Part VI

L'interface utilisateur

Interface swing

Definition (L'interface utilisateur (UI))

- Elle recouvre toutes les communications possibles entre un programme et son utilisateur, incluant l'interface graphique (GUI) :
  - ex : création de fenêtres, gestion d'évènements (clic souris, pression d'une touche,...).
- En Java, l'interface est gérée par :
  - le package `javax.swing`
  - le package `java.awt` (Abstract Windows Toolkit) :
    - désuet depuis Java 1.2.
  - le package `java.awt.event` (gestion des évènements)
- Tout programme implémentant une GUI `swing` doit posséder un conteneur principal :
  - une fenêtre (`JFrame`),
  - une fenêtre secondaire dépendant d'une autre (`JDialog`)
  - ou un applet, une région dans un navigateur (`JApplet`)

JFrame

Definition

`JFrame` est une classe du package `javax.swing` dont les instances sont des fenêtres possédant des propriétés telles que :

- des décorations
- une bordure
- un titre
- des boutons pour réduire, maximiser et fermer la fenêtre

et des méthodes pour :

- rendre la fenêtre visible (`setVisible`)
- obtenir ses composants (`getContentPane`)
- la redimensionner (`pack`, héritée de `java.awt`)

Exemple d'interface swing

Example (Programme affichant la fenêtre ci-dessus)

```
import javax.swing.*;
public class BonjourPresident{
    public static void main(String[] args){
        JFrame frame = new JFrame(“BonjourPresidentSwing”);
        JLabel label=new JLabel(“Bonjour President”);
        frame.getContentPane().add(label);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.pack();
        frame.setVisible(true);
    }
}
```

JPanel

Definition

- Le `JFrame`, un conteneur de haut niveau (top-level container) qui contient systématiquement, un conteneur intermédiaire :
  - le `JPanel`.
- `JPanel` est un conteneur intermédiaire qui permet de simplifier le positionnement des composants :
  - Par exemple, Bouton, `JButton` et label, `JLabel` qui sont des composants atomiques.
  - Autres exemples : `JComboBox`, `JTextField`, `JTable`,...
- Les composants sont organisés hiérarchiquement :
  - Un `JFrame` contient un `JPanel`.
  - Un `JPanel` contient un `JButton` et un `JLabel`.
- On peut ajouter des composants au `JFrame`, en les ajoutant au `JPanel`.

Autres composants

Composant	Nom
Boite de dialogue	`JDialog`
Panneau avec barre de défilement	`JScrollPane`
Panneau avec tabulation	`JTabbedPane`
Barre d'outils	`JToolBar`
Fenêtre imbriquées	`JInternalFrame`
Boites combinées	`JComboBox`
Liste	`JList`
Menu déroulant	`JMenu`
Barre de progression	`JProgressBar`
Sélecteur de fichier	`JFileChooser`
Table	`JTable`

Mise en page (Layout management)

- Possibilité d'ajouter du code HTML dans un label :
  `nomLabel.setText(“<html> code </html>”);`
- Possibilité d'ajouter du code sur certains composants :
  `Image icon = new ImageIcon(“path”,”description”);`
  `JButton j = new JButton(icon);`

Definition (Layout)

- Chaque conteneur possède un layout manager qui détermine la position de ses composants.
- On met en page une interface en paramètrant le layout du `JPanel` qui le compose.
- Cette mise en page peut être modifiée via la méthode `setLayout` :
  - Exemple : `pane.setLayout(new BorderLayout());`

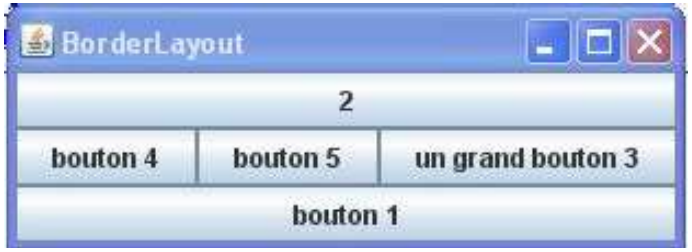
Mise en page (Layout management)

Definition

- Il existe 5 types de layout :
  1. `BorderLayout`
     - Les composants sont positionnés en fonction des points cardinaux
  2. `BoxLayout`
     - Les composants sont positionnés sur une seule ligne (ou colonne)
  3. `FlowLayout` (par défaut)
     - Les composants sont positionnés de gauche à droite, en commençant une nouvelle ligne si nécessaire
  4. `GridLayout`
     - Les composants sont positionnés dans les cellules (égales) d'un tableau
  5. `GridBagLayout`
     - Les composants sont positionnés dans les cellules d'un tableau où un composant peut occuper plus d'une cellule.

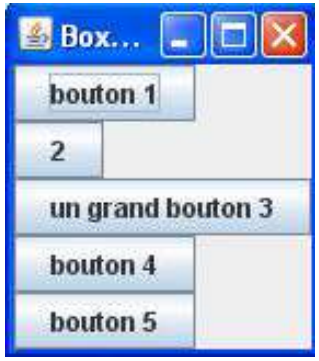
BorderLayout

```
JButton b1 = new JButton("bouton 1");
pane.add(b1,BorderLayout.SOUTH);
```



BoxLayout

```
pane.setLayout(new BoxLayout(pane,javax.swing.BoxLayout.PAGE_AXIS));
```

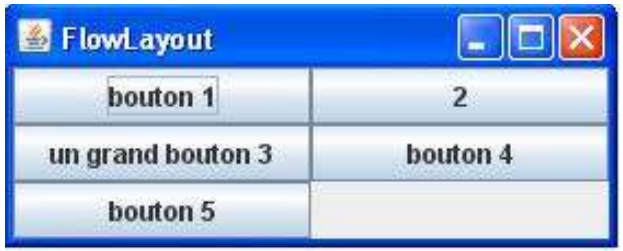


FlowLayout

```
pane.setLayout(new FlowLayout(FlowLayout.CENTER));
```

GridLayout

```
pane.setLayout(new GridLayout(3,2));
```



Les évènements

Definition

- Il existe différents types d'évènements :
  - clic de souris, pression sur une touche,...
- Un évènement est représenté par un objet qui contient des informations sur sa nature et sur sa source :
  - `ActionEvent`, `WindowEvent`, `MouseEvent`, `MouseMotionEvent`, ...
- Les objets réagissant à des évènements sont appelés des "event listeners".
- Un objet peut être notifié d'un évènement : il faut implanter une interface listener pour les évènements qui le concernent :
  - `ActionListener`, `WindowListener`, `MouseListener`, `MouseMotionListener`, ...
- Plusieurs listeners peuvent être associés à la même source et un listener peut être associé à plusieurs sources.

Gestionnaire d'évènements (event handler)

Definition

- Un gestionnaire d'évènements se compose de trois parties de code :
  - Une déclaration spécifiant :
    - qu'une classe d'event listener implémente une interface "listener"
    - ou qu'elle hérite d'une classe implémentant une interface "listener".
    - `public class MaClasse implements ActionListener{...}`
  - Des lignes de codes ajoutant des instances de la classe comme "listeners" sur des composants :
    - `unComposant.addActionListener(instanceDeMaClasse);`
  - Des lignes de codes implémentant les méthodes de l'interface "listener" dans la classe gestionnaire.
    - `public void actionPerformed(ActionEvent e){...}`

Gestionnaire d'évènements (event handler)

Remarques

- La méthode `actionPerformed` est la seule méthode de l'interface `ActionListener`.
- Les évènements sont traités dans l'ordre de leur apparition.
- Le code qui traite des évènements est exécuté dans le même thread.
  - Les évènements sont donc traités de manière séquentielle.
  - Il est donc nécessaire que le code soit rapide pour que l'interface graphique reste réactive.

Liste (non-exhaustive) d'évènements et de listeners

Evènement	Listener type
Pression sur un bouton	`ActionListener`
L'utilisateur ferme une fenêtre	`WindowListener`
L'utilisateur clique sur un bouton lorsque la souris est au dessus d'un composant	`MouseListener`
L'utilisateur bouge la souris au dessus d'un composant	`MouseMotionListener`
Un composant devient visible	`ComponentListener`
Un composant devient actif	`FocusListener`
Un changement de sélection dans une liste	`ListSelectionListener`

Evènements par composant

Les composants sont restreints à des évènements spéficiques :

- `ActionEvent : JButton, JComboBox`
- `WindowEvent : JFrame`
- `MouseEvent : JPanel, JButton, JTextField`

Un listener doit implémenter les méthodes de son interface :

- `ActionListener :`
  - `actionPerformed(ActionEvent e)`
- `WindowListener :`
  - `windowActivated(WindowEvent e)`
  - `windowClosed(WindowEvent e)`
  - `windowClosing(WindowEvent e)`
  - `windowDeactivated(WindowEvent e)`
  - `windowDeiconified(WindowEvent e)`
  - `windowIconified(WindowEvent e)`
  - `windowOpened(WindowEvent e)`

Evènements par composant

Un listener doit implémenter les méthodes de son interface (suite) :

- `MouseListener` :
  - `mouseClicked(MouseEvent e)`
  - `mouseEntered(MouseEvent e)`
  - `mouseExited(MouseEvent e)`
  - `mousePressed(MouseEvent e)`
  - `mouseReleased(MouseEvent e)`
- `MouseMotionListener` :
  - `mouseDragged(MouseEvent e)`
  - `mouseMoved(MouseEvent e)`

Remarque

Les classes dites “adapters" fournissent des implémantations par défaut :

- `WindowAdapter`, `MouseAdapter`, `MouseMotionAdapter`, `MouseInputAdapter`
- Le programmeur fournit seulement le code des méthodes que son programme va utiliser.

Exemple : les évènements souris

Un évènement souris a lieu :

- quand le curseur entre dans la zone d'un composant
- ou quand l'utilisateur clique sur un bouton

Les méthodes de la classe `MouseEvent` sont :

- `int getX()`
- `int getY()`
- `Component getComponent()` (héritée de `ComponentEvent`)
- `ComponentEvent boolean isShiftDown()` (héritée de `InputEvent`)
- `boolean isLeftMouseButton()` (méthode de la classe `javax.swing.SwingUtilities`)

Il y a deux listeners pour la souris :

- `MouseMotionListener` (les mouvements)
- `MouseListener` (tout le reste)

Exemple le bouton qui fait “clic”

Exemple le bouton qui fait “clic”

Example

```
import javax.swing.*; import java.awt.*; import java.awt.event.*;
public class SwingApplication {
    static private int numClicks = 0;
    static JLabel label = new JLabel("Nombre de clics :  " + "0 ");
    public static void main(String[] args) {
        JButton button = new JButton("Je suis un bouton");
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                numClicks++;
                label.setText("Nombre de clics :  " + numClicks);
            }
        });
        JFrame frame = new JFrame("Clic clic");
        JPanel pane=new JPanel();
        pane.add(button);
        pane.add(label);
        frame.add(pane);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.pack();
        frame.setVisible(true);
    }
}
```

Autre exemple : le convertisseur euro-dollar

Autre exemple : le convertisseur euro-dollar

Example

```
import javax.swing.*;import java.awt.event.*;
public class Convertisseur{
    static private double degre = 0;
    public static void main(String[] args) {
        final JTextField temp = new JTextField(5);
        final JLabel label = new JLabel("dollar");
        final JButton button = new JButton("Convertir");
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                try{
                    degre=0.66*Double.parseDouble(temp.getText());
                    label.setText(degre+"dollar");
                }catch(NumberFormatException d){ label.setText(degre+"dollar");}
            }
        });
        JFrame frame = new JFrame("convert"); JPanel pane=new JPanel();
        pane.add(temp); pane.add(button); pane.add(label);frame.add(pane);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.pack();
        frame.setVisible(true);
    }
}
```

L'interface KeyListener

Definition

L'interface `KeyListener` permet à une classe de répondre aux évènements liés au clavier :

- `public void keyTyped(KeyEvent e)`
- `public void keyPressed(KeyEvent e)`
- `public void keyReleased(KeyEvent e)`

Les méthodes de `KeyEvent` :

- `getKeyChar()` :
  - renvoie le caractere associé a la touche.
- `getKeyCode()` :
  - renvoie un code sous forme d'entier de type int pour les touches qui ne sont pas associées a un caractère (flèches, F1, F2, ..., Suppr, ...)

L'interface KeyListener

Example

```
import javax.swing.*;import java.awt.*; import java.awt.event.*;
public class APanel {
    JPanel pane = new JPanel();
    public Component createComponents(){
        pane.setBackground(Color.BLUE);
        pane.setPreferredSize(new Dimension(200,200));
        pane.addKeyListener(new KeyAdapter() {
            public void keyPressed(KeyEvent e) {
                JLabel j = new JLabel(); pane.add(j);
            j.setText("key pressed " + e.getKeyChar());}});
        pane.setFocusable(true);return pane;
    }
    public static void main(String[] args) {
        JFrame frame = new JFrame("Keyboard");
        APanel app = new APanel();
        Component contents = app.createComponents();
        frame.getContentPane().add(contents, BorderLayout.CENTER);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.pack();contents.requestFocus();
        frame.setVisible(true);
    }
}
```

Rappel : la classe `javax.swing.Timer`

- La classe `javax.swing.Timer` permet de répéter l'exécution d'une tâche à intervalle fixé ou d'exécuter une tâche après un délai.
- Le `Timer` genère un évènement de type action chaque fois que l'intervalle indiqué est depassé et jusqu'à ce que la méthode `stop()` soit appelée.
- Méthodologie :
  1. créer un thread avec `new Timer`
  2. associer un `ActionListener`
  3. lancer l'exécution du timer (`start`)
  4. programmer l'arrêt du timer (`stop`)

Remarque

Pour exécuter l'action une seule fois après un delai : `setRepeats(false)`

La classe javax.swing.Timer

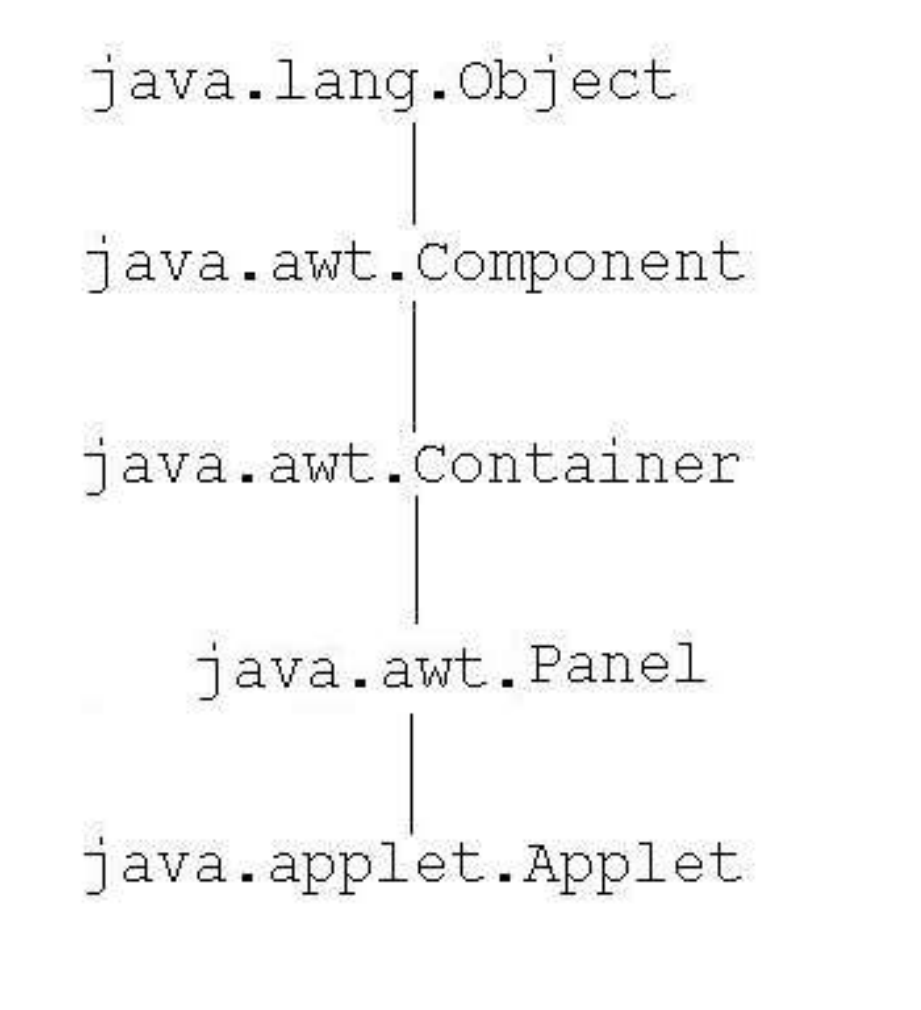
Example

```
import javax.swing.*; import java.awt.event.*; import java.awt.*;
public class SwingTimer {
    Timer minuteur; int i = 10; int delai = 1; JLabel label; JPanel pane;
    public Component createComponents(){
        pane = new JPanel(); label = new JLabel("compte = 10");
        pane.add(label); compter(); return pane;
    }
    public void compter() {
        ActionListener a = new ActionListener() {
            public void actionPerformed(ActionEvent e){
                i--; label.setText("compte = " + i);
                if (i == 0) { minuteur.stop(); }
            }};
        minuteur = new Timer(delai*1000, a); minuteur.start();
    }
    public static void main(String[] args) {
        JFrame frame = new JFrame("Swing Timer");
        SwingTimer st = new SwingTimer();
        Component contents = st.createComponents();
        frame.getContentPane().add(contents);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); frame.pack();
        frame.setVisible(true); }}
```

Les Applets

Definition

Une Applet est un petit programme web exécutable par un navigateur.
La classe `Applet` hérite des fonctionnalités de ses super-classes :



Ce qu'une applet ne peut pas faire

- L'execution d'une applet est une opération "sensible" puisqu'une machine cliente exécute un code provenant d'une autre machine.
- Pour des raisons de sécurité, les applets ne peuvent accéder a toutes les ressources de la machine cliente.
- Chaque browser adopte sa propre politique de sécurité.
- Dans tous les cas, une applet :
  - ne peut charger de bibliothèques ou définir des méthodes natives
  - ne peut avoir acces aux fichiers (en lecture ou écriture) de la machine cliente
  - ne peut établir de connexions avec une machine différente de la machine hébergeant l'applet
  - ne peut demarrer aucun programme sur la machine cliente
  - ne peut lire toutes les propriétés du système client
  - possède un "look and feel" différent de celui d'une application

Ce qu'une applet peut faire

Une applet :

- peut établir un connexion avec la machine hébergeant l'applet.
- peut afficher des documents HTML
- peut invoquer les méthodes publiques d'autres applets de la même page
- n'a aucune restriction sur les fichiers du systeme dont il a la charge
- peut continuer à s'exécuter même lorsque l'utilisateur quitte sa page

Les méthodes des applets

- `public void init():`
  - correspond au code d'initialisation d'un constructeur car les applets n'ont généralement pas de constructeur :
    - ★ car ils ne sont pas conçus pour être exécutés dans un environnement
- `public void start() :`
  - code d'exécution
- `public void stop() :`
  - code d'arrêt
- `public void destroy() :`
  - code de suppression
- `public void paint(Graphics g) :`
  - dessine la représentation de l'applet dans une fenêtre.
- `public void update () :`
  - augmente les performances de la méthode `paint`

Un exemple d'applet

Example

```
import java.applet.Applet; import java.awt.Graphics;
public class Simple extends Applet{
    StringBuffer buffer;
    public void init(){
        buffer =new StringBuffer();
        addItem(“Initialisation...”);
    }
    public void start(){addItem(“Execution...”);}
    public void stop(){addItem(“Arret...”);}
    public void destroy(){addItem(“Suppression...”);}
    void addItem(String mot){
        System.out.println(mot);
        buffer.append(mot);
        repaint();
    }
    public void paint(Graphics g) {
        g.drawRect(0,0,getSize().width-1,getSize().height-1); //rectangle autour
de l'Applet
        g.drawString(buffer.toString(),5,15);//dessine le buffer dans le
rectangle
    }
}
```

Applets et évènements

- Les applets héritant de la classe AWT Component, elles peuvent aussi gérer les évènements.
- Par exemple, si l'on ajoute le code suivant à l'exemple précédent :

```
import java.awt.event.Mouselistener;
import java.awt.event.MouseEvent;
public class Simple extends Applet implements MouseListener{
    ...
    public void init(){
       addMouseListener(this);
       ...
    }
    ...
    public void mouseClicked(MouseEvent event){
       addItem(“clic”);
    }
    ...
}
```

Applets et interface utilisateur

- Les applets héritant de la classe AWT Component, elles peuvent aussi gérer les composants permettant de construire une interface utilisateur (UI) en utilisant les méthodes :
  - `add` :
    - pour ajouter un composant
  - `remove` :
    - pour supprimer un composant
  - `setLayout` :
    - pour contrôler la disposition (taille, position) des composants

Applets et interface utilisateur

Example (Ajout d'un texte non-éditable au programme précédent)

```
import java.awt.TextField;
public class Simple extends Applet{
    TextField field; //utilisé à la place du StringBuffer
    ...
    public void init(){
        field = new TextField(); //création de l'attribut
        field.setEditable(false); //rendre le texte non-éditable
        //définir une nouvelle mise en page
        setLayout(new java.awt.GridLayout(1,0));
        add(field);
        addItem(``initialisation'');
    }
    ...
    void addItem(String mot){
        String t =field.getText();
        System.out.println(mot);
        field.setText(t+mot);//concaténation des deux mots
    }
    //La méthode paint n'est plus nécessaire car le composant TextField se paint
lui-même de manière automatique
    ...
}
```

Applet et HTML

- Ajouter une applet sur une page HTML :
  - `<APPLET CODE=''SousClasseApplet.class'' WIDTH=unEntier HEIGHT=unEntier> </APPLET>`
- Pour exécuter le code, utiliser :
  - un navigateur supportant java
  - ou la commande `appletviewer NomFichier.html`
- Pour utiliser plusieurs fichiers :
  - créer des archives jar (Java ARchive) :
    - `jar cvf file.zip *.class *.gif`
  - et les inclure dans le fichier HTML :
    - `<APPLET CODE=''SousClasseApplet.class'' ARCHIVE=''file, file1, file2'' WIDTH=unEntier HEIGHT=unEntier> </APPLET>`

Jouer de la musique avec son Applet

- Différentes méthodes permettent de traiter le son :
  - `AudioClip getAudioClip(URL)`
  - `AudioClip getAudioClip(URL, String)`
  - `void play(URL)`
  - `void play(URL, String)`
- L'interface `AudioClip` dispose des méthodes suivantes :
- `void loop()` :
  - joue le clip en boucle
- `void play()` :
  - joue le clip une seule fois
- `void stop()` :
  - arrête de jouer un clip

Jouer de la musique avec son Applet

Example

```
import java.applet.*; import java.awt.event.*; import java.net.*;
public class Son extends Applet {
    public void init(){
        URL u=null;
        try{
            u= new URL("file:///C:/Documents and Settings/pechoux/Mes
documents/travail/slidesJavaL3MIAGE/java/bark.au");
        }catch(MalformedURLException e){}
        final AudioClip clip1= getAudioClip(u);
        Button play1;
        play1 = new Button("Ouaf!");
        play1.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                clip1.play();
            }
        });
        add(play1);
    }
}
```