

Università degli Studi di Modena e Reggio Emilia

Facoltà di Ingegneria – Reggio Emilia

CORSO DI

RETI DI CALCOLATORI

Linguaggio Java: La Grafica

Prof. Franco Zambonelli

**Lucidi realizzati in collaborazione con
Ing. Enrico Denti - Univ. Bologna**

JAVA E LA GRAFICA

L'architettura Java è *graphics-ready*

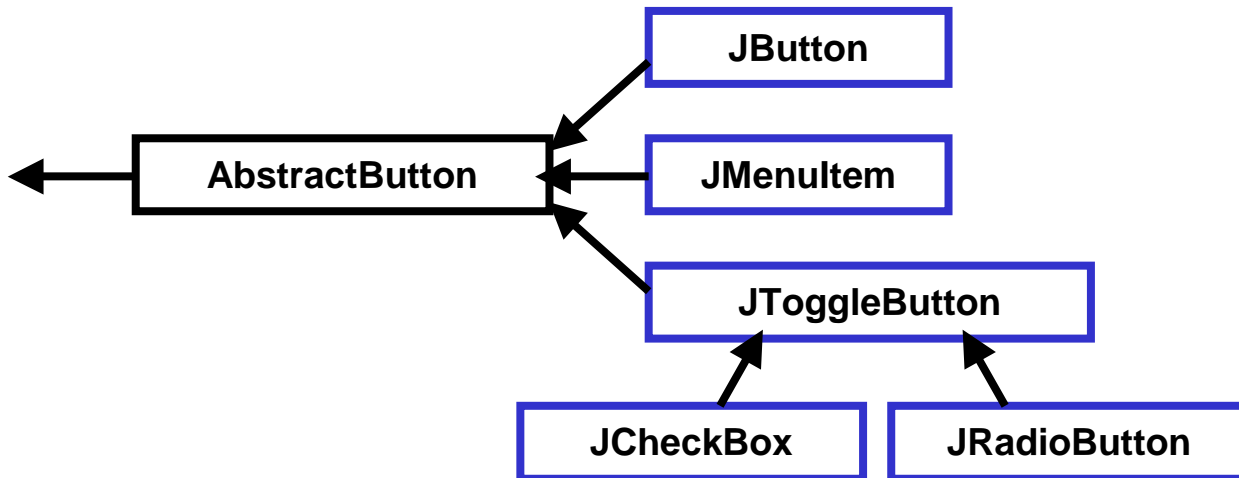
- Package `java.awt`
 - il primo package grafico (Java 1.0)
 - indipendente dalla piattaforma... o quasi!
- Package `javax.swing`
 - il nuovo package grafico (Java 2; versione preliminare da Java 1.1.6)
 - scritto esso stesso in Java, realmente indipendente dalla piattaforma

SWING: ARCHITETTURA

- Swing definisce una *gerarchia di classi* che forniscono ogni tipo di componente grafico
 - finestre, pannelli, frame, bottoni, aree di testo, checkbox, liste a discesa, etc etc
- **Programmazione “event-driven”:**
 - non più algoritmi stile input/elaborazione/output...
 - ... ma *reazione agli eventi* che l'utente, in modo interattivo, genera sui componenti grafici
- Concetti di evento e di ascoltatore degli eventi

Si può considerare un **paradigma di programmazione** a sé stante!!

SWING: GERARCHIA DI CLASSI



Container: tutti i componenti principali sono contenitori, destinati a contenere altri componenti

Window: le finestre sono casi particolari di contenitori e si distinguono in frame e finestre di dialogo

Jframe: componente finestra principale: ha un aspetto grafico, una cornice ridimensionabile e un titolo

Jcomponent: è il generico componente grafico

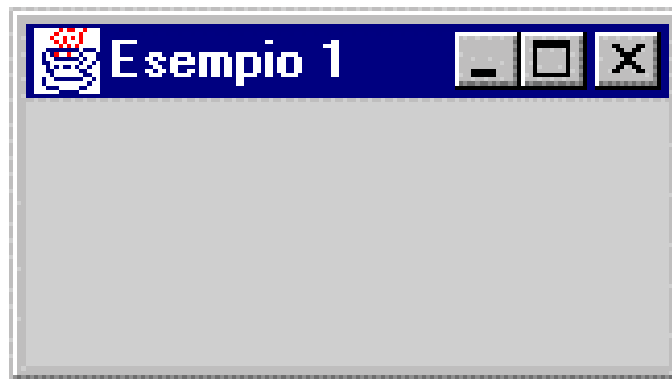
Jpanel: il pannello, un componente destinato a contenere altri componenti grafici per organizzarli

SWING: UN ESEMPIO

La più semplice applicazione grafica consiste in una classe il cui main crea un JFrame e lo rende visibile col metodo show():

```
import java.awt.*;
import javax.swing.*;
public class EsSwing1 {
    public static void main(String[] v){
        JFrame f = new JFrame("Esempio 1");
// crea un nuovo JFrame Inizialmente invisibile
// con titolo "Esempio 1"
        f.show();
// mostra il JFrame
    }
}
```

RISULTATO:



I comandi standard delle finestre sono già attivi

ATTENZIONE: la chiusura non distrugge il Frame ma lo nasconde soltanto. Per chiuderlo effettivamente ci vuole Ctrl+C

SWING: ESEMPIO 1

Con riferimento all'esempio precedente:

- La finestra che così nasce ha però *dimensioni nulle* (bisogna allargarla "a mano")
- Per impostare le dimensioni di un qualunque contenitore si usa `setSize()`, che ha come parametro un opportuno oggetto di classe `Dimension`:

```
f.setSize(new Dimension(300,150));  
// le misure x,y sono in pixel  
// tutto lo schermo: 800*600, 1024*768, etc.
```

- Inoltre, la finestra viene visualizzata *nell'angolo superiore sinistro* dello schermo
- Per impostare la posizione di un qualunque contenitore si usa `setLocation()`:

```
f.setLocation(200,100);  
// (0,0) = angolo superiore sinistro
```

- *Posizione e dimensioni* si possono anche fissare insieme, col metodo `setBounds()`

SWING: ESEMPIO MIGLIORATO

- Un esempio di finestra già dimensionata e collocata nel punto previsto dello schermo:

```
import java.awt.*;
import javax.swing.*;
public class EsSwing1 {
    public static void main(String[] v){
        JFrame f = new JFrame("Esempio 1");
        f.setBounds(200,100, 300,150)
        f.show();
    }
}
```

PERSONALIZZARE IL JFRAME

- **Un approccio efficace consiste nell'estendere `JFrame`, definendo una nuova classe:**

```
public class MyFrame extends JFrame {
    public MyFrame(){
        super(); setBounds(200,100,300,150);
    }
    public MyFrame(String titolo){
        super(titolo);
        setBounds(200,100, 300,150);
    }
}
```

ESEMPIO 2

Questo esempio usa un MyFrame:

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

public class EsSwing2 {
    public static void main(String[] v){
        MyFrame f = new MyFrame("Esempio 2");
        // posizione (200,100) dimensione (300,150)
        f.show();
    }
}
```


STRUTTURA DEL FRAME

- In Swing *non si possono aggiungere nuovi componenti direttamente al JFrame* Però...
- Dentro a ogni JFrame c'è un Container, recuperabile col metodo `getContentPane()`: è a lui che vanno aggiunti i nuovi componenti
- Tipicamente, si aggiunge un pannello (un JPanel o una nostra versione più specifica), tramite il metodo `add()`
 - sul pannello si può disegnare (forme, immagini...)
 - ...o aggiungere pulsanti, etichette, icone, (cioè aggiungere altri componenti!)

ESEMPIO 3

Aggiunta di un pannello al Container di un frame, tramite l'uso di `getContentPane()`:

```
import java.awt.*; import javax.swing.*;
public class EsSwing3 {
    public static void main(String[] v){
        MyFrame f = new MyFrame("Esempio 3");
        Container c = f.getContentPane();
        JPanel panel = new JPanel();
        c.add(panel);
        f.show();
    }
}
```

NOTA: non abbiamo disegnato niente, né aggiunto componenti, sul pannello! Però, avendo, il pannello, potremmo usarlo per disegnare e inserire altri componenti!

DISEGNARE SU UN PANNELLO

Per disegnare su un pannello occorre:

- definire una propria classe (MyPanel) che estenda il JPanel originale
- in tale classe, *ridefinire* `paintComponent()`, che è il metodo (ereditato da `JComponent`) che si occupa di disegnare il componente
 - ATTENZIONE: il nuovo `paintComponent()` da noi definito deve sempre richiamare il metodo `paintComponent()` originale, tramite `super`

Il nostro pannello personalizzato:

```
public class MyPanel extends JPanel {
    // nessun costruttore, va bene il default
    public void paintComponent(Graphics g){
        super.paintComponent(g);
        ...
    // qui aggiungeremo le nostre istruzioni di
    // disegno...
    // g è un oggetto gestito dal sistema a cui ci si
    // rivolge per disegnare
    }
}
```

Graphics g, di cui non ci dobbiamo occupare esplicitamente, è l'oggetto del sistema che effettivamente disegna ciò che ordiniamo

DISEGNARE SU UN PANNELLO

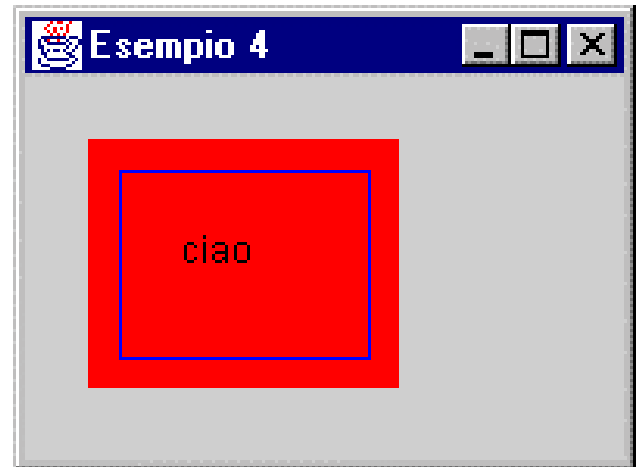
Quali metodi per disegnare?

- `drawImage()`, `drawLine()`, `drawRect()`,
`drawRoundRect()`, `draw3DRect()`, `drawOval()`,
`drawArc()`, `drawString()`, `drawPolygon()`,
`drawPolyLine()`
- `fillRect()`, `fillRoundRect()`, `fill3DRect()`,
`fillOval()`, `fillArc()`, `fillPolygon()`,
`fillPolyLine()`
- `getColor()`, `getFont()`, `setColor()`, `setFont()`,
`copyArea()`, `clearRect()`

ESEMPIO 4: DISEGNO DI FIGURE

Il pannello personalizzato con il disegno:

```
public class MyPanel extends JPanel {  
    public void paintComponent(Graphics g){  
        super.paintComponent(g);  
        g.setColor(Color.red);  
        // white, gray, lightGray, darkGray  
        // red, green, yellow, pink, etc. etc.  
        g.fillRect(20,20, 100,80);  
        g.setColor(Color.blue);  
        g.drawRect(30,30, 80,60);  
        g.setColor(Color.black);  
        g.drawString("ciao",50,60);  
    }  
}
```



Il main che lo crea e lo inserisce nel frame:

```
import java.awt.*; import javax.swing.*;  
public class EsSwing4 {  
    public static void main(String[] v){  
        MyFrame f = new MyFrame("Esempio 4");  
        // potremmo anche usare un JFrame standard...  
        Container c = f.getContentPane();  
        MyPanel panel = new MyPanel();  
        c.add(panel);  
        f.show();  
    }  
}
```

ESEMPIO: DISEGNO DI FIGURE

Per cambiare font:

- si crea un oggetto `Font` appropriato
- lo si imposta come font predefinito usando il metodo `setFont()`

```
Font f1 =
    new Font("Times", Font.BOLD, 20);
// nome del font, stile, dimensione in punti
// stili possibili: Font.PLAIN, Font.ITALIC
g.setFont(f1);
```

Recuperare le proprietà di un font

- Il font corrente si recupera con `getFont()`
- Dato un `Font`, le sue proprietà si recuperano con `getName()`, `getStyle()`, `getSize()`
- e si verificano con i predicati `isPlain()`, `isBold()`, `isItalic()`

```
Font f1 = g.getFont();
int size = f1.getSize();
int style = f1.getStyle();
String name = f1.getName();
```

ESEMPIO: GRAFICO DI F(X) - 1

Per disegnare il grafico di una funzione occorre

- creare un'apposita classe `FunctionPanel` che estenda `JPanel`, ridefinendo il metodo `paintComponent()` come appropriato, ad esempio:
 - sfondo bianco, cornice nera
 - assi cartesiani rossi, con estremi indicati
 - funzione disegnata in blu
- creare, nel main, un oggetto di tipo `FunctionPanel`

Definizione del solito main:

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

public class EsSwing5 {
    public static void main(String[] v){
        JFrame f = new JFrame("Grafico f(x)");
        Container c = f.getContentPane();
        FunctionPanel p = new FunctionPanel();
        c.add(p);
        f.setBounds(100,100,500,400);
        f.show();
    }
}
```

ESEMPIO: GRAFICO DI F(X) - 2

Definizione del pannello apposito:

```
class FunctionPanel extends JPanel {
    int xMin=-7, xMax=7, yMin=-1, yMax=1;
    // gli intervalli in cui vogliamo graficare
    int larghezza=500, altezza=400;
    // corrispondono alla grandezza del JFrame
    // ERA MEGLIO USARE UN COSTRUTTORE...
    float fattoreScalaX, fattoreScalaY;

    public void paintComponent(Graphics g){
        super.paintComponent(g); // va fatto sempre
        setBackground(Color.white); // fondo bianco
        fattoreScalaX=larghezza/((float)xMax-xMin);
        fattoreScalaY=altezza/((float)yMax-yMin);
        // dobbiamo fare le proporzioni tra
        // l'intervallo di valori della finestra
        // (500*400) e l'intervallo da graficare (14*2)

        // incornicia il grafico in nero
        g.setColor(Color.black);
        g.drawRect(0,0, larghezza-1, altezza-1);
        // e disegna degli assi cartesiani
        g.setColor(Color.red);
        g.drawLine(0, altezza/2, larghezza-1, altezza/2);
        g.drawLine(larghezza/2, 0, larghezza/2, altezza-1);
        // scrittura valori estremi degli assi
        g.drawString(""+xMin, 5, altezza/2-5);
        g.drawString(""+xMax, larghezza-10, altezza/2-5);
        g.drawString(""+yMax, larghezza/2+5, 15);
        g.drawString(""+yMin, larghezza/2+5, altezza-5);
    }
}
```

Continua.....

Continua grafico della funzione $f(x)$ - 3

```
// disegna il grafico della funzione in blu
g.setColor(Color.blue);
setPixel(g,xMin,f(xMin)); // punto iniziale

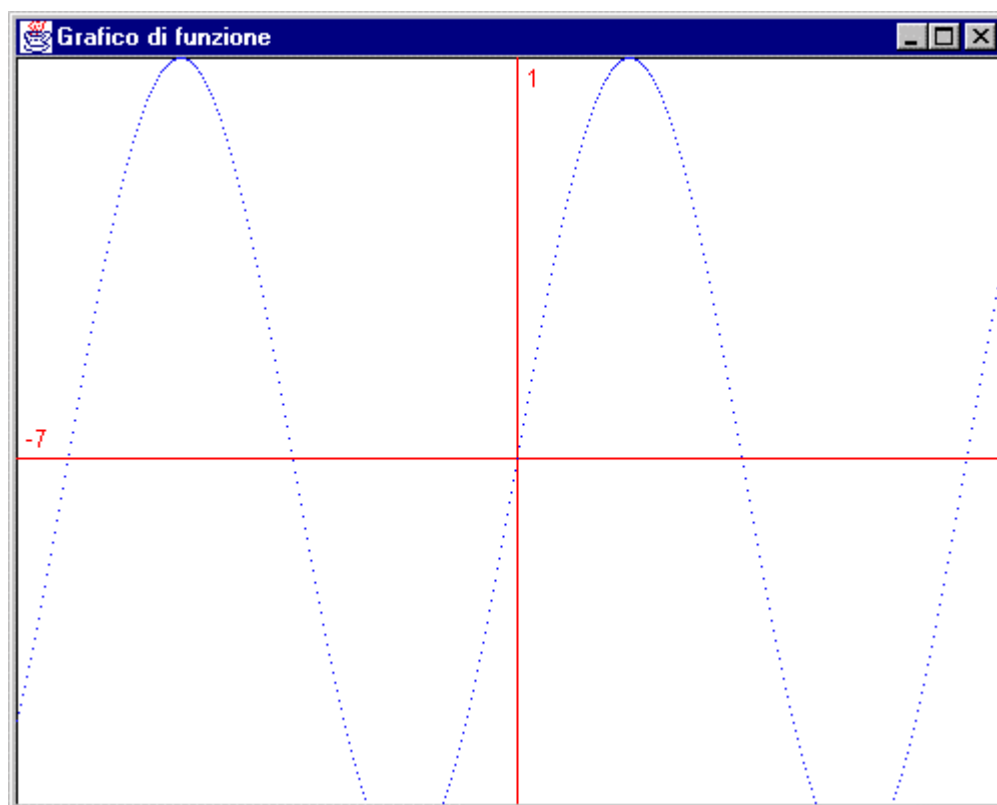
for (int ix=1; ix<larghezza; ix++){
// per ognuno dei pixel della finestra
float x = xMin+((float)ix)/fattoreScalaX;
setPixel(g,x,f(x));
}
}

// definizione della funzione,
// statica, da graficare
static float f(float x){
return (float)Math.sin(x);
// sin(x) è la funzione (statica!)
//che decidiamo di graficare:
//ovviamente potrebbe essere qualsiasi funzione
}

// questa serve per riportare i valori della
// funzione sui valori della finestra
void setPixel(Graphics g, float x, float y){
if (x<xMin || x>xMax || y<yMin || y>yMax )
return;
int ix = Math.round((x-xMin)*fattoreScalaX);
int iy = altezza-Math.round(
(y-yMin)*fattoreScalaY);
g.drawLine(ix,iy,ix,iy);
// disegna in effetti un singolo punto
}
}
```


ESEMPIO: GRAFICO DI F(X) - 4

Ecco ciò che si ottiene:



DISEGNARE IMMAGINI

Come si disegna un'immagine presa da un file (p.e. una immagine JPG)?

1) ci si procura un apposito oggetto Image

1a) si recupera il "toolkit di default":

```
Toolkit tk = Toolkit.getDefaultToolkit();
```

1b) si chiede al toolkit di recuperare l'immagine:

```
Image img = tk.getImage("new.gif");
```

Sono supportati i formati GIF e JPEG

Si può anche fornire un URL:

```
URL url = ...;
```

```
Image img = tk.getImage(url);
```

2) si disegna l'immagine con `drawImage()`

PROBLEMA: `drawImage()` ritorna al chiamante subito dopo aver *iniziato* il caricamento dell'immagine, senza attendere di averla caricata. C'è il rischio che l'immagine non faccia in tempo a visualizzarsi prima della fine del programma.

SOLUZIONE: si crea un oggetto `MediaTracker` dedicato ad occuparsi del caricamento dell'immagine, e a cui appunto il caricamento dell'immagine (o delle immagini), e gli si affida l'immagine da caricare

DISEGNARE IMMAGINI

Uso del MediaTracker

- 1) Nel costruttore del pannello, si crea un oggetto `MediaTracker`, precisandogli su quale componente avverrà il disegno... Di solito il parametro è `this` (il pannello stesso)

```
MediaTracker mt = new MediaTracker(this);
```

- 2) ...si aggiunge l'immagine al `MediaTracker`...

```
mt.addImage(img, 1);
```

Il secondo parametro è un numero intero, a nostra scelta, che identifica univocamente l'immagine.

- 3) ..e gli si dice di attendere il caricamento di tale immagine, usando il numero intero (**ID**) da noi assegnato

```
try { mt.waitForID(1); }  
catch (InterruptedException e) {}
```

Occorre un blocco `try/catch` perché l'attesa potrebbe essere interrotta da un'eccezione.

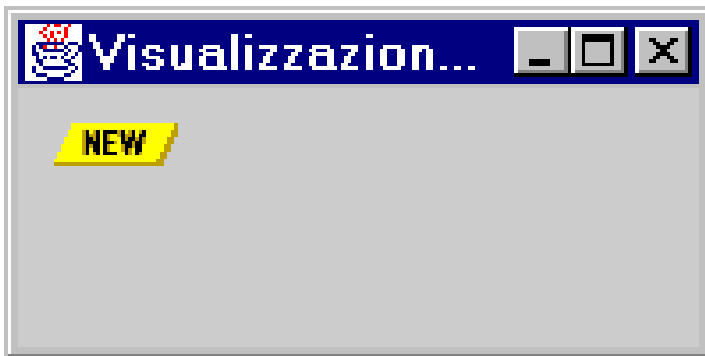
Se si devono attendere molte immagini:

```
try { mt.waitForAll(); }  
catch (InterruptedException e) {}
```

DISEGNARE IMMAGINI: ESEMPIO

```
public class ImgPanel extends JPanel {
    Image img1;
    public ImgPanel(){
        Toolkit tk = Toolkit.getDefaultToolkit();
        img1 = tk.getImage("new.gif");
        MediaTracker mt = new MediaTracker(this);
        mt.addImage(img1, 1);
        // aggiunta di eventuali altre immagini
        try { mt.waitForAll(); }
        catch (InterruptedException e){}
    }

    public void paintComponent(Graphics g){
        super.paintComponent(g);
        g.drawImage(img1, 30, 30, null);
        /* Immagine (img1), posizione nel pannello (30,30)
        e un oggetto (null, cioè nessuno) a cui notificare
        l'avvenuto caricamento */
    }
}
```



ESEMPIO 7: IL COMPONENTE JLabel

Oltre a disegnare, dentro ai pannelli si possono inserire altre componenti....

Il componente **JPanel** non fa altro che scrivere qualcosa nel pannello.

Il solito main:

```
import java.awt.*; import javax.swing.*;
public class EsSwing7 {
    public static void main(String[] v){
        JFrame f = new JFrame("Esempio 7");
        Container c = f.getContentPane();
        Es7Panel p = new Es7Panel();
        c.add(p);
        f.pack(); //pack dimensiona il frame in modo da
                //contenere esattamente il pannello
        f.show();
    } }
```

```
public class Es7Panel extends JPanel {
    public Es7Panel(){
        super();
        JLabel l = new JLabel("Etichetta");
        add(l);
    }
}
```

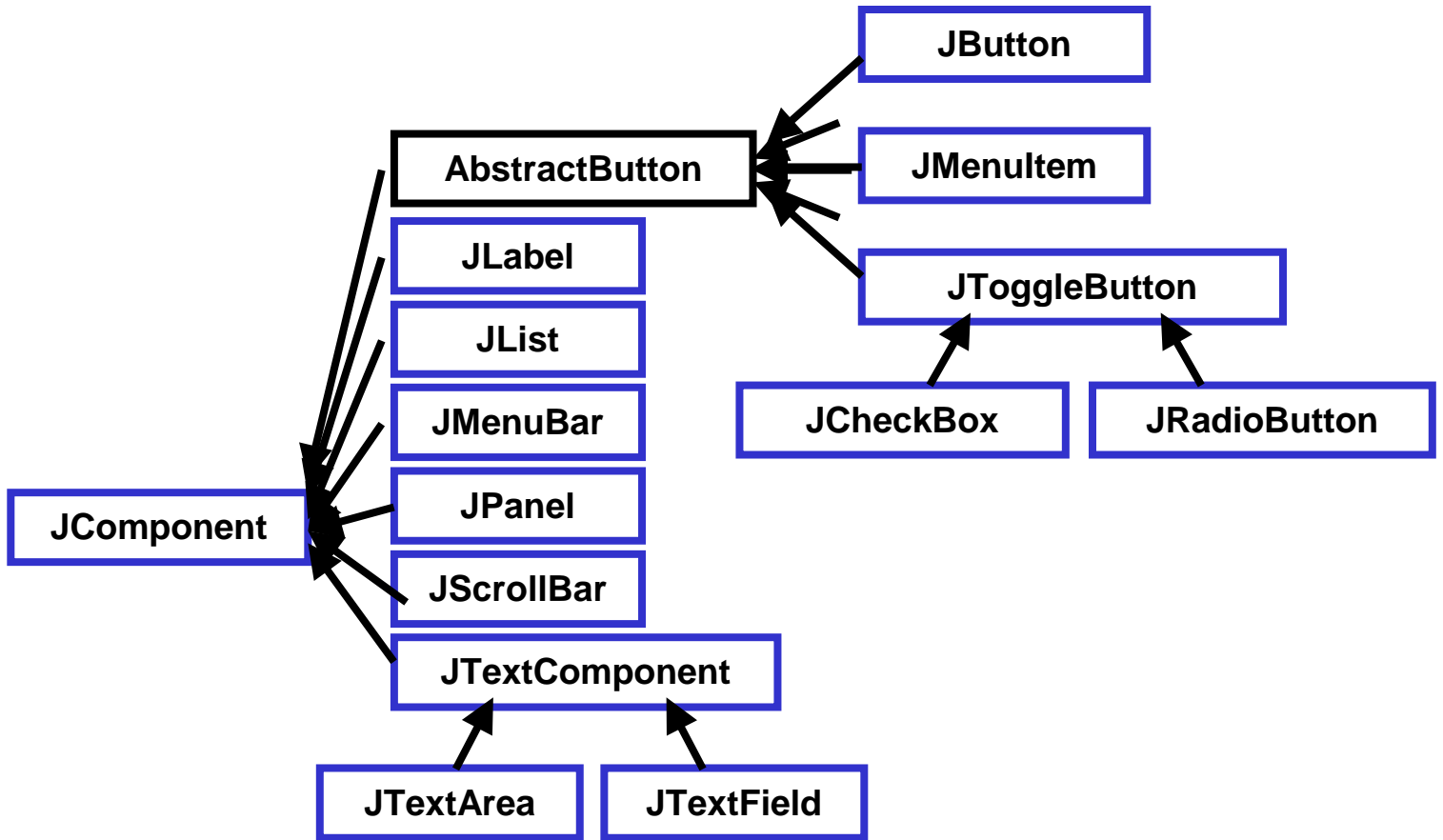


OLTRE IL SOLO DISEGNO: GLI EVENTI

- Finora, la grafica considerata consisteva nel *puro disegno* di forme e immagini
- È grafica "passiva": non consente all'utente alcuna interazione
 - si può solo **guardare** il disegno...!!
- La costruzione di interfacce grafiche richiede invece interattività
 - l'utente deve poter premere bottoni, scrivere testo, scegliere elementi da liste, etc etc

- Componenti *attivi*, che generano eventi

SWING: GERARCHIA DI CLASSI



JLabel: UNICO componente passivo, cioè che non genera eventi

Gli altri sono tutti componenti **ATTIVI** che generano eventi

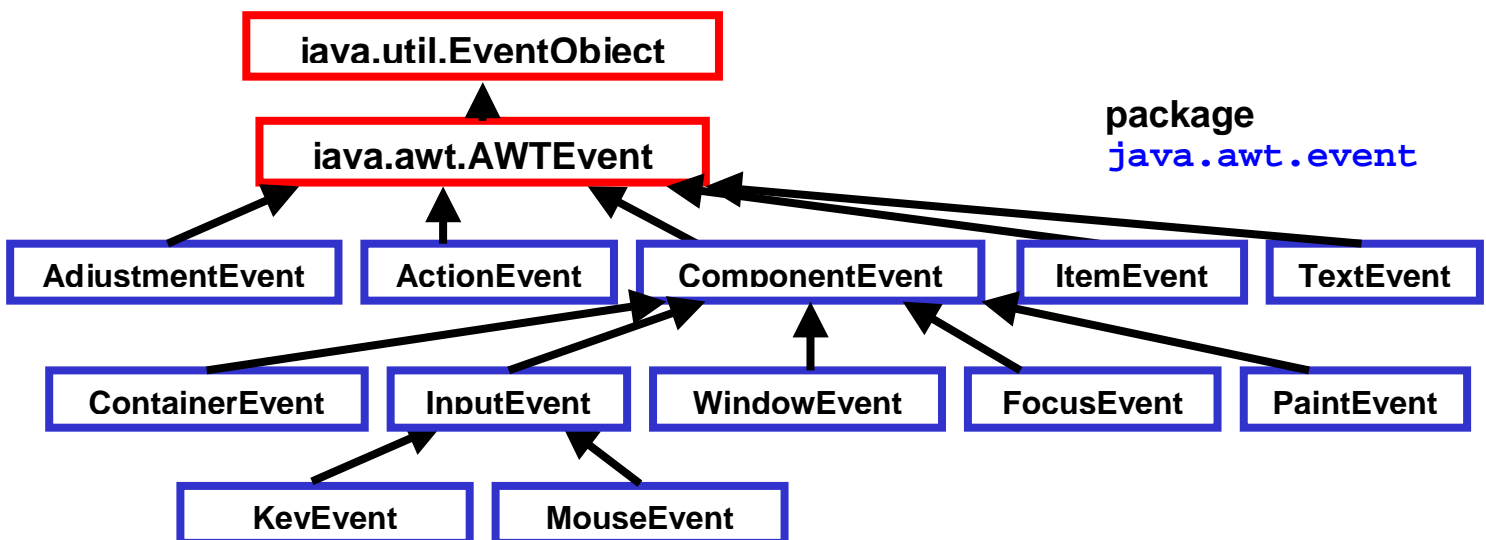
Esempio:

JButton: è il classico "bottono", e genera un evento quando viene premuto

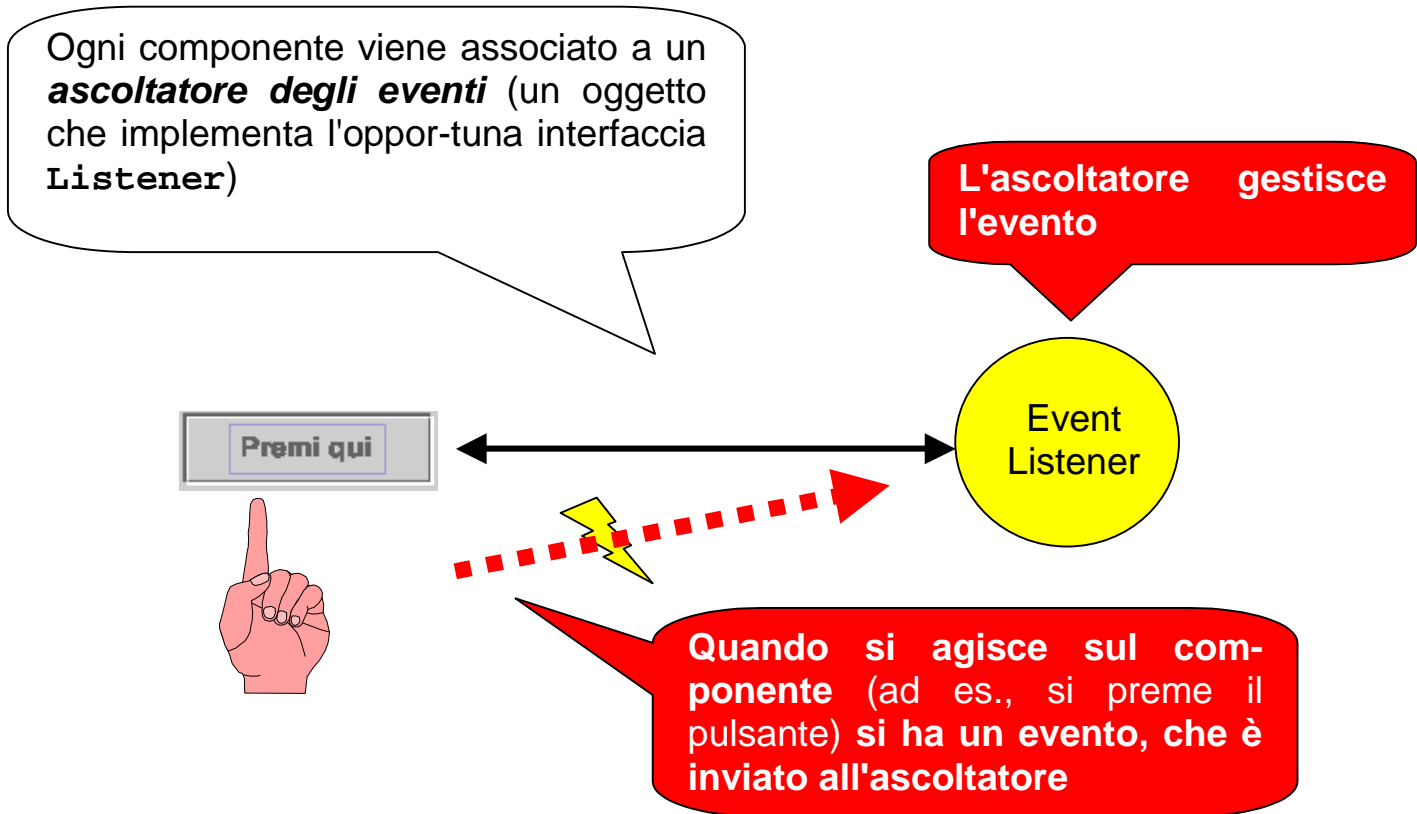
EVENTI

- Ogni componente grafico, quando si opera su di esso, genera un evento che descrive cosa è accaduto (attenzione: il concetto di evento non si applica necessariamente solo agli oggetti grafici, ma è generalmente con la grafica che esso assume rilevanza e comprensione immediata)
- Tipicamente, ogni componente può generare *molti tipi diversi di eventi*, in relazione a ciò che sta accadendo
 - un bottone può generare l'evento "azione" che significa che è stato premuto
 - una casella di opzione può generare l'evento "stato modificato" per la casella è stata selezionata o deselezionata

In Java, un *evento* è un oggetto, istanza di (una sottoclasse di) `java.util.EventObject`



GESTIONE DEGLI EVENTI



- Quando si interagisce con un componente "attivo" si genera un evento, che è un oggetto `Event` della (sotto)classe opportuna
 - l'oggetto `Event` contiene tutte le informazioni sull'evento (chi l'ha creato, cosa è successo, etc)
- Il sistema invia tale "oggetto Evento" all'oggetto ascoltatore degli eventi preventivamente *registrato* come tale, che gestisce l'evento.
- L'attività non è più algoritmica (input / computazione / output), è interattiva e reattiva

IL PULSANTE JButton

- Quando viene premuto, un bottone genera un evento di classe `ActionEvent`
- Questo evento viene inviato dal sistema allo specifico ascoltatore degli eventi per quel bottone.
- L'ascoltatore degli eventi deve implementare la interfaccia `ActionListener`,
 - può essere un oggetto di un'altra classe al di fuori del pannello...
 - .. o può essere anche il pannello stesso nel quale (`this`)
- Tale ascoltatore degli eventi deve implementare il metodo definito nella interfaccia `ActionListener`

```
void actionPerformed(ActionEvent ev);
```

che **gestisce l'evento**, nel senso che reagisce all'evento con opportune azioni

ESEMPIO 8: USO DI JButton

- Un'applicazione fatta da un'etichetta (JLabel) e un pulsante (JButton)
- L'etichetta può valere "Tizio" o "Caio"; all'inizio vale "Tizio"
- Premendo il bottone, l'etichetta deve commutare, diventando "Caio" se era "Tizio", o "Tizio" se era "Caio"



Architettura dell'applicazione

- Un pannello che contiene etichetta e pulsante
→ il costruttore del pannello crea l'etichetta e il pulsante
- Il pannello fa da *ascoltatore degli eventi* per il pulsante → il costruttore del pannello imposta il pannello stesso come *ascoltatore degli eventi* del pulsante

```
// Il codice del pannello...
public class Es8Panel extends JPanel implements
    ActionListener{
    private JLabel l;
    public Es8Panel(){
        super();
        l = new JLabel("Tizio");
        add(l);
        JButton b = new JButton("Tizio/Caio");
        // Tizio/Caio è l'etichetta del pulsante
        b.addActionListener(this);
        // registra l'oggetto panel stesso come
        // ascoltatore degli eventi
        add(b); }
}
```

Eventi da gestire:

- l'evento di azione sul pulsante deve provocare il *cambio del testo dell'etichetta*

Come si fa?

- il testo dell'etichetta si può recuperare con `getText()` e cambiare con `setText()`
- l'ascoltatore dell'evento, che implementa il metodo `ActionPerformed()`, deve recuperare il testo dell'etichetta e cambiarlo

```
public void actionPerformed(ActionEvent e){
    if (l.getText().equals("Tizio"))
        l.setText("Caio");
    else
        l.setText("Tizio");
}
}
```

ESEMPIO 8: Il solito main:

```
import java.awt.*; import javax.swing.*;
import java.awt.event.*;
// bisogna importare il package degli eventi!

public class EsSwing8 {
    public static void main(String[] v){
        JFrame f = new JFrame("Esempio 7");
        Container c = f.getContentPane();
        Es8Panel p = new Es8Panel();
        c.add(p);
        f.pack(); f.show();
    }
}
```



ESEMPIO 8: UNA VARIANTE

Architettura dell'applicazione

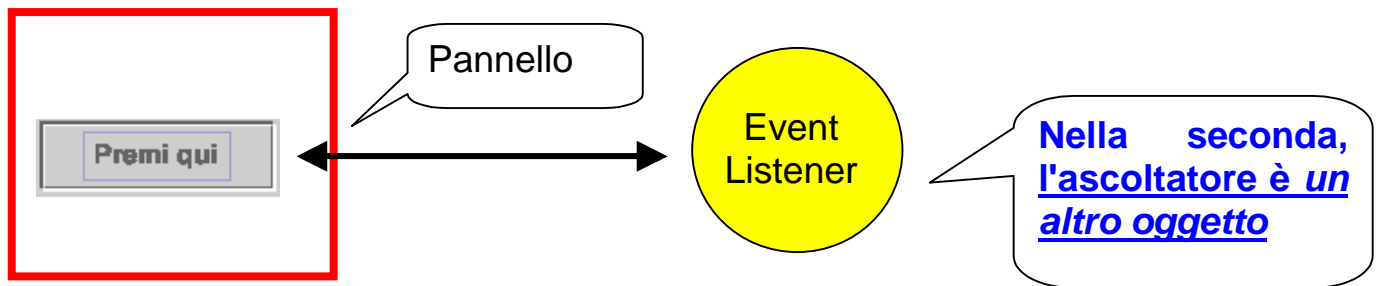
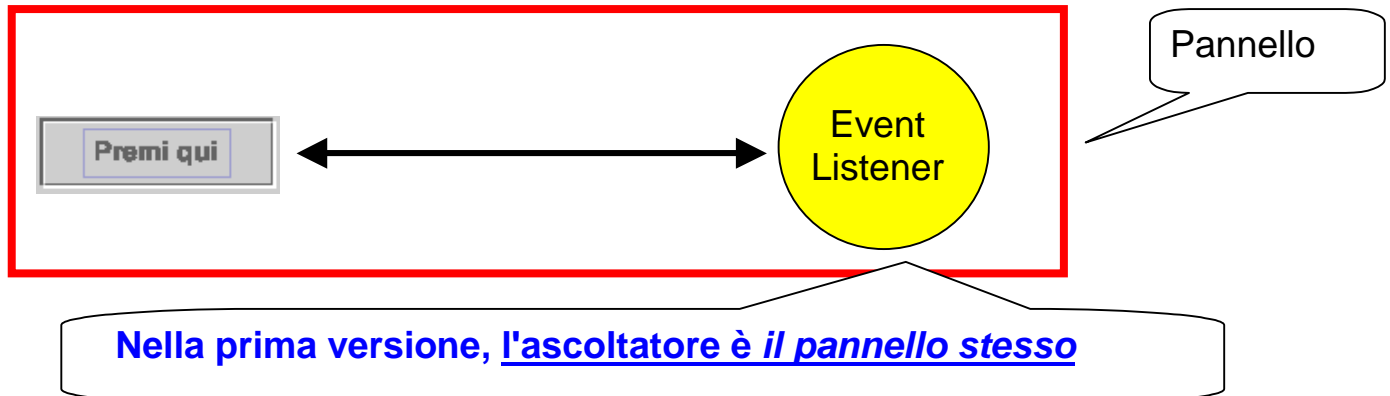
- Un pannello che contiene etichetta e pulsante
→ il costruttore del pannello crea l'etichetta e il pulsante
- L'*ascoltatore degli eventi* per il pulsante è un oggetto separato → il costruttore del pannello imposta tale oggetto come *ascoltatore degli eventi* del pulsante

```
public class Es8Panel extends JPanel {
    public Es8Panel(){
        super();
        JLabel l = new JLabel("Tizio");
        add(l);
        JButton b = new JButton("Tizio/Caio");
        b.addActionListener(new Es8Listener(l) );
        // crea un oggetto es8Listener e lo imposta
        // come ascoltatore degli eventi del bottone
        add(b);  } }
```

L'ascoltatore degli eventi:

```
class Es8Listener implements ActionListener{
    public void actionPerformed(ActionEvent e){
        if (l.getText().equals("Tizio"))
            l.setText("Caio");
        else
            l.setText("Tizio");
    }
    private JLabel l;
    public Es8Listener(JLabel label){l=label;}
    // deve farsi dare come parametro la JLabel su
    // cui dovrà andare ad agire
}
```

CONFRONTO FRA LE DUE VERSIONI



ESEMPIO 9: DUE PULSANTI

Scopo dell'applicazione

- Cambiare il colore di sfondo tramite *due pulsanti*: uno lo rende rossa, l'altro azzurro

Architettura dell'applicazione

- Un pannello che contiene i due pulsanti creati dal costruttore del pannello
- Un unico ascoltatore degli eventi per entrambi i pulsanti
 - necessità di capire, in `actionPerformed()`, quale pulsante è stato premuto

Il codice del pannello:

```
public class Es9Panel extends JPanel implements
    ActionListener {
    JButton b1, b2;
    public Es9Panel(){
        super();
        b1 = new JButton("Rosso");
        b2 = new JButton("Azzurro");
        b1.addActionListener(this);
        b2.addActionListener(this);
    // il pannello fa da ascoltatore degli
    // eventi per entrambi i pulsanti
        add(b1);
        add(b2);
    }
    ...
}
```


ESEMPIO 9, continua il codice del pannello...

```
...
public void actionPerformed(ActionEvent e){
    Object pulsantePremuto = e.getSource();
    // si recupera il riferimento all'oggetto
    // che ha generato l'evento
    if (pulsantePremuto==b1)
    // e si confronta questa con i riferimenti
    // agli oggetti bottoni b1 e b2
        setBackground(Color.red);
    if (pulsantePremuto==b2)
        setBackground(Color.cyan);
    }
}
```

Dato l'oggetto-evento, il suo metodo `getSource` restituisce un riferimento all'oggetto che ha generato l'evento stesso.

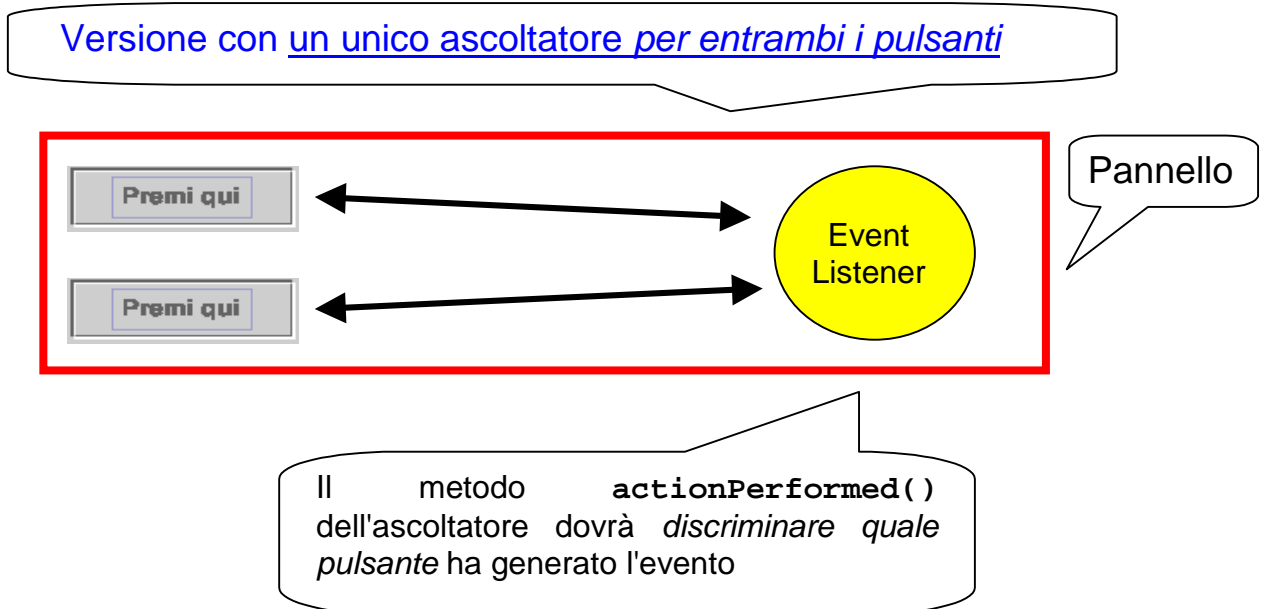


Un modo alternativo per capire chi aveva generato l'evento poteva essere quello di guardare l'etichetta associata al pulsante:

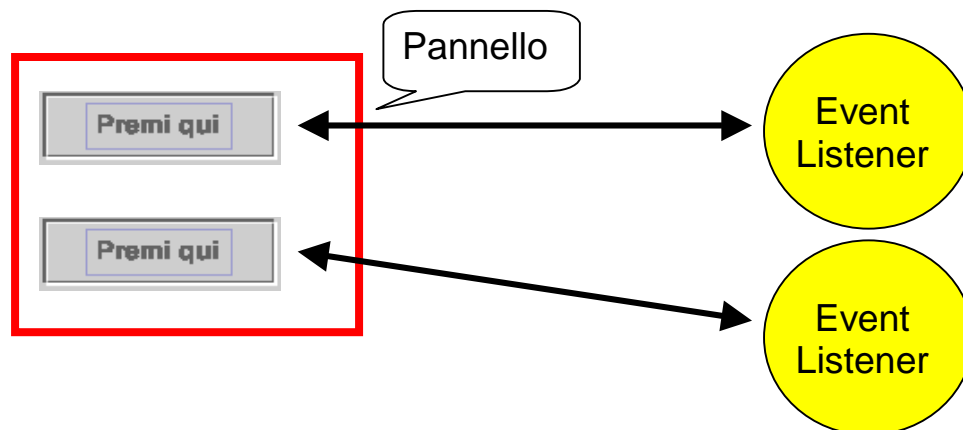
```
String nome = e.getActionCommand();
if (nome.equals("Rosso") ...
```

ESEMPIO 9: VARIANTE

Prima abbiamo definito un singolo ascoltatore per entrambi i pulsanti:



Se definiamo ascoltatori diversi per eventi diversi il sistema provvederà ad inviare gli eventi solo all'ascoltatore opportuno, e il metodo `actionPerformed` non deve più preoccuparsi di sapere quale pulsante è stato premuto



ESEMPIO 9: variante IL PANNELLO:

```
class Es9PanelBis extends JPanel {
    public Es9PanelBis(){
        super();
        JButton b1 = new JButton("Rosso");
        JButton b2 = new JButton("Azzurro");
        b1.addActionListener(
            new Es9Listener(this,Color.red) );
        b2.addActionListener(
            new Es9Listener(this,Color.cyan) );
        // crea due oggetti ascoltatori e a ognuno
        // passa il riferimento del pannello su cui
        // agire (this) e il colore da usare
        add(b1);
        add(b2);
    }
}
```

L'ascoltatore degli eventi:

```
class Es9Listener implements ActionListener{
    private JPanel pannello;
    private Color colore;
    public Es9Listener(JPanel p, Color c){
        pannello = p; colore = c;
    }

    public void actionPerformed(ActionEvent e){
        pannello.setBackground(colore);
    }
}
```

GLI EVENTI DI FINESTRA

Le operazioni sulle finestre (finestra chiusa, aperta, minimizzata, ingrandita...) generano un `WindowEvent`

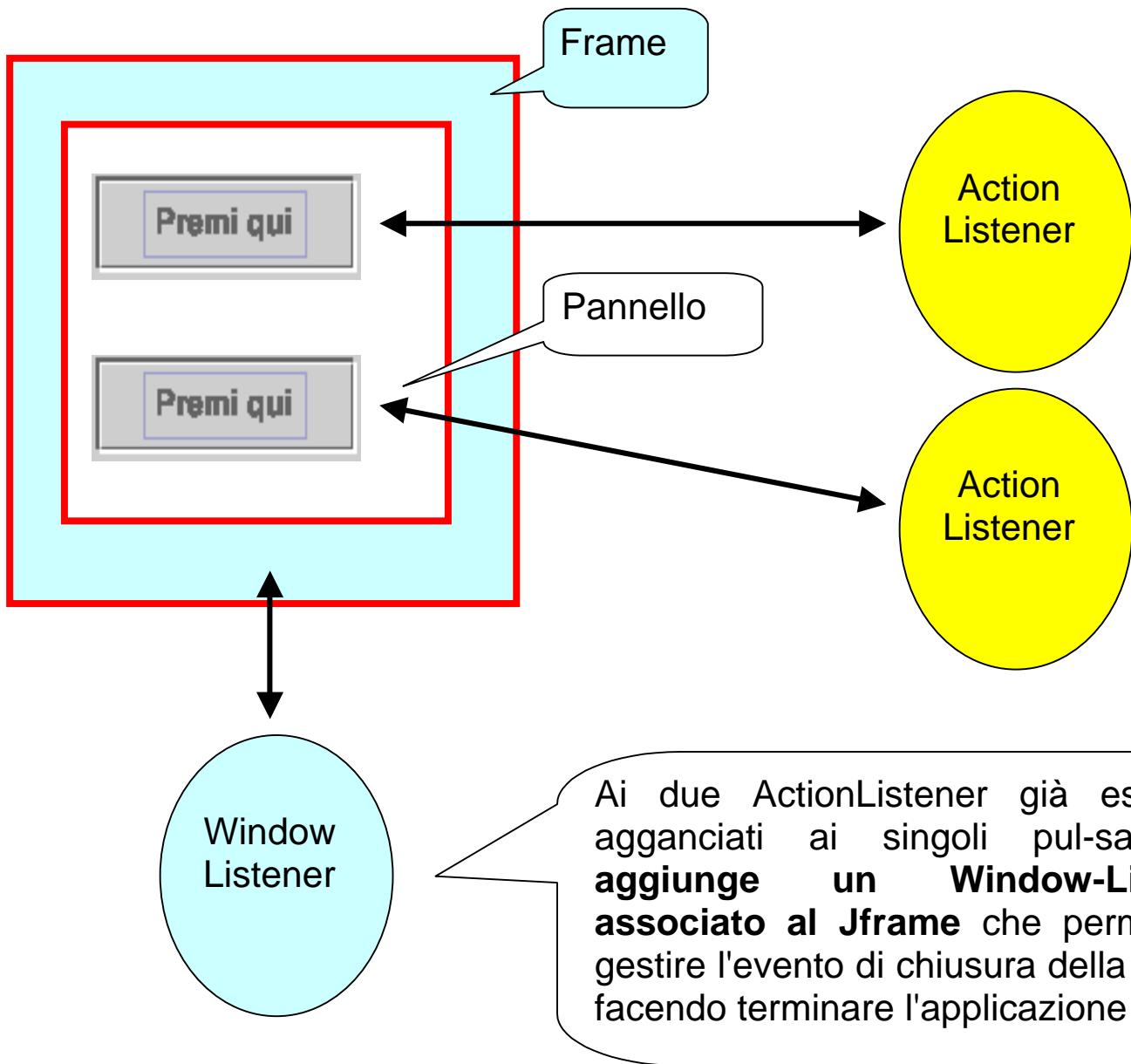
- Gli eventi di finestra sono gestiti dai metodi dichiarati dall'interfaccia `WindowListener`

```
public void windowClosed(WindowEvent e);  
public void windowClosing(WindowEvent e);  
public void windowOpened(WindowEvent e);  
public void windowIconified(WindowEvent e);  
public void windowDeiconified(WindowEvent e);  
public void windowActivated(WindowEvent e);  
public void windowDeactivated(WindowEvent e);
```

- ogni metodo viene scatenato dall'evento appropriato (p.e., quando si iconifica una finestra, nell'ascoltatore viene invocato il metodo `windowIconified()`) e gestisce l'evento appropriato, automaticamente
- Il comportamento predefinito di questi metodi *va già bene* tranne `windowClosing()`, che *non* fa uscire l'applicazione: nasconde solo la finestra.
- Per far sì che chiudendo la finestra del frame l'applicazione venga chiusa, il frame deve implementare l'interfaccia `WindowListener`, e ridefinire `windowClosing` in modo che invochi `System.exit()`
- Gli altri metodi devono essere *formalmente implementati*, ma, non dovendo svolgere compiti precisi, possono essere definiti semplicemente con un *corpo vuoto*:

```
public void WindowOpened(WindowEvent e){}
```

ESEMPIO 9 CON GESTIONE DELLA CHIUSURA DELLA FINESTRA



ESEMPIO 9 CON CHIUSURA

```
public class EsSwing9 {
    public static void main(String[] v){
        JFrame f = new JFrame("Esempio 9");
        Container c = f.getContentPane();
        Es9Panel p = new Es9Panel();
        c.add(p);
        f.addWindowListener( new Terminator() );
        // Terminator è la classe che implementa
        // l'interfaccia WindowListener
        f.pack();
        f.show();
    }
}

class Terminator implements WindowListener {
    public void windowClosed(WindowEvent e){}
    public void windowClosing(WindowEvent e){
        System.exit(0);
    }
    // in questo modo chiudendo la finestra
    // si esce dalla applicazione

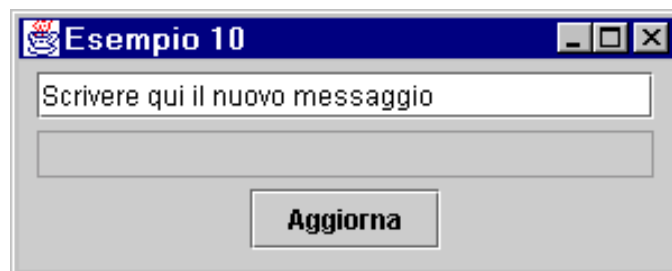
    public void windowOpened(WindowEvent e){}
    public void windowIconified(WindowEvent e){}
    public void windowDeiconified(WindowEvent e){}
    public void windowActivated(WindowEvent e){}
    public void windowDeactivated(WindowEvent e){}
}
```

IL CAMPO DI TESTO JTextField

- Il `JTextField` è un componente "campo di testo", usabile per scrivere e visualizzare *una riga* di testo
 - il campo di testo può essere editabile o no
 - il testo è accessibile con `getText()` / `setText()`
- Il campo di testo è parte di un oggetto `Document`
- *Ogni volta che il testo in esso contenuto cambia* si genera un `DocumentEvent` nel documento che contiene il campo di testo
- Se però è sufficiente registrare i cambiamenti solo quando si preme INVIO, basta gestire semplicemente il solito `ActionEvent`

ESEMPIO 10

- Un'applicazione comprendente un pulsante e due campi di testo
 - uno per scrivere testo, l'altro per visualizzarlo
- Quando si preme il pulsante, il testo del secondo campo (non modificabile dall'utente) viene cambiato, e reso uguale a quello scritto nel primo
- L'unico evento è ancora il pulsante premuto: *ancora non usiamo* il `DocumentEvent`



ESEMPIO 10 - 2

Il solito main:

```
public class EsSwing10 {
    public static void main(String[] v){
        JFrame f = new JFrame("Esempio 10");
        Container c = f.getContentPane();
        Es10Panel p = new Es10Panel();
        c.add(p);
        f.addWindowListener( new Terminator() );
        f.setSize(300,120);
        f.show();
    }
}
```

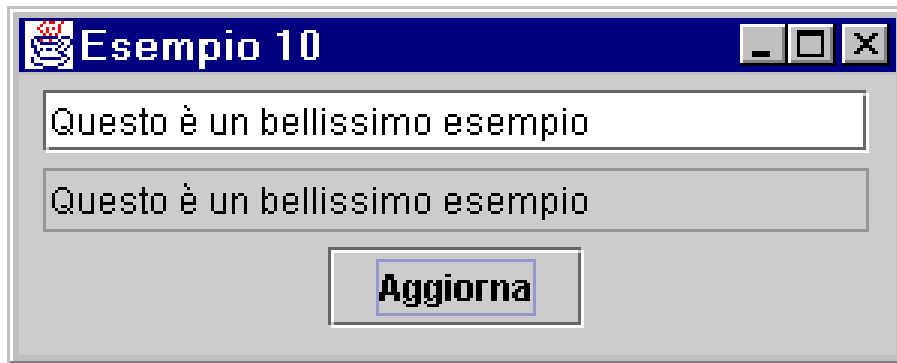
Il pannello:

```
class Es10Panel extends JPanel
    implements ActionListener {
    JButton b;
    JTextField txt1, txt2;
    public Es10Panel(){
        super();
        b = new JButton("Aggiorna");
        txt1=new JTextField("Scrivere qui il testo", 25);
        txt2 = new JTextField(25); // larghezza in caratt.
        txt2.setEditable(false); // non modificabile
        b.addActionListener(this);
        add(txt1);
        add(txt2);
        add(b);
    }
}
```


ESEMPIO 10 - 3

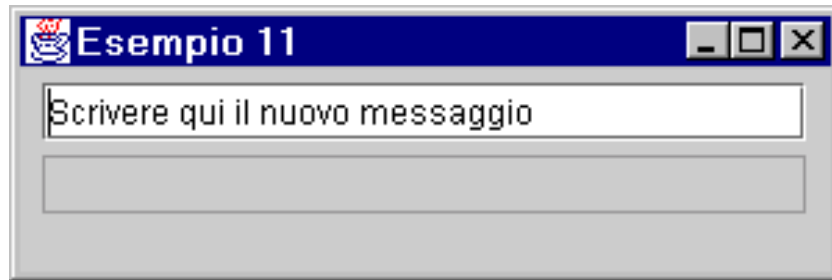
La gestione dell'evento "pulsante premuto":

```
public void actionPerformed(ActionEvent e){  
    txt2.setText( txt1.getText() );  
}  
  
}
```



ESEMPIO 11: VARIANTE ALL'ESEMPIO 10

- Niente più pulsante, solo i due campi di testo



- Sfruttiamo la pressione del tasto INVIO come pulsante, quindi intercettiamo l'ActionEvent (*ancora non usiamo il DocumentEvent*)
- Quando si preme INVIO, il testo del secondo campo (non modificabile dall'utente) viene cambiato, e reso uguale a quello scritto nel primo
- Dobbiamo mettere un ActionListener in Ascolto sul campo di testo txt1 pronto ad intercettare gli eventi di azione(ActionEvent (che si scatena con la pressione del tasto invio)

ESEMPIO 11

```
class Es11Panel extends JPanel
    implements ActionListener {
    JTextField txt1, txt2;
    public Es11Panel(){
        super();
        txt1=new JTextField("Scrivere qui il testo", 25);
        txt2 = new JTextField(25);
        txt2.setEditable(false);
        txt1.addActionListener(this);
        // gli eventi di txt1 vengono ascoltati da this
        add(txt1);
        add(txt2);
    }
    ...
}
```

La gestione dell'evento rimane inalterata: è cambiato solo colui che genera l'evento.

ESEMPIO 12: ULTERIORE VARIANTE

- Sfruttiamo il concetto di DOCUMENTO che sta dietro a ogni campo di testo
- A ogni modifica del contenuto, il documento di cui il campo di testo fa parte genera un `DocumentEvent` per segnalare l'avvenuto cambiamento
- Tale evento dev'essere gestito da un opportuno `DocumentListener` cioè da un oggetto di una classe che implementi l'interfaccia `DocumentListener`

DOCUMENT LISTENER

- L'interfaccia `DocumentListener` dichiara *tre metodi*:

```
void insertUpdate(DocumentEvent e);  
void removeUpdate(DocumentEvent e);  
void changedUpdate(DocumentEvent e);
```

Il terzo *non è mai chiamato* da un `JTextField`, serve solo per altri tipi di componenti

- L'oggetto-evento `DocumentEvent` passato come parametro in realtà è inutile, in quanto cosa sia accaduto è già implicito nel metodo chiamato; esso esiste solo per uniformità. La stessa cosa valeva per i `WindowListener`.

ESEMPIO 12 - 2

Nel nostro caso:

- l'azione da svolgere in caso di inserimento o rimozione di caratteri è *identica*, quindi i due metodi
`void insertUpdate(DocumentEvent e);`
`void removeUpdate(DocumentEvent e);`
saranno *identici* (purtroppo vanno comunque implementati entrambi)
- Il metodo `changedUpdate(DocumentEvent e)` è pure inutile, dato che `JTextField` non lo chiama, ma va comunque formalmente implementato.

ESEMPIO 12: CODICE

```
import javax.swing.event.*;
// solito main...
class Es12Panel extends JPanel
    implements DocumentListener {
// deve implementare l'interfaccia
    JTextField txt1, txt2;

public Es12Panel(){
    super();
    txt1= new JTextField("Scrivere qui il testo", 25);
    txt2 = new JTextField(25);
    txt2.setEditable(false);
    txt1.getDocument().addDocumentListener(this);
// ricava il documento di cui il campo
// di test txt1 fa parte e gli associa il
// pannello come listener
    add(txt1);
    add(txt2);
}

// La gestione dell'evento:

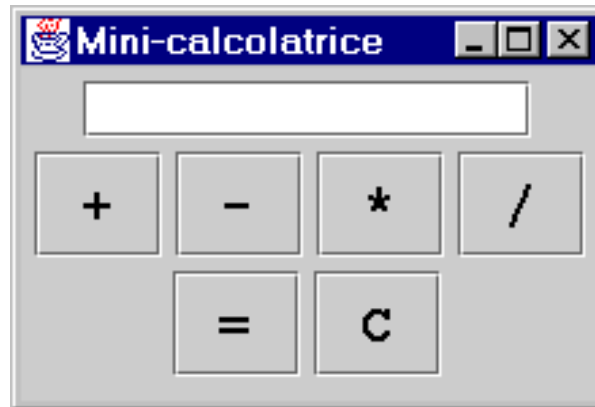
public void insertUpdate(DocumentEvent e){
    txt2.setText(txt1.getText()); }
public void removeUpdate(DocumentEvent e){
    txt2.setText(txt1.getText()); }
public void changedUpdate(DocumentEvent e){}
// implementazione formale
```

Ora, a ogni inserimento o cancellazione di caratteri l'aggiornamento è automatico

ESEMPIO: UNA MINI-CALCOLATRICE

Architettura:

- un pannello con un campo di testo e sei pulsanti
- un unico ActionListener per tutti i pulsanti (è il vero calcolatore)



Gestione degli eventi: Ogni volta che si preme un pulsante:

- si recupera il nome del pulsante (è la successiva operazione da svolgere)
- si legge il valore nel campo di testo
- si svolge l'operazione precedente

Esempio: $15 + 14 - 3 = + 8 =$

- quando si preme +, si memorizzano sia 15 sia l'operazione +
- quando si preme -, si legge 14, si fa la somma $15+14$, si memorizza 29, e si memorizza l'operazione -
- quando si preme =, si legge 3, si fa la sottrazione $29-3$, si memorizza 26, e si memorizza l'operazione =
- quando si preme + (dopo l' =), è come essere all'inizio: si memorizzano 26 (risultato precedente) e l'operazione +
- quando si preme =, si legge 8, si fa la somma $26+8$, si memorizza 34, e si memorizza l'operazione =
- ...eccetera...

MINI-CALCOLATRICE - 2

Il solito main:

```
public class EsSwingCalculator {
    public static void main(String[] v){
        JFrame f = new JFrame("Mini-calcolatrice");
        Container c = f.getContentPane();
        CalcPanel p = new CalcPanel();
        c.add(p);
        f.setSize(220,150);
        f.addWindowListener(new Terminator());
        // Per gestire la chiusura della finestra
        f.show();
    }
}
```

Un pulsante con un font "personalizzato" :

```
class CalcButton extends JButton {
    CalcButton(String n) {
        super(n);
        setFont(new Font("Courier",Font.BOLD,20));
        // estendiamo JButton per personalizzare il font
    }
}
```


MINI-CALCOLATRICE - 3

Il pannello:

```
class CalcPanel extends JPanel {
    JTextField txt;
    CalcButton sum, sub, mul, div, calc, canc;
    public CalcPanel(){
        super();
        txt = new JTextField(15);
        txt.setHorizontalAlignment(JTextField.RIGHT);
        calc = new CalcButton("=");
        sum = new CalcButton("+");
        sub = new CalcButton("-");
        mul = new CalcButton("*");
        div = new CalcButton("/");
        canc = new CalcButton("C");
        add(txt);
        add(sum); add(sub); add(mul);
        add(div); add(calc); add(canc);
        Calculator calcolatore = new Calculator(txt);
        // l'unico ascoltatore è questo oggetto
        // calcolatore che gestisce tutti gli eventi
        // e rappresenta il vero e proprio calcolatore
        sum.addActionListener(calcolatore);
        sub.addActionListener(calcolatore);
        mul.addActionListener(calcolatore);
        div.addActionListener(calcolatore);
        calc.addActionListener(calcolatore);
        canc.addActionListener(calcolatore);
    }
}
```

MINI-CALCOLATRICE - 3

Il listener / calcolatore:

```
class Calculator implements ActionListener {
    double res = 0; JTextField display;
    String opPrec = "nop";
    public Calculator(JTextField t) { display = t; }
    public void actionPerformed(ActionEvent e){
        double valore =
            Double.parseDouble(display.getText());
// recupera il valore dal campo di testo
// e lo converte da stringa a double
        display.setText("");
        display.requestFocus();
// fa si' che il campo di testo sia già
// selezionato, pronto per scriverci dentro

        String operazione = e.getActionCommand();
// recupera il nome del pulsante premuto
// e' un modo alternativo per capire, tra tanti
// bottoni, quale e' ha generato l'evento
        if (operazione.equals("C")) { //cancella tutto
            res = valore = 0; opPrec = new String("nop");
        } else { // esegui l'operazione precedente
            if (opPrec.equals("+")) res += valore; else
            if (opPrec.equals("-")) res -= valore; else
            if (opPrec.equals("*")) res *= valore; else
            if (opPrec.equals("/")) res /= valore; else
            if (opPrec.equals("nop")) res = valore;
            display.setText(""+res);
            opPrec = operazione;
//la prossima operazione da eseguire è la corrente
        } } }
```

IL CHECKBOX (casella di opzione)

- Il `JCheckBox` è una "casella di opzione", che può essere selezionata o deselezionata
 - lo stato è verificabile con `isSelected()` e modificabile con `setSelected()`
- *Ogni volta che lo stato della casella cambia*, si generano:
 - un `ActionEvent`, come per ogni pulsante
 - un `ItemEvent`, gestito da un `ItemListener`
- Solitamente conviene gestire l'`ItemEvent`, perché più specifico.
- L'`ItemListener` dichiara il metodo:

```
public void itemStateChanged(ItemEvent e)
```

che deve essere implementato dalla classe che realizza l'ascoltatore degli eventi.

- *In caso di più caselle gestite dallo stesso listener*, il metodo `e.getItemSelectable()` restituisce un riferimento all'oggetto sorgente dell'evento.

ESEMPIO 13

- Un'applicazione comprendente una checkbox e un campo di testo (non modificabile), che riflette lo stato della checkbox



- Alla checkbox è associato un `ItemListener`, che intercetta gli eventi di selezione / deselegione implementando il metodo `itemStateChanged()`

```
class          Es13Panel                extends          JPanel
  implements ItemListener {

  JTextField txt; JCheckBox ck1;

  public Es13Panel(){
    super();
    txt = new JTextField(10);
    txt.setEditable(false);
    ck1 = new JCheckBox("Opzione");
    ck1.addItemListener(this);
    add(ck1); add(txt);
  }
  public void itemStateChanged(ItemEvent e){
    if (ck1.isSelected())
      txt.setText("Opzione attivata");
    else txt.setText("Opzione disattivata");
  }
}
```

ESEMPIO 14: PIÙ CASELLE DI OPZIONE

- Un'applicazione con due checkbox e un campo di testo che ne riflette lo stato



- Lo stesso `ItemListener` è associato a *entrambe* le checkbox: usa `e.getItemSelectable()` per dedurre quale casella è stata modificata

ESEMPIO 14

```
class          Es14Panel          extends          JPanel
  implements  ItemListener  {

  JTextField txt1, txt2;
  JCheckBox  c1, c2;

  public Es14Panel(){
    super();
    txt1 = new JTextField(15);
    txt1.setEditable(false);
    txt2 = new JTextField(15);
    txt2.setEditable(false);
    c1 = new JCheckBox("Mele");
    c1.addItemListener(this);
    c2 = new JCheckBox("Pere");
    c2.addItemListener(this);
    add(c1);    add(c2);
    add(txt1); add(txt2);
  }

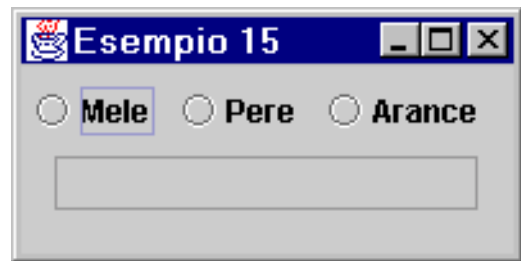
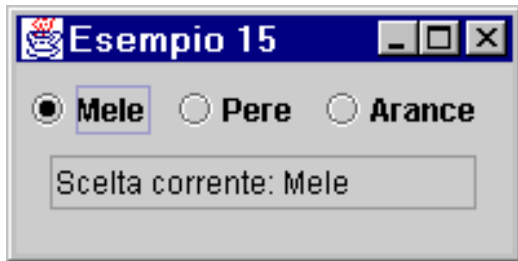
  public void itemStateChanged(ItemEvent e){
    Object source = e.getItemSelectable();
    if (source==c1)
      txt1.setText("Sono cambiate le mele");
    else
      txt1.setText("Sono cambiate le pere");
    // ora si controlla lo stato globale
    String frase = (ck1.isSelected() ? "Mele " : "")
      + (ck2.isSelected() ? "Pere" : "");
    txt2.setText(frase);
  }
}
```

IL RADIOBUTTON

- Il `JRadioButton` è una casella di opzione che fa parte di un gruppo: in ogni istante può essere attiva una sola casella del gruppo
- Quando si cambia la casella selezionata, si generano *tre* eventi
 - un `ItemEvent` per la casella deselezionata, uno per la casella selezionata, e un `ActionEvent` da parte della casella selezionata (pulsante premuto)
- In pratica:
 - si creano i `JRadioButton` che servono
 - si crea un oggetto `ButtonGroup` e si aggiungono i `JRadioButton` al gruppo

ESEMPIO 15

- Un'applicazione comprendente un gruppo di tre radiobutton, con un campo di testo che ne riflette lo stato



- Solitamente conviene gestire l'ActionEvent (più che l'ItemEvent) perché ogni cambio di selezione ne genera uno solo (a fronte di *due* ItemEvent), il che semplifica la gestione.

ESEMPIO 15

```
class      Es15Panel      extends      JPanel
  implements ActionListener {

  JTextField  txt;
  JRadioButton b1, b2, b3;      ButtonGroup  grp;

  public Es15Panel(){
    super();
    txt = new JTextField(15); txt.setEditable(false);
    b1  = new JRadioButton("Mele");
    b2  = new JRadioButton("Pere");
    b3  = new JRadioButton("Arance");
    grp = new ButtonGroup();
    grp.add(b1);  grp.add(b2);  grp.add(b3);
    b1.addActionListener(this); add(b1);
    b2.addActionListener(this); add(b2);
    b3.addActionListener(this); add(b3);
    add(txt);
  }

  public void actionPerformed(ActionEvent e){
    String scelta = e.getActionCommand();
    txt.setText("Scelta corrente: " + scelta);
  }
}
```

LA LISTA JList

- Una `JList` è una *lista di valori* fra cui si può sceglierne uno o più
- Quando si sceglie una voce si genera un evento `ListSelectionEvent`, gestito da un `ListSelectionListener`
- Il listener deve implementare il metodo `void valueChanged(ListSelectionEvent)`
- Per recuperare la/e voce/i scelta/e si usano `getSelectedValue()` e `getSelectedValues()`

ESEMPIO 16

- Un'applicazione con una lista e un campo di testo che riflette la selezione corrente



- Per intercettare le selezioni occorre gestire il `ListSelectionEvent`
- Di norma, `JList` non mostra una barra di scorrimento verticale: se la si vuole, va aggiunta a parte

ESEMPIO 16

Il codice:

```
class      Es16Panel      extends      JPanel
  implements ListSelectionListener {
  JTextField txt;  JList list;
  public Es16Panel(){
    super();
    txt = new JTextField(15);
    txt.setEditable(false);
    String voci[]={"Rosso", "Giallo", "Verde", "Blu"};
    list = new JList(voci);
    list.addListSelectionListener(this);
    add(list); add(txt);
  }

  public void valueChanged(ListSelectionEvent e){
    String scelta = (String) list.getSelectedValue();
    txt.setText("Scelta corrente: " + scelta);
  }
}
```

ESEMPIO 16: VARIANTE

Con gli usuali tasti SHIFT e CTRL, sono possibili anche *selezioni multiple*:

- con SHIFT si selezionano tutte le voci comprese fra due estremi, con CTRL si selezionano voci sparse
- `getSelectedValue()` restituisce solo la prima, per averle tutte occorre `getSelectedValues()`

Per gestire le selezioni multiple basta cambiare l'implementazione di `valueChanged()`:

```
public void valueChanged(ListSelectionEvent e){
    Object[] scelte = list.getSelectedValues();
    StringBuffer s = new StringBuffer();
    for (int i=0; i<scelte.length; i++)
        s.append((String)scelte[i] + " ");
    txt.setText("Scelte: " + s);
}
```

ESEMPIO 16: ULTERIORE VARIANTE

Per aggiungere una barra di scorrimento, si sfrutta un `JScrollPane`, e si fissa un numero massimo di elementi visualizzabili per la lista:



```
public Es18Panel(){
    ...
    list = new JList(voci);
    JScrollPane pane = new JScrollPane(list);
    list.setVisibleRowCount(3);
    list.addListSelectionListener(this);
    add(pane); // invece che add(list)
    add(txt);
}
```

LA CASELLA COMBINATA

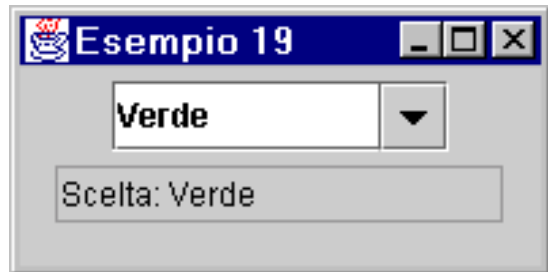
- Una `JComboBox` è una *lista di valori a discesa*, in cui si può o sceglierne uno, o scrivere un valore diverso
 - combina il campo di testo con la lista di valori



- Per configurare l'elenco delle voci proposte, si usa il metodo `addItem()`
- Per recuperare la voce scelta o scritta, si usa `getSelectedItem()`
- Quando si sceglie una voce o se ne scrive una nuova, si genera un `ActionEvent`

ESEMPIO 19

- Un'applicazione con una casella combinata e un campo di testo che riflette la selezione



- Ponendo `setEditable(true)`, si può anche scrivere un valore diverso da quelli proposti:



ESEMPIO 19: codice

```
class Es19Panel extends JPanel implements
  ActionListener {
  JTextField txt; JComboBox list;
  public Es19Panel(){
    super();
    txt = new JTextField(15);
    txt.setEditable(false);
    list = new JComboBox();
    list.setEditable(true);
    // per poter aggiungere nuove voci!

    list.addItem("Rosso"); list.addItem("Giallo");
    list.addItem("Verde"); list.addItem("Blu");
    list.addActionListener(this);
    add(list);
    add(txt);
  }
}
```

La gestione dell'evento:

```
public void actionPerformed(ActionEvent e){
  String scelta = (String) list.getSelectedItem();
  // recupera la voce selezionata o scritta
  // dall'utente
  txt.setText("Scelta: " + scelta);
}
```

LA GESTIONE DEL LAYOUT

- Quando si aggiungono componenti a un contenitore (in particolare: a un pannello), *la loro posizione è decisa dal Gestore di Layout (Layout Manager)*
- Il gestore predefinito *per un pannello* è `FlowLayout`, che dispone i componenti in fila (da sinistra a destra e dall'alto in basso)
 - semplice, ma non sempre esteticamente efficace
- Esistono comunque altri gestori alternativi, più o meno complessi.

LAYOUT MANAGER

Oltre a `FlowLayout`, vi sono:

- `BorderLayout`, che dispone i componenti lungo i bordi (nord, sud, ovest, est) o al centro
- `GridLayout`, che dispone i componenti in una griglia $m \times n$
- `GridBagLayout`, che dispone i componenti in una griglia $m \times n$ *flessibile*
 - righe e colonne a dimensione variabile
 - molto flessibile e potente, ma difficile da usare
- `BoxLayout`, che dispone i componenti o in orizzontale o in verticale, in un'unica casella (layout predefinito per il componente `Box`)
- *nessun layout manager*
 - si specifica la posizione assoluta (x,y) del componente
 - sconsigliato perché *dipendente dalla piattaforma*

Per cambiare Layout Manager:

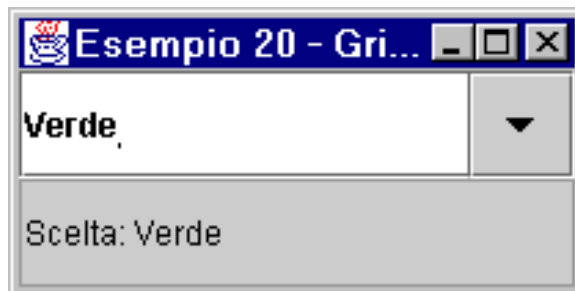
```
setLayout(new GridLayout(4,5))
```

LO STESSO PANNELLO CON...

FlowLayout



GridLayout
(griglia 2 x 1)



BorderLayout
(nord e sud)



Senza alcun layout
(posizioni a piacere)



PROGETTARE UN'INTERFACCIA

- Spesso, per creare un'interfaccia grafica completa, efficace e gradevole *non basta un singolo gestore di layout*
- Approccio tipico:
 - 1) *suddividere l'area in zone*, corrispondenti ad altrettanti pannelli
 - 2) applicare a ogni zona il layout manager più opportuno