Programmation orientée objet en langage JAVA

Chapitre 7 : Les gestionnaires d'événements

Claude Duvallet

Université du Havre
UFR Sciences et Techniques
25 rue Philippe Lebon - BP 540
76058 LE HAVRE CEDEX
Claude.Duvallet@gmail.com
http://litis.univ-lehavre.fr/~duvallet/

Les gestionnaires d'événements

Interfaces

- Elles sont semblables à des classes abstraites sans aucune implémentation.
- En quelque sorte, elles ne regroupents que des méthodes abstraites.
- On peut aussi y définir des constantes.
- Les classes peuvent implémenter les interfaces et définir les méthodes qui y sont exposées.
- Pour définir une interface, on utilise le mot-clef `interface`.
- Pour définir une classe utilisant une interface, on utilise le mot-clé `implements`.
- Une classe peut implémenter 0 ou plusieurs interfaces ce qui permet de simuler l'héritage multiple.

Interfaces : exemple

- Exemple d'interface :

```
    public interface Comparaison{
private int nombre = 10;
        public boolean estEgal (Object o);
        public boolean estDifférent (Object o);
        public boolean superieurOuEgal (Object o);
        public boolean inferieurOuEgal (Object o);
    }
```

- Utilisation de l'interface Comparaison lors de la création d'une classe :

```
    public class NombrePremier implements Comparaison{
        public boolean estEgal (Object o){
            // Code
        }
        public boolean estDifférent (Object o){
            // Code
        }
        public boolean superieurOuEgal (Object o){
            // Code
        }
        public boolean inferieurOuEgal (Object o){
            // Code
        }
    }
```

La gestion des événements avec le JDK 1.0 (1/2)

- Principe :
  - pour effectuer une action sur les événements en provenance de l'utilisateur, il faut coder les méthodes de type `action()` ou la méthode `handleEvent()`.
  - si ces méthodes retournent `true`, l'événement est consommé et il n'est pas propagé dans la hiérarchie de composants graphiques. Sinon, l'événement remonte la chaine des contenants.
- Deux alternatives pour le programmeur :
  1. sous-classer les composants graphiques standards de l'AWT (`Button, Panel, etc.`) et coder les méthodes `action()` ou la méthode `handleEvent()`.
  2. capturer tous les événements en un seul endroit avec une grande méthode `handleEvent()` placée dans le conteneur principal.

La gestion des événements avec le JDK 1.0 (2/2)

- Inconvénients du modèle :
  - profusion excessive de classes dérivées uniquement pour des besoins de gestion des événements.
  - pas de séparation nette entre l'interface utilisateur et les traitements fonctionnels qui lui sont associés.
  - pas de filtre des événements : ils sont automatiquement distribués aux composants qu'ils les gèrent ou non.

$\Rightarrow$ composants non réutilisables.

$\Rightarrow$ mauvaises performances.

$\Rightarrow$ problèmes de mise au point.

$\Rightarrow$ programmation peu fiable.

La gestion des événements avec le JDK 1.1

- Un nouveau modèle basé sur la délégation :
  - les composants délèguent la gestion des événements utilisateur à une classe extérieure.
- Avantages :
  1. Il n'est plus nécessaire de créer une classe par composant IHM.
  2. Le filtrage devient plus intelligent : chaque composant ne transmet à l'application que les événements qu'elle attend.
  3. Il est possible de séparer les traitement fonctionnels des événements de l'interface utilisateur.
  4. La gestion des événements est plus rigoureuses : absence de codes de retour.

Principe de la délégation

- Les événements sont des objets (`java.event.EventObject`) qui forment une hiérarchie d'événements.
- Une source d'événement (un composant graphique dérivé de `Component`) émet un événement vers un "délégué" capable de le traiter.
- Le délégué indique qu'il est intéressé par un événement en implémentant une (ou plusieurs) interface spécifique dérivant de (`java.util.EventListener`).
- Pour relier effectivement la source et le délégué, le délégué doit préalablement s'enregistrer auprès de la source.

Exemple : événement à partir d'un bouton (Version 1)

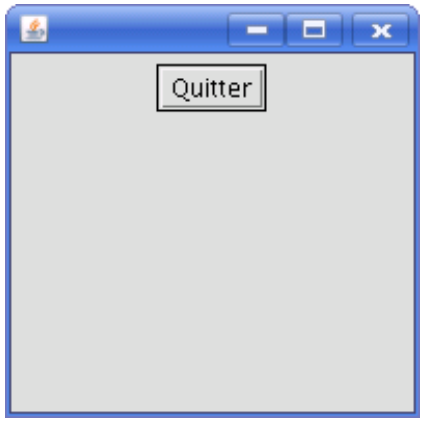
- Fenêtre contenant un bouton qui ferme l'application :

  La classe `FenetreAvecBouton` :

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

public class FenetreAvecBouton extends JFrame{

    public FenetreAvecBouton (){
        Jpanel pane = new JPanel ();
        JButton unBouton = new JButton("Quitter");
        pane.add (unBouton);
        unBouton.addActionListener(new MonListener())
        setContentPane(pane);
    }

    public static void main (String args []) {
        new FenetreAvecBouton().setVisible (true);
    }
}
```

La classe `MonListener` :

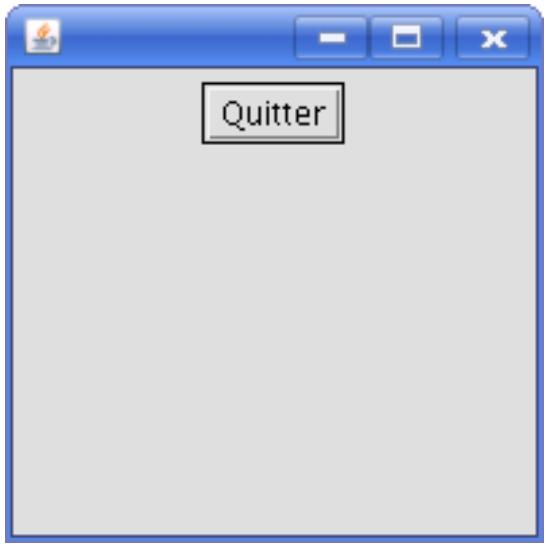
```
import java.awt.*;
import java.awt.event.*;

public class MonListener implements ActionListener {

    public void actionPerformed (ActionEvent ae) {
        System.exit (0);
    }
}
```



Exemple : événement à partir d'un bouton (Version 2)

- La fenêtre principale est transformé en écouteur d'événements et on vérifie qui a émis l'événement :

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

public class FenetreAvecBouton extends JFrame
                               implements ActionListener {

    private JButton unBouton;

    public FenetreAvecBouton (){
        Jpanel pane = new JPanel ();
        JButton unBouton = new JButton("Quitter");
        pane.add (unBouton);
        unBouton.addActionListener(this)
        setContentPane(pane);
    }


    public void actionPerformed (ActionEvent ae) {
        if (e.getSource()==unBouton)
            System.exit (0);
    }

    public static void main (String args []) {
        new FenetreAvecBouton().setVisible (true);
    }
}
```



Les écouteurs d'événements (1/2)

- Afin de rendre l'interface graphique réceptive aux évènements utilisateurs, il est nécessaire de créer des écouteurs d'évènements (listener) associés aux composants :
  - clique sur des boutons,
  - gestion du clavier,
  - évènements liés à la souris,
  - ...
- Une action de l'utilisateur avec un composant génère un évènement :
  - cliquer sur un bouton,
  - sélectionner un élément dans une liste,
  - ...

Les écouteurs d'événements (2/2)

- Réception des évènements utilisateurs :
  - il est nécessaire de créer des écouteurs d'évènements, associés aux composants,
  - les évènements contiennent des informations :
    - le type de l'évènement,
    - le composant qui a généré l'évènement :
    - la méthode `getSource()` détermine le composant qui a envoyé l'évènement.
    - le package `java.awt.event.*` contient tous les écouteurs.
- Chaque écouteur gère un type d'évènements précis :
  - action sur un composant : `ActionListener`
  - lorsqu'un composant est ajusté : `AdjustementListener`
  - composant reçoit ou perd le focus : `FocusListener`
  - lorsqu'une case à cocher est modifiée : `ItemListener`
  - saisie de texte au clavier : `KeyListener`
  - clics de souris, entrée ou sortie d'un pointeur : `MouseListener`
  - mouvement de souris : `MouseMotionListener`

Les événements graphiques

- Pour chaque catégorie d'événements, il existe une interface qui doit être définie par toute classe souhaitant recevoir cette catégorie événements.
  - cette interface exige que toutes les méthodes soient définies.
  - ces méthodes sont appelées lorsque des événements particuliers surviennent.

Catégories d'événements (1/2)

Catégorie	**Interface**	**Méthode**
Action	*ActionListener*	*actionPerformed*
Item	*ItemListener*	*itemStateChanged*
Key	*KeyListener*	*keyPressed* *keyReleased* *keyTyped*
Focus	*FocusListener*	*focusGained* *focusLost*
Adjustment	*AdjustmentListener*	*adjustmentValueChanged*
Text	TextListener	*textValueChanged*

Catégories d'événements (2/2)

Catégorie	**Interface**	**Méthode**
Mouse	*MouseListener*	*mousePressed*
		mouseReleased
		mouseEntered
		mouseExcited
		mouseClicked
	MouseMotionListener	*mouseDragged*
		mouseMoved
Window	*WindowListener*	*windowClosed*
		windowOpened
		windowIconified
		windowDeiconified
		windowActivated
		windowDeactivated

Ajouter un écouteur

- Pour ajouter un écouteur à un composant : add
  - `addActionListener`
    - `JButton, JCheckBox, JComboBox, JTextField et JRadioButton.`
  - `addAdjustementListener`
    - `JScrollBar`
  - `addFocusListener`
    - tous les composants.
  - `addItemListener`
    - `JCheckBox, JRadioButton et JComboBox.`
  - `addKeyListener`
    - saisie de texte au clavier.
  - `addMouseListener, addMouseMotionListener`
    - tous les composants.