GUI (Graphical User Interfaces)

1 Principes généraux d'interaction

1.1 découplage application / interfaces : Modèles et (Vues-Contrôleurs)

Modèle : objet "métier",

Vue : objet réalisant une présentation graphique d'un objet métier, une vue doit refléter correctement l'état de l'objet métier

Contrôleur : objet responsable de la communication entre un utilisateur et une vue (par exemple ordre de réaffichage) ou entre un utilisateur et l'objet métier (ordre fonctionnel).

Les vues réalisent des affichages en fonction de l'état des modèles.

Les vues évoluent en fonction de l'état des modèles et des désirs des utilisateurs transmis via les contrôleurs.

Les modèles évoluent en fonction de leur logique interne et des ordres transmis vie les contrôleurs.

1.2 Principe d'indépendance

Dans une application,

plus la gestion des objets métiers est réalisée de façon indépendante de l'interface graphique,

plus l'application métier et l'interface seront faciles à maintenir et faire évoluer,

plus interface sera facile à changer.

Réalisation du principe d'indépendance Schéma de conception observateur - Communication par évènements.

2 Interfaces en Java : AWT versus SWING

AWT (Abstract Window Toolkit) produit des interfaces dont le code est indépendant du système d'exécution mais qui produit des résultats

Swing est basée sur AWT, est plus riche et est indépendante des machines.

3 Premier schémas d'interfacage MVC : l'interface pilote l'objet métier

Le modèle est indépendant de l'interface mais il est piloté par l'interface qui sait donc quand il lui arrive quelque chose.

On distingue

- le modèle : l'objet ou les objets métiers.
- l'interface représentée par un objet complexe réalisant à la fois la vue et le contrôleur. La partie vue de l'interface réalise une présentation graphique de l'objet métier. La partie contrôleur de l'interface est à l'écoute des ordres de l'utilisateur, les transmets à l'objet métier et gère le réaffichage.

Exemple : un compteur.

3.1 La partie "vue" de l'interface

Une vue est matérialisée sur un écran par un objet graphique de type *Component*.

Les composants GUI sont des composants simples ou des conteneurs de composants.

Un conteneur est constitué d'autres composants graphiques (schéma de conception).

```
java.lang.Object
  +--java.awt.Component
    +--java.awt.Container
      +--java.awt.Window (fenêtre SE sans cadre ni menu)
        +--java.awt.Frame implements MenuContainer (cadre et menu)
      +--java.awt.panel (conteneur pouvant être contenu - flowLayout)
      +--java.awt.box (idem - BoxLayout)
    +--java.awt.Button (composant simple)
    +--java.awt.Canvas (composant simple pour dessiner - macpaint)
    +--java.awt.Checkbox
    +--java.awt.Choice
+ ...
```

Exemple de classes de composants : Button, Canvas, CheckBox, Choice, List, TextArea, TextField, Label, Scrollbar.

3.2 Les Présentoires (Layout)

Les instances des sous-classes de layout définissent différentes possibilités de présentation géographique (disposition) des composants à l'intérieur d'un conteneur.

Tout conteneur doit posséder un présentoire de composants.

```
    aPanel.setLayout(new FlowLayout());
```

Différents types de présentoires.

- FlowLayout() : en ligne de gauche à droite.
- GridLayout(nbLignes, nbCols, largeurCols, hauteurLignes) - ajout gauche droite puis haut bas.
- BoxLayout - alignements verticaux ou horizontaux
- GridBagLayout - complexe (VPL)

3.3 L'exemple

```
package gui;
import standard.Compteur;
import java.awt.*;
import java.awt.event.*;

public class CompteurAWT1 extends Frame implements ActionListener{

  Compteur cpt;   Button b1, b2, b3;   Label l;

  public CompteurAWT1()  {this(new Compteur());
  }

  public CompteurAWT1(Compteur c){
    super("Un Compteur");
    //Un flow layout insère les composants de gauche à droite
    this.setLayout(new FlowLayout());
    b1 = new Button("raz");
```

```
    b1.addActionListener(this);
    this.add(b1);
    b2 = new Button("incr");
    b2.addActionListener(this);
    this.add(b2);
    b3 = new Button("decr");
    b3.addActionListener(this);
    this.add(b3);
    cpt = c;
    l = new Label(String.valueOf(cpt.getValue()), Label.CENTER);
    this.add(l);
  }

  public static void main(String[] args){
    Frame w = new CompteurAWT1();
    w.setVisible(true);
    w.pack();
    w.show();
  }
```

3.4 Gestion des ordres de l'utilisateur - Les contrôleurs

Les ordres de l'utilisateur (clavier, souris) sont captés au niveau de la machine virtuelle et transmis aux programmes via un protocole dédié dit "protocole de communication par évènements".

L'utilisation de ce protocole n'est pas limitée à la réalisation d'interfaces.

La communication par évènements - Première vision.

Le protocole d'écoute.

Tout objet souhaitant être prévenu de l'occurence d'un type d'évènement doit

1. s'inscrire comme écouteur de ce type d'évènements auprès d'un objet qui les émets,
2. être en mesure d'exécuter une méthode correspondant à ce type d'évènements.

Exemple de sortes d'écouteurs corresponsant à différents évènements.

Act that results in the event	Listener type
User clicks buton, presses Return, or chooses a menu item	ActionListener
User closes a frame (main window)	WindowListener
User presses a mouse button while the cursor is over a component	MouseListener
User moves the mouse over a component	MouseMotionListener
Component becomes visible	ComponentListener
Component gets the keyboard focus	FocusListener
Table or list selection changes	ListSelectionListener

3.5 Exemple

```
public class CompteurAWT1 extends Frame implements ActionListener{

  public void actionPerformed(ActionEvent e) {
    // alternative : (Button)(e.getSource()) == b1
    if (e.getActionCommand() == "raz")
      {cpt.raz();}
    if (e.getActionCommand() == "incr")
      {cpt.incr();}
    if (e.getActionCommand() == "decr")
      {cpt.decr();}
    l.setText(String.valueOf(cpt.getValue()));
  }
```

4 Le modèle de communication par évènements

Présentation générala. Voir par ailleurs.

5 Second schéma d'interaction : l'interface écoute le modèle

De la même façon que l'interface écoute l'utilisateur via les évènements souris et claviers, l'interface écoute aussi le modèle via des évènements que celui-ci signale.

Cette écoute peut être modélisée par une communication d'évènements entre le modèle et la vue.

5.1 Exemple avec le compteur - utilisation de SWING

You can identify Swing components because their names start with J.

The AWT components are in the java.awt package, whereas the Swing components are in the javax.swing package.

5.2 Compteur MVC et SWING

Le compteur est modifié par un programme client et par deux interfaces dans une application globale.

```
package gui;
import standard.Compteur;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class CompteurSWING2
  extends JFrame
{

  Compteur cpt;
  JButton b1, b2, b3;
  JLabel l;

  public CompteurSWING2(){
    this(new Compteur());
  }

  public CompteurSWING2(Compteur c){
    super("Un Compteur");
cpt = c;
    //ici particularité SWING
    Container co = this.getContentPane();
    co.setLayout(new BorderLayout());
    b1 = new JButton("raz");
    co.add(b1, BorderLayout.SOUTH);
    b2 = new JButton("incr");
    co.add(b2, BorderLayout.EAST);
    b3 = new JButton("decr");
    co.add(b3, BorderLayout.WEST);
    l = new JLabel(String.valueOf(c.getValue()), JLabel.CENTER);
    co.add(l,BorderLayout.CENTER);

    ButtonListener bl = new ButtonListener();

    b1.addActionListener(bl);
    b2.addActionListener(bl);
```

```
    b3.addActionListener(bl);
  }

  // L'interface réagit à l'occurence d'un clic sur les boutons
  // Classe imbriquée - voir inner classes

  class ButtonListener implements ActionLister{

    public void actionPerformed(ActionEvent e) {
      // alternative : (Button)(e.getSource()) == b1
    if (e.getActionCommand() == "raz")
      {cpt.raz();}
    if (e.getActionCommand() == "incr")
      {cpt.incr();}
    if (e.getActionCommand() == "decr")
      {cpt.decr();}
  }
  }

  // L'interface réagit à une modification du compteur

  public void modelChanged(ModelChangedEvent e) {
    l.setText(String.valueOf(cpt.getValue()));

  public static void main(String[] args){
    JFrame frame = new CompteurSWING2();

    // adaptation au gestionnaire de fenêtre courant
    try {
      UIManager.setLookAndFeel
(UIManager.getCrossPlatformLookAndFeelClassName());
    } catch(Exception e) {}

//
    frame.addWindowListener(new WindowAdapter() {
public void windowClosing(WindowEvent e) {System.exit(0);}
      });



    frame.pack();
    frame.setVisible(true);
  }
}
```