GESTIONE DEGLI EVENTI
in Java 1.1

Gianna Bellè

Aprile 1999

Eventi

Le componenti interagiscono con un programma che le usa, attraverso l'uso di eventi.
Esempio: quando un utente clicca una componente con il mouse, quella componente *fires* un evento di tipo *mouse-clicked*.

Gli eventi nell' Abstract Windows Toolkit sono organizzati in una gerarchia di classi, la cui radice è la classe astratta `AWTEvent`

- `ActionEvent`
- `AdjustmentEvent`
- `ComponentEvent`
  - ...
  - `InputEvent`
    - `KeyEvent`
    - `MouseEvent`
  - ...
- `ItemEvent`
- `TextEvent`

Due categorie di eventi:

- low-level (sottoclassi di `ComponentEvent`)
- high-level o *semantics*

Esempio: cliccare un bottone

Event Source, Event Listeners

Il modello degli eventi in Java è detto *notification-based system*.

Oggetti che *fire* eventi sono detti *event sources*.

Per ogni tipo di evento generato da un *event source*, l'*event source* tiene una lista di *event listener* per quel tipo di evento.

Quando un evento del corrispondente tipo viene generato dal sorgente, l'ascoltatore viene avvisato.

AWT Event Flow

L'evento viene generato da una componente.

Questo evento viene posto in una coda.

Viene dato alla componente via `dispatchEvent()`.

Se l'evento è abilitato, `dispatchEvent()` chiama `processEvent()`, il quale avvisa gli ascoltatori di tale evento.

Abilitare gli eventi

Abilitare un tipo di evento per un sorgente significa che gli eventi di quel tipo saranno passati al metodo `processEvent()` del sorgente.

Un tipo di evento è automaticamente abilitato per un sorgente quando un ascolatore si è registrato presso quel sorgente.

Listener e Adapter

1. creare un EventListener
2. associare un Listener con un Component

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

class MyActionListener implements ActionListener {
  public void actionPerformed(ActionEvent event) {
    doSomeAction();
  }
}

public class MyApplet extends Applet {
  public void init() {
    Button b1 = new Button("Button 1");
    b1.addActionListener(new MyActionListener());
    ...
  }
  ...
}
```

Gestione degli eventi (Java 1.1)

Mirror.java (from Core Web Programming)

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


/** A listener used to send identical text to
 *  two text areas. Used in the Mirror class below.
 */

class MirrorListener extends KeyAdapter {
  private TextArea area1, area2;

  public MirrorListener(TextArea area1,
                        TextArea area2) {
    this.area1 = area1;
    this.area2 = area2;
  }

  public void keyPressed(KeyEvent event) {
    String key = String.valueOf(event.getKeyChar());
    area1.append(key);
    area2.append(key);
    event.consume();
  }
}
```



```
/** An applet that creates two text areas and
 *  attaches the same KeyListener to both.
 */

public class Mirror extends Applet {
  public void init() {
    setLayout(new GridLayout(1, 2));
    TextArea area1 = new TextArea();
    TextArea area2 = new TextArea();
    MirrorListener mirror =
      new MirrorListener(area1, area2);
    area1.addKeyListener(mirror);
    area2.addKeyListener(mirror);
    add(area1);
    add(area2);
  }
}
```

Classi Inner

Definizioni di classi come membri di altre classi, localmente dentro un blocco di statements, oppure dentro un'espressione. Sembra come una chiamata ad un costruttore di classe, ma include la definizione della classe appena dopo le parentesi.

```
public class Outer {
  protected void foo() { ...}

  class Inner {
    public void bar() {
      do SomethingWith(foo());
    }
  }

  private Inner test = new Inner();
  ...
}
```

oppure

```
SomeType st =
  new SomeType() {
    // Class definition
  };
doSomethingWith(st);
```


Vantaggi classi inner

- utili per definire listeners
- hanno pieno accesso a dati e metodi `protected` della classe che le contiene

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

public class DrawCircles4 extends Applet {
  public void init() {
    setForeground(Color.blue);
    MouseListener circleListener =
      new MouseAdapter() {
        private final int radius = 25;

        public void mousePressed(MouseEvent event) {
          Graphics g = getGraphics();
          g.fillOval(event.getX()-radius,
                     event.getY()-radius,
                     2*radius,
                     2*radius);
        }
    };
    addMouseListener(circleListener);
  }
}
```

Gestione degli eventi (Java 1.1)

Low-level Event Processing

```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*; // For MouseEvent

public class MouseReporter extends Applet {
  public void init() {
    enableEvents(AWTEvent.MOUSE_EVENT_MASK |
                 AWTEvent.MOUSE_MOTION_EVENT_MASK);
  }

  public void processMouseEvent(MouseEvent event) {
    System.out.println("Mouse enter/exit or click at (" +
                       event.getX() + "," +
                       event.getY() + ").");
    // In case there are MouseListeners attached:
    super.processMouseEvent(event);
  }

  public void processMouseMotionEvent(MouseEvent event) {
    System.out.println("Mouse move/drag at (" +
                       event.getX() + "," +
                       event.getY() + ").");
    // In case there are MouseMotionListeners attached:
    super.processMouseMotionEvent(event);
  }
}
```