Fondamenti di informatica
un approccio a oggetti con Java

Luca Cabibbo

Eventi

Capitolo 28

settembre 2001

Contenuti

- Eventi
  - gestione di eventi mediante interfacce
  - gestione di eventi mediante adattatori
- Componenti, eventi e ascoltatori
- Gestione di eventi e IDE

Introduzione

L'interfaccia grafica di una applicazione ha lo scopo di consentire l'interazione tra l'utente e l'applicazione

- l'utente può accedere alle informazioni visualizzate da una interfaccia grafica
- l'utente può richiedere all'applicazione di eseguire delle operazioni agendo sui componenti dell'interfaccia grafica mediante le periferiche di ingresso del calcolatore (in particolare, mouse e tastiera)

Vengono ora studiati

- gli eventi, che sono il meccanismo di gestione delle interfacce grafiche, ovvero delle azioni eseguite dagli utenti

Eventi

L'interfaccia grafica di una applicazione ha lo scopo di

- mostrare informazioni all'utente
- consentire all'utente di richiedere l'esecuzione di operazioni

- ad esempio, in questa interfaccia grafica è possibile riconoscere un bottone e una etichetta
  - il bottone è un elemento dell'interfaccia grafica con cui l'utente può interagire, premendolo (ad es., con il mouse)
  - l'etichetta ha lo scopo di mostrare quante volte il bottone è stato premuto
- ciascuna interazione dell'utente con l'interfaccia grafica dell'applicazione deve essere notificata all'applicazione, al fine di provvedere all'esecuzione delle azioni corrispondenti
- un **evento** è la notifica di una interazione tra l'utente e l'interfaccia grafica

Eventi e gestione di eventi

Quando si verifica un evento, relativamente alla gestione di tale evento sono significativi i seguenti tre oggetti

- un oggetto che rappresenta l'**evento** che si è verificato
  - gli eventi sono rappresentati da oggetti istanza della classe **EventObject** e delle sue sotto-classi
- il componente dell'interfaccia grafica che ha causato l'evento, chiamato la **sorgente dell'evento**
  - ad esempio, un bottone
- un oggetto che deve gestire l'evento, chiamato **gestore dell'evento** o **ascoltatore** (**listener**) **dell'evento**
  - i gestori di eventi sono rappresentati da oggetti istanza di classi che implementano l'interfaccia **EventListener** e le interfacce da essa derivate
  - una interfaccia grafica deve definire uno o più ascoltatori
  - ciascun ascoltatore è dedicato alla gestione di una o più tipologie di eventi che si possono verificare nell'interfaccia grafica

Eventi e gestione di eventi

In pratica, per gestire gli eventi di una interfaccia grafica

- è necessario definire delle classi per implementare ascoltatori per gli eventi di interesse
  - un ascoltatore definisce un metodo per ciascun evento che può verificarsi, che descrive le azioni che devono essere eseguite in corrispondenza al verificarsi di un tale evento
  - una stessa classe può implementare più ascoltatori per tipologie di eventi differenti, in quanto è possibile che una classe implementi più interfacce
- istanziare gli oggetti ascoltatori necessari
- registrare ciascuna sorgente di eventi presso l'ascoltatore corrispondente
  - la registrazione di una sorgente presso un ascoltatore rende l'ascoltatore il responsabile degli eventi generati da quella sorgente
  - gli eventi generati da una sorgente di eventi non registrata non vengono gestiti, ovvero, vengono ignorati

Gestione di eventi mediante interfacce

Viene ora mostrata, mediante un esempio, la realizzazione di un ascoltatore di eventi basata sull'implementazione di una interfaccia

Si vuole realizzare un applet che è rosso se il mouse si trova al suo interno, ed è verde se il mouse si trova al suo esterno

- viene definita una classe che implementa l'applet
- gli eventi significativi sono l'ingresso e l'uscita del puntatore del mouse nella regione dell'applet
  - la sorgente di tali eventi è l'applet (e non il mouse)
- eventi di questo tipo sono di competenza dell'interfaccia **MouseListener** (del package **java.awt.event**)
  - la classe che implementa l'applet viene definita in modo tale da implementare anche l'interfaccia **MouseListener** (in questo modo, l'applet sarà l'ascoltatore dei suoi eventi)

L'interfaccia MouseListener

L'interfaccia **MouseListener** contiene la dichiarazione dei seguenti metodi

- **void mouseClicked(MouseEvent e)**
  - viene invocato quando il mouse viene cliccato in un componente, passandogli come argomento l'oggetto evento corrispondente
- **void mouseEntered(MouseEvent e)**
  - viene invocato quando il mouse entra in un componente
- **void mouseExited(MouseEvent e)**
  - viene invocato quando il mouse esce da un componente
- **void mousePressed(MouseEvent e)**
  - viene invocato quando un tasto del mouse viene premuto in un componente
- **void mouseReleased(MouseEvent e)**
  - viene invocato quando un tasto del mouse viene rilasciato in un componente

Applet che cambia colore

```
import javax.swing.JApplet;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.event.MouseListener;
import java.awt.event.MouseEvent;

/* Applet che esemplifica la gestione
 * di eventi del mouse. */
public class AppletColorato extends JApplet
        implements MouseListener {

    /* indica se il puntatore del mouse si trova
     * all'interno dell'applet */
    private boolean mouseInterno;

    ... segue ...
```

- la classe estende **JApplet** e implementa **MouseListener**
- la variabile d'istanza **mouseInterno** viene gestita dai vari metodi per indicare se il puntatore del mouse si trova all'interno o all'esterno dell'applet

Applet che cambia colore

```
/* Inizializzazione dell'applet. */
public void init() {
    /* ipotizzo che il puntatore del mouse sia esterno
     * all'applet */
    this.mouseInterno = false;
    /* registra l'applet stesso come
     * ascoltatore degli eventi del mouse */
    this.addMouseListener(this);
}

... segue ...
```

- l'invocazione **this.addMouseListener(this)** specifica che
  - il componente **this** (l'applet)
  - si registra presso l'ascoltatore **this** (il parametro nell'invocazione, che indica questo oggetto che implementa **MouseListener**)
  - affinché vengano gestiti gli eventi causati dal mouse nel suo interno

Applet che cambia colore

```
/* Disegno dell'applet. */
public void paint(Graphics g) {
    /* se il puntatore del mouse si trova all'interno
     * dell'applet, disegna un rettangolo rosso,
     * altrimenti un rettangolo verde */
    Color c;

    if (this.mouseInterno)
        c = Color.red;
    else
        c = Color.green;
    g.setColor(c);
    g.fillRect(0, 0, this.getWidth(), this.getHeight());
}

... segue ...
```

- il metodo **paint(…)** disegna un rettangolo il cui colore viene determinato sulla base della posizione del mouse (rappresentata dalla variabile **mouseInterno**)

Applet che cambia colore

```
/* Gestione degli eventi del mouse. */

/* Il mouse è entrato nell'area dell'applet. */
public void mouseEntered(MouseEvent e) {
    this.mouseInterno = true;
    this.repaint();
}
/* Il mouse è uscito dall'area dell'applet. */
public void mouseExited(MouseEvent e) {
    this.mouseInterno = false;
    this.repaint();
}

... segue ...
```

- in corrispondenza agli eventi per l'ingresso e l'uscita del mouse nell'applet
  - viene cambiato il valore della variabile **mouseInterno**
  - viene ridisegnato l'applet — **repaint()** invoca **paint(...)** in modo indiretto

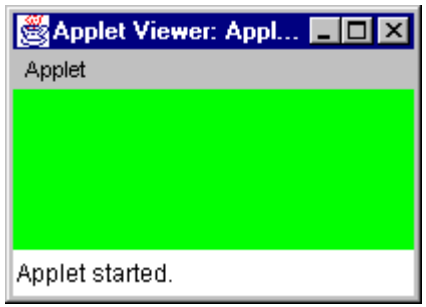
Applet che cambia colore

```
    /* Il mouse è stato premuto. */
    public void mousePressed(MouseEvent e) { }
    /* Il mouse è stato rilasciato. */
    public void mouseReleased(MouseEvent e) { }
    /* Il mouse è stato cliccato. */
    public void mouseClicked(MouseEvent e) { }
}
```

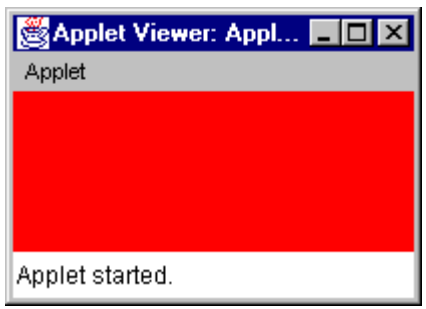
- in corrispondenza degli altri eventi del mouse non deve essere fatto nulla

Applet che cambia colore

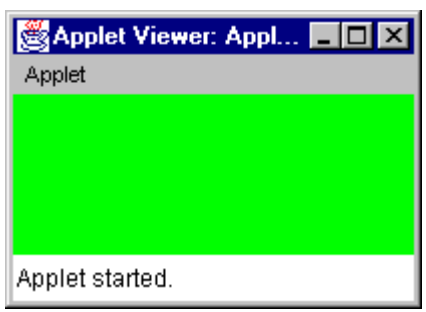
Il puntatore del mouse si trova all'esterno dell'applet



- il puntatore del mouse viene posizionato all'interno dell'applet



- il puntatore del mouse torna all'esterno dell'applet



Gestione di eventi mediante adattatori

Nell'esempio mostrato in precedenza, la sorgente degli eventi (l'applet) e l'ascoltatore degli eventi sono lo stesso oggetto

- la classe che è stata definita ha le seguenti caratteristiche
  - estende la classe **JApplet** (prototipo della sorgente degli eventi)
  - implementa l'interfaccia **MouseListener** (che descrive le operazioni che un ascoltatore di eventi del mouse deve saper eseguire)

Viene ora mostrato una diversa implementazione per lo stesso applet, in cui l'ascoltatore degli eventi è realizzato mediante un oggetto diverso dalla sorgente degli eventi

- è possibile implementare l'ascoltatore definendo una classe (diversa da quella che implementa l'applet) che implementa l'interfaccia **MouseListener**
- viene invece mostrata una implementazione più semplice, basata sull'uso di un adattatore

Adattatori

Per **adattatore** di una interfaccia **I** si intende una classe A_I con le seguenti caratteristiche

- la classe A_I implementa l'interfaccia **I**
- nella classe A_I, l'implementazione dei metodi dichiarati dall'interfaccia **I** è la più banale possibile
  - in particolare, i metodi vengono solitamente implementati con il corpo vuoto

Avendo a disposizione un adattatore A_I per una interfaccia **I**, è possibile implementare l'interfaccia **I** definendo una classe che estende A_I

- questa soluzione è vantaggiosa se si vuole implementare l'interfaccia **I** in modo tale che solo alcuni dei suoi metodi vengono definiti in modo non banale
  - nell'esempio precedente, solo due dei cinque metodi di **MouseListener** sono stati implementati in modo non banale

Gestione di eventi e adattatori

Le API di Java contengono, oltre alla definizione delle interfacce per la gestione di eventi, anche la definizione dei corrispondenti adattatori

- ad esempio, la classe **MouseInputAdapter** (del package **javax.swing.event**) è un adattatore per le interfacce **MouseListener** e **MouseMotionListener**

Lo svantaggio degli adattatori è che, essendo classi, vanno estese (anziché implementate)

- di conseguenza, non è possibile definire una classe che, contemporaneamente, estende un applet e un adattatore
- in pratica, per realizzare un ascoltatore mediante un adattatore, è necessario definire una nuova classe
- tuttavia, è possibile realizzare tale obiettivo in modo semplice utilizzando una forma sintattica che consente di definire una "classe interna anonima" che implementa una interfaccia o estende un'altra classe e, contestualmente, di istanziare un oggetto da tale classe interna anonima

Applet che cambia colore — con adattatore

```
import javax.swing.JApplet;
import java.awt.Color;
import java.awt.Graphics;
import javax.swing.event.MouseInputAdapter;
import java.awt.event.MouseEvent;

/* Applet che esemplifica la gestione di eventi del mouse,
 * e degli adattatori. */
public class AppletColoratoConAdattatore extends JApplet {
    /* indica se il puntatore del mouse si trova
     * all'interno dell'applet */
    private boolean mouseInterno;

    ... inizializzazione dell'applet ...

    /* Disegno dell'applet. */
    public void paint(Graphics g) {
        ... come nella classe AppletColorato ...
    }
}
```

Applet che cambia colore — con adattatore

```
/* Inizializzazione dell'applet. */
public void init() {
    MouseInputAdapter aem;     // ascoltatore per gli
                               // eventi del mouse

    /* ipotesi: puntatore esterno all'applet */
    this.mouseInterno = false;
    /* crea un ascoltatore degli eventi del mouse */
    aem = new MouseInputAdapter() {
          /* Il mouse è entrato nell'applet. */
          public void mouseEntered(MouseEvent e) {
              AppletConAdattatore.this.mouseInterno = true;
              AppletConAdattatore.this.repaint();
          }
          /* Il mouse è uscito dall'applet. */
          public void mouseExited(MouseEvent e) {
              AppletConAdattatore.this.mouseInterno = false;
              AppletConAdattatore.this.repaint();
          }
      };
    /* registra gli eventi dell'applet */
    this.addMouseListener(aem);
}
```

Creazione di oggetti da classi interne anonime

L'espressione

```
new MouseInputAdapter() {
    /* Il mouse è entrato nell'applet. */
    public void mouseEntered(MouseEvent e) {
        AppletConAdattatore.this.mouseInterno = true;
        AppletConAdattatore.this.repaint();
    }
    /* Il mouse è uscito dall'applet. */
    public void mouseExited(MouseEvent e) {
        AppletConAdattatore.this.mouseInterno = false;
        AppletConAdattatore.this.repaint();
    }
}
```

- ha lo scopo di istanziare un oggetto da una nuova classe (a cui non è assegnato un nome, e perciò è chiamata anonima) che estende la classe **MouseInputAdapter**, ridefinendone i metodi **mouseEntered(...)** e **mouseExited(...)**

Creazione di oggetti da classi interne anonime

- la classe definita è chiamata interna, perché è stata definita nell'ambito di un'altra classe (quella che definisce l'applet)
- gli oggetti istanziati da una classe interna hanno la possibilità di accedere a variabili e metodi dell'oggetto (esterno) che li ha creati
  - un riferimento all'oggetto **AppletConAdattatore** che ha creato l'adattatore è l'espressione **AppletConAdattatore.this**
- quindi, l'istruzione

```
AppletConAdattatore.this.repaint();
```

  - ha lo scopo di richiedere all'applet (che ha creato l'adattatore) di ridisegnarsi

Componenti, eventi e ascoltatori

La gestione delle eventi nelle interfacce grafiche è basata su eventi, ascoltatori e sorgenti di eventi

- in una interfaccia grafica vengono normalmente utilizzati
  - uno o più componenti
  - uno o più ascoltatori di eventi
- gli eventi sono rappresentati da oggetti istanza della classe **EventObject** e delle sue sotto-classi
  - ciascuna sotto-classe di **EventObject** rappresenta una particolare tipologia di eventi
- gli ascoltatori sono oggetti di classi che estendono l'interfaccia **EventListener** e le interfacce da essa derivate
  - ciascuna estensione di **EventListener** rappresenta un ascoltatore in grado di gestire una particolare tipologia di eventi
- ciascun componente che è sorgente di eventi significativi deve registrarsi presso opportuni ascoltatori

Tipologie di eventi

Ciascun componente può generare diverse tipologie di eventi

- ad esempio, un bottone può generare
  - un evento di tipo “azione” (ovvero, un evento di tipo **ActionEvent**, del package **java.awt.event**), se viene premuto (mediante l’uso del mouse)
  - un evento di tipo “mouse” (ovvero, un evento di tipo **MouseEvent**, del package **java.awt.event**), se il puntatore del mouse viene posizionato nell’area del bottone

Ciascun ascoltatore può gestire diverse tipologie di eventi

- ad esempio, un ascoltatore può ascoltare e gestire
  - eventi di tipo **ActionEvent**, se implementa l’interfaccia **ActionListener**
  - eventi di tipo **MouseEvent**, se implementa l’interfaccia **MouseListener**
  - sia eventi di tipo **ActionEvent** che eventi di tipo **MouseEvent**, se implementa entrambe le interfacce

JButton, ActionEvent e ActionListener

Un bottone (un componente di tipo **JButton**), quando viene premuto, genera un evento di tipo **ActionEvent**

- un evento di tipo **ActionEvent** può essere gestito da un oggetto che implementa l'interfaccia **ActionListener**
  - l'interfaccia **ActionListener** contiene solo la dichiarazione del metodo **void actionPerformed(ActionEvent e)**

In pratica, per gestire l'evento "un bottone è stato premuto" si deve

- creare un oggetto ascoltatore che implementa l'interfaccia **ActionListener**
  - ad esempio, creando un oggetto da una classe interna anonima che implementa l'interfaccia **ActionListener**
- registrare il bottone presso tale oggetto ascoltatore
- tali attività possono essere svolte nell'ambito del metodo per l'inizializzazione dell'interfaccia grafica

Esempio — conta quante volte viene premuto un bottone

Come esempio, viene mostrata la realizzazione di una applicazione che conta quante volte viene premuto un bottone

- l'interfaccia grafica dell'applicazione contiene
  - un bottone
  - una etichetta, che indica quante volte il bottone è stato premuto
- l'applicazione **Contatore** viene definita come completamento dell'applicazione tipo descritta nel capitolo precedente

Il metodo inizializzaGUI()

Il metodo **inizializzaGUI()** avrà la seguente struttura

```
/* Inizializza l'interfaccia dell'applicazione. */
private void inizializzaGUI() {
    /* pannello, usato come contenitore intermedio */
    JPanel jPanel;

    /* crea il pannello */
    jPanel = new JPanel();

    ... crea i componenti dell'interfaccia ...

    ... crea gli ascoltatori di eventi ...

    ... registra le sorgenti di eventi ...

    ... aggiunge i componenti al pannello ...

    /* aggiunge il pannello alla frame */
    this.getContentPane().add(jPanel);
}
```

Conta quante volte viene premuto un bottone

Variabili d'istanza per il bottone e l'etichetta

```
/* un bottone */
private JButton jButton;
/* etichetta per visualizzare quante volte è
 * stato premuto jButton */
private JLabel jLabelContatore;
```

Variabili d'istanza per contatore quante volte è stato premuto il bottone

```
/* conta quante volte è stato premuto jButton */
private int contaClick;
```

Inizializzazione di **contaClick** (nel costruttore)

```
/* finora il bottone non è mai stato premuto */
this.contaClick = 0;
```

Conta quante volte viene premuto un bottone

Crea i componenti dell'interfaccia grafica

```
/* crea il bottone */
jButton = new JButton("Premimi!");

/* crea l'etichetta per visualizzare quante volte è
 * stato premuto jButton */
jLabelContatore = new JLabel();
jLabelContatore.setText("Il bottone è stato premuto " +
                        this.contaClick + " volte");
```

Conta quante volte viene premuto un bottone

Crea l'ascoltatore per gli eventi di azione

- bisogna creare un oggetto che implementa l'interfaccia **ActionListener** del package **java.awt.event**

```
/* ascoltatore per eventi di azione */
ActionListener asc;

/* crea un ascoltatore degli eventi di azione */
asc = new ActionListener() {
          /* Si è verificato un evento di azione. */
          public void actionPerformed(ActionEvent e) {
              /* l'unico evento di azione possibile
               * è la pressione del bottone jButton */
              Contatore.this.conteggiaClick();
          }
      };
```

Conta quante volte viene premuto un bottone

Registra le sorgenti di eventi

- una sorgente di eventi, per registrarsi presso un ascoltatore di eventi di tipo **ActionEvent**, deve usare il metodo **addActionListener(...)**
  - ciascuna diversa tipologia di eventi generata da una sorgente di eventi va registrata usando un differente metodo
  - una sorgente di eventi che genera eventi di più tipologie può registrarsi presso un diverso ascoltatore per ciascuna tipologia di eventi che genera

```
/* registra gli eventi di jButton */
jButton.addActionListener(asc);
```

Di norma, la registrazione degli eventi generati da un componente deve precedere l'inserimento del componente nel contenitore in cui va collocato

Conta quante volte viene premuto un bottone

Aggiunge i componenti al pannello

```
/* aggiunge i componenti al pannello */
jPanel.add(jButton);
jPanel.add(jLabelContatore);
```

La classe **Contatore** deve inoltre contenere la definizione del metodo **conteggiaClick()**, invocato quando **jButton** viene premuto

```
/* Il bottone jButton è stato premuto. */
private void conteggiaClick() {
    /* incrementa contaClick */
    this.contaClick++;
    /* aggiorna il valore visualizzato da
     * jLabelContatore */
    jLabelContatore.setText(
        "Il bottone è stato premuto " +
        this.contaClick + " volte");
}
```

Esecuzione di Contatore

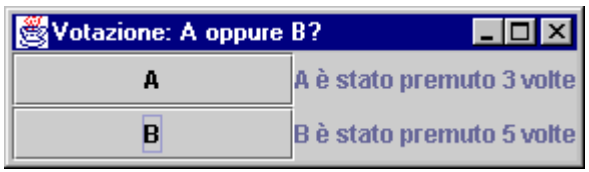
Inizialmente il bottone non è stato premuto

- il bottone viene premuto per la prima volta

- il bottone viene premuto per la seconda volta

Esempio — votazione

Come ulteriore esempio, vengono mostrate alcune possibili realizzazioni per una applicazione che conta quante volte ciascuno dei suoi due bottoni viene premuto



- in questo caso sono presenti due sorgenti di eventi
- intuitivamente, per questa applicazione sono possibili due diverse realizzazioni
  - una basata su due ascoltatori, un ascoltatore per ciascuna sorgente di eventi
  - una basata su un singolo ascoltatore, che gestisce gli eventi generati da entrambe le sorgenti di eventi

Soluzione basata su due ascoltatori

Viene creato un ascoltatore per gli eventi di azione del bottone **bottoneA** e uno (simile, non mostrato) per gli eventi di **bottoneB**

```
/* ascoltatore per eventi di azione */
ActionListener ascA;    // ascoltatore per bottoneA
... similmente per l'ascoltatore di bottoneB ...

/* crea l'ascoltatore per gli eventi di bottoneA */
ascA = new ActionListener() {
          /* Si è verificato un evento di azione. */
          public void actionPerformed(ActionEvent e) {
              /* l'unico evento di azione possibile è:
               * è stato premuto bottoneA */
              VotazioneDue.this.contaA();
          }
      };
 /* registra gli eventi del bottone A */
 bottoneA.addActionListener(ascA);

 ... similmente per l'ascoltatore di bottoneB ...
```

Soluzione basata su un ascoltatore

Viene creato un solo ascoltatore, per gli eventi di azione di **bottoneA** e di **bottoneB**

```
/* ascoltatore per eventi di azione */
ActionListener asc;

/* crea un ascoltatore degli eventi di azione */
asc = new ActionListener() {
          /* Si è verificato un evento di azione. */
          public void actionPerformed(ActionEvent e) {
              ... da definire ...
          }
      };

/* registra gli eventi del bottone A */
bottoneA.addActionListener(asc);
/* registra gli eventi del bottone B */
bottoneB.addActionListener(asc);
```

Soluzione basata su un ascoltatore

Nella soluzione con un solo ascoltatore per più sorgenti di eventi, il metodo **actionPerformed(...)** deve determinare l'evento che si è effettivamente verificato e che ne ha causato l'esecuzione

- in questo caso è sufficiente determinare la sorgente dell'evento (perché è possibile un solo evento per ciascuna sorgente)
  - è possibile determinare la sorgente di un evento **e** mediante l'espressione **e.getSource()**
- in altri casi, è possibile determinare l'evento effettivo esaminando altre proprietà dell'evento passato come argomento nell'invocazione del metodo di gestione dell'evento
  - ad esempio, a partire da un evento **e** generato dal mouse è possibile determinare la posizione del mouse al momento dell'evento mediante le espressioni **e.getX()** ed **e.getY()**

Soluzione basata su un ascoltatore

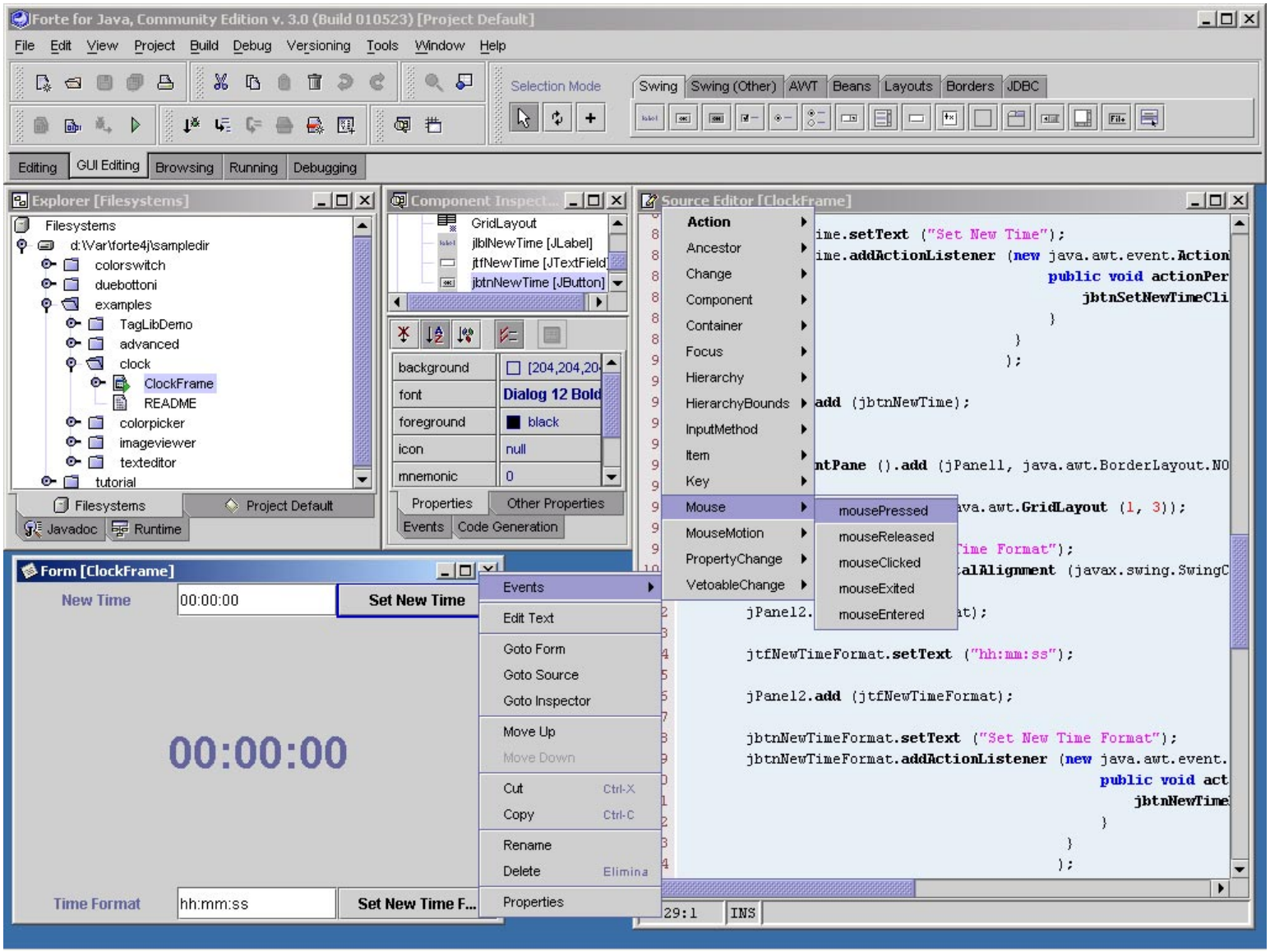
```
/* Si è verificato un evento di azione. */
public void actionPerformed(ActionEvent e) {
    Object sorgente;    // sorgente dell'evento e

    /* gli eventi possibili sono:
     * (1) è stato premuto bottoneA,
     * (2) è stato premuto bottoneB.
     * L'evento viene determinato sulla base della
     * sua sorgente */
    sorgente = e.getSource();
    if (sorgente==Votazione.this.bottoneA)
        Votazione.this.contaA();
    else if (sorgente==Votazione.this.bottoneB)
        Votazione.this.contaB();
}
```

Gestione di eventi e IDE

La realizzazione di una interfaccia grafica richiede normalmente la gestione degli eventi significativi generati da ciascun componente dell'interfaccia grafica

- il programmatore dovrebbe ripetere, per ciascun componente dell'interfaccia grafica e per ciascuna tipologia significativa di eventi generati dal componente
  - la creazione di un ascoltatore per quella tipologia di eventi per quel componente, con la registrazione del componente presso quell'ascoltatore
  - la definizioni di metodi che implementano le operazioni da eseguire in corrispondenza agli eventi significativi
- in questo caso, la prima attività (creazione dell'ascoltatore e registrazione del componente) può essere gestita in modo completamente standard, mentre la seconda attività (scrittura dei metodi) richiede la scrittura di metodi ad hoc
  - i GUI editor (strumenti per la realizzazione di interfacce grafiche) supportano anche la gestione degli eventi

Gestione degli eventi con un GUI editor — Forte for Java



Gestione degli eventi in un GUI editor

Per ciascun componente dell'interfaccia grafica, è possibile

- selezionare le tipologie di eventi significative
  - per ciascun evento significativo, il GUI editor genererà automaticamente il codice per la creazione dell'ascoltatore per quell'evento e la registrazione del componente presso l'ascoltatore
  - inoltre, il GUI editor genererà un metodo (con il corpo inizialmente vuoto) dedicato all'implementazione delle operazioni da eseguire in corrispondenza di tale evento, che dovrà essere completato dal programmatore