# Java e la grafica

L'architettura Java supporta direttamente il concetto di applicazione grafica.

- package grafico `java.awt`: primo package grafico, non completamente indipendente dalla piattaforma.
- nuovo package grafico `java.swing`: scritto in Java e realmente indipendente dalla piattaforma.

# Java e la grafica

La trattazione seguita in questi lucidi attinge dalle lezioni del Prof. Denti della Facoltà di Ingegneria dell'Università di Bologna (lucidi disponibili su web) e dal libro *Programmazione a oggetti in Java*, Cabri-Zambonelli, Pitagora editrice.

Altri riferimenti (oltre al libro di testo):

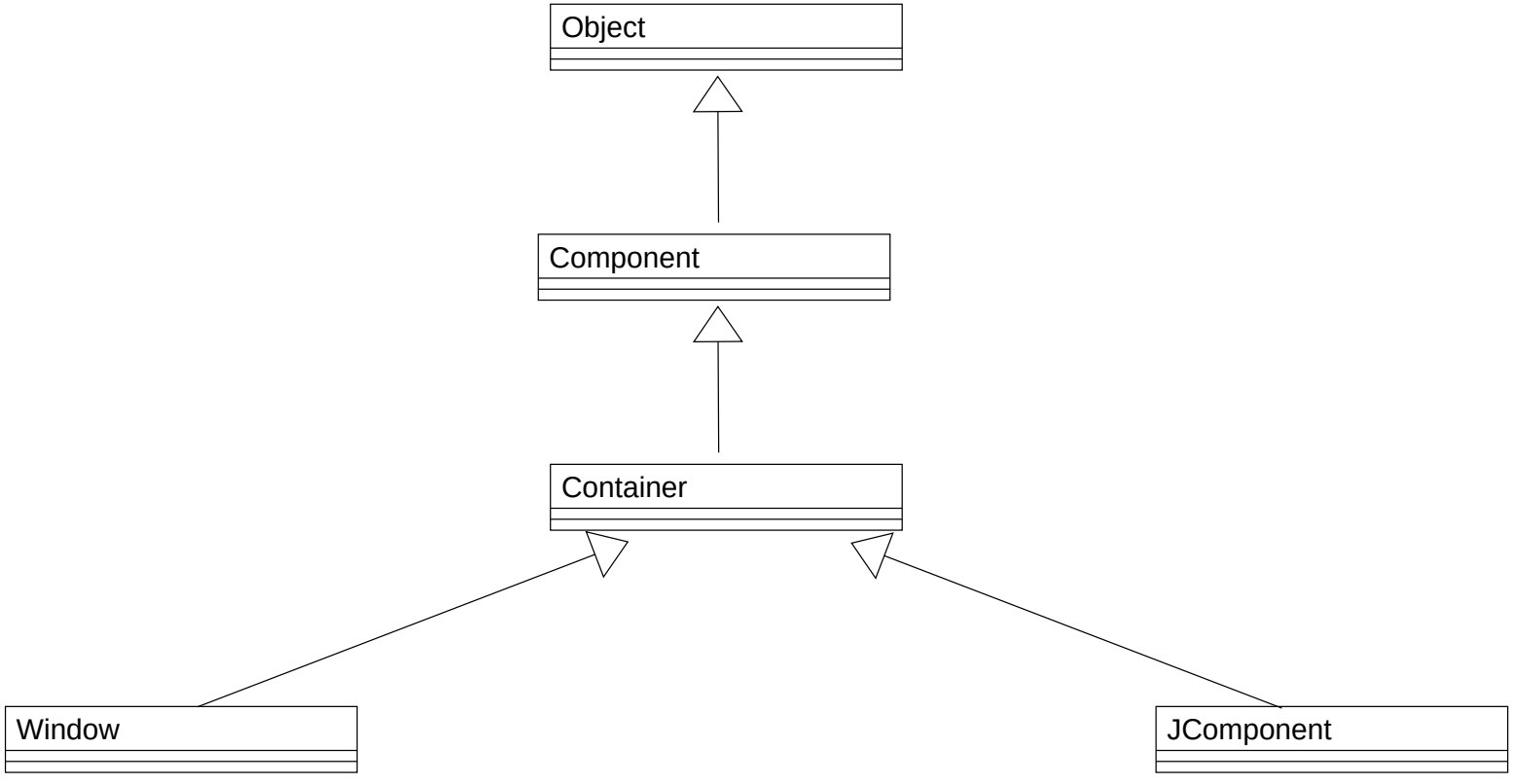
- *Thinking in Java*, Bruce Eckel (libro disponibile anche in versione elettronica gratuita).
- Documentazione della Sun (`java.sun.com`), in particolare, è disponibile un tutorial su swing.

# Swing: Architettura

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

# Swing: gerarchia di classi

Le classi il cui nome inizia con ’J’ sono del package swing, mentre le altre sono le classi già preesistenti nel package awt.

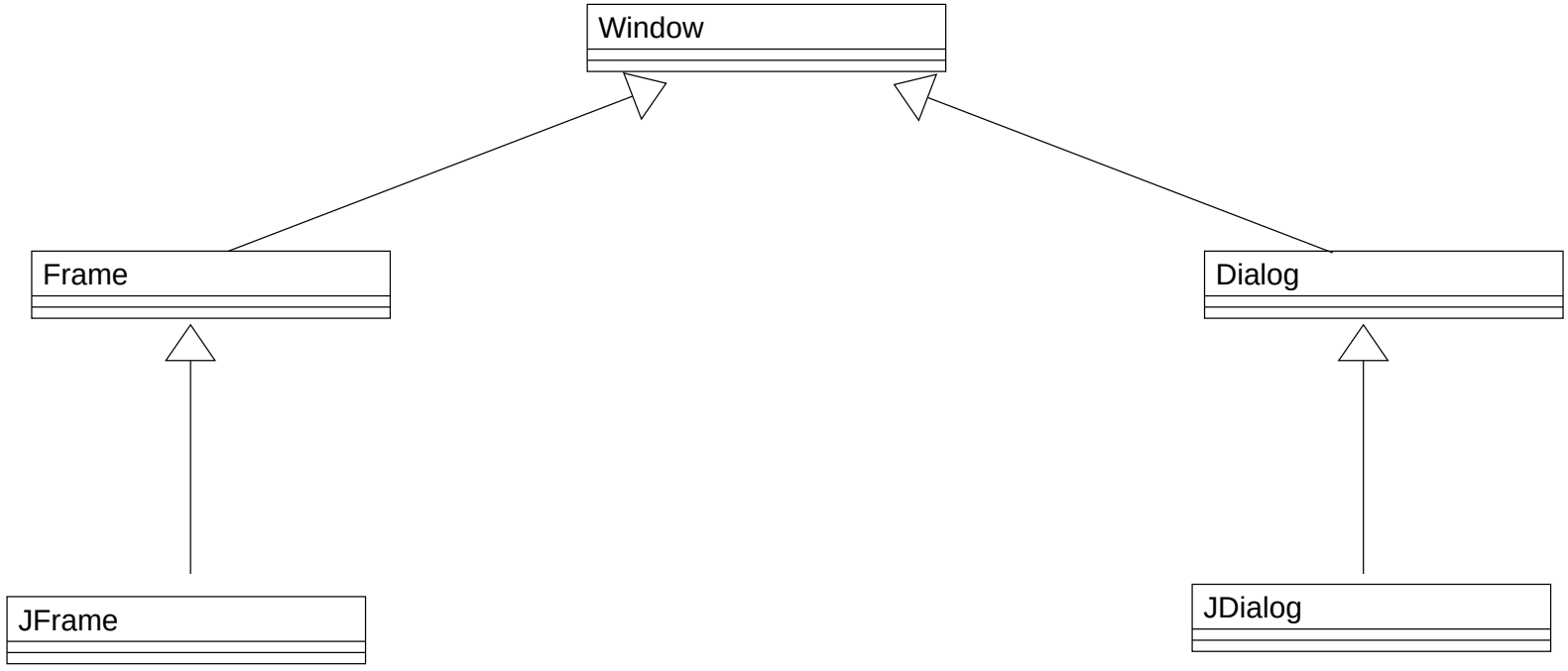


# Swing: gerarchia di classi

- Tutti i componenti principali sono *contenitori* e possono contenere altri componenti
- Le finestre sono casi particolari di contenitori e si distinguono in Frame e Finestre di Dialogo
- Gli oggetti della classe `JComponent` e delle sue sottoclassi sono componenti grafici tipici (bottoni, liste, menu,...).

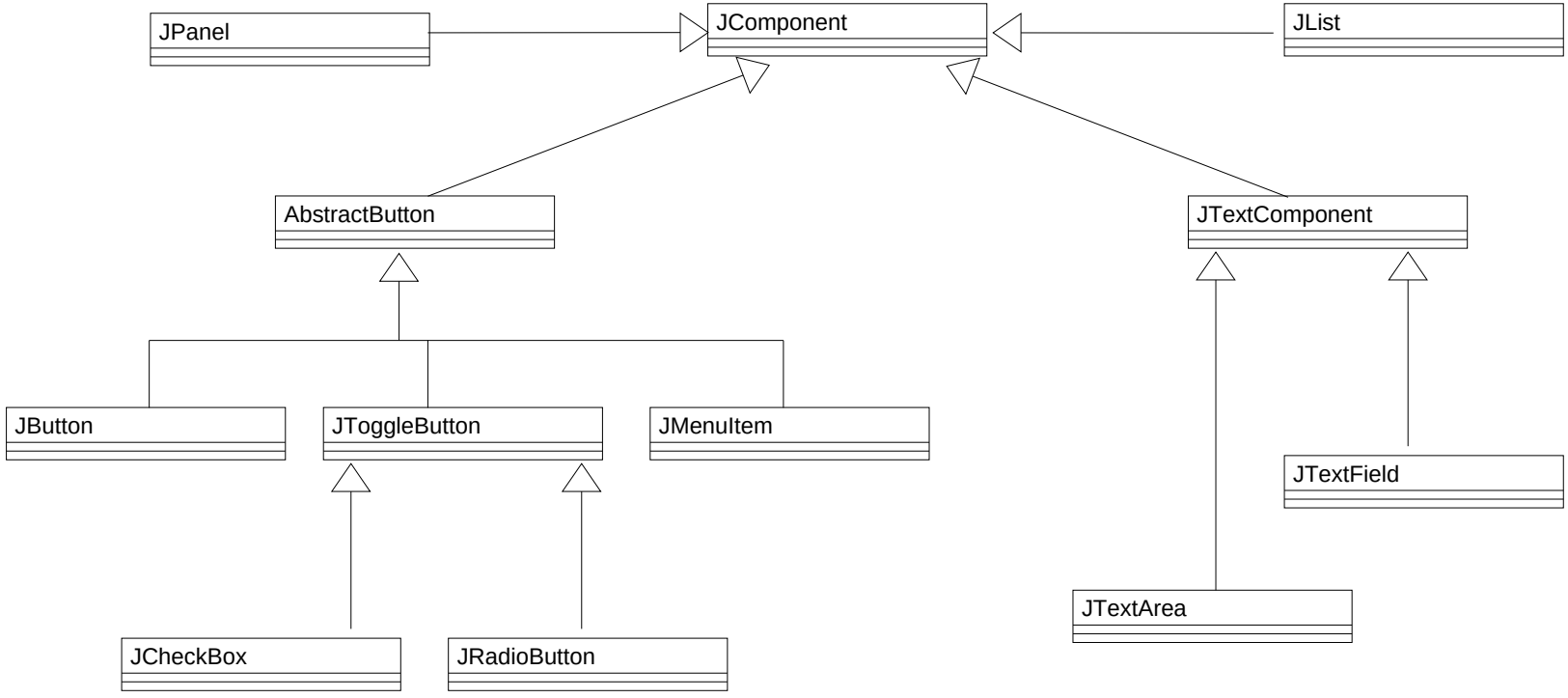
# Swing: gerarchia di classi

Un ramo della gerarchia è dedicato a oggetti “finestra”



# Swing: gerarchia di classi

L'altro ramo della gerarchia è dedicato a componenti grafici veri e propri



# Contenitori e pannelli

- Il frame non può essere utilizzato direttamente come contenitore nel quale scrivere e disegnare.
- Per aggiungere elementi ad una finestra dobbiamo utilizzare un oggetto della classe `Container`.
- Pensiamo al `Container` come ad una parete grezza, non ancora intonacata
- ed usiamo oggetti pannello (`JPanel`) come elementi per decorare e abbellire la parete.

# Contenitori e pannelli

In sintesi e “praticamente”:

1. Creiamo un frame (`JFrame`) che funge da struttura di base per la finestra. Viene automaticamente creato un oggetto `Container`.
2. Recuperiamo il riferimento al `Container` al quale poi aggiungeremo le componenti grafiche (`JPanel`).
3. Usiamo oggetti della classe `JPanel` o sue derivate come “affreschi” nei quali mettere oggetti grafici.

# Contenitori e pannelli

Schema di base:

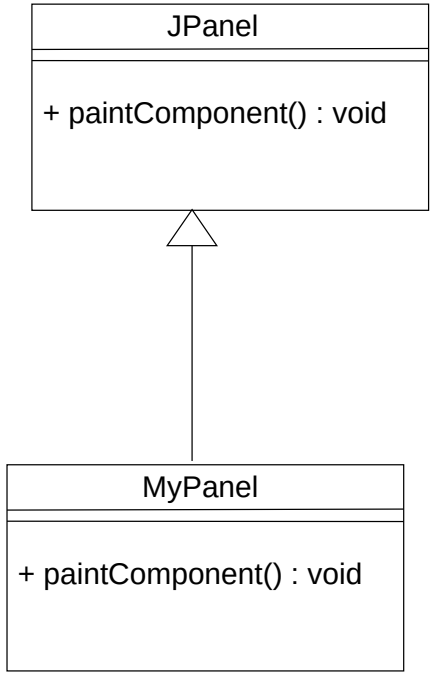
```
JFrame jf = new JFrame(‘‘Finestra’’);
// si crea il frame

Container cjf = jf.getContentPane();
// si recupera il riferimento
// al contenitore

JPanel p = new JPanel();
cjf.add(p);
// si crea il pannello
// e lo si ‘‘attacca’’ al frame
```

# Pannelli

Per disegnare su un pannello, si deriva una classe da `JPanel` e si ridefinisce il metodo `paintComponent()`, che viene invocato **automaticamente**.



# Pannelli

- Il metodo `paintComponent()` accetta un parametro di tipo `Graphics`: `paintComponent(Graphics g)`.
- L'oggetto della classe `Graphics` rappresenta il "pittore" al quale chiediamo servizi di disegno (per esempio, tracciamento di linee e forme geometriche): `g.dipingiQualcosa()` (fare riferimento alla documentazione della Sun per i metodi a disposizione).
- **IMPORTANTE**: Il metodo `paintComponent()` deve, PER PRIMA COSA, invocare il metodo `paintComponent()` della classe genitore tramite `super.paintComponent()`.

# Pannello e componenti

- Oltre che disegni, ad un pannello si possono aggiungere altri componenti, la maggior parte dei quali sono *attivi*, cioè generano eventi quando manipolati.
- I componenti sono oggetti della classe `JComponent` e sue sottoclassi.
- Anche `JPanel` deriva da `JComponent`, quindi è possibile suddividere un pannello in “sotto-pannelli”, aggiungendoli come normali componenti.
- Esempi di componenti: bottoni di vario tipo, liste, menu, ecc.

# Pannello e componenti

In pratica:

1. si crea l'oggetto della classe del componente desiderato
2. si invoca il metodo `add( )` del pannello passando come argomento il riferimento al componente da aggiungere.

# Componente JLabel

- Il componente della classe `JLabel` ha l'unico scopo di contenere una scritta.
- Rappresenta un rettangolino, all'interno del quale vi è la stringa inserita, oppure un'immagine.

# JLabel con testo

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

public class PanelLabel extends JPanel {
  JLabel jl;
  // riferimento come attributo

  public PanelLabel(){
    super();
    jl = new JLabel("Sono una bella etichetta!"
    // si crea la label

    add(jl);
    // e la si aggiunge al pannello
    }}
```

# JLabel con testo

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

public class EsLabel {
  public static void main(String[] v){
    JFrame f = new JFrame("Esempio di Label");
    Container c = f.getContentPane();
    PanelLabel panel = new PanelLabel();
    c.add(panel);
    f.setDefaultCloseOperation(JFrame.EXIT_ON_C
    f.pack();
    f.setVisible(true); //sostituisce f.show()
  }
}
```

# JLabel con icona grafica

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

public class ImgPanel extends JPanel {
  JLabel lb2;

  public ImgPanel(){
  super();
  JLabel lb2 = new JLabel(new ImageIcon("img.gif"));
  add(lb2);
  }
}
```

# Esempio con icona grafica

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

public class EsLabel2 {
  public static void main(String[] v){
    JFrame f = new JFrame("Label con grafica");
    Container c = f.getContentPane();
    ImgPanel panel = new ImgPanel();
    c.add(panel);
    f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    f.pack();
    f.setVisible(true);
  }
}
```

# Componenti attivi

- I componenti attivi permettono l'interazione con l'utente.
- Ogni componente attivo, quando l'utente opera su di esso, genera un **evento** che descrive ciò che è avvenuto.
- In generale, ogni componente può generare diversi tipi di evento.
- Si passa allo scenario **model/view/controller**.

# read/eval/print

Scenario “usuale” di computazione:

1. READ: acquisizione dati in ingresso (interazione con l’utente)
2. EVAL: computazione (nessuna interazione)
3. PRINT: emissione risultati in uscita (interazione con l’utente)

# Programmazione ad eventi

Le interfacce grafiche hanno modificato radicalmente lo schema read/eval/print, perché permettono all'utente di interagire durante l'elaborazione e di determinarne il flusso in modo non prevedibile a priori.

Si svolgono azioni non più in conseguenza del proprio flusso di controllo (interno), ma in risposta ad eventi generati dall'esterno.

L'applicazione non ha più un ordine preciso di esecuzione, ma è composta da una collezione di procedure, ognuna delle quali deve essere eseguita in corrispondenza a uno specifico evento di interazione.

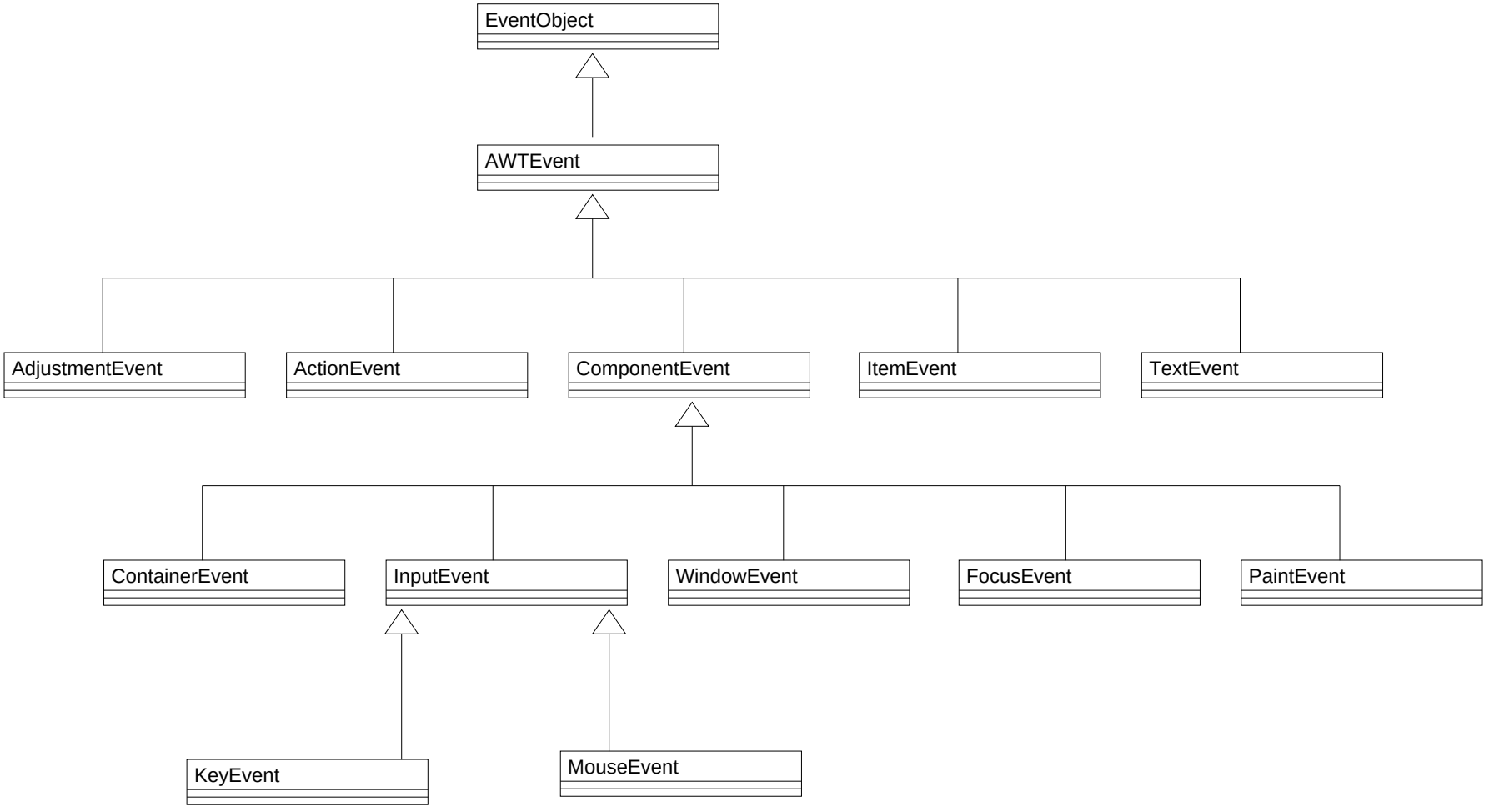
# model/view/controller

- MODEL: rappresenta la struttura dei dati nell'applicazione e le relative operazioni.
- VIEW: presenta i dati all'utente in qualche forma. Possono esserci più viste qualora sia utile presentare i dati in modi diversi (es: testo, e html).
- CONTROLLER: reagisce alle azioni dell'utente in maniera analoga agli interrupt hardware. Recupera le informazioni relative all'interazione, chiama i metodi opportuni e richiede la vista appropriata.

# Gestione degli eventi

- All’atto di un’interazione con un componente grafico, la Java Virtual Machine crea automaticamente e implicitamente un oggetto.
- Tale oggetto è istanza di una specifica classe di eventi e ha lo scopo di descrivere le proprietà dell’evento stesso.
- Esistono diverse classi di eventi, una per ogni tipologia di interazione con componenti grafici.

# Classi di eventi (vista parziale)



# Gestione degli eventi

- Una volta generato l'oggetto evento, questo viene inviato ad un oggetto **ascoltatore degli eventi** (**event listener**).
- L'event listener deve essere definito e creato da noi e deve essere associato al componente attivo, cosicché quando si genera un evento, la JVM sappia a chi inviare l'oggetto evento.
- L'event listener gestisce l'evento mediante un opportuno metodo, che non è altro che l'implementazione di un particolare metodo di un'interfaccia associata a tali tipi di eventi.

# JButton

- Quando è premuto, un bottone genera un evento della classe `ActionEvent`.
- Questo evento è inviato dal sistema allo specifico ascoltatore di eventi associato a quel bottone.
- Tale ascoltatore di eventi deve realizzare l'interfaccia `ActionListener`, e cioè implementare il metodo `actionPerformed(ActionEvent e)`.

# Esempio di uso di JButton

- Progettiamo un’applicazione fatta da un’etichetta (JLabel) e un bottone (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”.

# Esempio di uso di JButton

Architettura dell'applicazione:

- Un pannello che contiene etichetta e bottone $\rightarrow$ il costruttore del pannello crea l'etichetta e il bottone.
- Il pannello fa da ascoltatore degli eventi per il pulsante $\rightarrow$ il costruttore del pannello imposta il pannello stesso come ascoltatore degli eventi del pulsante (quindi il pannello dovrà implementare l'interfaccia `ActionListener`).

# Esempio di uso di JButton

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

public class Es8Panel extends JPanel
  implements ActionListener {
  // il pannello implementa l'interfaccia
  // ActionListener

  private JLabel l;
  private JButton b;

// ...
```

# Esempio di uso di JButton

```
// ...
  public Es8Panel(){
    super();
    l = new JLabel("Tizio");
    add(l);
    b = new JButton("Tizio/Caio");
    add(b);
    // il costruttore crea etichetta
    // e bottone e li aggiunge al pannello

    b.addActionListener(this);
    // associa il pannello al bottone
    // come event listener
  }
// ...
```

# Esempio di uso di JButton

```
// ...

// implementa il metodo actionPerformed()
// dell'interfaccia ActionListener

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

# Esempio di uso di JButton

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

public class EsSwing8 {
    public static void main(String[] v){
JFrame f = new JFrame("Esempio JButton");
Container c = f.getContentPane();
Es8Panel panel = new Es8Panel();
c.add(panel);
f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE
f.pack();
f.setVisible(true);
    }
}
```

# Variante

- Un pannello che contiene etichetta e bottone $\rightarrow$ il costruttore del pannello crea l'etichetta e il pulsante.
- L'ascoltatore di eventi per il pulsante è un oggetto separato $\rightarrow$ il costruttore del pannello imposta tale oggetto come ascoltatore degli eventi del pulsante.

# Variante

```
public class Es8bisPanel extends JPanel{
public Es8bisPanel(){
    super();
    JLabel l = new JLabel("Tizio");
    add(l);
    JButton b = new JButton("Tizio/Caio");
    b.addActionListener(new Es8Listener(l));
    add(b);
}
}
```

# Variante

```
public 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;}
}
```

# Esempio con due pulsanti

Scopo dell'applicazione: cambiare lo sfondo tramite due pulsanti; uno lo rende rosso, l'altro blu.

- Un pannello che contiene i due pulsanti creati dal costruttore del pannello.
- Due ascoltatori separati, uno per ciascun pulsante. Sarebbe possibile anche usare un solo ascoltatore, differenziando le azioni in base al pulsante premuto.

# Classe pannello

```
public class Es9Panel extends JPanel{
  public Es9Panel(){
  super();
  JButton b1 = new JButton("Rosso");
  JButton b2 = new JButton("Blu");
  b1.addActionListener(new Es9Listener(this,Color.red));
  b2.addActionListener(new Es9Listener(this,Color.blue));
  add(b1);
  add(b2);
  }
}
```

# Classe event listener

```
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);
  }
}
```

# Classe con main

```
public class EsSwing9 {
  public static void main(String[] v){
    JFrame f = new JFrame("Due bottoni");
    Container c = f.getContentPane();
    Es9Panel panel = new Es9Panel();
    c.add(panel);
    f.setDefaultCloseOperation(JFrame.EXIT_ON_C
    f.pack();
    f.setVisible(true);
  }
}
```

# Variante

- Usiamo un solo ascoltatore degli eventi; è quindi necessario discriminare tra i due pulsanti
- si usa in metodo `getSource()`, invocato sull'oggetto della classe `ActionEvent` che è automaticamente passato al metodo `actionPerformed`.
- Per semplicita', supponiamo che sia il pannello a fare da ascoltatore degli eventi.

# Classe pannello

```
public class Es9bisPanel extends JPanel
    implements ActionListener{
    JButton b1,b2;
        public Es9bisPanel(){
            super();
            b1 = new JButton("Rosso");
            b2 = new JButton("Blu");
            b1.addActionListener(this);
            b2.addActionListener(this);
            add(b1);
            add(b2);}
    public void actionPerformed(ActionEvent e){
        if (e.getSource() == b1) setBackground(Color.red);
        else setBackground(Color.blue);
    }
}
```

# Il main

```
public class EsSwing9bis {
    public static void main(String[] v){
        JFrame f = new JFrame("Due bottoni");
        Container c = f.getContentPane();
        Es9bisPanel panel = new Es9bisPanel();
        c.add(panel);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.pack();
        f.setVisible(true);
    }
}
```

# Eventi di finestra

- Le operazioni eseguite sulle finestre (finestra aperta, chiusa, minimizzata, ingrandita,...) generano un oggetto della classe `WindowEvent`.
- Gli eventi di finestra sono gestiti dalla JVM, che attua comportamenti predefiniti e irrevocabili.
- In più, il sistema invoca i metodi dichiarati dall'interfaccia `WindowListener`.

# Metodi di WindowListener

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

# Chiusura della finestra

Per far sì che chiudendo la finestra del frame l'applicazione termini, è necessario implementare l'interfaccia `WindowListener`. In particolare si deve ridefinire `WindowClosing` in modo che invochi `System.exit()`.

Poiché implementare un'interfaccia implica implementare tutti i metodi da essa dichiarati, occorre formalmente implementarli tutti.

Non dovendo aggiungere altri comportamenti specifici oltre a quello di chiusura, basta definire gli altri col corpo vuoto.

# Chiusura della finestra

Definiamo una nostra classe che funga da event listener per l'evento di chiusura finestra.

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


class Terminator implements WindowListener {
  public void windowClosed(WindowEvent e){}
  public void windowClosing(WindowEvent e){
    System.exit(0);}
  public void windowIconified(WindowEvent e){}
  public void windowDeiconified(WindowEvent e){}
  public void windowActivated(WindowEvent e){}
  public void windowDeactivated(WindowEvent e){}
  public void windowOpened(WindowEvent e){}
}
```

# Componente JTextField

Il JTextField è un componente “campo di testo”, utilizzabile per scrivere e visualizzare una riga di testo.

- il campo di testo può esere editabile o no
- si può accedere al testo con `getText()` e `setText()`

# Componente JTextField

- Ogni volta che il testo cambia si genera un `DocumentEvent`
- Se però è sufficiente registrare i cambiamenti solo quando si preme un bottone o il tasto INVIO, basta gestire il solito `ActionEvent`.

# Esempio

- Sviluppiamo un’applicazione comprendente un pulsante e due campi di testo: uno per scriverlo, 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.
- Impostiamo il pannello come ascoltatore degli eventi.

# Il main

```
public class EsSwing10 {
  public static void main(String[] v){
    JFrame f = new JFrame("Esempio");
    Container c = f.getContentPane();
    Es10Panel p = new Es10Panel();
    c.add(p);
    f.addWindowListener( new Terminator() );
    f.pack();
    f.setVisible(true);
  }
}
```

# Il pannello

```
class Es10Panel extends JPanel
  implements ActionListener {

  JButton b;
  JTextField txt1,txt2;

  public Es10Panel(){
    super();
    b = new JButton("Aggiorna");
    txt1 = new JTextField("Scrivere il testo qui",25);
    txt2 = new JTextField(25);
    txt2.setEditable(false);
    b.addActionListener(this);
    add(txt1);add(txt2);add(b);
  }
// ...
```

# Il pannello

```
// ...

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

# Variante

Ora eliminiamo il bottone e associamo l'esecuzione della procedura di aggiornamento all'evento generato dalla pressione del tasto INVIO

```
class Es11Panel extends JPanel
  implements ActionListener {
  JTextField txt1,txt2;
  public Es11Panel(){
    super();
    txt1 = new JTextField("Scrivere il testo qui",25);
    txt2 = new JTextField(25);
    txt2.setEditable(false);
    txt1.addActionListener(this);
    add(txt1);add(txt2);
  }
  public void actionPerformed(ActionEvent e){
    txt2.setText(txt1.getText());}}
```

# Componente JCheckBox

- Il componente `JCheckBox` è una casella di opzione che può essere selezionata/deselezionata.
- Lo stato è verificabile tramite `isSelected()` ed è 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 è più specifico.

# Componente JCheckBox

- L'`ItemListener` dichiara il metodo `itemStateChanged()` che deve essere implementato dalla classe che realizza l'ascoltatore di eventi.
- In caso di più caselle gestite dallo stesso listener, il metodo `getItemSelectable()` restituisce un riferimento all'oggetto sorgente dell'evento.

# Esempio

- Progettiamo un’applicazione nella quale l’utente può selezionare una o più band.
- In un campo di testo si indica il numero di elementi correntemente selezionati.
- Il pannello è l’ascoltatore degli eventi.
- Non avremo bisogno di sapere quale checkbox è selezionata (quindi non useremo getItemSelectable()), ma solo il numero di quelle selezionate in ogni momento.

# Il main

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

public class EsBand1 {
  public static void main(String[] v){
    JFrame f = new JFrame("Seleziona Band");
    Container c = f.getContentPane();
    MolteCaselle p = new MolteCaselle();
    c.add(p);
    f.setDefaultCloseOperation(JFrame.EXIT_ON_C
    f.pack(); f.setVisible(true);
  }
}
```

# Il pannello

```
class MolteCaselle extends JPanel
  implements ItemListener{

  JTextField txt,txa;
  // due campi di testo
  // come titolo e report

  JCheckBox ck[];
  // array di checkbox

//...
```

# Il pannello

```
// ...
  public MolteCaselle(){
    super();
    txt = new JTextField("Seleziona",15);
    txa = new JTextField(25);
    txt.setEditable(false);
    txa.setEditable(false);
    ck = new JCheckBox[5];

    ck[0] = new JCheckBox("Eric Clapton");
    ck[1] = new JCheckBox("Dire Straits");
    ck[2] = new JCheckBox("Aerosmith");
    ck[3] = new JCheckBox("Boston");
    ck[4] = new JCheckBox("Dave Matthews");
// ...
```

# Il pannello

```
// ...
    for (int i=0; i<5; i++)
      ck[i].addItemListener(this);
    // il pannello \‘e  associato
    // a tutte le checkbox

    setLayout(new GridLayout(7,1));
    // il layout del pannello
    // \‘e  una griglia di 7 righe
    // e 1 colonna

    add(txt);
    for (int i=0; i<5; i++) add(ck[i]);
    add(txa);
    // si aggiungono i componenti
  }
//
```

# Il pannello

```
// ...
  // Occorre implementare il metodo:
  // ad ogni modifica in una checkbox
  // si ricalcola il numero
  // di quelle selezionate

  public void itemStateChanged(ItemEvent e){
    int cont = 0;
    for (int i=0; i<5; i++)
      if (ck[i].isSelected()) cont++;
    txa.setText(cont+" elementi selezionati");
  }
}
```

# Componente JRadioButton

- I “bottoni radio” costituiscono ancora caselle di selezione.
- Possono essere resi parte di un **gruppo** in modo tale che soltanto un bottone sia selezionato in ogni momento.
- Così la selezione di un bottone causa la deselezione automatica di quello precedentemente selezionato.

# Componente JRadioButton

Per rendere i `JRadioButton` parte dello stesso gruppo è necessario:

1. Creare un oggetto della classe `ButtonGroup`
2. Aggiungere a tale oggetto tutti i singoli `JRadioButton` che si vuole facciano parte del gruppo.

```
//...
ButtonGroup bg = new ButtonGroup();
add(radiobutton1);
add(radiobutton2);
//...
```

# JRadioButton ed eventi

La selezione di un JRadioButton genera sempre almeno tre eventi:

- Un `ActionEvent` per la nuova casella selezionata.
- Due `ItemEvent`: uno per la casella selezionata e uno per la casella da deselezionare.

Niente paura: Come per gli oggetti `JCheckBox`, la gestione degli eventi può essere effettuata con un oggetto che implementi l'interfaccia `ItemListener`.

# Esempio

- Modifichiamo l’esempio precedente in cui erano selezionate delle band. In questo caso è consentita soltanto una selezione.
- Per quanto riguarda la gestione degli eventi, non lasciamo la gestione al pannello, ma creiamo un nuovo oggetto che implementi l’interfaccia `ItemListener`.

# Il main

```
public class EsBand2 {
  public static void main(String[] v){
  JFrame f = new JFrame("Seleziona Band");
  Container c = f.getContentPane();
  PannelloRadio p = new PannelloRadio();
  c.add(p);
  f.setDefaultCloseOperation(JFrame.EXIT_ON_CLO
  f.pack();
  f.setVisible(true);
  }
}
```

# Il pannello

```
class PannelloRadio extends JPanel{
  private JTextField txt,txa;
  private JRadioButton rb[];
  private ButtonGroup bg;

  public PannelloRadio(){
    super();
    RadioButtonListener rbl;
    txt = new JTextField("Seleziona",15);
    txa = new JTextField(25);
    txt.setEditable(false);
    txa.setEditable(false);
//...
```

# Il pannello

```
//...
    rb = new JRadioButton[5];
    rb[0] = new JRadioButton("Eric Clapton");
    rb[1] = new JRadioButton("Dire Straits");
    rb[2] = new JRadioButton("Aerosmith");
    rb[3] = new JRadioButton("Boston");
    rb[4] = new JRadioButton("Dave Matthews");

    bg = new ButtonGroup();
    // si crea l'oggetto
    // 'gruppo di bottoni'

    for (int i=0; i<5; i++) bg.add(rb[i]);
    // si aggiungono i bottoni al gruppo
//...
```

# Il pannello

```
//...
    rbl = new RadioButtonListener(rb,txa);
    // si crea l'oggetto ascoltatore
    // che deve avere i riferimenti
    // ai bottoni e il riferimento
    // al campo di testo inferiore

    for (int i=0; i<5; i++)
      rb[i].addItemListener(rbl);
    // si associa il listener ai bottoni

    setLayout(new GridLayout(7,1));
    add(txt);
    for (int i=0; i<5; i++) add(rb[i]);
    add(txa); }}
```

# Event listener

```
class RadioButtonListener implements ItemListener {
  private JRadioButton rb[];
  private JTextField txa;

  public RadioButtonListener(JRadioButton radioButton[],
                             JTextField t){
    rb = radioButton;
    txa = t;
  }
//...
```

# Event listener

```
//...
  public void itemStateChanged(ItemEvent e){
    int cont = 0;
    for (int i=0; i<rb.length; i++)
      if (rb[i].isSelected()) cont = i;
    // cerchiamo l'ordine dell'elemento
    cont++;
    // vogliamo che sia da 1 a 5 (non da 0 a 4)
    txa.setText("Selezionato elemento " + cont);
  }
}
```

# Componente JList

- Il componente `JList` serve per rappresentare una lista di elementi e selezionarne uno o più di uno.
- Un oggetto `JList` è creato passandogli come parametro una lista di stringhe che rappresentano gli elementi della lista.
- Per recuperare un singolo valore selezionato della lista si può usare il metodo `getSelectedValue()`.
- Per recuperare tutti gli elementi selezionati si può usare il metodo `getSelectedValues()` che restituisce un array di `Object` (da convertire poi in stringhe).

# JList ed eventi

- Ogni volta che si seleziona un elemento della lista, si genera un evento della classe `ListSelectionEvent`;
- tale evento dovrà essere gestito da un ascoltatore che implementi l'interfaccia `ListSelectionListener`;
- l'oggetto ascoltatore dovrà implementare il metodo `valueChanged`, che accetta come parametro un `ListSelectionEvent`.

# Esempio

- Modifichiamo l'esempio precedente in cui erano selezionate delle band. In questo caso, presentiamo una lista con l'elenco degli elementi selezionabili. L'utente può selezionare uno o più elementi (tenendo premuto il tasto CTRL).
- Per quanto riguarda la gestione degli eventi, non lasciamo la gestione al pannello, ma creiamo un nuovo oggetto che implementi l'interfaccia `ListSelectionListener`.
- Utilizziamo il componente `JTextArea` per mostrare tutti gli elementi selezionati.

# Il main

```
public class EsBand3 {
  public static void main(String[] v){
    JFrame f = new JFrame("Seleziona Band");
    Container c = f.getContentPane();
    PannelloLista p = new PannelloLista();
    c.add(p);
    f.setDefaultCloseOperation(JFrame.EXIT_ON_C
    f.pack();
    f.setVisible(true);
    }
}
```

# Il pannello

```
class PannelloLista extends JPanel{
  JTextField txt;
  JTextArea txa;
  JList lista;

  public PannelloLista(){
    super();
    MyListSelectionListener ll;
    txt = new JTextField("Seleziona",15);
    txa = new JTextArea("Nessuna selezione",7,20);
    txt.setEditable(false);
    txa.setEditable(false);
//...
```

# Il pannello

```
//...
    String band[]= {"Eric Clapton","Dire Straits",
                    "Aerosmith","Boston",
                    "Dave Matthews"};
    lista = new JList(band);
    // si crea la lista con gli elementi
    // contenuti nell'array di stringhe

    ll = new MyListSelectionListener(lista,txa);
    lista.addListSelectionListener(ll);

    add(txt); add(lista); add(txa);
  }
}
```

# Event listener

```
public class MyListSelectionListener
  implements ListSelectionListener {

  JList lista;
  JTextArea txt;

  public MyListSelectionListener(JList l,JTextArea t)
  {
    lista = l;
    txt = t;
  }
//...
```

# Event listener

```
//...
    public void valueChanged(ListSelectionEvent e){
      Object scelte[] = lista.getSelectedValues();
      // mettiamo in un array gli elem. selezionati

      StringBuffer s = new StringBuffer();
      // creiamo un buffer di stringhe
      // nel quale metteremo le stringhe
      // degli elementi selezionati

      for (int i=0; i<scelte.length; i++)
        s.append((String)scelte[i] + "\n");
      // notare il casting...
      txt.setText("Selezioni:\n" + s);}}
```