Programmazione Orientata agli Oggetti in Linguaggio Java

Programmazione Grafica: Eventi

versione 1.0

G. Mecca – Università della Basilicata – mecca@unibas.it

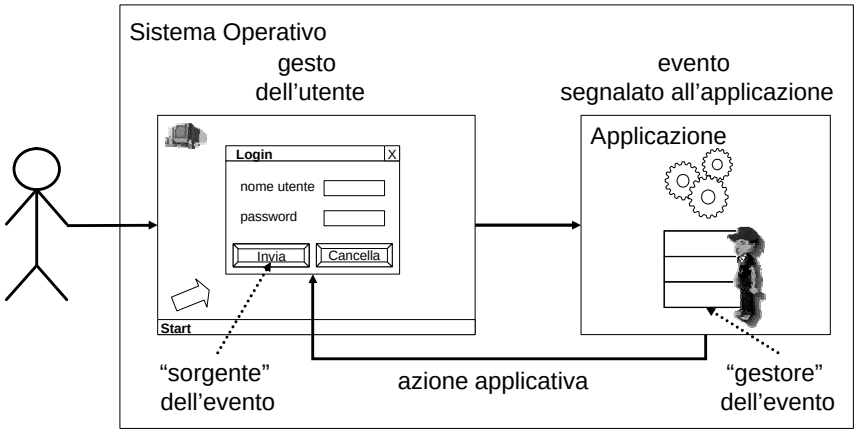
Sommario

- Introduzione
- Gestione degli Eventi in Java
  - ⇨Tipi di Eventi
  - ⇨Publish and Subscribe
  - ⇨Publish and Subscribe in AWT/Swing
- Programmazione Asincrona

Introduzione

- Flusso di esecuzione nell'applicazione
  - ⇨l'utente esegue dei gesti interagendo con l'interfaccia
  - ⇨i gesti vengono catturati dal Window manager (e parzialmente gestiti per ottenere i necessari effetti grafici)
  - ⇨il Window manager segnala gli eventi all'applicazione
  - ⇨l'applicazione li gestisce eseguendo azioni

Introduzione



Introduzione

- Problemi da risolvere
  - ⇨stabilire i meccanismi di comunicazione tra il Window Manager e la macchina virtuale Java (come segnalare l'evento alla m.v.)
  - ⇨stabilire il modo per associare nella macchina virtuale il gestore all'evento (a chi inoltrare l'evento)
  - ⇨stabilire come eseguire l'azione applicativa (come deve reagire il gestore all'evento)

Introduzione

- Nei toolkit tradizionali (es: C)
  - ⇨la comunicazione viene gestita attraverso una coda (coda degli eventi – "event queue")
  - ⇨il Window manager inserisce per ciascun evento una struttura in una coda
  - ⇨è compito del programmatore analizzare periodicamente la coda, estrarre i nuovi eventi, analizzarne il tipo, e decidere l'azione

Introduzione

- Caratteristiche di questo approccio
  - ⇨la sorgente e il gestore sono del tutto disaccoppiati (qualunque modulo può gestire gli eventi generati da una sorgente)
  - ⇨molto flessibile
  - ⇨ma molto difficile da programmare
  - ⇨gestione centralizzata degli eventi (lunghi if molto nidificati per riconoscere l'evento)
  - ⇨i gestori non sono tipati

Introduzione

- In Visual Basic
  - ⇨ogni componente grafico è in grado di reagire ad una serie di eventi fissati
  - ⇨che gestisce lui stesso, eseguendo metodi fissati
  - ⇨es: supponendo di avere introdotto un bottone bottoneHelp, a seguito di un click sul bottone viene eseguito il metodo bottoneHelp_Click())

Introduzione

- Caratteristiche di questo approccio
  - ⇨molto semplice da programmare
  - ⇨ma molto rigido
  - ⇨la sorgente e il gestore devono necessariamente coincidere
  - ⇨non è possibile inoltrare un evento a più gestori
  - ⇨la logica applicativa (del gestore) è eccessivamente accoppiata all'interfaccia

Gestione degli Eventi in Java

- In Java
  - ⇨una infrastruttura nuova per la gestione degli eventi
- Caratteristiche
  - ⇨completamente orientata agli oggetti
  - ⇨approccio fortemente tipato
  - ⇨meccanismo di tipo editore/sottoscrittore ("publish and subscribe")

Gestione degli Eventi in Java

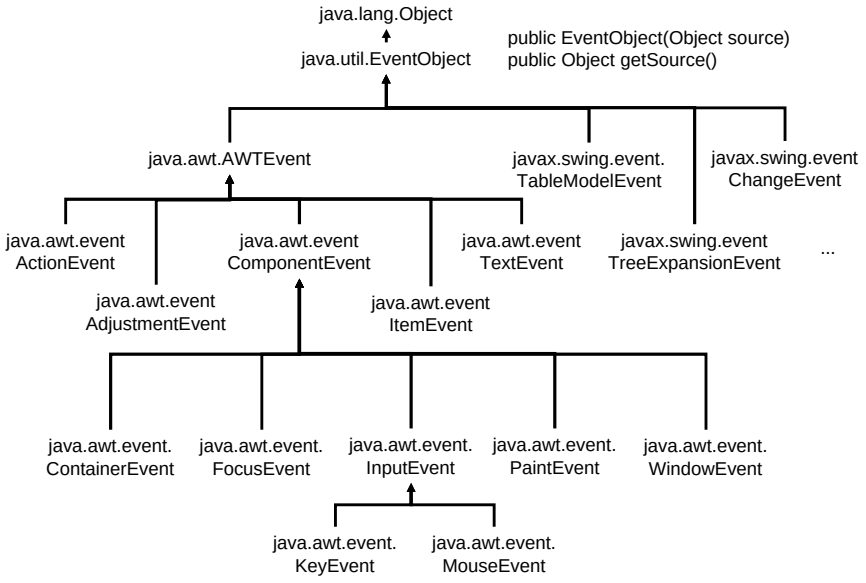
- Completamente orientata agli oggetti
  - una sorgente è un oggetto (scontato: componente grafico)
  - un evento è un oggetto
  - un gestore è un oggetto
- Approccio fortemente tipato
  - gli eventi sono tipati
  - le sorgenti sono tipate
  - i gestori sono tipati

Tipi di Eventi

- Evento
  - oggetto utilizzato per segnalare un gesto dell'utente sull'interfaccia
  - ogni gesto sull'interfaccia inserisce un oggetto evento nella coda degli eventi
- Coda degli eventi ("Event Queue")
  - coda di riferimenti agli eventi verificati
  - java.awt.event.EventQueue

Tipi di Eventi

- Gli eventi sono tipati
  - ⇨gesti diversi dell'utente vengono rappresentati con oggetti di classi diverse
  - ⇨che estendono java.util.EventObject
- La classe java.util.EventObject
  - ⇨il metodo fondamentale è Objec getSource() che restituisce un riferimento all'oggetto sorgente dell'evento



Tipi di Eventi

- Nota
  - ⇨non tutti gli eventi sono uguali
- Classificazione di awt
  - ⇨eventi semantici: eventi rilevanti per la logica applicativa dell'applicazione
  - ⇨eventi di basso livello: eventi che vengono principalmente gestiti dal framework grafico (Window manager + AWT + Swing)

Tipi di Eventi

ATTENZIONE
agli eventi di tipo
ActionEvent

- java.awt.event.ActionEvent
  - ⇨l'evento semantico per eccellenza
  - ⇨l'utente ha schiacciato un bottone
  - ⇨o selezionato una voce in un menu
  - ⇨o digitato invio in un campo di testo
  - ⇨o selezionato un pulsante checkbox o radio
  - ⇨a questo deve sempre corrispondere una azione applicativa
  - ⇨non viene gestito dal framework grafico

Tipi di Eventi

- Altri esempi di eventi semantici
  - ⇨java.awt.event.AdjustmentEvent
  - ⇨java.awt.event.ItemEvent
  - ⇨java.awt.event.TextEvent
- Gli altri eventi
  - ⇨sono considerati di basso livello
  - ⇨normalmente sono gestiti dal framework
  - ⇨ma in alcuni casi può essere necessario catturarli

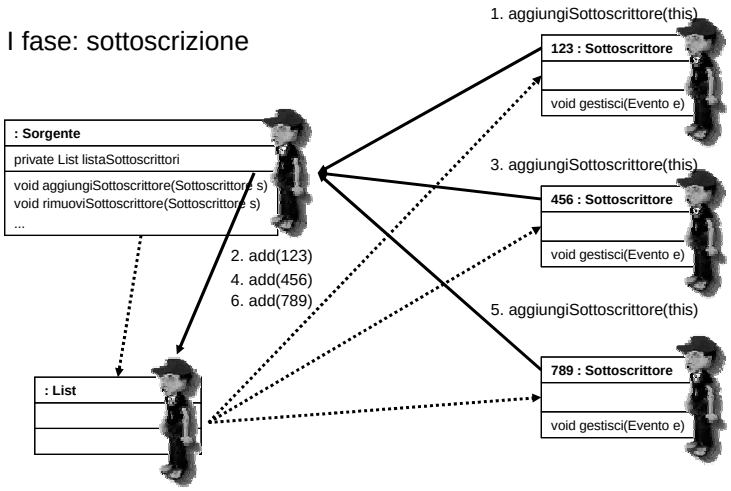
Tipi di Eventi

- Esempi di eventi di basso livello
  - ⇨java.awt.KeyEvent, java.awt.MouseEvent
  - ⇨vengono gestiti quasi esclusivamente dal framework grafico
  - ⇨contribuiscono a generare un ActionEvent (es: click di un bottone)
  - ⇨normalmente non corrispondono ad un'azione nell'applicazione
  - ⇨ma se voglio programmare un videogioco...

Publish and Subscribe

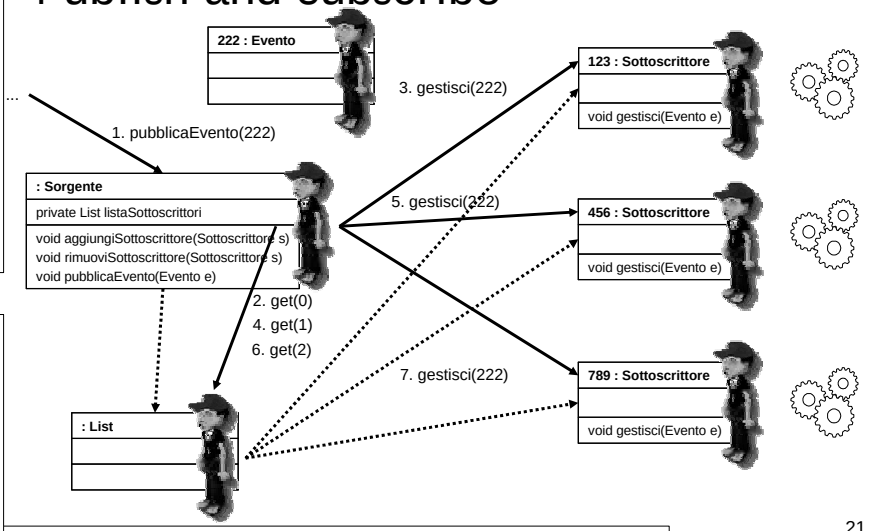
- Publish and Subscribe
  - ⇨meccanismo di sottoscrizione attraverso il quale il gestore si “iscrive” alla lista di gestori di eventi della sorgente
  - ⇨la sorgente mantiene liste di iscritti all’evento
  - ⇨in corrispondenza dell’evento notifica (“rende pubblico”) l’evento chiamando un metodo opportuno di tutti i sottoscrittori

Publish and Subscribe



Publish and Subscribe

II fase: pubblicazione

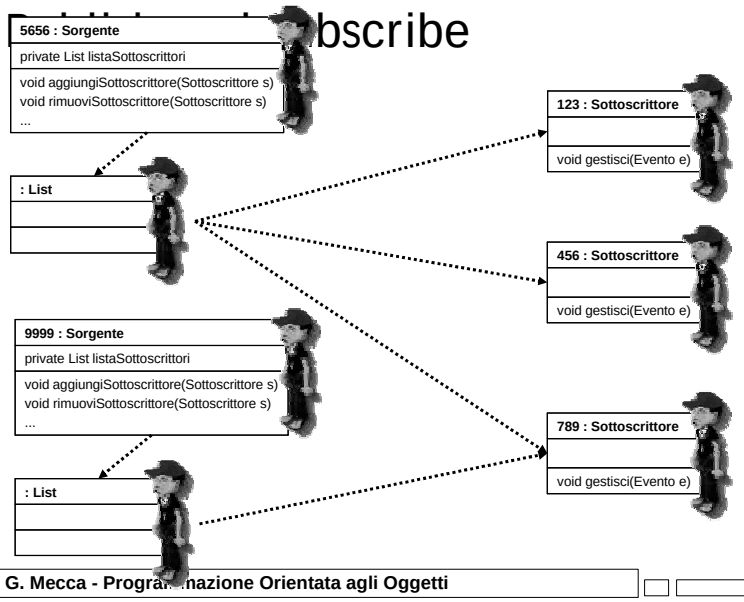


Publish and Subscribe

- Analogo
  - ⇨il sottoscrittore è il cliente di un albergo
  - ⇨la sorgente è il portiere dell'albergo
  - ⇨il cliente chiede la sveglia facendo una chiamata la sera prima ("sottoscrive" la sveglia)
  - ⇨il portiere la mattina dopo chiama tutti i clienti che hanno richiesto la sveglia ("pubblica" la sveglia)

Publish and Subscribe

- Vantaggio principale di questo approccio
  - ⇨rispetto agli approcci tradizionali è un buon compromesso tra flessibilità e semplicità
  - ⇨il codice di gestione dell'evento è separato rispetto al codice della sorgente (minore accoppiamento)
  - ⇨una stessa sorgente può avere molti gestori
  - ⇨uno stesso gestore può sottoscrivere gli eventi di varie sorgenti



Publish and Subscribe

- Terminologia
  - ⇨il metodo che viene chiamato per gestire l'evento viene chiamato metodo di "callback"
  - ⇨consente alla sorgente di "richiamare" il gestore per chiedergli di gestire l'evento
  - ⇨il gestore può essere un oggetto qualunque purchè implementi un'interfaccia opportuna
  - ⇨che prevede il metodo di callback

Publish and Subscribe

- Terminologia
  - ⇨un gestore di eventi viene anche detto ascoltatore o sensore ("listener")
  - ⇨per segnalare il fatto che resta in ascolto delle segnalazioni di evento da parte della sorgente
  - ⇨più correttamente, il gestore viene "risvegliato" dalla sorgente per gestire l'evento

Publish and Subscribe

- Terminologia
  - ⇨pubblicare l'evento viene spesso detto come "sparare" l'evento ("fire event")
- Chi chiama il metodo pubblicaEvento()
  - ⇨un componente esterno (es: il framework grafico)
  - ⇨oppure la sorgente stessa nel caso in cui eseguendo uno dei suoi metodi si accorga che è necessario pubblicare un evento

Publish and Subscribe

ATTENZIONE
il meccanismo è un meccanismo generale per gestire eventi

- Nota
  - ⇨si tratta di un meccanismo del tutto generale
  - ⇨indipendente, in linea di principio, dagli eventi grafici
  - ⇨consente di gestire altri tipi di eventi
- Esempi
  - ⇨eventi temporizzati
  - ⇨javax.swing.Timer
  - ⇨sorgente di eventi a cadenza fissata

Publish and Subscribe in AWT/Swing

- La gestione degli eventi in AWT
  - ⇨utilizza esattamente questo meccanismo
- Caratteristiche
  - ⇨le sorgenti di evento sono i componenti dell'interfaccia grafica
  - ⇨gli eventi sono oggetti di tipo appropriato che rappresentano i gesti dell'utente
  - ⇨le sorgenti sono tipate
  - ⇨i ricevitori sono tipati

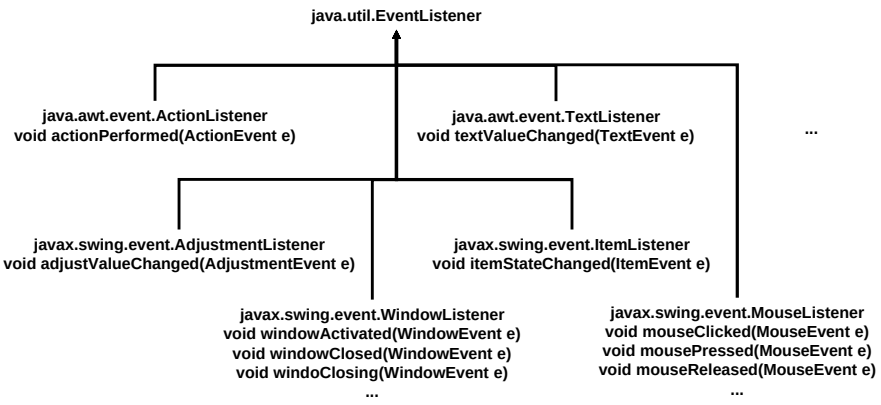
Publish and Subscribe in AWT/Swing

- Le sorgenti sono tipate
  - ⇨ciascuna sorgente può generare solo alcuni eventi (es: bottone – ActionEvent, non AdjustmentEvent)
  - ⇨mantiene una lista di sottoscrittori per ciascun evento che può generare
  - ⇨i sottoscrittori sono avvisati solo per gli eventi che li riguardano, e non per tutti gli eventi

Publish and Subscribe in AWT/Swing

- I ricevitori sono tipati
  - ⇨per ogni evento esiste un tipo di gestore corrispondente
  - ⇨per iscriversi alla notifica di un evento, un gestore deve avere il tipo opportuno
  - ⇨ovvero deve implementare una interfaccia
  - ⇨es: ActionEvent – ActionListener, AdjustmentEvent – AdjustmentListener

Publish and Subscribe



Publish and Subscribe in AWT/Swing

- I metodi di sottoscrizione
  - ciascuna sorgente ha un metodo per aggiungere sottoscrittori di tipo opportuno
- Esempio: la classe JButton
  - addActionListener(ActionListener l)
  - removeActionListener(ActionListener l)
- Esempio: la classe JFrame
  - addWindowListener(WindowListener l)
  - removeWindowListener(WindowListener l)

Publish and Subscribe in AWT/Swing

- Esempio
  - convertitore di voti
- La sorgente di eventi
  - il pulsante “Converti”
- L’evento
  - di tipo ActionEvent
- Il gestore
  - deve implementare ActionListener

Publish and Subscribe in AWT/Swing

- Un problema
  - ⇨la sorgente dell'evento è il bottone
  - ⇨ma per gestire la conversione è necessario acquisire il valore del campo di testo che contiene il voto in 30mi
  - ⇨poi modificare il valore dell'etichetta risultato
  - ⇨di conseguenza per gestire l'evento al gestore non è sufficiente il riferimento alla sorgente

Publish and Subscribe in AWT/Swing

- I soluzione
  - ⇨il gestore è implementato in una propria classe pubblica che implementa ActionListener
  - ⇨mantiene un riferimento al frame
  - ⇨il frame fornisce un metodo pubblico getValoreVoto() attraverso cui risalire al voto
  - ⇨e un metodo setRisultato(String) per cambiare il risultato

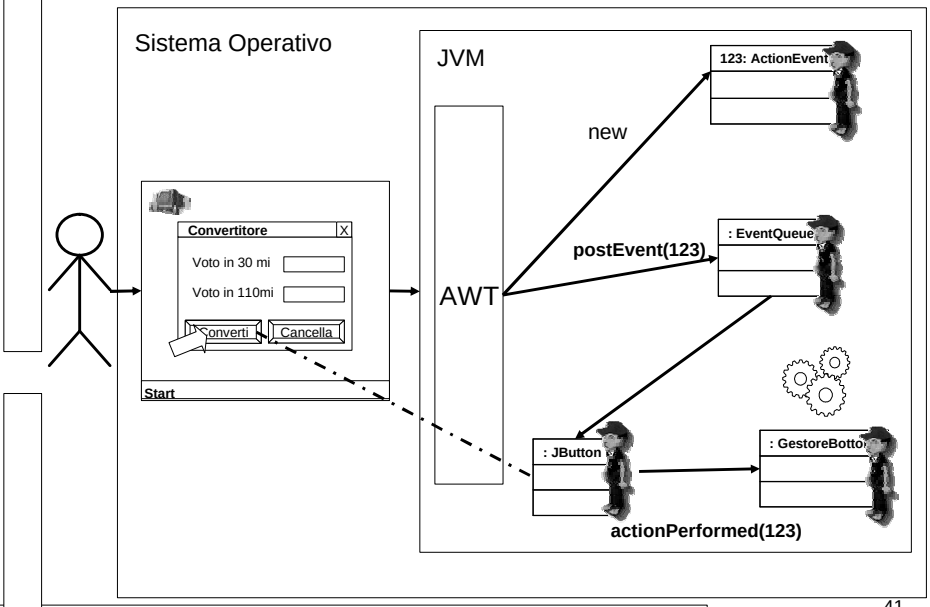
```
package it.unibas.convertitore;
public class Principale extends javax.swing.JFrame {

    public Principale() { inizializza(); }

    private javax.swing.JPanel pannello;
    private javax.swing.JButton bottoneConverti;
    private javax.swing.JLabel etichettaRisultato;
    private javax.swing.JTextField campoTestoVoto;

    private void inizializza() {
        this.pannello = new javax.swing.JPanel();
        this.campoTestoVoto = new javax.swing.JTextField();
        this.campoTestoVoto.setText("Immetti il valore in 30mi");
        this.bottoneConverti = new javax.swing.JButton();
        this.bottoneConverti.setText("Converti");
        this.etichettaRisultato = new javax.swing.JLabel();
        this.etichettaRisultato.setText("Risultato: ");
        this.bottoneConverti.addActionListener(new GestoreBottone(this));
        this.pannello.add(campoTestoVoto);
        this.pannello.add(bottoneConverti);
        this.pannello.add(etichettaRisultato);
        this.getContentPane().add(pannello);

        this.setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        this.setTitle("Convertitore");
        this.pack();
        this.setVisible(true);
    }
```


```
package it.unibas.convertitore;

public class Principale extends javax.swing.JFrame {

    ... // continua

    public String getValoreVoto() {
        return this.campoTestoVoto.getText();
    }

    public void setRisultato(String risultato) {
        this.etichettaRisultato.setText(risultato);
    }

    public void finestraErrore() {
        javax.swing.JOptionPane.showMessageDialog(this, "ERRORE: valore scorretto");
    }

    public static void main(String args[]) {
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                new Principale();
            }
        });
    }
}
```

```
package it.unibas.convertitore;

public class GestoreBottone implements java.awt.event.ActionListener {

    private Principale frame;

    public GestoreBottone(Principale frame) {
        this.frame = frame;
    }

    public void actionPerformed(java.awt.event.ActionEvent evt) {
        try {
            String stringa30mi = this.frame.getValoreVoto();
            if (stringa30mi == null) {
                this.frame.finestraErrore();
            } else {
                double voto30 = Double.parseDouble(stringa30mi);
                if (voto30 < 18 || voto30 > 30) {
                    this.frame.finestraErrore();
                } else {
                    double voto110 = voto30 / 30 * 110;
                    this.frame.setRisultato("Risultato: " + voto110);
                }
            }
        } catch (NumberFormatException e) {
            System.out.println("Errore: valore scorretto");
            this.frame.finestraErrore();
        }
    }
}
```


Publish and Subscribe in AWT/Swing

- La sottoscrizione
  - ⇨in questo caso viene effettuata dal frame
  - ⇨che crea il gestore
  - ⇨e lo aggiunge ai sottoscrittori della sorgente
  - ⇨GestoreBottone gestore = new GestoreBottone(this);
  - ⇨this.bottoneConverti.addActionListener(gestore);
  - ⇨da quel momento, ogni volta che il bottone pubblica un ActionEvent, verrà chiamato il metodo actionPerformed() di gestore



Publish and Subscribe in AWT/Swing

- Difetto di questa soluzione
  - ⇨costringe ad aggiungere metodi al frame per l'accesso ai componenti
  - ⇨costringe ad aggiungere il riferimento al frame al gestore
- II soluzione
  - ⇨la scrittura del codice si semplifica utilizzando una classe interna al frame
  - ⇨che ha accesso a tutte le proprietà private

```
package it.unibas.convertitore;
public class Principale extends javax.swing.JFrame {

   public Principale() { inizializza(); }

   private javax.swing.JPanel pannello;
   private javax.swing.JButton bottoneConverti;
   private javax.swing.JLabel etichettaRisultato;
   private javax.swing.JTextField campoTestoVoto;

   private void inizializza() {
      ...
      this.bottoneConverti.addActionListener(new GestoreBottone());
      ...
   }

   public void finestraErrore() {
      javax.swing.JOptionPane.showMessageDialog(this, "ERRORE: valore scorretto");
   }

  public static void main(String args[]) {
      javax.swing.SwingUtilities.invokeLater(new Runnable() {
         public void run() { new Principale(); }
      });
   }
```


```
   // continua Principale.java
   private class GestoreBottone implements java.awt.event.ActionListener {

      public void actionPerformed(java.awt.event.ActionEvent evt) {
         try {
            String stringa30mi = campoTestoVoto.getText();
            if (stringa30mi == null) {
               finestraErrore();
            } else {
               double voto30 = Double.parseDouble(stringa30mi);
               if (voto30 < 18 || voto30 > 30) {
                  finestraErrore();
               } else {
                  double voto110 = voto30 / 30 * 110;
                  etichettaRisultato.setText("Risultato: " + voto110);
               }
            }
         } catch (NumberFormatException e) {
            System.out.println("Errore: valore scorretto");
            finestraErrore();
         }
      }
} // fine Principale
```

Publish and Subscribe in AWT/Swing

- Una ulteriore modifica tipica
  - ⇨dal momento che (in questo caso) il gestore viene utilizzato esclusivamente in un punto del codice è possibile trasformare la classe interna in classe interna anonima
  - ⇨per tenere breve il codice della classe interna anonima, il codice viene estratto da actionPerformed() e spostato in un metodo a parte della classe Principale
  - ⇨il gestore fa solo da “intermediario”

```
package it.unibas.convertitore;
public class Principale extends javax.swing.JFrame {

   public Principale() { inizializza(); }

   private javax.swing.JPanel pannello;
   private javax.swing.JButton bottoneConverti;
   private javax.swing.JLabel etichettaRisultato;
   private javax.swing.JTextField campoTestoVoto;

   private void inizializza() {
      ...
      this.bottoneConverti.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent e) {
               converti(e);
            }
         }
      );
      ...
   }

   ...
```

```
// continua Principale.java
 private void converti(java.awt.event.ActionEvent evt) {
    try {
       String stringa30mi = campoTestoVoto.getText();
       if (stringa30mi == null) {
          finestraErrore();
       } else {
          double voto30 = Double.parseDouble(stringa30mi);
          if (voto30 < 18 || voto30 > 30) {
             finestraErrore();
          } else {
             double voto110 = voto30 / 30 * 110;
             etichettaRisultato.setText("Risultato: " + voto110);
          }
       }
    } catch (NumberFormatException e) {
       System.out.println("Errore: valore scorretto");
       finestraErrore();
    }
 }
} // fine Principale
```


Publish and Subscribe in AWT/Swing

- Lo stesso approccio
  - ⇨potrebbe essere seguito per l'altro evento significativo di questa applicazione
  - ⇨la chiusura della finestra
  - ⇨in alternativa alla chiamata del metodo setDefaultCloseOperation()

Publish and Subscribe in AWT/Swing

- In questo caso
  - ⇨sarebbe necessario creare un oggetto che implementi l'interfaccia WindowListener
  - ⇨in particolare definisca il metodo windowClosed() specificando System.exit(0);
  - ⇨aggiungerlo al frame tra i sottoscrittori con addWindowListener()

Programmazione Asincrona

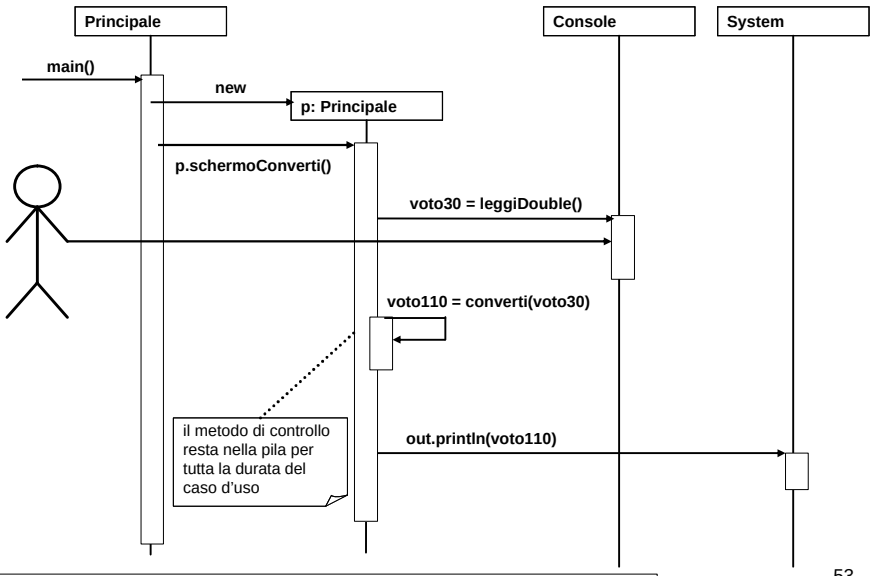
- Attenzione a quello che sta succedendo
  - ⇨questo meccanismo di programmazione è completamente diverso rispetto a quello utilizzato finora
- Finora
  - ⇨programmazione "sincrona"
- Da questo momento in poi
  - ⇨programmazione "asincrona"

Programmazione Asincrona

- Programmazione sincrona
  - ⇨stile tipico di programmazione con la console
  - ⇨il metodo A chiama il metodo B
  - ⇨A viene sospeso
  - ⇨il record di attivazione di B viene aggiunto alla pila
  - ⇨comincia l'esecuzione di B
  - ⇨al termine di B l'esecuzione di A riprende
  - ⇨A attende che l'esecuzione di B termini

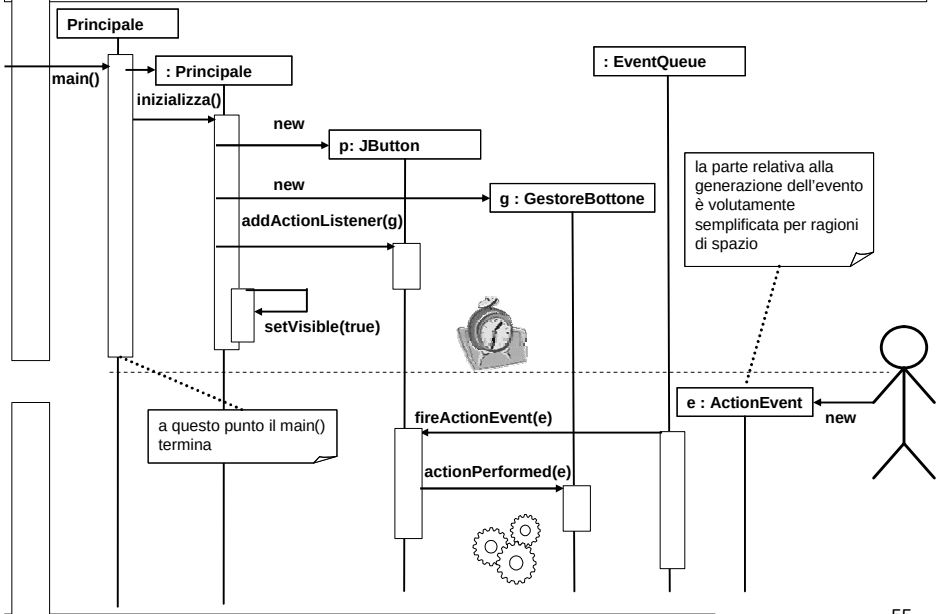
Programmazione Asincrona

- Esempio
  - ⇨conversioni con la console
  - ⇨un metodo di Principale (es: schermoConverti()) acquisisce il valore in 30mi chiamando Console.leggiDouble()
  - ⇨poi chiama il metodo di conversione
  - ⇨alla fine chiama System.out.println()
  - ⇨per tutta la durata dell'esecuzione, lo schermo mantiene il controllo del caso d'uso



Programmazione Asincrona

- Programmazione asincrona
  - ⇨un componente A chiama un metodo del componente B (es: per essere aggiunto ai sottoscrittori)
  - ⇨il metodo di A termina
  - ⇨successivamente si verifica un evento significativo (es: gesto dell'utente)
  - ⇨A viene invocato da B per gestire l'evento



Programmazione Asincrona

- La differenza
  - ⇨rispetto alla programmazione sincrona è notevole
  - ⇨il metodo actionPerformed() NON viene chiamato da un metodo che appartiene alla logica applicativa dell'applicazione
  - ⇨bensì dal framework grafico a seguito di azioni dell'utente

Programmazione Asincrona

ATTENZIONE
Cambiamento di stile di programmazione

- Lo stile di programmazione
  - ⇨il programmatore predispone l'infrastruttura applicativa
  - ⇨inizializzando i componenti grafici
  - ⇨scrivendo il codice della gestione degli eventi e registrando i gestori alle sorgenti
  - ⇨il codice di gestione deve essere scritto per essere eseguito a seguito di un evento che si manifesterà successivamente

Riassumendo

- Introduzione
- Gestione degli Eventi in Java
  - ⇨Tipi di Eventi
  - ⇨Publish and Subscribe
  - ⇨Publish and Subscribe in AWT/Swing
- Programmazione Asincrona

Termini della Licenza