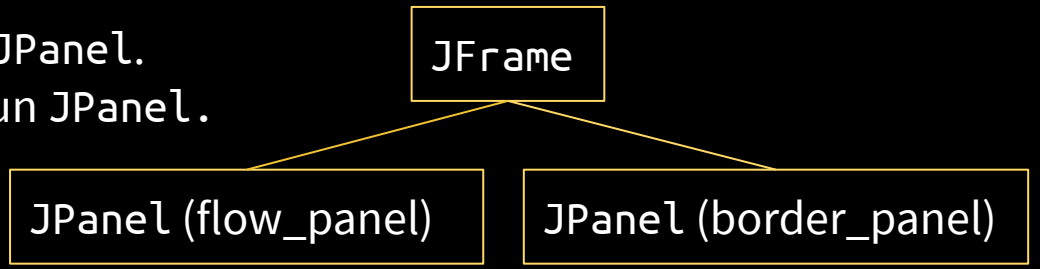
```
1 window.setLayout(new FlowLayout());
2 window.add(new JButton("North"), BorderLayout.NORTH);
3 window.add(new JButton("South"), BorderLayout.SOUTH);
4 window.add(new JButton("Center"), BorderLayout.CENTER);
5 window.add(new JButton("West"), BorderLayout.WEST);
6 window.add(new JButton("East"), BorderLayout.EAST);
7 window.setVisible(true);
```

- `CardLayout`: Dispose les éléments comme une pile de carte, un élément étant visible à la fois.
- `GridLayout`: Dispose les éléments suivant une grille régulière, tous les composants ont la même taille.
- `BoxLayout`: Idem mais permet de régler le nombre de colonne et de ligne.
- `GridBagLayout`: Permet de régler la taille des cellules.

Introduction à Swing

Organisation hiérarchique des composants

- Les différents composants graphiques sont organisés dans une hiérarchie en arbre.
- Il est existe des composants racines (root) qui ne possèdent pas de parents notamment `JFrame` et `JDialog`.
- On peut ajouter des `JPanel` dans des `JPanel`.
- Impossible d'ajouter un `JFrame` dans un `JPanel`.



Quelques bonnes pratiques:

- N'ajouter qu'un composant `JPanel` à la `JFrame`.
- Travailler ensuite dessus en lui ajoutant d'autres composants (`JButton, JLabel`…)
  - En effet, une `JFrame` représente une fenêtre, on pourrait utiliser le même `JPanel` dans une applet par exemple (`JApplet`)
  - Il est possible de faire des dessins géométriques dans un `JPanel` mais pas dans une `JFrame`. Il faut forcément lui ajouter un `JPanel avant.`

Introduction à Swing

Organisation hiérarchique des composants

- La méthode `JFrame.validate()` doit être appelée si la fenêtre est visible et que vous avez modifié les composants.

- Autres composants graphiques:
  - `JTextField` : Champs de texte, pour saisir des informations de l'utilisateur.
  - `JCheckBox` : Cases à cocher.
  - `JRadioButton` : Boutons radio.
  - `JList` : Liste d'éléments à sélectionner.
  - `JScrollPane` : Ajoute un "scroll" à un composant.
  - `JComboBox` : Liste d'éléments, on peut en sélectionner un seul.
    - Ils héritent tous de la classe JComponent sauf les composants racines (JFrame…)

Programmation événementielle

- La console est *linéaire*, l'utilisateur ne peut faire que ce qui lui ai demandé.
- Une GUI a plusieurs boutons, on ne sait pas à l'avance où l'utilisateur va cliquer.
- Besoin d'un nouveau *paradigme* de programmation: la programmation événementielle.

- L'idée est d'associer *à l'avance* des actions aux différents éléments graphiques.
- Lorsque l'utilisateur clique sur un bouton (*événement*), l'*action associée est invoquée*.
- Une action est implémentée sous forme de classes:

➔ L'utilisateur dirige le flux d'exécution du programme et non pas le programme qui dirige l'utilisateur.

Programmation événementielle

Les Listeners (écouteurs)

- Classe associée à un composant graphique.
- Ses méthodes seront appelées lorsqu'un événement aura lieu (comme cliquer sur un bouton, compléter un champ texte…)
- Plusieurs types de Listener:
  - `ActionListener`, événements spécifiques à un composant.
  - `MouseListener`, événements de la souris.
  - `KeyboardListener`, événements du clavier.
  - …

Programmation événementielle

ActionListener

```java
// ... in main
JButton b = new JButton("click me");
b.addActionListener(new ClickMe());
// ...

class ClickMe implements ActionListener
{
    public void actionPerformed(ActionEvent e)
    {
        System.out.println("Hello");
    }
}
```

- La méthode `actionPerformed` est appelée à chaque fois que l'utilisateur clique sur le bouton.
- `ActionEvent` contient des données sur l'événement, à l'instar de :
  - `getActionCommand()` : Une chaîne de caractère décrivant l'action, pour un bouton ça sera son label.
  - `getModifier()` : Un entier indiquant si une (ou plusieurs) touche de contrôle (alt/ctrl/. . .) était pressée quand l'événement à eu lieu.

Programmation événementielle *ActionListener*

- L'interface `MouseListener` propose 5 méthodes représentant les différentes actions d'une souris.

```java
// ... in main
JPanel main = new JPanel(new FlowLayout());
main.addMouseListener(new PrintCoordinate());
// ...

class PrintCoordinate implements MouseListener
{
    public void mouseClicked(MouseEvent e)
    {
        System.out.println(e.getPoint());
    }
    public void mousePressed(MouseEvent e) {}
    public void mouseReleased(MouseEvent e) {}
    public void mouseEntered(MouseEvent e) {}
    public void mouseExited(MouseEvent e) {}
}
```

- L'objet passé en paramètre, `MouseEvent`, contient entre autres:
  - `getClickCount()`, nombre de clics pour cet événement
  - `getPoint()`, les coordonnées du clic.
  - …

Programmation événementielle

Autres Listener

- `KeyListener` pour les touches du clavier.
- `ItemListener` pour les listes d'éléments.
- …

- Plein de listener spécifiques aux composants graphiques.
- Les composants possèdent des fonctions addXXXXListener pour les listeners supportés.
- En fonction du composant, consulter:

https://docs.oracle.com/javase/tutorial/uiswing/components/componentlist.html

Architecture d’un projet avec GUI

- Afin de coder proprement il faut que la GUI et la logique du code soit bien séparées.
- Permet de ne pas tout ré-implémenter si la partie UI est modifiée.
- Différentes UI: Console, Swing, Android, Windows (Phone & OS), …

➔ Par où commencer ?

```
1 public class WarCardGame
2 {
3     public static void main(String[] args)
4     {
5         GameGUI game_ui = new GameGUI ();
6         game_ui.start();
7     }
8 }
```

L’interface graphique (ici la classe `GameGUI`) est d’abord créée et est démarrée dans le main.

Architecture d'un projet avec GUI

Les classes métiers

- Les classes métiers représentent les données et ne contiennent généralement pas d'I/O:

➔ Elles ne manipulent jamais directement d'objets `Swing`.

```
1 public class GameGUI
2 {
3     private final Game game = new Game();
4     private final JButton next_card = new JButton("next");
5     // ...
6 }
```

- C'est la GUI qui instancie la classe métier, ici `Game`.
- Séparer l'interface graphique des classes métiers :
  - `upmc.wcg.WarCardGame` : contient le main.
  - `upmc.wcg.ui.*` : contient toutes les classes de l'interface graphique.
  - `upmc.wcg.game.*` : contient toutes les classes métiers faisant les calculs sur les données du jeu.
- Astuce: Si une classe ou une méthode peut être ré-utilisée dans avec une autre interface graphique alors c'est une classe métier, sinon une classe d'UI.

Architecture d’un projet avec GUI

Flux d’exécution du code

Si l’utilisateur veut faire avancer le jeu d’une étape et clique sur *next*:

1. La méthode `actionPerformed` du listener écoutant le bouton *next* est exécutée.
2. L’appel est transféré à la classe `Game`.
3. Celle-ci calcule la prochaine étape et retourne le résultat de la manche.
4. L’interface graphique récupère le résultat et met à jour les éléments correspondants.

Architecture du projet:

- Séparer l’interface graphique des classes métiers :
  - `upmc.wcg.WarCardGame` : contient le main.
  - `upmc.wcg.ui.*` : contient toutes les classes de l’interface graphique.
  - `upmc.wcg.game.*` : contient toutes les classes métiers faisant les calculs sur les données du jeu.