GRAFICA ED EVENTI SWING E AWT

Dott. Denis Ferraretti

denis.ferraretti@unife.it

lezioni

- Lunedì 22 febbraio 2010 ore 11.00 - 13.30
- Mercoledì 24 febbraio 2010 ore 11.00 - 13.30
- Lunedì 1 marzo 2010 ore 11.00 - 13.30

Tutte le lezioni sono nel
laboratorio di Informatica Grande.

argomenti

- SWING: architettura e gerarchia
- Componenti principali: JFrame, JPanel, Graphics
- Gestione degli eventi
- Ulteriori esempi

Java Foundation Classes

- **Swing GUI:** Include ogni componente dai Bottoni (JButton), ai pannelli condivisi (JSplitPane), alle tabelle (JTables). Vi sono componenti capaci di Ordinamenti, Stampe, e anche supporto per il Drag and Drop.
- **Pluggable Look-And-Feel:** Il Look-and-Feel delle applicazioni è personalizzabile, per esempio l'applicazione può avere il look and feel proprio del JAVA o del sistema operativo Windows. Molti package look and feel sono disponibili in rete (anche free).
- **Accessibility API:** Abilita tecnologie di assistenza per gli utenti come ad esempio Sintetizzatori vocali, o schermi in Braille.
- **Java 2D API:** Include funzioni che permettono allo sviluppatore di includere in modo semplice immagini, di includerle nei componenti e nelle applets, e anche di inviarle alle stampanti in alta qualità.
- **Internationalization:** Permette di sviluppare applicazioni per l'interazione degli utenti worldwide, includendo supporto per le lingue orientali come il Gapponese, Cinese e Koreano.

Java Package Grafici

I Package grafici che comprende tutt'ora il JDK sono

a grandi linee i seguenti due:

- **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

Scendiamo più in dettaglio nel package swing.

Java Package: SWING

- Il Package di Swing è potente, flessibile e IMMENSO.

`javax.accessibility`	`javax.swing.plaf`	`javax.swing.text`
`javax.swing`	`javax.swing.plaf.basic`	`javax.swing.text.html`
`javax.swing.border`	`javax.swing.plaf.metal`	`javax.swing.text.html.parser`
`javax.swing.colorchooser`	`javax.swing.plaf.multi`	`javax.swing.text.rtf`
`javax.swing.event`	`javax.swing.plaf.synth`	`javax.swing.tree`
`javax.swing.filechooser`	`javax.swing.table`	`javax.swing.undo`

- Il numero dei package disponibili è alto, molti programmi usano un piccolo subset di package disponibili.
- Per partire ne bastano 2:
  - **`javax.swing`**
  - **`javax.swing.event`** (non sempre richiesto).

Swing: ARCHITETTURA

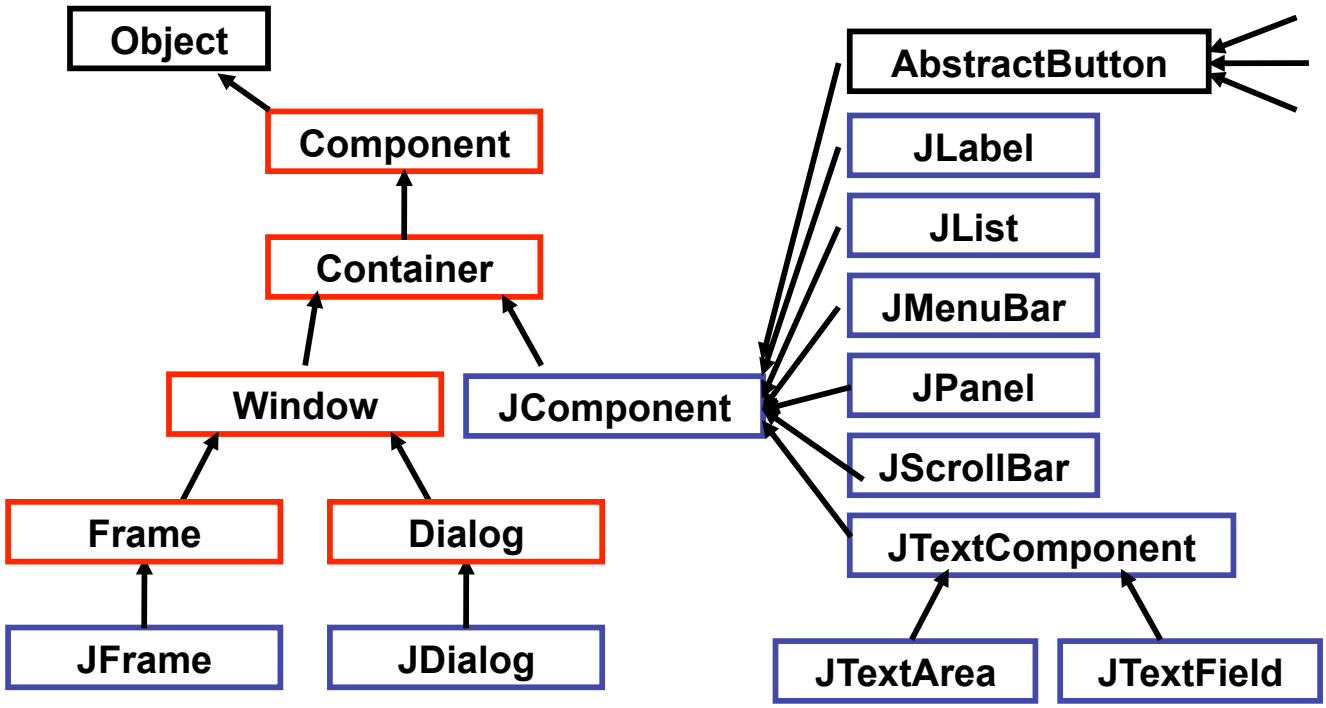
- Swing definisce una gerarchia di classi che forniscono ogni tipo di componente grafico
  - finestre, pannelli, frame, bottoni, aree di testo, checkbox, liste a discesa, etc etc

- 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

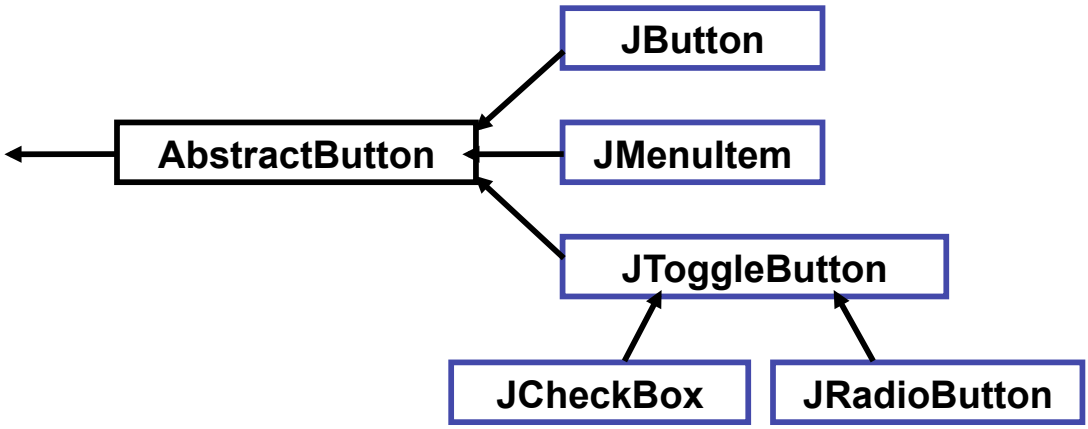
Swing: GERARCHIA

La gerarchia delle classi grafiche del java:



Swing: GERARCHIA

La gerarchia delle classi di grafiche del java:



Swing: COMPONENTI PRINCIPALI

- **Container:** tutti i componenti principali sono contenitori.
- **Window:** le finestre sono dei casi particolari di contenitori, ci sono Frame e finestre di Dialogo.
- **JFrame:** è il componente che associamo generalmente alla finestra principale di un programma con interfaccia grafica.
- **JComponent:** questo è il generico componente grafico di swing. (possiede il metodo per disegnarsi).
- **JPanel:** i pannelli sono quei componenti, che hanno il compito di contenere altri componenti e organizzarli, ma possono anche essere usati per mostrare immagini.

Swing: JFrame

- In Swing non si possono aggiungere nuovi componenti direttamente al **`JFrame`**

- 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, etc

Swing: JFrame esempio1

- Mostriamo come si può generare una finestra principale, utilizzando il package javax.swing:

```
import javax.swing.*;

public class MyFrame1 {
   /* Costruttore della classe */
   public MyFrame1() {
    //Istanzio il nuovo oggetto JFrame
    JFrame frame = new JFrame("Primo esmepio di Frame");
    //Setto le dimensioni della finestra
    frame.setSize(400, 200);
    //Questo metodo rende la finestra NON ridimensionabile
    frame.setResizable(false);
    //Setto la posizione della finestra sullo schermo
    frame.setLocation(150, 500);
    //visualizzo la finestra
    frame.setVisible(true);
   }
   //Classe statica Main
   public static void main(String[] args) {
    MyFrame1 f = new MyFrame1();
   }
}
```

Swing: JFrame esempio1

- Ed ecco qua il risultato dell'esecuzione:

- Notiamo che se si preme il pulsante di chiusura dell'applicazione questa rimane in esecuzione. Il motivo di questo comportamento è che non abbiamo inserito l'evento di chiusura.
- Se vogliamo farlo possiamo aggiungere la riga:
  - `frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);`

Swing: JFrame esempio2

- Nell'esempio precedente non è stato usata l'ereditarietà infatti nel costruttore noi abbiamo creato un oggetto **`JFrame`**.

- Generalmente è preferibile estendere la classe JFrame per generare un componente personalizzato per la nostra applicazione.

- Vediamo quindi un esempio di come si può usare l'ereditarietà per prelevare da JFrame tutte le caratteristiche che ci servono e aggiungere quelle che vogliamo inserire nella nostra applicazione.

Swing: JFrame esempio2

- Questa classe estende JFrame, questo significa che preleviamo tutti i metodi e le variabili dalla superclasse, e possiamo riutilizzarli con la nostra che si chiama MyFrame2.

```
import javax.swing.*;
public class MyFrame2 extends JFrame {

    public MyFrame2(String titolo) {
    super(titolo);
    setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    setBounds(50, 50, 150, 500);
    setVisible(true);
    }

    public MyFrame2() {
    super();
    setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    setBounds(250, 50, 150, 500);
    setVisible(true);
    }

    public static void main(String[] args) {
    MyFrame2 finestra1 = new MyFrame2("Prova!");
    MyFrame2 finestra2 = new MyFrame2();
    }
}
```

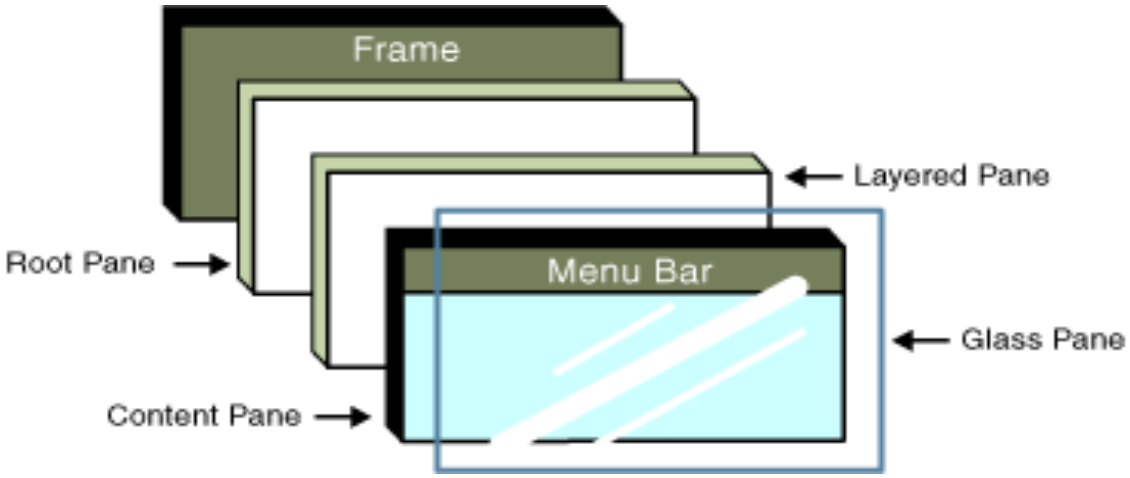

Swing: JFrame esempio2

- Nell'esempio, per usare l'ereditarietà è stato necessario estendere la classe **`JFrame`** con una nostra classe personalizzata tramite la parola chiave **`extends`**, in questo modo ereditiamo tutti i metodi e i comportamenti della classe **`JFrame`** nella nostra classe.

- L'ereditarietà ci permette anche di chiedere aiuto alla superclasse nella creazione dei costruttori, tramite la parola chiave **`super`**. Questa richiama il costruttore della classe **`JFrame`** ma l'oggetto che viene creato è di tipo **`MyFrame2`**, quindi ha le caratteristiche di un **`JFrame`** ma comprende anche le personalizzazioni che abbiamo fatto.

Swing: JFrame

- Una JFrame è composta da diversi strati, ciascuno dei quali può essere immaginato come un pannello con una funzione differente. Ad esempio il Glass Pane è trasparente e responsabile di raccogliere gli eventi.
- Il Content Pane invece è il contenitore principale a cui si aggiungono i componenti da visualizzare e organizzare.



Swing: JFrame e JPanel

- Abbiamo visto come sia possibile utilizzare le superclassi per definire delle classi nostre che hanno già un comportamento definito, ma l'ereditarietà viene utilizzata anche per eseguire una "sovrascrittura" di un metodo per cambiarne il comportamento. Questa operazione prende il nome di "Override".
- Possiamo ad esempio considerare una finestra (JFrame) che disegna un pannello (JPanel), il quale anziché venire disegnato vuoto e con sfondo grigio, disegna delle forme e dei colori.
- In questo caso cambiamo il comportamento di un componente swing e sovrascriviamo i metodi che il java utilizza per disegnare i componenti, per passargli le istruzioni di disegno che vogliamo.

Swing: JFrame e JPanel

- Prepariamo un JFrame che possieda al suo interno un JPanel:

```
import javax.swing.*;
import java.awt.*;
public class MyFrame3 extends JFrame {
   public MyFrame3(String titolo) {
    super(titolo);
    this.setSize(400, 500);
    Container c = this.getContentPane();
    MyPanel pannello = new MyPanel();
    //JPanel pannello = new JPanel();
    c.add(pannello);
    setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    setVisible(true);
   }

   public static void main(String[] args) {
MyFrame3 finestra = new MyFrame3("Finestra con Pannello!");
   }
}
```


Swing: JFrame e JPanel

- Il codice che abbiamo mostrato, non può essere eseguito ancora poiché manca la definizione della classe **`MyPanel`**. Se vogliamo eseguire comunque l'esempio possiamo togliere il commento alla riga commentata e inserirlo in quella precedente. Vedremo una **`JFrame`** con un pannello con sfondo grigio (esattamente come nel primo esempio.)
- Proviamo ora a estendere la classe **`JPanel`** e andare a sovrascrivere il metodo che viene utilizzato per il disegno di questo componente:

  `paintComponent(Graphics g);`
- L'oggetto **`Graphics`** fa parte del package **`java.awt`** ed è molto utile per il disegno di oggetti, soprattutto nella sua estensione **`Graphics2D`**.

Swing: JFrame e JPanel

- La struttura base della classe sarà la seguente:

```
import javax.swing.JPanel;
import java.awt.Graphics;
import java.awt.Color;
import java.awt.Font;
public class MyPanel extends JPanel {

  public MyPanel() {
   super();
   setBackground(Color.WHITE);
  }
```

...

Swing: JFrame e JPanel

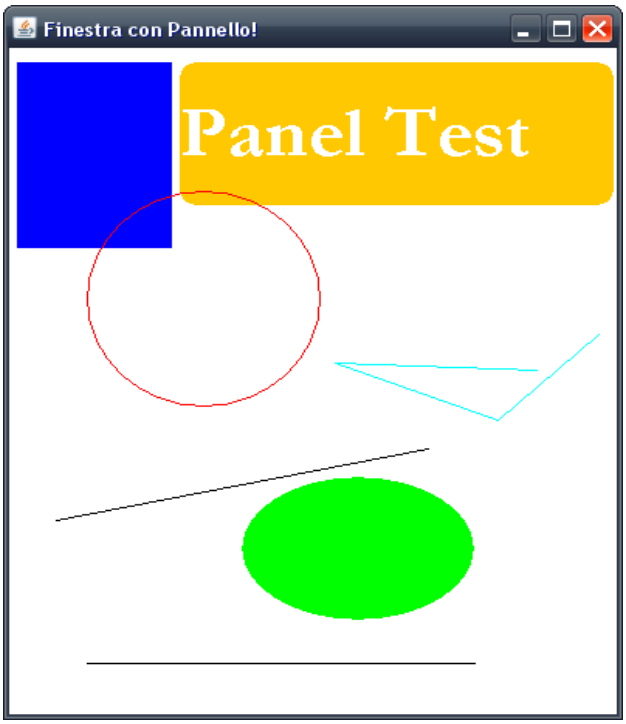
- Aggiungiamo l'implementazione del metodo:

```
public void paintComponent(Graphics g) {
      super.paintComponent(g);
 g.setColor(Color.BLUE);
 g.fillRect(5, 10, 100, 130);

 g.setColor(Color.ORANGE);
 g.fillRoundRect(110, 10, 280, 100, 20, 20);

 g.setColor(Color.RED);
 g.drawOval(50, 100, 150, 150);

 g.setColor(Color.GREEN);
 g.fillOval(150, 300, 150, 100);

 g.setColor(Color.BLACK);
 g.drawLine(50, 430, 300, 430);
 g.drawLine(30, 330, 270, 280);

 g.setColor(Color.WHITE);
 g.setFont(new Font("Garamond", Font.BOLD, 50));
 g.drawString("Panel Test", 110, 75);

 g.setColor(Color.CYAN);
 int[] x = new int[]{380, 315, 210, 340};
 int[] y = new int[]{200, 260, 220, 225};
 g.drawPolyline(x, y, 4);
    }
}
```

STRUMENTI JAVA PER LO SVILUPPO DI INTERFACCE UTENTE E SERVIZI DI RETE E LORO APPLICAZIONE

Swing: JFrame e JPanel

- Il risultato dell'esecuzione è il seguente:



Swing: JFrame e JPanel

- Abbiamo visto come per disegnare sia stato utilizzato l'oggetto **`Graphics`**. Questo possiede molti metodi, che possono essere suddivisi in due grandi gruppi:
  - Metodi di disegno e riempimento che servono per disegnare (in gergo "to render") delle forme base, del testo e delle immagini.
  - Metodi per il setting di attributi che modificano come avvengono il disegno e il riempimento. Per esempio i metodi **`setFont`** e **`setColor`**.
- In realtà tutti gli oggetti **`Graphics`** della **`javax.swing`** sono oggetti **`Graphics2D`**, molto più avanzato di **`Graphics`**, e potente. Viene mantenuto **`Graphics`** nella firma dei metodi per compatibilità tra versioni.

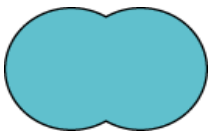
Swing: Graphics2D

- Nella classe sono presenti dei Rendering Hints, ovvero dei metodi per settare alcuni parametri che si occupano di migliorare la visualizzazione, di velocizzarla e di cambiare per esempio il tipo di riempimento.
- Per esempio si può settare la proprietà di Antialiasing del testo che per dimensioni di font troppo grandi o troppo piccole, risulta illeggibile.

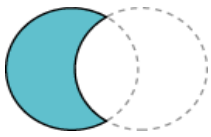
```
public void paint(Graphics g) {
 Graphics2D g2 = (Graphics2D)g;
 RenderingHints rh = new RenderingHints(
 RenderingHints.KEY_TEXT_ANTIALIASING,
 RenderingHints.VALUE_TEXT_ANTIALIAS_ON);
 g2.setRenderingHints(rh);
}
```


Swing: Graphics2D

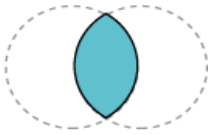
- E' possibile utilizzare le primitive del Graphics2D anche per fare grafica CAG (Constructive Area Geometry) ovvero per ottenere forme complesse utilizzando forme più semplici combinate con logica booleana.



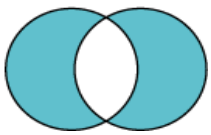
Unione



Sottrazione



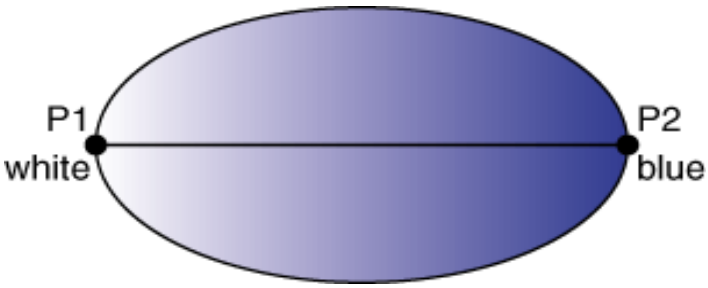
Intersezione



Or esclusivo

Swing: Graphics2D

- Tramite altre primitive definite dal Java, è possibile utilizzare la grafica 2D per eseguire trasformazioni sulle figure disegnate come rotazioni, traslazioni, stiramenti delle figure e anche utilizzare trasparenze e riempimenti a gradiente oppure a texture.



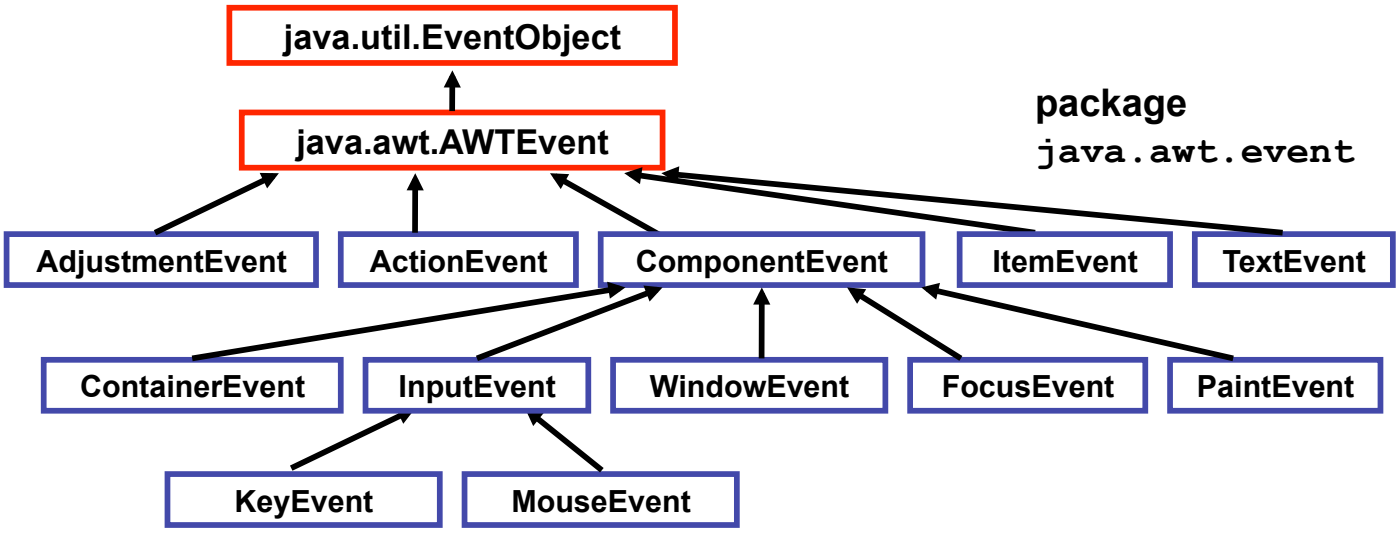
- Il riempimento a gradiente utilizza 2 colori di base associati a 2 punti spaziali. Sfuma il primo colore mentre si muove verso il secondo punto nel disegno dell'immagine.

Swing: Eventi

- Finora si sono utilizzate alcune delle funzionalità di swing, ma l'interazione con l'utente non è stata presa in considerazione.
- Ogni applizazione deve poter interagire con l'utente, e per fare questo deve essere reattiva su alcuni componenti, che quindi devono essere in grado di determinare il proprio stato e lanciare opportuni segnali quando questo viene modificato dall'utente.
- Ogni componente grafico, quando si opera su di esso, genera un evento che descrive cosa è accaduto.
- 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 "modificato" per indicare che la casella è stata selezionata / deselezionata.

Swing: Eventi

- In Java, un evento è un oggetto, istanza di (una sottoclasse di) **java.util.EventObject**
- Vediamo la gerarchia delle classi per gli eventi:



Swing: Eventi

- Un modo semplice per definire un metodo di "ascoltare" gli eventi che possono essere generati da un componente è quello di implementare un **`ActionListener`**.

- Per scrivere un **`ActionListerer`** bisogna seguire i seguenti passi:
  - Dichiarare un gestore di eventi (event handler) che implementa l'interfaccia **`ActionListener`**, oppure estende una classe che a sua volta l'interfaccia **`ActionListener`**.

```
public class MyClass implements ActionListener {
```

  - Registrare una istanza della classe gestore di eventi come listener di uno o più eventi.

```
someComponent.addActionListener(instanceOfMyClass);
```

  - Includere il codice che implementa l'interfaccia ActionListener per gestire l'evento.

```
public void actionPerformed(ActionEvent e) {
    ...//code that reacts to the action...
}
```

Swing: Eventi un esempio

Per provare a gestire qualche evento, proviamo a scrivere un programma che implementi una semplice 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

Swing: Eventi un esempio

- Partiamo dall'interfaccia grafica:
- Per creare i bottoni, abbiamo bisogno di una classe che ci permetta di farlo: JButton.

```
import javax.swing.JButton;
import java.awt.Font;
public class CalcButton extends JButton {
   public CalcButton(String n) {
    super(n);
   setFont(new Font("Bitstream Vera Sans Mono", Font.BOLD, 20));
   }
}
```

- Creiamo ora il display della calcolatrice:

```
import javax.swing.JTextField;
import java.awt.*;
public class CalcDisplay extends JTextField {
   public CalcDisplay(int n) {
    super(n);
setFont(new Font("Bitstream Vera Sans Mono", Font.PLAIN, 30));
   }
}
```

Swing: Eventi un esempio

- Bisogna implementare la Frame del calcolatore:

```
import javax.swing.JFrame;
import javax.swing.JTextField;
import javax.swing.JPanel;
import java.awt.Container;
public class Calculator extends JFrame {
 public Calculator() {
     super();
     setBounds(200, 100, 400, 120);
     JPanel ui = createUI();
     Container c = this.getContentPane();
     c.add(ui);
     setTitle("Mini Calcolatrice");
     setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
     setVisible(true);
 }

 public static void main(String[] args) {
     Calculator calc = new Calculator();
 }
}
```


Swing: Eventi un esempio

- Metodo per creare l'interfaccia grafica:

```
private JPanel createUI() {
     JPanel pannello = new JPanel();
     CalcDisplay testo = new CalcDisplay(15);
     testo.setHorizontalAlignment(JTextField.RIGHT);
     CalcButton calc = new CalcButton("=");
     CalcButton canc = new CalcButton("C");
     CalcButton sum = new CalcButton("+");
     CalcButton sub = new CalcButton("-");
     CalcButton div = new CalcButton("/");
     CalcButton mul = new CalcButton("*");
     pannello.add(testo);
     pannello.add(sum);
     pannello.add(sub);
     pannello.add(div);
     pannello.add(mul);
     pannello.add(calc);
     pannello.add(canc);
     CalcEngine engine = new CalcEngine(testo);
     sum.addActionListener(engine);
     sub.addActionListener(engine);
     mul.addActionListener(engine);
     div.addActionListener(engine);
     calc.addActionListener(engine);
     canc.addActionListener(engine);
     return pannello;
}
```

Swing: Eventi un esempio

- Creazione del motore che ascolta le azioni ed esegue i calcoli:

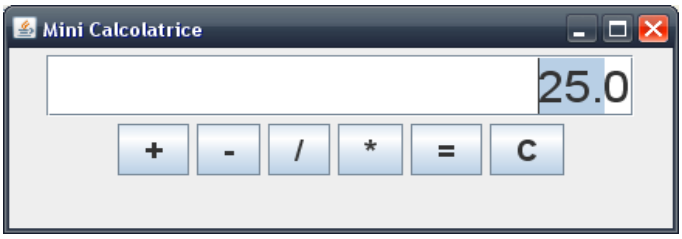
```
import java.awt.event.*;
public class CalcEngine implements ActionListener {
   public CalcEngine(CalcDisplay dis) {
    display = dis;
    precOp = "nop";
   }

   public void actionPerformed(ActionEvent ev) {
       ...//Implementazione del metodo...
   }
   private CalcDisplay display;
   private double result;
   private String precOp;
}
```


Swing: Eventi un esempio

```
   public void actionPerformed(ActionEvent ev) {
        double valore = Double.parseDouble(display.getText());
        display.setText("");
        display.requestFocus();
        String operazione = ev.getActionCommand();
        if (operazione.equals("C")) {
                result = 0;
                valore = 0;
                precOp = "nop";
        } else {
                if (precOp.equals("+")) {
                        result += valore;
                } else if (precOp.equals("-")) {
                        result -= valore;
                } else if (precOp.equals("*")) {
                        result *= valore;
                } else if (precOp.equals("/")) {
                        result /= valore;
                } else if (precOp.equals("nop")) {
                        result = valore;
                }
                display.setText("" + result);
                precOp = operazione;
      }
}
```

Swing: Eventi un esempio

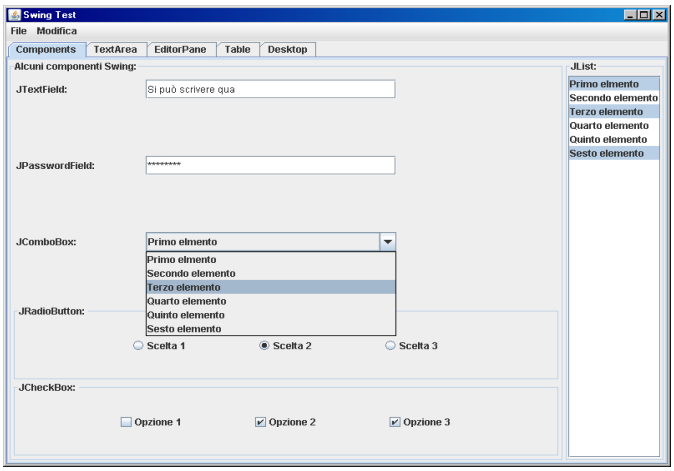
- Il risultato dovrebbe essere il seguente:

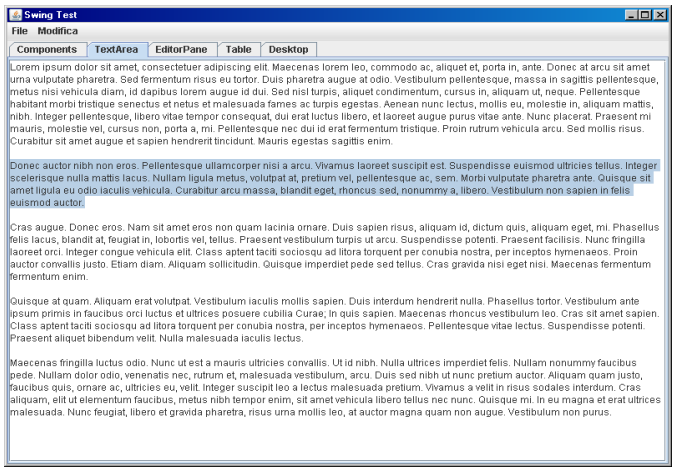


- Abbiamo utilizzato un unico gestore di evento per prelevare e gestire tutti gli eventi dei pulsanti che abbiamo creato.
- Tramite il metodo **`String getActionCommand();`** abbiamo prelevato il nome del bottone che era responsabile dell'evento.
- Ci sono pattern di programmazione migliori di questo.

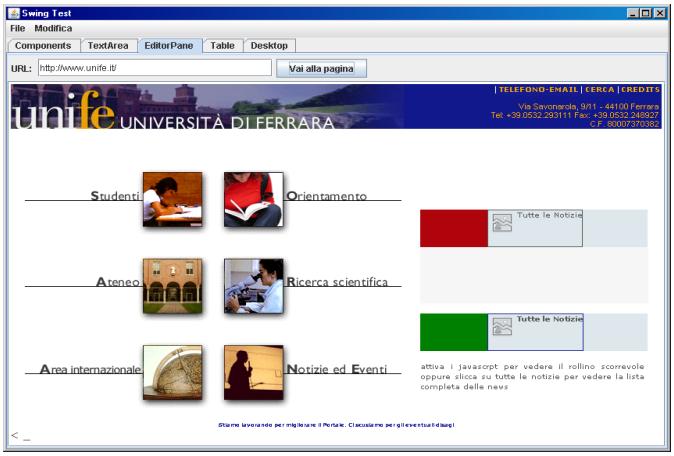
Ulteriori esempi 1

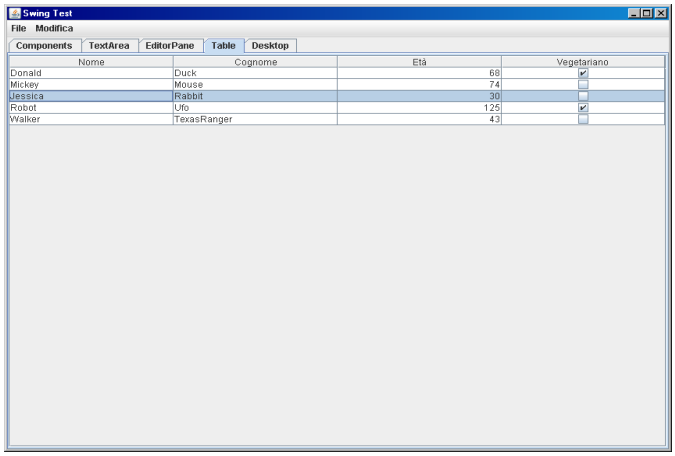
Esempio di utilizzo dei fondamentali componenti di SWING.



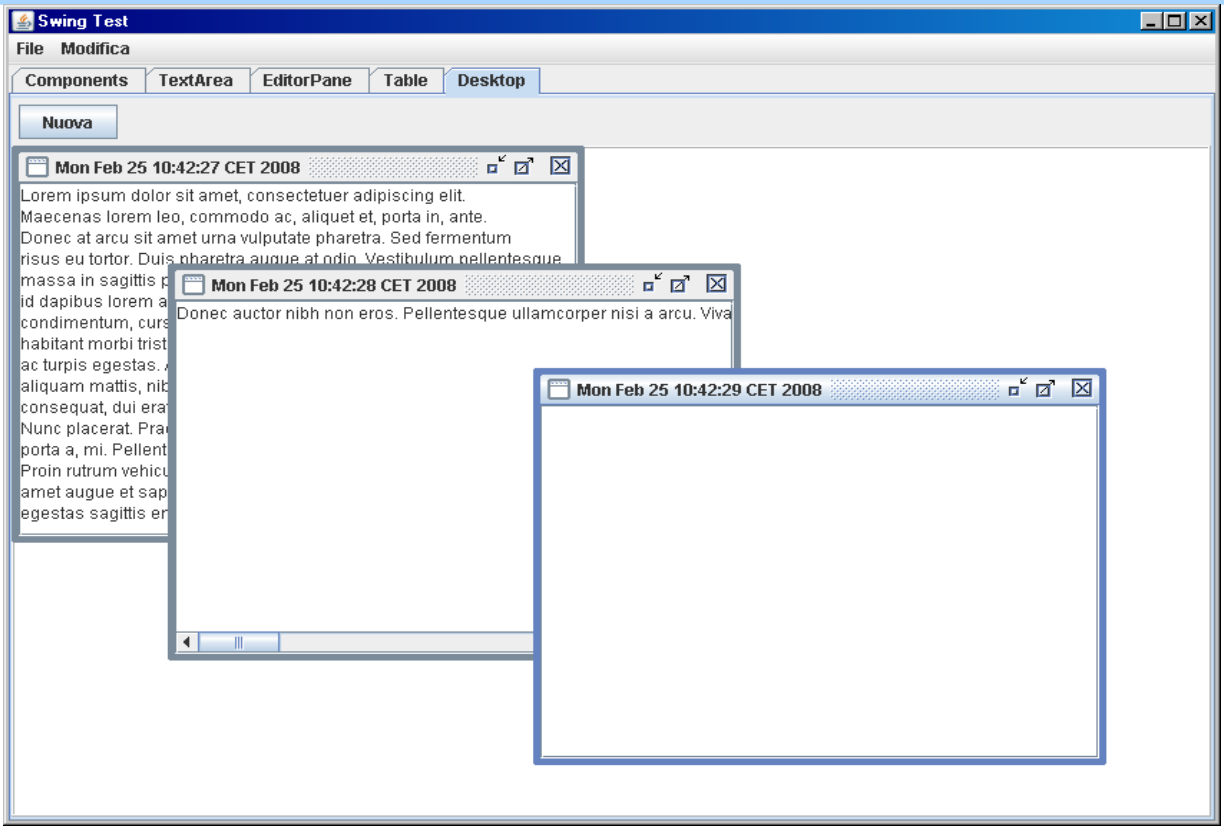


Ulteriori esempi 1





Ulteriori esempi 1



STRUMENTI JAVA PER LO SVILUPPO DI INTERFACCE UTENTE E SERVIZI DI RETE E LORO APPLICAZIONE

Ulteriori esempi 2

Applicazione per il disegno di funzioni.