## Eventi di azione

(Interfaces *ActionListener*, Classes ActionEvent)

Sono generati quando si premono bottoni, si selezionano voci di menù, si preme invio mentre si scrive in un campo di testo. In *awt* sono generati da componenti delle classi Button, TextField, List e MenuItem

Molti controlli Swing oltre a JButton generano eventi che possono essere catturati da un *actionListener* cioè sono sorgente di eventi ActionEvent

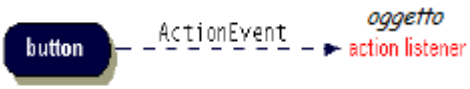
> *Eventi di azione* sono provocati dall'utente che agisce su un componente (click su un pulsante); vale per gli oggetti istanza delle classi JButton, JCheckBox, JComboBox, JRadioButton (controlli derivati da AbstractButton) e JTextField.

I seguenti eventi ad esempio *notificano* un evento azione:

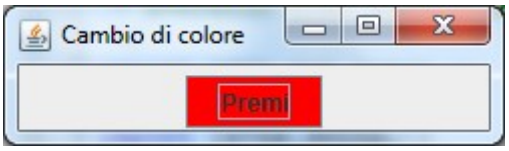
- Quando si premere INVIO in un campo di testo
- Quando si seleziona una checkbox, un radiobutton o una voce in un combobox
- Quando si seleziona una voce di un menu o si clicca su un bottone

Esempio di struttura di **applicazione** con gestione ***eventi*** generati[1] da ***interazione con interfaccia grafica***:

un pulsante (parte della GUI) è **origine** dell'evento (di azione)
tutta la classe è *ascoltatore* dell'evento (di azione)
esiste un solo metodo nell'***interfaccia***




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

public class NomeClasse  implements ActionListener  {     // classe ascoltatore
                                                          // con  interfaccia per eventi di azione
  private JFrame f;
  private Container c;
  private JPanel p;
  private JButton b;

  public NomeClasse(){                              // costruttore

    f = new JFrame ("Cambio di colore");
    c = f.getContentPane();
    p = new JPanel();
    b = new JButton("Premi");
    b.addActionListener (this);        // lega la classe di ascolto cioè l'applicazione stessa
                                       // all'origine dell'evento cioè il pulsante
    b.setBackground (Color.red);
    p.add(b);
    c.add(p);
    f.setLocation(200,200);
    f.setSize(250,70);
    f.setBackground(Color.white);
    f.setVisible(true);
    f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);   // senza esplicita gestione di eventi di finestra
}
```




[1] Altre applicazioni a eventi in Java: applicazioni *embedded* basate su *interrupt* (ad esempio i ***driver di dispositivo***) ed applicazioni distribuite basate su scambio asincrono di eventi (ad esempio quelle di ***network management***)

```
                                                    // implemento il  metodo dell'interfaccia di ascolto
 public void actionPerformed (ActionEvent e)  {   // unico metodo da ridefinire

            b.setBackground (Color.cyan);            // cambia colore di sfondo del bottone
          }

public static void main (String [] args) {

        NomeClasse  o = new NomeClasse  ();
 }
}// fine applicazione
```

## Eventi – metodi utili

| | |
|---|---|
| Object *getSource*() | restituisce la componente su cui si è verificato l'evento e si può scoprire con "*instanceof*" a quale classe appartiene. |
| *getSelectedObjects()[0]* | ritorna array (length 1) con label  o null se non selezionato; usato con operatore di cast (String); in alternativa, per classe Checkbox, anche *getText*() |

• **ActionEvent**:

| | |
|---|---|
| String *getActionCommand*() | se l'evento è generato su un TextField restituisce la stringa contenuta nel TextField in alternativa a *getText*() |

• **KeyEvent**:

| | |
|---|---|
| char *getChar*() | restituisce  il carattere associato al tasto che ha generato l'evento |
| char *getKeyChar*() | |
| int *getKeyCode*() | restituisce  il numero corrispondente (ad esempio il tasto ESC è 27) |

• **MouseEvent**:

| | |
|---|---|
| int *getX*(), int *getY*() | restituiscono le coordinate del cursore |

## Definizione e Registrazione di Listeners

• **Definizione**, due possibilità

- implementando il Listener

```
class MyXXXListener implements XXXListener {
// definizione dei metodi definiti da XXXListener
}
```

- estendendo l'adapter

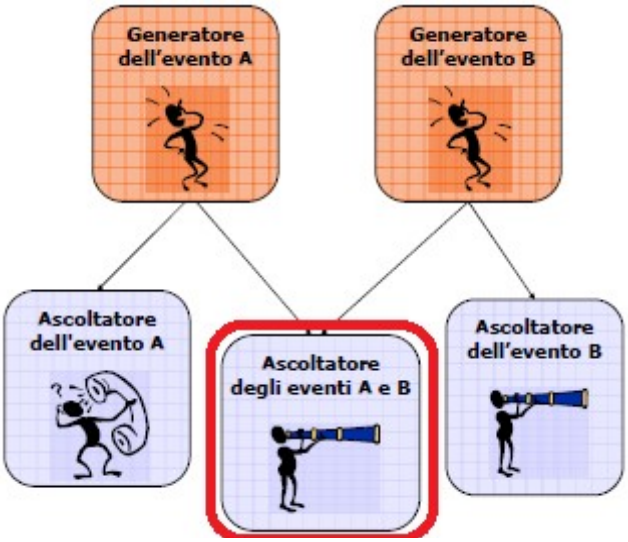
```
class MyXXXListener extends XXXAdapter {
// override i metodi definiti da XXXAdapter
}
```

• **Registrazione**:

- ogni event source ha un metodo per registrare un corrispondente listener (cioè **notificare** all'ascoltatore)
- addXXXListener (XXXListener listenerObject)

È possibile installare uno stesso ascoltatore, ad esempio, per diversi bottoni di una *toolbar*

→ il vantaggio risiede nel fatto che i diversi bottoni hanno lo stesso ascoltatore con **unico metodo di risposta** (invece che più ascoltatori separati che replicano il metodo, a volte per *fare la stessa cosa*)

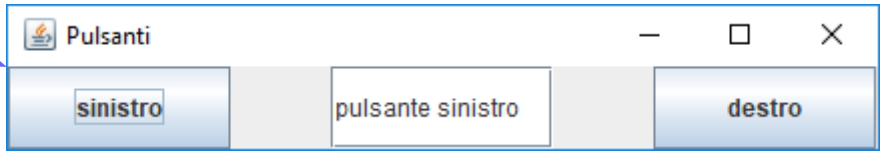



## Registrazione

- addXXXListener
  - registra un ascoltatore associandolo a un componente grafico
- removeXXXListener
  - rimuove l'ascoltatore

**Esercizio**: pulsanti con **stessa classe di ascolto** e **stesso metodo di risposta** **actionPerformed()**

```
// esempio con gestione eventi della GUI
// Uso di due pulsanti

import javax.swing.*;
import java.awt.event.*;
import java.awt.*;

public class Pulsanti extends JFrame {
        Container c;
        JTextField t = new JTextField (30);
        JButton b1 = new JButton("sinistro");
        JButton b2 = new JButton("destro");

        public Pulsanti()  {
            super();
            setTitle("Pulsanti");
            setSize (450, 80);
            setLocation (50,50);
            setResizable(true);                                    // ridimensionabile con mouse
            setLayout(new GridLayout (1,3,50,0));
            c = getContentPane();                          // recupera il "muro grezzo"
            b1.addActionListener (new Ascolta());                                  //  notifica ad ascoltatore anonimo
            b2.addActionListener (new Ascolta());
            c.add(b1);
            c.add(t);
            c.add(b2);
            setVisible(true);
          }
         private class Ascolta implements ActionListener {

           public void actionPerformed (ActionEvent e)  {

              String bottone = e.getActionCommand();          // recupera testo mostrato da pulsante
               if (bottone.equals ("sinistro") )
                     t.setText("pulsante sinistro");
               else
                     t.setText("pulsante destro");
             }
           } // fine class interna
          public static void main(String [] a){
                    new Pulsanti();
         }
      } // fine applicazione
```




**Problema**: scrivere un ascoltatore (*listener*) che alla pressione di bottoni diversi mostri il contenuto della JTextField inserita nella finestra istanza dell'applicazione

→ L'ascoltatore deve aver accesso al riferimento a quel JTextField di quella particolare istanza

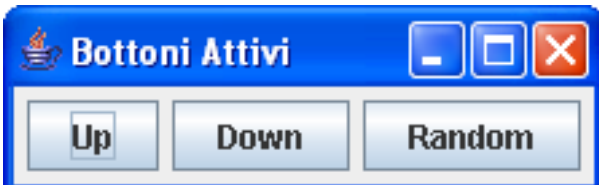
**Prima** soluzione prima adottata: si può utilizzare una **classe *interna*** come *ActionListener*
Tecnica accettabile se l'ascoltatore "fà poco" altrimenti la classe cresce sensibilmente in dimensioni

***Svantaggio***: si accorpano due classi che rappresentano due aspetti concettualmente diversi

**Seconda** soluzione: si può progettare l'ascoltatore prevedendo un costruttore che prende in ingresso il riferimento alla finestra contenente il bottone. La classe resta esterna ma può accedere a tutti gli oggetti della finestra (a patto che questi *non siano privati*)

## Un Listener per N Controlli

**Primo caso:** uso classe Listener **interna**




```
public class MyFrame extends JFrame {
   // ……..
  JButton  up= new JButton("Up");
  JButton down= new JButton("Down");
  JButton random= new JButton("Random");

  Listener ascoltatore = new Listener();  // ascoltatore non anonimo

  public MyFrame() {   ...
        up.addActionListener(ascoltatore);              // componenti attivi della stessa natura
        down.addActionListener(ascoltatore);
        random.addActionListener(ascoltatore);
  // ……..
}

  private class Listener implements ActionListener {            // classe interna

    public void actionPerformed (ActionEvent e) {

        Object src = e.getSource();              // si individua l'oggetto mittente della segnalazione
        if (src == up)  { codice associato alla pressione del bottone Up }
        else if (src == down) { codice associato alla pressione del bottone Down }
        else if (src == random) { codice associato alla pressione del bottone Random }
   }
 }
}
```

Se i componenti sono di **diversa natura**, ad esempio un pulsante ed un campo di testo, entrambi in grado di generare un **evento "di azione"**, il seguente codice può essere utile ad intercettare il mittente della segnalazione :

```
public void actionPerformed (ActionEvent evt){
   Object src = evt.getSource();                          // si individua il mittente della segnalazione
   if (src instanceof JTextField)
                        funzioneCampoDiTesto();   // scelta del giusto gestore dell'evento
  else if (src instanceof Jbutton)
                          funzionePulsante();
}
```

Quando si usano **classi esterne**, è possibile "capire" il componente che ha notificato l'evento associando ai diversi componenti un diverso valore della **proprietà *actionCommand***

*nb:*

```
but[i]=new JButton (mess[i]);
but[i].setToolTipText (mtool[i]);

but[i].setActionCommand (mtool[i] );  // per modificare la stringa identificativa dell'oggetto
                                      // registrato (associato) al gestore d'evento
```

*e.**getActionCommand** () ritorna String → default l'etichetta associata al bottone (mess[i])*

**Secondo caso:** classe Listener **esterna**.

Nell'esempio seguente i possibili diversi valori della proprietà *actionCommand* sono memorizzati come **costanti** della classe ascoltatore Listener **esterna**

```
public class MyFrame extends JFrame {
// …….....

JButton  up= new JButton("Up");
JButton down= new JButton("Down");
JButton random= new JButton("Random");

Listener ascolt = new Listener();

public MyFrame() {
// …....
up.addActionListener(ascolt);
up.setActionCommand(ascolt.UPOPT);
down.addActionListener(ascolt);
down.setActionCommand(ascolt.DOWNOPT);
random.addActionListener(ascolt);
random.setAction Command(ascolt.RANDOMOPT);
// …....
}
}
// classe esterna
public class Listener implements ActionListener {

public final static String UPOPT = "Up";                    // costanti
public final static String DOWNOPT = "Down";
public final static String RANDOMOPT = "Random";

public void actionPerformed(ActionEvent e) {
String com = e.getActionCommand();            // recupero della stringa-etichetta associata al bottone
if (com == UPOPT)
   upOpt();
else if (com == DOWNOPT)
        downOpt();
      else if (com == RANDOMOPT)
              randomOpt();
}
private void upOpt() { ... }
private void randomOpt(){ ... }
private void downOpt(){ ... }

}
```

## Confronto soluzioni:

In senso assoluto, le prestazioni migliori si ottengono usando l'implementazione diretta dell'interfaccia-ascoltatore, poiché non sono generate né classi né oggetti ulteriori, rispetto all'unica concretizzazione del Tipo ascoltatore.

Le classi *interne* hanno il pregio di consentire una migliore gestione della struttura interna dell'oggetto (comportamenti *adatti* in ambito OO), appesantendo il caricamento dell'applicazione di quel (tanto o poco) necessario al ClassLoader per caricare e risolvere un certo numero di classi in più, rispetto alla soluzione dell'implementazione diretta.

Quale che sia la scelta adottata, ciò che importa è la consapevolezza degli effetti.