Java

creare finestre

G. Prencipe
prencipe@di.unipi.it

Introduzione

- L'obiettivo originale delle librerie per *interfacce grafiche utente* (*GUI*) in Java1.0 era di permettere al programmatore di costruire interfacce che apparivano buone su tutte le piattaforme
- Questo obiettivo non è stato raggiunto. Infatti le *Abstract Window Toolkit* (*AWT*) producevano interfacce ugualmente medicri su tutti i sistemi
- Inoltre erano piuttosto restrittive
  - Si potevano utilizzare solo 4 font e non si potevano accedere alle potenzialità delle GUI del sitema operativo sottostante

Introduzione

- Inoltre il modello di programmazione per le AWT non era orientato agli oggetti!!
  - Il problema era nel fatto che queste librerie sono state progettate e realizzate nel giro di un mese dalla Sun!!
- La situazione migliorò in Java1.1 con il modello AWT ad eventi, con un approccio più pulito e orientato agli oggetti
  - Inoltre vennero aggiunti i *JavaBeans*

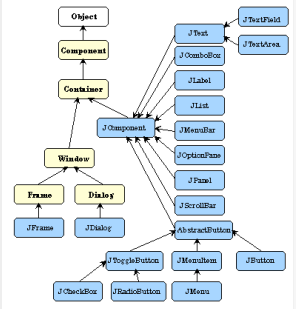
Introduzione

- Java2 (1.2) completò la trasformazione essenzialmente rimpiazzando le vecchie librerie con le *Java Foundation Classes* (*JFC*), la cui porzione dedicata alle GUI è nota come *Swing*
- Descriveremo solo le *Swing*, che si trovano in **javax.swing**
  - Come al solito riferire sempre la documentazione, dato che queste librerie sono molto vaste, è non è possibile descrivere tutto
- Le *Swing* possono essere utilizzate sia all'interno di *applets* che di normali applicazioni

Introduzione

- Cominceremo analizzando l'utilizzo delle *Swing* in normali applicazioni
  - In pratica esploreremo le potenzialità grafiche di Java e delle *Swing*
- Poi ne vedremo l'utilizzo con le applets
  - Le applets sono piccoli programmi che possono essere eseguiti all'interno di un browser Web

Gerarchia grafica



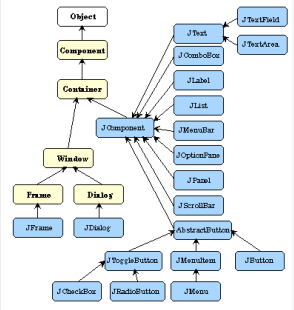
La parte delle *Swing* è quella in blu

Quella in giallo riguarda le AWT

Le componenti Swing hanno nome simile alle componenti AWT, ma inizia per **J**

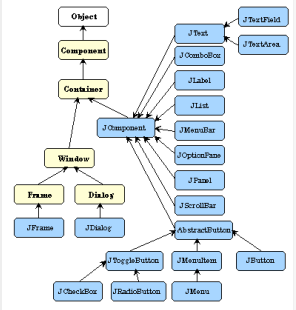
Non sono rappresentate tutte le classi

Gerarchia grafica



Un **Component** (classe astratta) è un oggetto avente una rappresentazione grafica, e che può essere visualizzato sullo schermo (bottoni, scrollbars, ecc.)

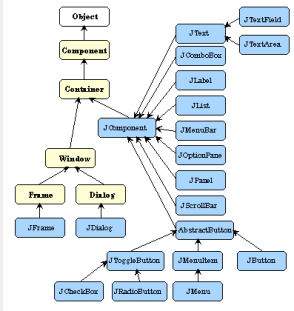
Gerarchia grafica



Un **Component** (classe astratta) è un oggetto avente una rappresentazione grafica, e che può essere visualizzato sullo schermo (bottoni, scrollbars, ecc.)

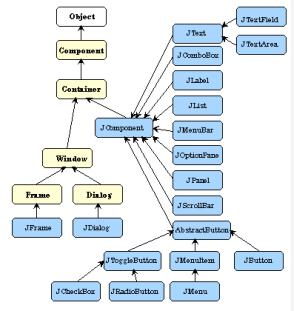
Un generico oggetto **Container** di AWT è un **Component** che può *contenere* altre *componenti* AWT

Gerarchia grafica



Le componenti *Swing*, cioè sottoclassi di **JComponent**, sono anche **Container** e quindi possono contenere altre componenti *Swing*

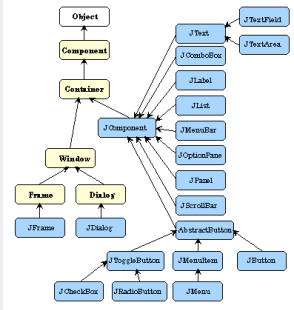
Gerarchia grafica



Le componenti *Swing*, cioè sottoclassi di **JComponent**, sono anche **Container** e quindi possono contenere altre componenti *Swing*

Le componenti *Swing* hanno comunque disponibili tutti i metodi di **Component** e **Container** delle vecchie AWT

Gerarchia grafica



JFrame, **JApplet**, **JDialog**, **JWindow** sono le componenti *Swing heavyweight*

Sono le componenti che sono *in cima* a qualsiasi gerarchia Swing (sono detti *top-level container*)

Sono utilizzati per contenere le componenti *lightweight* (bottoni, testi, ecc.)

Regole generali

- Per essere visualizzata sullo schermo, ogni componente della GUI deve essere parte di una *gerarchia di contenimento*
  - Gerarchia di contenimento: un albero di componenti che ha un contenitore top-level come radice
- Swing fornisce le seguenti classi *top-level container*
  - **JFrame**, **JDialog**, **JWindow**, e **Japplet**
  - Sono in cima ad ogni gerarchia di contenimento *Swing*

Regole generali

- Ogni programma che utilizza componenti Swing ha almeno una top-level container
  - Esso è la radice di una gerarchia di contenimento
- Tipicamente, una applicazione con GUI basata sulel Swing ha come radice un **JFrame**
- Una applet basata sulle Swing ha almeno una gerarchia di contenimento radicata in un oggetto **JApplet**

Regole generali

- Ogni componente GUI può essere contenuta *una sola volta*
  - Se una componente è già in un contenitore e si cerca di aggiungerla a un altro contenitore, la componente viene rimossa dal primo e aggiunta al secondo contenitore
- Ogni top-level container contiene un *pannello di contenimento* (*content pane*) che contiene (direttamente o indirettamente) le componenti visibili in quel contenitore top-level

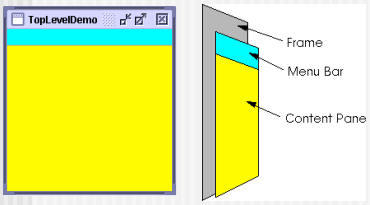
Content pane

- È un contenitore che serve a contenere componenti grafiche *lightweight*
  - Cioè, bottoni, aree di testo, ecc.
- È contenuto all'interno di un contenitore *heavyweight*
  - Contenitori top-level

Regole generali

- Opzionalmente è possibile aggiungere una barra dei menù a un top-level container.
- Per convenzione, la barra dei menù è posizionata all'interno del top-level container, ma fuori del content pane

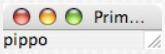
Esempio



In generale, per visualizzare le componenti grafiche, bisogna
- Creare un top-level container
- Creare un content pane
- Aggiungere le componenti grafiche nel content pane

Esempio

```
import javax.swing.*;
import java.awt.*;
public class PrimoFrame  {
    public static void main(String[] args) {
          JFrame f = new JFrame("Primo Frame");
          Container cp = f.getContentPane();
          cp.add(new JLabel("pippo!"));
          f.setVisible(true);
    }
} ///:~
```



- Lo scopo di questo pezzo di codice è di inserire una etichetta di testo su un **JFrame**, che è una delle possibili finestre offerte dalle Swing

Esempio

```
import javax.swing.*;
import java.awt.*;
public class PrimoFrame  {
    public static void main(String[] args) {
          JFrame f = new JFrame("Primo Frame");
          Container cp = f.getContentPane();
          cp.add(new JLabel("pippo!"));
          f.setVisible(true);
    }
} ///:~
```

- Il *frame* (radice della gerarchia) viene creato invocano il costruttore a cui passiamo il titolo del frame
- Inizialmente il frame non è visibile
  - Viene reso visibile invocando **setVisible(true)** in **Component**
  - Tutti i componenti sono visibili per default, ad eccezione dei componenti top-level

Esempio

```
import javax.swing.*;
import java.awt.*;
public class PrimoFrame  {
    public static void main(String[] args) {
          JFrame f = new JFrame("Primo Frame");
          Container cp = f.getContentPane();
          cp.add(new JLabel("pippo!"));
          f.setVisible(true);
    }
} ///:~
```

- Il content pane viene ottenuto invocando il metodo **getContentPane()** in **JFrame**
- Questo metodo restituisce un **Container**

Esempio

```
import javax.swing.*;
import java.awt.*;
public class PrimoFrame  {
    public static void main(String[] args) {
          JFrame f = new JFrame("Primo Frame");
          Container cp = f.getContentPane();
          cp.add(new JLabel("pippo!"));
          f.setVisible(true);
    }
} ///:~
```

- L'etichetta viene creata utilizzando la classe **JLabel**
  - Il costruttore di etichette prende una **String**a e crea l'etichetta

Esempio

```
import javax.swing.*;
import java.awt.*;
public class PrimoFrame  {
    public static void main(String[] args) {
          JFrame f = new JFrame("Primo Frame");
          Container cp = f.getContentPane();
          cp.add(new JLabel("pippo!"));
          f.setVisible(true);
    }
} ///:~
```

- Il metodo **Component add(Component comp)** è nella classe **java.awt.Container**
  - **JLabel** è un **Component**
  - **add()** aggiunge una componente a un contenitore, e restituisce la stessa componente **comp** (in questo esempio il valore ritornato è ignorato)

Creare un elemento grafico

- Quindi, per ora abbiamo visto che le componenti grafiche devono essere aggiunte a un contenitore
- Il contenitore visto finora è quello che ci restituisce il metodo **getContentPane()** in **JFrame**

Content pane

- Con l'esempio precedente, abbiamo creato un frame, ottenuto da esso un content pane, e aggiunto la componente grafica
- Il content pane ottenuto da **getContentPane()** è un **Container**, cioè non si trova nella gerarchia dei **JComponent**, ma nella parte delle vecchie AWT
- Vediamo quali componenti ci vengono offerte dalla *Swing* da poter utilizzare come contenitori di componenti grafiche

JPanel

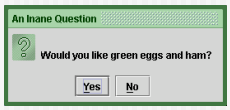
- **JPanel** è un esempio di contenitore per componenti *lightweight* che è una **JComponent** (cioè nella nuova gerarchia introdotta dalle Swing)

Esempio

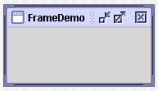
```
import javax.swing.*;
import java.awt.*;
public class PrimoFrameJPanel  {
    public static void main(String[] args) {
          JFrame f = new JFrame("Primo Frame");
          JPanel p = new JPanel();
          f.setContentPane(p);
          p.add(new JLabel("pippo!"));
          f.setVisible(true);
     }
} ///:~
```

Carrellata grafica della gerarchia

Applet



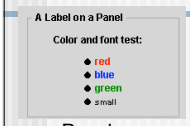
Dialog



Frame

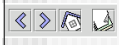
- Top-Level Containers: le componenti in cima a qualsiasi gerarchia di contenimento Swing

Carrellata grafica della gerarchia



Panel

Scroll pane



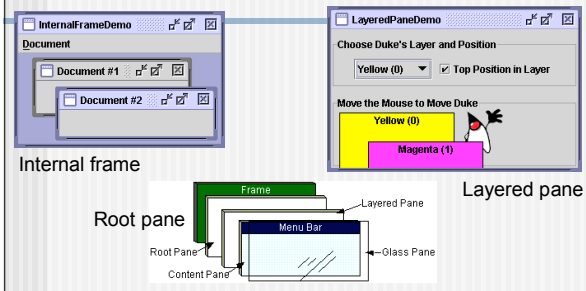
Tool bar

Tabbed pane

Split pane

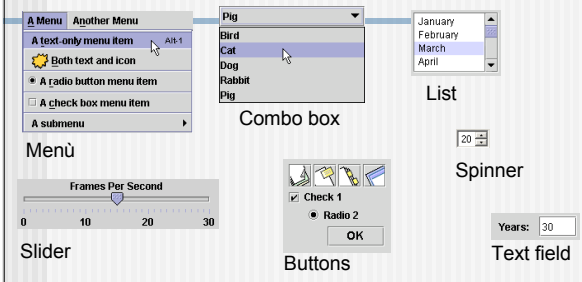
- General-Purpose Containers: conteniter intermedi che possono essere utilizzati in varie circostanze

Carrellata grafica della gerarchia



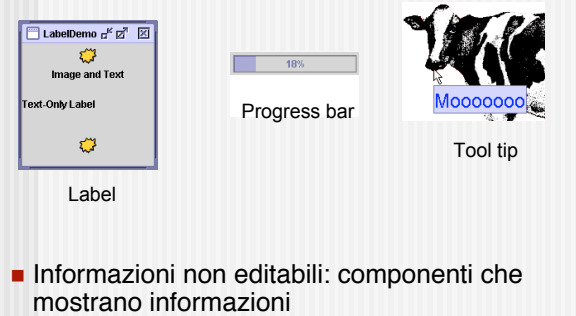
- Special-Purpose Containers: contenitori intermedi che hanno ruoli specifici nelle interfacce

Carrellata grafica della gerarchia



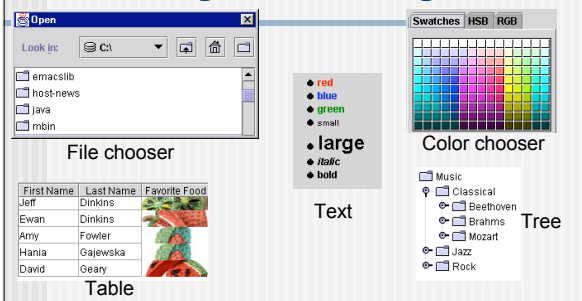
- Basic Controls: componenti per ottenere input dall'utente

Carrellata grafica della gerarchia



- Informazioni non editabili: componenti che mostrano informazioni

Carrellata grafica della gerarchia



- Informazioni formattate interattive: componenti che visualizzano informazioni formattate (eventualmente) modificabili

Creare bottoni

- Tutti i bottoni sono sottoclassi di **AbstractButton**
- Un tipo di bottone semplice è offerto da **JButton**. Per crearne uno bisogna invocare il costruttore di **JButton** con argomento il nome che vogliamo sul bottone
  - Il nome dato al momento della creazione può essere recuperato con il metodo **getTex()** in **AbstractButton**
  - Si possono fare anche cose più sofisticate
- **JButton** è una componente
- Vediamo un esempio

Esempio

```
public class PrimiBottoni {
  public static void main(String[] args) {
    JButton
          b1 = new JButton("Bottone 1"),
          b2 = new JButton("Bottone 2");
    JFrame f = new JFrame("Primi Bottoni");
    Container c = f.getContentPane();
    c.setLayout(new FlowLayout());
    c.add(b1);
    c.add(b2);
    f.setSize(100,100);
    f.setVisible(true);
  }
} ///:~
```

Prima di aggiungere i bottoni, si assegna al contenitore un *manager di layout*

Estende direttamente **Object**

Serve per piazzare i componenti (quando c'è n'è più di 1) da sinistra a destra e dal basso in alto

Esempio

```
public class PrimiBottoni {
  public static void main(String[] args) {
    JButton
          b1 = new JButton("Bottone 1"),
          b2 = new JButton("Bottone 2");
    JFrame f = new JFrame("Primi Bottoni");
    Container c = f.getContentPane();
    c.setLayout(new FlowLayout());
    c.add(b1);
    c.add(b2);
    f.setSize(100,100);
    f.setVisible(true);
  }
} ///:~
```

Il metodo **setSize(width,height)** nella classe **Component** definisce la dimensione della componente (in questo caso del *frame*)

Esempio

```
public class PrimiBottoni {
  public static void main(String[] args) {
    JButton
          b1 = new JButton("Bottone 1"),
          b2 = new JButton("Bottone 2");
    JFrame f = new JFrame("Primi Bottoni");
    Container c = f.getContentPane();
    c.setLayout(new FlowLayout());
    c.add(b1);
    c.add(b2);
    f.setSize(100,100);
    f.setVisible(true);
  }
} ///:~
```

Aree di testo

- Un **JTextField** permette di inserire una casella che può contenere *una sola riga di testo*
- Tra i suoi costruttori troviamo quelli che prendono come argomento
  - Una **String**a: creano una casella contenente quella **String**a come testo
  - Un **int**: creano una casella vuota della dimensione specificata (l'**int** rappresenta il numero di caratteri che può contenere)

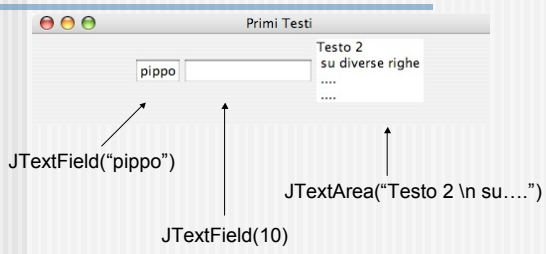
Aree di testo

- Una **JTextArea** è come un **JTextField**, con la differenza che può contenere più di una riga
- Inoltre offre maggiori funzionalità
  - Il metodo **append()**: permette di inserire facilmente testo nell'area
  - In questo modo è addirittura possibile riversare l'output dei nostri programma in un **JTextField** e poi usare le *scrollbar* per navigare l'output (usando Eclipse non ci sono particolari vantaggi, ma programmando con un semplice editor di testi decisamente sì)

Esempio

```
public class AreeTesto {
    public static void main(String[] args) {
    JTextField t1 = new JTextField("pippo");
    JTextField t2 = new JTextField(10);
    JTextArea t3 = new JTextArea("Testo 2 \n su
                        diverse righe \n .... \n ....");
    JFrame f = new JFrame("Primi Testi");
    Container c = f.getContentPane();
    c.setLayout(new FlowLayout());
    c.add(t1);
    c.add(t2);
    c.add(t3);
    f.setSize(500,500);
    f.setVisible(true);
    }}
```

Aree di testo



Controllare il layout

- Il modo con cui sono piazzate le componenti è controllato da un *layout manager*
- **JApplet**, **JFrame**, **JWindow** e **JDialog** sono **Container** che possono contenere e visualizzare componenti (oggetti in **Components**)
- Il metodo **setLayout()** in **Container** permette di scegliere diversi layout manager

BorderLayout

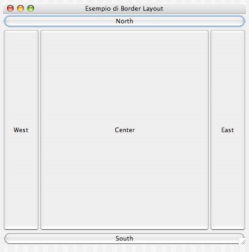
- Il **BorderLayout** può piazzare componenti in 4 regioni e in un'area centrale
- Quando si aggiunge una componente a un contenitore che utilizza questo layout manager, le componenti vengono automaticamente inserite nell'area centrale
- Per specificare una zona diversa, bisogna passare al metodo **add()** un valore costante che specifica la regione d'interesse
  - **BorderLayout.SOUTH**, **BorderLayout.North**, **BorderLayout.EAST**, **BorderLayout.WEST**, **BorderLayout.CENTER** (default)

BorderLayout

- Inoltre, le componenti vengono espanse per riempire tutto lo spazio a disposizione
  - Quelle piazzate non al centro sono espanse in una sola dimensione, quella al centro in entrambe
- Vediamo un esempio

Esempio

```
JFrame f = new JFrame("Esempio di Border Layout");
Container cp = f.getContentPane();
cp.setLayout(new BorderLayout());
cp.add(BorderLayout.NORTH, b1);
cp.add(BorderLayout.SOUTH, b2);
cp.add(BorderLayout.EAST, b3);
cp.add(BorderLayout.WEST, b4);
cp.add(BorderLayout.CENTER, b5);
```



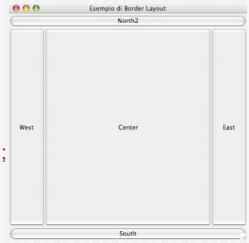
BorderLayout

- Utilizzando un **BorderLayout**, se aggiungiamo altre componenti nelle stesse posizioni di componenti già esistenti, le prime vengono “sovrascritte”
- Vediamo un esempio

Esempio

```
JFrame f = new JFrame("Esempio di Border Layout");
Container cp = f.getContentPane();
cp.setLayout(new BorderLayout());
cp.add(BorderLayout.NORTH, b1);
cp.add(BorderLayout.SOUTH, b2);
cp.add(BorderLayout.EAST, b3);
cp.add(BorderLayout.WEST, b4);
cp.add(BorderLayout.CENTER, b5);
cp.add(BorderLayout.NORTH, b6);
```

b6 ha nome **North2**



FlowLayout

- Questo layout manager consente di aggiungere un qualsiasi numero di componenti, e le sistema posizionandole da sinistra a destra (fino a quando è disponibile spazio in orizzontale), e dal basso in alto
- Inoltre le componenti non vengono espanse per occupare tutto lo spazio disponibile
  - Viene assegnato lo spazio “giusto”
- Vediamo un esempio

Esempio

```
JFrame f = new JFrame("Esempio di Flow Layout");
Container cp = f.getContentPane();
cp.setLayout(new FlowLayout());
cp.add(b1);
cp.add(b2);
cp.add(b3);
cp.add(b4);
cp.add(b5);
cp.add(b6);
cp.add(b7);
```

GridLayout

- Con un **GridLayout** si definisce una griglia e le componenti sono inserite nelle celle della griglia, da sinistra a destra e dall'alto in basso
- Nel costruttore si specifica il numero di righe e di colonne della griglia
- Se nel costruttore sia il numero di righe che il numero di colonne sono diversi da zero, il numero di colonne è ignorato
  - Il numero di colonne è automaticamente determinato in base al numero di elementi inseriti
- Vediamo un esempio

Esempio

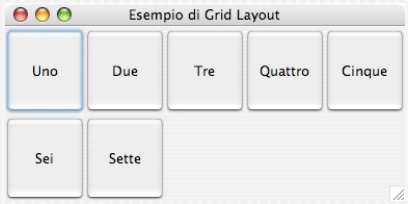
```
JFrame f = new JFrame("Esempio di Flow Layout");
Container cp = f.getContentPane();
cp.setLayout(new GridLayout(4,5));
cp.add(b1);
cp.add(b2);
cp.add(b3);
cp.add(b4);
cp.add(b5);
cp.add(b6);
cp.add(b7);
```

GridLayout

- Specificando a 0 il numero di righe, viene considerato il numero di colonne passato al costruttore
- Vediamo un esempio

Esempio

```
JFrame f = new JFrame("Esempio di Flow Layout");
Container cp = f.getContentPane();
cp.setLayout(new GridLayout(0,5));
cp.add(b1);
cp.add(b2);
cp.add(b3);
cp.add(b4);
cp.add(b5);
cp.add(b6);
cp.add(b7);
```



GridBagLayout

- Con questo layout viene fornita la possibilità di controllare *esattamente* come le regioni della finestra verranno disposte e come vengono riformattate in caso di *ridimensionamento* della finestra stessa
- È il layout manager più complicato
- Viene tipicamente utilizzato nella generazione automatica di codice da parte di costruttori di interfacce
- Una versione leggermente più semplificata è fornita dal **BoxLayout**

Posizionamento assoluto

- È possibile specificare la posizione assoluta delle componenti in questo modo
  - Invocando **setLayout(null)**
  - Invocare **setBounds()** per ogni componente, e passandogli come argomento un rettangolo specificato in coordinate dei pixel

Catturare eventi

- Eseguendo il codice che crea bottoni, ci accorgiamo che cliccando sui pulsanti non accade nulla
- A questo punto entrano in gioco gli eventi (come la pressione di un bottone, e la scrittura di codice da eseguire quando questi accadono)
- Con le *Swing* si tiene separata l'*interfaccia grafica* (ad es. i bottoni) dall'*implementazione* (cosa eseguire quando accade un evento)

Catturare eventi

- Ogni componente *Swing* può riportare tutti gli eventi che accadono
  - Movimento mouse, click su un bottone, ecc.
- Se non si è interessati a qualche evento, semplicemente non si registra il nostro interesse per quell'evento
- In altre parole, bisogna registrare gli eventi di nostro interesse, associando del codice da eseguire quando essi accadono

Catturare eventi -- bottoni

- Nel caso di **JButton**, l'evento che ci interessa registrare è la *pressione del bottone*
- Bisogna invocare il metodo **addActionListener()** di **JButton**
- Questo metodo si aspetta come argomento un oggetto che implementa l'interfaccia **ActionListener** che contiene il solo metodo **actionPerformed()**

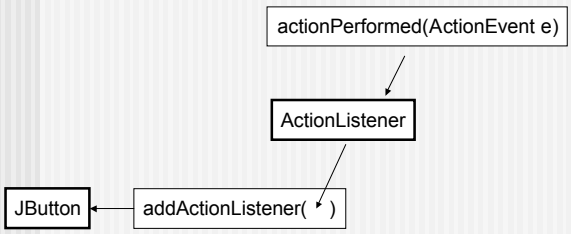
Catturare eventi -- bottoni

- Il metodo **actionPerformed(ActionEvent e)** prende come argomento un **ActionEvent** che descrive l'evento
- **ActionEvent** contiene tutte le informazioni sull'evento e da chi è stato generato

Catturare eventi -- bottoni

- Quindi, per collegare codice a un bottone, bisogna
  - Implementare **ActionListener** in una classe
  - Registrare un oggetto di questa classe presso il **JButton** tramite **addActionListener()**
- Il metodo implementato **actionPerformed()** verrà così eseguito alla pressione del bottone

Catturare eventi -- bottoni



Esempio di schema

```
public class MioFrame {
    JFrame f = new JFrame();
    /* Dichiarazione componenti */
    JButton b = new JButton();
    ….
    /* Definizione di tutti i Listener */
    class MioListener implements ….Listener {….}
    ….
    MioFrame(){
          Container c = f.getContentPane();
          /* Aggiunta componenti c */
          /* Registrare i listener presso le componenti */
          f.setVisible(true);
          f.setSize(100,100);}
public static void main(String[] args){
    new MioFrame();}}
```

- Schema d'esempio per strutturare una semplice interfaccia grafica
- Come si nota, la definizione dei listener avviene tipicamente tramite classi interne

Esempio di schema alternativo

```
public class MioFrame extends JFrame {
    /* Dichiarazione componenti */
    JButton b = new JButton();
    ….
    /* Definizione di tutti i Listener */
    class MioListener implements ….Listener {….}
    ….
    MioFrame(){
          Container c = getContentPane();
          /* Aggiunta componenti c */
          /* Registrare i listener presso le componenti */
          setVisible(true);
          setSize(100,100);}
public static void main(String[] args){
    new MioFrame();}}
```

- Alternativamente, la nostra classe può estendere la classe che definisce il top-level container d'interesse

Esercizio

- Partendo dal codice **PrimiBottoni**, scrivere una classe **PrimiBottoni2** dove
  - Si definiscono due bottoni
  - Si implementa **ActionListener** e quindi il metodo **actionPerformed()** (tipica cosa che viene bene in una classe annidata)
  - In **actionPerformed()** stampare il nome del bottone premuto
    - Con il metodo **getSource()** in **java.util.EventObject** (di cui **ActionEvent** è sottoclasse) determinare l'oggetto che ha causato l'evento
    - Poi, con il metodo **getText()** di **JButton** ricavare il testo associato al bottone

Esercizio

- Una possibile soluzione dell'esercizio precedente prevede la definizione di una classe annidata che implementa **ActionListener**
- Questa è una tipica cosa che si fa bene utilizzando anche una *classe interna anonima*
  - La scelta dell'utilizzo di una classe interna anonima o di una semplice classe interna è a discrezione del programmatore
- Vediamo

Esercizio

- Quindi

```
private ActionListener bl = new ActionListener () {
 public void actionPerformed(ActionEvent e) {
     String name = ((JButton)e.getSource()).getText();
     System.out.println(name);
 }
};
```

Il modello a eventi di *Swing*

- Nel modello a eventi di *Swing* una componente può *generare* un evento
- Ogni tipo di evento è rappresentato da una classe distinta
- Quando un evento è generato, viene ricevuto da uno o più *listeners* (ascoltatori), che agiscono sull'evento
- Quindi la sorgente dell'evento e il posto in cui viene ricevuto possono essere distinti

Il modello a eventi di *Swing*

- Ogni *event listener* è un oggetto di una classe che implementa una particolare interfaccia
- Quindi, tutto quello che bisogna fare è creare un oggetto *listener* e registrarlo presso la componente che genera l'evento
- La registrazione si effettua invocando il metodo **addXXXListener()** nella componente che genera l'evento
  - XXX rappresenta il tipo dell'evento

Il modello a eventi di *Swing*

- Per scoprire quali tipi di eventi una certa componente può gestire è sufficiente esaminare i nomi dei metodi *addListener*
- Quando si implementa una interfaccia listener, l'unica restrizione è che bisogna implementare l'interfaccia appropriata per la componente che può generare l'evento

Eventi e tipi di listener

- Tutte le componenti Swing includono metodi **addXXXListener()** e **removeXXXListener()** per aggiungere e rimuovere i listener appropriati
  - Si nota anche che XXX è il tipo dell'argomento per il metodo
    - Es. **addActionListener(ActionListener a)**

Eventi e tipi di listener

Event, listener interface and add- and remove-methods	Components supporting this event
ActionEvent **ActionListener** **addActionListener( )** **removeActionListener( )**	**JButton, JList, JTextField, JMenuItem** and its derivatives including **JCheckBoxMenuItem, JMenu,** and **JpopupMenu.**
AdjustmentEvent **AdjustmentListener** **addAdjustmentListener( )** **removeAdjustmentListener( )**	**JScrollbar** and anything you create that implements the **Adjustable interface.**
ComponentEvent **ComponentListener** **addComponentListener( )** **removeComponentListener( )**	***Component** and its derivatives, including **JButton, JCheckBox, JComboBox, Container, JPanel, JApplet, JScrollPane, Window, JDialog, JFileDialog, JFrame, JLabel, JList, JScrollbar, JTextArea,** and **JTextField.**
ContainerEvent **ContainerListener** **addContainerListener( )** **removeContainerListener( )**	**Container** and its derivatives, including **JPanel, JApplet, JScrollPane, Window, JDialog, JFileDialog,** and **JFrame.**
FocusEvent **FocusListener** **addFocusListener( )** **removeFocusListener( )**	**Component** and **derivatives***.
KeyEvent **KeyListener**	**Component** and **derivatives***.

Event, listener interface and add- and remove-methods	Components supporting this event
addKeyListener( ) **removeKeyListener( )**	
MouseEvent (for both clicks and motion) **MouseListener** **addMouseListener( )** **removeMouseListener( )**	**Component** and **derivatives***.
MouseEvent[8] (for both clicks and motion) **MouseMotionListener** **addMouseMotionListener( )** **removeMouseMotionListener( )**	**Component** and **derivatives***.
WindowEvent **WindowListener** **addWindowListener( )** **removeWindowListener( )**	**Window** and its derivatives, including **JDialog, JFileDialog, and JFrame.**
ItemEvent **ItemListener** **addItemListener( )** **removeItemListener( )**	**JCheckBox, JCheckBoxMenuItem, JComboBox, JList,** and anything that implements the **ItemSelectable interface.**
TextEvent **TextListener** **addTextListener( )** **removeTextListener( )**	Anything derived from **JTextComponent,** including **JTextArea** and **JTextField.**

- Principali eventi e listener associati

Eventi e tipi di listener

- Si nota come ogni componente gestisce supporta solo alcuni tipi di eventi
- Alcune interfacce come ActionListener prevedono l'implementazione di un solo metodo
- Altre hanno più di un metodo
  - In questo caso bisogna implementarli tutti, anche se non servono tutti

Eventi e tipi di listener

- Ad esempio in **MouseListener** ci sono sia **MouseClicked()** e **MouseEntered()** e bisogna implementarli entrambi anche se a noi serve catturare solo il click
- Per ovviare a questo problema alcune delle interfacce sono fornite di *adapters* che forniscono implementazioni con metodi vuoti
  - Quindi, si può semplicemente ereditare dagli *adapters* e riscrivere solo i metodi di interesse

Adapters

Listener interface w/ adapter	Methods in interface
ActionListener	actionPerformed(ActionEvent)
AdjustmentListener	adjustmentValueChanged(AdjustmentEvent)
ComponentListener ComponentAdapter	componentHidden(ComponentEvent) componentShown(ComponentEvent) componentMoved(ComponentEvent) componentResized(ComponentEvent)
ContainerListener ContainerAdapter	componentAdded(ContainerEvent) componentRemoved(ContainerEvent)
FocusListener FocusAdapter	focusGained(FocusEvent) focusLost(FocusEvent)
KeyListener KeyAdapter	keyPressed(KeyEvent) keyReleased(KeyEvent) keyTyped(KeyEvent)
MouseListener MouseAdapter	mouseClicked(MouseEvent) mouseEntered(MouseEvent) mouseExited(MouseEvent) mousePressed(MouseEvent) mouseReleased(MouseEvent)
MouseMotionListener MouseMotionAdapter	mouseDragged(MouseEvent) mouseMoved(MouseEvent)
WindowListener WindowAdapter	windowOpened(WindowEvent) windowClosing(WindowEvent) windowClosed(WindowEvent) windowActivated(WindowEvent) windowDeactivated(WindowEvent) windowIconified(WindowEvent) windowDeiconified(WindowEvent)
ItemListener	itemStateChanged(ItemEvent)

Esercizio -- eventi multipli

- Creare un frame che contenga tre bottoni e una area di testo (**JTextField**) inizializzata a 30 colonne
- Implementare **ActionListener** e **MouseListener** in modo che nell'area di testo venga visualizzato il tipo dell'evento generato (utilizzare il metodo **paramString()** degli eventi)
- Registrare **ActionListener** e **MouseListener** per i primi due bottoni. Per il terzo bottone registrare solo **ActionListener**
- Nota: utilizzare *classi interne anonime* per l'implementazione delle interfacce legate agli eventi

Principali componenti

- A questo punto faremo una rapida carrellata delle principali componenti presenti nelle *Swing*
- Per maggiori informazioni fare sempre riferimento alla documentazione di Java offerta dalla Sun

Bottoni

- Le *Swing* includono un certo numero di bottoni di diverso tipo
- Tutti sono sottoclassi di **AbstractButton**
  - **JButton**, **BasicArrowButton**, **JCheckBox**, **JRadioButton**

ButtonGroup

- Per creare bottoni radio (**JRadioButton**) che siano selezionabili in maniera esclusiva, bisogna aggiungerli a un *gruppo di bottoni*
- In generale, qualsiasi **AbstractButton** può essere aggiunto a un **ButtonGroup** (sottoclasse di **Object**) con il metodo **add(AbstractButton ab)** di **ButtonGroup**

Esempio -- ButtonGroup

```
JRadioButton
        b1 = new JRadioButton("Selezione 1"),
        b2 = new JRadioButton("Selezione 2");
ButtonGroup bg = new ButtonGroup();
JFrame f = new JFrame("Primi Bottoni");
Container cp = f.getContentPane();
cp.setLayout(new FlowLayout());
bg.add(b1);
bg.add(b2);
cp.add(b1);
cp.add(b2);
```

Icone

- **Icon** è una interfaccia di *Swing* che serve a gestire icone (piccole immagini) utili a decorare elementi di una interfaccia grafica
- Una classe che la implementa è **ImageIcon**, a cui è sufficiente passare il nome di un file di una immagine per generare una icona
- È possibile usare **Icon** in una **JLabel** o in qualsiasi classe che eredita da **AbstractButton**
  - Serve per aggiungere immagini a una etichetta o a uno qualsiasi dei bottoni

Icone

- Per aggiungere una immagine a una **JLabel** è sufficiente invocare il costruttore che prende tra gli argomenti una **Icon**

```
new JLabel(new
   ImageIcon(nomeFileImmagine.gif))
```

- In maniera del tutto simile si aggiungono icone ai bottoni
  - Invocando i costruttori giusti

I suggerimenti (*tool tips*)

- La classe **JComponent** (da cui derivano quasi tutte le componenti *Swing*) contiene il metodo **setToolTipText(String)**
- Quando il mouse si sposta (per un certo periodo di tempo) su una componente per cui è stato impostato un suggerimento, appare un piccolo rettangolo contenente il suggerimento indicato dalla **String**a

Bordi

- **JComponent** contiene il metodo **setBorder(Border)** che consente di aggiungere bordi a qualsiasi componente visibile
- **Border** è una interfaccia
- Alcune sue implementazioni si trovano nelle sottoclassi di **AbstractBorder**
  - Esse implementano vari tipi di bordi da poter aggiungere alle componenti

JScrollPane

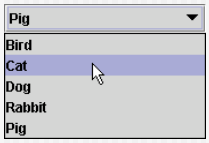
- **JScrollPane** fornisce una visione con barre di scorrimento di una componente *lightweight*
- È sufficiente passare al suo costruttore una componente, e vengono create le barre di scorrimento che sono attivate non appena si rendono necessarie

JTextPane

- Permette di gestire aree di testo
- Offre funzionalità più sofisticate rispetto a **JTextArea**

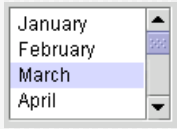
Combo boxes (liste drop-down)

- **JComboBox** serve per generare delle liste a discesa da cui si può selezionare una sola delle scelte offerte nella lista
- Gli elementi vengono aggiunti con il metodo **addItem(Object)**



JList

- **JList**, a differenza di **JComboBox**, crea delle liste che non sono a discesa (occupano un certo numero di linee sullo schermo e non cambia)
- Inoltre permette selezioni multiple degli elementi nella lista



JTabbedPane

- **JTabbedPane** definisce delle finestre con separatori

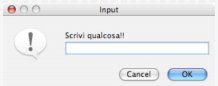
Finestre di dialogo

- **JOptionPane** serve per creare finestre di dialogo
- È possibile crearne di sofisticate, ma quelle più comuni sono create con i seguenti metodi **statici**
  - **showConfirmDialog()**: dialogo che chiede conferma (si, no, annulla)
  - **showInputDialog()**: chiede l'immissione di qualche input
  - **showMessageDialog()**: visualizza un messaggio di testo
  - **showOptionDialog()**: unifica le tre

Finestre di dialogo

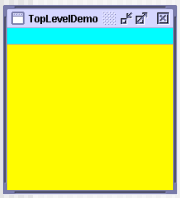
showConfirmDialog()

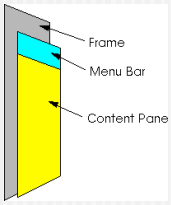
showMessageDialog()



showInputDialog()

JMenu



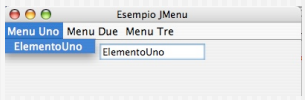


- Ecco come le barre dei menu si collocano tipicamente all'interno di un top-level container

JMenu

- Ogni componente in grado di contenere un menù ha un metodo **setJMenuBar()** che accetta come argomento un **JMenuBar**
  - È possibile avere solo un **JMenuBar** in una componente
- Si aggiungono **JMenu** a **JMenuBar** e **JMenuItem** a **Jmenu**
- Ogni **JMenuItem** può avere un **ActionListener** registrato
  - Il relativo evento viene generato quando quell'item viene selzionato

Esempio




```
ActionListener al = new ActionListener(){
public void actionPerformed(ActionEvent e){
    String name=((JMenuItem)e.getSource()).getText();
    t.setText(name);
}};
f.setJMenuBar(mb);
JMenu jm1=new JMenu("Menu Uno");
JMenu jm2=new JMenu("Menu Due");
JMenu jm3=new JMenu("Menu Tre");
mb.add(jm1);                mb.add(jm2);                mb.add(jm3);
JMenuItem jmi1=new JMenuItem("ElementoUno");
JMenuItem jmi2=new JMenuItem("ElementoDue");
JMenuItem jmi3=new JMenuItem("ElementoTre");
jm1.add(jmi1);              jm2.add(jmi2);              jm3.add(jmi3);
jmi1.addActionListener(al);                 jmi2.addActionListener(al);
jmi3.addActionListener(al);
```

Finestre di dialogo

- Le finestre di dialogo sono finestre che sono create per dialogare con l'utente
- Sono simili ai frame visti finora
- La differenza è che la loro costruzione può essere ordinata da un'altra finestra (frame o dialogo)
- Quindi, tra i suoi costruttori ne troviamo alcuni che prendono come parametro anche un *owner*, cioè la finestra da cui parte il dialogo
- Le finestre di dialogo sono gestite dalla classe **JDialog**
  - È uno dei top-level container

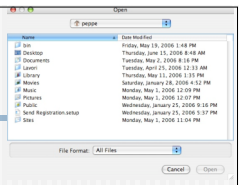
JDialog

- In sostanza sono del tutto simili ai frame visti finora
- Dopo essere stato creato, viene reso visibile invocando il metodo **setVisible()**
- Il metodo **dispose()** chiude il dialogo

Esercizio

- In apparenza complesso
- Partiamo dall'esercizio in cui poniamo due bottoni su un frame e poi li colleghiamo a un **actionListener** (che stampa il nome dei bottoni)
- Questa volta alla pressione dei bottoni deve comparire una finestra di dialogo con un bottone
  - Il nome del bottone è "Cancella" + nomeBottonePremuto
- Alla pressione del bottone si chiude la finestra di dialogo

JFileChooser



- **JFileChooser** serve a creare finestre di dialogo per navigare il file system
- Ci sono vari costruttori a seconda del tipo di view del file system che si vuole ottenere
- I metodi **showOpenDialog()** (in figura) e **showSaveDialog()** aprono delle finestre con bottoni per aprire e salvare un file, rispettivamente

Swing e HTML

- Qualsiasi componente che prende testo in input, può prendere anche testo HTML che sarà formattato conseguentemente

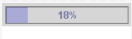
```
JButton b = new JButton("<html><b>Bottone");
```

- Nota: non si è costretti a inserire le normali chiusure dei tag HTML

Sliders e progress bar

- **JSlider** permette la definizione degli slider per inserire valori numerici

- **JProgressBar** permette la definizione di barre che visualizzano dati in percentuale relativa da "vuoto" a "pieno"

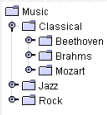


- Per entrambi i costruttori possono prendere il valore minimo, massimo, e corrente

Sliders e progress bar

- Il cuore di queste due classi sono i metodi **getMinimum()**, **getMaximum()** e **getValue()**
  - Servono a recuperare il valore minimo, massimo e corrente di sliders e progress bar
  - Esistono anche i relativi metodi set

Alberi



- La classe **JTree** offre un modo di visualizzare un insieme di dati in maniera gerarchica
  - Il costruttore più semplice prende un array di oggetti
- È una delle librerie più vaste nelle Swing
- Collegate a questa classe è l'interfaccia **TreeNode**, implementata in **DefaultMutableTreeNode,** che descrive i nodi di una gerrachia

Altre componenti grafiche

- Esistono numerose altre componenti grafiche offerte dalle *Swing*
- Il modo migliore per conoscerle e imparare ad usarle è di fare riferimento ala documentazione e ai tutorials presenti sul sito *http://java.sun.com*

Editori grafici di GUI

- Eclipse offre un plugin per progettare GUI graficamente
- Il nome del plugin è *Visual Editor* (*VE*)
- Vediamo come installare plugin in Eclipse

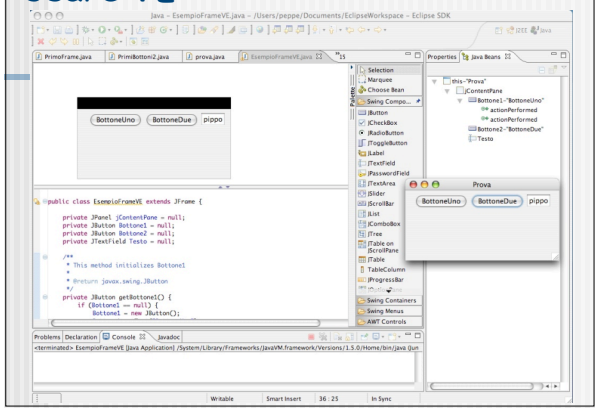
Plug-in di Eclipse

- Per scaricare e installare plug-in in Eclipse utilizzare sempre l'*update manager*
  - Dal sito di Eclipse, trovare la URL da cui scaricare il plug-in di interessa tramite l'update manager
  - In eclipse, da Help, Software Updates, Find and Install, selezionare *Search for new features to install*
  - Selezionare *New Remote Site* e inserire la URL trovata in precedenza

Usare VE

- Per utilizzare il plugin VE di Eclipse è sufficiente
  - Selezionare (dopo aver creato un progetto) *New, Other*
  - A questo punto selezionare l'opzione *Swing*
  - Scegliere il tipo di top-level container da cui iniziare
  - Dopo la conferma della scelta, viene avviato VE
  - Il suo utilizzo è abbastanza intuitivo
  - Sulla destra è situata la *Palette* da cui selezionare le componenti
  - Dal menù di contesto (pulsante destro del mouse) delle componenti è possibile aggiungere la gestione degli eventi

Usare VE



Java

creare finestre

fine