Gestion des Evénements

Ilhem BOUSSAID

USTHB

26 novembre 2009

Plan

Les bases

- Les swing sont des **JavaBeans**
- Ils communiquent donc grâce aux **événements**
- Les événements permettent d'associer un **traitement** à une **action**

Parcours des événements :

- Plusieurs boucles envoient des événements dans une file unique **System.getSystemEventQueue()**
- Un Thread lit la file et distribue les événements à la racine de l'arbre des composants associés à la fenêtre de l'événement
- L'événement est acheminé au composant auquel il est destiné

Les événements

- Programmation par événement
  1. Un écouteur s'abonne à une source.
  2. La source déclenche l'événement.
  3. La source le passe aux abonnés.
  4. Les abonnés réagissent.
- Les composants sont des sources.
- Événements déclenchés par une action utilisateur.

Evénements élémentaires

- Tous les constituants de l'interface sont sensibles à ces événements.
- Il s'agit des événements du clavier et de la souris.
- Les événements correspondants sont les suivants :

Type d'événement	événement	classe ou interface	classe d'événement	nom de méthode associée
Clavier	touche appuyée	KeyAdapter KeyListener	KeyEvent	keyPressed
	touche lâchée			keyReleased
	touche tapée			keyTyped
Souris	clic	MouseAdapter MouseListener	MouseEvent	mouseClicked
	le curseur rentre dans le composant			mouseEntered
	le curseur sort du composant			mouseExited
	un bouton de la souris appuyé			mousePressed
	un bouton de la souris lâché			mouseReleased
	souris déplacée avec un bouton appuyé	MouseMotionAdapter MouseMotionListener	MouseMotionEvent	mouseDragged
	souris déplacée			mouseMoved

Evénements de composants d'interface

- Les événements de composants d'interface se produisent lorsque l'on coche une case, on sélectionne un élément dans une liste, etc.
- Les événements correspondants sont les suivants :

Type d'événement	Evénement	Classe d'événement
Action	action sur le composant (clic sur un bouton, choix dans un menu ...)	**ActionEvent**
Ajustement	déplacement d'un ascenseur	**AdjustmentEvent**
Changement	modification de la position d'un composant	**ChangeEvent**
Elément	modification de l'état d'un élément (liste, case à cocher...)	**ItemEvent**
Document	modification dans un texte	**DocumentEvent**
Curseur	déplacement du curseur d'insertion dans un texte	**CaretEvent**
Sélection	sélection d'un élément dans une liste	**ListSelectionEvent**

Evénements de composants d'interface

Type d'événement	événement	interface associée	*classe d'événement*	nom de méthode associée
Action	activation de bouton, case cochée, choix dans un menu ou choix d'un fichier, saisie de texte	**ActionListener**	**ActionEvent**	**actionPerformed**
Ajustement	modification de la position de l'ascenseur	**AdjustmentListener**	**AdjustmentEvent**	**adjustmentValueChanged**
Changement	modification de la position d'un curseur ou d'une barre de progression	**ChangeListener**	**ChangeEvent**	**stateChanged**
Elément	sélection d'un élément	**ItemListener**	**ItemEvent**	**itemStateChanged**
Document	modification, insertion ou suppression de texte	**DocumentListener**	**DocumentEvent**	**changedUpdate removeUpdate insertUpdate**
Curseur	déplacement du curseur d'insertion	**CaretListener**	**CaretEvent**	**caretUpdate**
Sélection	sélection d'un ou plusieurs éléments	**ListSelectionListener**	**ListSelectionEvent**	**valueChanged**

Plan

Event Listeners

- Event Listeners : objets qui **écoutent** (détectent) les événements
  - à chaque classe d'**Event** correspond une classe d'**EventListener**
- exemple :
  - **ActionListener** : sensible aux **ActionEvent**(s)
  - **KeyListener** : sensible aux **KeyEvent**(s)
  - etc.
- **Cas particulier** : deux Listeners pour les **MouseEvent**(s) :
  - MouseListener
  - MouseMotionListener

Méthodes des XxxListeners

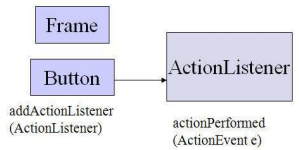
- définies par le programmeur
- **automatiquement** appelées quand l'événement correspondant se produit
- Exemple :
  - Evénement : ActionEvent
  - Listener : ActionListener
  - Méthode : actionPerformed(ActionEvent) ....
- Les méthodes d'un XxxListener
  - récupèrent le XxxEvent en argument
  - doivent toutes être implémentées car les Listeners sont des Interfaces
- Enregistrement des Listeners
  - **addXxxListener(XxxListener)**
  - **removeXxxListener(XxxListener)**

Associer un écouteur à un événement

- Pour chaque élément pour lequel on veut traiter les événements il faudra :
  - Créer une classe obtenue, selon le cas, par **implémentation d'une interface** ou par **héritage d'un modèle d'écouteur (Adapter)** correspondant au type d'événement que l'on désire traiter.
  - Y écrire les actions associées aux événements dans les méthodes correspondantes.
  - associer cette classe au composant par la méthode **addxxxListener** de celui-ci. On utilisera selon le cas **addActionListener**, **addMouseListener** ... (se reporter aux tableaux précédents pour les noms)

Associer un écouteur à un événement

- Chaque classe composant définit **add<nom>Listener(<nom>Listener)**
- Exemple :



Exemple

Définition à l'extérieur

```
public class MaFrame extends JFrame {
MaFrame()
{
monJButton.addMouseListener(new MonMouseListener());
/**
* MonMouseListener Ecouteur = new MonMouseListener();
* monJButton.addMouseListener(Ecouteur);
**/
}
...
}

public class MonMouseListener implements MouseListener {
public void mousePressed(MouseEvent e) { }
public void mouseReleased(MouseEvent e) { }
public void mouseEntered(MouseEvent e) { }
public void mouseExited(MouseEvent e) { }
public void mouseClicked(MouseEvent e) {
System.exit(0);
}
}
```


Exemple

Définition dans la classe qui gère l'action

```
public class MaFrame extends JFrame implements MouseListener {

MaFrame()
{
monJButton.addMouseListener(this);
}

public void mousePressed(MouseEvent e) { }
public void mouseReleased(MouseEvent e) { }
public void mouseEntered(MouseEvent e) { }
public void mouseExited(MouseEvent e) { }
public void mouseClicked(MouseEvent e) {
System.exit(0);
}
}
```


Plan

Les adaptateurs

- L'écouteur défini à partir d'une **interface XXXListener** doit écrire toutes les méthodes même si elles sont vides.
- Pour chaque interface **XXXListener** possédant **plusieurs méthodes**, Java fournit une classe particulière **XXXAdapter**, appelée **adaptateur**, qui implémente toutes les méthodes de l'interface avec un corps vide.
- Pour définir un écouteur d'événements de type **XXXEvent**, il suffit alors de **dériver** l'écouteur de la classe **XXXAdapter** et de **redéfinir uniquement les méthodes voulues**.

Les adaptateurs

- Les adaptateurs d'événement portent le suffixe Adapter (au lieu de Listener).
  - Interfaces : MouseListener, KeyListener
  - Classes : MouseAdapter, KeyAdapter
- Pour l'écouteur défini par héritage d'une classe Adapter, seules les méthodes utiles sont écrites (les autres sont héritées vides).
- Exemple :

```
public class MaFrame extends MouseAdapter {
JFrame frame;
MaFrame(){
monJButton.addMouseListener(this);
}

public void mouseClicked(MouseEvent e) {
System.exit(0);
}
}
```


Variante

```
class MaFenetre extends JFrame {
public MaFenetre () {
super("Une fenetre qui traite les clics souris") ;
setSize(300, 200) ;
addMouseListener(new EcouteurSouris());
}
}

class EcouteurSouris extends MouseAdapter {
/*redéfinition uniquement de la méthode appelée lors d'un clic souris*/
public void mouseClicked(MouseEvent e) {
System.out.println("clic dans la fenetre") ;
}
}
```


Différences

```
public class MaFrame extends JFrame implements
    MouseListener{

MaFrame()
{
    monJButton.addMouseListener(this);
}

public void mousePressed(MouseEvente){}
public void mouseReleased(MouseEvente){}
public void mouseEntered(MouseEvente){}
public void mouseExited(MouseEvente){}
public void mouseClicked(MouseEvente){
    System.exit(0);
}
}
```

```
public class MaFrame extends MouseAdapter{

JFrame frame;
MaFrame()
{
    monJButton.addMouseListener(this);
}

public void mouseClicked(MouseEvente){
    System.exit(0);
}
}
```


Plan

Les classes internes (Inner classes)

- Une classe est dite interne lorsque sa définition est située à l'intérieure d'une autre classe (ou d'une méthode).
- Exemple

```
class MaFenetre extends JFrame {
public MaFenetre () {
super("Une fenetre qui gere sa fermeture") ;
setSize(300, 200) ;
addWindowListener(new EcouteurFermer());
}

/*classe interne à la classe MaFenetre*/
class EcouteurFermer extends WindowAdapter {
public void windowClosing(WindowEvent e) {
System.exit(0); }
} /*fin classe interne*/
}
```


Les classes internes (Inner classes)

- Un objet d'une classe interne est toujours associé, au moment de son instanciation, à un objet d'une classe externe dont on dit qu'il lui a donné naissance.
- Un objet d'une classe interne a toujours accès aux champs et méthodes (même privés) de l'objet externe lui ayant donné naissance.
- Un objet d'une classe externe a toujours accès aux champs et méthodes (même privés) de l'objet interne auquel il a donné naissance.

Les classes anonymes

- Une **classe anonyme** est une classe sans nom. Elle peut **dériver d'une autre classe**.
- Exemple

```
class MaFenetre extends JFrame {
public MaFenetre () {
super("Une fenetre qui gere sa fermeture") ;
setSize(300, 200) ;
addWindowListener(new WindowAdapter() {
/*classe anonyme dérivant de la classe WindowAdapter*/
public void windowClosing(WindowEvent e) {
System.exit(0); }
}) ; /*fin classe anonyme*/
}
}
```


Les Classes internes anonymes

Le mécanisme des classes internes anonymes est adapté au cas de figure où l'on a besoin à un instant donné d'une instance de classe, et on est certain de ne pas avoir à créer d'autres instances ailleurs, ni à référencer cette classe plus loin.

Le mécanisme consiste à appeler le constructeur et à construire en même temps la classe. L'appel au constructeur ne nécessite pas le nom de la classe : on indique le nom de la **classe héritée** ou de l'**interface implémentée** par la classe interne anonyme.

Les classes anonymes

- Par exemple, la déclaration d'une classe interne anonyme pour la gestion du clic sur le bouton donne quelque chose comme ceci :

```
class MaFenetre extends JFrame {
private JButton UnBouton ;
public MaFenetre () {
super("Une␣fenetre␣avec␣un␣bouton") ;
setSize(300, 200) ;
UnBouton = new JButton("Un␣bouton") ;
getContentPane().add(UnBouton) ;

UnBouton.addActionListener(new ActionListener(){
/*classe anonyme implémentant l'interface ActionListener*/
public void actionPerformed(ActionEvent e) {
System.exit(0); }
} ) ; /*fin classe anonyme*/
}
}
}
```