Laboratorio di sistemi interattivi

Lezione 8: Note su AWT e Swing

Generazione di interfacce utente

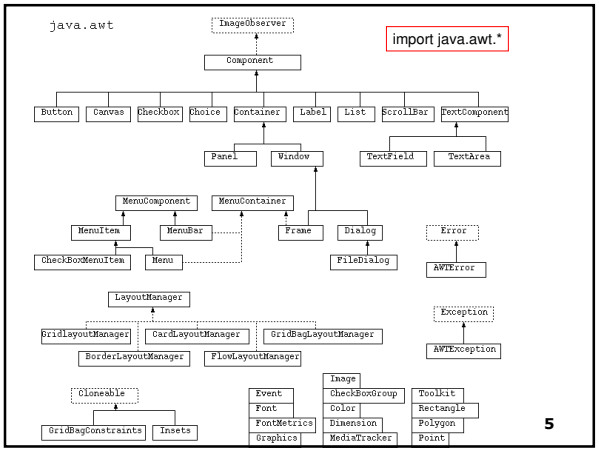
- La GUI viene composta aggiungendo componenti a oggetti **Container**
- Occorre definire opportuni **gestori di eventi** per reagire a interazione utente con GUI
- Occorre rendere **visibili** gli elementi della GUI

GUI: cosa c'è da fare

- **Graphical Component** costituiscono la Graphical User Interface.
- **Listener method** ricevono gli eventi e rispondono.
- **Application method** svolgono le operazioni richieste dall'utente.

AWT – Abstract Windowing Toolkit vs SWING

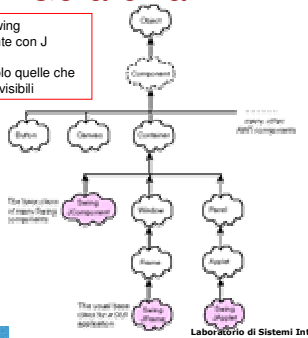
- Tutte le classi Java GUI sono basate sulle classi fondamentali che si trovano nell' *Application Windowing Toolkit*, o AWT.
- Le classi fondamentali di AWT sono usate come base di un insieme più datato di componenti.
- I componenti Swing arrivano con Java release 1.1.2 e sono inclusi in tutte le release successive.
- I componenti AWT più vecchi sono simili a quelli Swing, ma i componenti Swing sono più versatili
- I componenti Swing si integrano facilmente con **Java Foundation Classes** (fornisce molte classi utilizzate per il moderno sviluppo software).
- Sun Microsystems raccomanda l'uso di Swing per le nuove applicazioni.



Gerarchia AWT

Nomi classi UI di Swing iniziano generalmente con J

Le classi UI sono solo quelle che creano componenti visibili



Gerarchia AWT (Container)

- Component
  - Container
    - **Window**: window top-level
      - **Dialog**: input da utente
      - **Frame**: window top-level con titolo
    - **Panel**: contenitore generico
      - Applet: piccolo programma immerso in un'altra applicazione - deve essere superclasse di qualunque applet incluso in una pagina web in quanto fornisce un'interfaccia standard tra l'applet e il suo ambiente.
    - **ScrollPane**: implementa scrolling automatico orizzontale e/o verticale per un singolo componente figlio

Gerarchia AWT (Container)

- Component
  - Container
    - Window
      - Dialog
      - Frame
    - Panel
      - Applet
    - Scrollpane

Esempio: Class java.awt.Dialog

```
java.lang.Object
 |
+----java.awt.Component
          |
          +----java.awt.Container
                    |
                    +----java.awt.Window
                              |
                              +----java.awt.Dialog
```

Dialog extends Window

Gerarchia AWT (Component)

- **Component**
  - **Button**: il gesto di cliccare un bottone con il mouse è associato con un'istanza di **ActionEvent**, che viene emessa quando il muse viene sia premuto che rilasciato sul bottone. Se un'applicazione è interessata a sapere quando il bottone è stato premuto ma non rilasciato, come gesto distinto, può specializzare **processMouseEvent**, o può registrarsi come listener per gli eventi legati al mouse chiamando **addMouseListener**. Entrambi i metodi sono definiti da **Component**, la superclasse astratta di tutti i componenti.
  - **Canvas**: un'area rettangolare vuota sullo schermo in cui l'applicazione può disegnare o catturare eventi utente
  - **Checkbox**: componente grafico che può essere "on" (true) o "off" (false) (cambia con click)
  - **Choice:** presenta un menu pop-up di scelte. La scelta corrente è mostrata come titolo
  - **Container:** visti prima
  - **Label**: componente per inserire testo (una singola linea read-only) in un contenitore
  - **List**: lista di item di testo, inizializzata in modo che se ne possa selezionare uno o più
  - **Scrollbar**: permette all'utente di selezionare da un insieme di valori (slider)
  - **TextComponent**: superclasse di ogni componente che permette di editare del testo
    - **TextArea**: regione multi-linea che mostra testo (editabile o read-only)
    - **TextField**: permette di editare una singola linea di codice

AWT Checkbox

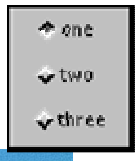

```
setLayout(new GridLayout(3, 1));
add(new Checkbox("one", null, true));
add(new Checkbox("two"));
add(new Checkbox("three"));
```

CheckboxGroup

AWT CheckboxGroup

In alternativa, più check box possono essere raggruppati sotto il controllo di un singolo oggetto usando la classe **CheckboxGroup**.
In un check box group, al massimo un button può essere "on" in ogni momento. Cliccare su un check box in un gruppo per settarlo on forza tutti gli altri nello stesso gruppo a diventare off

java.lang.Object
|
└ **java.awt.CheckboxGroup**

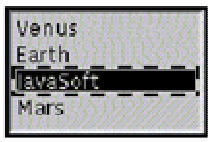



```
setLayout(new GridLayout(3, 1));
 CheckboxGroup cbg = new CheckboxGroup();
add(new Checkbox("one", cbg, true));
add(new Checkbox("two", cbg, false));
add(new Checkbox("three", cbg, false));
```


AWT Choice


```
Choice ColorChooser = new Choice();
ColorChooser.add("Green");
ColorChooser.add("Red");
ColorChooser.add("Blue");
```

AWT List



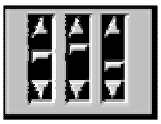

```
List lst = new List(4, false);
lst.add("Mercury");
lst.add("Venus");
lst.add("Earth");
lst.add("JavaSoft");
lst.add("Mars");
lst.add("Jupiter");
lst.add("Saturn");
lst.add("Uranus");
lst.add("Neptune");
lst.add("Pluto");
cnt.add(lst);
```

selezione multipla?

numero elementi visibili

container

AWT Scrollbar



Es. slider per valori RGB – per ognuno

```
redSlider=new Scrollbar(Scrollbar.VERTICAL, 0, 1, 0, 255);
add(redSlider);
```

Scrollbar(int orientation, int init_value, int visible, int minimum, int maximum)

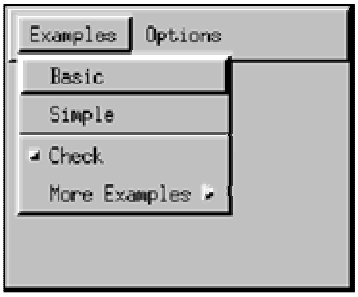
Es. range di valori

```
ranger = new Scrollbar(Scrollbar.HORIZONTAL, 0, 60, 0, 300);
add(ranger);
```

Altre componenti AWT

- **MenuItem**: tutti gli item in un menu devono appartenere alla classe MenuItem, o a una delle sue sottoclassi.
  - **CheckboxMenuItem**: una check box che può essere inclusa in un menu
  - **Menu**: un componente menu pull-down che può essere mostrato da una barra di menu

AWT Menu



Menu con 5 item.
I primi due item sono menu item semplici, con label "Basic" e "Simple".
Di seguito c'è un separatore, anche lui un mnu item creato con la label "-".
Poi c'è una istanza di CheckboxMenuItem con label "Check".
Il menu item finale è un sottomenu con label "More Examples", e questo sottomenu è un' instanza di Menu.

Creazione Eventi

- AWTEvent (**AWTEvent**(Object source, int id))
  - Metodi
    - getSource, getId, isConsumed
  - **ActionEvent** (getWhen): è avvenuta una azione definita in un componente (quando= timestamp)
  - **AdjustmentEvent** (getValue): evento di aggiustamento in un oggetto (valore corrente dell'evento di correzione)
  - **AncestorEvent** (getAncestor): evento avvenuto in un ancestor riportato ad un discendente (identità dell'ancestor)
  - **ComponentEvent** (getComponent): evento di basso livello che indica che un componente ha cambiato posizione, taglia o visibilità (originatore)
  - **HierarchyEvent** (getChangedFlags): indica una variazione nella gerarchia cui il componente appartiene (bitmask con tipo eventi che si sono verificati)
  - **InputMethodEvent** (getText): informazione sul testo che viene composto usando un metodo di input
  - **InvocationEvent** (dispatch): esegue il metodo run() su un Runnable quando fatto partire dal thread AWT event dispatcher
  - **ItemEvent** (getStateChange):indica che un item è stato selezionato o deselezionato
  - **TextEvent** (paramString): indica che un oggeto testo è stato modificato

Gestione Eventi

- Per ogni XXXEvent
  - Interfaccia XXXListener
  - Metodi specifici invocati in base al tipo di evento

Gestione Layout

Container ha metodo
setLayout(LayoutManager lm)

Component ha metodi
Dimension getMaximumSize(), Dimension getMinimumSize(), Dimension getPreferredSize() e analoghi metodi set

```
public interface LayoutManager {
    void addLayoutComponent(String name, Component comp)
    void layoutContainer(Container parent)
    Dimension minimumLayoutSize(Container parent)
    Dimension preferredLayoutSize(Container parent)
    void removeLayoutComponent(Component comp)
}
```

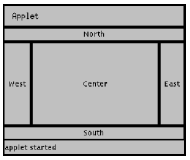

Dinamica gestione Layout

- Layout installato sul Container
- Cambiamento di visibilità, composizione, o esplicita richiesta da programma, provoca chiamata di layoutContainer().
- Si ottengono Dimensioni del Container
- Tutte le componenti su cui fare layout sono interrogate per le loro dimensioni.
- Componenti distribuite su spazio Container in base ai vincoli associati

Implementazioni del Layout

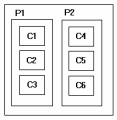
- Specifiche a Container particolari
  - ComboBoxLayoutManager, TitlePaneLayout, InternalFrameLayout, ButtonAreaLayout, BasicScrollBarUI, DividerLayout, BasicHorizontalLayoutManager,, BasicVerticalLayoutManager, TabbedPaneLayout, DefaultMenuLayout, ScrollPaneLayout, ViewportLayout
- Riutilizzabili
  - BorderLayout
  - BoxLayout
  - CardLayout
  - FlowLayout
  - GridLayout
  - GridBagLayout
  - SpringLayout

BorderLayout




```
public class buttonDir extends Applet {
    public void init() {
            setLayout(new BorderLayout());
            add(new Button("North"), BorderLayout.NORTH);
            add(new Button("South"), BorderLayout.SOUTH);
            add(new Button("East"), BorderLayout.EAST);
            add(new Button("West"), BorderLayout.WEST);
            add(new Button("Center"), BorderLayout.CENTER);
    }
```


BoxLayout

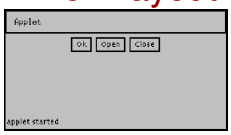


- Componenti presentate orizzontalmente o verticalmente.
- Ordine di aggiunta al container determina ordine di presentazione
- Dimensioni delle componenti variano, tentando di preservare la direzione

CardLayout

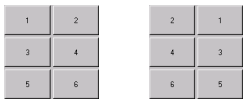
- Componenti presentate una alla volta
- Ordine di aggiunta determina ordine di presentazione
- Metodi first, last, next, previous
- void **show**(Container parent, String name) per accesso diretto

FlowLayout



- Elementi arrangiati orizzontalmente fino a riempire una linea
- Nuove linee aggiunte quando dimensione container insufficiente
- Orientamento sinistra/destra o viceversa
- Ordine di presentazione legato a ordine di aggiunta al container.
- Costanti determinano organizzazione linee

GridLayout



- GridLayout(int rows, int cols)
- Spazio container griglia con celle di uguale grandezza
- Una componente per ogni cella
- Tutte le componenti vengono presentate.
- Numero righe determinante
- Numero colonne significativo solo quando righe =0
- Quando sia il numero di righe che di colonne sono diversi da zero, il numero di colonne è ignorato, ma quello visualizzato viene determinato dal numero di righe e dal numero di componenti da visualizzare. Ad esempio, se sono state specificate 3 righe e due colonne ma occorre visualizzare nove componenti, verranno create tre righe e tre colonne.

GridBagLayout

- Ne parliamo più avanti …

SpringLayout

- Definisce relazioni direzionali fra bordi di componenti
- Es. lato sinistro distante 5 pixel da lato destro di altra componente
- Un solo vincolo per ogni lato

void putConstraint(String e1, Component c1, Spring s, String e2, Component c2)

Sistema di "molle" mantiene i vincoli

Ogni edge appartiene all'insieme SpringLayout.NORTH, SpringLayout.SOUTH, SpringLayout.EAST, SpringLayout.WEST

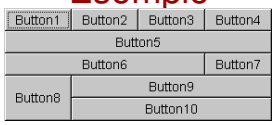
GridBagLayout

- Griglia virtuale. Componenti possono estendersi su più celle.
- Righe e colonne non hanno grandezza uniforme
- Fattori principali: numero di componenti nella linea più lunga, numero di righe e grandezza delle componenti per determinare numero e grandezza delle celle.

GridBagConstraints

- Per ogni componente da disporre vengono usati dei vincoli, mantenuti in un oggetto GridBagConstraints (addLayoutComponent(Component comp, Object constraints))
  - gridx/gridy: posizione componente (cella partenza), valori interi o RELATIVE (prossimo nel layout)
  - gridwidth/gridheight: numero di celle per la componente. Valori interi o RELATIVE (prossimo) o REMAINDER (ultimo)
  - weightx/weighty: espansione in caso di ridimensionamento
  - fill: come la componente riempie o meno la cella. HORIZONTAL, VERTICAL, BOTH, NONE
  - anchor: posizionamento della cella nell'area, se di dimensioni minori
  - ipadx/ipady: distanza fra componente e bordo cella
  - insets: distanza fra componenti

Esempio



Ognuno dei dieci componenti ha il campo fill del GridBagConstraints object associato settato a GridBagConstraints.BOTH. Inoltre, i componenti hanno:

Button1, Button2, Button3: weightx = 1.0
Button4: weightx = 1.0, gridwidth = GridBagConstraints.REMAINDER
Button5: gridwidth = GridBagConstraints.REMAINDER
Button6: gridwidth = GridBagConstraints.RELATIVE
Button7: gridwidth = GridBagConstraints.REMAINDER
Button8: gridheight = 2, weighty = 1.0
Button9, Button 10: gridwidth = GridBagConstraints.REMAINDER

Codice completo alla pagina
http://java.sun.com/j2se/1.4.2/docs/api/java/awt/GridBagLayout.html

Schema costruzione layout

- Decidere taglia e posizione dei componenti.
- Creare un oggetto GridBagLayout e uno GridBagConstraints.
- Il contenitore usa l'oggetto the GridBagLayout.
- Customizzare i settaggi di GridBagConstraints (o usare i default).
- Creare il Component A.
- Passare il Component A e l'oggetto GridBagConstraints all'oggetto GridBagLayout.
- Aggiungere il Component A al container. Il punto precedente ha comunicato al layout del contenitore quali settaggi usare per il componente.
- Creare il Component B.
- Modificare i settaggi dell'oggetto GridBagConstraints o lasciarli immutati.
- Passare il Component B e l'oggetto GridBagConstraints all'oggetto GridBagLayout.
- Aggiungere il Component B al contenitore.
- eccetera . . . .

Disegno in AWT

Component mette a disposizione metodi
Graphics getGraphics(): crea un contesto grafico per il componente
void paint(Graphics g): disegna il componente, ad esempio quando viene mostrato per la prima volta

Graphics = classe astratta, quindi le applicazioni non possono richiamare il costruttore direttamente. I contesti grafici possono essere creati a partire da altri contesti grafici o richiamando getGraphics su un componente (Component.getGraphics()).
Un oggetto Graphics incapsula informazioni di stato richieste per il rendering, tra cui:

- L'oggetto Component su cui disegnare.
- Un'origine della traslazione per le coordinante di rendering e aggancio.
- L'oggetto grafico (clip) corrente.
- Il colore corrente.
- Il font corrente.
- L'operazione logica corrente su pixel (XOR o Paint (overwrite)).
- Il colore attuale di alternanza per XOR (quando si disegna si passa da quello corrente a questo e viceversa).

Metodi per disegno

- Forme vuote: draw3DRect, drawArc, drawBytes, drawChars, drawImage, drawLine, drawOval, drawPolygon, drawPolyline, drawRect, drawRoundRect
- Versioni fill delle forme: fill3DRect, fillArc, ecc.
- Definizione del contesto di disegno
  - setClip(shape o rect), setColor(color), setFont(font), setPaintMode(), setXORMode(color)

Disegno con Java2D

- Java 2D API è un insieme di classi per grafica 2D avanzata
- I seguenti package si trovano nel package AWT ma in realtà sono parte di Java2D
  - java.awt.color
  - java.awt.font
  - java.awt.geom
  - java.awt.image
  - java.awt.image.renderable
  - java.awt.print

Disegno con Java2D

- Classi per rappresentare oggetti grafici
  - Interface Shape
    - contains, getBounds() (ritorna enclosing rectangle), getPathIterator, intersects
  - Classi
    - RectangularShape (astratta)
      - Rectangle2D RoundRectangle2D, Ellipse2D, Arc2D
        - » Rectangle, Arc2D.Double, Arc2D.Float, Ellipse2D.Double, Ellipse2D. Float
    - Line2D
    - …

Disegno diretto di elementi

- Graphics
  - Graphics2D
    - draw e fill chiamati con un oggetto Shape passato come argomento

```
public class MyCanvas extends Canvas {
   Shape[] drawings;
   public void paint(Graphics g) {
        if (!g instanceof Graphics2D) throw new Exception();
        Graphics2D g2d = (Graphics2D)g;
        int ln = drawings.length;
        for(int i=0; i<=ln, i++) g.draw(drawings[i]);
   }
}
```


Gestione disegno

- Oggetti Shape non hanno connessione diretta con il disegno.
- Dopo modifica di una Shape occorre richiamare paint()
- Disegno non fornisce supporto di interazione
  - Eventi generati sulla canvas vanno interpretati in relazione alle proprietà degli oggetti Shape