Toute la vérité sur les évènements de l'AWT 1.1

Alain PEIGNIER
peignier@mygale.org

Les classes permettant de travailler avec des fenêtres dans un environnement graphique (AWT) fonctionnent depuis la version 1.1.1 de Java selon un nouveau modèle d'événements. Le plus dur n'est pas de comprendre le fonctionnement très simple de cette technique, c'est plutôt de bien l'utiliser.

Les événements dans l'AWT

Le but ici n'est pas refaire un réchauffé de la doc Java disponible sur le Net. Attention, anglophobes sévères, apprendre la langue de Shakespeare au plus vite.

- http://www.javasoft.com/products/jdk/1.1/docs/api/Package-java.awt.html
- http://www.javasoft.com/products/jdk/1.1/docs/guide/awt/index.html
- http://java.sun.com/docs/books/tutorial/post1.0/ui/eventmodel.html

Pour synthétiser : Java utilise des événements qui ne sont pas directement ceux que produit la plate-forme. Adieu constantes entières qui circulent de fonction en fonction au bon gré des librairies, Java implémente un système tout beau tout propre et fait maison. Toutes les victimes du Borland C++ sous Windows ont déjà compris l'intérêt de la manoeuvre : la SIMPLICITÉ.

Plus clair, se serait en effet transparent; en voici la preuve :

Les Events

Les événements sont les porteurs d'information. Ceux qui renseignement votre application par leur existence d'une part et par les informations qu'ils peuvent contenir (touche de clavier appuyée, nom de l'action déclenchée, position de la souris).

On décompose les événements en plusieurs types :

- ActionEvent (cf classe http://www.javasoft.com/products/jdk/1.1/docs/api/java.awt.event.ActionEvent.html)
- KeyEvent (cf classe http://www.javasoft.com/products/jdk/1.1/docs/api/java.awt.event.KeyEvent.html)
- WindowEvent (cf classe http://www.javasoft.com/products/jdk/1.1/docs/api/java.awt.event.WindowEvent.html)
- FocusEvent (cf classe http://www.javasoft.com/products/jdk/1.1/docs/api/java.awt.event.FocusEvent.html)
- MouseEvent (cf classe http://www.javasoft.com/products/jdk/1.1/docs/api/java.awt.event.MouseEvent.html)
- ItemEvent (cf classe http://www.javasoft.com/products/jdk/1.1/docs/api/java.awt.event.ItemEvent.html)

Les événements MouseEvent et KeyEvent permettent de réceptionner les actions de l'utilisateur. Les événements WindowEvent permettent de gérer le comportement des fenêtres. Les événements FocusEvent quant eux permettent de prévenir un composant graphique (bouton, menu, liste ...) que le focus lui arrive, s'en va ... Les événements ItemEvent sont un peu particuliers et ne s'appliquent qu'aux composants ou l'on peut sélectionner quelque chose (cf interface java.awt.ItemSelectable) les List, Choice ... Et pour finir par les plus importants : les événements ActionEvent qui permettent de faire ce qui vous voulez. On les utilise en particulier pour les boutons (cf classe java.awt.Button) qui lorsqu'on les déclenche envoient un événement ActionEvent a qui veut l'entendre.

Il manque encore la moitie des acteurs de la gestion des événements :

les Listeners

Les Listeners sont les récepteurs des événements. Ce sont les méthodes chargées d'analyser et de traiter les informations de l'événement qu'on leur soumet.

Évidemment chaque Event trouve son réceptacle dans l'une des interfaces suivantes :

- ActionListener (cf classe [])
- KeyListener (cf classe [])
- WindowListener (cf classe [])
- FocusListener (cf classe [])
- MouseListener (cf classe [])
- ItemListener (cf classe [])

Comme vous avez dû le remarquer dans la documentation Java, ce sont des interfaces. Vous êtes donc libre d'implémenter les méthodes nécessaires où bon vous semble, ce que nous verrons plus tard.

Pour qu'un Listener reçoive des événements, il faut signaler son existence à la ou les source(s) d'Events que l'on veut gérer. Pour cela, les sources de l'AWT sont toutes dotées de méthodes addTypeListener et removeTypeListener. Dès lors qu'un Listener est ajouté par la méthode addTypeListener, il fait parti d'une file d'attente dont chaque participant est mis au courant de chaque TypeEvent provenant de la source (cf méthode processEvent).

```
private Vector actionListeners = new Vector();

public void addActionListener(ActionListener actionListener) {
  actionListeners.addElement(actionListener);
}

public void removeActionListener(ActionListener actionListener) {
  actionListeners.removeElement(actionListener);
}

protected void processActionEvent(ActionEvent event) {
  for (Enumeration e=actionListeners.elements(); e.hasMoreElement();)
    ((ActionListener)e.nextElement()).actionPerformed(event);
}
```

Traitement des Listeners

On peut par ce mécanisme "enregistrer" un Listener auprès de plusieurs sources. Les événements lui parviendront et il pourra utiliser la méthode AWTEvent.getSource() (cf classe java.awt.AWTEvent) pour déterminer quelle source est concernée.

On peut aussi "enregistrer" plusieurs Listeners auprès d'une même source. Attention à ne peut enregistrer deux fois un même listener chez une source ...

Vivre en paix avec des Listeners et des Events

Le modèle est simple. reste à l'appliquer avec autant de succès.

Le choix le plus important est l'endroit où l'on implante les méthodes des Listeners. Toute l'astuce consiste à minimiser le nombre de classes utilisées.

Premier exemple

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

public class ButtonWithEvent extends Frame implements ActionListener {

  public final static String QUITTER = "Quitter";

  public ButtonWithEvent() {
    super("Button With Event");

    Button b = new Button(QUITTER);
    b.addActionListener(this);
    add(b);

    pack();
  }

  public void actionPerformed(ActionEvent e) {
    String actionCommand = e.getActionCommand();
    if (actionCommand.equals(QUITTER)) dispose();
  }
}
```

On se sert en fait de la fenêtre comme support aux méthodes des listeners. L'astuce fonctionne bien pour les Listeners qui nécessite une seule méthode, c'est a dire ActionListener et ItemListener.

Pour KeyListener, WindowListener, FocusListener, il y a plusieurs méthodes définies dans l'interface alors qu'on n'utilise généralement qu'une seule d'entre elles : par exemple pour le KeyListener, la méthode la plus utilisée est keyPressed(KeyEvent) et on néglige bien souvent les deux autres.

Heureusement, les gentils créateurs de Java ont pense a nos petits doigts et ont laisse traîner trois classes : KeyAdapter, WindowAdapter et FocusAdapter. Elles ne font que implémenter les

fonctions des Listeners correspondant pour des méthodes vides, mais elles sont bel et bien salvatrices : il ne reste qu'a hériter le ou les méthodes intéressante(s).

```
      static class keyAdapter extends KeyAdapter {

        private ButtonWithEvent fenetre;

        public KeyAdapter (ButtonWithEvent fenetre) {
          this.fenetre = fenetre;
        }

        public void keyPressed(KeyEvent e) {
          switch (e.getKeyCode()) {
            case KeyEvent.ESCAPE : fenetre.dispose();
          }
        }
      }
```

A ajouter a la classe ButtonWithEvent précédemment décrite.

Et ajouter dans le constructeur b.addKeyListener(new keyAdapter(this));

Une technique directement due a la syntaxe de Java permet d'instancier une classe, tout en déclarant certaines variables et certaines méthodes.

```
      WindowAdapter l = new WindowAdapter() {

        public void windowClosing(WindowEvent e) {
          ((Window)e.getSource()).dispose();
        }
      }

      addWindowAdapter(l);
```

Instanciation et déclaration simultanée d'un WindowAdapter

On peut faire encore plus fort et tout implémenter en posant par des ActionEvent. Chaque événement est traite et transforme en un ActionEvent correspondant. Par exemple, le WindowListener transforme le WINDOWCLOSING en ActionEvent QUITTER, le KeyListener, le KeyEvent.ESCAPE en ActionEvent QUITTER.

La fenetre devient une antenne à événements, les transforme et ne gère alors que des ActionEvents. Toute la gestion est ainsi centralise autour de la méthode actionPerformed.

```
      static class keyAdapter extends KeyAdapter {

        private ButtonWithEvent fenetre;

        public KeyAdapter (ButtonWithEvent fenetre) {
          this.fenetre = fenetre;
```

```
    }

    public void keyPressed(KeyEvent e) {
      String actionCommand = "";

      switch (e.getKeyCode()) {
        case KeyEvent.ESCAPE : actionCommand = fenetre.QUITTER;
      }

      if (actionCommand.length() > 0) {
        ActionEvent aEvent = new ActionEvent(this,
          ActionEvent.ACTIONPERFORMED,actionCommand);
        fenetre.actionPerformed(aEvent);
      }
    }
  }
```

Voir la librairie Swing ou/et mourir

Dans les futurs composants graphiques de l'AWT, l'usage des événements passe au cran supérieur. L'apparition des nouveaux événements, en particulier, ChangeEvent permet d'implémenter facilement des interfaces intelligentes avec des composants interagissant entre eux.

On notera aussi l'apparition d'une gestion plus aisée des raccourcis clavier qui se branchent directement sur des ActionEvents.

Pour ceux qui croient avoir tout compris sur les événements, découvrez vite l'enfer des Events et des Listeners : la JTable de la libraire Swing, la plus grosse concentration d'événements au Web-metre carre.

Vous trouverez tout ça dans l'api Swing (Java Foundation Classes).

Petite note de fin

Pour tout commentaire, remarque, question, erreur infâme : mailto:peignier@mygale.org.