Il Linguaggio Java

Introduzione al package grafico Swing

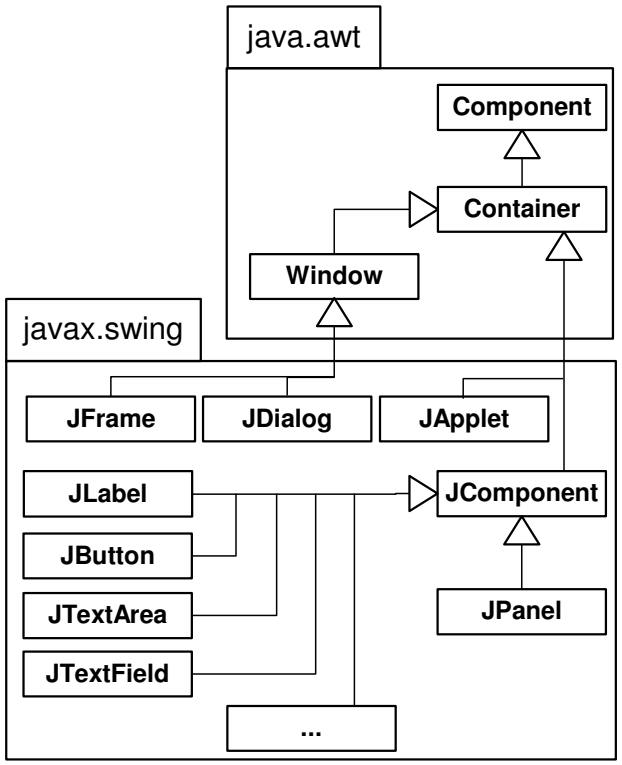
Architettura di javax.swing

• *Java* supporta direttamente nella propria architettura il concetto di applicazione grafica, tramite il package *javax.swing*, in cui i componenti vengono direttamente disegnati dalla JVM, e quindi sono indipendenti dalla piattaforma.

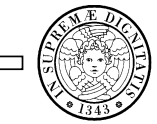
• Il primo package grafico (java.awt, 1996) fu invece implementato in codice nativo, per cui si appoggia alle chiamate del sistema operativo (la grafica dipende dalla piattaforma).

• Le classi grafiche di Swing *estendono* quelle di AWT, ed iniziano per "J" per distinguersi da queste ultime.

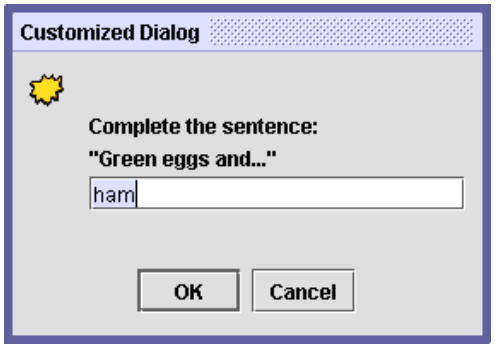
• Tutti i componenti grafici sono dei contenitori (*Container*), e possono contenere altri componenti.

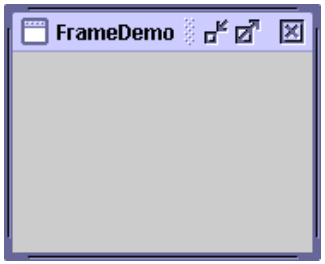


Frame, finestra di dialogo, applet

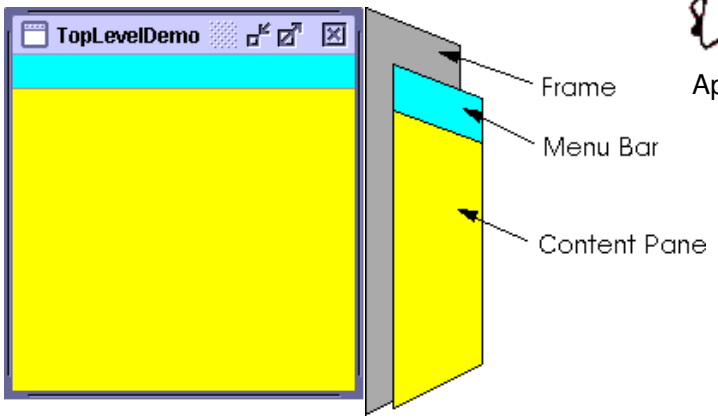


• Una interfaccia grafica “swing” é normalmente costruita con un *pannello principale*, un *pannello intermedio* e dei *componenti atomici*.





Applet



• Il pannello principale é una *frame* (JFrame), o una *finestra di dialogo* (JDialog), oppure una *applet* (JApplet).

• Il pannello intermedio é un pannello (JPanel), unico componente del pannello principale, ed adoperato come pannello di altri componenti atomici e/o di altri pannelli.

• I componenti atomici, visibili effettivamente sull’ interfaccia, sono: etichetta (JLabel), bottone (JButton), area di testo (JTextArea), campo di testo (JTextField), etc.

Frame, pannello e componenti

• Una finestra viene implementata con una classe che estende JFrame:

```
public class MiaInterfaccia extends JFrame {…
```

• Nella classe, si dichiarano i campi grafici della frame (i componenti atomici) ed altri campi dato:

```
private JButton[]   bottoni;
private JLabel[]    etichette;
private int         contatore;
```

• Nel costruttore, si invoca `super()`, si assegnano le proprietà della frame, si invoca un metodo `inizializzaGUI()`, e si inizializzano i campi dato.

```
public MiaInterfaccia() {
    super();
    setTitle("Titolo");
    ...
    inizializzaGUI();
    ...
    contatore = 0;
```

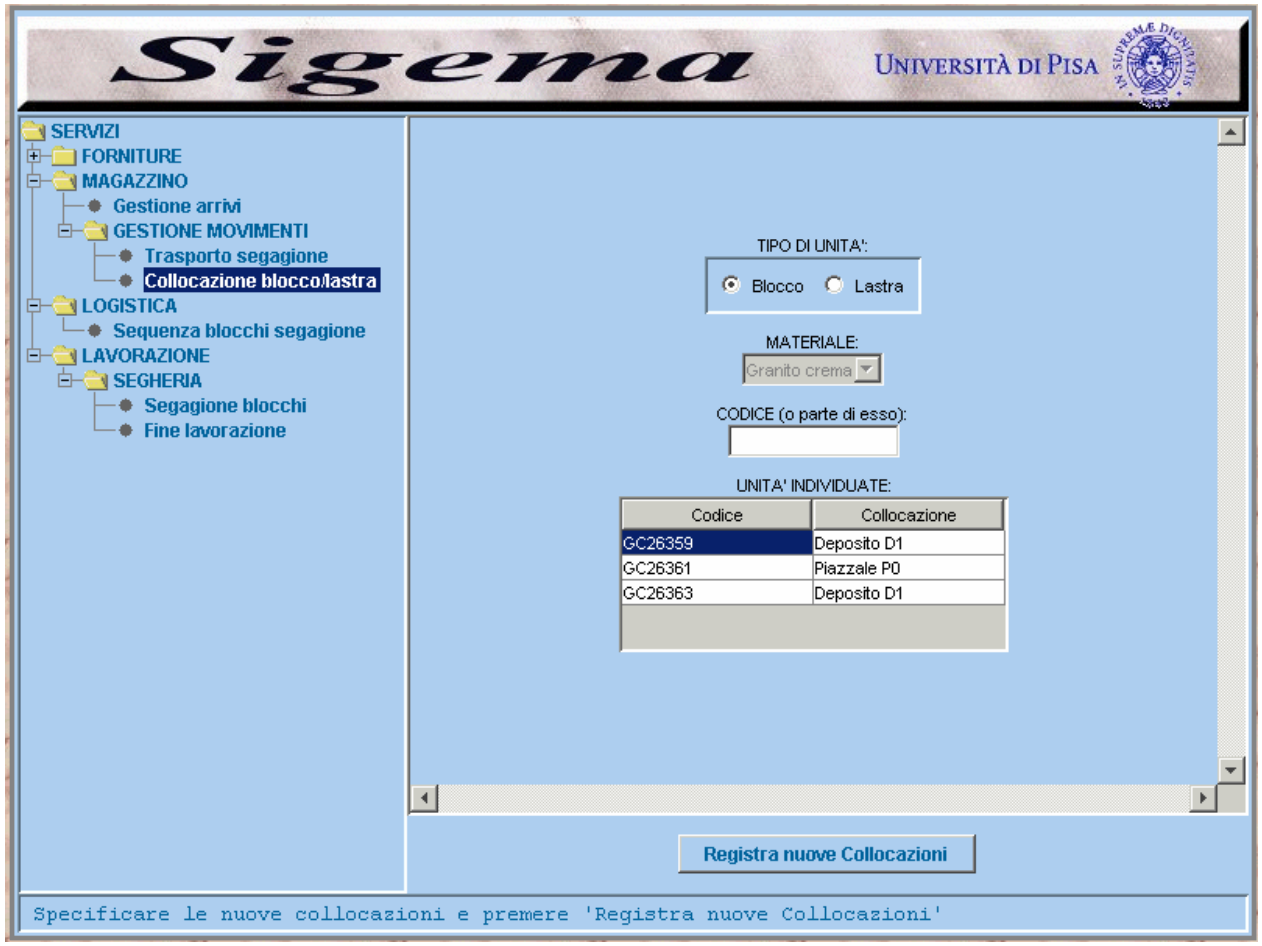
Frame, pannello e componenti

• Nel metodo `inizializzaGUI()`, si crea il pannello intermedio, ossia un pannello (1), e le istanze dei componenti (3); infine si 'attaccano' i componenti al pannello (4), ed il pannello alla frame (5):

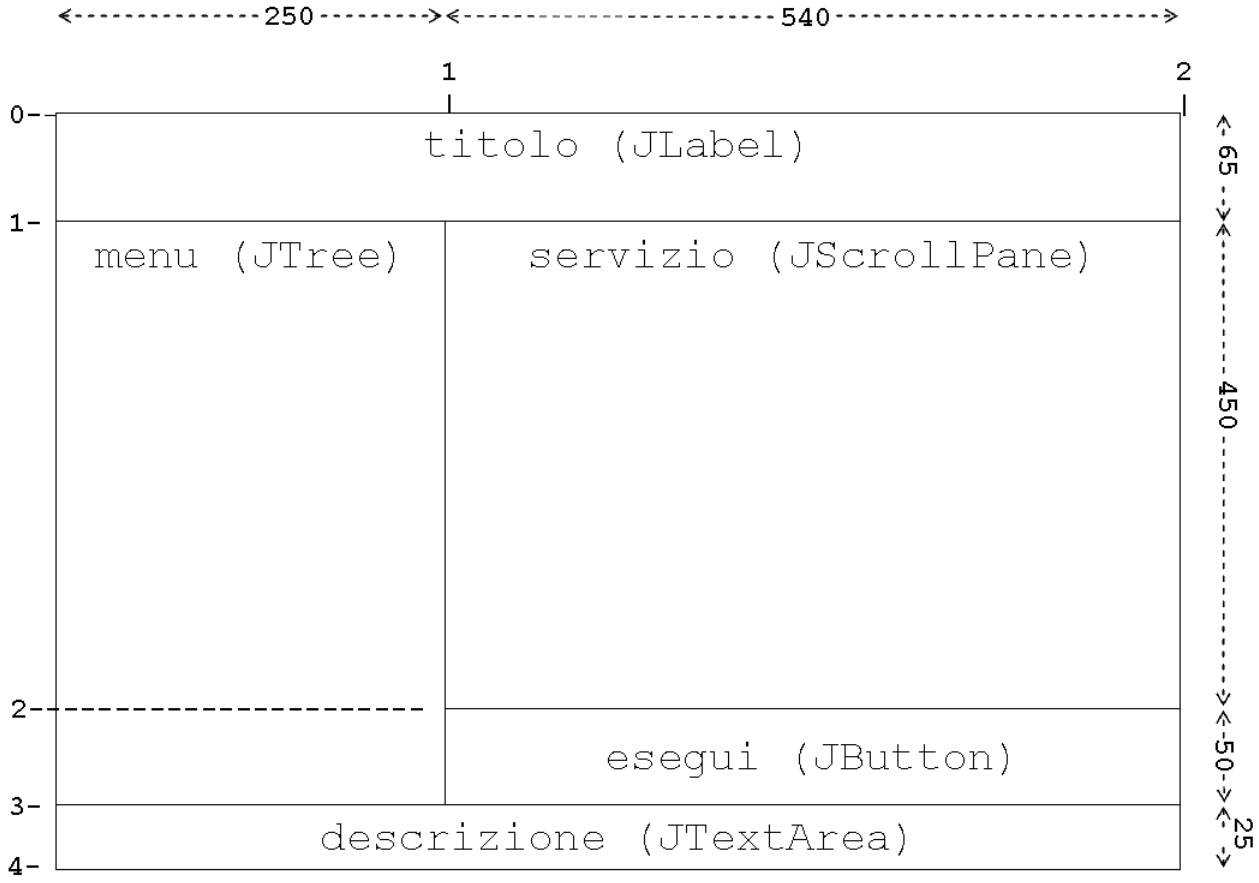
• Quando si aggiungono componenti ad un pannello, la loro posizione é decisa da un gestore di layout, che per default é `FlowLayout`. In `inizializzaGUI()` si può creare un altro gestore, es. `GridLayout` che li dispone su una griglia m x n (2).

```
JPanel pannello = new JPanel();                 //(1)
pannello.setLayout( new GridLayout(2,2) );      //(2)
...
bottoni = new JButton[2];                       //(3)
...
bottoni[i] = new JButton(" Bottone " + i);      //(3)
...
pannello.add(bottoni[i]);                       //(4)

getContentPane().add(pannello);                 //(5)
```

GridBagLayout, esempio



GridBagLayout, esempio



GridBagLayout, esempio

```
//...
public class SigemaClient extends JFrame {
//...
  private JPanel              pannello;
  private GridBagLayout       gbLayout;
  private GridBagConstraints  gbConstraints;

  private void aggiungiComponente( Component c, int x, int y, int dx, int dy) {
    gbConstraints.gridx = x;
    gbConstraints.gridy = y;
    gbConstraints.gridwidth = dx;
    gbConstraints.gridheight = dy;
    gbLayout.setConstraints(c, gbConstraints);
    pannello.add(c);
  }

   private void inizializzaGUI() {
        //...
        pannello              = new JPanel();
        gbLayout        = new GridBagLayout();
        gbConstraints   = new GridBagConstraints();
        pannello.setLayout( gbLayout );
        pannello.setBorder(BorderFactory.createLineBorder(Color.gray,3));
        // ...
        gbConstraints.weightx = 0;
        gbConstraints.weighty = 0;
        gbConstraints.fill = GridBagConstraints.NONE;

        gbConstraints.anchor = GridBagConstraints.WEST;
        aggiungiComponente(titolo,        0, 0, GridBagConstraints.REMAINDER, 1);
        aggiungiComponente(menu,          0, 1, GridBagConstraints.RELATIVE, GridBagConstraints.RELATIVE);

        gbConstraints.anchor = GridBagConstraints.CENTER;
        aggiungiComponente(servizio,      1, 1, GridBagConstraints.REMAINDER, 1);
        aggiungiComponente(esegui,        1, 2, GridBagConstraints.REMAINDER, GridBagConstraints.RELATIVE);
        aggiungiComponente(descrizione,   0, 3, GridBagConstraints.REMAINDER, GridBagConstraints.REMAINDER);

        getContentPane().add(pannello);
   }
   //...
}
```

Frame, aggiornamento dei componenti

• È possibile nascondere/mostrare i componenti (1) e validarli (2) ossia aggiornarne la raffigurazione.

```
pannello.setVisible(false);        //(1)
pannello.setVisible(true);
pannello.validate();               //(2)
```

• Tutti i componenti inseriti in un pannello si possono 'staccare' mediante il numero d'ordine di inserimento (3) o un riferimento (4).

```
for (int i=pannello.getComponentCount();i>=0;i--)
  pannello.remove(pannello.getComponent(i)); //(3)

  pannello.remove(bottoni[j]);                //(4)
```

• Il metodo `java.awt.Window.`**`dispose`**`()` libera le risorse video native usate dalla frame e dai suoi componenti, marcando tali componenti come non *visualizzabili.*

• Il metodo `java.awt.Window.`**`show`**`()` rende *visualizzabili* la frame e i suoi componenti, ripristinandone lo stato al momento della `dispose()`, rendendoli visibili, e portando la finestra davanti alle altre.

Frame, aggiornamento dei componenti

• Il metodo `java.awt.Component.`**`isDisplayable`**`()` ritorna `true` se il componente è *visualizzabile*, ossia è connesso a risorse video native.

• Un componente è reso *visualizzabile* quando è aggiunto ad una gerarchia di contenitori visualizzabili o quando tale gerarchia è resa tale attraverso il metodo `show` (o `pack`) sulla frame in alto della gerarchia.

• Un componente è reso *non visualizzabile* quando viene rimosso da una gerarchia di contenitori visualizzabili o quando tale gerarchia è resa tale attraverso il metodo `dispose`.

• Solo quando tutte le frame sono *non visualizzabili*, la JVM può terminare la propria esecuzione.

• Il metodo `f.`**`setDefaultCloseOperation`**`(JFrame.EXIT_ON_CLOSE)` provoca l'esecuzione di una `System.exit()` quando l'utente chiude la frame cliccando sulla "X" in alto adestra.

• In caso contrario, ed in assenza di `dispose`, la JVM non termina neanche dopo l'esecuzione del metodo `main`.

Frame, aggiornamento dei componenti

• Esempio: l'applicazione non termina dopo l'esecuzione del metodo `main`, neanche dopo la pressione del tasto "X" (il valore predefinito è `WindowConstants.HIDE_ON_CLOSE`). L'utente deve uccidere il processo con CTRL+C.

```
import javax.swing.*;
public class JFrameTest {
  public static void main(String[] args) {
    JFrame f = new JFrame();
    f.show();              // oppure f.setVisible(true);
    //(1)
  }
}
```

• Inserendo in *`//(1)`* l'istruzione

```
f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

l'applicazione termina alla pressione del tasto "X", mentre inserendo

```
f.dispose();
```

(in tal caso *`f.setVisible(false)`* non è sufficiente) l'applicazione termina automaticamente dopo aver eseguito il `main`.

Programmazione ad eventi

• Nelle applicazioni con interfacce a caratteri, il programmatore ha un forte controllo su quando invocare i metodi, richiedere o emettere dati; l'utente é passivo. Il paradigma di sviluppo é READ – EVAL – PRINT:

READ: l'utente inserisce i dati di ingresso
EVAL: il programma elabora
PRINT: l'utente legge i dati di uscita

• Nelle applicazioni dotate di interfacce grafiche interattive, il programmatore ha un lieve controllo sull'ordine di esecuzione dei metodi; é l'utente che li sceglie mediante la generazione di *eventi*. Il paradigma di sviluppo é MODEL – VIEW – CONTROLLER:

MODEL: si definiscono campi e metodi
VIEW: si definisce la presentazione dei dati
CONTROLLER: si definisce il controllo e la gestione degli eventi
(in modo analogo agli interrupt hardware).

• **evento**: azione eseguita dall' utente su un componente attivo dell' interfaccia grafica (es. la pressione di un tasto, il movimento o il click del mouse). Esistono diverse classi di eventi, che estendono `EventObject`, e sono contenute nel package `java.awt.event`.

Evento, sorgente, ascoltatore

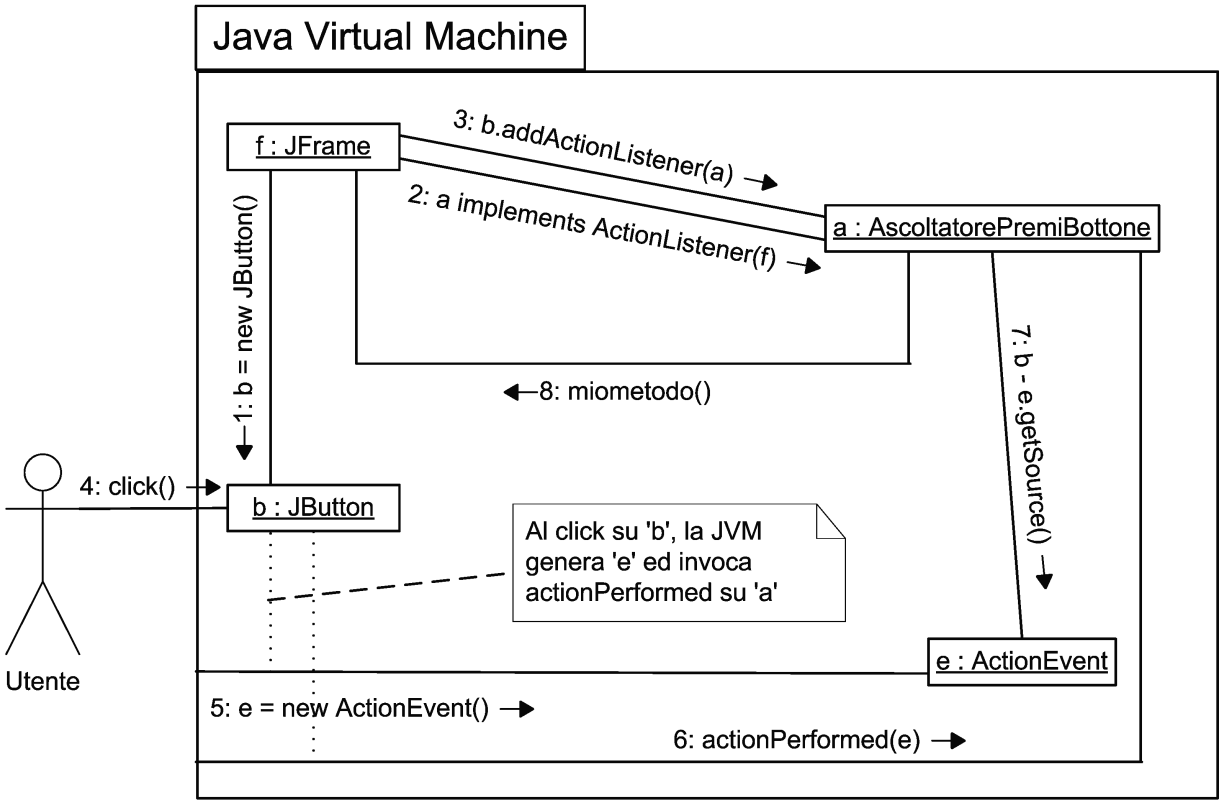
• La JVM genera un oggetto evento, della classe opportuna, ad ogni interazione con una ***sorgente*** di eventi (componente grafico), che sia stata opportunamente registrata (2), ossia associata ad un oggetto ***ascoltatore*** *(o gestore)* dell'evento (1), es. in `inizializzaGUI()`:

```
ActionListener gestore =
        new AscoltatorePremiBottone(this); //(1)
...
bottoni[0].addActionListener(gestore);     //(2)
```

• L'ascoltatore di uno o più tipi di evento é una classe, definita dal programmatore in un altro modulo, che deve implementare l'interfaccia `EventListener` o le sue estensioni. Essa deve contenere un campo (5) che viene inizializzato con l'oggetto frame specifico che contiene le sorgenti (1), ed un metodo predefinito per ogni tipo di evento (3), che invoca quindi un metodo della frame (4) con istruzioni relative all'evento.

```
public class AscoltatorePremiBottone implements...
  public void actionPerformed(ActionEvent e) { //(3)
    interf.conteggiaBottone();...              //(4)
  private MiaInterfaccia interf;               //(5)
```

Evento, sorgente, ascoltatore



Semplice interfaccia grafica

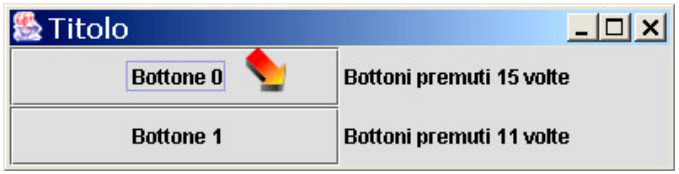
```
// MiaInterfaccia.java
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class MiaInterfaccia extends JFrame {

  // costruttore
  public MiaInterfaccia() {
    super();
    setTitle("Titolo");
    setSize(400,100);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    inizializzaGUI();
    show();
    contatore = 0;
  }

  // altri metodi
  private void inizializzaGUI() {

    // crea il pannello intermedio (pannello)
    JPanel pannello = new JPanel();

    // crea gestore del layout
    pannello.setLayout( new GridLayout(2,2) );

    // crea i componenti
    bottoni = new JButton[2];
    etichette = new JLabel[2];
    for (int i = 0; i < bottoni.length; i++) {
      bottoni[i] = new JButton(" Bottone " + i);
      etichette[i] = new JLabel(" Etichetta " + i);
    }

    // crea i gestori di eventi
    ActionListener gestore =
      new AscoltatorePremiBottone(this);

    // associa i componenti ai rispettivi gestori
    bottoni[0].addActionListener(gestore);

    // attacca i componenti al pannello
    for (int i=0; i<bottoni.length; i++) {
      pannello.add(bottoni[i]);
      pannello.add(etichette[i]);
    }

    // attacca il pannello alla Frame
    getContentPane().add(pannello);

  }

  public void conteggiaBottone() {
    contatore++;
    etichette[0].setText(" Bottone premuto " +
                         contatore + " volte");
  }

  public static void main(String[] args) {
        new MiaInterfaccia();
  }

  // campi interfaccia
  private JButton[] bottoni;
  private JLabel[]  etichette;

  // campi dato
  private int contatore;
}
```

Semplice interfaccia grafica

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

public class AscoltatorePremiBottone
                      implements ActionListener {

  public AscoltatorePremiBottone(
                     MiaInterfaccia interfaccia) {
    interf = interfaccia;
  }

  public void actionPerformed(ActionEvent e) {
    interf.conteggiaBottone();
  }

  private MiaInterfaccia interf;
}
```

- Il metodo `Object getSource()` restituisce il riferimento alla sorgente, e può servire a gestire più sorgenti di eventi dello stesso tipo.

```
// MiaInterfaccia.java (modifiche)
...
public int trovaBottone(JButton b) {
  for (int i=0; i<bottoni.length; i++)
     if (b==bottoni[i])
       return i;
  return -1;
}

// AscoltatorePremiBottone.java (modifiche)
import java.awt.event.*;
import javax.swing.*;
...
public void actionPerformed(ActionEvent e) {
  int i = interf.trovaBottone(
                          (JButton)e.getSource());
  interf.conteggiaBottone(i);
}
```



Ascoltatori principali

Interfaccia	Tipo di eventi
`ActionListener`	Quando si premono bottoni, selezionano voci di menù, si preme invio mentre si scrive in un campo di testo
`ComponentListener`	Quando un componente viene nascosto, spostato, mostrato o ridimensionato
`FocusListener`	Quando un componente ottiene o perde il focus
`ItemListener`	Quando un elemento (es. di una lista) viene selezionato o deselezionato
`KeyListener`	Quando viene premuto, rilasciato, battuto un tasto
`MouseMotionListener`	Quando il mouse viene trascinato o spostato
`MouseListener`	Quando si clicca, si rilascia, si entra, si esce, sul componente
`TextListener`	Quando cambia il valore di un campo testo
`WindowListener`	Quando la finestra viene attivata, chiusa, disattivata, ripristinata, ridotta a icona, etc.

Applet

• *Applet*: piccolo programma, scritto in Java per essere eseguito come componente *embedded* di un' altra applicazione, tipicamente un browser.

• Occorre abilitare il browser ad eseguire del bytecode, quindi integrarlo con un Java Runtime Environment (una JVM).

• Dal punto di vista di Java, una Applet è un pannello principale come la Frame, ossia uno spazio su cui si attacca un pannello con vari componenti, oppure si disegna. Viene definita come una classe che estende `JApplet`.

• Dal punto vista del browser, una pagina html riferisce una Applet come un proprio componente, specificando opportuni attributi (come le dimensioni del riquadro, la main-class) o parametri di ingresso.

```
<!-- miapagina.html -->
<APPLET CODE = "Disegna.class" CODEBASE = "miadir"
        WIDTH = 400  HEIGHT = 400 >
<PARAM NAME = "MIOFILE" VALUE = "input.txt" >
<PARAM NAME = "MIOTEST" VALUE = "true" >
</APPLET>
```

Applet

- Il browser riserva alla applet un suo spazio nella pagina, ne scarica il codice e la esegue, gestendola con opportuni metodi:

- **init()**, inizializza l'applet, eseguito ogni volta che viene caricata o aggiornata
- **start()**, dopo la creazione dell'applet, per animazioni o lanciare thread
- **stop()**, quando si vuole fermare l'esecuzione dei thread
- **destroy()**, cleanup finale, alla terminazione
- **paint(Graphics g)**, invocato da update(), per disegnare sul pannello
- **update(Graphics g)**, invoca `paint()`, per aggiornare i disegni. Graphics rappresenta la superficie su cui si può disegnare immagini, forme, stringhe (con drawString()), etc.

- Un' applet non é eseguibile come applicazione a sè, non ha un main ed un Frame che la contenga.
- È possibile aggiungere un metodo `main` il quale crea un JFrame, aggiunge ad esso una JApplet (invece che JPanel) ed invoca esplicitamente l'init().
- Se l'applet viene eseguita da un browser, il `main` verrà semplicemente ignorato.
- In alternativa, è possibile estendere la JApplet con una nuova classe che includa il `main`

Applet con disegni

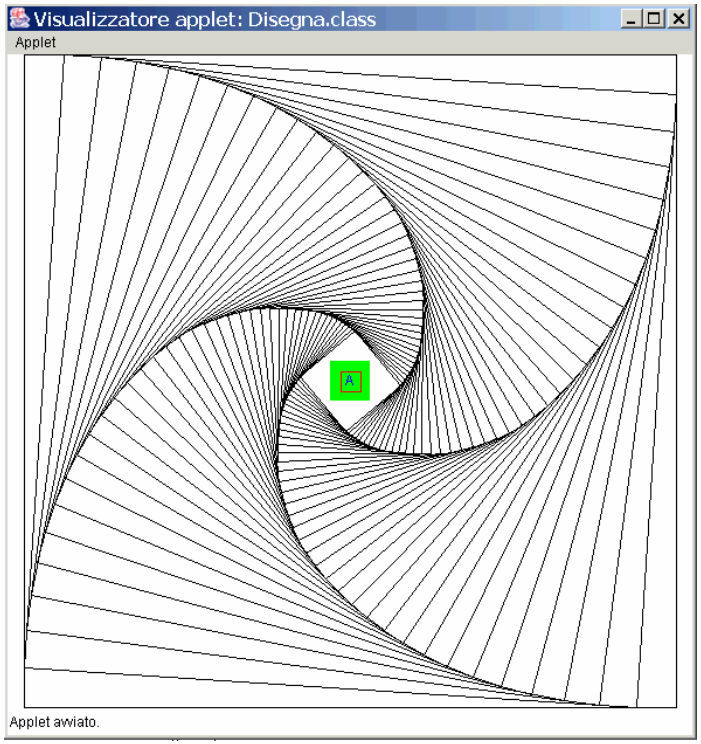
```
<!-- miapagina.html -->
<APPLET CODE = "Disegna.class"
        WIDTH = 400  HEIGHT = 400 >
</APPLET>
```

```
// Disegna.java
import javax.swing.*;
import java.awt.*;
public class Disegna extends JApplet {
  public void paint(Graphics g) {
    int   cx = getSize().width/2,
          cy = getSize().height/2;
    int x = cy, y = cy, d = 16;
    for(int i=0;i<40;i++) {
        g.drawLine(cx+x,cy+y,cx-y,cy+x);
        g.drawLine(cx-y,cy+x,cx-x,cy-y);
        g.drawLine(cx-x,cy-y,cx+y,cy-x);
        g.drawLine(cx+y,cy-x,cx+x,cy+y);
        int tmp = -(y+x)/d;
        y += (x-y)/d;
        x += tmp;
    }
    x = getSize().width/2;
    y = getSize().height/2;
    d = (x+y)/(2*d);
    g.setColor(Color.green);
    g.fillRect(x-d,y-d,2*d,2*d);
    g.setColor(Color.red);
    g.drawRect(x-d/2,y-d/2,d,d);
    g.setColor(Color.blue);
    g.drawString("A",x-d/4,y+d/4);
  }
}
```

- Digitando

  `appletviewer pagina.html`

ovvero aprendo la pagina con un browser:



Applet con lista di controllo

```
<!-- miapagina.html -->
<APPLET CODE = "CheckList.class"
        WIDTH = 100  HEIGHT = 200 >
</APPLET>
// CheckList.java
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class CheckList extends JApplet implements ItemListener {
    public void init(){
        txt = new JTextField("Seleziona i preferiti",25);
        txa = new JTextField(25);
        txt.setEditable(false);
        txa.setEditable(false);
        ck = new JCheckBox[5];
        ck[0]=new JCheckBox("Rosso");
        ...
        ck[4]=new JCheckBox("Verde");
        for (int i=0; i<5; i++)
            ck[i].addItemListener(this);
        Container c = getContentPane();
        c.setLayout(new GridLayout(7,1));
        c.add(txt);
        for (int i=0; i<5; i++) c.add(ck[i]);
            c.add(txa);
    }
    public void itemStateChanged(ItemEvent e){
        int cont = 0;
        for (int i=0; i<5; i++)
          if (ck[i].isSelected()) cont++;
            txa.setText(cont + " colori selezionati");
    }
    private JTextField  txt,txa;
    private JCheckBox   ck[];
}
```

- Il metodo `itemStateChanged` viene automaticamente invocato dopo un `ItemEvent`.
- Digitando `appletviewer pagina.html` ovvero aprendo la pagina con un browser:

Applet e sicurezza

• Una *applet* può essere caricata dal File System o dalla rete.

• Se l'applet arriva dalla rete, essa viene sottoposta alle restrizioni imposte dal **Security Manager** (SM) della JVM residente nel browser.

• Un SM può essere completamente personalizzabile ma non può essere modificato da un applet. Ogni browser ha un solo manager.

• I potenziali problemi di sicurezza causati da codice eseguibile sono:

- ***Integrity attacks*** (modifiche a file e memoria, chiusura processi)
- ***Availabilities attacks*** (allocazioni abnormi di memoria, di finestre, modifiche alla priorità dei processi)
- ***Disclosure attacks*** (prelievo di informazioni private)
- **Annoyance *attacks*** (visualizzazioni ed emissioni sonore fastidiose)

Applet e sicurezza

- Ad ogni esecuzione di metodo la JVM consulta il SM, che genera una *SecurityException* se lo considera potenzialmente pericoloso.

- Il SM controlla anche:
  - le operazioni sui Socket (`connect`,`accept`,...)
  - gli accessi al File System per proteggere file e dati personali
  - la creazione di programmi di sistema, chiamate ai processi di sistema

- Un'applet non firmata:
  - non puo' caricare librerie o definire metodi nativi
  - non puo' leggere o scrivere file sul sistema su cui e' in esecuzione, né avviare programmi
  - non puo' effettuare connessioni attraverso la rete se non con l'host da cui proviene
  - non puo' leggere alcune propieta' del sistema