Programmazione III

Interfacce grafiche (seconda parte)

Ardissono Liliana

Dai lucidi del Prof. Alberto Martelli

Vediamo un altro esempio (ButtonTest da Core Java)

Vogliamo realizzare un pannello con tre pulsanti, ognuno dei quali può modificare il colore di sfondo del pannello.

Il pannello verrà inserito in un **JFrame**.

```
class ButtonPanel extends Jpanel {
   public ButtonPanel()
   {
      JButton yellowButton = new JButton("Yellow");
      JButton blueButton = new JButton("Blue");
      JButton redButton = new JButton("Red");

      add(yellowButton);
      add(blueButton);
      add(redButton);
      .........
   }
```

Definiamo i *listener* dei pulsanti come implementazioni della interfaccia **ActionListener**.

E' sufficiente definire una sola classe **ColorAction** che implementa **ActionListener**, definendo il costruttore con un parametro di tipo **Color**, che rappresenta il colore di sfondo che si vuole ottenere.

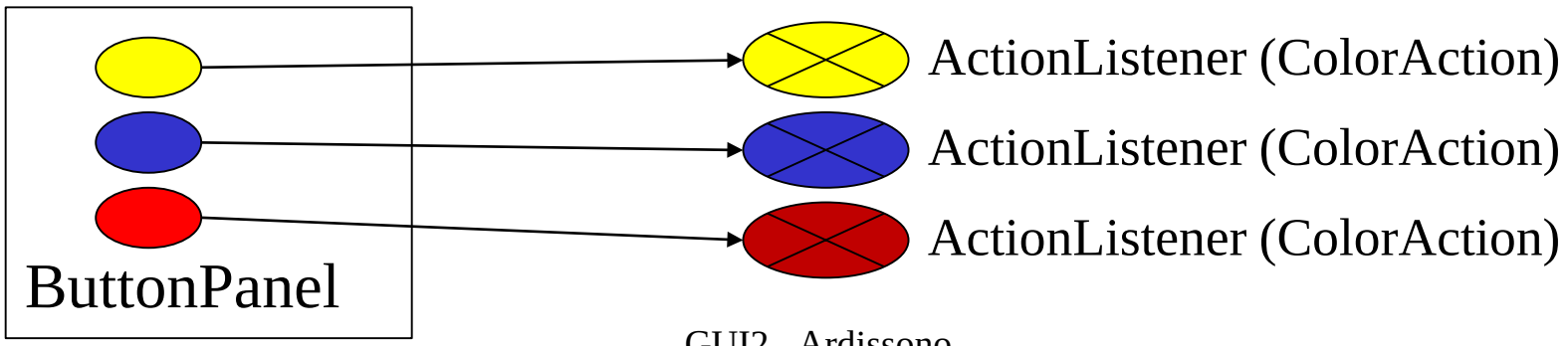
Il metodo *actionPerformed* dovrà eseguire il metodo *setBackground* sul pannello e quindi il costruttore di **ColorAction** dovrà avere anche un altro parametro: il **ButtonPanel** di cui si vuole modificare il colore di sfondo.

```
class ColorAction implements ActionListener
   {
      public ColorAction(Color c, ButtonPanel p)
      {
         backgroundColor = c;
         bp = p;
      }

      public void actionPerformed(ActionEvent event)
      {
         bp.setBackground(backgroundColor);
      }

      private Color backgroundColor;
      private ButtonPanel bp;
   }
```

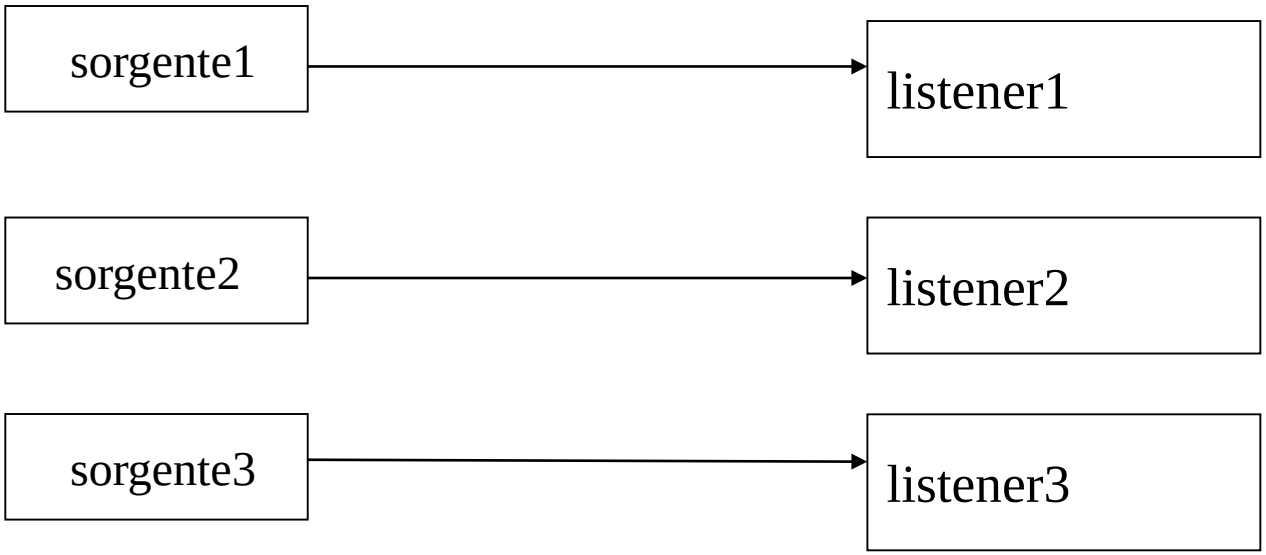
Infine dobbiamo generare tre istanze di **ColorAction** corrispondenti ai tre colori, e registrarle nei pulsanti relativi.

Il costruttore di **ButtonPanel** viene completato come segue:

```
// create button actions
ColorAction yellowAction = new
ColorAction(Color.YELLOW,this);
ColorAction blueAction = new ColorAction(Color.BLUE,this);
ColorAction redAction = new ColorAction(Color.RED,this);

// associate actions with buttons
yellowButton.addActionListener(yellowAction);
blueButton.addActionListener(blueAction);
redButton.addActionListener(redAction);
```



Organizzazione della GUI dell’esempio visto: ogni sorgente di eventi ha il suo listener



Listener come classi interne degli oggetti grafici

Gli oggetti **ColorAction** che implementano i *listener* devono accedere al panel per modificarne lo sfondo. Questo richiede di passare il panel come parametro al costruttore di **ColorAction**.

Però, il listener **`ColorAction`** può essere definito come una **classe interna** della classe **`ButtonPanel`**. In questo caso il parametro **`ButtonPanel`** non serve più, perché **`ColorAction`** può accedere direttamente al metodo **`setBackground`** del pannello.

→ definiamo la classe **ColorAction** come classe interna della classe **ButtonPanel**. In questo modo gli oggetti **ColorAction** possono accedere ai campi e metodi, anche privati, di **ButtonPanel**.

```
class ButtonPanel extends JPanel
{
   ....
   private class ColorAction implements ActionListener
   {
      public ColorAction(Color c)
      {
         backgroundColor = c;
      }

      public void actionPerformed(ActionEvent event)
      {
         setBackground(backgroundColor);
      }

      private Color backgroundColor;
   }
   ...
}
```

Se una classe è usata una sola volta noi possiamo evitare di darle un nome, usando la notazione delle **classi anonime**. Il codice è più sintetico e leggibile

```
public class Beeper extends JFrame {
   .....
   Beeper() {
      button = new JButton("Click Me");
      panel = new JPanel();
      panel.add(button);
      add(panel);
      button.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e){
            Toolkit.getDefaultToolkit().beep();}
      });
   ....
}
```

Abbiamo già visto che un *listener* può essere implementato da un componente (**JFrame**, **JPanel**, ...) definendo i metodi del *listener* nel componente stesso.

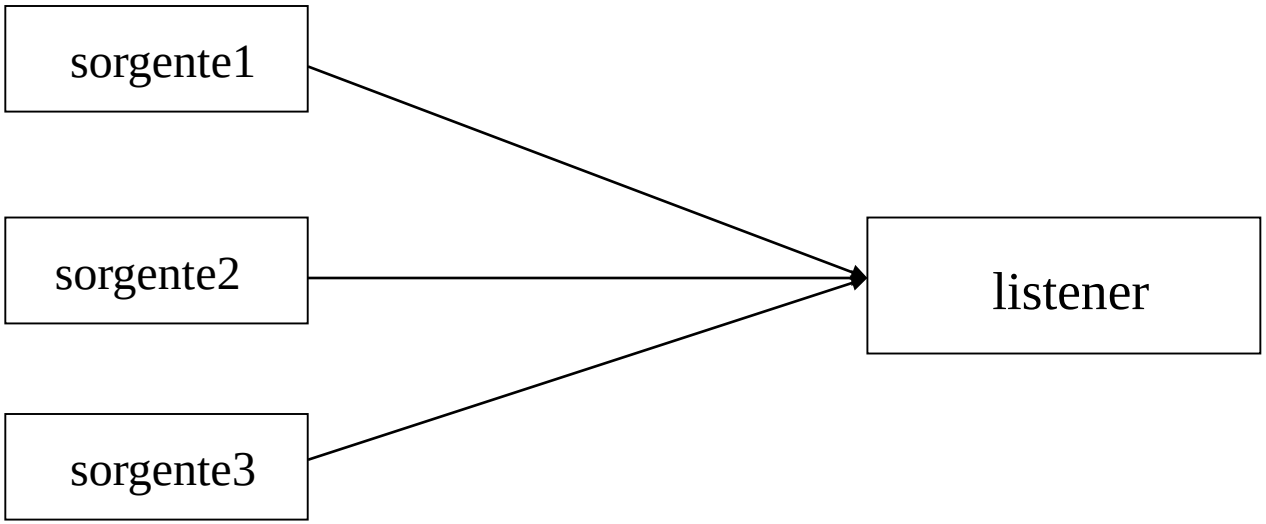
Nel caso della finestra che cambia il colore dello sfondo, non possiamo però procedere direttamente in questo modo perché

•il programma usa tre istanze di **ColorAction**, ciascuna con il suo metodo **actionPerformed**,

•ma esiste un solo **ButtonPanel** → ci può essere un solo metodo **actionPerformed** associato a questa classe.

Si può comunque modificare il programma in modo che ci sia un solo metodo **actionPerformed** della classe **ButtonPanel**, che funge da *listener* per tutti i bottoni.

Ecco quindi una possibile organizzazione alternativa della GUI dell'esempio visto: un solo listener gestisce gli eventi di molteplici sorgenti



```
class ButtonPanel extends JPanel implements ActionListener {
   public ButtonPanel() {
      .......
      .......
      yellowButton.addActionListener(this);
      blueButton.addActionListener(this);
      redButton.addActionListener(this);
   }


   public void actionPerformed(ActionEvent event) {
      String command = event.getActionCommand();
      if (command.equals("Yellow"))
                           setBackground(Color.YELLOW);
      else if (command.equals("Blue"))
                           setBackground(Color.BLUE);
      else if (command.equals("Red"))
                           setBackground(Color.RED);
   }
}
```

In questo esempio, un unico *listener*, il **ButtonPanel**, si registra presso i tre pulsanti.

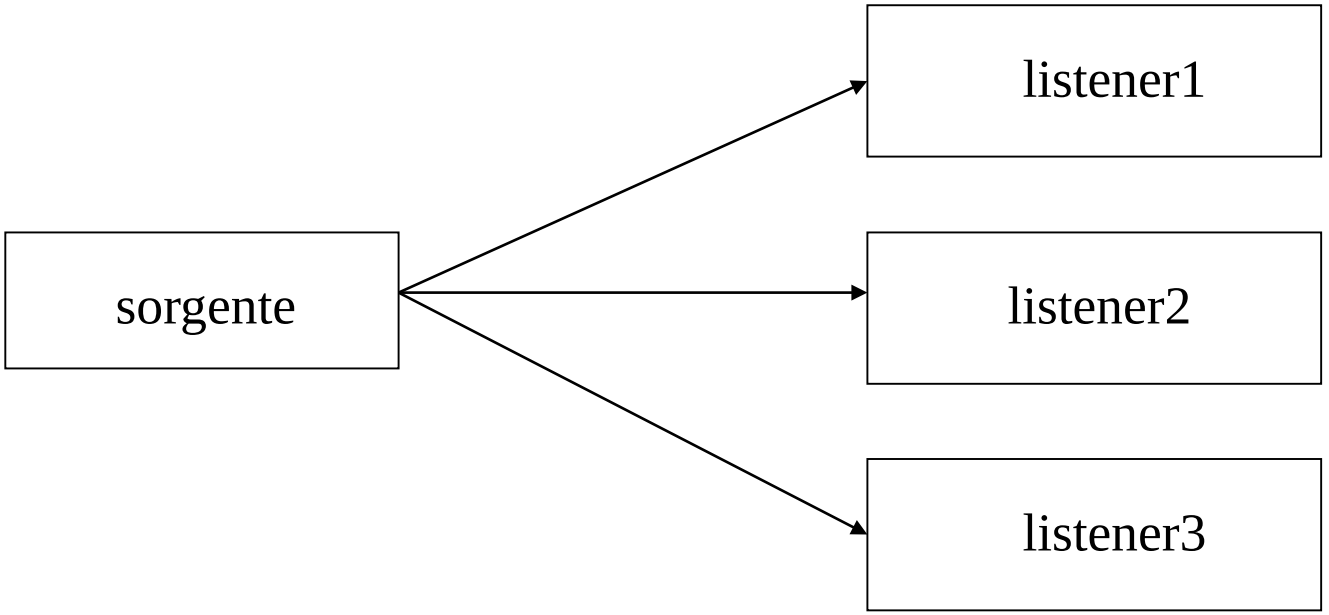
Quando un pulsante genera un **ActionEvent**, l'evento viene inviato al ButtonPanel e si attiva il metodo **actionPerformed**.

Questo metodo determina da quale pulsante arriva l'evento, ed esegue l'azione relativa.

Si raccomanda di utilizzare classi interne per realizzare gestori di eventi piuttosto che trasformare le classi esistenti in *listener*.

Potrebbe essere necessario associare più di un listener alla stessa sorgente di eventi, per gestire diverse tipologie di evento generate dalla sorgente, oppure per gestire in modo diverso gli stessi eventi.

L'esempio **MulticastTest** di Core Java realizza la situazione opposta in cui lo stesso evento viene inviato a più di un listener.



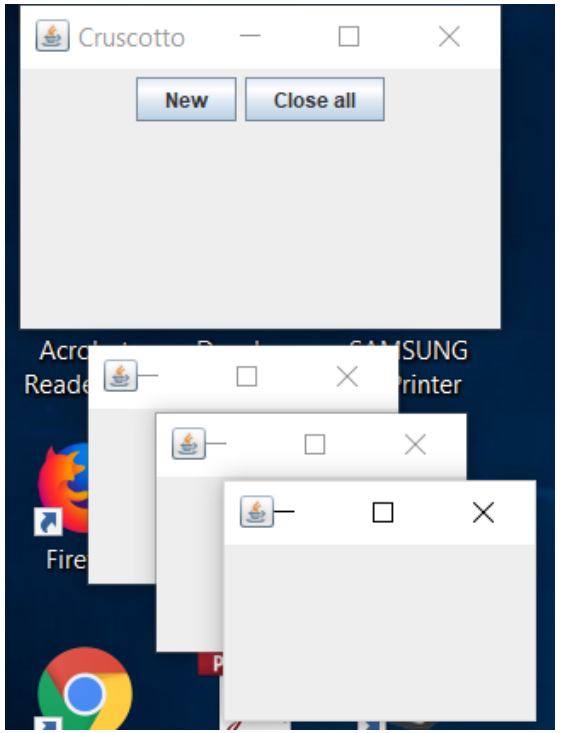
Definiamo un frame con due pulsanti:

•il pulsante **New** genera una nuova finestra (**BlankFrame**) ogni volta che viene premuto

•il pulsante **Close all** chiude tutte le finestre.

Quando si crea un **BlankFrame**, si crea un *listener* il cui metodo *actionPerformed* chiude il frame a cui è associato, e si registra questo *listener* nel pulsante **Close all**.

Quando **Close all** viene premuto, l'evento generato viene inviato a tutti i *listener*, che chiudono il frame relativo.



```
class MulticastPanel extends Jpanel {
   public MulticastPanel() {
     JButton newButton = new JButton("New");
     add(newButton);
     final JButton closeAllButton =
                                  new JButton("Close all");
     add(closeAllButton);

     ActionListener newListener =
        new ActionListener() {
          public void actionPerformed(ActionEvent event) {
             BlankFrame frame =
                        new BlankFrame(closeAllButton);
             frame.setVisible(true);
          }
        };
     newButton.addActionListener(newListener);
   }
}
```

```
class BlankFrame extends Jframe {
   public BlankFrame(final JButton closeButton) {
      ......

      closeListener = new
        ActionListener() {
          public void actionPerformed(ActionEvent ev){
           closeButton.removeActionListener(closeListener);
           dispose();
          }
        };
       closeButton.addActionListener(closeListener);
   }

   private ActionListener closeListener;
   ...
}
```

Layout

Java fornisce diversi manager di layout predefiniti, fra cui:

- **FlowLayout -** i componenti vengono inseriti uno dopo l'altro e riga per riga.
- **BorderLayout -** i componenti vengono inseriti in posizione nord, sud, est, ovest e centro.
- **GridLayout -** i componenti vengono disposti in una tabella.
- **BoxLayout -** i componenti vengono disposti uno per riga.

Ogni contenitore ha un layout predefinito.

Ad esempio per il *JPanel* è *FlowLayout*.

E' possibile cambiare il layout con il metodo *setLayout*.

```
JPanel p = new JPanel();
p.setLayout(new GridLayout(4,4));
```