Le langage Java

Apprentissage en lien avec le langage UML

Emmanuel Bruno (bruno@univ-tln.fr)

Département d'Informatique – UFR de Sciences et Techniques – Université du Sud Toulon-Var

Octobre 2008

Le langage Java

Apprentissage en lien avec le langage UML

Emmanuel Bruno (bruno@univ-tln.fr)

Département d'Informatique – UFR de Sciences et Techniques – Université du Sud Toulon-Var

Octobre 2008

Interface avec l'utilisateur

- Les programmes peuvent interagir avec un utilisateur
  - Demande d'information
  - Saisie de données
  - Affichage des résultats
- Cet échange se fait via un interface utilisateur (UI en anglais)
  - Elle peut en mode texte ou graphique

Interface graphique

- Une interface graphique (GUI) est composée d'une ou plusieurs fenêtres
- Une fenetre contient un ensemble de composants graphiques (*widgets* en anglais) :
  - boutons
  - listes déroulantes
  - menus
  - champ texte
  - ...
- Difficulté majeure :
  - Le programme est guidé par les actions de l'utilisateur
  - Celui peut utiliser plusieurs objets graphiques à un même moment
    - saisie d'un texte, clic sur un bouton, ...
  - L'ordre de ces actions ne peut pas être complètement prévu

Programmation événementielle

- Pour simplifier la programme dans ce type de cas on utilise un nouveau mode de programmation
  - La programmation événementielle
- La structure d'un programme de ce type est :
  - les actions de l'utilisateur produisent des événements qui sont empilés
  - le programme traite les évènements avec des actions préprogrammées

Les écouteurs

- Dans les IHM Java, on distinguera deux types d'objets es *observateurs* et les *observés*
- les widgets (boutons, ...) sont les observés
- on associe aux composants graphiques des observateurs (*listeners* en Anglais, ou aussi écouteur)
  - un écouteur traite des classes d'évènements particuliers (clic souris, frappe au clavier, ...).
- Le fonctionnement est le suivant :
  - L'écouteur est prévenu par le composant graphique qu'un évènement les concerne
  - L'écouteur va exécuter le code prévu pour réagir à cet évènement

Les API en Java

- On distingue deux bibliothèques :
  - AWT (Abstract Window Toolkit), à partir du JDK 1.1
  - Swing, à partir de Java 2
- Elles composent avec Java2D l'ensemble de bibliothèques (*framework*) graphiques proposé par Java appelé
  - Java Foundation Classes : JFC
  - Swing s'appuie sur AWT
    - Le fonctionnement est le même
    - les classes de Swing héritent des classes de AWT
- Comment choisir en AWT et SWING
  - Swing permet tout ce qu'AWT permet et même plus
  - Les composants swing sont plus jolis mais plus lourd (et donc plus lent)

Les paquetages de base

- AWT : `java.awt` et `java.awt.event`
- Swing : `javax.swing`, `javax.swing.event`
  - plus ceux pour des composants de haut niveau dans `javax.swing` :
    - `table`, `tree`, `text`, `filechooser`, `colorchooser`

  et ceux liés à l'apparence aussi dans `javax.swing` :
  - pluggable look and feel (plaf)
  - `plaf`, `plaf.basic`, `plaf.metal`, `plaf.windows`, `plaf.motif`

Un premier exemple : ouvrir un fenêtre

```
package coursSSI3.exemples.ihm;

import java.awt.Frame;

import javax.swing.JFrame;

public class Fenetre extends JFrame {
        public Fenetre() {
                super("Une fenetre");
                setSize(300, 200);
                pack(); //Ne pas mettre si la fenetre est vide
                setVisible(true);
        }

        public static void main(String Args[]) {
                Fenetre f = new Fenetre();
        }

}
```

Listing 1 – coursSSI3/exemples/ihm/Fenetre.java

Taille d'une fenêtre

- Il est possible de spécifier préciser les caractéristiques de la fenêtre
  - Taille : `setSize(int l, int h)` (ou instance de `Dimension`)
  - Position : `setLocation(int x, int y)` (ou instance de `Point`), fixe le sommet haut/gauche
  - Les deux : `setBounds(int x, int y, int l, int h)` (ou instance `Rectangle`)
- Il est aussi possible d'adapter la fenêtre à son contenu :
  - La méthode `pack()` fixe la taille nécessaire à la fenêtre la taille en fonction des tailles préférées de chacun de ses composants

Classe `java.awt.Toolkit`

- Les sous-classes de la classe abstraite `Toolkit` permettent de communiquer avec le système d'exploitation de la machine réelle.
- La méthode `getDefaultToolkit()` retourne un instance d'une classe qui implante `Toolkit` (cf. propriété `awt.toolkit`)
- Les méthodes classiques sont : `getScreenSize()`, `getScreenResolution()`, `beep()`, `getImage()`, `createImage()`, `getSystemEventQueue()`

Positionnement d'une fenêtre et icône

```
package coursSSI3.exemples.ihm;
import java.awt.*;
import javax.swing.JFrame;

public class FenetrePlacee extends JFrame {
        public FenetrePlacee() {
                Toolkit tk = Toolkit.getDefaultToolkit();
                Dimension d = tk.getScreenSize();
                int hauteurEcran = d.height;
                int largeurEcran = d.width;
                setSize(largeurEcran / 2, hauteurEcran / 2);
                setLocation(largeurEcran / 4, hauteurEcran / 4);
                Image img = tk.getImage("icone.gif");
                setIconImage(img);
                //pack();
                setVisible(true);
        }

        public static void main(String[] args) {
                FenetrePlacee f = new FenetrePlacee();
        }
}
```

Listing 2 – coursSSI3/exemples/ihm/FenetrePlacee.java

Composants lourds

- Pour afficher des fenêtres (instances de JFrame), Java s'appuie sur les fenêtres fournies par le système d'exploitation hôte dans lequel tourne la JVM
- Les composants Java qui, comme les JFrame, s'appuient sur des composants du système hôte sont dit « lourds »
- L'utilisation de composants lourds améliore la rapidité d'exécution mais nuit à la portabilité et impose les fonctionnalités des composants

Composants légers

- Au contraire de AWT qui utilise les widgets du système d'exploitation pour tous ses composants graphiques (fenêtres, boutons, listes, menus, etc.), Swing ne les utilise que pour les fenêtres de base « top-level »
- Les autres composants, dits légers, sont dessinés dans ces containers lourds, par du code « pur Java »
- Attention, les composants lourds s'affichent toujours au-dessus des composants légers

Containers lourds

- Il y a trois sortes de containers lourds (un autre, `JWindow`, est plus rarement utilisé) :
  - `JFrame` fenêtre pour les applications
  - `JApplet` pour les applets
  - `JDialog` pour les fenêtres de dialogue
- Pour construire une interface graphique avec Swing, il faut créer un (ou plusieurs) container lourd et placer à l'intérieur les composants légers qui forment l'interface graphique

Libérer les ressources associées à une `JFrame`

- En tant que composant lourd, une `JFrame` utilise des ressources du système sous-jacent
- Si on ne veut plus utiliser une `JFrame` (ou JDialog ou JWindow), mais continuer l'application, il faut lancer la méthode `dispose()` de la fenêtre ; les ressources seront rendues au système
- Voir aussi la constante `DISPOSE_ON_CLOSE` de l'interface `javax.swing.WindowConstants`

Classe `JComponent`

- La plupart des widgets de Swing sont des instances de sous-classes de la classe JComponent
- Les instances des sous-classes de `JComponent` sont de composants « légers »
- `JComponent` hérite de la classe `Container`
- Tous les composants légers des sous-classes de JComponent peuvent donc contenir d'autres composants

Les Containers

- Des composants sont destinés spécifiquement à recevoir d'autres éléments graphiques :
  - les containers « top-level » lourds JFrame, JApplet, JDialog, JWindow
  - les containers « intermédiaires » légers JPanel, JScrollPane, JSplitPane, JTabbedPane, Box (ce dernier est léger mais n'hérite pas de JComponent)

JPanel

- JPanel est la classe mère des containers intermédiaires les plus simples
- Un JPanel sert à regrouper des composants dans une zone d'écran
- Il n'a pas d'aspect visuel déterminé ; son aspect visuel est donné par les composants qu'il contient
- Il peut aussi servir de composant dans lequel on peut dessiner ce que l'on veut, ou faire afficher une image (par la méthode paintComponent)

Ajouter des composants

- Les containers « top-level » ne peuvent contenir directement d'autres composants
- Ils sont associés à un autre container, le « content pane » dans lequel on peut ajouter les composants
- On obtient ce content pane par (topLevel est un container lourd quelconque ; JFrame par exemple)
  - `Container contentPane = topLevel.getContentPane() ;`

Placer les composants

```
package coursSSI3.exemples.ihm;

import java.awt.*;
import javax.swing.*;

public class Composants extends Fenetre {

        public Composants() {
                super();
                Container contentPane = getContentPane();
                JLabel label = new JLabel("Bonjour");
                JButton b1 = new JButton("Cliquez moi !");
                contentPane.add(label, BorderLayout.NORTH);
                contentPane.add(b1, BorderLayout.SOUTH);
                pack();
        }

        public static void main(String[] args) {
                Composants c = new Composants();
        }

}
```

Listing 3 – coursSSI3/exemples/ihm/Composants.java

Les layout managers

- l'utilisateur peut changer la taille d'une fenêtre ; les composants de la fenêtre doivent alors être repositionnés
- Les fenêtres (plus généralement les containers) utilisent des gestionnaires de mise en place (layout manager) pour repositionner leurs composants
- Il existe plusieurs types de layout managers avec des algorithmes de placement différents
- Quand on ajoute un composant dans un container on ne donne pas la position exacte du composant
- On donne plutôt des indications de positionnement au gestionnaire de mise en place
  - explicites (`BorderLayout.NORTH`)
  - ou implicites (ordre d'ajout dans le container)

Algorithme de placement

- Un layout manager place les composants « au mieux » suivant
  - l'algorithme de placement qui lui est propre
  - les indications de positionnement des composants
  - la taille du container
  - les tailles préférées des composants

`Dimension` et Taille des composants

- La classe `java.awt.Dimension` est utilisée pour donner des dimensions de composants en pixels
  - Elle possède deux variables d'instance publiques de type `int`
    - `height`, `width`
  - Constructeur : `Dimension(int, int)`
- Tous les composants graphiques (`Component`) peuvent indiquer leurs tailles pour l'affichage
  - taille maximum, taille préférée, taille minimum
- Les méthodes : `{get|set}{Maximum|Preferred|Minimum}Size()`

Taille préférée

- La taille préférée est la plus utilisée par les layout managers ; un composant peut l'indiquer en redéfinissant la méthode `Dimension getPreferredSize()`
- On peut aussi l'imposer « de l'extérieur » avec la méthode `void setPreferredSize(Dimension)`
- Mais le plus souvent, les gestionnaires de mise en place ne tiendront pas compte des tailles imposées de l'extérieur

Layout manager

- Fixer le layout manager
  - Par défaut, les fenêtres `JFrame` ont un gestionnaire de mise en place qui est une instance de la classe `BorderLayout`
  - On peut changer le gestionnaire de mise en place d'un Container par la méthode `setLayout(LayoutManager)` de la classe `Container`
- Les types les plus courants de gestionnaire de mise en place :
  - `BorderLayout` : placer aux quatre points cardinaux
  - `FlowLayout` : placer à la suite
  - `GridLayout` : placer dans une grille
  - `BoxLayout` : placer verticalement ou horizontalement
  - `GridBagLayout` : placements complexes

BorderLayout

- Affiche au maximum 5 composants (aux 4 points cardinaux et au centre)
- Essaie de respecter la hauteur préférée du nord et du sud et la largeur préférée de l'est et de l'ouest ; le centre occupe toute la place restante
- layout manager par défaut de JFrame et JDialog
- Les composants sont centrés dans leur zone
- On peut spécifier des espacement horizontaux et verticaux minimaux entre les composants
- Si on oublie de spécifier le placement lors de l'ajout d'un composant, celui-ci est placé au centre (source de bug !)
- Règle pratique : l'est et l'ouest peuvent être étirés en hauteur mais pas en largeur ; le contraire pour le nord et le sud ; le centre peut être étiré en hauteur et en largeur

Placement dans une fenêtre complexe

- Pour disposer les composants d'une fenêtre de structure graphique complexe on peut :
  - utiliser des containers intermédiaires, ayant leur propre type de gestionnaire de placement, et pouvant éventuellement contenir d'autres containers
  - utiliser un gestionnaire de placement de type GridBagLayout (plus souple mais parfois plus lourd à mettre en oeuvre)
  - mixer ces 2 possibilités

Utiliser un JPanel

```
package coursSSI3.exemples.ihm;
import java.awt.*;
import javax.swing.*;

public class JPanelExemple extends Fenetre {
        public JPanelExemple() {
                Container contentPane = getContentPane();
                JPanel panelBoutons = new JPanel();
                JButton b1 = new JButton("Cliquez moi !");
                JButton b2 = new JButton("Et moi aussi !");
                panelBoutons.add(b1);
                panelBoutons.add(b2);
                contentPane.add(panelBoutons, BorderLayout.NORTH);
                JTextArea textArea = new JTextArea(15, 5);
                contentPane.add(textArea, BorderLayout.CENTER);
                JButton quitter = new JButton("Quitter");
                contentPane.add(quitter, BorderLayout.SOUTH);
                pack();
        }
        public static void main(String Args[]) {
                JPanelExemple f = new JPanelExemple();
        }
}
```

Listing 4 – coursSSI3/exemples/ihm/JPanelExemple.java

FlowLayout

- Rangement de haut en bas et de gauche à droite
- Les composants sont affichés à leur taille préférée
- layout manager par défaut de JPanel et JApplet
- Attention, la taille préférée d'un container géré par un FlowLayout est calculée en considérant que tous les composants sont sur une seule ligne

Code avec FlowLayout

```
package coursSSI3.exemples.ihm;
import java.awt.*;
import javax.swing.*;

public class FlowLayoutExplicite extends JFrame {
        public FlowLayoutExplicite() {
                JPanel panel = new JPanel();
                JTextField zoneSaisie;
                JButton bouton;
                panel.setLayout(
                        new FlowLayout(FlowLayout.LEFT, 5, 4));
                panel.add(zoneSaisie = new JTextField(20));
                panel.add(bouton = new JButton("OK"));
                getContentPane().add(panel);
                pack();
                setVisible(true);
        }
        public static void main(String[] args) {
                FlowLayoutExplicite f =
                        new FlowLayoutExplicite();
        }

}
```

Listing 5 – coursSSI3/exemples/ihm/FlowLayoutExplicite.java

GridLayout

- Les composants sont disposés en lignes et en colonnes
- Les composants ont tous la même dimension
- Ils occupent toute la place qui leur est allouée
- On remplit la grille ligne par ligne

Code avec GridLayout

```
package coursSSI3.exemples.ihm;
import java.awt.*;
import javax.swing.*;

public class GridLayoutExemple extends JFrame {
        public GridLayoutExemple() {
                JTextField s1,s2,s3;
                JButton b1,b2,b3;
                JPanel panel = new JPanel();
                panel.setLayout(new GridLayout(3, 2));
                panel.add(s1 = new JTextField(20)); // (1,1)
                panel.add(b1 = new JButton("OK")); // (1,2)
                panel.add(s2 = new JTextField(20)); // (2,1)
                panel.add(b2 = new JButton("OK")); // (2,2)
                panel.add(s3 = new JTextField(20)); // (3,1)
                panel.add(b3 = new JButton("OK")); // (3,2)
                getContentPane().add(panel);
                pack();
                setVisible(true);
        }
        public static void main(String[] args) {
                GridLayoutExemple g = new GridLayoutExemple();
                }
}
```

Listing 6 – coursSSI3/exemples/ihm/GridLayoutExemple.java

- On doit indiquer le nombre de lignes ou le nombre de colonnes et mettre 0 pour l'autre nombre (si on donne les 2 nombres, le nombre de colonnes est

GridBagLayout

- Comme GridLayout, mais un composant peut occuper plusieurs « cases » du quadrillage ; la disposition de chaque composant est précisée par une instance de la classe GridBagConstraints
- C'est le layout manager le plus souple mais aussi le plus complexe

Code avec GridBagLayout

```
package coursSSI3.exemples.ihm;
import java.awt.*;import javax.swing.*;
public class GribBagExemple extends JFrame {
 public GribBagExemple() {
  JPanel panel = new JPanel();JButton b1,b2,b3;
  panel.setLayout(new GridBagLayout());
  GridBagConstraints c1 = new GridBagConstraints();
  c1.gridx = 0;c1.gridy = 0;
  c1.gridheight = 1;c1.gridwidth = 2;
  c1.weightx = 0.7;c1.weighty = 0.5;
  panel.add(b1 = new JButton("Bouton␣1"), c1);
  GridBagConstraints c2 = new GridBagConstraints();
  c2.gridx = 0;c2.gridy = 1;
  c2.gridheight = 1;c2.gridwidth = 1;
  panel.add(b2 = new JButton("Bouton␣2"), c2);
  GridBagConstraints c3 = new GridBagConstraints();
  c3.gridx = 1;c3.gridy = 1;
  c3.gridheight = 1;c3.gridwidth = 1;
  panel.add(b3 = new JButton("Bouton␣3"), c3);
  getContentPane().add(panel);pack();setVisible(true);
 }
 public static void main(String[] args) {
  GribBagExemple g = new GribBagExemple(); }
}
```

Listing 7 – coursSSI3/exemples/ihm/GribBagExemple.java

Autres contraintes

- fill détermine si un composant occupe toute la place dans son espace réservé (constantes de la classe GridBagConstraint : BOTH, NONE, HORIZONTAL, VERTICAL)
- anchor dit où placer le composant quand il est plus petit que son espace réservé (CENTER, SOUTH, ...)
- insets ajoute des espaces autour des composants : contraintes.insets = new Insets(5,0,0,0) (ou contraintes.insets.left = 5)
- ipadx, iapdy ajoutent des pixels à la taille minimum des composants

BoxLayout

- Aligne les composants sur une colonne ou une ligne (on choisit à la création)
- Respecte la largeur (resp. hauteur) préférée et maximum, et l'alignement horizontal (resp. vertical)
- Layout manager par défaut de Box et de JToolBar
- Pour un alignement vertical, les composants sont affichés centrés et si possible
  - à leur largeur préférée
  - respecte leur hauteur maximum et minimum (`get{Maxi|Mini}mumSize()`)
- Pour un alignement horizontal, idem en intervertissant largeur et hauteur
- Pour changer les alignements, on peut utiliser les méthodes de la classe `Component setAlignment{X|Y}`
- Alignement (Constantes de `Component`) :
  - `{LEFT|CENTER|RIGHT}_ ALIGNMENT`
  - `{TOP|BOTTOM}_ ALIGNMENT`

Problèmes d'alignement

- Si tous les composants géré par un BoxLayout n'ont pas le même alignement, on peut avoir des résultats imprévisibles
- Par exemple, le seul composant aligné à gauche peut être le seul qui n'est pas aligné à gauche !
- Il vaut donc mieux avoir le même alignement pour tous les composants

Classe Box

- Cette classe est un container qui utilise un BoxLayout pour ranger ses composants horizontalement ou verticalement
- Elle fournit des méthodes static pour obtenir des composants invisibles pour affiner la disposition de composants dans un container quelconque : glue, étais et zones rigides

Code avec Box

```
package coursSSI3.exemples.ihm;
import java.awt.*;
import javax.swing.*;
public class BoxSample extends Fenetre {
        public BoxSample() {
                // Penser a BoxLayout
                Box box = Box.createVerticalBox();
                box.add(new JButton("Action 1"));
                box.add(new JButton("Action 2"));
                box.add(Box.createVerticalStrut(5));
                box.add(new JButton("Action 3"));
                box.add(new JButton("Action 4"));
                box.add(Box.createRigidArea(
                        new Dimension(5, 35)));
                box.add(new JButton("OK!"));
                JPanel panel = new JPanel();
                panel.add(box);
                getContentPane().add(panel);
                pack();
        }
        public static void main(String[] args) {
                BoxSample b = new BoxSample();
        }
}
```

Listing 8 – coursSSI3/exemples/ihm/BoxSample.java

CardLayout et JTabbedPane

- CardLayout : Affichage des composants à tour de rôle
- JTabbedPane : Affichage d'onglets

```
package coursSSI3.exemples.ihm;
import javax.swing.*;
public class Onglet extends Fenetre{
        JPanel p1, p2, p3;
        public Onglet() {
                JTabbedPane onglets = new JTabbedPane();
                setSize( 300, 200 );
                JPanel panel = new JPanel();
                onglets.addTab("Option",p1=new JPanel());
                onglets.addTab("Resultats",p2=new JPanel());
                onglets.addTab("Aide",p3=new JPanel());
                panel.add(onglets);
                p1.add(new JButton("Options"));
                p2.add(new JButton("Resultats"));
                p3.add(new JButton("Aide"));
                getContentPane().add(panel);setVisible(true);
        }
        public static void main(String[] args) {
                Onglet o = new Onglet();
        }
}
```

Listing 9 – coursSSI3/exemples/ihm/Onglet.java

Traiter les événements

- L'utilisateur utilise le clavier et la souris pour intervenir
- Le système d'exploitation engendre des événements
- Le programme doit lier des traitements à ces événements
  - **bas niveau**, générés directement par des actions élémentaires de l'utilisateur
  - « **logiques** » de plus haut niveau, plusieurs actions élémentaires correspondants à une action complète

Exemples d'événements

- De bas niveau :
  - appui sur un bouton de souris ou une touche du clavier
  - relâchement du bouton de souris ou de la touche
  - déplacer le pointeur de souris
- Logiques :
  - frappe d'un A majuscule
  - clic de souris
  - lancer une action (clic sur un bouton par exemple)
  - choisir un élément dans une liste
  - modifier le texte d'une zone de saisie

Événements engendrés

- La frappe d'un A majuscule engendre 5 événements :
- 4 événements de bas niveau :
  - appui sur la touche Majuscule
  - appui sur la touche A
  - relâchement de la touche A
  - relâchement de la touche Majuscule
- 1 événement logique :
- frappe du caractère « A » majuscule

Classes d'événements

- Les événements sont représentés par des instances de sous-classes de java.util.EventObject
- Liés directement aux actions de l'utilisateur : KeyEvent, MouseEvent
- De haut niveau : FocusEvent, WindowEvent, ActionEvent, ItemEvent, ComponentEvent fenêtre ouverte, fermée, icônifiée ou désicônifiée, composant déplacé, retaillé, caché ou montré déclenchent une action choix dans une liste, dans une boîte à cocher

Écouteurs

- Chacun des composants graphiques a ses observateurs (ou écouteurs, listeners)
- Un écouteur doit s'enregistrer auprès des composants qu'il souhaite écouter, en lui indiquant le type d'événement qui l'intéresse (par exemple, clic de souris)
- Il peut ensuite se désenregistrer
- Relation écouteurs - écoutés
  - Un composant peut avoir plusieurs écouteurs (par exemple, 2 écouteurs pour les clics de souris et un autre pour les frappes de touches du clavier)
  - Un écouteur peut écouter plusieurs composants

ActionEvent

- Cette classe décrit des événements de haut niveau très utilisés qui correspondent à une action de l'utilisateur qui va le plus souvent déclencher un traitement (une action) :
  - clic sur un bouton
  - return dans une zone de saisie de texte
  - choix dans un menu
- Ces événements sont très fréquemment utilisés et ils sont très simples à traiter

Interface ActionListener

- Un objet ecouteur intéressé par les événements de type « action » (classe ActionEvent) doit appartenir à une classe qui implémente l'interface java.awt.event.ActionListener
- Définition de ActionListener :

```
public interface ActionListener
extends EventListener {
void actionPerformed(ActionEvent e) ; }
```

- Inscription comme ActionListener
  - On inscrit un tel écouteur auprès d'un composant nommé composant par la méthode composant.addActionListener(ecouteur) ;
  - On précise ainsi que ecouteur est intéressé par les événements ActionEvent engendrés par composant

Message déclenché par un événement

- Un événement unActionEvent engendré par une action de l'utilisateur sur bouton, provoquera l'envoi d'un message actionPerformed à tous les écouteurs : ecouteur.actionPerformed(unActionEvent) ;
- Ces messages sont envoyés automatiquement à tous les écouteurs qui se sont enregistrés auprès du bouton

Interface Action

- Cette interface hérite de ActionListener
- Elle permet de partager des informations et des comportements communs à plusieurs composants
- Par exemple, un bouton, un choix de menu et une icône d'une barre de menu peuvent faire quitter une application
- Elle est étudiée à la fin de cette partie du cours

Conventions de nommage

- Si un composant graphique peut engendrer des événements de type TrucEvent sa classe (ou une de ses classes ancêtres) déclare les méthodes {add—remove}TrucListener()
- L'interface écouteur s'appellera TrucListener

Écouteur MouseListener

- Des interfaces d'écouteurs peuvent avoir de nombreuses méthodes
- Par exemple, les méthodes déclarées par l'interface MouseListener sont :
  - `void mouseClicked(MouseEvent e)`
  - `void mouseEntered(MouseEvent e)`
  - `void mouseExited(MouseEvent e)`
  - `void mousePressed(MouseEvent e)`
  - `void mouseReleased(MouseEvent e)`

Adaptateurs

- Pour éviter au programmeur d'avoir à implanter toutes les méthodes d'une interface « écouteur », AWT fournit des classes (on les appelle des adaptateurs), qui implantent toutes ces méthodes
- Le code des méthodes ne fait rien
- Permet au programmeur de ne redéfinir dans une sous-classe que les méthodes qui l'intéressent
- Dans java.awt.event : KeyAdapter, MouseAdapter, MouseMotionAdapter, FocusAdapter, ComponentAdapter, WindowAdapter

Exemple de code d'un écouteur

- Un exemple d'écouteur sur un bouton

```
package coursSSI3.exemples.ihm;

import java.awt.event.*;
import javax.swing.JButton;

class EcouteurBouton extends MouseAdapter {
        public void mousePressed(MouseEvent e) {
                ((JButton) e.getSource()).setText("CLIC!");
        }

        public void mouseClicked(MouseEvent e) {
                ((JButton) e.getSource()).setText(
                  "x:"+e.getX()+" Y:"+e.getY());
                }
}
```

Listing 10 – coursSSI3/exemples/ihm/EcouteurBouton.java

Tester les touches modificatrices

- Pour tester si l'utilisateur a appuyé sur les touches Maj, Ctrl, Alt pendant qu'il cliquait sur la souris ou qu'il appuyait sur une autre touche, on utilise les constantes static de type int suivantes de la classe InputEvent : SHIFT_MASK , CTRL_MASK, ALT_MASK, META_MASK, ALT_GRAPH_MASK
- d'autres constantes pour tester le bouton de la souris : BUTTON1_MASK, BUTTON2_MASK, BUTTON3_MASK
- Exemple d'utilisation : `if ((e.getModifiers() & SHIFT_MASK) == 0)`
- Pour les boutons de la souris, on peut aussi utiliser des méthodes static de la classe SwingUtilities telles que isLeftMouseButton

Classe de l'écouteur

- Soit C la classe du container graphique qui contient le composant graphique
- Plusieurs solutions pour choisir la classe de l'écouteur de ce composant graphique :
  - classe C
  - autre classe spécifiquement créée pour écouter le composant :
  - classe externe à la classe C (rare)
  - classe interne de la classe C
  - classe interne anonyme de la classe C (fréquent)

Solution 1 : classe C

- Solution simple
- Mais peu extensible :
  - si les composants sont nombreux, la classe devient vite très encombrée
  - de plus, les méthodes « callback » comporteront alors de nombreux embranchements pour distinguer les cas des nombreux composants écoutés

Exemple solution 1

```
package coursSSI3.exemples.ihm;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class TestEcouteur1 extends JFrame
  implements ActionListener {
 private JButton b1 = new JButton("B1"), b2 = new JButton("B2");
 public void actionPerformed(ActionEvent e) {
  Object source = e.getSource();
  if (source == b1) System.out.println("Clic sur bouton 1");
  else if (source == b2) System.out.println("Clic sur bouton 2");
 }
 public TestEcouteur1() {
  JPanel panel;
  getContentPane().add(panel = new JPanel());
  panel.add(b1);panel.add(b2);
  b1.addActionListener(this);
  b2.addActionListener(this);
  pack();setVisible(true);
 }
 public static void main(String[] args) {
  TestEcouteur1 t = new TestEcouteur1();
 }
}
```

Listing 11 – coursSSI3/exemples/ihm/TestEcouteur1.java

Remarque sur les boutons

- On peut associer à chaque bouton (et choix de menus) une chaîne de caractères par `bouton.setActionCommand("chaine")` ;
- Cette chaîne
  - par défaut, est le texte affiché sur le bouton
  - permet d'identifier un bouton ou un choix de menu, indépendamment du texte affiché
  - peut être modifiée suivant l'état du bouton
  - peut être récupérée par la méthode `getActionCommand()` de la classe `ActionEvent`

Solution 2 : classe interne

- Solution plus extensible : chaque composant (ou chaque type ou groupe de composants) a sa propre classe écouteur
- Le plus souvent, la classe écouteur est une classe interne de la classe C

Exemple

```
package coursSSI3.exemples.ihm;
import java.awt.*;import java.awt.event.*;
import javax.swing.*;
public class TestEcouteur2 extends JFrame  {
 class EcouteurBouton implements ActionListener {
  public void actionPerformed(ActionEvent e) {
   String commande = e.getActionCommand();
   if (commande.equals("b1"))System.out.println("Clic␣b1!");
   else if (commande.equals("b2"))System.out.println("Clic␣b2!");
  }
 }
 private JButton b1 = new JButton("B1"), b2 = new JButton("B2");
 public TestEcouteur2() {
  JPanel panel;getContentPane().add(panel = new JPanel());
  panel.add(b1);panel.add(b2);
  ActionListener eb = new EcouteurBouton();
  b1.addActionListener(eb); b1.setActionCommand("b1");
  b2.addActionListener(eb); b2.setActionCommand("b2");
  pack();setVisible(true);
 }
 public static void main(String[] args) {
  TestEcouteur2 t = new TestEcouteur2();}
}
```

Listing 12 – coursSSI3/exemples/ihm/TestEcouteur2.java

Solution 3 : classe interne anonyme

- Si la classe écoute un seul composant et ne comporte pas de méthodes trop longues, la classe est le plus souvent une classe interne anonyme
- L'intérêt est que le code de l'écouteur est proche du code du composant
- Rappel : une classe interne locale peut utiliser les variables locales et les paramètres de la méthode, seulement s'ils sont final

Classe anonyme

```
package coursSSI3.exemples.ihm;

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

import coursSSI3.exemples.animaux.Chien;

public class ClasseAnonyme {

        public static void main(String[] args) {
                Chien c1 = new Chien();
                c1.aboyer();
                Chien c2 = new Chien() {
                        public void aboyer() {
                                System.out.println("Grr");
                        }
                };
                c2.aboyer();
        }
}
```

Listing 13 – coursSSI3/exemples/ihm/ClasseAnonyme.java

Exemple JButton

```
package coursSSI3.exemples.ihm;
import java.awt.*;import java.awt.event.*;
import javax.swing.*;
public class TestEcouteur3 extends JFrame  {
  private JButton b1 = new JButton("Bouton 1"),
      b2 = new JButton("Bouton 2");
  public TestEcouteur3() {
    JPanel panel;
    getContentPane().add(panel = new JPanel());
    panel.add(b1);panel.add(b2);
    b1.addActionListener(new ActionListener() {
          public void actionPerformed(ActionEvent e) {
          System.out.println("Clic sur b1 !");} });
    b2.addActionListener(new ActionListener() {
          public void actionPerformed(ActionEvent e) {
          System.out.println("Clic sur b2 !");} });
    pack();setVisible(true);
  }
  public static void main(String[] args) {
    TestEcouteur3 t = new TestEcouteur3();
  }
}
```

Listing 14 – coursSSI3/exemples/ihm/TestEcouteur3.java

Fermer une fenêtre

- Pour terminer l'application à la fermeture de la fenêtre, on ajoute dans le constructeur de la fenêtre un écouteur :

```
package coursSSI3.exemples.ihm;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class FermetureFenetre1 extends Fenetre {

  public FermetureFenetre1() {
    addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e){
        System.out.println(
         "L'utilisateur tente de fermer la fenetre");
        System.exit(0);
      }
    });
  }

  public static void main(String[] args) {
    FermetureFenetre1 f = new FermetureFenetre1();
  }

}
```

Listing 15 – coursSSI3/exemples/ihm/FermetureFenetre1.java

Fermer une fenêtre

```
package coursSSI3.exemples.ihm;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class FermetureFenetre2 extends Fenetre {
        public FermetureFenetre2() {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); }

        void windowClosing(WindowEvent e){
                System.out.println(
                  "L'utilisateur tente de fermer la fenetre");}

        void windowClosed(WindowEvent e) {
                System.out.println("La fenetre est fermee");}

        public static void main(String[] args) {
                FermetureFenetre2 f = new FermetureFenetre2();}
}
```

Listing 16 – coursSSI3/exemples/ihm/FermetureFenetre2.java

- Autres actions (`WindowConstants`) `DO_NOTHING_ON_CLOSE`, `HIDE_ON_CLOSE`, `DISPOSE_ON_CLOSE`

Événements clavier

- Un événement clavier de bas niveau est engendré par une action élémentaire de l'utilisateur, par exemple, appuyer sur une touche du clavier (génère un appel à la méthode keyPressed() de KeyListener)
- Plusieurs actions élémentaires de l'utilisateur peuvent engendrer une seul événement de haut niveau ; ainsi, appuyer et relâcher sur les touches Shift et A pour obtenir un A majuscule, génère un appel à la méthode keyTyped() de KeyListener

KeyEvent

- Cette classe correspond aux événements (evt) engendrés par l'utilisation du clavier
- 3 types d'événements repérés par evt.getID() :
  - KeyEvent.KEY_PRESSED et KeyEvent.KEY_RELEASED sont des événements de bas niveau et correspondent à une action sur une seule touche du clavier
  - KeyEvent.KEY_TYPED est un événement de haut niveau qui correspond à l'entrée d'un caractère Unicode (peut correspondre à une combinaison de touches comme Shift-A pour A)

KeyEvent

- Si on veut repérer la saisie d'un caractère Unicode, il est plus simple d'utiliser les événements de type KEY_TYPED
- Pour les autres touches qui ne renvoient pas de caractères Unicode, telle la touche F1 ou les flèches, il faut utiliser les événements KEY_PRESSED ou KEY_RELEASED

KeyListener

public interface KeyListener extends EventListener void keyPressed(KeyEvent e) ; void keyReleased(KeyEvent e) ; void keyTyped(KeyEvent e) ;

- Si on n'est intéressé que par une des méthodes, on peut hériter de la classe KeyAdapter
- Dans ces méthodes, on peut utiliser les méthodes getKeyChar() (dans keyTyped) et getKeyCode() (dans les 2 autres)

Exemple de KeyListener

```
package coursSSI3.exemples.ihm;
import java.awt.*;import java.awt.event.*;import javax.swing.*;
public class TestClavier extends Fenetre {
  JButton b;
  public class ecouteurK extends KeyAdapter {
    public void keyTyped(KeyEvent e) {
      System.out.println(e.getKeyChar());
      b.add(new Label(
       new Character(e.getKeyChar()).toString()));
      pack();
    }
    public void keyReleased(KeyEvent e) {
      if (e.getKeyCode() == KeyEvent.VK_ESCAPE)
          System.out.println("Escape");
    }
  }
  public TestClavier() {
    JPanel panel;
    getContentPane().add(panel = new JPanel());
    panel.add(b = new JButton("bouton 1"));
    addKeyListener(new ecouteurK()); pack();
  }
  public static void main(String[] args) {
    TestClavier t = new TestClavier(); } }
```

Listing 17 – coursSSI3/exemples/ihm/TestClavier.java

Exemple de menu

```
package coursSSI3.exemples.ihm;
import java.awt.*;import java.awt.event.*;import javax.swing.*;
public class MenuExemple extends JFrame implements ActionListener {
 public MenuExemple() {
  setTitle("Exemple de menu");setSize(320, 240);
  MenuBar barreMenu = new MenuBar();
  setMenuBar(barreMenu);
  Menu menuFichier = new Menu("Fichier");barreMenu.add(menuFichier);
  Menu menuAide = new Menu("Aide"); barreMenu.add(menuAide);
  MenuItem menuOuvrir = new MenuItem("Ouvrir");
  menuOuvrir.addActionListener(this);
  menuFichier.add(menuOuvrir);
  MenuItem menuEnregistrer = new MenuItem("Enregistrer");
  menuEnregistrer.addActionListener(this);
  menuFichier.add(menuEnregistrer);
  MenuItem menuQuitter = new MenuItem("Quitter");
  menuQuitter.addActionListener(this);
  menuFichier.add(menuQuitter);
  this.setVisible(true);
 }
 public void actionPerformed(ActionEvent e) {
  String menuSelectionne = e.getActionCommand();
  if (menuSelectionne.equals("Quitter")) {
   this.setVisible(false);this.dispose();
   System.exit(0);
  } else if (menuSelectionne.equals("Enregistrer")) {
   System.out.println("ENREGISTREMENT");
  } else if (menuSelectionne.equals("Ouvrir"))
   System.out.println("OUVERTURE");
 }
```