

Grafica con Groovy (autore: Vittorio Albertoni)

Premessa

Possiamo ritenere il linguaggio di programmazione Groovy una semplificazione del linguaggio Java. Con esso, infatti, scriviamo, con una sintassi molto più semplice di quella richiesta da Java, programmi che girano sulla Java Virtual Machine e che hanno la stessa potenza di programmi scritti in linguaggio Java, anche se senza essere all'altezza di affrontare progetti molto impegnativi con le garanzie di affidabilità di Java.

Sul mio blog, all'indirizzo www.vittal.it, è disponibile un manualetto sulle basi di questo linguaggio, in formato PDF, intitolato «groovy» e allegato all'articolo «Groovy: Java stenografato» pubblicato nel novembre 2025, da considerarsi propedeutico a questo manualetto.

Dal momento che tutte le librerie Java sono a disposizione di Groovy, lo sono anche le librerie grafiche.

Nel campo della grafica un primo obiettivo può essere quello di realizzare interfacce grafiche per i nostri programmi, le così dette GUI (Graphical User Interface).

In questo caso la ponderosa parte del programma per elaborazioni varie di natura non grafica è convenientemente scrivibile nel relativamente semplice linguaggio Groovy e la grafica serve solo per realizzare un interfacciamento con l'utente un tantino più simpatico della fredda riga di comando.

Per fare questo non dobbiamo affrontare eccessive difficoltà, soprattutto se ci accontentiamo di cose normali. E anche la necessaria integrazione tra linguaggio Groovy e linguaggio Java anche per cose un tantino più complicate non è poi così difficile.

Altro obiettivo può essere quello di creare grafica del tipo di quella che vivacizzava le applet Java di buona memoria, quella grafica che utilizza la libreria Java Graphics, con cui realizzare disegni e figure di varia natura.

In questo caso la parte preponderante del programma è quella della realizzazione del disegno, che va scritta in linguaggio Java, e Groovy serve solo per far vedere ciò che abbiamo programmato.

Teoricamente con questa libreria potremmo realizzare grafici di statistica descrittiva (istogrammi, torte, ecc.) ma il lavoro di programmazione sarebbe improponibile.

Purtroppo, infatti, Java non ci offre strumenti per fare queste cose in maniera semplice e servirebbero librerie esterne di arricchimento, come `jfreechart` o simili, il cui utilizzo esula dalle capacità di un dilettante.

In questo manualetto vediamo semplicemente come arricchire i nostri programmi con GUI e come creare qualche giochetto grafico.

Indice

1	Contesto grafico Java in Groovy per interfacce grafiche	3
2	Oggetti contenitori	5
3	Colori	7
4	Font	8
5	Componenti indispensabili per una GUI	9
5.1	JTextField	9
5.2	JLabel	9
5.3	JButton	9
6	Disegno della GUI	10
7	Eventi	11
8	Un programma di utilità	11
9	Menu	13
9.1	JMenuBar	13
9.2	JMenu	13
9.3	JMenuItem	14
10	Qualche altro esempio	15
11	Contesto grafico Java in Groovy per il disegno	18
12	Strumenti per disegnare	19
13	Realizzare il disegno	19
14	Esempio di grafica creativa	20
15	Esempio di grafica illustrativa	21
16	Animazioni	23

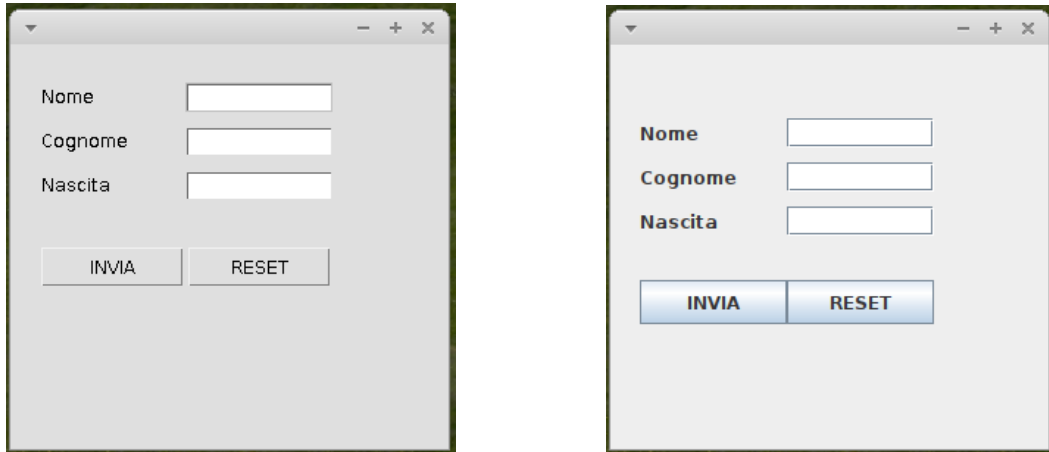
1 Contesto grafico Java in Groovy per interfacce grafiche

Il package Java originariamente dedicato agli strumenti per creare interfacce grafiche utente è **java.awt** (Java Abstract Widget Toolkit).

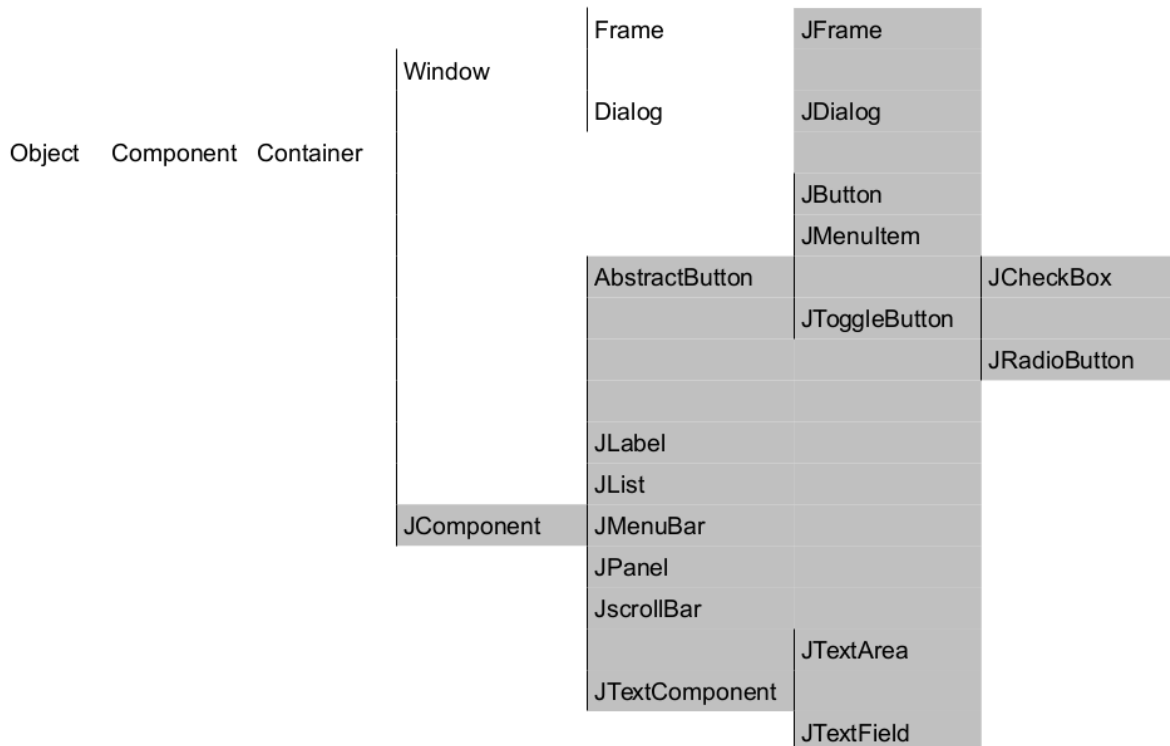
Gli strumenti grafici contenuti in questo package sono alquanto grezzi e, soprattutto, non garantiscono comportamenti uniformi su diversi sistemi operativi e su diversi hardware.

Per un sistema che si fregia del motto «write once run anywhere» non è il massimo e si è rimediato con l'apprestamento di un altro package che integra java.awt, che si chiama **javax.swing** e che risolve questi problemi.

Queste due illustrazioni fanno vedere la differenza di resa grafica tra un programma che utilizza java.awt (sulla sinistra) e un programma che utilizza javax.swing (sulla destra).



La gerarchia di classi che evidenzia l'integrazione tra le componenti grafiche dei due packages è la seguente:

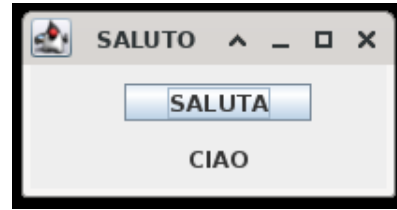


La zona ombreggiata rappresenta le classi di javax.swing che sostituiscono classi di java.awt: praticamente esse hanno lo stesso nome con una J davanti.

Il pulsante si chiama Button in java.awt e si chiama JButton in javax.swing.

Poniamo che il nostro obiettivo sia di creare una semplicissima GUI nella quale cliccando su un pulsante veniamo salutati con un Ciao.

Una cosa di questo tipo



con sulla sinistra la GUI al suo lancio e sulla destra la GUI dopo aver cliccato sul pulsante SALUTA.

Utilizzando l'apparato di scripting di Groovy abbiamo tre modi di raggiungere questo obiettivo.

con linguaggio Java

L'interprete Groovy riconosce il linguaggio Java puro. Pertanto, se conosciamo Java e le sue librerie grafiche, possiamo stilare lo script in questo modo

```
import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
class Saluto
{
    public static void main(String[] args)
    {
        JFrame mioFrame = new JFrame("SALUTO")
        mioFrame.setLayout(null)
        mioFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)
        mioFrame.setBounds(200,200,200,100)
        JButton pulsante = new JButton("SALUTA")
        pulsante.setBounds(50,10,100,20)
        mioFrame.add(pulsante)
        JLabel messaggio = new JLabel()
        messaggio.setBounds(50,40,100,20)
        messaggio.setHorizontalAlignment(JLabel.CENTER)
        mioFrame.add(messaggio)
        pulsante.addActionListener(new ActionListener()
        {
            @Override
            public void actionPerformed(ActionEvent e)
            {
                messaggio.setText("CIAO");
            }
        });
        mioFrame.setVisible(true)
    }
}
```

Le indentazioni non servirebbero ma le ho usate, e lo farò sempre in questo manualetto, per rendere più evidente la struttura dello script.

La cosa più ostica per un principiante è la macchinosa gestione dell'evento corrispondente alla pressione del pulsante.

Per il resto il tutto è abbastanza abordabile, anche perché ho evitato di ricorrere alla complicata gestione del layout attraverso il layout manager di Java.

con linguaggio Groovy

Pur non potendo prescindere dalla conoscenza della libreria `javax.swing` e dei suoi meccanismi possiamo redigere lo script senza conoscere il linguaggio Java in questo modo

```
import javax.swing.*
mioFrame = new JFrame("SALUTO")
mioFrame.setLayout(null)
mioFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)
mioFrame.setBounds(200,200,200,100)
JButton pulsante = new JButton("SALUTA")
pulsante.setBounds(50,10,100,20)
mioFrame.add(pulsante)
JLabel messaggio = new JLabel()
messaggio.setBounds(50,40,100,20)
messaggio.setHorizontalAlignment(JLabel.CENTER)
mioFrame.add(messaggio)
pulsante.addActionListener
{
    messaggio.setText("CIAO")
}
mioFrame.setVisible(true)
```

Qui abbiamo la gestione molto facilitata dell'evento corrispondente alla pressione del pulsante.

Per il resto come prima.

con il tool Swing Builder

Fedele alla sua vocazione stenografica, Groovy ci offre un tool destinato a semplificare anche la costruzione e il maneggio dei widget di `javax.swing`.

E' il package `groovy.swing.SwingBuilder`

Utilizzando questo strumento lo script diventa il seguente.

```
import groovy.swing.SwingBuilder
import javax.swing.*
def swing = new SwingBuilder()
mioFrame = swing.frame(title:'SALUTO', location: [200,200], size:[200,100], layout: null)
{
    button(text:"SALUTA", location:[50,10], size:[100,20],actionPerformed:{messaggio.text ="CIAO"})
    messaggio = label(horizontalAlignment: JLabel.CENTER,location:[50,40], size:[100,20])
}
mioFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)
mioFrame.show()
```

Per la verità, come si può vedere dal raffronto tra lo script redatto con linguaggio Groovy e questo, la semplificazione è molto relativa e ritengo che, una volta acquisite le definizioni e i metodi dei widget del package `javax.swing`, sia molto più semplice redigere lo script con il linguaggio Groovy senza bisogno di imparare altro.

Nel seguito rammento pertanto i più importanti widget del package `javax.swing` e il modo di utilizzarli con il linguaggio Groovy.

2 Oggetti contenitori

L'oggetto contenitore previsto da Java per le GUI è il Frame (in italiano Telaio): noi usiamo quello della libreria Swing, che si chiama **JFrame**.

JFrame intelaia una finestra che ha l'aspetto di quelle che abbiamo visto nella pagina precedente.

Tipicamente abbiamo la barra superiore



che contiene il titolo della finestra e i pulsanti standardizzati per il trattamento della finestra in un sistema a finestre.

La parte sottostante è predisposta per accogliere una barra per il menu dell'applicazione e l'area che vediamo è predisposta per accogliere gli oggetti, chiamati componenti (noi useremo i JComponent della libreria Swing), con cui costruire l'applicazione: questa area si chiama `ContentPane`.

Uno dei componenti che possiamo inserire in quest'area è il **JPanel**, che è a sua volta un contenitore.

Per costruire GUI complesse l'utilizzo dei pannelli ci dà modo di suddividere il `ContentPane` in tante sotto-aree contenitrici, a ciascuna delle quali applicare una diversa geometria per la localizzazione dei componenti attraverso quello che in Java si chiama `LayoutManager`.

La gestione di tutto questo è tutt'altro che semplice e direi che per i nostri obiettivi dilettanteschi e per evitare eccessive complicazioni possiamo rinunciare all'utilizzo dei pannelli e del `LayoutManager`.

Costruiamo il nostro telaio con:

```
new JFrame()
```

e creiamo così una finestra senza titolo.

Per dare un titolo alla finestra possiamo inserirlo tra le parentesi tonde come stringa (tra apici semplici o doppi).

Il linguaggio Java prevede per questo oggetto decine di metodi e qui cito quelli che servono per fare le cose più importanti e meno ricercate.

```
setTitle(<stringa_titolo>)
```

per dare un titolo ad una finestra creata senza titolo o per modificarlo

```
setLocation(x, y)
```

per localizzare la finestra sullo schermo

(x e y sono numeri interi indicanti le coordinate del punto in cui si colloca l'angolo in alto a sinistra della finestra)

```
setSize(<larghezza>, <altezza>)
```

per dimensionare la finestra

(<larghezza> e <altezza> sono numeri interi indicanti rispettivamente i pixel di larghezza e di altezza della finestra)

```
setBounds(x, y, <larghezza>, <altezza>)
```

per indicare, ad un tempo, localizzazione e dimensione della finestra

```
setVisible(<valore_booleano>)
```

per rendere visibile (`true`) o nascondere (`false`) la finestra

```
add(<componente>)
```

per inserire un componente nella finestra

```
setLayout(LayoutManager <tipo_layout>)
```

per eventualmente scegliere un tipo di gestione del layout.

Come ho detto prima, noi dilettanti rinunciamo al layout management e, tra le parentesi tonde, indichiamo semplicemente la parola `null`

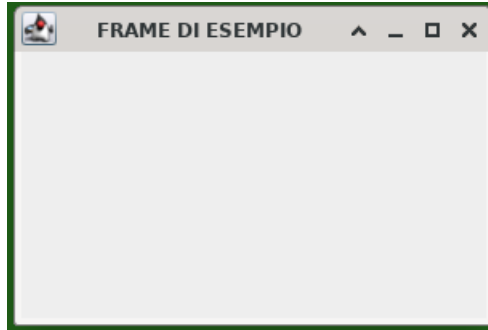
Nel caso non prevedessimo nella nostra GUI altri modi per chiudere la finestra e ci affidassimo a quello di default del sistema a finestre standard (cliccare sulla X in alto a destra della finestra), per chiudere anche il terminale da cui abbiamo aperta la finestra stessa lanciando lo script Groovy utilizziamo il metodo

```
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE).
```

Con questo script

```
import javax.swing.*
mioFrame = new JFrame("FRAME DI ESEMPIO")
mioFrame.setBounds(200, 100, 300, 200)
mioFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)
mioFrame.setVisible(true)
```

otteniamo la finestra



larga 300 pixel e alta 200 pixel, posizionata in alto a sinistra dello schermo, a 100 pixel dall'alto e a 200 pixel dal lato sinistro.

Come faremo per gli altri oggetti componenti che ci serviranno, utilizziamo il costruttore (`new JFrame("FRAME DI ESEMPIO")`) per assegnare l'oggetto ad una variabile (`mioFrame`), che eredita e viene ad essere dotata di tutti i metodi previsti dal linguaggio Java per l'oggetto assegnato alla variabile, metodi che si richiamano utilizzando la sintassi della programmazione per oggetti

`<nome_variabile>.<metodo>`

3 Colori

Se non si sceglie diversamente, come avviene per default in tutti i sistemi a finestre prodotti con i vari strumenti disponibili, ciò che facciamo viene reso con un unico colore di base e sue varianti: per esempio in nero le scritte e le linee, in grigio e relative sfumature le aree dei vari componenti.

Grazie al package `Java.awt` possiamo disporre di strumenti per scegliere colori diversi con cui abbellire e vivacizzare le nostre GUI.

Innanzitutto abbiamo i metodi di cui è dotato qualsiasi oggetto suscettibile di avere un colore:

`setBackground(<colore>)`

per indicare un colore per lo sfondo di un oggetto,

`setForeground(<colore>)`

per indicare un colore per le scritte contenute in un oggetto.

L'indicazione dell'argomento `<colore>` per questi metodi viene effettuata ricorrendo alla classe `Color` di `Java.awt` che può essere utilizzata come modificatore di colore o come costruttore di colore.

La sintassi del modificatore è

`Color.<nome_colore>`

dove `<nome_colore>` può essere `white`, `black`, `darkGray`, `gray`, `lightGray`, `blue`, `cyan`, `green`, `orange`, `magenta`, `yellow`.

La sintassi del costruttore è

`new Color(r, g, b)`

dove `r`, `g` e `b` sono numeri interi, compresi tra 0 e 255 indicanti, rispettivamente, l'intensità delle componenti di rosso, di verde e di blu nel colore desiderato.

Ricordo che `new Color(0,0,0)` indica nessun colore, cioè il colore nero e `new Color(255,255,255)` indica tutti i colori insieme, cioè il colore bianco.

Vediamo come possiamo colorare di verde la finestra che abbiamo costruito con lo script del precedente Capitolo.

Innanzitutto, oltre alla libreria `Javax.swing`, dobbiamo importare la libreria `Java.awt` con

```
import java.awt.*
```

Poi la prima idea che viene è di inserire, tra la prima e l'ultima istruzione dello script, l'istruzione

```
mioFrame.setBackground(Color.green)
```

Eseguendo lo script ci aspettiamo di vedere la nostra finestra colorata di verde e se compilassimo lo script esso si compilerebbe senza errori.

La finestra, tuttavia, si presenta con il colore standard di prima.

La mancanza di errori di compilazione è dovuta al fatto che l'istruzione è corretta, in quanto l'oggetto `JFrame` possiede il metodo `setBackground`.

Il fatto è che con questo metodo, applicato al frame, coloriamo lo sfondo del frame stesso quando l'ultima istruzione dello script (`setVisible(true)`) in realtà rende visibile il `ContentPane` del frame.

E' pertanto su questo che dobbiamo agire e lo facciamo con l'istruzione

```
mioFrame.getContentPane().setBackground(Color.green)
```

In questo modo otteniamo la finestra



Per quanto detto, arriveremmo allo stesso risultato con l'istruzione

```
mioFrame.getContentPane().setBackground(new Color(0,255,0))
```

in cui esprimiamo il colore con il costruttore anziché con il modificatore.

Ovviamente il vantaggio del costruttore è quello di consentirci di creare qualsiasi colore, con qualsiasi sfumatura e la nostra creatività non è più limitata dal dover usare solo i colori riconosciuti dal modificatore.

4 Font

Per default Java scrive i caratteri utilizzando il font Tahoma, di normale aspetto e dimensionato sugli 11 punti: una cosa esteticamente accettabile.

Se la nostra creatività ha altre esigenze esiste un facile modo per cambiare questa impostazione di default ed è quello di utilizzare il costruttore

```
new Font(<stringa_nome>, Font.<serie>, <punti>)
```

dove

<stringa_nome> è il nome del font tra apici,

<serie> è l'aspetto del carattere e può essere `PLAIN`, `BOLD` o `ITALIC`,

<punti> è un numero intero che indica la dimensione del carattere.

Il font di cui indichiamo il nome deve essere uno di quelli installati sul sistema su cui lavoriamo.

Se così non fosse verrebbe ignorata l'indicazione <stringa_nome> e, fatte salve le altre due indicazioni, verrebbe utilizzato il font di default Tahoma.

Secondo la sintassi Groovy l'oggetto font così costruito lo assegniamo ad una variabile.

Qualsiasi oggetto contenente una scritta è dotato del metodo

`setFont()`

tra le cui parentesi tonde richiamiamo il nome della variabile cui abbiamo assegnato il font che vogliamo utilizzare in quell'oggetto.

5 Componenti indispensabili per una GUI

La GUI serve per acquisire informazioni dall'utente, per dare informazioni all'utente e per avere dall'utente istruzioni su quali compiti svolgere.

La libreria Swing ci fornisce tre strumenti atti a questi scopi: **JTextField** per il primo, **JLabel** per il secondo e, ma non solo, **JButton** per il terzo.

5.1 JTextField

E' una finestra che serve per immettere una sola riga di testo o un solo numero.

La costruiamo con

```
new JTextField()
```

Per default viene costruita con sfondo bianco e si può cambiare questo colore con il metodo `setBackground()` secondo quanto indicato nel Capitolo 3.

Per default si allinea a sinistra il testo inserito e si può cambiare questo con il metodo `setHorizontalAlignment(JTextField.<tipo_allineamento>)` dove `<tipo_allineamento>` può essere `LEFT`, `CENTER` o `RIGHT`.

Il metodo

```
getText()
```

legge il contenuto della finestra e lo ritorna come stringa.

5.2 JLabel

E' una finestra per esporre testo o numeri.

La costruiamo con

```
new JLabel()
```

All'interno delle parentesi possiamo indicare una stringa di testo da esporre già al momento della costruzione.

Per default acquisisce il colore di sfondo del `contentPane` e usa il colore nero per le scritte. Si può scegliere un colore diverso per le scritte con il metodo `setForeground()` secondo quanto indicato nel Capitolo 3.

Quanto al carattere tipografico della scritta vale quanto detto nel precedente Capitolo 4.

Il metodo

```
setText(<stringa>)
```

inserisce testo nella label, sostituendo il testo che eventualmente c'era prima.

Per default si allinea a sinistra il testo inserito e si può cambiare questo con il metodo `setHorizontalAlignment(JLabel.<tipo_allineamento>)` dove `<tipo_allineamento>` può essere `LEFT`, `CENTER` o `RIGHT`.

5.3 JButton

E' un pulsante sul quale si clicca con il mouse per far fare qualche cosa al computer.

Lo costruiamo con

```
new JButton()
```

All'interno delle parentesi possiamo indicare una stringa di testo da esporre sul pulsante già al momento della costruzione.

Allo stesso modo possiamo anche indicare l'indirizzo di una icona da esporre sul pulsante.

Con i metodi `setText()` e `setIcon()` possiamo inserire dopo la costruzione una scritta o una icona o modificare quelle esistenti.

Con i metodi `setBackground()` e `setForeground()` possiamo modificare i colori di default secondo quanto indicato nel Capitolo 3 e per quanto riguarda l'eventuale scelta di un carattere per la scritta diverso da quello di default vale quanto detto nel precedente Capitolo 4.

6 Disegno della GUI

Come accennato nel Capitolo 2, Java ha vari strumenti per gestire il collocamento dei componenti nel content pane del Frame o nei pannelli, i layout manager.

Chi conosce il linguaggio Java sa dell'esistenza della classe `FlowLayout`, della classe `GridLayout`, della classe `BoxLayout`, della classe `BorderLayout`, ecc. e della possibilità di combinare variamente questi gestori attraverso l'utilizzo di vari `JPanel` inseriti in `JFrame` per costruire GUI anche molto complesse.

E se ci mettiamo su questa strada siamo costretti a ricercare queste combinazioni in quanto, nel realizzare la GUI, ci accorgiamo che in una certa zona della GUI stessa viene bene usare il `FlowLayout` ma in un'altra conviene usare il `GridLayout` e così via.

Se utilizziamo questi strumenti per costruire una GUI semplice ci costringiamo a complicare inutilmente il percorso per fare una cosa facile.

In questo manualetto propongo di non utilizzare i layout manager ma di costruire noi il layout con le istruzioni di base che vedremo, previste dal linguaggio Java appunto per chi non voglia utilizzare i layout manager.

Ovviamente esiste una differenza tra i due percorsi e tra i risultati cui si perviene attraverso i due percorsi.

La collocazione degli elementi con i layout manager è una collocazione relativa e, per default, è relativa anche la dimensione dei vari componenti: la posizione di un componente è di fianco, sotto o sopra un altro componente e lo spazio che occupa si dimensiona in relazione allo spazio occupato dai componenti circostanti.

Se non usiamo i layout manager la collocazione e il dimensionamento dei vari componenti è assoluta: dobbiamo cioè indicare noi esattamente la dimensione del componente e il luogo dove va collocato.

La più vistosa differenza di risultato è che una GUI costruita con i layout manager sicuramente si adatta a qualsiasi tipo di schermo di computer e, se la ridimensioniamo utilizzando le maniglie laterali per allargarla, allungarla o restringerla il tracciato video della GUI, cioè il suo aspetto, il suo layout, rimarrà sempre quello e i componenti della GUI adatteranno il posizionamento relativo e le dimensioni a quelle del modificato telaio della GUI stessa.

Una GUI costruita con il posizionamento assoluto, senza layout manager, se la ridimensioniamo utilizzando le maniglie laterali per allargarla conserverà la stessa posizione e dimensione dei componenti che aveva prima dell'allargamento e se utilizziamo le maniglie laterali per restringerla faremo scomparire i componenti presenti nelle aree tagliate.

Fatta questa premessa, vediamo come gestire il così detto *absolute layout*, cioè come disegnare la GUI senza l'ausilio di alcun layout manager.

La prima cosa da fare è dichiarare questa intenzione nello script, subito dopo la creazione del frame, con l'istruzione

```
<nome_frame>.setLayout(null)
```

Dopo di che disegniamo i vari componenti avvalendoci dei seguenti metodi che ciascun componente possiede e poi li collochiamo utilizzando il metodo `.add` del Frame:

```
setLocation(x, y)
```

per indicare il punto del frame in cui collocare l'angolo in alto a sinistra del componente (x e y sono i pixel di coordinata, essendo 0 e 0 l'origine in alto a sinistra del frame)

```
setSize(<larghezza>, <altezza>)
```

per indicare la dimensione del componente

(<larghezza> e <altezza> indicano larghezza e altezza del componente in pixel)

```
setBounds(x, y, <larghezza>, <altezza>)
```

per indicare, insieme, le due cose di prima.

7 Eventi

Quando si esegue un programma che si interfaccia con l'utente attraverso il terminale il computer segue il percorso previsto dal programma (siamo nella così detta programmazione imperativa), coinvolge l'utente quando previsto per fare ciò che il programma prevede e il programma termina come e quando previsto al suo interno.

Nei programmi dotati di GUI l'utente è più protagonista, può avere la possibilità di introdurre un dato prima dell'altro, può dare il via all'esecuzione del programma dopo aver controllato di avere inserito i dati giusti, può avere a disposizione diverse opzioni di esecuzione di certe parti del programma, può uscire dal programma ancor prima di eseguirlo completamente, ecc.

Per i piccoli programmi che facciamo noi dilettanti, comunque per programmi semplicemente destinati al calcolo o all'esecuzione di compiti molto specifici privi di opzionalità, la differenza è relativa: anzi, molte volte è assolutamente inutile e superfluo ricorrere a interfacce grafiche per l'utente.

Se però pensiamo, per esempio, ad un programma di word processing dobbiamo ammettere che esso non potrebbe esistere senza una GUI e l'alternativa di produrre un file di testo da terminale, anche se ancora possibile sul sistema Linux, fa ormai parte della preistoria del computer.

Quando si utilizza una GUI, oltre che costruirla con gli strumenti che abbiamo visto nei capitoli precedenti, dobbiamo prevedere il meccanismo con il quale captare i segnali che l'utente invia al computer attraverso l'interfaccia grafica, con il tasto Invio oppure con il posizionamento e il click del mouse, e collegare a quei segnali l'azione del computer.

Nel gergo Java i segnali sono detti eventi e il package `java.awt` contiene una nutrita serie di classi dotate di decine di metodi per gestire una vastissima serie di eventi.

Si tratta di un armamentario molto complesso, difficile da apprendere e di uso riservato a professionisti.

In questo manualetto per principianti e dilettanti propongo una grande semplificazione e mostro come possiamo gestire gli eventi semplicemente ricorrendo ad un metodo di cui Groovy ha dotato i componenti `TextField` e `Button` che abbiamo visto nel Capitolo 5:

```
addActionListener{}
```

Nel caso del componente `TextField` questo metodo capta l'evento pressione del tasto INVIO della tastiera.

Nel caso del componente `Button` questo metodo capta l'evento click con tasto sinistro del mouse.

Tra le parentesi graffe scriviamo le istruzioni per dire al computer che cosa deve fare dal momento in cui viene captato l'evento.

Possiamo scrivere queste istruzioni su più righe tra la parentesi graffa di apertura e la parentesi graffa di chiusura.

8 Un programma di utilità

Applico ciò che abbiamo visto alla stesura di un programmino che mostra il colore nascente dalla combinazione dei colori rosso, verde e blu e che può servire per la scelta del colore con cui arricchire le nostre creazioni.

Lo script è il seguente:

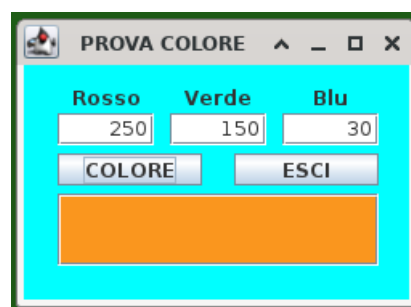
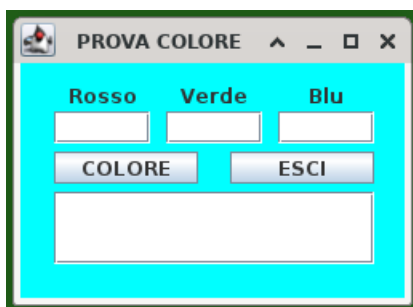
```
import java.awt.*
import javax.swing.*
finestra = new JFrame("PROVA COLORE")
finestra.setBounds(200, 200, 250, 180)
finestra.setLayout(null)
finestra.getContentPane().setBackground(Color.cyan)
scritta_1 = new JLabel("Rosso")
```

```

scritta_2 = new JLabel("Verde")
scritta_3 = new JLabel("Blu")
scritta_1.setBounds(21, 10, 60, 20)
scritta_1.setHorizontalAlignment(JLabel.CENTER)
scritta_2.setBounds(91, 10, 60, 20)
scritta_2.setHorizontalAlignment(JLabel.CENTER)
scritta_3.setBounds(161, 10, 60, 20)
scritta_3.setHorizontalAlignment(JLabel.CENTER)
finestra.add(scritta_1)
finestra.add(scritta_2)
finestra.add(scritta_3)
input_1 = new JTextField()
input_2 = new JTextField()
input_3 = new JTextField()
input_1.setBounds(21, 30, 60, 20)
input_1.setHorizontalAlignment(JTextField.RIGHT)
input_2.setBounds(91, 30, 60, 20)
input_2.setHorizontalAlignment(JTextField.RIGHT)
input_3.setBounds(161, 30, 60, 20)
input_3.setHorizontalAlignment(JTextField.RIGHT)
finestra.add(input_1)
finestra.add(input_2)
finestra.add(input_3)
pulsante_1 = new JButton("COLORE")
pulsante_2 = new JButton("ESCI")
pulsante_1.setBounds(21, 55, 90, 20)
pulsante_2.setBounds(131, 55, 90, 20)
finestra.add(pulsante_1)
finestra.add(pulsante_2)
area_colore = new JTextField()
area_colore.setBounds(21, 80, 200, 45)
finestra.add(area_colore)
pulsante_2.addActionListener
{
    System.exit(0)
}
pulsante_1.addActionListener
{
    r = (input_1.getText()) as int
    g = (input_2.getText()) as int
    b = (input_3.getText()) as int
    area_colore.setBackground(new Color(r, g, b))
}
finestra.setVisible(true)

```

Eseguendo lo script otteniamo



Sulla sinistra abbiamo la finestrella vuota in attesa dei dati per agire e sulla destra abbiamo la finestrella come appare dopo che abbiamo inserito i dati richiesti e premuto il pulsante con

la scritta COLORE, finestra che ora mostra un campione del colore tipo legno noce chiaro che deriva dalla combinazione RGB 250, 150, 30.

Ho ritenuto di arricchire l'applicazione con la possibilità di uscire con un metodo più professionale rispetto a quello di terminare l'applicazione cliccando sulla X in alto a destra della finestra: il metodo `System.exit(0)` di Java.

Questo è un esempio di script per il quale è necessario avere una GUI: con uno script a riga di comando, infatti, non riusciremmo mai a far vedere un colore.

9 Menu

L'invio di segnali al computer per creare eventi può avvenire non solo utilizzando componenti della GUI, quali la finestra di testo o il pulsante, ma anche utilizzando un menu.

Gli applicativi che utilizziamo per scrivere una lettera, per creare un file MIDI con un sequencer, per disegnare la pianta di un appartamento e quant'altro, in genere hanno addirittura una barra di menu e una barra di strumenti che servono per fare le stesse cose: la prima funziona con una serie di voci svolgibili a tendina che ci mostra le possibili scelte, la seconda ci presenta le stesse possibili scelte attraverso una serie di pulsanti con icone descrittive. La scelta di ciò che ci aggrada possiamo farla cliccando su una voce di menu oppure cliccando su un pulsante.

Senza pretendere di creare anche noi GUI così sovrabbondanti di strumenti di scelta, possono capitarci comunque situazioni in cui l'uso di un menu sarebbe la scelta migliore rispetto a quanto abbiamo visto finora.

Pertanto è bene che vediamo come si costruisce un menu utilizzando le classi contenute nel package `javax.swing`.

9.1 JMenuBar

Tra l'intestazione di un `Frame` e la zona visibile del `ContentPane` esiste una zona invisibile fino a quando non ha un contenuto visibile destinata ad ospitare la barra del menu.

Per utilizzare questa zona dobbiamo inserirvi questa barra e lo facciamo creandola con il costruttore

```
new JMenuBar()
```

e definendola meglio con il metodo

```
setBounds(0, 0, <larghezza_frame>, 20)
```

dove, con le coordinate 0 e 0, allochiamo la barra all'origine del frame e con gli altri due parametri assegniamo alla barra la stessa larghezza del frame (ripetendola in numero intero) e un'altezza ragionevole (20 pixel).

Perché questo funzioni ricordiamo la scelta di non utilizzare layout manager con l'istruzione

```
<nome_frame>.setLayout(null)
```

A questo punto inseriamo la barra nel frame con

```
<nome_frame>.add(<nome_barra_menu>)
```

9.2 JMenu

È l'oggetto che contiene una voce del menu, quella che compare nella barra del menu e cliccando sulla quale si apre la tendina che espone le varie scelte possibili.

Il costruttore è

```
new JMenu(<stringa_voce>)
```

dove `<stringa_voce>` è, tra apici, la voce di menu che deve apparire nella barra del menu.

Inseriamo questo oggetto e tutti gli eventuali altri dello stesso tipo, uno per uno, nella barra del menu con

```
<nome_barra_menu>.add(<nome_menu>)
```

9.3 JMenuItem

E' l'oggetto che contiene la scelta proposta da una voce di menu.

Lo costruiamo con

```
new JMenuItem(<stringa_scelta>)
```

dove <stringa_scelta> è, tra apici, la sotto-voce di menu che vediamo nella tendina che si apre cliccando sulla voce di menu.

Inseriamo questo oggetto e tutti gli eventuali altri dello stesso tipo, uno per uno, nel menu con

```
<nome_menu>.add(<nome_menu_item>)
```

A questo punto associamo a ciascun menu_item l'azione che deve svolgere il computer in corrispondenza della scelta di quel menu_item con

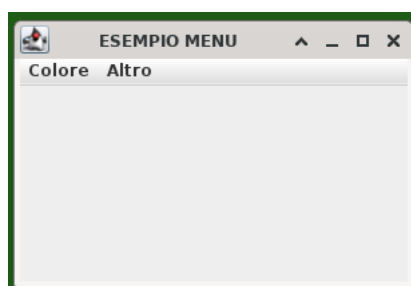
```
<nome_menu_item>.addActionListener{}
```

Un esempio per capirci.

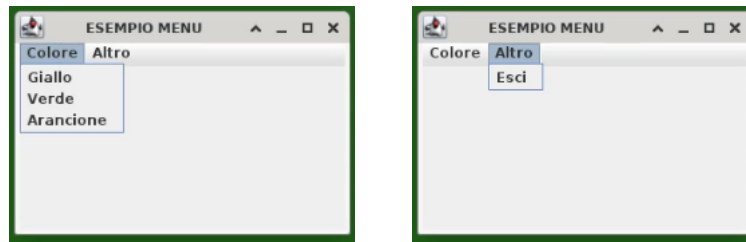
Questo script

```
import javax.swing.*
import java.awt.*
mioFrame = new JFrame("ESEMPIO MENU")
mioFrame.setBounds(200,200,300,200)
mioFrame.setLayout(null)
miaMenuBar = new JMenuBar()
miaMenuBar.setBounds(0,0,300,20)
mioFrame.add(miaMenuBar)
mioMenu_1 = new JMenu("Colore")
miaMenuBar.add(mioMenu_1)
menuVoce_1 = new JMenuItem("Giallo")
mioMenu_1.add(menuVoce_1)
menuVoce_2 = new JMenuItem("Verde")
mioMenu_1.add(menuVoce_2)
menuVoce_3 = new JMenuItem("Arancione")
mioMenu_1.add(menuVoce_3)
mioMenu_2 = new JMenu("Altro")
miaMenuBar.add(mioMenu_2)
menuVoce_4 = new JMenuItem("Esci")
mioMenu_2.add(menuVoce_4)
menuVoce_1.addActionListener{mioFrame.getContentPane().setBackground(Color.yellow)}
menuVoce_2.addActionListener{mioFrame.getContentPane().setBackground(Color.green)}
menuVoce_3.addActionListener{mioFrame.getContentPane().setBackground(Color.orange)}
menuVoce_4.addActionListener{System.exit(0)}
mioFrame.setVisible(true)
```

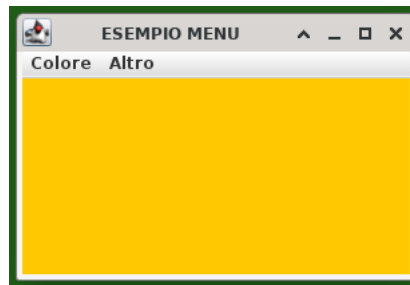
produce questa GUI



Il contenuto dei menu è il seguente



Cliccando, per esempio, su Arancione otteniamo questo



10 Qualche altro esempio

Potremmo essere interessati a calcolare il punto di rugiada e la temperatura percepita in presenza di una certa temperatura dell'aria e di un certo livello di umidità relativa.

La psicrometria ci fornisce le formule per fare questi calcoli.

Per determinare il punto di rugiada abbiamo la formula di Magnus-Tetens

$$punto\ di\ rugiada = \frac{237,7a}{17,27-a}$$

dove, con t che esprime la temperatura e u che esprime l'umidità relativa unitaria,

$$a = \frac{17,27t}{237,7+t} + \log(u)$$

Per determinare la temperatura percepita possiamo calcolare l'indice Humidex con la formula

$$h = t + \frac{5}{9}(e - 10)$$

dove, sempre con t che esprime la temperatura e u che esprime l'umidità relativa unitaria,

$$e = 6,11 * 10^{\frac{7,5t}{237,7+t}} * u$$

Con questo script lo possiamo fare in maniera molto bella utilizzando le formule che abbiamo visto

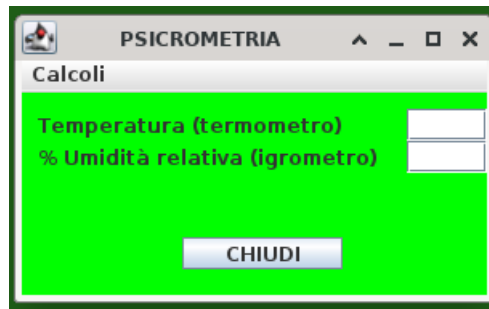
```
import java.awt.*
import javax.swing.*
def double pr(double t, double u)
{
  def a = (17.27*t)/(237.7+t)+Math.log(u/100)
  return (237.7*a)/(17.27-a)
}
def double tp(double t, double u)
{
  def e = 6.11*(Math.pow(10.0,((7.5*t)/(237.7+t))))*u/100
  return t+5.0/9*(e-10)
}
finestra = new JFrame("PSICROMETRIA")
finestra.setBounds(400, 200, 300, 180)
finestra.setLayout(null)
```

```

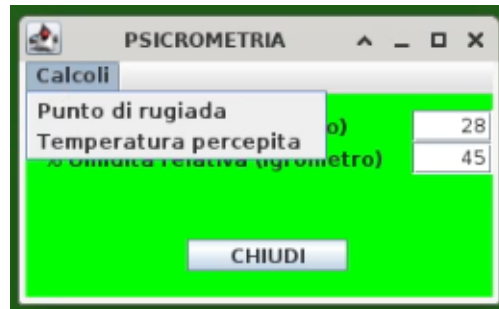
finestra.getContentPane().setBackground(Color.green)
menu_bar = new JMenuBar()
menu_bar.setBounds(0, 0, 300, 20)
finestra.add(menu_bar)
menu = new JMenu("Calcoli")
menu_bar.add(menu)
m_i_1 = new JMenuItem("Punto di rugiada")
menu.add(m_i_1)
m_i_2 = new JMenuItem("Temperatura percepita")
menu.add(m_i_2)
frase_1 = new JLabel("Temperatura (termometro)")
frase_1.setBounds(10, 30, 250, 20)
finestra.add(frase_1)
input_1 = new JTextField()
input_1.setBounds(240, 30, 50, 20)
input_1.setHorizontalAlignment(JTextField.RIGHT)
finestra.add(input_1)
frase_2 = new JLabel("% Umidità relativa (igrometro)")
frase_2.setBounds(10, 50, 250, 20)
finestra.add(frase_2)
input_2 = new JTextField()
input_2.setBounds(240, 50, 50, 20)
input_2.setHorizontalAlignment(JTextField.RIGHT)
finestra.add(input_2)
risultato = new JLabel()
risultato.setBounds(10, 80, 280, 20)
risultato.setHorizontalAlignment(JLabel.CENTER)
finestra.add(risultato)
pulsante = new JButton("CHIUDI")
pulsante.setBounds(100, 110, 100, 20)
finestra.add(pulsante)
m_i_1.addActionListener
{
    def t = input_1.getText()
    def tt = t.toDouble()
    def u = input_2.getText()
    def uu = u.toDouble()
    def r = pr(tt, uu)
    def rr = (r*10).toInteger()
    def rrr = rr.toFloat()/10
    risultato.setText("Punto di rugiada: " + rrr.toString() + " gradi")
}
m_i_2.addActionListener
{
    def t = input_1.getText()
    def tt = t.toDouble()
    def u = input_2.getText()
    def uu = u.toDouble()
    def r = tp(tt, uu)
    def rr = (r*10).toInteger()
    def rrr = rr.toFloat()/10
    risultato.setText("Temperatura percepita: " + rrr.toString() + " gradi")
}
pulsante.addActionListener{System.exit(0)}
finestra.setVisible(true)

```

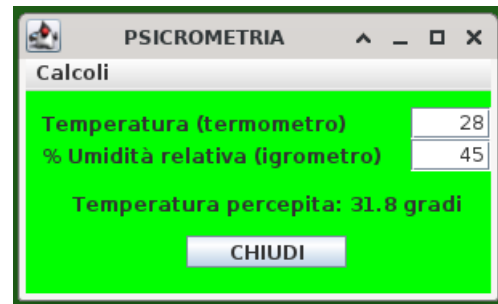
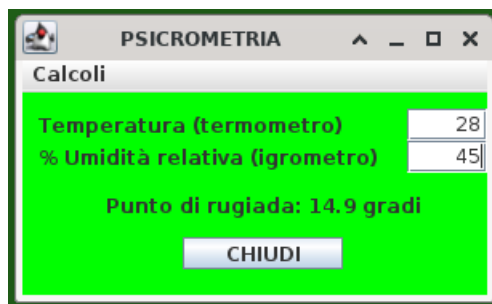
Lo script produce questa finestra



Una volta inseriti i dati della temperatura e dell'umidità relativa in percentuale, da menu scegliamo il calcolo da effettuare



Sulla sinistra abbiamo la finestra dopo aver scelto di calcolare il punto di rugiada e sulla destra abbiamo la finestra dopo aver scelto di calcolare la temperatura percepita



Un altro script per un'azione molto banale ma utile per esemplificare l'uso di colori e font.

```
import java.awt.*
import javax.swing.*
colore_1 = new Color(200, 200, 150)
colore_2 = new Color(100, 100, 150)
colore_3 = new Color(255, 30, 80)
colore_4 = new Color(160, 190, 80)
colore_5 = new Color(30, 30, 255)
font_1 = new Font("eufm10", Font.PLAIN, 22)
font_2 = new Font("MuseJazz text", Font.BOLD, 40)
finestra = new JFrame("SALUTO")
finestra.setLayout(null)
finestra.setBounds(300, 200, 300, 190)
finestra.getContentPane().setBackground(colore_1)
scritta = new JLabel("Come ti chiami?")
scritta.setForeground(colore_2)
scritta.setFont(font_1)
scritta.setBounds(10, 20, 180, 25)
finestra.add(scritta)
input = new JTextField()
```

```

input.setBounds(190, 15, 100, 25)
finestra.add(input)
saluto = new JLabel()
saluto.setBounds(10, 60, 280, 40)
saluto.setForeground(colore_3)
saluto.setFont(font_2)
saluto.setHorizontalAlignment(JLabel.CENTER)
finestra.add(saluto)
input.addActionListener
{
nome = input.getText()
saluto.setText("Ciao, $nome")
}
pulsante = new JButton("ESCI")
pulsante.setBounds(110, 110, 80, 30)
pulsante.setBackground(colore_4)
pulsante.setForeground(colore_5)
finestra.add(pulsante)
pulsante.addActionListener{System.exit(0)}
finestra.setVisible(true)

```

Sulla sinistra la finestra appena lanciato lo script, sulla destra la finestra dopo avere inserito il nome della persona da salutare e premuto INVIO.



11 Contesto grafico Java in Groovy per il disegno

Il package Java in cui troviamo gli strumenti per il disegno è **java.awt** (Java Abstract Widget Toolkit).

Questi strumenti sono metodi della classe **Graphics** e della sua sottoclasse **Graphics2D**: gli strumenti contenuti nella prima sono semplici e sufficienti per componenti di base, mentre gli strumenti contenuti nella seconda comprendono quelli della prima e altri più avanzati.

Per la realizzazione del disegno abbiamo a disposizione i metodi `paint()` della libreria `java.awt` e `paintComponent()` della libreria `javax.swing`, ai quali il sistema passa automaticamente un oggetto, chiamato `g`, della classe `Graphics` e il disegno si realizza chiamando i metodi della classe `Graphics` sull'oggetto `g`.

Se intendiamo utilizzare i metodi più evoluti della classe `Graphics2D` possiamo creare un altro oggetto, che chiamiamo `g2`, di classe `Graphics2D` attraverso conversione esplicita (casting) dell'oggetto `g`, con l'istruzione

```
Graphics2D g2 = (Graphics2D)g;
```

e realizzare il disegno chiamando i metodi della classe `Graphics2D` sull'oggetto `g2`.

Abbiamo poi il problema di dove collocare il nostro disegno, che si colloca creando una classe entro la quale richiamare i metodi `paint()` o `paintComponent()`.

Le classi disponibili, estendendo le quali creare questa classe, sono la classe **Canvas** di `java.awt` o la classe **JPanel** di `javax.swing`.

Per mostrare, infine, il nostro disegno dobbiamo istanziare l'oggetto della classe Canvas o JPanel in un JFrame.

Il linguaggio Groovy interviene solo in questo momento e tutto quanto abbiamo fatto prima è in puro linguaggio Java (con la sola licenza di non usare i punti e virgola a fine riga).

Pertanto le semplificazioni di Groovy in questo campo sono molto relative.

12 Strumenti per disegnare

Richiamo qui i principali metodi che ci mette a disposizione la classe Graphics2D per disegnare, rimandando alla documentazione Java per approfondimenti.

Metodi di disegno comuni

`drawLine(x1, y1, x2, y2)`

disegna una linea retta tra i punti di coordinate (x1,y1) e (x2,y2)

`drawRect(x, y, larghezza, altezza)`

disegna il contorno di un rettangolo con le dimensioni indicate e l'angolo in alto a sinistra nel punto (x,y)

se larghezza e altezza sono uguali si disegna un quadrato

`fillRect(x, y, larghezza, altezza)`

disegna un rettangolo pieno

`drawOval(x, y, larghezza, altezza)`

disegna il contorno di un'ellisse iscritta in un rettangolo delle dimensioni indicate avente l'angolo in alto a sinistra nel punto (x,y)

se larghezza e altezza sono uguali si disegna un cerchio

`fillOval(x, y, larghezza, altezza)`

disegna un'ellisse piena

`drawString(stringa, x, y)`

disegna una stringa di testo a partire dal punto (x,y)

Metodi di impostazione e trasformazione

`setColor(Color.<nome>)`

imposta il colore corrente per il disegno.

<nome> può essere black, white, red, green, blue, cyan, darkGray, gray, lightGray, magenta, orange, pink, yellow

`setColor(new Color(R,G,B))`

imposta il colore corrente per il disegno secondo la codifica RGB

`setFont(new Font(<nome>, <stile>, <grandezza>))`

imposta il font da utilizzare per disegnare testo

<nome> è una stringa che identifica il font

<stile> può essere Font.PLAIN, Font.BOLD, Font.ITALIC

<grandezza> è un intero che rappresenta la dimensione del font

`setStroke(new BasicStroke(<intero>))`

imposta lo spessore della linea da disegnare

`translate(x, y)`

sposta l'origine del sistema di coordinate

13 Realizzare il disegno

Da quanto detto finora abbiamo capito che il linguaggio Java ci offre diverse alternative per scrivere programmi di disegno e dobbiamo scegliere quale utilizzare.

Personalmente ritengo che i migliori risultati siano ottenibili con una certa facilità orientandoci sull'utilizzo della libreria grafica Graphics2D di java.awt su pannello e frame javax.swing. Questa scelta comporta che il nostro programma di grafica abbia la seguente struttura

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

class <nome> extends JPanel
{
    void paintComponent(Graphics g)
    {
        Graphics2D g2 = (Graphics2D)g
        <istruzioni per il disegno>
    }
}

JFrame frame = new JFrame("<titolo>")
frame.setBounds(<int>, <int>, <int>, <int>)
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)
frame.add(new <nome>())
frame.setVisible(true)
```

Spaziature e rientri non sono richiesti e li ho usati solo per evidenziare meglio la struttura del programma.

Importiamo innanzi tutto i package java che ci servono (java.awt per la libreria Graphics2D e javax.swing per le classi JPanel e JFrame e per il metodo paintComponent()).

Poi dichiariamo una nuova classe chiamata <nome> che eredita tutte le proprietà e i metodi della classe JPanel.

Al suo interno chiamiamo il metodo che viene utilizzato per disegnare sul componente grafico JPanel. La parola chiave void significa che non restituisce alcun valore, e paintComponent(Graphics g) definisce il metodo stesso, che accetta un oggetto Graphics chiamato g come argomento, su cui verranno eseguiti tutti i comandi di disegno.

Facciamo il casting in modo da sostituire all'oggetto Graphics chiamato g un oggetto Graphics2D chiamato g2.

Finalmente scriviamo le istruzioni per il disegno, utilizzando i metodi dell'oggetto g2, di cui abbiamo visto i principali nel precedente Capitolo, con la sintassi g2.<metodo>

Fin qui è tutto linguaggio Java.

Con il linguaggio Groovy proseguiamo creando un JFrame con un <titolo>, sistemandolo e dimensionandolo in modo che contenga tutto ciò che abbiamo disegnato e al suo interno istanziamo la classe del disegno e rendiamo visibile il Frame.

14 Esempio di grafica creativa

Nella pagina successiva si trova il programma con il quale è stato disegnata questa schermata



```

import java.awt.*
import javax.swing.*
class prove extends JPanel
{
    void paintComponent(Graphics g)
    {
        Graphics2D g2 = (Graphics2D)g
        g2.setStroke(new BasicStroke(3))
        g2.drawLine(20,20,50,50)
        g2.setStroke(new BasicStroke(5))
        g2.setColor(new Color(0,250,0))
        g2.drawOval(70,20,70,70)
        g2.setColor(Color.red)
        g2.fillOval(170,20,70,70)
        g2.setColor(Color.yellow)
        g2.fillRect(270,30,110,50)
        g2.setStroke(new BasicStroke(1))
        g2.setColor(Color.blue)
        g2.drawLine(20,120,380,120)
        g2.setFont(new Font("Courier", Font.PLAIN, 28))
        g2.drawString("Disegni di esempio", 50, 150)
        g2.drawLine(20,160,380,160)
    }
}
JFrame frame = new JFrame("ESEMPI")
frame.setBounds(200, 200, 400, 200)
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)
frame.add(new prove())
frame.setVisible(true)

```

Qui è riepilogato un po' tutto ciò che abbiamo visto.

15 Esempio di grafica illustrativa

Utilizzando gli strumenti di disegno che abbiamo visto sarebbe possibile creare grafici di statistica descrittiva (istogrammi, torte, ecc.) ma, soprattutto se siamo in presenza di dataset di una certa consistenza possiamo immaginare a che tipo di lavoro andremmo incontro.

Purtroppo Java non ha librerie che si prestano a semplificare questo lavoro e per esigenze di questo tipo dobbiamo ricorrere a librerie esterne il cui utilizzo non è alla portata di dilettante.

Una cosa che possiamo tuttavia affrontare con relativamente meno fatica è la creazione di grafici di funzione di questo tipo



Questo è il listato del programma che ha creato il grafico

```
import java.awt.*
import javax.swing.*
class Grafico extends JPanel
{
    // La funzione da plottare: f(x) = sin(x)
    private double f(double x)
    {
        return Math.sin(x)
    }
    void paintComponent(Graphics g)
    {
        Graphics2D g2 = (Graphics2D) g
        // Disegna gli assi con origine al centro
        int width = getWidth()
        int height = getHeight()
        int originX = width/2
        int originY = height/2
        g2.setColor(Color.BLACK)
        g2.drawLine(0,originY,width,originY) // Asse X
        g2.drawLine(originX,0,originX,height) // Asse Y
        g2.drawString("0",originX-10,originY+15)
        // Scala: un'unità (es. 1) = 50 pixel
        double scale=50.0
        // Calcola e disegna i punti in rosso
        g2.setColor(Color.RED)
        for (int i=-originX;i<width-originX;i++)
        {
            double x1=(double)i/scale
            double y1=f(x1)
            double x2=(double)(i + 1)/scale
            double y2=f(x2)
            // Converti le coordinate matematiche in coordinate pixel di Swing
            int screenX1=originX+i
            int screenY1=originY-(int)(y1*scale) // Sottrae perché l'asse Y di Swing è invertito
            int screenX2=originX+i+1
            int screenY2=originY-(int)(y2*scale)
            g2.drawLine(screenX1,screenY1,screenX2,screenY2)
        }
    }
}
JFrame frame = new JFrame("Grafico di funzione")
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)
frame.add(new Grafico())
frame.setBounds(200,200,600,300)
frame.setVisible(true)
```

Ovviamente se nel costruire la funzione da plottare inseriamo la formula di un'altra funzione matematica otteniamo il grafico di quest'altra.

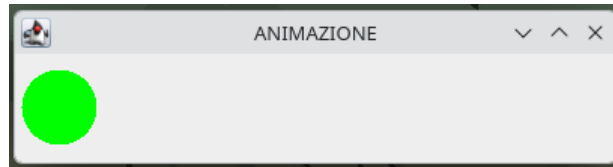
Come si vede, la scrittura del programma richiede una conoscenza più che elementare del linguaggio di programmazione Java. Purtroppo in un campo in cui a nulla servono le semplificazioni del linguaggio Groovy.

16 Animazioni

Dal disegno statico che abbiamo visto nel Capitolo 14, con qualche complicazione aggiuntiva possiamo passare al disegno di cose che si muovono.

Anche in questo caso la vera e propria animazione viene creata con codice Java e il linguaggio Groovy è utile solo per semplificare la fase di creazione della finestra in cui mostrare l'animazione.

Se vogliamo che in questa finestra



la pallina verde rotoli da sinistra a destra e poi scompaia scriviamo questo programma

```
import java.awt.*
import javax.swing.*
class Animazione extends JPanel
{
    // posizione iniziale
    int x = 0
    int y = 5
    // velocità orizzontale
    int dx = 2
    Animazione()
    {
        // Timer Swing: chiama muovi() e repaint() ogni 15 millisecondi
        new Timer(15,
            {
                muovi()
                repaint()
            }
        ).start()
    }
    void muovi() //modifica le coordinate di posizione dell'oggetto da disegnare
    {
        x=x+dx
    }
    void paintComponent(Graphics g)
    {
        super.paintComponent(g) //ridisegna ad ogni repaint() con la pallina nella nuova posizione
        Graphics2D g2 = (Graphics2D)g
        g2.setColor(Color.green)
        g2.fillOval(x,y,50,50)
    }
}
JFrame frame = new JFrame("ANIMAZIONE")
frame.setBounds(200,200,400,100)
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)
frame.add(new Animazione())
frame.setVisible(true)
```

L'animazione viene generata dal richiamo a cadenza regolare (ottenuta dalla classe Timer di javax.swing) della funzione muovi(), che creiamo noi per modificare le coordinate dell'oggetto da muovere, e repaint(), che sta nella classe component del package java.awt, e che serve per fare ridisegnare un componente.