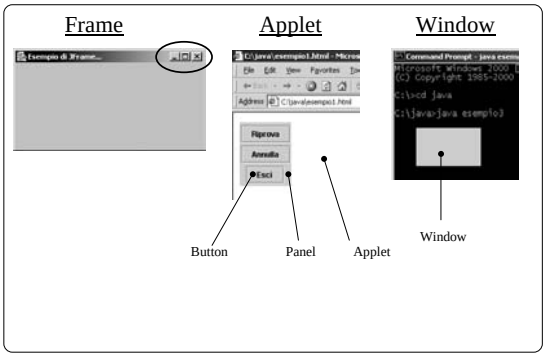
Programmazione grafica

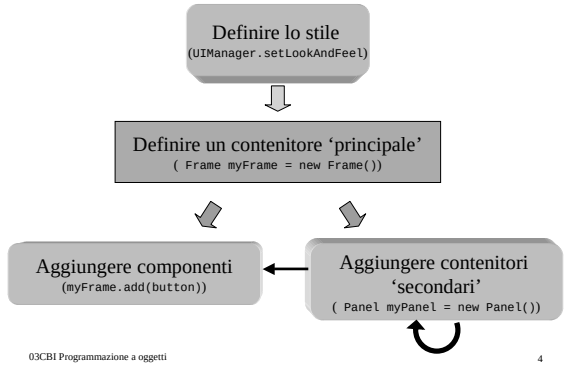
Concetti della programmazione grafica

- Impostare un Look & Feel (= Stile)
  - Microsoft → stile *Windows*
  - Macintosh → stile *Mac*
  - Java → stile *Metal*
- Definire uno (o più) contenitori principali
  - Window, Frame, Applet
- Aggiungere i componenti ai contenitori
  - Button
  - RadioBox
  - Ecc.
- Disporre i componenti/contenitori secondo un **layout**

Differenza Frame – Applet – Window



Procedura

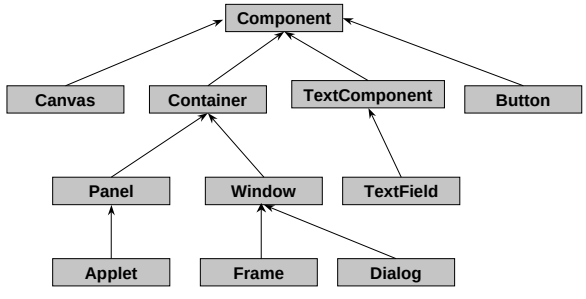


Package `java.awt.*`

Fornisce le seguenti funzionalità:

- **Componenti**: bottoni, checkbox, scrollbar, ecc.
- Supporto per "containers" secondari: sono ancora componenti!
- Gestione di:
  - eventi di sistema
  - eventi generati dagli utenti su parti dell'UI
- Layout: i componenti sono inseriti in modo **indipendente** dalla piattaforma

Le sottoclassi di `Component` in `java.awt`



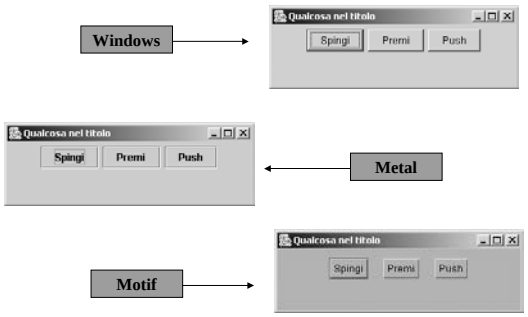
Package `javax.swing`

- Contiene gli stessi componenti di `java.awt`, ma con il nome differente (`JButton`, `JFrame`, ecc.)
- Tutti questi componenti derivano da `JComponent`
- Vantaggi:
  - fornisce una serie di componenti 'leggeri' (light-weight) che presentano lo stesso aspetto/comportamento su tutte le piattaforme
  - Il look and feel è modificabile al volo
- Swing è un'estensione di AWT → tuttavia è diversa la gestione degli eventi nei due package

Definire uno stile (passo 1 di 4)

- La classe di riferimento è `UIManager`, che appartiene al package `java.lang`
- Possibilità:
  - `UIManager.setLookAndFeel( "com.sun.java.swing.plaf.windows.`**`WindowsLookAndFeel`**`");`
  - `UIManager.setLookAndFeel("com.sun.java.swing.plaf.motif.`**`MotifLookAndFeel`**`");`
  - `UIManager.setLookAndFeel("javax.swing.plaf.mac.`**`MacLookAndFeel`**`");` → SOLO SU PIATTAFORME MACINTOSH
  - `UIManager.setLookAndFeel("javax.swing.plaf.metal.`**`MetalLookAndFeel`**`"); [DEFAULT]`
- La classe `UIManager` va sempre fatta precedere da un blocco `TRY..CATCH`

Esempi di Look & Feel



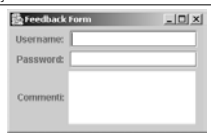
Impostare un contenitore (Passo 2 di 4)

- Passi da compiere:
  - La classe deve estendere il contenitore primario scelto (`class MioContenitore` **`extends`** `JFrame`)
  - È necessario creare un primo contenitore secondario e definirlo come 'interno' (`Jpanel finestra = new Jpanel(); (…) setContentPane(finestra)`)
  - I componenti vanno aggiunti a `finestra`:

```
JButton BottonePremi = new JButton ("Premi");
finestra.add(BottonePremi);
```


Un esempio completo

```
import javax.swing.*;

public class Form extends
javax.swing.JFrame {
JTextField username = new
          JTextField(15);
JPasswordField password = new
          JPasswordField(15);
JTextArea commenti = new JTextArea(4,
          15);

public Form() {
super("Feedback Form");
setSize(260, 160);

setDefaultCloseOperation(EXIT_ON_CLOSE)
;

JPanel pane = new JPanel();
JLabel usernameLabel = new
          JLabel("Username: ");
JLabel passwordLabel = new
          JLabel("Password: ");
JLabel commentiLabel = new
          JLabel("Commenti: ");
          commenti.setLineWrap(true);
          commenti.setWrapStyleWord(true);
          pane.add(usernameLabel);
          pane.add(username);
          pane.add(passwordLabel);
          pane.add(password);
          pane.add(commentiLabel);
          pane.add(commenti);
          setContentPane(pane);

          show();
     }

     public static void main(String[]
arguments) {
          Form input = new Form();
     } }
```



Funzioni base dei contenitori

- Messaggi (`msg`) nella chiusura delle finestre → nella forma "`setDefaultCloseOperation(msg)`"
  - `EXIT_ON_CLOSE`
  - `DO_NOTHING_ON_CLOSE`
  - `DISPOSE_ON_CLOSE`
  - `HIDE_ON_CLOSE`
- `SetSize(int base, int altezza)` → definisce le dimensioni del pannello esterno
- `setBounds (int xSupSin, int ySupSin, int base, int altezza)` → specifica anche la posizione in cui si trova inizialmente il pannello
- Inserire il contenitore secondario nel primario:

```
contPrimario.setContentPane(contSecondario);
```

Inserire i componenti (fase 3 di 4)

Componenti Swing – Bottoni

- Un bottone è un componente: innesca un'**azione** con l'**evento** di pressione.
- Un bottone è creato mediante i costruttori:
  - `JButton();` crea un bottone senza testo (senza label)
  - `JButton(String);` crea un bottone con una label contenente il testo dato
- E' un *componente* → eredita tutti i metodi delle classi `JComponent` (`javax.swing`) e `Component` (`java.awt`)
- E' un *contenitore* → eredita tutti i metodi di `java.awt.Container.class`

Componenti Swing – Etichette

- `JLabel();` crea etichetta vuota, allineata a sinistra
- `JLabel(String);` crea etichetta con testo dato, allineata a sinistra
- `JLabel(String, int);` crea etichetta con testo dato, allineata secondo quanto specificato nel secondo parametro (`SwingConstants.LEFT, SwingConstants.RIGHT, SwingConstants.CENTER`).
- Metodi attivabili su una etichetta: `getText(), setText(String), getAlignement(), setAlignement(int).`

Componenti Swing – Campi di testo

- I campi testo permettono l'introduzione di stringhe di testo da parte dell'utente.
- Per creare un text field si usano i costruttori:
  - `JTextField();`
  - `JTextField (String), JTextField (String, int);`
- `t.setEchoChar('*');` //visualizza il carattere scelto ('*') per ogni carattere immesso

Esempio

```
tf1 = new JTextField();
tf2 = new JTextField("", 20);
tf3 = new JTextField("Hello");
tf4 = new JTextField("Hello",30);
```

Componenti Swing – Aree di testo

- Servono a gestire più di una linea di testo per volta.
- Costruttori:
  - `JTextArea (int1, int2)` → `int1`: n.ro righe, `int2`: n.ro colonne
  - `JTextArea (String, int1, int2)` → `String` è un testo di default
- Metodi utili:
  - `getText(), setText(String);`
  - `append(String), insert(String, int);`
  - `void setLineWrap(boolean)` → se `true`, va a capo autonomamente
  - `void setWrapStyleWord(boolean)` → se è `true`, va a capo con la parola intera

Componenti Swing – Controlli/Opzioni

- Caselle di controllo: `JCheckBox(String, boolean)`
- Pulsanti di opzione: `JRadioButton(String, boolean)`
- Metodi utili:
  - `void setSelected(boolean)` → con `true`, setta il componente ad 'acceso'
  - `boolean isSelected()` → restituisce `true` se il componente è acceso
- Per default sono non-esclusivi: possono essere presenti più `RadioButton` accesi in contemporanea
- Per garantire la mutua esclusione: il `RadioButton` (o il `CheckBox`) sono aggiunti a un `ButtonGroup`

Esempio

```
import javax.swing.*;
class Autori extends JFrame {
        JRadioButton[] lista = new JRadioButton[4];
        Autori() {
                super("Scelta di un autore");
                setSize(140, 190);
                setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                lista[0] = new JRadioButton("Jehoshua", true);
                lista[1] = new JRadioButton("McEwan");
                lista[2] = new JRadioButton("Tamaro");
                lista[3] = new JRadioButton("Steel");
                JPanel pannello = new JPanel();
                ButtonGroup gruppo = new ButtonGroup();
                for (int i = 0; i < lista.length; i++) {
                        gruppo.add(lista[i]);
                        pannello.add(lista[i]);
                }
                setContentPane(pannello);
                show();
        }
        public static void main (String args[]) {
                Autori newLista = new Autori();
        }}
```

Componenti Swing – Finestre di Dialogo

- Usate per ricevere input, fornire informazioni, avvisare l'utente, ecc.
- Possibilità:
  - Finestre di conferma
  - Finestre di dialogo e input
  - Finestre di dialogo e messaggio
  - Finestra di dialogo e opzioni
- Sono un metodo molto più efficiente dei flussi di input/output per leggere da tastiera
- Ogni finestra è gestita da un metodo diverso della stessa classe (`JOptionPane.class`)

Finestre di dialogo per conferma

- Il metodo attivatore è `showConfirmDialog(Component, Object)` → restituisce un intero
- In alternativa: `showConfirmDialog(Component, Object, String, int1, int2);`
  - `Component:` in quale componente appare la finestra (con `"null"`, la finestra appare nel centro dello schermo)
  - `Object:` il messaggio o l'immagine contenuta
  - `String:` titolo della finestra
  - `int1:` quali pulsanti appaiono (due possibilità: `YES_NO_OPTION`, oppure `YES_NO_CANCEL_OPTION`)
  - `int2:` l'icona del messaggio (possibilità: `ERROR_MESSAGE`, `INFORMATION_MESSAGE`, `PLAIN_MESSAGE`, `QUESTION_MESSAGE`, `WARNING_MESSAGE`)

Finestre di dialogo di input

- Il metodo di riferimento è `showInputDialog`
- Primitiva: `String showInputDialog(Component, Object)`
- In alternativa: `String showInputDialog(Component, Object, String, int)`
  - `Component:` in quale componente appare la finestra
  - `Object:` messaggio di richiesta input
  - `String:` titolo
  - `int:` tipo di messaggio (stessa codifica della finestra di conferma)
- L'input è immediato: `String risposta = JOptionPane.showInputDialog(null, "Il tuo dolce preferito?", "Domande…", JOptionPane.QUESTION_MESSAGE)`

Finestre di dialogo di messaggio

- Il metodo di riferimento è in questo caso `showMessageDialog`:
- `void showMessageDialog(Component, Object)`
- `void showMessageDialog(Component, Object, String, int)`
- Non restituisce valori e possiede un bottone di default (OK)

Finestre di dialogo di opzione

- Il metodo di riferimento è `showOptionDialog`
- Dà la possibilità di realizzare delle scelte su bottoni personalizzati
- Primitiva: `int showOptionDialog(Component, Object, String, int1, int2, Icon, Object[], Object)`
  - int1 → `YES_NO_OPTION`, `YES_NO_CANCEL_OPTION` oppure altro ('0')
  - int2 → tipo di messaggio ('0' per nessun messaggio)
  - Icon → icona al posto di int2 (`"null"` per nessuna scelta)
  - `Object[]` → array di oggetti tra cui scegliere
  - `Object` → scelta di default

Esempio

```
(…)
JButton[] sport = new JButton[4];
sport[0] = new JButton("nuoto");
sport[1] = new JButton("calcio");
sport[2] = new JButton("boxe");
sport[3] = new JButton("bocce");
int risposta = JOptionPane.showOptionDialog(null, "Qual e'
  il tuo sport preferito?", "Domande sportive", 0,
  JOptionPane.INFORMATION_MESSAGE, null, sport, sport[3]);
(…)
```

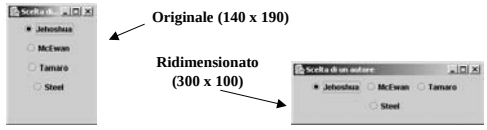
Esercizio

- Creare un'applicazione Java in grado di prendere in input da utente le informazioni
  - Nome del sito internet
  - Indirizzo URL
  - Informazioni generali {Personale, Business, Educativo}
- Con esse, creare tre coppie Etichette/Linee di testo (etichetta = "nome", testo = inserito da utente)
- Provare a modificare le dimensioni del pannello principale
- Infine, **prima** di aggiungere i componenti al pannello, usare `pannello.setLayout(new GridLayout(3,2))`, e provare ancora a modificare le dimensioni

LAYOUT

Cos'è un layout?

- Tutti gli esempi grafici precedenti, quando ridimensionati, modificano la disposizione dei componenti:



- Questo comportamento è una necessità: Java si adatta a molte piattaforme (visualizzazione differente per sistemi differenti)
- Soluzione in Visual Basic → disposizione 'assoluta' (x,y)

I gestori di layout in Java

- *Layout* → indica **dove** sono localizzati i componenti
- *Gestore di layout* → determina il **metodo** di disposizione degli stessi componenti (`import java.awt.*`)
- Un pannello ↔ un gestore di layout
- Quindi: pannelli diversi possono avere gestori diversi
- Metodologia:
  - Creare un'istanza dalla classe del gestore:
    `FlowLayout f = new FlowLayout()`
  - Creare un pannello, e per prima cosa assegnargli il gestore:
    ```
    JPanel pannello = new JPanel();
    pannello.setLayout(f);
    ```
  - **Dopo**, aggiungere i componenti al pannello:
    `pannello.add(JButton); (…)`

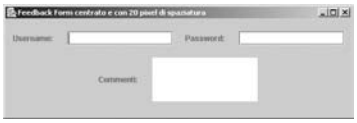
Gestori di layout – FlowLayout

- E' il layout di base delle applicazioni/applet grafici
- Disposizione: da sinistra a destra, a partire dall'angolo in alto a sinistra
- Costruttori:
  - `FlowLayout f = new FlowLayout();`
  - `FlowLayout f = new FlowLayout(int1);`
  - `FlowLayout f = new FlowLayout(int1, int2, int3);`
- Elementi dei costruttori:
  - `int1`: allineamento di base (`FlowLayout.LEFT`, `FlowLayout.RIGHT`, `FlowLayout.CENTER`)
  - `int2`: spazio orizzontale tra componenti (default: 3 pixel)
  - `int3`: spazio verticale tra componenti (default: 3 pixel)

Esempi di FlowLayout (default in Java)

(FlowLayout.RIGHT)

(FlowLayout.CENTER,20,20)



Gestori di layout – GridLayout

- Divide lo schermo in una griglia di righe e colonne
- Riempimento: dalla casella in alto a sinistra
- Costruttori:
  - `GridLayout g = new GridLayout(int1, int2);`
  - `GridLayout g = new GridLayout(int1, int2, int3, int4);`
- Argomenti:
  - `int1`: numero di righe;
  - `int2`: numero di colonne;
  - `int3`: spaziatura (in pixel) tra due caselle orizzontali (default: 0 pixel)
  - `int4`: spaziatura (in pixel) tra due caselle verticali (default: 0 pixel)

Esempi di GridLayout

4 righe, 1 colonna: distanza 0

2 righe, 2 colonne: distanza 0 pixel

(Distanza min = 0)

2 righe, 2 colonne: distanza 10 pixel

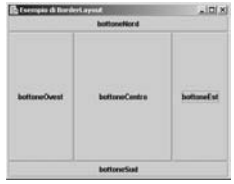
(Distanza min = 10)

Gestori di layout – BorderLayout

- Divide lo schermo in cinque zone (“North”, “South”, “East”, “West”, “Center”)
- Il riempimento è ‘mirato’:

```
JPanel pannello = new JPanel();
BorderLayout b = new BorderLayout();
pannello.setLayout(b);
pannello.add(“North”, bottoneNord);
pannello.add(“West”, bottoneOvest);
Etc…
```

Quale zona dello schermo — **Quale componente aggiungere**



- Costruttore alternativo: `BorderLayout(int1, int2)`, dove i due argomenti sono gli spazi relativi tra i componenti orizzontali e verticali

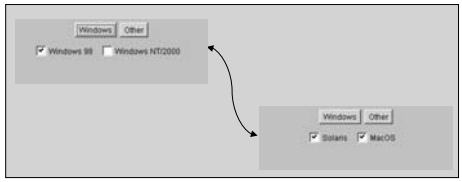
Gestori di layout – CardLayout

- Gli ultimi due gestori hanno proprietà avanzate
- Con CardLayout è possibile avere diversi pannelli nel frame, ma mostrarne uno solo per volta
- I pannelli si chiamano *schede*
- Metodologia:
  - Creare un pannello primario
  - Il **suo** layout è CardLayout
  - Creare pannelli secondari
  - Aggiungerli al primario con:
    `pannPrimario.add(“TitoloSecondario”, nomeSecondario)`

Scheda = pannello secondario

Pannello primario

Esempio di CardLayout



- Le due schede sono mutuamente esclusive: quando la prima diventa visibile, la seconda entra nel sottofondo e viceversa
- Come renderle visibili? Bisogna agire tramite il gestore stesso:

```
CardLayout c = new CardLayout;
c.show(UnoDeiPannelli, “NomePannello”);
```

Gestori di layout – GridBagLayout

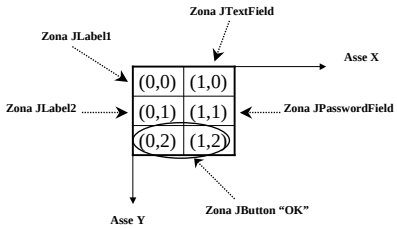
- Estensione del layout a griglia (GridLayout)
- È possibile regolare gli elementi della griglia con meccanismi di personalizzazione
- Metodologia di utilizzo:
  1. Creare un'istanza della classe GridBagLayout
  2. Creare un'istanza dello 'strumento di regolazione' (classe GridBagConstraints)
  3. Regolare ciascun componente
  4. Informare il gestore delle regolazioni avvenute
  5. Aggiungere i componenti al pannello

Esercizio

- Per comprendere il significato delle regolazioni, provare a creare un componente di questo tipo:

GridBagConstraints in dettaglio

- Il progetto precedente si può visualizzare in modo schematico nel modo seguente:



Regolazioni su GridBagConstraints (1)

1. Bisogna definire in quali posizioni (x,y) vengono inseriti i componenti
2. Il pulsante "OK" deve occupare **due** celle: gli altri componenti occupano una sola cella
3. L'ampiezza dei componenti è variabile (l'etichetta "nome" occupa circa 30% della linea…)
4. Come si dispongono i componenti nelle celle (il pulsante "OK" è centrato, ecc.)

Regolazioni su GridBagConstraints (2)

- La classe `GridBagConstraints` ha diversi attributi:
  - `gridx` → posizione x del componente (sulla griglia)
  - `gridy` → posizione y del componente (sulla griglia)
  - `gridwidth` → quante celle occupa il componente (in *orizzontale*)
  - `gridheight` → quante celle occupa il componente (in *verticale*)
  - `weightx` → grandezza relativa (%) di una cella rispetto alla *riga* che la contiene
  - `weighty` → grandezza relativa (%) di una cella rispetto alla *colonna* che la contiene
  - `fill` → il componente occupa tutta la cella?
  - `anchor` → se non la occupa tutta, dov'è ancorato?

Regolazioni su GridBagConstraints (3)

- I valori di `fill` sono: `BOTH, NONE, HORIZONTAL, VERTICAL`
- I valori di `anchor` sono: `CENTER, NORTH, NORTHEAST, EAST, SOUTHEAST, SOUTH, SOUTHWEST, WEST, NORTHWEST`
- Perciò…

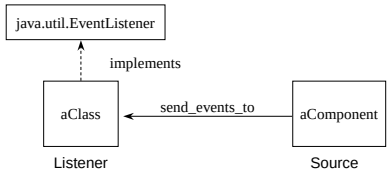
```
GridBagLayout griglia = new GridBagLayout();
pannello.setLayout(griglia);
GridBagConstraints Gbc = new GridBagConstraints();
JLabel etichetta1 = new JLabel ("Nome:", JLabel.LEFT);
        Gbc.gridx = 0;
        Gbc.gridy = 0;
        Gbc.gridwidth = 1;
        Gbc.gridheigth = 1;
        Gbc.weightx = 30;
        Gbc.weigthy = 40;
        Gbc.fill = GridBagConstraints.NONE;
        Gbc.anchor = GridBagConstraints.EAST;
griglia.setConstraints(Gbc, etichetta1);
pannello.add(etichetta1);
```

Gestione Eventi

Event Delegation Model

- Da Java1.1
  - Gli eventi sono classificati in base al tipo (MouseEvent, KeyEvent, ecc.)
  - Gli eventi vengono generati in Componenti sorgenti
  - Un oggetto può essere registrato come ascoltatore (listener) di un tipo di evento mandando un messaggio al componente sorgente

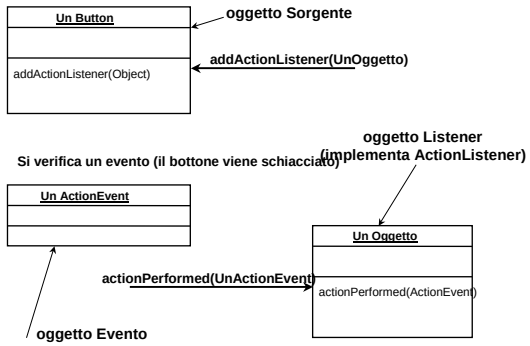
- Al verificarsi di un evento il gestore dell'AWT invia un apposito messaggio a tutti gli ascoltatori registrati (l'Evento viene passato come parametro)
- Un ascoltatore deve implementare un'opportuna **interfaccia** (per rendere possibile il call-back)



Eventi

- Gli eventi sono rappresentati da una gerarchia di classi. Ogni classe è definita dai dati che rappresentano quel tipo di evento.
- Alcune classi che rappresentano un insieme di eventi (MouseEvent) possono contenere un **id** che identifica il tipo esatto.

Esempio



Eventi in java: sorgenti e tipi

Sorgente	Evento
Pulsante	Eventi di AZIONE → quando si preme il pulsante
Casella di scelta	Eventi di ELEMENTO → selezione/deselezione del controllo
Voce di menu	Eventi di AZIONE → quando è selezionata una voce del menu; eventi di ELEMENTO → quando è attivata una voce di menu 'selezionabile'
Finestra	Eventi di FINESTRA → quando è attivata, massimizzata, resa icona, ecc.
ecc.	

Differenza 'selezione – attivazione'

Gestione degli eventi in Java

- Problema: bisogna rendere *ricettiva* l'interfaccia Java
- Eventi contemplati in Java:
  - Action event → *clic* su bottoni
  - Adjustment event → azioni sulle barre di scorrimento
  - Focus event → punto il mouse su un campo di testo
  - Item event → *clic* su RadioButton, CheckBoxButton
  - Key event → input da tastiera
  - Mouse event → *clic* (non contemplati sopra)
  - Mouse-motion event → spostamento semplice del mouse
  - Window event → ingrandire, chiudere una finestra

Un esempio di gestione degli eventi (1/4)

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

public class MouseEvents extends JFrame implements
  MouseListener, MouseMotionListener {

  int mouseX = 0, mouseY = 0;

  public MouseEvents() {
      super("Eventi del mouse");
      setSize(300,120);
      setDefaultCloseOperation(EXIT_ON_CLOSE);
      JPanel pannello = new JPanel();
      pannello.addMouseListener(this);
      pannello.addMouseMotionListener(this);
      setContentPane(pannello);
      show();
  }
```


Un esempio di gestione degli eventi (2/4)

```
public void mouseClicked(MouseEvent me) {
      mouseX = 0;
      mouseY = 10;
      System.out.println("Rilevato click del mouse.");
  }

  public void mouseEntered(MouseEvent me) {
      mouseX = 0;
      mouseY = 10; // posizione di 'entrata' = in alto a
  sinistra (sotto il bordo del frame)
      System.out.println("Il mouse e' nella zona sensibile.");
  }

  public void mouseExited(MouseEvent me) {
      mouseX = 0;
      mouseY = 10;
      System.out.println("Il mouse e' fuori dalla zona
  sensibile.");
  }
```


Un esempio di gestione degli eventi (3/4)

```
public void mousePressed(MouseEvent me) {
      mouseX = me.getX();
      mouseY = me.getY();
      System.out.println("Mouse premuto");
  }

  public void mouseReleased(MouseEvent me) {
      mouseX = me.getX();
      mouseY = me.getY();
      System.out.println("Mouse rilasciato");
  }

  public void mouseDragged(MouseEvent me) {
      mouseX = me.getX();
      mouseY = me.getY();
      System.out.println("Si sta trascinando il mouse in
  coordinate " + mouseX + ", " + mouseY);
  }
```

Un esempio di gestione degli eventi (4/4)

```
  public void mouseMoved(MouseEvent me) {
      System.out.println("Si sta muovendo il mouse in
  coordinate " + me.getX() + ", " + me.getY());
  }

  public static void main (String args[]) {
      JFrame frame = new MouseEvents();
  }
}
```


Come gestire gli eventi in Java

- Il principio su cui si basano gli eventi è abbastanza simile alle eccezioni:
  - La classe dichiara quale evento è in grado di trattare (uno o più di uno) → implementa una o più interfacce
  - Si associa un *ascoltatore* ai componenti 'a rischio' (`JButton`, `JTextField`, ecc.) → `JButton.addActionListener(this)`
  - Attenzione! Dato che si implementano delle interfacce, bisogna riscrivere ***tutti*** i metodi di quelle interfacce!

```
class FrameConEventi extends JFrame implements InterfacciaConEventi {
     JComponent componenteARischio = new JComponent();
     componenteARischio.addAscoltatore(this);
     void metodoInterfacciaConEventi() {…}
     void altroMetodoInterfacciaConEventi() {…}
}//end classe
```


Interfacce di evento (1)

1. **`ActionListener`** → metodi da riscrivere:
   - `void actionPerformed (ActionEvent evt)`
2. **`FocusListener`** → metodi da riscrivere:
   - `void focusGained (FocusEvent evt)`
   - `void focusLost (FocusEvent evt)`
3. **`ItemListener`** → metodi da riscrivere:
   - `void itemStateChanged (ItemEvent evt)`
4. **`MouseListener`** (→ metodi da riscrivere:
   - `void mouseClicked (MouseEvent evt)`
   - `void mouseEntered (MouseEvent evt)`
   - `void mouseExited (MouseEvent evt)`
   - `void mousePressed (MouseEvent evt)`
   - `void mouseReleased (MouseEvent evt)`
5. **`MouseMotionListener`** (→ metodi da riscrivere:
   - `void mouseDragged (MouseEvent evt)`
   - `void mouseMoved (MouseEvent evt)`

Interfacce di evento (2)

6. **`KeyListener`** → metodi da riscrivere:
   - `void keyPressed (KeyEvent evt)`
   - `void keyReleased(KeyEvent evt)`
   - `void keyTyped(KeyEvent evt)`
7. **`WindowListener`** (→ metodi da riscrivere:
   - `void windowActivated(WindowEvent evt)`
   - `void windowClosed (WindowEvent evt)`
   - `void windowClosing (WindowEvent evt)`
   - `void windowDeactivated (WindowEvent evt)`
   - `void windowDeiconified (WindowEvent evt)`
   - `void windowIconified (WindowEvent evt)`
   - `void windowOpened (WindowEvent evt)`

Aggiungere un ascoltatore

- Due metodi equivalenti:
  - il componente aggiunge l'ascoltatore su di sé
    - `JButton.addActionListener(this)`
  - il pannello principale (ad es. JFrame) aggiunge gli ascoltatori ai componenti
    - `JFrame.addActionListener(JButton)`
- Come trattare l'evento?
  1. Scoprire chi ha generato l'evento

```
Object ob = evt.getSource();
if (ob == BottoneAutoDistruzione) // notare l'operatore '=='
   distruggiTutto();
```

  2. Gestire l'evento con i metodi delle classi relative (`KeyEvent.class`, `WindowEvent.class`, ecc.)

Esercizio

- Il CardLayout presentato prima ha una gestione interna degli eventi:
  - Premendo il pulsante "Windows" viene mostrata la scheda relativa a Windows
  - Premendo "Other" viene visualizzata la seconda scheda
- Provare a ottenere questo risultato

Come gestire gli eventi

- Ogni funzione che appare nelle interfacce presenta un argomento comune (`KeyEvent`, `MouseEvent`, ecc.)
- Ciascuno di questi argomenti è un oggetto, e definisce una serie di metodi per ottenere 'informazioni' sull'evento accaduto:
- Esempi :
  - `ActionListener`
    - → `String getActionCommand()`: restituisce la stringa relativa al *componente* che ha generato il comando
    - → `String paramString()`: restituisce la stringa che descrive il tipo di evento {è comune a tutti gli oggetti-evento}

Metodi associati agli oggetti-evento

- `ItemEvent`:
  - `int getStateChange()`: restituisce `SELECTED` o `DESELECTED` a seconda che il `RadioButton` o il `CheckBox` sia stato attivato o meno
- `KeyEvent`:
  - `char getKeyChar()`: restituisce il carattere digitato da tastiera → utile anche nel caso di un case da tastiera
  - `int getKeyCode()`: restituisce il codice UniCode del carattere
  - `String getKeyText()`: sottolinea gli eventi dovuti a tasti come "Insert", "PageUp", ecc. [non i modificatori come "Shift" o "Ctrl"]
- `WindowEvent`:
  - eventi possibili:
    - `WINDOW_ACTIVATED`
    - `WINDOW_CLOSED`
    - `WINDOW_CLOSING`
    - `WINDOW_DEACTIVATED`
    - `WINDOW_DEICONIFIED`
    - `WINDOW_GAINED_FOCUS`
    - `WINDOW_ICONIFIED`
    - ecc.
  - metodi possibili:
    - `Window getWindow()`
    - `Window getOtherWindow()`

Esercizio

- Costruire un'interfaccia grafica per l'esercizio delle votazioni:
  - quali porzioni di codice bisogna mantenere intatte?
  - quali bisogna riscrivere?
  - quali si possono adattare?

Per chi ha pazienza (e voglia)…

Esercizio:

- Scrivere il programma Calcolatrice.java che realizza le funzionalità di una semplice calcolatrice. I requisiti dell'interfaccia grafica sono:
  - 10 bottoni con le cifre da 0 a 9 disposti come su una calcolatrice tradizionale;
  - i bottoni relativi alle operazioni di somma, sottrazione, moltiplicazione e divisione;
  - il bottone "CE" per cancellare l'ultimo numero battuto;
  - il bottone "C" per azzerare qualsiasi operazione;
  - il bottone "=" per richiedere il risultato;
  - il bottone "." per inserire cifre decimali;
  - una label per rappresentare il display della calcolatrice.

Osservazioni

- Modellare il comportamento di una "semplice" calcolatrice tascabile non è un'attività banale: per non rendere questo esercizio troppo pesante dal punto di vista degli algoritmi da usare, si può semplificare nel modo seguente l'algoritmo di calcolo delle espressioni aritmetiche introdotte con le seguenti ipotesi:
  - le espressioni coinvolgono sempre e solo 2 operandi;
  - l'utente inserisce sempre e solo il primo operando, l'operatore, il secondo operando ed il tasto "="; il risultato diventa il primo operando per l'operazione successiva
- l'unica eccezione alla regola (2) avviene per i bottoni "C", "CE" ed "Off" che possono essere premuti in qualsiasi momento.

Algoritmo di gestione (1)

```
stato = NESSUN_OPERATORE
buffer = 0
per ogni bottone premuto:
   se lo stato è NESSUN_OPERATORE allora
        se il bottone premuto è una cifra allora
                accoda a buffer la cifra
                visualizza il buffer
        se il bottone premuto è un segno allora
                operatore1 = buffer
                operando = segno
                buffer = 0
                stato = UN_OPERATORE
        se il bottone premuto è C oppure CE
                buffer = 0
                visualizza il buffer
   altrimenti se lo stato è UN_OPERATORE allora
        se il bottone premuto è una cifra allora
                accoda a buffer la cifra
                visualizza il buffer
        se il bottone premuto è "=" allora
                operatore2 = buffer
                calcola l'espressione operatore1,segno,operatore2 e visualizzala
                buffer = 0
                stato = NESSUN_OPERATORE
```

Algoritmo di gestione (2)

```
se il bottone premuto è C
       buffer = 0
       visualizza il buffer
       stato = NESSUN_OPERATORE
se il bottone premuto è CE
       buffer = 0
       visualizza il buffer
```

Applet

Motivazione

- pagine html sono statiche
- applet aggiungono
  - capacita' di elaborazione
  - interazione dinamica con utente

Applet Java

- Programmi che richiedono un browser per essere eseguiti (utilizzano ambiente di run-time del browser)
- Generalmente di piccole dimensioni
- Soggetti a restrizioni di security (sandbox): i browser possono impedire loro di
  - lanciare altre applicazioni
  - accedere ai file system locali
  - accedere a informazioni sul sistema su cui girano
  - comunicare liberamente via rete

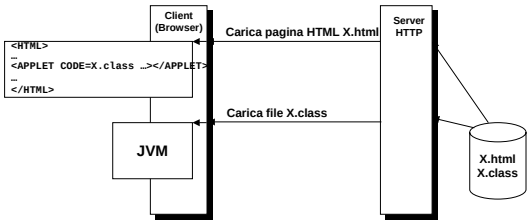
Vincoli

- per questioni di security
  - no scrittura/lettura su file system locale
  - no comunicazione se non server da cui sono state downloaded
  - no fork di processi in locale, no esecuzione programmi locali

Funzionamento

- tag <applet> nella pagina html
- il browser fa download delle classi java necessarie
- VM genera istanza della classe iniziale
- <applet code = "x.class">
  - non c'e' main
- browser chiama i metodi su questa istanza, in funzione eventi o chiamate di altri metodi

Esecuzione di Applet Java



- <html>
- <applet code="WelcomeApplet.class" width=300 height=30>
- </applet>
- </html>

```
import javax.swing.JApplet;  // import class JApplet
import java.awt.Graphics;    // import class Graphics

public class WelcomeApplet extends JApplet {
  public void paint( Graphics g )
  {
    g.drawString( "Welcome to Java Programming!", 25, 25 );
    g.drawLine( 15, 10, 210, 10 );
    g.drawLine( 15, 30, 210, 30 );     }}
```

Es: I am a simple program

- Come applicazione


```
import java.awt.*; import java.awt.event.*;import javax.swing.*;
public class SwingUI extends JFrame implements ActionListener {
   JLabel text, clicked;   JButton button, clickButton;   JPanel panel;
        private boolean clickMeMode = true;
  public SwingUI() {       //Begin Constructor
          text = new JLabel("I'm a Simple Program");
          button = new JButton("Click Me");
          button.addActionListener(this);

          panel = new JPanel();     panel.setLayout(new BorderLayout());
          panel.setBackground(Color.white);     getContentPane().add(panel);
          panel.add(BorderLayout.CENTER, text);
          panel.add(BorderLayout.SOUTH, button);
  } //End Constructor
```

```
public void actionPerformed(ActionEvent event){
   //   Object source = event.getSource();
      if (clickMeMode) {
        text.setText("Button Clicked");
        button.setText("Click Again");
        clickMeMode = false;
      } else {
        text.setText("I'm a Simple Program");
        button.setText("Click Me");
        clickMeMode = true;
      }    }
}
```


Es: I am a simple program

- Trasformazione in Applet

modifica

```
import java.awt.*; import java.awt.event.*;import javax.swing.*;
public class SwingUI extends JApplet implements ActionListener {
  JLabel text, clicked;  JButton button, clickButton;  JPanel panel;
     private boolean clickMeMode = true;
 public SwingUI() {       //Begin Constructor
        text = new JLabel("I'm a Simple Applet");
        button = new JButton("Click Me");
        button.addActionListener(this);

        panel = new JPanel();    panel.setLayout(new BorderLayout());
        panel.setBackground(Color.white);    getContentPane().add(panel);
        panel.add(BorderLayout.CENTER, text);
        panel.add(BorderLayout.SOUTH, button);
 } //End Constructor
```

```
public void actionPerformed(ActionEvent event){
  //   Object source = event.getSource();
     if (clickMeMode) {
       text.setText("Button Clicked");
       button.setText("Click Again");
       clickMeMode = false;
     } else {
       text.setText("I'm a Simple Program");
       button.setText("Click Me");
       clickMeMode = true;
     }   }
public void init(){
   SwingUI s = new SwingUI();
 }
}
```

```
<html>
<applet code="SwingUI.class" width=300 height=30>
</applet>
</html>
```

```
public class AdditionApplet extends JApplet {
double sum;
public void init()
{     String firstNumber,  secondNumber;   double number1,  number2;
      // read in numbers from user
  firstNumber = JOptionPane.showInputDialog("Enter  floating-point value" );
secondNumber =JOptionPane.showInputDialog("Enter floating-point value" );
  // convert numbers from type String to type double
  number1 = Double.parseDouble( firstNumber );
  number2 = Double.parseDouble( secondNumber );
  sum = number1 + number2; }

public void paint( Graphics g )
{    g.drawRect( 15, 10, 270, 20 );
  g.drawString( "The sum is " + sum, 25, 25 );   }}
```


Metodi di base

- Ereditati vuoti (da javax.swingJApplet o java.awt.Applet) o override
  - init() // inizializza l'istanza (in pratica sostituisce costruttore)
  - start() // fa partire l'esecuzione dell'istanza - in genere chiamato dopo init() o dopo stop() (utente torna alla pagina)

- stop()// sospende esecuzione (chiamato quando utente abbandona la pagina in cui applet sta girando)
- destroy() // distrugge l'istanza, rilascia le risorse (quando browser termina)
- paint() // aggiorna la parte di schermo controllata da applet

Ordine chiamata

- init()
- start()
- paint()

- chiamati nell'ordine da appletViewer o browser

Passaggio parametri browser - applet

- nel browser: uso del tag PARAM
- <Applet Code="myApplet.class" width=100 height=100 >
- <param name=font value="TimesRoman">
- <param name=size value="36">
- </applet>

- nell'applet
- metodo getParameter() // tipicamente in init()
- riceve stringa con nome parametro, rende stringa con valore, o null
- String s1 = getParameter("font");
- String s2 = getParameter("size");

Esempio MouseListener (1)

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="MouseEventsApplet" width=300 height=100>
</applet>
*/

public class MouseEventsApplet extends Applet implements
  MouseListener, MouseMotionListener {

  String msg = "";
  int mouseX = 0, mouseY = 0;

  public void init() {
      addMouseListener(this);
      addMouseMotionListener(this);
  }
```


Esempio MouseListener (2)

```
public void mouseClicked(MouseEvent me) {
      mouseX = 0;
      mouseY = 10;
      msg = "Rilevato click del mouse.";
      repaint();
  }

  public void mouseEntered(MouseEvent me) {
      mouseX = 0;
      mouseY = 10;
      msg = "Il mouse e' nella zona sensibile.";
      repaint();
  }

  public void mouseExited(MouseEvent me) {
      mouseX = 0;
      mouseY = 10;
      msg = "Il mouse e' fuori dalla zona sensibile.";
      repaint();
  }
```


Esempio MouseListener (3)

```
  public void mousePressed(MouseEvent me) {
      mouseX = me.getX();
      mouseY = me.getY();
      msg = "Mouse premuto";
      repaint();
  }

  public void mouseReleased(MouseEvent me) {
      mouseX = me.getX();
      mouseY = me.getY();
      msg = "Mouse rilasciato";
      repaint();
  }

  public void mouseDragged(MouseEvent me) {
      mouseX = me.getX();
      mouseY = me.getY();
      msg = "*";
      showStatus("Si sta trascinando il mouse in coordinate " +
                 mouseX + ", " + mouseY);
```

Esempio MouseListener (4)

```
        repaint();
    }

    public void mouseMoved(MouseEvent me) {
        showStatus("Si sta muovendo il mouse in coordinate " +
    me.getX() + ", " + me.getY());
    }

    public void paint(Graphics g) {
        g.drawString(msg, mouseX, mouseY);
    }
}
```