Cours 5 – Interface homme-interface

Programmation objets, web et mobiles en Java
Licence 3 Professionnelle - Multimédia

Pierre Talbot (talbot@ircam.fr)

Doctorant UPMC/IRCAM

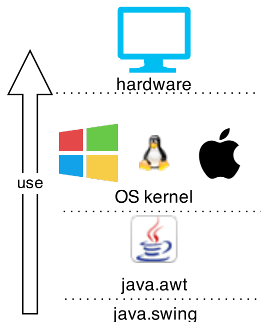
4 novembre 2014

Le menu

- Introduction
- Base de la programmation graphique avec Swing
- Programmation événementielle
- Architecture d'un projet avec GUI

Abstractions graphiques

1. Chaque OS possède sa propre API pour créer des GUI (*graphical user interface*).
2. Les frameworks comme Qt (C++) ou Awt/Swing (Java) propose une interface uniforme pour tous les OS.



Librairies Java

- `java.awt` est une première abstraction graphique consistant principalement en des *wrappers* des composants 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.

Librairies Java

- `java.awt` est une première abstraction graphique consistant principalement en des *wrappers* des composants 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` donc on ne l'utilise pas sans `awt` !
- Les classes de `swing` commence par 'J' (`JFrame`) contrairement à celles de `awt` (`Frame`).

Le menu

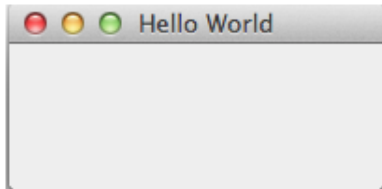
- Introduction
- Base de la programmation graphique avec Swing
  - Gestionnaire de mise en forme
  - Organisation hiérarchique des composants
- Programmation événementielle
- Architecture d'un projet avec GUI

JFrame

```
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);
  }
}
```

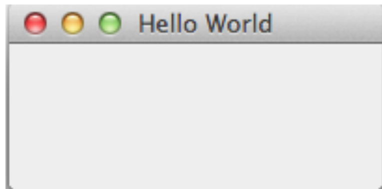
JFrame

```
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);
  }
}
```



JFrame

```
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);
  }
}
```

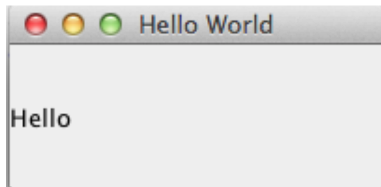


La fenêtre utilise automatiquement le style du système d'exploitation sous-jacent (JFrame hérite de awt.Frame encapsulant l'objet système).

JLabel

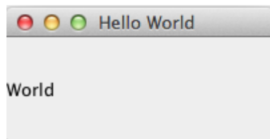
On peut ajouter des labels, c'est-à-dire du texte dans la fenêtre.

```
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);
  }
}
```



Deux JLabel ?

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



Problème

- Il n'y a qu'un seul label d'affiché...
- En vérité, les deux sont affichés l'un au dessus de l'autre et donc, un est caché.

Le menu

- Introduction
- Base de la programmation graphique avec Swing
  - Gestionnaire de mise en forme
  - Organisation hiérarchique des composants
- Programmation événementielle
- Architecture d'un projet avec GUI

Gestionnaire de mise en forme (*Layout Manager*)

Problématique

- Quand on appelle `window.add(label)` successivement il faut bien que le système choisisse où les afficher.
- Néanmoins, en tant que développeur, on ne veut pas préciser leur emplacement exact avec des coordonnées...
- Il y a donc des gestionnaires qui arrange les composants automatiquement d'une certaine manière.

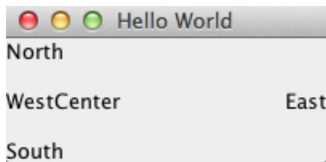
BorderLayout avec JLabel

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

BorderLayout avec JLabel

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

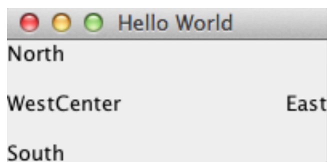
```
window.add(new JLabel("North"), BorderLayout.NORTH);
window.add(new JLabel("South"), BorderLayout.SOUTH);
window.add(new JLabel("Center"), BorderLayout.CENTER);
window.add(new JLabel("West"), BorderLayout.WEST);
window.add(new JLabel("East"), BorderLayout.EAST);
window.setVisible(true);
```



BorderLayout avec JLabel

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

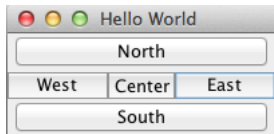
```
window.add(new JLabel("North"), BorderLayout.NORTH);
window.add(new JLabel("South"), BorderLayout.SOUTH);
window.add(new JLabel("Center"), BorderLayout.CENTER);
window.add(new JLabel("West"), BorderLayout.WEST);
window.add(new JLabel("East"), BorderLayout.EAST);
window.setVisible(true);
```



On ne voit pas bien les différentes zones car les labels sont alignés à gauche ou droite.

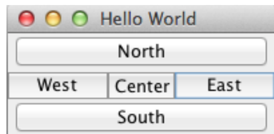
BorderLayout avec JButton

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



BorderLayout avec JButton

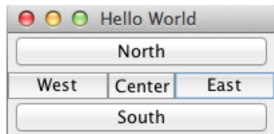
```
window.add(new JButton("North"), BorderLayout.NORTH);
window.add(new JButton("South"), BorderLayout.SOUTH);
window.add(new JButton("Center"), BorderLayout.CENTER);
window.add(new JButton("West"), BorderLayout.WEST);
window.add(new JButton("East"), BorderLayout.EAST);
window.setVisible(true);
```



- Contrairement aux labels, les boutons prennent par défaut le maximum de place.
- On peut donc maintenant dire pourquoi ajouter deux labels à la suite ne marche pas.

BorderLayout avec JButton

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



- Contrairement aux labels, les boutons prennent par défaut le maximum de place.
- On peut donc maintenant dire pourquoi ajouter deux labels à la suite ne marche pas.
- Car les deux labels sont par défaut ajoutés à la zone `BorderLayout.CENTER` et se superposent.

FlowLayout

- Ajoute des éléments comme du texte : les uns après les autres.
- Par défaut celui de `JPanel`.

FlowLayout

- Ajoute des éléments comme du texte : les uns après les autres.
- Par défaut celui de `JPanel`.

```
window.setLayout(new FlowLayout());
window.add(new JButton("North"), BorderLayout.NORTH);
window.add(new JButton("South"), BorderLayout.SOUTH);
window.add(new JButton("Center"), BorderLayout.CENTER);
window.add(new JButton("West"), BorderLayout.WEST);
window.add(new JButton("East"), BorderLayout.EAST);
```

Autres gestionnaires de mise en forme

Suivant comment on veut organiser les différents composants, il existe différents gestionnaires permettant plus ou moins de libertés.

- `CardLayout` : Disposer les éléments comme une pile de carte, un élément étant visible à la fois.

Autres gestionnaires de mise en forme

Suivant comment on veut organiser les différents composants, il existe différents gestionnaires permettant plus ou moins de libertés.

- `CardLayout` : Disposer les éléments comme une pile de carte, un élément étant visible à la fois.
- `GridLayout` : Disposer les éléments comme une grille régulière, tous les composants ont la même taille.

Autres gestionnaires de mise en forme

Suivant comment on veut organiser les différents composants, il existe différents gestionnaires permettant plus ou moins de libertés.

- `CardLayout` : Disposer les éléments comme une pile de carte, un élément étant visible à la fois.
- `GridLayout` : Disposer les éléments comme une grille régulière, tous les composants ont la même taille.
- `BoxLayout` : Idem `GridLayout` mais permet de régler le nombre de colonne et ligne (plus souple).

Autres gestionnaires de mise en forme

Suivant comment on veut organiser les différents composants, il existe différents gestionnaires permettant plus ou moins de libertés.

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

Autres gestionnaires de mise en forme

Suivant comment on veut organiser les différents composants, il existe différents gestionnaires permettant plus ou moins de libertés.

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

Mots-clés : *Layout manager*

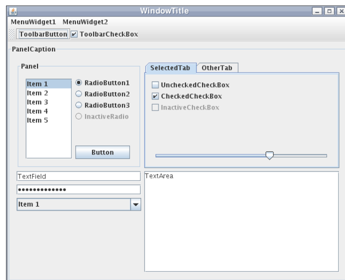
Le menu

- Introduction
- Base de la programmation graphique avec Swing
  - Gestionnaire de mise en forme
  - Organisation hiérarchique des composants
- Programmation événementielle
- Architecture d'un projet avec GUI

JPanel

Problématique

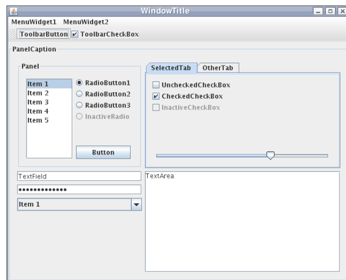
Un unique gestionnaire de mise en forme n'est *pas* suffisant pour une interface classique.



`JPanel`

Problématique

Un unique gestionnaire de mise en forme n'est *pas* suffisant pour une interface classique.



Solution

On utilise alors des `JPanel` qui sont des composants contenus dans une `JFrame` et qui ont leur propre gestionnaire de mise en forme.

JPanel

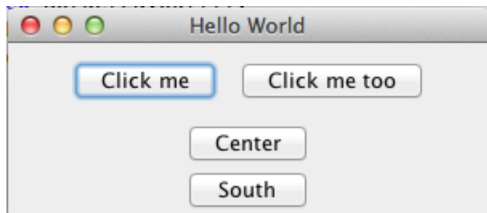
```
JPanel flow_panel = new JPanel(new FlowLayout());
flow_panel.add(new JButton("Click me"));
flow_panel.add(new JButton("Click me too"));
```

JPanel

```
JPanel flow_panel = new JPanel(new FlowLayout());
flow_panel.add(new JButton("Click me"));
flow_panel.add(new JButton("Click me too"));

JPanel border_panel = new JPanel(new BorderLayout());
border_panel.add(new JButton("South"), BorderLayout.SOUTH);
border_panel.add(new JButton("Center"), BorderLayout.CENTER);
```

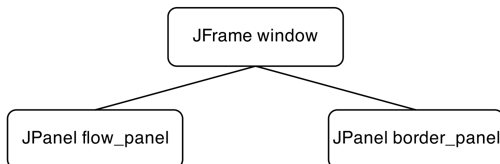
JPanel

```
JPanel flow_panel = new JPanel(new FlowLayout());
flow_panel.add(new JButton("Click me"));
flow_panel.add(new JButton("Click me too"));

JPanel border_panel = new JPanel(new BorderLayout());
border_panel.add(new JButton("South"), BorderLayout.SOUTH);
border_panel.add(new JButton("Center"), BorderLayout.CENTER);

JFrame window = new JFrame("Hello world");
window.setLayout(new FlowLayout());
window.add(flow_panel);
window.add(border_panel);
```



Organisation hiérarchique des éléments graphiques

- Les différents *composants* graphiques sont organisés dans une hiérarchie en arbre.
- Il existe des composants *racines* (*root*) ne possédant pas de parent, notamment `JFrame` et `JDialog`.
- On peut ajouter des `JPanel` dans des `JPanel`.



Quelques bonnes pratiques

- Généralement, on n'ajoute qu'un composant `JPanel` à la `JFrame`.
- On travaille ensuite dessus en lui ajoutant d'autres composants (`JFrame`, `JLabel`, ...).

Quelques bonnes pratiques

- Généralement, on n'ajoute qu'un composant `JPanel` à la `JFrame`.
- On travaille ensuite dessus en lui ajoutant d'autres composants (`JFrame`, `JLabel`, ...).
- On fait ça car une `JFrame` représente une fenêtre, on pourrait utiliser le même `JPanel` dans une applet par exemple (`JApplet`).
- En plus on peut faire des dessins (géométriques) dans un `JPanel` mais pas dans une `JFrame`.

Quelques bonnes pratiques

- Généralement, on n'ajoute qu'un composant `JPanel` à la `JFrame`.
- On travaille ensuite dessus en lui ajoutant d'autres composants (`JFrame`, `JLabel`, ...).
- On fait ça car une `JFrame` représente une fenêtre, on pourrait utiliser le même `JPanel` dans une applet par exemple (`JApplet`).
- En plus on peut faire des dessins (géométriques) dans un `JPanel` mais pas dans une `JFrame`.

`validate()`

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.

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.

Note : Ils héritent tous de la classe `JComponent` sauf les composants racines (`JFrame`, . . .).
Voir `http://docs.oracle.com/javase/tutorial/uiswing/components/jcomponent.html`

Le menu

- Introduction
- Base de la programmation graphique avec Swing
- Programmation événementielle
- Architecture d'un projet avec GUI

De la console aux GUIs

- La console est linéaire, l'utilisateur ne peut faire que ce qui lui est demandé.
- Exemple : `"Veuillez entrer votre choix :"`.

De la console aux GUIs

- La console est linéaire, l'utilisateur ne peut faire que ce qui lui est demandé.
- Exemple : `"Veuillez entrer votre choix :"`.
- Une GUI a plusieurs boutons donc on ne sait pas à l'avance où l'utilisateur va cliquer.
- On a besoin d'un nouveau *paradigme* de programmation : *La programmation événementielle*.

Programmation événementielle

- L'idée est simple : on associe *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.

Programmation événementielle

- L'idée est simple : on associe *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.

Programmation événementielle

L'utilisateur dirige le flux d'exécution du programme, ce n'est pas le programme qui dirige l'utilisateur (comme en console).

Les écouteurs (*Listener*)

Listener

- C'est une classe qu'on associe à un composant graphique.
- Ses méthodes seront appelées lorsqu'un événement aura lieu (cliquer sur un bouton, compléter un champ texte, . . .).

Les écouteurs (*Listener*)

Listener

- C'est une classe qu'on associe à un composant graphique.
- Ses méthodes seront appelées lorsqu'un événement aura lieu (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.
- ...

ActionListener

```
// ... 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");
  }
}
```

ActionListener

```
// ... 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.

ActionListener

```
// ... 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 :
  1. `getActionCommand()` : Une chaîne de caractère décrivant l'action, pour un bouton ça sera son label.
  2. `getModifier()` : Un entier indiquant si une (ou plusieurs) touche de contrôle (alt/ctrl/...) était pressée quand l'événement à eu lieu.
  3. ...

MouseListener

```
// ... 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'interface MouseListener propose 5 méthodes représentants les différentes actions d'une souris.

MouseListener

```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'interface `MouseListener` propose 5 méthodes représentants les différentes actions d'une souris.
- L'objet passé en paramètre, `MouseEvent`, contient entre autres :
  1. `getClickCount()`, nombre de cliques pour cet événement.
  2. `getPoint()`, les coordonnées du clique.
  3. ...

D'autres `Listenner`

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

D'autres Listenner

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

Trouver un `Listener`

- Plein de `Listener` spécifiques aux composants graphiques.
- Les composants possèdent des fonctions `addXXXXListener` pour les *listeners* supportés.
- En fonction du composant, on consulte un *how-to* (`docs.oracle.com/javase/tutorial/uiswing/components/componentlist.html`).

Le menu

- Introduction
- Base de la programmation graphique avec Swing
- Programmation événementielle
- Architecture d'un projet avec GUI

Séparer l'interface graphique et le code

- Pour coder proprement, on veut que la GUI et la logique du code soit bien séparé.
- Ça permet notamment de ne pas devoir tout ré-implémenter si on change la partie UI.
- Différentes UI : Console, Swing, Android, Windows (Phone et OS),. . .

Par où commencer ?

```
public class WarCardGame {
  public static void main(String[] args) {
    GameGUI game_ui = new GameGUI();
    game_ui.start();
  }
}
```

Typiquement l'interface graphique (ici la classe `GameGUI`) est d'abord créée et démarrée dans le `main`.

Les classes métiers

- Les classes métiers représentent les données et ne contiennent généralement pas d'I/O.
- Par conséquent elles ne manipulent *jamais* directement d'objets Swing.

```
public class GameGUI {
  private final Game game = new Game();
  private final JButton next_card = new JButton("next");
  // ...
}
```

C'est la GUI qui instancie la classe métier, ici `Game`.

Flux d'exécution du code

Situation

L'utilisateur veut faire avancer le jeu d'une étape et clique sur *next*.

Flux d'exécution du code

Situation

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.

Flux d'exécution du code

Situation

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 *forward* à la classe `Game`.

Flux d'exécution du code

Situation

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 *forward* à la classe `Game`.
3. Celle-ci calcule la prochaine étape et retourne le résultat de la manche.

Flux d'exécution du code

Situation

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 *forward* 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.

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.

Astuce

- Il arrive qu'on se demande si une classe ou méthode doit être dans les classes métiers ou de GUI.
- Si elle peut être ré-utilisée avec une autre interface graphique, comme la console, alors c'est une classe métier, sinon une classe d'UI.

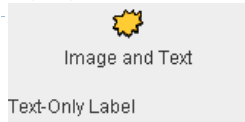
Caractéristiques de Swing

- Le tutorial Swing officiel est LA référence en la matière:

http://download.oracle.com/javase/tutorial/uiswing/TOC.html

- Propose divers composants de haut niveau standards
- Extensible : on peut étendre les composants de base pour affiner leur comportement (affichage, événements)
- Les éléments de base:
  - JFrame : une fenêtre graphique, contient un JPanel (son contentPane)
  - JPanel : un conteneur d'autres objets, permet de décider de l'agencement des objets (cf. Layout)
  - Component : les divers contrôles d'IHM

Les composants usuels de saisie



- JLabel : un texte (étiquette) non éditable

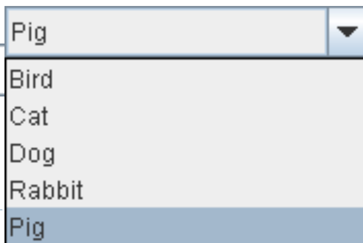
This is an editable JTextArea. A text area is a "plain" text component, which means that although it can display text in any font, all of the text is in the same font.

- JTextField, JTextArea : zones de saisie de texte

City: Santa Rosa

- JPasswordField : saisie cachée

Enter the password: ••••••••



- JComboBox : saisie assistée

Présentation d'information structurée

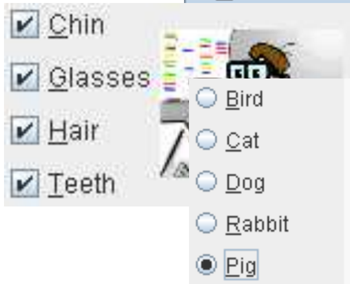
- Liste d'éléments (sélectionnables): JList
- Tableau d'éléments, on peut configurer la façon dont chaque cellule est affichée: JTable
- Vue arborescente : JTree

Host	User	Password	Last Modified
Biocca Games	Freddy	!#asf6Awwzb	Mar 16, 2006
zabble	ichabod	Tazb!34$fZ	Mar 6, 2006
Sun Developer	fraz@hotmail.co...	AasW541!fbZ	Feb 22, 2006
Heirloom Seeds	shams@gmail....	bkz[ADF78!	Jul 29, 2005
Pacific Zoo Shop	seal@hotmail.c...	vbAf124%z	Feb 22, 2006

Les contrôles

- JButton : un bouton cliquable, texte et/ou image

- Choix parmi une liste : JRadioButton, JCheckBox

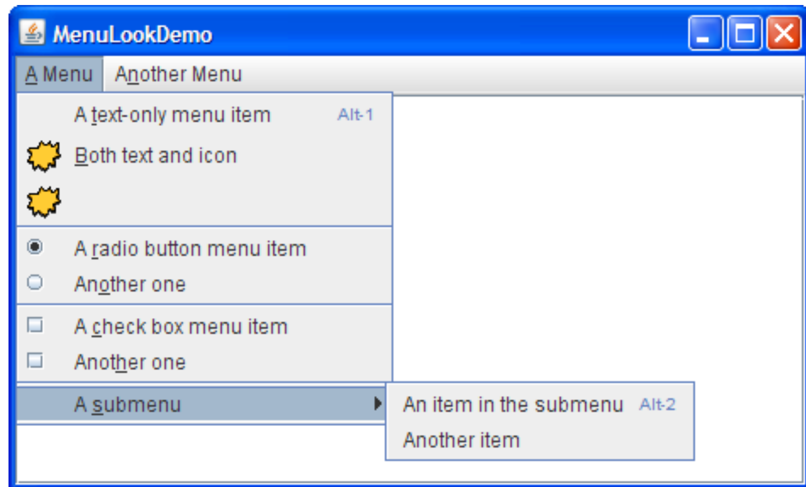


- Choix sur un intervalle :

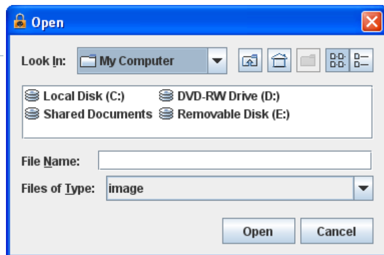
JSlider, JSpinner

Les menus

- JMenuBar, JMenu et JMenuItem



Les dialogues standard



- JFileChooser : choix d'un fichier
- JOptionPane pour créer des

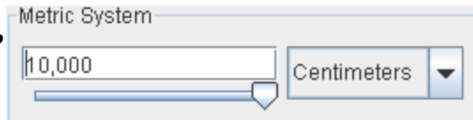
 interactions modales simples :

e.g.

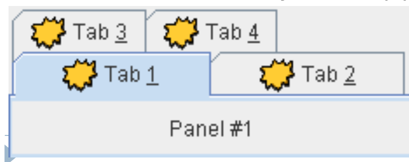
```
JOptionPane.showMessageDialog(frame,
   "Eggs are not supposed to be green.",
   "Inane error",
   JOptionPane.ERROR_MESSAGE);
```

Agencement des composants : JPanel

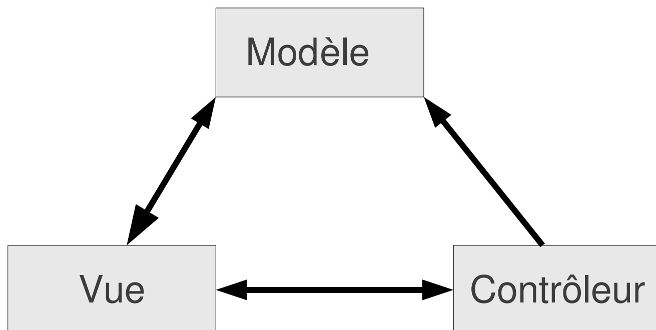
- JPanel : le conteneur de base, muni d'un Layout, décide de la disposition



- JScrollPane: si le contenu est trop gros, s'adapte à la taille
- JTabbedPane, JSplitPane : conteneurs de JPanel(s)

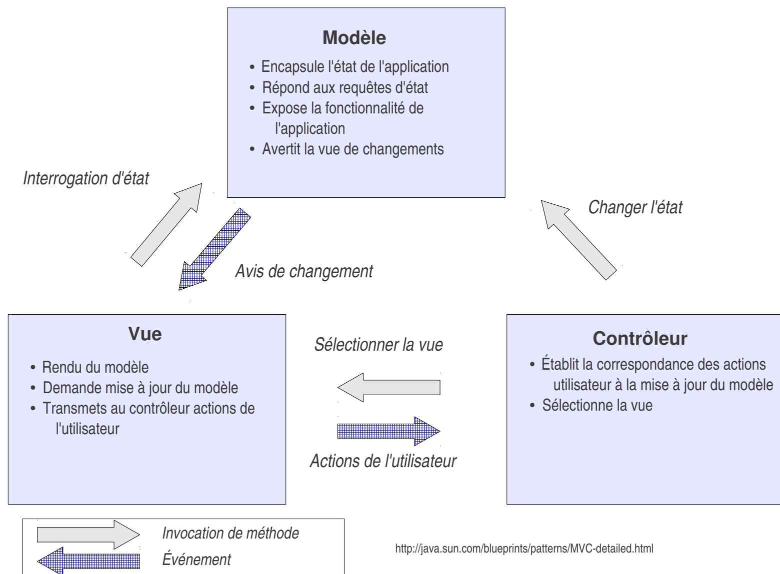


Architecture modèle-vue-contrôleur



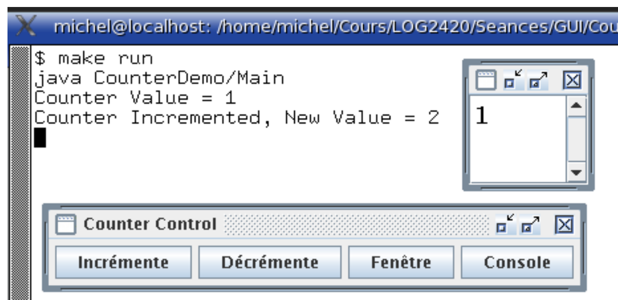
Architecture modèle-vue-contrôleur

Détails



ÉCOLE POLYTECHNIQUE MONTRÉAL

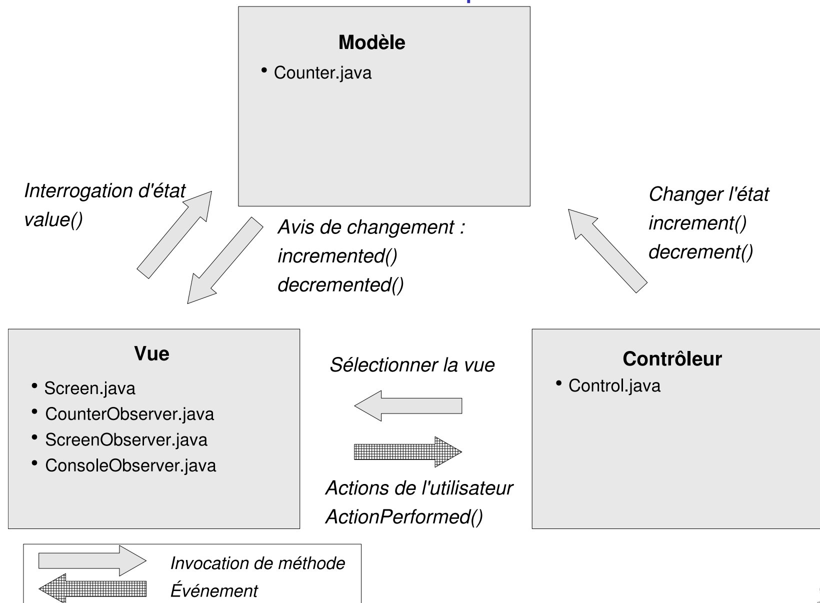
Le “Counter Example”



- Le programme Java CounterDemo [1] illustre deux vues du modèle du compteur
  - une vue “console”
  - une seconde où le compteur s’affiche dans une fenêtre

1. Cet exemple est adapté de `http://courses.csail.mit.edu/6.170/old-www/2006-Fall/lectures/lectures.html`.

Modèle-vue-contrôleur de l'exemple “Counter demo”



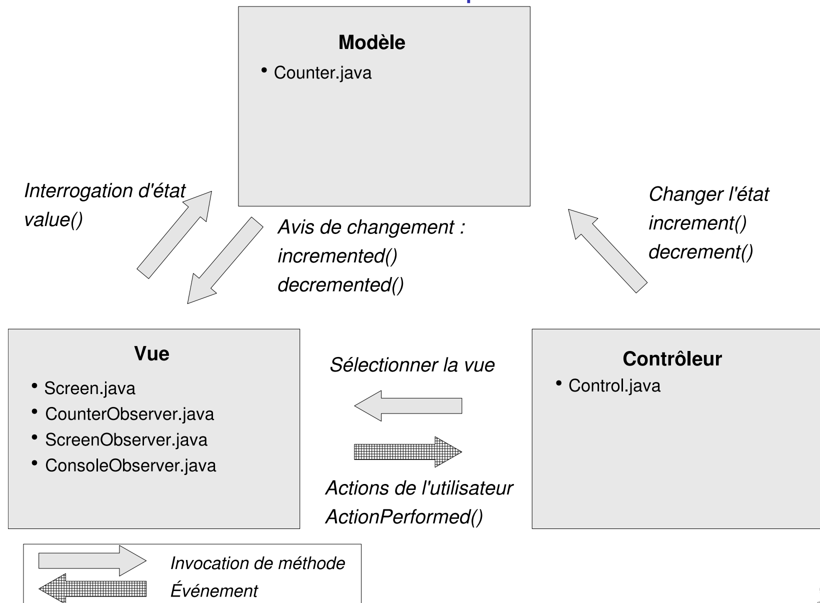
ÉCOLE POLYTECHNIQUE MONTRÉAL

Counter.java

```
public class Counter {
   int value = 0;
   CounterObserver observer;

   public void setObserver(CounterObserver co) {
         observer = co;
         co.value();
   }
   public void increment() {
         value++;
         observer.incremented();
   }
   public void decrement() {
         value--;
         observer.decremented();
   }
   public int value() { return value; }
}
```

ÉCOLE POLYTECHNIQUE MONTRÉAL

Modèle-vue-contrôleur de l'exemple “Counter demo”



Control.java

Initialisation

```
public class Control extends JPanel implements ActionListener {
    protected JButton b1, b2, b3, b4;
    CounterObserver screen, console;
    protected Counter counter;
    Worker worker;

    public void initialize() {
        b1 = new JButton("Incrémente");
        b1.setActionCommand("increment");
        b1.addActionListener(this);
        b1.setToolTipText("Incrémente le compteur");

        //Ajouter les composants à ce conteneur en utilisant les
FlowLayout par défaut
        add(b1);
    }

    }
}
```


ÉCOLE POLYTECHNIQUE MONTRÉAL

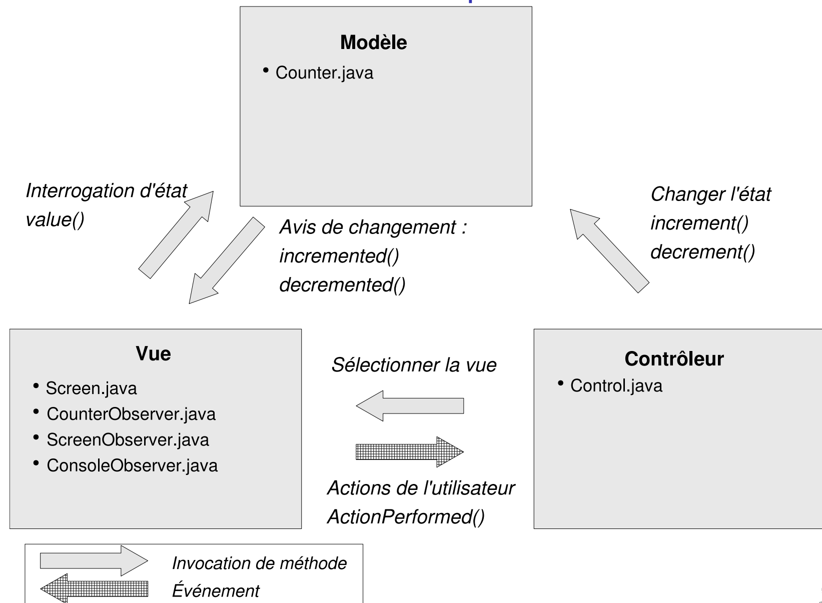
Control.java

Gestion des événements

```
public class Control extends JPanel implements ActionListener {
    protected JButton b1, b2, b3, b4;
    CounterObserver screen, console;
    protected Counter counter;
    Worker worker;
(...)
    public void actionPerformed(ActionEvent e) {
        if ("increment".equals(e.getActionCommand())) {
        counter.increment();
        } else if ("decrement".equals(e.getActionCommand())) {
        counter.decrement();
        } else if ("screen".equals(e.getActionCommand())) {
        counter.setObserver(screen);
        } else if ("console".equals(e.getActionCommand())) {
        counter.setObserver(console);
        }
    }
(...)
}
```

ÉCOLE POLYTECHNIQUE MONTRÉAL

Modèle-vue-contrôleur de l'exemple “Counter demo”



ÉCOLE POLYTECHNIQUE MONTRÉAL

CounterObserver.java

```
interface CounterObserver {
    public void value();
    public void incremented();
    public void decremented();
}
```

ÉCOLE POLYTECHNIQUE MONTRÉAL

ConsoleObserver.java

```
class ConsoleObserver implements CounterObserver {
    Counter counter;
    ConsoleObserver(Counter c) {
        counter = c;
    }
    public void value() {
        System.out.print("Counter Value = ");
        System.out.print(counter.value());
        System.out.print("\n");
    }
    public void incremented() {
        System.out.print("Counter Incremented, New Value =");
        System.out.print(counter.value());
        System.out.print("\n");
    }
    public void decremented() {
        System.out.print("Counter Decremented, New Value =");
        System.out.print(counter.value());
        System.out.print("\n");
    }
```

ScreenObserver.java

```
class ScreenObserver implements CounterObserver {
    Screen screen;
    Counter counter;
    ... /* constructeur */
    public void value() {
        String text = Integer.toString(counter.value());
        screen.setText(text);
    }
    public void incremented() {
        String text = Integer.toString(counter.value());
        screen.setText(text);
    }
    public void decremented() {
        String text = Integer.toString(counter.value());
        screen.setText(text);
    }
}
```

ÉCOLE POLYTECHNIQUE MONTRÉAL

Le MVC et la réalité

Tiré d'un article sur l'architecture de Swing :

- "We quickly discovered that this split didn't work well in practical terms because the view and controller parts of a component required a tight coupling (for example, it was very difficult to write a generic controller that didn't know specifics about the view). So we collapsed these two entities into a single UI (user-interface) object," (Java Swing Architecture, `www.oracle.com/technetwork/java/architecture-142923.html`)

La solution appliquée en pratique :

- Le contrôleur et la vue sont souvent regroupés

Rappel des concepts

- L'architecture MVC vise à séparer le modèle de la vue et du comportement
- Le patron "observateur" (*observer*, apparenté au *listener* en Java) est à la base de la modularité et permet de changer de vue dans l'exemple présenté

ÉCOLE POLYTECHNIQUE MONTRÉAL