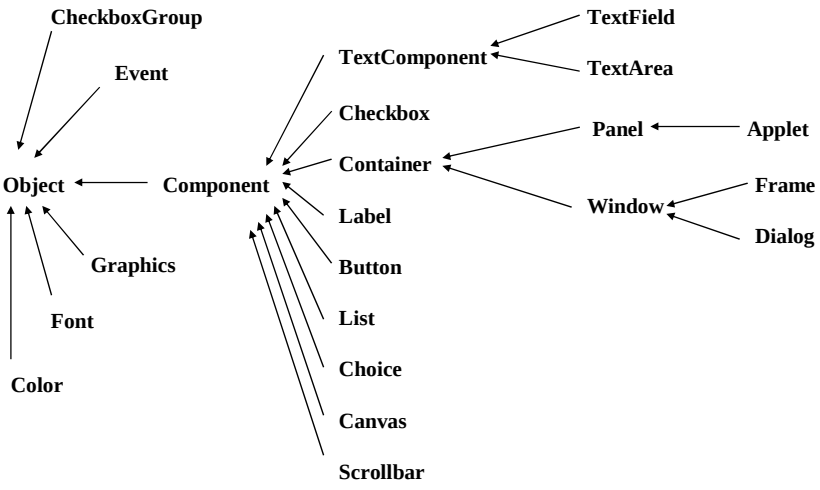
AWT: Abstract Window Toolkit

- E' una libreria che offre le componenti GUI essenziali
- Tutte le componenti GUI che sono visualizzabili sono sottoclassi della classe astratta `Component`
- `Container` è una sottoclasse astratta di Component che permette ad altre componenti di essere annidate all'interno di essa.
- Le sottoclassi più significative di `Container` sono
  - `Panel`
  - `Window`

La gerarchia di classi in java.awt



Componenti e relative classi

java.awt.Button	pulsanti per generare una qualche azione
java.awt.Checkbox	pulsanti con valore booleano on/off
java.awt.TextField	per ricevere input dalla tastiera (1 linea)
java.awt.TextArea	aree più grande per display ed editing
java.awt.Label	etichette (sola lettura)
java.awt.List	per selzionare una opzione in una serie proposta
java.awt.Choice	idem, ma solo una visibile per volta
java.awt.Scrollbar	
java.awt.Canvas	area di disegno

java.awt.Menu, java.awt.MenuItem, java.awt.CheckboxMenuItem
java.awt.Panel
java.awt.Window

Metodi di base per queste Componenti: `add` e `remove`

Window

- E' un oggetto di `java.awt.Window`
- Una `window` è una finestra autonoma sullo schermo, che è indipendente da altri contenitori
- Ci sono due tipi di finestre:
  - `frame`: finestra con titolo e angoli di ridimensionamento
  - `dialog`: finestra senza barra di menu. Può essere spostata ma non ridimensionata

Frames

- Il costruttore `Frame(String)` nella classe Frame crea un nuovo oggetto, invisibile, con il titolo specificato da `String`.
- Tutti i componenti del frame devono essere aggiunti quando non è ancora visibile
- Il frame deve essere dimensionato (in pixel): per default ha dimensione 0x0, e quindi è invisibile
- Poi, il frame viene reso visibile chiamando il metodo `setVisible(true)` oppure il metodo `show()`.

Esempio di frame

```
import java.awt.*;
public class MiaFrame extends Frame{

    public static void main(String args[]){
           MiaFrame f = new MiaFrame("Ciao!");
           f.setSize(200,300);
           f.setBackground(Color.red);
           f.setVisible(true);
    }

    public MiaFrame (String linea){
           super(linea);
    }
}
```

- Eseguendolo, compare una finestra rossa con titolo "Ciao", di dimensione 200x300 pixels
- Attenzione: questa finestra non può essere distrutta che con `Ctrl-Alt-Del`

Esempio (completo)

```
import java.awt.*;
public class MiaFrame extends Frame{

        public static void main(String args[]){
                MiaFrame f = new MiaFrame("Ciao!");
                f.setSize(500,500);
                f.setBackground(Color.red);
                f.setVisible(true);
        }

        public MiaFrame (String linea){
                super(linea);
        }

        public boolean handleEvent(Event e){
                if (e.id == Event.WINDOW_DESTROY)
                        System.exit(0);
                return super.handleEvent(e);
        }
}
```

Riscrittura del metodo `handleEvent` della classe `Component`

Inserire un testo in un frame

- Ogni volta che si vuole inserire un testo o un disegno in una finestra bisogna riscrivere il metodo paint della classe `Component`, che ha un parametro di tipo `Graphics`:
- Un oggetto di tipo `Graphics` e' una scatola per dipingere, che contiene gli strumenti per colorare, scrivere, cambiare stili, penne, colori ecc.
- Ad es. su di un oggetto di tipo Graphics può essere invocato il metodo:

```
drawString(String s, int x, int y)
```

Esempio

```
import java.awt.*;
public class MiaFrame extends Frame{

    public static void main(String args[]){
        MiaFrame f = new MiaFrame("Ciao!");
        f.setSize(300,200);
        f.setBackground(Color.red);
        f.setVisible(true);
    }

    public MiaFrame (String linea){
        super(linea);
    }

    public boolean handleEvent(Event e){
        if (e.id == Event.WINDOW_DESTROY)System.exit(0);
        return super.handleEvent(e);
    }

    public void paint(Graphics g){
        g.drawString("Carissimo Pinocchio",75,100);
    }
}
```

- il programma genera una finestra rossa, con titolo "Ciao", contenente la scritta "Carissimo Pinocchio"

Metodi `update` e `paint`

- Non esiste un modo in Java per ottenere un disegno persistente: bisogna sempre scrivere il codice in un metodo `paint` e ridisegnare la finestra
- Ogni volta che una finestra deve essere aggiornata, il gestore degli eventi chiama la funzione `update`.
- L'implementazione di default di `update` (della classe `Component`) consiste nel cancellare il background e chiamare il metodo `paint`.
- Sia `paint` che `update` hanno un unico parametro di tipo `Graphics`.

Panel

- E' un oggetto di `java.awt.Panel`
- Identifica un'area rettangolare nella quale possono essere collocati altri elementi
- Un `panel` deve essere contenuto in un altro container o dentro una finestra del web browser: nelle applicazioni stand-alone, il panel, dopo essere stato creato, deve essere aggiunto, per essere visibile, a un oggetto di tipo `Window` o `Frame`, mediante il metodo `add()` della superclasse `Container`.

Esempio

```
import java.awt.*;
public class FrameConPanel extends Frame{
    public FrameConPanel(String linea){
        super(linea);
    }
    public static void main(String args[]){
        FrameConPanel fp=new FrameConPanel("Ciao");
        Panel p=new Panel();

        fp.setLayout(new FlowLayout(FlowLayout.LEFT,0,0));
        fp.setSize(250,250);
        fp.setBackground(Color.red);
        p.setBackground(Color.blue);

        fp.add(p);
        fp.show();
    }
    public boolean handleEvent(Event e){
        if (e.id == Event.WINDOW_DESTROY)System.exit(0);
        return super.handleEvent(e);
    }
}
```

- il programma genera una finestra rossa (con titolo "Ciao") con un quadrato blu in alto a sinistra

Metodi della classe Graphics

```
import java.awt.*;

public class Grafici extends Frame{
          public void paint(Graphics g){
                    g.drawLine(10,15,50,100);     // linea (10,15)-(50,100)
                    g.drawRect(10,50,100,100);    // rettangolo vuoto
                    g.fillRect(150,50,100,100);   // rettangolo pieno
                    g.drawRoundRect(280,50,50,50,10,20);// rettangolo smussato
                    g.drawOval(350,50,70,130);    // ovale
          }
          public static void main(String args[]){
                    Grafici f = new Grafici("Ciao!");
                    f.setSize(500,300);
                    f.setBackground(Color.yellow);
                    f.show();
          }
          public Grafici (String linea){
                    super(linea);
          }
          public boolean handleEvent(Event e){
                    if (e.id == Event.WINDOW_DESTROY) System.exit(0);
                    return super.handleEvent(e);
          }
}
```


Layout Managers

- Sono responsabili della politica di layout e del dimensionamento di ogni componente grafica e delle sue sottocomponenti, ad es.
  - Flow layout
    - per Panels e Frames
    - dispone gli elementi in fila, da sinistra a destra, assegnando ad ognuno la loro dimensione preferita (centrati, per default)
  - Border layout
    - per Windows, Dialogs e Frames
    - contiene 5 aree distinte: North, South, East,West e Center
  - Grid layout
    - permette di gestire tabelle (linee e colonne fissate)
  - Card layout
    - permette di trattare l'interfaccia come una serie di carte, visibili una alla volta

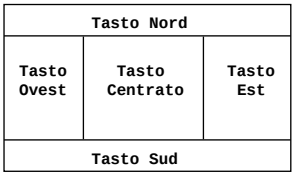
Flow Layout

```
import java.awt.*;
public class ProvaFlowLayout{
    private Frame f;
    private Button b1;
    private Button b2;
    public static void main(String args[]){
        ProvaFlowLayout finestra=new ProvaFlowLayout();
        finestra.inizia();
    }
    public void inizia(){
        f = new Frame("Esempio di Flow Layout");
        f.setLayout(new FlowLayout());
        // f.setLayout(new FlowLayout(FlowLayout.RIGHT,20,40));
        // f.setLayout(new FlowLayout(FlowLayout.LEFT,20,40));
        b1 = new Button("Premi qui");
        b2 = new Button("Non premere qui");
        f.add(b1); f.add(b2);
        f.pack(); f.show();
    }
}               // aggiungere metodo handleEvent !!
```

- Produce una finestra ridimensionabile con due bottoni, allineati al centro (risp. a ds. e a sin.)

Border Layout

```
import java.awt.*;
public class ProvaBorderLayout{
    private Frame f;
    public static void main(String args[]){
        ProvaBorderLayout finestra= new ProvaBorderLayout();
        finestra.inizia();
    }

    public void inizia(){
        f = new Frame("BorderLayout");
        f.add("North",  new Button("uno"));
        f.add("West",   new Button("due"));
        f.add("South",  new Button("tre"));
        f.add(BorderLayout.CENTER, new Button("quattro"));
        f.add(BorderLayout.EAST, new Button("cinque"));
        f.setSize(250,250);
        f.show();
     }
}
```



- Produce una finestra ridimensionabile con 5 bottoni: uno centrale e gli altri sui lati (in alto, in basso, a sin. e a ds.)

Grid layout

```
import java.awt.*;
public class ProvaGrid{
     private Frame f;
     private Button b1,b2,b3,b4,b5,b6;
     public static void main(String args[]){
          ProvaGrid griglia = new ProvaGrid();
          griglia.inizia();
     }
     public void inizia(){
          f = new Frame("Esempio di griglia");
          f.setLayout(new GridLayout(3,2));
          b1 = new Button("tasto 1");
          b2 = new Button("tasto 2");
          b3 = new Button("tasto 3");
          b4 = new Button("tasto 4");
          b5 = new Button("tasto 5");
          b6 = new Button("tasto 6");
          f.add(b1); f.add(b2); f.add(b3); f.add(b4); f.add(b5); f.add(b6);
          f.pack(); f.show();
     }
}
```

1	2
3	4
5	6

- Produce una finestra con una tabellina di 2 righe e 3 colonne di pulsanti attivabili, ingrandibile.

Eventi

- **Evento**
  oggetto che descrive "cos'è successo"
- **Sorgente dell'evento**
  ciò che ha generato l'evento
  ad es. un click del mouse genera un oggetto istanza della classe `ActionEvent`
- **Gestore dell'evento**
  metodo che riceve un oggetto di tipo evento , lo riconosce e ne gestisce gli effetti

La gestione degli eventi

- JDK 1.0: *modello gerarchico*: un evento che non è gestito dalla componente che lo ha generato si propaga al contenitore di tale componente. (vedere esempi precedenti!)
- JDK 1.1: *modello a delega*: l'evento è mandato alla componente che lo ha generato, ma è cura di ogni componente definire dei "listener" che contengono gestori degli eventi e possono riceverli ed elaborarli
- Ogni evento ha un listener corrispondente. Gli eventi che non hanno listeners registrati non si propagano

Esempio

- Quando l'oggetto di tipo `Button` viene "clickato" dal mouse, viene generata un'istanza della classe `ActionEvent`
- Questo oggetto viene recepito attraverso il metodo `actionPerformed()` da ogni oggetto di tipo `ActionListener` che sia stato registrato, attraverso il metodo `addActionListener()`, per gestire l'evento associato a tale pulsante
- Il metodo `getActionCommand()` della classe `ActionEvent` restituisce il nome del comando associato all'evento, per esempio il nome dell'etichetta del pulsante selezionato

Risponde con un messaggio

```
import java.awt.*;

public class ProvaPulsante{
        public static void main(String args[]){
                Frame f = new Frame("Prova");
                Button b = new Button("Clickami!");
                b.addActionListener(new GestisciPulsanti());
                f.add(b,"Center"); f.pack(); f.show();
        }
}
```

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

public class GestisciPulsanti implements ActionListener{
    public void actionPerformed(ActionEvent e){
        System.out.println("E' successo qualcosa!");
        System.out.println("L'etichetta del pulsante e': " +
        e.getActionCommand());
    }
}
```


Risponde con una nuova finestra

```
import java.awt.*;

public class ProvaPulsante{
        public static void main(String args[]){
                Frame f = new Frame("Prova");
                Button b = new Button("Clickami!");
                b.addActionListener(new GestisciPulsanti());
                f.add(b,"Center"); f.pack(); f.show();
        }
}
```

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

public class GestisciPulsanti implements ActionListener{
    public void actionPerformed(ActionEvent e){
        Frame f = new Frame("ProvaGestione");
        f.setBackground(Color.blue);
        f.setSize(200,200);
        f.show();

    }
}
```

Esempio (con listener)

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

public class MiaFrame extends Frame{
    public static void main(String args[]){
        MiaFrame f = new MiaFrame("Ciao!");
        f.setSize(500,500);
        f.setBackground(Color.red);
        f.addWindowListener( new CloseWindowAndExit() );
        f.setVisible(true);
    }

    public MiaFrame (String linea){
        super(linea);
    }
}

class CloseWindowAndExit extends WindowAdapter {
    public void windowClosing (WindowEvent e)
    {
        System.exit(0);
    }
}
```


Tracciare i movimenti del mouse

- Il prossimo esempio mostra un codice Java per tenere traccia dei movimenti del mouse, sia nel caso in cui il pulsante sia premuto (*mouse dragging*) che nel caso in cui il mouse viene semplicemente spostato (*mouse moving*)
- Gli spostamenti del mouse possono essere catturati da una classe che implementi l'interfaccia `MouseMotionListener`. Ma per catturare anche gli altri eventi associati al mouse (es. *clicking*), la stessa classe deve implementare anche l'interfaccia `MouseListener`

Tracciare i movimenti del mouse

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

public class DueAscoltatori implements MouseMotionListener,MouseListener{
     private Frame f;
     private TextField tf;

     public static void main(String args []){
          DueAscoltatori due = new DueAscoltatori();
          due.inizia();
     }
     public void inizia(){
          f = new Frame("Esempio");
          f.add(new Label("clicka e sposta il mouse"), BorderLayout.NORTH);
          tf = new TextField(30);
          f.add(tf,BorderLayout.SOUTH);
          f.addMouseMotionListener(this);
          f.addMouseListener(this);
          f.setSize(200,200);
          f.show();
     }
```



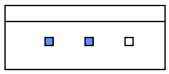
```
     // metodi di MouseMotionListener (devono essere implementati tutti!)
     public void mouseDragged(MouseEvent e){
          String s= "Sposta il mouse: (" + e.getX() + "," + e.getY() +")";
          tf.setText(s);
     }
     public void mouseMoved(MouseEvent e){}

     // metodi di MouseListener (devono essere implementati tutti!)
     public void mouseClicked(MouseEvent e){
     }
     public void mouseEntered(MouseEvent e){
          String s="Il mouse è nella finestra";
          tf.setText(s);
     }
     public void mouseExited (MouseEvent e){
          String s="Il mouse è uscito dalla finestra";
          tf.setText(s);
     }
     public void mousePressed (MouseEvent e){}
     public void mouseReleased(MouseEvent e){}
}
```

Listeners multipli

- Il fatto che un evento possa essere recepito da più listeners permette di descrivere facilmente situazioni in cui allo stesso evento devono corrispondere azioni multiple
- Inoltre consente di descrivere situazioni in cui parti non correlate del programma devono reagire agli stessi eventi (ad es. un sistema di help che illustra l'utilizzo di pulsanti/menu'….)
- Il meccanismo dei listeners permette di chiamare un metodo `addXXXListener` quante volte è necessario: tutti i listeners registrati gestiranno l'evento quando esso viene generato (l'ordine è arbitrario)

Checkboxes (stato on/off)

```
import java.awt.*;
public class ProvaCheck extends Frame{
     public static void main(String args[]){
          Frame f=new Frame("Checkbox");
          f.setLayout(new FlowLayout());
          Checkbox uno = new Checkbox("Uno",true);
          Checkbox due = new Checkbox("Due",true);
          Checkbox tre = new Checkbox("Tre",false);
          uno.addItemListener(new GestoreCheck());
          due.addItemListener(new GestoreCheck());
          tre.addItemListener(new GestoreCheck());
          f.add(uno); f.add(due); f.add(tre);
          f.pack(); f.show();
     }
}
```



```
import java.awt.event.*;
class GestoreCheck implements ItemListener{
          public void itemStateChanged(ItemEvent ev){
                    String stato = "non selezionato";
                    if (ev.getStateChange() == ItemEvent.SELECTED){
                              stato = "selezionato";
                    }
                    System.out.println(ev.getItem() + " " + stato);
          }
}
```

Choice (selezione da una lista)

```
import java.awt.*;

public class ProvaChoice extends Frame{
     public static void main(String args[]){
          Frame f=new Frame("Choice");
               f.setLayout(new FlowLayout());
          Choice scelta = new Choice();
          scelta.add("Primo");
          scelta.add("Secondo");
          scelta.add("Terzo");
          scelta.addItemListener(new GestoreChoice());
          f.add(scelta); f.pack(); f.show();
     }
}
```


```
import java.awt.event.*;
class GestoreChoice implements ItemListener{
          public void itemStateChanged(ItemEvent ev){
                    System.out.println(ev.getItem());
          }
}
```


TextField (linea di testo in input)

```
import java.awt.*;
public class ProvaText extends Frame{
  public static void main(String args[]){
     Frame f=new Frame("TextField");
     f.setLayout(new FlowLayout());
     TextField linea =
          new TextField("Dimmi qualcosa",30);
     linea.addActionListener(new GText());
     linea.addTextListener(new G2Text());
     f.add(linea); f.pack(); f.show();
  }
}
```

```
import java.awt.*;
import java.awt.event.*;
public class GText implements ActionListener{
  static int dec = 0;
  public void actionPerformed(ActionEvent e){
     Frame f = new Frame("Nuova Entry!");
     f.setBackground(Color.blue);
     f.setSize(200 - dec,200 - dec); dec +=10;
     f.show();
  }
}
```

```
import java.awt.event.*;
class G2Text implements TextListener{
  public void textValueChanged(TextEvent ev){
          System.out.println(ev);
  }
}
```