Partie I : Programmation graphique en Swing

Programmation objet 2 (2012-2013)

Chapitre 4 Evénements

Gestion des évènements

1. Evénements
2. Gestion des événements
3. Ecouteurs

Evénements

- En sciences, il s’agit d’un changement d’état ou de contexte, lié à une modification substantielle de la valeur d'un paramètre mesurable, dans un intervalle de temps bref à l'échelle de l'expérience

Gestion des événements

- Swing utilise un design pattern (Observable-Observer) pour la gestion des événements
- Ce design pattern contient trois éléments :
  - Composant graphique (Observable)
    - Source d'évènement
    - Associé à une liste des écouteurs
    - notifie les écouteurs lorsque un évènement se produit
  - Evénement
    - Une action sur un composition graphique
  - Ecouteur (Observer)
    - traite des événements

Evénements et écouteurs

- Types d'événements et d'écouteurs
  - ActionEvent /ActionListener
  - MouseEvent /MouseListener, MouseMotionListener
  - WindowEvent /WindowListener

ActionEvent et ActionListener

- ActionEvent correspond au déclanchement de l'action sur un composant graphique:
  - Le clic sur un bouton
  - La frappe de la touche « enter » sur un textField
  - Etc.
- http://docs.oracle.com/javase/1.4.2/docs/api/java/awt/event/ActionEvent.html

Classe interne : JetDeEvent1

```
JetDePanel() {
    // creer deux De
    deGauche  = new DePanel();
    deDroit = new DePanel();
    // creer deux buttons
    jetButton1 = new JButton("Joueur1");
    jetButton2 = new JButton("Joueur2");
    // créer une instance de JetListener
    JetListener jetListener = new JetListener();
    jetButton1.addActionListener(jetListener);
    jetButton2.addActionListener(jetListener);
    //... placer components
    this.setLayout(new BorderLayout());
    this.add(jetButton1, BorderLayout.NORTH);
    this.add(deGauche , BorderLayout.WEST);
    this.add(deDroit, BorderLayout.EAST);
    this.add(jetButton2, BorderLayout.SOUTH);
  }
```

- JetListener sous forme de classe interne
- Avantage :
  - accède facilement aux variables d'instance de la classe JetPanel
  - permet plusieurs instances

Classe interne : JetDeEvent1

```
// classe interne : JetListener
   private class JetListener implements
      ActionListener {
       public void actionPerformed(ActionEvent
      e) {
        Object source = e.getSource();
        if (source == jetButton1)
                deGauche.jet();
        else
                deDroit.jet();
      }
   }
}
```

- Les événements écoutés sont de la classe "ActionEvent" qui hérite de la superclasse Event
- Méthodes permettant d'obtenir des informations sur ce qui est survenu
  - getSource() donne le composant source de l'événement
  - getActionCommand() donne le nom d'action, par défaut le texte du bouton, du menu, ...

Classe interne anonyme : JetDeEvent2

```
// creer un button pour jetter les deux De et
 ajouter listener
jetButton = new JButton("Jet");
// classes interne anonyme
jetButton.addActionListener(new ActionListener
() {
   public void actionPerformed(ActionEvent e) {
   System.out.println("clic sur bouton : "
            +"\nsource : "+e.getSource().toString()
            +"\naction command :
 "+e.getActionCommand().toString()
             +"\nActionEvent : "+e.toString());
            deGauche.jet();
            deDroit.jet();
   }
}
);
```

- JetListener sous forme de classe interne anonyme
- Avantage :
  - accède facilement aux variables d'instance de la classe JetDePanel
- Inconvénient :
  - n'a qu'1 instance
  - code moins lisible

MouseEvent et MouseListener

- La souris (le pointeur) passant sur la surface graphique du composant, içi un JPanel, provoque des événements :
  - mouvement
  - entrée/sortie du panel
  - enfoncement d'un bouton
- Le programme MouseEvent.java affiche ces événements en console :

```
mouseMoved - MouseEvent : java.awt.event.MouseEvent
[MOUSE_MOVED(197,0),button=0,clickCount=0] on
javax.swing.JPanel[,0,0,200x100,layout=java.awt.FlowLayout,
alignmentX=0.0,alignmentY=0.0,border=,flags=9,maximumSize=,
minimumSize=, preferredSize=java.awt.Dimension
[width=200,height=100]] mouseExited - MouseEvent :
java.awt.event.MouseEvent[MOUSE_EXITED(197,-
1),button=0,clickCount=0] on javax.swing.JPanel[,
0,0,200x100,layout=java.awt.FlowLayout,
alignmentX=0.0,alignmentY=0.0,border=,flags=9,maximumSize=,
minimumSize=, preferredSize=java.awt.Dimension
[width=200,height=100]]
```

MouseEvent et MouseListener

```
panel.addMouseListener(new MouseListener() {
     public void mouseClicked(MouseEvent me) {
          System.out.println("mouseClicked - MouseEvent : "+me.toString());
     }
     public void mouseEntered(MouseEvent me) { … idem }
     public void mouseExited(MouseEvent me) { … idem }
     public void mousePressed(MouseEvent me) { … idem }
     public void mouseReleased(MouseEvent me) { … idem }
});
```

Les méthodes des MouseListener :

- mouseEntered() pour un événement MouseEvent « entrée de souris »
- mouseExited() « sortie de souris »
- mousePressed() « bouton pressé »
- mouseReleased() « bouton relaché »
- mouseClicked() « bouton pressé puis relaché dans sa zone graphique »

Ces méthodes permettent de recevoir et d'écouter les événements qui se sont produits

MouseEvent et MouseMotionListener

```
panel.addMouseMotionListener(new MouseMotionListener() {
        public void mouseMoved(MouseEvent me) { … idem }
        public void mouseDragged(MouseEvent me) { … idem }
});
```

Les méthodes desMouseMotionListener :

- mouseMoved() pour un événement MouseEvent « mouvement de souris »
- mouseDragged() «mouvement de souris avec bouton enfoncé »

L'événement en paramètre donné des méthodes est celui qui s'est produit :
il contient des informations de ce qui s'est passé.