Gestion des évènements

Table des matières

Gestion de la souris (1)

Nous allons créer une série d'applets qui vont réagir à différents évènements générés par la souris. Ces évènements se divisent en deux groupes : évènements concernant un lieu particulier, évènements concernant un déplacement de la souris. Les applets se mettront à l'écoute de ces évènements, cela se fera en utilisant des interfaces spécifiques.

Notion d'interface

Une interface est un ensemble de déclarations de variables et de de méthodes. Lorsqu'une classe implémente une interface, elle donne une existence réelle à ces variables et méthodes. La notion d'interface permet ainsi de remplacer en la simplifiant celle d'héritage multiple qu'on rencontre en C++.

Lorsqu'une classe implémente une interface, elle le déclare grâce au mot-clé **implements**. Ceci donne :

```
public class ma_classe implements une_interface {
....
}
```

Le compilateur java n'effectuera correctement son travail que si les variables et méthodes annoncées dans l'interface sont bien présentes dans la classe.

Il est possible d'utiliser plusieurs interfaces simultanément, il suffit de les déclarer après le mot-clé implements séparées par des virgules.

L'interface MouseListener

L'interface MouseListener permet à une classe de répondre aux évènements souris de base. Elle est définie dans le package java.awt.event qu'il faut donc importer. Elle contient les méthodes suivantes :

```
public void mouseClicked(MouseEvent e) {}
//Invoked when the mouse has been clicked on a component.

public void mousePressed(MouseEvent e) {}
//Invoked when a mouse button has been pressed on a component.

public void mouseReleased(MouseEvent e) {}
//Invoked when a mouse button has been released on a component.

public void mouseEntered(MouseEvent e) {}
//Invoked when the mouse enters a component.

public void mouseExited(MouseEvent e) {}
//Invoked when the mouse exits a component.
```

Ces méthodes ont comme paramètre un objet de type MouseEvent; celui-ci possède deux méthodes getX() et getY() qui renvoient les coordonnées du point où se trouve la souris.

Un exemple

Créons une applet nommée click qui affiche le mot Bonjour à l'endroit où l'utilisateur clique.

Entête et déclarations

On importe les packages nécessaires et on déclare l'utilisation de l'interface MouseListener.

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

public class click extends java.applet.Applet
 implements MouseListener {
 ....
}
```

Champs utilisés et affichage

L'applet click utilisera 3 champs : le contenu du message à afficher et les coordonnées de l'endroit où doit s'effectuer l'affichage.
On déclare donc ces trois champs et on écrit la méthode paint qui les utilise.

```
String msg="Bonjour";
int xmsg=20;
int ymsg=20;

public void paint(Graphics g) {
 g.drawString(msg,xmsg,ymsg);
}
```

La méthode paint peut être complétée pour encadrer le message affiché.

Installation d'un écouteur

Pour pouvoir gérer les évènements souris il est nécessaire de déclarer un écouteur, c'est à dire une classe qui implémente l'interface MouseListener. Dans notre cas c'est l'applet elle-même qui sera l'écouteur. Cette déclaration peut se faire dans la méthode init :

```
public void init() {
 addMouseListener(this);
}
```

Rappel : this représente la classe que l'on définit donc ici l'applet.

Note : il est aussi possible de définir une classe spéciale implémentant l'interface MouseListener.

La méthode init peut être complétée pour définir des couleurs d'affichage et récupérer un message passé en paramètre à l'applet.

Définition des méthodes de l'interface MouseListener

Nous n'utiliserons pour cet exemple que la méthode mouseClicked. Il est cependant nécessaire de déclarer les autres en les laissant vides. Ceci nous donne :

```
        public void mouseClicked(MouseEvent e) {
        //Invoked when the mouse has been clicked on a component.
         xmsg=e.getX();
         ymsg=e.getY();
         repaint();
        }

        public void mousePressed(MouseEvent e) {}
        //Invoked when a mouse button has been pressed on a component.

        public void mouseReleased(MouseEvent e) {}
        //Invoked when a mouse button has been released on a component.

        public void mouseEntered(MouseEvent e) {}
        //Invoked when the mouse enters a component.

        public void mouseExited(MouseEvent e) {}
        //Invoked when the mouse exits a component.
```

La méthode mouseClicked enregistre donc les coordonnées de la souris dans les champs prévus à cet effet. Elle appelle ensuite la méthode repaint de l'applet qui provoque un réaffichage par un appel à la méthode update qui efface le contenu avec la couleur de fond et appelle la méthode paint que nous avons définie.

Gestion de la souris (2)

Nous allons créer une applet qui va réagir aux déplacements de la souris. Elle affichera un message qui pourra être déplacé en suivant la souris. Pour permettre cette opération nous utiliserons l'interface MouseMotionListener.

L'interface MouseMotionListener

L'interface MouseMotionListener permet à une classe de répondre aux évènements liés au déplacement de la souris. Elle est définie dans le package java.awt.event qu'il faut donc importer. Elle contient les méthodes suivantes :

```
        public void mouseDragged(MouseEvent e) {
        //Invoked when a mouse button is pressed on a component and then
dragged.
        //Mouse drag events will continue to be delivered to the component
        //where the first originated until the mouse button is released
        //(regardless of whether the mouse position is within the bounds of
the component).
        }

        public void mouseMoved(MouseEvent e) {
        //Invoked when the mouse button has been moved on a component (with
no buttons no down).
        }
```

Ces méthodes ont comme paramètre un objet de type MouseEvent; celui-ci possède deux méthodes getX() et getY() qui renvoient les coordonnées du point où se trouve la souris.

Un exemple

Créons une applet nommée drag qui affiche le mot Bonjour et permet de le déplacer avec la souris.

Entête et déclarations

On importe les packages nécessaires et on déclare l'utilisation des interfaces MouseListener et MouseMotionListener.

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

public class drag extends java.applet.Applet
 implements MouseListener,MouseMotionListener {
 ....
}
```

Champs utilisés et affichage

L'applet click utilisera 3 champs : le contenu du message à afficher et les coordonnées de l'endroit où doit s'effectuer l'affichage.
On déclare donc ces trois champs et on écrit la méthode paint qui les utilise.

```
String msg="Bonjour";
int xmsg=20;
int ymsg=20;

public void paint(Graphics g) {
 g.drawString(msg,xmsg,ymsg);
}
```

La méthode paint peut être complétée pour encadrer le message affiché.

Installation des écouteurs

Pour pouvoir gérer les évènements souris il est nécessaire de déclarer deux écouteurs, c'est à dire des classes qui implémentent les interfaces MouseListener et MouseMotionListener.
Dans notre cas c'est l'applet elle-même qui fournira les écouteurs. Cette déclaration peut se faire dans la méthode init :

```
public void init() {
 addMouseListener(this);
 addMouseMotionListener(this);
}
```

Rappel : this représente la classe que l'on définit donc ici l'applet.

Note : il est aussi possible de définir des classes spéciales implémentant les interfaces MouseListener et MouseMotionListener.

La méthode init peut être complétée pour définir des couleurs d'affichage et récupérer un message passé en paramètre à l'applet.

Définition des méthodes de l'interface MouseListener

Nous n'utiliserons que la méthode mousePressed qui permettra d'afficher le message à l'endroit où l'on a cliqué. Ceci nous donne :

```
public void mouseClicked(MouseEvent e) {}
//Invoked when the mouse has been clicked on a component.

public void mousePressed(MouseEvent e) {
//Invoked when a mouse button has been pressed on a component.
 xmsg=e.getX();
 ymsg=e.getY();
 repaint();
}

public void mouseReleased(MouseEvent e) {}
//Invoked when a mouse button has been released on a component.

public void mouseEntered(MouseEvent e) {}
//Invoked when the mouse enters a component.

public void mouseExited(MouseEvent e) {}
//Invoked when the mouse exits a component.
```

La méthode mousePressed enregistre donc les coordonnées de la souris dans les champs prévus à cet effet, puis elle appelle ensuite la méthode repaint de l'applet.

Définition des méthodes de l'interface MouseListener

Nous n'utiliserons que la méthode mouseDragged qui gère les déplacements de la souris qui ont lieu avec le bouton gauche pressé. Ceci nous donne :

```
public void mouseDragged(MouseEvent e) {
 //vérifier que la souris est dans la zone
 if (contains(e.getX(),e.getY())) {
  xm=e.getX();
  ym=e.getY();
  repaint();
 }
}

public void mouseMoved(MouseEvent e) {}
```

La méthode mouseDragged vérifie que le pointeur de la souris se trouve bien à l'intérieur de l'applet et dans ce cas enregistre la nouvelle position d'affichage et appelle la méthode repaint pour redessiner l'applet.

Gestion du clavier

Nous allons créer une applet qui va réagir à l'appui sur des touches du clavier. Elle permettra d'afficher un mot de 10 lettres. Pour permettre cette opération nous utiliserons l'interface KeyListener.

L'interface KeyListener

L'interface KeyListener permet à une classe de répondre aux évènements liés au clavier (appui sur une touche, relachement, ...) Elle est définie dans le package java.awt.event qu'il faut donc importer. Elle contient les méthodes suivantes :

```
public void keyTyped(KeyEvent e) {
//Invoked when a key has been typed.
//This event occurs when a key press is followed by a key release.
}

public void keyPressed(KeyEvent e) {
//Invoked when a key has been pressed.
}

public void keyReleased(KeyEvent e) {
//Invoked when a key has been released.
}
```

Ces méthodes ont comme paramètre un objet de type KeyEvent; celui-ci possède des méthodes permettant de savoir quelle touche a été utilisée. On trouve en particulier :
- **getKeyChar()** qui renvoie le caractère associé à la touche.
- **getKeyCode()** qui renvoie un code sous forme d'entier de type int pour les touches qui ne sont pas associées à un caractère : flèches, F1, F2, ..., Suppr, etc...

Les codes renvoyés par getKeyCode() correspondent à des constantes définies dans la classe KeyEvent. On a par exemple : VK_CANCEL (touche Echap), VK_ENTER (touche Entrée), VK_UP (flèche haut), etc... Il suffit de consulter la documentation du JDK (classe KeyEvent) pour obtenir la liste complète des codes utilisables.

Un exemple

Créons une applet nommée testclavier qui permet à l'utilisateur d'entrer un mot de 10 lettres au clavier.

Entête et déclarations

On importe les packages nécessaires et on déclare l'utilisation de l'interface KeyListener.

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

public class testclavier extends java.applet.Applet
 implements KeyListener {
 ....
}
```

Champs utilisés et affichage

L'applet testclavier utilisera 1 champ unique sous forme de String qui contiendra le mot de 10 lettres à afficher.
On déclare donc ce champ et on écrit la méthode paint qui l'utilise.

```
String S="";

public void paint(Graphics g) {
 g.drawString("Ecrire un mot de 10 lettres...",10,30);
 g.drawString(S, 10, 60);
}
```

Installation de l'écouteur

Pour pouvoir gérer les évènements clavier il est nécessaire de déclarer un écouteur (classe qui implémente l'interface KeyListener). Dans notre cas c'est l'applet elle-même qui fournira l'écouteur. Cette déclaration peut se faire dans la méthode init :

```
public void init() {
 addKeyListener(this);
}
```

Problème du focus

Les évènements clavier ne parviennent à l'applet que lorsque celle-ci a le focus. Cela se produit lorsqu'on clique sur sa surface, mais l'applet peut peut aussi demander le focus au système en utilisant la méthode requestFocus(). C'est ce que nous ferons dans la méthode start().

```
public void start() {
 requestFocus();
}
```

Définition des méthodes de l'interface KeyListener

Nous n'utiliserons que la méthode keyPressed qui enregistrera le caractère sur lequel l'utilisateur a appuyé. Ceci nous donne :

```
// Saisie des touches tapées au clavier
public void keyPressed(KeyEvent evt) {
 char Caract = evt.getKeyChar();
 if (S.length()<10)
  if (Caract!=0) {
   S = S + Caract;
   repaint();
  }
}

public void keyTyped(KeyEvent evt) { }

public void keyReleased(KeyEvent evt) { }
```

La méthode keyPressed récupère le caractère envoyé et l'ajoute à la chaîne à afficher si celle-ci ne contient pas encore 10 caractères.

Ecouteurs d'évènements

Il y a plusieurs interfaces écouteurs d'évènements dans le package java.awt.event.

ActionListener

- **Evènements** : clic, touche entrée, sélection d'un élément
- **Méthode** :
  - actionPerformed(ActionEvent e)
- **Composants** : Button, List, MenuItem, TextField avec la méthode addActionListener

AdjustementListener

- **Evènements** : déplacement du curseur d'une barre de défilement
- **Méthode** :
  - adjustmentValueChanged(AdjustmentEvent e)
- **Composants** : Scrollbar avec la méthode addAdjustementListener

ComponentListener

- **Evènements** : déplacement, affichage, masquage ou modification de taille de composants
- **Méthodes** :
  - componentHidden(ComponentEvent e)
  - componentMoved(ComponentEvent e)
  - componentResized(ComponentEvent e)
  - componentShown(ComponentEvent e)
- **Composants** : Component avec la méthode addComponentListener

ContainerListener

- **Evènements** : ajout ou suppression d'un composant dans un conteneur
- **Méthodes** :
  - componentAdded(ContainerEvent e)
  - componentRemoved(ContainerEvent e)
- **Composants** : Container avec la méthode addContainerListener

FocusListener

- **Evènements** : obtention ou perte du focus par un composant
- **Méthodes** :
  - focusGained(FocusEvent e)
  - focusLost(FocusEvent e)
- **Composants** : Component avec la méthode addFocusListener

ItemListener

- **Evènements** : sélection dans une liste ou dans un groupe de cases à cocher
- **Méthodes** :
    - itemStateChanged(ItemEvent e)
- **Composants** : Checkbox, CheckboxMenuItem, Choice, List avec la méthode addItemListener

KeyListener

- **Evènements** : action sur une touche du clavier (pressée ou relachée)
- **Méthodes** :
    - keyPressed(KeyEvent e)
    - keyReleased(KeyEvent e)
    - keyTyped(KeyEvent e)
- **Composants** : Component avec la méthode addKeyListener

MouseListener

- **Evènements** : clic sur bouton, déplacement du pointeur
- **Méthodes** :
    - mouseClicked(MouseEvent e)
    - mouseEntered(MouseEvent e)
    - mouseExited(MouseEvent e)
    - mousePressed(MouseEvent e)
    - mouseReleased(MouseEvent e)
- **Composants** : Component avec la méthode addMouseListener

MouseMotionListener

- **Evènements** : évènements de glisser-déplacer
- **Méthodes** :
    - mouseDragged(MouseEvent e)
    - mouseMoved(MouseEvent e)
- **Composants** : Component avec la méthode addMouseMotionListener

TextListener

- **Evènements** : modification du texte d'un composant texte
- **Méthodes** :
    - textValueChanged(TextEvent e)
- **Composants** : TextComponent avec la méthode addTextListener

WindowListener

- **Evènements** : fenêtre activee, désactivée, réduite, fermée, ...
- **Méthodes** :
    - windowActivated(WindowEvent e)
    - windowClosed(WindowEvent e)
    - windowClosing(WindowEvent e)

    - windowDeactivated(WindowEvent e)
    - windowDeiconified(WindowEvent e)
    - windowIconified(WindowEvent e)
    - windowOpened(WindowEvent e)
- **Composants** : Window avec la méthode addWindowListener

Exercices

1. **Ardoise magique**
   Créer une applet permettant de dessiner en pilotant un curseur à l'aide des touches fléchées du clavier. La touche Suppr doit permettre d'effacer le dessin.
   Idée : *créer deux tableaux de 1000 entiers qui contiennent les abscisses et les ordonnées des différents points à relier pour obtenir le dessin.*
2. **Changement de couleurs**
   Créer une applet qui affiche un texte en couleur, qui change de couleur lorsqu'on appuie sur le bouton de la souris et qui rétablit la couleur d'origine lorsqu'on relâche le bouton de la souris.
3. **Gestion du Focus**
   Ecrire une applet qui change de couleur de fond lorsqu'elle reçoit ou perd le focus. (chercher dans la documentation l'interface FocusListener)
4. **Segment**
   Ecrire une applet qui dessine un segment et qui permet de déplacer ses extrémités à la souris.