INTERFACCE GRAFICHE IN JAVA

[Per corso su Java vedere http://java.sun.com/docs/books/tutorial/]

Introduzione

- Java prevede nella sua API librerie di classi da usare in un programma per creare e gestire GUI.
- Parte delle Java Foundation Classes (JFC), librerie di classi standard Java.
- In Java sia le finestre che gli elementi/dispositivi di interfaccia sono chiamati **componenti**.
- Allo stato attuale si hanno due librerie di componenti:
  - Java AWT -- Java Swing

Note storiche

Il primo a nascere e' Java Abstract Widget Toolkit (AWT), gia' in Java 1.0, migliorato in Java 1.1. Invariato nelle versioni successive.

- componenti di interfaccia meno sofisticati (es. bottoni possono avere etichette solo testuali)
- AWT components sono "heavyweight"
- piu' limitato nella varieta' di componenti di interfaccia forniti
- Java Swing e' presente a partire da Java 1.2
  1. maggiore varieta' di compomenti di interfaccia
  2. componenti piu' sofisticati (es. posso avere bottoni etichettati con testo e/o immagini)
  3. Swing components sono "lightweight" (piu' efficienti)
  4. aspetto riconfurabile (la stessa interfaccia puo' essere visualizzata con aspetto stile Windows o stile Motif ecc.)
  5. funzioni avanzate per grafica 2D
- AWT e Swing hanno struttura e modo d'uso molto simile.
- Le classi di componenti di AWT hanno equivalente in Swing.
- Componenti Swing si riconoscono perche' il nome inizia per "**J**" (es: Button e JButton).

Metodo orientato a oggetti

- Essendo Java un linguaggio object-oriented, tutto e' classe:
  - ogni tipo di componente di interfaccia: finestre, dispositivi...
  - ogni informazione accessoria associata ad una componente di interfaccia:
    - fonts, layout managers che controllano il posizionamento dei dipositivi in una finestra...
  - ogni tipo di evento, le callback per gli eventi: per definire una callback devo definire una classe "listener" (=ascoltatrice) del tipo di evento che mi interessa

- Le classi sono organizzate in gerarchie di ereditarieta' secondo paradigma object-oriented.

Schema di un programma che realizza interfaccia in Java - Esempio

- Nell'esempio, l'interfaccia consta di una finestra contenente un bottone e un'etichetta posizionati uno sopra l'altro.
- L'etichetta mostra il numero di volte in cui il bottone e' stato azionato.
- Quando utente aziona il bottone, l'etichetta si incrementa di una unita'.

VERSIONE AWT

```
import java.awt.*;
import java.awt.event.*;
public class AWTExample extends Frame
{
    private static String labelPrefix = "Number of clicks: ";
    private int numClicks = 0;
    Label label = null;
    Button button = null;
    public AWTExample()
    {
        label = new Label(labelPrefix + "0    ");
        button = new Button("Click me!");
        button.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent e)
            {   numClicks++;
                label.setText(labelPrefix + numClicks);
            }
        });
        setLayout(new GridLayout(2, 1));
        add(button);
        add(label);
        addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent e)
            {   System.exit(0);
            }
        });
        pack();
        setVisible(true);
    }

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

VERSIONE SWING

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

public class SwingExample extends JFrame
{
    private static String labelPrefix = "Number of clicks: ";
    private int numClicks = 0;
    JLabel label = null;
    JButton button = null;

    public SwingExample()
    {
        label = new JLabel(labelPrefix + "0    ");
        button = new JButton("Click me!");
        button.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent e)
            {   numClicks++;
                label.setText(labelPrefix + numClicks);
            }
        });
        Container panel = getContentPane();
        panel.setLayout(new GridLayout(2, 1));
        panel.add(button);
        panel.add(label);
        addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent e)
            {   System.exit(0);
            }
        });
        pack();
        setVisible(true);
    }

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

Filosofia generale per creazione di una interfaccia

1. Prima di tutto occorre importare i packages necessari per la gestione e l'utilizzo di particolari classi (swing, etc…)
2. Quindi stabilire finestra top-level, ovvero un FRAME che è padre di tutta l'interfaccia
3. Stabilire il contenuto dell'interfaccia e come organizzarlo nel FRAME, ovvero se creare pannelli e per ognuno di questi gestire il Layout…
4. Stabilire gestione degli eventi nelle componenti, associando un ascoltatore di eventi per ogni oggetti che deve essere "animato"
5. Far partire l'interfaccia, mediante il metodo main() e visualizzando il FRAME principale.

Importare i packages

- Abbiamo visto che le JFC sono divise in packages.
- Abbiamo anche visto che i moduli principali di AWT sono: java.awt e java.awt.event.
- Il modulo principale di Swing: **javax.swing**,
- Swing non e' altro che un'estensione di AWT.

Stabilire finestra top-level

- AWTExample non è altro che una sottoclasse di Frame
- Analogamente nel caso Swing SwingExample e' sottoclasse di JFrame.
- I metodi costruttori delle classi gestiscono i punti 3,4,5: in questo caso infatti, l'interfaccia crea se stessa e parte.
- Il main semplicemente esegue la costruzione di un oggetto della classe.

Stabilire il contenuto della finestra

- Stabilendo il contenuto della finestra occorre aggiungere le componenti che ci sono necessarie.
- Occorre, inoltre, stabilire il layout manager dei contenitori usati
- Nel caso di AWT: li aggiungo direttamente nel Frame (che e' un contenitore).
- In Swing: li aggiungo nel "pannello di contenuto" associato al JFrame.
- La relazione (gerarchia) di contenimento ha la seguente rappresentazione ad albero:

VERSIONE AWT:

```
        Frame  (finestra principale, contenitore)
          |
      +---+-----+
      |         |
  JButton    JLabel     (bottone ed etichetta)
```

VERSIONE SWING:

```
        JFrame  (finestra principale,
                                contenitore top-level)
          |
     content panel  (contenitore intermedio)
          |
      +---+-----+
      |         |
  JButton    JLabel     (bottone ed etichetta)
```

- Si osservi che, nel caso di una interfaccia piu' compelssa si potrebbero avere piu' livelli nella gerarchia di contenimento.
- Il layout manager stabilisce come sono posizionati i componenti dentro a un contenitore.
- Si consideri l'utilizzo di un layout a griglia con 2 righe e 1 colonna.
- L'istruzione pack fa si' che il contenuto della finestra venga compattato per occupare il minimo spazio necessario.

Stabilire gestione degli eventi

- Il bottone è responsabile della gestione "conta dei click" e deve modificare l'etichetta della label.
- La finestra principale (con gli usuali bottoni) gestisce l'evento "chiusura" della finestra dicendo al programma di terminare.
- Nella fase di gestione dell'evento "attivazione" (classe Actionevent) e' necessario definire una sotto-classe di ActionListener che implementi il metodo ActionPerformed eseguendo l'operazione desiderata.
- Inoltre occorre creare un oggetto di tale classe ed associarlo al bottone. In pratica io associo un oggetto "ascoltatore" di eventi di azione (ActionEvent) al botton e **delego** l'ascoltatore di fare le operazioni necessarie una volta che il bottone è premuto.
- Per evento "chiusura" (classe Windowevent) della finestra potrei fare una cosa simile.

- Nel caso dell'esempio vi è una differenza: qui definisco sottoclasse di WindowAdapter invece che di WindowListener perche' intendo riferinire solo il metodo windowClosing (WindowListener ha anche altri metodi).
- Esiste un adapter per ogni classe listener che prevede piu' di un metodo.
- Le classi listener con un metodo solo non hanno bisogno di adapter).

Far partire l'interfaccia

- Perché l'interfaccia possa essere visualizzata occorre un metodo che renda visibile il Frame principale.
- L'istruzione **setVisible** mappa sullo schermo la finestra top-level e tutto il suo contenuto.
- Da questo momento si innesca automaticamente il "ciclo degli eventi".

Componenti Java

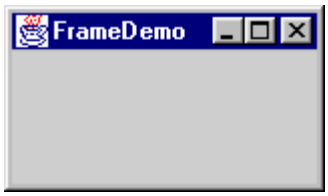
Nella strutturazione di una interfaccia entrano grossomodo in gioco tre diversi tipi di componenti:

- **Le Finestre top-level** che sono gestite dal window manager
- **I Dispositivi** incaricati di mostrare informazioni o catturare un certo input in un certo istante, o in una certa situazione.
  - Tali dispositivi sono collocati dentro le finestre mediante una gerarchia di **contenitori intermedi** (idea delle scatole cinesi)
- Tutti gli elementi in Java sono detti **componenti** e sono classi che ereditano dalla classe base Component (oppure MenuComponent) in AWT, e da JComponent in Swing.
- Ogni contenitore ha un layout manager che stabilisce in che modo gli oggetti devono esser dislocati al suo interno.

Contenitori in Java

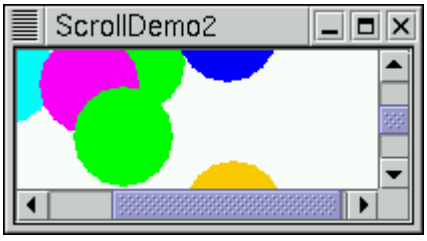
Contenitori Top-level

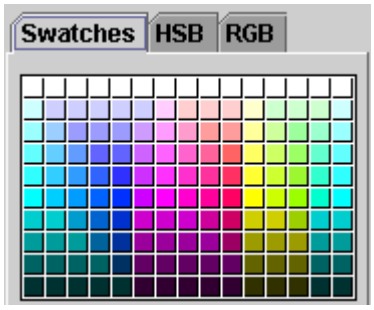
- Realizzati come finestre top-level nell'ambiente in cui sara' eseguito il programma Java.
- Frame / JFrame: finestra primaria di un'applicazione
- Dialog / JDialog: finestra secondaria di dialogo



Contenitori intermedi

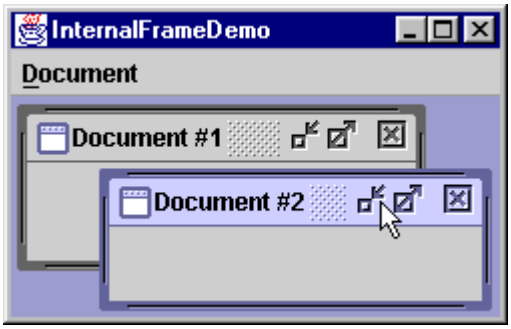
- Contenitori di uso generale:
- Panel / JPanel: pannello a scomparto unico
- ScrollPane / JScrollPane: pannello a scomparto unico con barre di scorrimento orizzontali e/o verticali
- JSplitPane: pannello bipartito in orizzontale o in verticale
- JTabbedPane: pannello a schedario
- JToolBar: barra orizzontale o verticale contenente componenti (in genere bottoni con etichette grafiche) che puo' essere trascinata e posizionata a piacere





Contenitori per usi speciali:

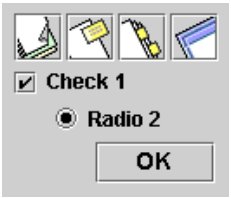
- Menubar / JMenuBar: barra di menu' associata ad un frame, puo' contenere componenti di classe Menu / JMenu (ved. piu' avanti)
- JInternalFrame: sotto-finestra applicativa (finestra di documento), compare all'interno della finestra principale
- JDesktopPane: pannello che simula una desktop all'interno di una finestra, usato per contenere degli internal frames
- JLayeredPane: pannelli a livelli (vi posso collocare oggetti assegnando ad ogni oggetto una profondita', e gli oggetti sono disegnati sovrapposti fra loro in base alla profondita'



Dispositivi in Java

Dispositivi di controllo

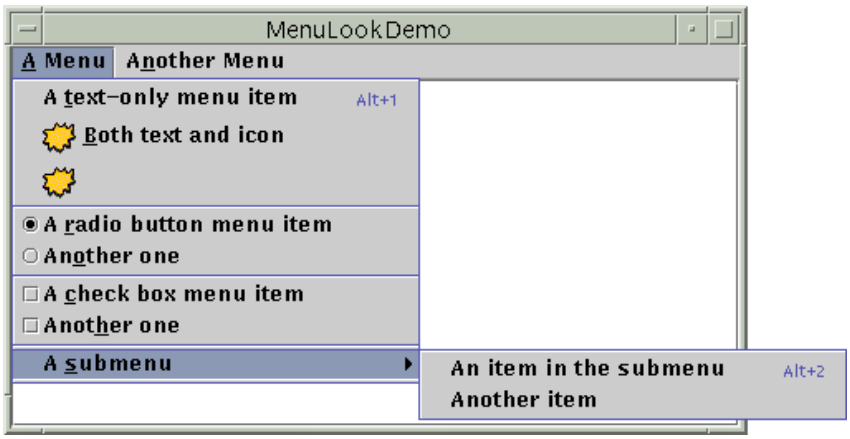
- In grado di ricevere input e mostrare informazioni di stato.
- Vari tipi di bottoni
- Button / JButton: bottone di comando
- Checkbox / JCheckBox: bottone a due stati
- Gruppo di bottoni radio (bottoni a due stati che lavorano in modo mutuamente esclusivo), realizzati in modo diverso in AWT e Swing:
  - in Swing classe JRadioButton
  - in AWT si usa classe Checkbox



Vari tipi di menu' e di voci di menu'

- Le classi di voci di menu' rispecchiano le classi di bottoni:
  - MenuItem, CheckBoxMenuItem, in AWT,
  - JMenuItem, JCheckBoxMenuItem, JRadioButtonMenuItem in Swing
- Classi per i menu':
  - Menu / JMenu: menu' semplice, creo le voci a parte e le aggiungo al menu' per popolare la tendina.

- Puo' essere usato come voce di un altro menu', realizzando in tal modo menu' a cascata. Puo' essere aggiunto ad una Menubar / JMenuBar.
- PopupMenu / JPopupMenu: menu' contestuale a scomparsa (pop-up).



Altri dispositivi

- List / JList: liste di alternative da cui posso selezionarne una
- JSlider: potenziometro scorrevole
- TextField / JTextField: campo di input testuale a linea singola
- JPasswordField: campo di input testuale che non mostra la stringa inserita

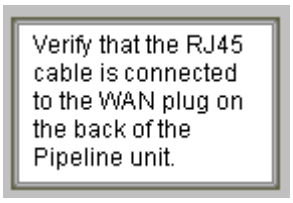
Dispositivi per mostrare informazioni

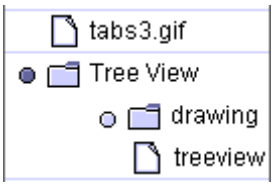
- Non sensibili a input:
- Label, JLabel: etichetta testuale in AWT, testuale e/o grafica in Swing
- JProgressBar: mostra avanzamento nell'esecuzione di un'operazione
- JToolTip: messaggi di aiuto in sovra-impressione

Dispositivi per mostrare / acquisire informazioni formattate

- Mostrano informazioni complesse, possono essere anche editabili:
- TextArea / JTextArea: area di testo su piu' linee
- JTable: tabella organizzata a righe e colonne

- JTree: informazioni organizzate ad albero, es: per navigare nel file system





Dispositivi con funzioni speciali

- Per scegliere un file nel file system:
  - In AWT ho una speciale finestra di dialogo: FileDialog
  - In Swing ho componente JFileChooser che puo' essere collocata in qualsiasi contenitore
- Per scegliere un colore: JColorChooser

Layout Management

- Processo di stabilire automaticamente dimensioni e posizione delle componenti all'interno di un contenitore.
- Ogni contenitore ha un layout manager.
- Potrei lavorare senza layout manager, ma allora dovrei fornire posizione assoluta di ogni componente all'interno del contenitore, ed avrei problemi quando il contenitore top-level viene redimensionato.

Classi di layout managers

- FlowLayout: i componenti sono messi uno dopo l'altro orizzontalmente finche' ci stanno, poi si va a capo e si forma un'altra linea, ecc.
- BoxLayout: i componenti sono messi in fila (orizzontalmente o verticalmente)
- BorderLayout: il contenitore e' diviso in 5 zone (North South East West Center)

```
+---------------------------+
|           NORTH           |
+------+------------+-------+
| EAST |   CENTER   | WEST  |
+------+------------+-------+
|           SOUTH           |
+---------------------------+
```

- GridLayout: tabella con un certo numero di righe e colonne.

- GridBagLayout: tabella dove un componente puo' occupare piu' righe e/o piu' colonne
- CardLayout: permette a due componenti di coesistere su un'area essendo visualizzati in alternativa uno all'altro

Gestione degli eventi in Java

- Classi per gli **eventi**
- Classi **event listener**, che definiscono **modi di reagire** a eventi di una certa classe
- Parte di AWT, usate anche in Swing.

Classi di eventi

Gli **eventi** sono classificati a seconda della loro causa scatenante, che puo' essere:

- un'azione fisica: cliccare mouse, muovere mouse, premere tasto su tastiera...
- un'azione "logica", la cui corrispondenza con azione fisica dipende dal sistema su cui il programma Java sara' eseguito: es. evento "selezione" su un sistema puo' coincidere con doppio click del tasto sinistro del mouse, su un altro sistema puo' essere effettuata in modo diverso

Classi di eventi hanno nomi del tipo XXXEvent dove XXX e' la causa. Es: MouseEvent...

- Ogni classe di componente e' una potenziale sorgente di certe classi di eventi. Es: un bottone puo' originare eventi di classe "azionamento" (classe ActionEvent).

Event listeners

- Posso registrare un event listener per ricevere eventi di una certa classe da una certa componente dell'interfaccia.
- Vi e' un event listener per ogni classe di evento: se la classe evento e' XXXEvent, la classe listener e' XXXEventListener.
- La classe listener prevede uno o piu' metodi che scattano quando il listener riceve un evento di quella classe.

- In realta' un listener non e' una classe, ma un'interfaccia (come una classe, ma l'implenentazione dei metodi non e' definita).
- Stabilita la classe di eventi che voglio catturare, devo creare una sottoclasse del listener corrispondente, fornendo il codice dei suoi metodi (essendo i listener interfacce e non classi, non si dice "sottoclasse" bensi' "implementazione").
- Poi associo il listener alla componente sulla quale voglio catturare gli eventi.

ESEMPIO: **VOGLIO CATTURARE AZIONAMENTO DI UN BOTTONE.**

- La classe di evento e' ActionEvent
- Creo sottoclasse di ActionListener che definisce implementazione del metodo ActionPerformed con le operazioni che intendo far fare al bottone quando azionato
- Creo il bottone (istanza della classe JButton), creo un action listener (istanza della sottoclasse di ActionListener da me definita), associo questo action listener al bottone
- Quando l'utente azionera' il bottone, scattera' l'action listener che eseguira' le operazioni.
- Questo paradigma si chiama delegation perche' il bottone delega l'azione da eseguire ad un altro oggetto (il listener).
- Posso associare lo stesso event listener a piu' componenti nell'interfaccia: questi reagiranno allo stesso evento nello stesso modo.
- Posso associare piu' event listener allo stesso oggetto: questo avra' piu' reazioni.
- Il fatto che i listeners siano interfacce e non classi (cioe' che non implementino i metodi) implica che per i listener che hanno piu' metodi devo fornire un'implementazione di tutti i metodi, inclusi quelli che non mi interessano (l'implementazione di questi non fara' nulla: {})
- Per comodita' sono forniti **adapters** che forniscono implementazione standard di un listener (dove tutti i metodi non fanno nulla): creo sottoclasse dell'adapter ridefinendo solo i metodi che mi interessano.

ESEMPIO:

- La classe WindowEvent corrisponde a eventi sulle finestre top-level: apertura / chiusura, iconificazione / deiconificazione...
- La classe WindowListener ha un metodo per ciascuno degli eventi di cui sopra.
- Se voglio reagire solo all'evento di chiusura:
  - uso WindowListener e scrivo codice per tutti i metodi (quelli che non interessano non faranno nulla)
  - oppure uso WindowAdapter e scrivo codice solo per il metodo che mi interessa

Generalita' sui componenti Java

- Occorre osservare che tutte le componenti di JAVA hanno in comune un insieme di possibili **metodi** che permettono la gestione della lettura e l'assegnazione di certe proprieta' generali alle variabili delle componenti stesse.
- Tutte le classi componenti hanno inoltre in comune un insieme di **eventi** che possono catturare.
- Le caratteristiche comuni sono tutte contenute nella classe base Component (in AWT) e JComponent (in Swing).
- Inoltre JComponent deriva per ereditarieta' da Component, per cui JComponent ha tutti i metodi di Component ed altri in piu'.

Metodi comuni a tutti i componenti

VISIBILITA'

- Per visualizzare una componente a schemro è necessario mapparla. La mappatura si fa con il metodo void setVisible(boolean); con argomento true per mappare e false per demappare
- E possibile controllare se una certa componente è, in un certo istante mappata con boolean isVisible()
- Le Componenti **NON** sono mappate automaticamente. Il che vuol dire che se io non faccio setVisible() della frame principale in un progetto non riesco a vedere nulla.
- Occorre mappare i componenti top-level (frame, dialoghi), il che provoca automaticamente mappatura di tutti i sotto-componenti.

ABILITAZIONE

- Per abilita' si intende capacita' di un componente ad interagire con l'utente.
  - void setEnabled(boolean); con argomento true per abilitare e false per disabilitare
  - boolean isEnabled() per controllare se abilitata

COLORI E FONT

- void setFont(Font) e il corrispondente Font getFont().

  Esempio:
  comp.setFont("Helvetica",Font.PLAIN,14)
  assegna font Helvetica normale a dimensione 14 punti.

- void setBackground(Color) e void setForeground(Color) con i corrispondenti "get".

  Esempio:
  comp.setBackground(new Color(0.6,0.3,1.0))
  assegna colore di sfondo mediante le sue componenti Red Green Blue.

- Classe Color ha anche costanti che denotano i colori piu' diffusi, es: Color.red,...

BORDI (SOLO SWING)

- I bordi possono servire solo come ornamento ma anche per raggruppare assieme dispositivi logicamente correlati, aggiungendo, ad esempio, un titolo.
  - void setBorder(Border) assegna il bordo della componente (di default non c'e' bordo)
  - BorderFactory e' una classe che fornisce metodi per creare istanze della classe Border da usare come argomenti in setBorder
- Ci vari tipi di bordi, in un bordo si possono inserire titoli e/o grafica. Esempi:

```
comp.setBorder(
```

```
            BorderFactory.createLineBorder(Color.black));
            //crea bordo a linea semplice,
            // nero, senza titolo

      b1 = BorderFactory.createLineBorder(Color.black);
      b2 = BorderFactory.createTitledBorder(b1,"titolo",

   TitleBorder.CENTER,TitleBorder.BELOW_BOTTOM);
      comp.setBorder(b2);
            // crea bordo come prima piu' titolo
            // posizionato al centro sotto il riquadro
            // incorniciato dal bordo.
```

MESSAGGI IN SOVRA-IMPRESSIONE

- Nel caso di implementazione di interfaccia con SWING è possibile far comparire una riga di spiegazione a comparsa e scomparsa, che aiuti l'utente (un po' come succede in windows).
- Tale hint è associabile a qualsiasi componente.
- Appare quando utente si ferma col mouse sopra la componente.

ESEMPIO:

```
comp.setToolTipText("Questo e' il messaggio");
```

DIMENSIONI E ALLINEAMENTI

- Ogni componente in Java possiede ***tre dimensioni*** (minima, massima, preferita) di cui il layout manager del contenitore in cui la componente si trova *puo'* tenere conto.
- Ogni componente possiede inoltre ***due allineamenti*** (orizzontale, verticale) di cui il layout manager puo' tenere conto.
- Queste dimensioni e allineamenti sono stabiliti dal sistema in modo generalmente sensato
  - ESEMPIO: Per un bottone la dimensione preferita e' quella grande a sufficienza per mostrare tutta l'etichetta
- Java AWT fornisce metodi solo per leggere queste dimensioni e allineamenti (nome che inizia per "get"), mentre non posso assegnarli esplicitamente.
- **Java Swing fornisce anche metodi per assegnarle** (nome che inizia per "set").
  - void setMinimumSize(Dimension) mi setta la dimensione minima della componente relativa ad un valore fornito dall'oggetto Dimension.
  - Dimension getMinimumSize() restituisce un oggetto Dimension da cui posso prelevare le dimensioni della componente.
  - la stessa cosa per MaximumSize e PreferredSize

- una dimensione si crea con new Dimension(w,h) dove gli argomenti sono larghezza ed altezza in pixel
- la larghezza ed altezza di una dimensione d possono leggersi accedendo alle variabili d.width e d.height (dove d è un oggetto di tipo Dimension)
- void setAlignmentX(float) ed analogamente setAlignmentY con i relativi "get".
  - Il parametro e' un numero tra 0 e 1 che specifica come questa componente e' allineata a quella vicina: 0 = allineamento all'origine, 1 = allineamento all'estremita' opposta all'origine, 0.5 = allineamento centrato, ecc.

Altri metodi relativi alle dimensioni:

- Dimension getSize() ritorna le dimensioni effettive di una componente
- setSize(Dimension) assegna le dimensioni di una componente, **ma puo' essere prevaricata dal layout manager del contenitore in cui la componente e' posta**, o dall'interazione dell'utente (nel caso di componente a top-level)

Eventi comuni a tutti i componenti

Classi di eventi che qualsiasi componente puo' catturare:

- **ComponentEvent:** cambiamenti di stato della componente (posizione, dimensione, visibilita')
- **FocusEvent**: guadagno / perdita del focus della tastiera
- **KeyEvent**: eventi da tastiera
- **MouseEvent**: eventi da mouse

Per ciascuna di queste classi, alla componente posso associare un listener.

- **Nota:** nelle varie sotto-classi di componenti alcuni di questi eventi sono gestiti internamente. Esempio: un campo di input testuale reagisce a guadagno / perdita del focus e a battitura di caratteri da tastiera in modo autonomo.
- Tutti i metodi dei listener hanno parametri come segue:

```
public void nomefunzione(XXXEvent e)
```

- Posso chiamare su e i metodi della classe di evento XXXEvent per reperire informazioni che servono per reagire all'evento.

ESEMPIO:

- getSource() e' definita per ogni classe di evento e ritorna l'oggetto su cui l'evento si e' verificato.
- Tale oggetto va esplicitamente convertito in componente prima di usarlo: Component c = (Component)e.getSource();
- oppure convertirlo nella particolare classe della componente a cui e' associato il listener dell'evento.

EVENTI DI COMPONENTE

ComponentListener prevede i seguenti metodi di reazione ad eventi:

- componentShown, componentHidden: scattano quando la componente diventa visibile / invisibile
- componentMoved, componentResized: scattano quando la componente cambia posizione / dimensioni

FOCUS DELLA TASTIERA

FocusListener prevede i seguenti metodi:

- focusGained, focusLost: scattano quando la componente ottiene / perde il focus

INPUT DA TASTIERA

KeyListener prevede i seguenti metodi:

- keyPressed, keyReleased: scattano quando utente preme / rilascia un tasto mentre il focus e' su questa componente (qualsiasi tasto, anche tasti di controllo e di funzione)
- keyTyped: scatta quando utente batte (preme e rilascia subito) un tasto corrispondente ad un carattere stampabile (non tasti di controllo / funzione)
- La pressione di un carattere stampabile fa scattare tutti e tre i metodi, quella di un tasto non stampabile fa scattare i primi due.
- Al carattere stampabile si accede con e.getKeyChar() (dove e e' l'evento di classe KeyEvent) e al codice del carattere con e.getKeyCode().

EVENTI DA MOUSE

- La classe di eventi MouseEvent ha due listener.
- Si e' ritenuto opportuno separare gli eventi di movimento del mouse, che comportano molto lavoro per gestirli (ogni movimento di un pixel genera un evento) e definire un listener a parte per questi.
- MouseListener reagisce a tutti gli eventi tranne quelli di movimento, con i metodi:
  - mouseEntered, mouseExited: scattano quando il mouse entra / esce dalla componente
  - mousePressed, mouseReleased: scattano quando utente preme / rilascia un bottone del mouse dentro la componente
  - mouseClicked: scatta quando utente clicca (preme e rilascia in rapida sequenza) un bottone dentro la componente
- Se e e' l'evento di classe MouseEvent, la posizione del mouse all'interno del sistema di coordinate della componente si ottiene con e.getX(), e.getY()(in pixel).
- Il click del mouse scatena tre reazioni: pressione, rilascio, click.
- Il numero di pressioni e rilasci contenuti nel click si legge con e.getClickCount() (click semplice, doppio...).

- MouseMotionListener reagisce agli eventi di movimento del mouse, con i metodi:
  - mouseMoved: scatta quando il mouse si muove senza tasti premuti
  - mouseDragged: scatta quando il mouse si muove con un tasto premuto

Finestre (contenitori) top-level

- Si intendano per finestre e contenitori top-level quelle componenti che contengono tutti gli elementi grafici responsabili dell'interfaccia.
- Tra questi quindi vi sarà la finestra principale dell'applicazione come pure tutte le finestre di dialogo (richiesta apertura file, salva con nome… etc…)

Classe Frame / JFrame (AWT/SWING)

- Finestra top-level con tutte le decorazioni del WM (window manager):
- In una finestra top-level è possibile applicare un bordo, una barra del titolo, e dei controlli (per chiusura, iconificazione ecc.).
- La finestra top-level è le finestra primaria di un'applicazione.

CREAZIONE

- Per creare una Frame/JFrame occorre richiamare il metodo costruttore della classe Frame/JFrame.

```
Frame myframe = new Frame();
JFrame myframe = new JFrame("titolo");
```

- Il titolo apparira' sulla barra del titolo aggiunta dal WM.

BARRA DI MENU'

- Un frame puo' avere una ***barra di menu',*** richiamata con la classe MenuBar in AWT o JmenuBar in Swing).
- Per aggiungere la barra di menu' ad un frame:

```
mymenubar = new JMenuBar();
myframe.setMenubar(mymenubar);
```

- Alla barra di menu' poi aggiungero' menu'.

CONTENUTO DI UN FRAME

- Oltre alla barra di menu' un frame e' vuoto.
- Per riempirlo devo aggiungere componenti

- in AWT le aggiungo al frame stesso, che e' un contenitore, chiamando:

  myframe.add(...);

- in Swing le aggiungo al **suo pannello del contenuto** (content pane, contenitore contenuto nel frame che copre l'area lasciata libera dalla menu' bar), chiamando:

```
Container mycontent = myframe.getContentPane();
mycontent.add(...)
```

Classe Dialog / JDialog

- è una finestra top-level dipendente funzionalmente da un'altra (modale o no), intesa come finestra temporanea.
- Usata come **finestra secondaria di un'applicazione** per mostrare informazioni e richiere input (finestra di dialogo).
- Quando distruggo un frame, distruggo anche tutti i suoi dialoghi.
- Quando iconifico / deiconifico il frame, spariscono / riappaiono i suoi dialoghi. Questo comportamento e' gestito automaticamente dal sistema.

CREAZIONE

```
mydialog = new Dialog(myframe);
mydialog = new Dialog(myframe, "titolo");
mydialog = new Dialog(myframe, true/false);
mydialog = new Dialog(myframe, "titolo", true/false);
```

e uguale con JDialog.

- Crea dialogo dipendente da myframe (al posto di un frame potrei avere un altro dialogo).
- Il titolo apparira' sulla barra aggiunda da WM (se la aggiunge) alla finestra del dialogo.
- La costante true/false indica se il dialogo sara' ***modale*** o no (per default non e' modale).

PROPRIETA' DEI DIALOGHI

- setModal(boolean) e boolean isModal() stabilisce / controlla se dialogo e' modale
- setResizable(boolean) e boolean isResizable() stabilisce / controlla se dialogo e' ridimensionabile dall'utente

CONTENUTO

- Il dialogo creato e' vuoto. Per definirne il contenuto devo aggiungere componenti, nello stesso modo in cui si fa per i frame

DIALOGHI PRONTI ALL'USO

- In Swing esistono **option panels**, tipi di pannelli predefiniti da mostrare dentro un dialogo, che contengono:
  - un messaggio
  - un'icona (opzionale) accanto al messaggio: icone predefinite per question, information, warning, error, posso anche personalizzare l'icona
  - un numero di bottoni per chiusura: 1 (ok), 2 (yes/no) o 3 (yes/no/cancel), posso anche personalizzare le etichette
- Creo un JOptionPane e lo metto dentro al mio dialogo.

ESEMPIO:

```
JOptionPane myoption = new JOptionPane ("messaggio",
            JOptionPane.QUESTION_MESSAGE,
            JOptionPane.YES_NO_OPTION);
JDialog mydialog = new JDialog(myframe,
                  "titolo", true);
mydialog.setContentPane(myoption);
mydialog.pack();
```

- crea dialogo che dipende funzionalmente da myframe
- Classe JOptionPane ha metodi per mostrare dialoghi modali associati ad un frame: showMessageDialog (con un bottone), showOptionDialog (con numero di bottoni a piacere), ed altri.

ESEMPIO 1:

```
JOptionPane.showMessageDialog(myframe, "messaggio",
"titolo",JOptionPane.PLAIN_MESSAGE);
```

- Il messaggio e' mostrato nella finestra,
- il titolo apparira' sulla barra del titolo aggiunta dal WM (se sara' aggiunta).
- PLAIN_MESSAGE significa che non e' presente l'icona, alternative: WARNING_MESSAGE, ERROR_MESSAGE, INFORMATION_MESSAGE...

ESEMPIO 2:

```
Object[] options = {"questa", "quella", "l'altra"};
int n = JOptionPane.showOptionDialog(myframe, "Quale delle tre?",
"titolo",JOptionPane.YES_NO_CANCEL_OPTION,
JOptionPane.QUESTION_MESSAGE, new ImageIcon("myicon.gif"), options,
options[2]);
```

- Mostra dialogo con tre bottoni (YES_NO_CANCEL_OPTION), le cui etichette sono le tre stringhe contenute in options, e l'opzione di default e' la terza, l'icona e' presa dal file myicon.gif:
- Esistono anche dialoghi predefiniti che consentono immissione di una stringa.
- I metodi ShowXXXDialog ritornano una costante indicante il bottone premuto: JDialog.OK_OPTION, JDialog.YES_OPTION, JDialog.NO_OPTION, JDialog.CANCEL_OPTION (se 3 bottoni). Oppure JDialog.CLOSE_OPTION se dialogo chiuso agendo sul bordo della finestra.

- L'optione selezionata in un JOptionPane si legge con:
  int n = (Integer)myoption.getValue().intValue();

EVENTI DI FINESTRA

- I contenitori top-level (frame e dialoghi) possono catturare eventi di classe WindowEvent.
- Il WindowListener corrispondente prevede i seguenti metodi di reazione ad eventi:
  - windowOpened scatta quando la finestra viene aperta (es. quando mappata la prima volta)
  - windowClosing scatta quando la finestra sta per essere chiusa (es. dall'utente agendo sulle decorazioni del WM)
  - windowClosed scatta quando la finestra e' chiusa
  - windowIconified e windowDeconified scattano rispettivamente quando la finestra e' iconificata e deiconificata
  - windowActivated e windowDeactivated scattano rispettivamente quando la finestra diventa le finestra attiva sul desktop e quando cessa di esserlo

METODI

Metodi utili comuni a tutte le finestre top-level:

- `String getTitle(), String setTitle(String)` legge, assegna la stringa titolo della finestra
- `setSize(Dimension)` assegna dimensioni
- `pack()` stabilisce che le dimensioni saranno le minime sufficienti a mostrare tutto il contenuto della finestra
- `setVisible(boolean)` mappa su schermo (o demappa)

Prima di mappare una finestra, occorre sempre stabilirne la dimensione con `setSize` o `pack`.

Uso dei contenitori

- Tutti i contenitori hanno un metodo che stabilisce il layout manager e un metodo che consente di aggiungere componenti nel contenitore.

Layout manager

- Per default, il layout manager e' sempre FlowLayout.
- Per cambiare il layout manager (**prima** di aggiungere componenti):

```
mycontainer.setLayout(mylayout);
```

- Il metodo per creare il layout manager dipende dalla classe.
- Una caratteristica comune è data dalla spaziatura orizzontale e da quella verticale: spazio inserito fra due componenti contigue (interi, in pixel)

CLASSE FLOWLAYOUT

```
mylayout = new FlowLayout();
mylayout = new FlowLayout(allineamento);
mylayout = new FlowLayout(spazX,spazY);
mylayout = new FlowLayout(allineamento,spazX,spazY);
```

- dove l'allineamento puo' essere FlowLayout.LEFT, FlowLayout.RIGHT oppure FlowLayout.CENTER (componenti allineate a sinistra, a destra o centrate).

CLASSE GRIDLAYOUT

```
mylayout = new GridLayout(numrighe,numcolonne);
mylayout = new GridLayout(numrighe,numcolonne,spazX,spazY);
```

CLASSE BORDERLAYOUT

```
mylayout = new BorderLayout();
mylayout = new BorderLayout(spazX,spazY);
```

CLASSE BOXLAYOUT

- **Solo** in **Swing**, organizza componenti in riga o colonna.

```
mylayout = new BoxLayout(contenitore,direzione);
```

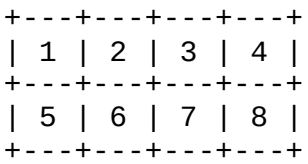
- Dove contenitore e' quello a cui verra' associato il layout manager, direzione puo' essere BoxLayout.X_AXIS oppure BoxLayout.Y_AXIS.
- BoxLayout ha metodi particolari per controllo accurato di allineamento, spaziatura, dimensioni delle componenti al suo interno.

Aggiunta di componenti

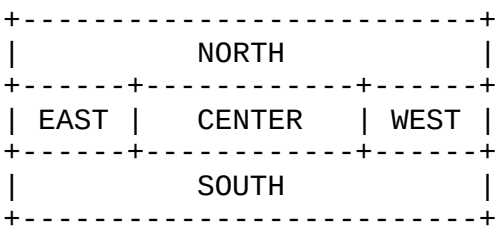
- Il metodo add della classe contenitore consente di aggiungere oggetti:

```
mycontainer.add(componente);
mycontainer.add(componente,posizione);
```

- Se il contenitore ha layout manager FlowLayout o GridLayout (o BoxLayout), allora si usa la prima forma di add:
  - in FlowLayout, le componenti sono messe in fila nello stesso ordine in cui sono aggiunte
  - in GridLayout, le componenti sono prese nell'ordine e vanno a riempire la griglia riga dopo riga



- Se il contenitore ha layout manager BorderLayout, allora si usa la seconda forma di add, dove la posizione e' una fra BorderLayout.NORTH, BorderLayout.SOUTH, BorderLayout.EAST, BorderLayout.WEST, BorderLayout.CENTER



- Il contenitore a cui viene applicato il layout e' diviso in 5 zone, ciascun componente va aggiunto ad una zona specifica,
- non è possibile mettere piu' di un componente per zona!
- E' permesso lasciare vuote alcune zone.

SUGGERIMENTI AL LAYOUT MANAGER

- Alcune componenti possono dare **suggerimenti** al layout manager, ma l'ultima decisione spetta a questo.
- Di fatto i suggerimenti sono le dimensioni minime, massime, preferite, e l'allineamento.

IMPACCHETTAMENTO

- I contenitori top-level hanno metodo `pack`, che assegna le dimensioni della finestra e di tutto il suo contenuto tenendo conto delle dimensioni preferite della finestra e del layout delle sue sottocomponenti.
- Da chiamare **dopo** aver aggiunto tutte le componenti e sottocomponenti alla finestra e **prima** di mapparla.

RIMOZIONE DI COMPONENTI

- Posso aggiungere e anche togliere componenti da un contenitore a run-time. Simmetrico al metodo add esiste metodo remove.

Alcune classi di contenitori intermedi

Pannelli a scomparto unico

CLASSE PANEL / JPANEL

- Senza barre di scorrimento. Creazione:

```
mypanel = new Panel();
mypanel = new Panel(mylayout);
```

- Analogo con JPanel. Se non specificato, il layout manager e' FlowLayout.

CLASSE SCROLLPANE / JSCROLLPANE

- Con barre di scorrimento. Consente di visualizzare (una parte alla volta) un contenuto piu' grande dell'area occupata sullo schermo dal pannello.
- Creazione in AWT:

```
mypanel = new ScrollPane();
mypanel = new ScrollPane(scrolling);
```

  - Dove scrolling determina il criterio usato per decidere se ci sono le barre, e puo' essere uno tra ScrollPane.SCROLLBARS_ALWAYS (barre sempre), ScrollPane.SCROLLBARS_AS_NEEDED (solo se il contenuto e' piu' grande del contenitore), ScrollPane.SCROLLBARS_NEVER (mai).
  - Se non specificato, per default e' "as needed".
  - La componente dentro lo scrolled pane si aggiunge con "add".

- Creazione in Swing:

```
mypanel = new JScrollPane();
mypanel = new JScrollPane(comp);
mypanel = new JScrollPane(scrollY,scrollX);
mypanel = new JScrollPane(comp,scrollY,scrollX);
```

- Dove comp e' la componente da collocare all'interno del pannello.

- I parametri scrollY e scrollX controllano separatamente la presenza / assenza delle barre e possono valere JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED, JScrollPane.VERTICAL_SCROLLBAR_NEVER, JScrollPane.VERTICAL_SCROLLBAR_ALWAYS (e idem per HORIZONTAL).
- Aggiungere la componente con "add" non sembra funzionare.
- Importante impostare le dimensioni del pannello (con setPreferredSize), altrimenti per default il pannello si dimensiona grande quanto la componente contenuta al suo interno.

PANNELLI BIPARTITI

- Pannello bipartito con linea di divisione orizzontale o verticale.
- Contiene due componenti dalle parti opposte del divisorio.
- L'utente puo' trascrinare col mouse il divisorio per stabilire quanto spazio va dato a ciascuna delle due parti.
- Solo Swing:

```
mypanel = new JSplitPane();
mypanel = new JSplitPane(direzione);
mypanel = new JSplitPane(true/false);
mypanel = new JSplitPane(direzione,comp1,comp2);
mypanel = new JSplitPane(direzione,true/false,comp1,comp2);
```

- La direzione e' una fra JSplitPane.HORIZONTAL_SPLIT (defalult) e JSplitPane.VERTICAL_SPLIT.
- La costante booleana indica se il contenuto dei due scomparti viene ridisegnato in tempo reale mentre l'utente sposta il divisorio (per default no).
- I parametri comp1, comp2 sono le due componenti da mettere nei due scomparti (la prima e' quella in alto / a sinistra).

CONTENUTO

- Uno split pane puo' contenere solo due componenti e non ha bisogno di specificare layout manager.
- Se non ho specificato le due componenti alla creazione, Posso aggiungerle dopo con add nella sua prima forma (la prima componente aggiunta e' quella in alto / a sinistra),
- oppure con setTopComponent(Component) e analogo con Bottom, Left, Right.

POSIZIONE DEL DIVISORIO

- Posso cambiare la direzione del divisorio con setOrientation(direzione).
- Per default il pannello determina la posizione del divisorio in base alle dimensioni preferite dei due componenti che contiene.
- Per spostare il divisorio:

```
mypanel.setDividerLocation(percentuale);
```

```
mypanel.setDividerLocation(pixel);
```

- Le posizioni ammissibili per il divisorio risentono delle dimensioni minime dei due componenti che contiene.

Alcune classi di dispositivi

ETICHETTE

- Classe Label, JLabel e' etichetta testuale in AWT, testuale e/o grafica in Swing.
- Creazione in AWT

```
Label label = new Label("stringa di testo");
```

- Creazione in Swing

```
ImageIcon icon = new ImageIcon("immagine.gif");
label1 = new JLabel("solo testo");
label2 = new JLabel("testo e immagine",icon);
label3 = new JLabel(icon);
```

- Puo' esserci un ultimo argomento: l'allineamento, che puo' essere JLabel.LEFT, JLabel.RIGHT, JLabel.CENTER, JLabel.LEADING, JLabel.TRAILING.
- Ci sono altri metodi per stabilire allineamento orizzontale / verticale, spaziatura.

Bottoni

- Bottone di comando: Button / JButton
- Bottone a due stati (selezionato o non selezionato) o check box: Checkbox / JCheckBox

CREAZIONE

- In AWT il creatore prende come argomento una stringa (etichetta da visualizzare).
- In Swing prende una stringa, un'icona, oppure una stringa e un'icona.
- Esempi in Swing:

```
mybutton = new JButton("titolo");
mybutton = new JButton("titolo",icona);
mybutton = new JButton(icona);
```

- Stessa cosa si puo' fare con JCheckBox, con in piu' un ultimo argomento booleano opzionale che stabilisce se il bottone e' inizialmente selezionato o no (di default non e' selezionato).
- La prima chiamata e' legale anche con Button o Checkbox (AWT).

REAZIONE A EVENTI

- Bottone di comando puo' catturare evento di azionamento ActionEvent: l'utente ha azionato il bottone, in che modo dipende dalla piattaforma (di solito col mouse).
- Il corrispondente ActionListener ha un solo metodo actionPerformed che scatta quando il bottone e' azionato.
- In ciascuna finestra top-level posso definire al piu' un solo bottone "di default" che e' sensibile anche alla pressione del tasto "return" (il sistema distingue graficamente tale bottone).

```
finestratop.getRootPane().setDefaultButton(bottone);
```

- Bottone a due stati puo' catturare evento di azionamento ed evento di cambio stato ItemEvent.
- ItemListener prevede un solo metodo ItemStateChanged, che scatta quando il bottone cambia stato (da selezionato a deselezionato o viceversa). Il programma legge il nuovo stato con int getStateChange() che ritorna uno fra ItemEvent.SELECTED e ItemEvent.DESELECTED.

ALCUNE FUNZIONI UTILI SU BOTTONI A DUE STATI

- Per selezionare / controllare se e' selezionato:
- boolean isSelected() ritorna se componente selezionato setSelected(boolean) forza selezione
  - boolean isSelected() ritorna true se componente selezionato
  - setSelected(boolean) forza selezione

GRUPPI DI BOTTONI RADIO

- Gruppo di bottoni radio e' un gruppo di bottoni a due stati dove in ogni istante un solo bottone puo' essere selezionato.
- La pressione di un bottone del gruppo provoca automaticamente il rilascio di tutti gli altri bottoni del gruppo.
- Gruppi di radio-bottoni servono a implementare scelte mutuamente esclusive.

IN AWT

- un bottone radio e' un check box creato nel contesto di un certo gruppo
- esiste classe CheckboxGroup per gruppo di bottoni a due stati
- classe Checkbox ha un altro costruttore che prende come argomento un gruppo, il bottone viene creato all'interno di quel gruppo
- i checkbox creati nello stesso gruppo lavorano in modo mutuamente esclusivo

ESEMPIO:

```
CheckboxGroup cbg = new CheckboxGroup();
add(new Checkbox("uno", cbg, true));
add(new Checkbox("due", cbg, false));
add(new Checkbox("tre", cbg, false));
```

- L'ultimo argomento booleano stabilisce se il bottone e' inizialmente selezionato (di default non e' selezionato).

IN SWING

- classe apposita di bottoni a due stati adatti ad essere inseriti in un gruppo radio JRadioButton
- creo vari JRadioButton e li aggiungo ad un unico ButtonGroup

ESEMPIO:

```
rb1 = new JRadioButton("uno",true);
rb2 = new JRadioButton("due");
rb3 = new JRadioButton("tre");
ButtonGroup bg = new ButtonGroup();
bg.add(rb1);
bg.add(rb2);
bg.add(rb3);
```

- Radio button cattura eventi ActionEvent e ItemEvent.
  Quando utente preme un bottone radio in un gruppo, si generano tre eventi:
  - ItemEvent sul bottone premuto: suo stato cambia da non selezionato a selezionato
  - ItemEvent sul bottone che era selezionato in precedenza: suo stato cambia da selezionato a non selezionato
  - ActionEvent sul bottone premuto

Menu' e voci di menu'

- Menu' permette scelta di un'opzione da una lista di voci che e' presentata come una tendina a scomparsa.
- Un menu' puo' essere:
  - parte della barra di menu' di un frame: il titolo e' permanentemente mostrato sulla barra e quando utente la aziona appare la tendina
  - pop-up: associato ad un altra componente, e' normalmente invisibile, la tendina appare quando utente aziona la componente alla quale il menu' e' associato

ESEMPIO DI MENU' INSERITI IN BARRA:

```
mybar = new MenuBar();
myframe.setMenuBar(mybar);
menu1 = new Menu("Primo menu'");
mybar.add(menu1);
menu2 = new Menu("Altro menu'");
mybar.add(menu2);
```

- Analogamente si crea menu' pop-up:

```
mypop = new PopupMenu("Menu' pop-up");
```

- Codice analogo nel caso Swing.

DEFINIZIONE DELLE VOCI DI MENU'

- Ogni voce del menu' (MenuItem) ha caratteristiche analoghe a un bottone. Le classi di voci di menu' rispecchiano le classi di bottoni: JMenuItem, JCheckBoxMenuItem, in AWT, JMenuItem, JCheckBoxMenuItem, JRadioButtonMenuItem in Swing.
- Creo le varie voci separatamente e poi le aggiungo ad un Menu / JMenu. Un Menu / JMenu puo' essere usato come voce di un altro menu', realizzando in tal modo un menu' a cascata.

ESEMPIO:

```
menu1 = new Menu("Primo menu'");
menu1.add (new MenuItem("voce semplice"));
menu1.add (new CheckboxMenuItem("voce a due stati"));
menu2 = new Menu("sotto menu'");
menu1.add (menu2);
menu2.add("una voce");
menu2.add("altra voce");
```

- Crea un menu' (menu1) con tre voci: una semplice, una a check box e una costituita da un sottomenu' (menu2).
- In piu' in Swing posso mettere testo e/o icone nelle voci di menu':

```
item = JMenuItem(stringa);
item = JMenuItem(icona);
item = JMenuItem(stringa,icona);
```

- Analogo per JCheckBoxMenuItem con in piu' ultimo argomento booleano (opzionale) che specifica se voce e' inizialmente selezionata o no.
- In piu' in Swing ho classe JRadioBoxMenuItem per realizzare voci a due stati mutuamente esclusive.

EVENTI NEI MENU'

- Quelle che reagiscono a eventi sono le singole voci dei menu'.
- Le voci che sono sottomenu' le gestisce automaticamente il sistema (la loro reazione e' mostrare la tendina del sottomenu')
  - Voci di menu' semplici (MenuItem / JMenuItem) sono sensibili a ActionEvent, come i bottoni di comando.
    - Associo un ActionListener alla voce di menu'.
  - Voci di menu' a due stati (CheckboxMenuItem / JCheckBoxMenuItem / JRarioButtonMenuItem) sono sensibili anche a ItemEvent, come i bottoni a due stati.

MENU' POP-UP

- Per assegnare un menu' pop-up ad un componente, si associa a quel componente un MouseListener che faccia apparire il menu' pop-up quando l'utente aziona il mouse.

ESEMPIO:

```
public class PopFrame extends Frame
{
    PopupMenu popup;

    public PopFrame()
    {
        Label label = new Label("Try bringing
                                up a popup menu!");
        add(label);

        popup = new PopupMenu("A Popup Menu");
        add(popup);
        MenuItem mi1 = new MenuItem("aaa");
        popup.add(mi1);
        MenuItem mi2 = new MenuItem("bbb");
        popup.add(mi2);

        MouseListener listener = new PopupListener();
        addMouseListener(listener);
        label.addMouseListener(listener);
    }

    class PopupListener extends MouseAdapter
    {
        public void mousePressed(MouseEvent e)
        {
            maybeShowPopup(e);
        }

        public void mouseReleased(MouseEvent e)
```

```
            {
                maybeShowPopup(e);
            }

            private void maybeShowPopup(MouseEvent e)
            {
                if (e.isPopupTrigger())
                    popup.show(e.getComponent(),
                                           e.getX(), e.getY());
            }
        }
    }
```

KEYBOARD ALTERNATIVES

- Scorciatoie da tastiera.
  - Mnemonici: tasti per selezionare una voce da un menu' la cui tendina sia correntemente visibile
  - Acceletatori: combinazioni di tasti per selezionare una voce da un menu' anche se la tendina non e' visibile
- Esistono appositi metodi per associare mnemonici e acceleratori alle voci di menu'.