GUI con Kotlin (autore: Vittorio Albertoni)

Premessa

Possiamo ritenere il linguaggio di programmazione Kotlin 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[1].

Peraltro, dove non arriva Kotlin con le sue semplificazioni, se serve arriva Java: infatti tutte le librerie Java sono a disposizione di Kotlin.

Tra queste, le librerie grafiche di Java: gli autori di Kotlin, infatti, non hanno sin qui ritenuto di dotare questo linguaggio di packages di grafica.

Ovviamente, se utilizziamo librerie Java, per questo utilizzo le semplificazioni di Kotlin non esistono più e dobbiamo soggiacere alla più complicata sintassi richiesta dalle librerie Java, integrandola con la sintassi del linguaggio Kotlin.

Nel campo della grafica, che qui ci interessa, se l'obiettivo è quello di creare grafica del tipo di quella che vivacizzava le applet Java di buona memoria, quella grafica che utilizza le librerie Java Graphics e Graphics2D, non serve Kotlin: dovremmo scrivere un sacco di cose in linguaggio Java e utilizzare Kotlin solo per farle vedere.

Diversa la situazione se l'obiettivo è quello di realizzare interfacce grafiche per i nostri programmi, le così dette GUI (Graphical User Interface).

In questo caso, infatti, può esistere una ponderosa parte del programma (per elaborazioni varie di natura non grafica) convenientemente scrivibile nel relativamente semplice linguaggio Kotlin 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 Kotlin e linguaggio Java per queste cose non è poi così difficile.

E' ciò che cerco di dimostrare in questo manualetto.

[1] Sul mio blog, all'indirizzo *www.vittal.it*, è disponibile un manualetto sulle basi di questo linguaggio, in formato PDF, intitolato «kotlin» e allegato all'articolo «Kotlin: Java facilitato» pubblicato nel luglio 2019.

Indice

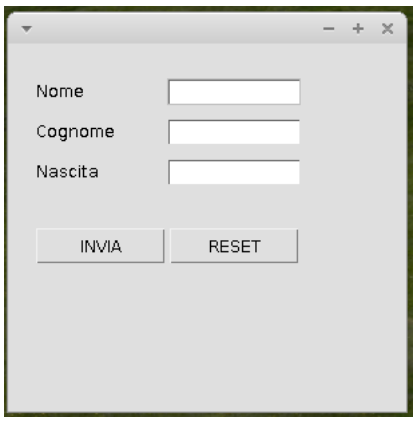
1 Contesto grafico in Java

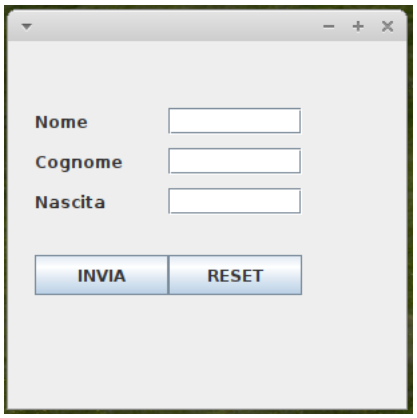
Il package Java originariamente dedicato agli strumenti per creare interfacce grafiche utente è **java.awt** (Java Abstract Widget Toolkit).

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

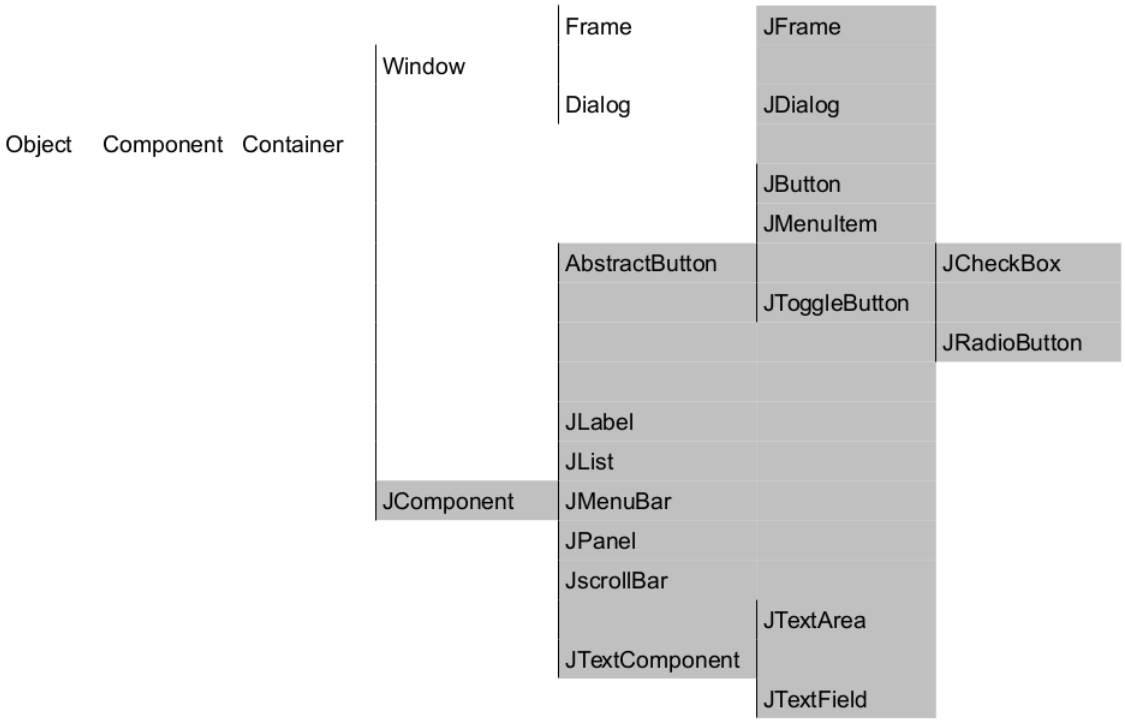
Per un sistema che si fregia del motto «write once run anywhere» non è il massimo e si è rimediato con l'apprestamento di un altro package che integra java.awt, che si chiama **javax.swing** e che risolve questi problemi.

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.

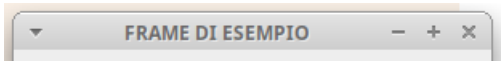
Il pulsante si chiama Button in java.awt e si chiama JButton in javax.swing.

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.

Il metodo costruttore del nostro telaio è:
`JFrame()`
e crea una finestra senza titolo.
Per dare un titolo alla finestra possiamo inserirlo tra le parentesi tonde come stringa (tra doppi apici).

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 Kotlin utilizziamo il metodo
`setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)`.

Con questo script

```
import javax.swing.*
fun main()
{
var mioFrame = 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 (`JFrame("FRAME DI ESEMPIO")`) per assegnare l'oggetto ad una variabile Kotlin (`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.

Innanzi tutto 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

`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 `Color(0,0,0)` indica nessun colore, cioè il colore nero e `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.
Innanzi tutto, 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)`

Dal momento che lo script si compila senza errori, eseguendo lo script ci aspettiamo di vedere la nostra finestra colorata di verde.
Ciò, tuttavia, non avviene e la finestra 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(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

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

dove
<stringa_nome> è il nome del font tra doppi 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 Kotlin 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 finestrella che serve per immettere una sola riga di testo o un solo numero.

Il metodo costruttore è
`JTextField()`

Per default viene costruito 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 finestrella e lo ritorna come stringa.

5.2 JLabel

E' una finestrella per esporre testo o numeri.

Il metodo costruttore è
`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.

Il metodo costruttore è

`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 posizionamento relativo e 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 sono dotati i componenti JTextField e JButton che abbiamo visto nel Capitolo 5:

`addActionListener{}`

Nel caso del componente JTextField questo metodo capta l'evento pressione del tasto INVIO della testiera.

Nel caso del componente JButton 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 a patto che la prima parentesi graffa si trovi sulla stessa riga dove richiamiamo il metodo.

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.*
import kotlin.system.*
fun main()
{
    var finestra = JFrame("PROVA COLORE")
    finestra.setBounds(200, 200, 220, 160)
    finestra.setLayout(null)
    finestra.getContentPane().setBackground(Color.cyan)
    var scritta_1 = JLabel("Rosso")
    var scritta_2 = JLabel("Verde")
    var scritta_3 = JLabel("Blu")
    scritta_1.setBounds(10, 10, 60, 20)
    scritta_1.setHorizontalAlignment(JLabel.CENTER)
    scritta_2.setBounds(80, 10, 60, 20)
    scritta_2.setHorizontalAlignment(JLabel.CENTER)
    scritta_3.setBounds(150, 10, 60, 20)
    scritta_3.setHorizontalAlignment(JLabel.CENTER)
    finestra.add(scritta_1)
    finestra.add(scritta_2)
    finestra.add(scritta_3)
    var input_1 = JTextField()
    var input_2 = JTextField()
    var input_3 = JTextField()
    input_1.setBounds(10, 30, 60, 20)
    input_1.setHorizontalAlignment(JTextField.RIGHT)
    input_2.setBounds(80, 30, 60, 20)
    input_2.setHorizontalAlignment(JTextField.RIGHT)
    input_3.setBounds(150, 30, 60, 20)
    input_3.setHorizontalAlignment(JTextField.RIGHT)
    finestra.add(input_1)
    finestra.add(input_2)
    finestra.add(input_3)
    var pulsante_1 = JButton("COLORE")
    var pulsante_2 = JButton("ESCI")
    pulsante_1.setBounds(10, 55, 90, 20)
    pulsante_2.setBounds(120, 55, 90, 20)
    finestra.add(pulsante_1)
    finestra.add(pulsante_2)
    var area_colore = JTextField()
    area_colore.setBounds(10, 80, 200, 45)
    finestra.add(area_colore)
    pulsante_2.addActionListener{exitProcess(0)}
    pulsante_1.addActionListener{
        var r = (input_1.getText()).toInt()
        var g = (input_2.getText()).toInt()
        var b = (input_3.getText()).toInt()
    area_colore.setBackground(Color(r, g, b))
    }
    finestra.setVisible(true)
}
```

Lo salviamo con un nome che ne richiami la funzione, ad esempio `prova_colore.kt`.

Lo compiliamo con

`kotlinc prova_colore.kt`

e lanciamo l'eseguibile in byte-code, che si chiama Prova_coloreKt, con

`kotlin Prova_coloreKt`

ottenendo

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, finestrella 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 finestrella.

Per fare questo ho dovuto utilizzare il metodo `exitProcess(0)` del package `system` di Kotlin, importato all'inizio dello script.

Se compiliamo il nostro script con

`kotlinc prova_colore.kt -include-runtime -d prova_colore.jar`

otteniamo il file `prova_colore.jar` che possiamo eseguire sulla Java Virtual Machine di Java anche senza avere installato sul computer Kotlin.

Questo è un esempio di programma per il quale è necessario avere una GUI: con un programma 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 finestrella 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 inserivi questa barra e lo facciamo creandola con il costruttore

`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

E' 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
`JMenu(`<stringa_voce>`)`
dove <stringa_voce> è, tra doppi 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.

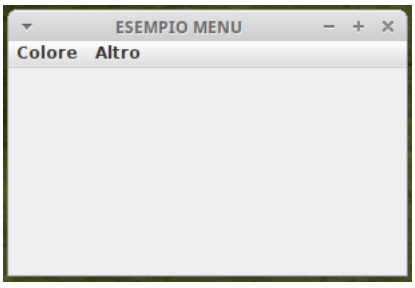
Il costruttore è
`JMenuItem(`<stringa_scelta>`)`
dove <stringa_scelta> è, tra doppi apici, la sotto-voce di menu che vediamo nella tendina che si apre cliccando sulla voce di menu.

Inseriamo questo oggetto e tutti gli eventuali altri dello stesso tipo, uno per uno, nel menu con
<nome_menu>`.add(`<nome_menu_item>`)`

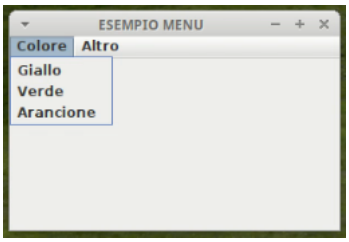
A questo punto associamo a ciascun menu_item l'azione che deve svolgere il computer in corrispondenza della scelta di quel menu_item con
<nome_menu_item>`.addActionListener{}`

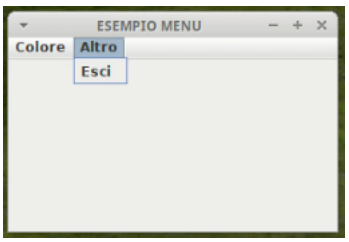
Un esempio per capirci.

Lo script che troviamo nella pagina che segue produce questa GUI

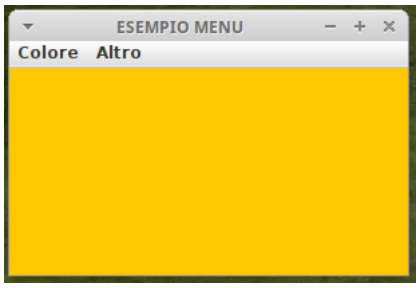


Il contenuto dei menu è il seguente





Cliccando, per esempio, su Arancione otteniamo questo



Questo è lo script

```
import javax.swing.*
import java.awt.*
import kotlin.system.*
fun main()
{
var mioFrame = JFrame("ESEMPIO MENU")
mioFrame.setBounds(200,200,300,200)
mioFrame.setLayout(null)
var miaMenuBar = JMenuBar()
miaMenuBar.setBounds(0,0,300,20)
mioFrame.add(miaMenuBar)
var mioMenu_1 = JMenu("Colore")
miaMenuBar.add(mioMenu_1)
var menuVoce_1 = JMenuItem("Giallo")
mioMenu_1.add(menuVoce_1)
var menuVoce_2 = JMenuItem("Verde")
mioMenu_1.add(menuVoce_2)
var menuVoce_3 = JMenuItem("Arancione")
mioMenu_1.add(menuVoce_3)
var mioMenu_2 = JMenu("Altro")
miaMenuBar.add(mioMenu_2)
var menuVoce_4 = 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{exitProcess(0)}
mioFrame.setVisible(true)
}
```

10 Qualche altro esempio

Lo script che riporto nella pagina che segue può essere utile per 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

$$puntorugiada = \frac{237{,}7a}{17{,}27 - a}$$

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,

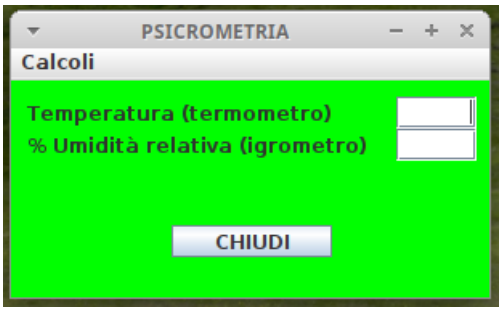
$$e = 6,11 * 10^{\frac{7,5t}{237,7+t}} * u$$

Lo script è il seguente

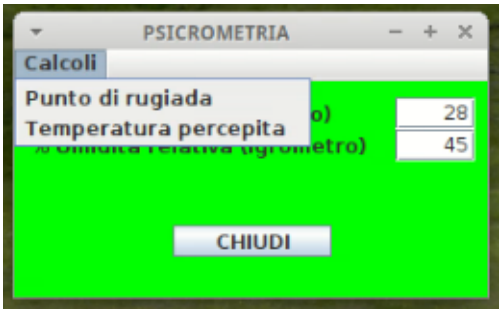
```
import java.awt.*
import javax.swing.*
import kotlin.system.*

fun pr(t:Double, u:Double):  Double
{
    var a = (17.27*t)/(237.7+t)+Math.log(u/100)
    return (237.7*a)/(17.27-a)
}

fun tp(t:Double, u:Double):  Double
{
    var e = 6.11*(Math.pow(10.0,((7.5*t)/(237.7+t))))*u/100
    return t+5.0/9*(e-10)
}

fun main()
{
    var finestra = JFrame("PSICROMETRIA")
    finestra.setBounds(400, 200, 300, 180)
    finestra.setLayout(null)
    finestra.getContentPane().setBackground(Color.green)
    var menu_bar = JMenuBar()
    menu_bar.setBounds(0, 0, 300, 20)
    finestra.add(menu_bar)
    var menu = JMenu("Calcoli")
    menu_bar.add(menu)
    var m_i_1 = JMenuItem("Punto di rugiada")
    menu.add(m_i_1)
    var m_i_2 = JMenuItem("Temperatura percepita")
    menu.add(m_i_2)
    var frase_1 = JLabel("Temperatura (termometro)")
    frase_1.setBounds(10, 30, 250, 20)
    finestra.add(frase_1)
    var input_1 = JTextField()
    input_1.setBounds(240, 30, 50, 20)
    input_1.setHorizontalAlignment(JTextField.RIGHT)
    finestra.add(input_1)
    var frase_2 = JLabel("% Umiditâ relativa (igrometro)")
    frase_2.setBounds(10, 50, 250, 20)
    finestra.add(frase_2)
    var input_2 = JTextField()
    input_2.setBounds(240, 50, 50, 20)
    input_2.setHorizontalAlignment(JTextField.RIGHT)
    finestra.add(input_2)
    var risultato = JLabel()
    risultato.setBounds(10, 80, 280, 20)
    risultato.setHorizontalAlignment(JLabel.CENTER)
    finestra.add(risultato)
    var pulsante = JButton("CHIUDI")
    pulsante.setBounds(100, 110, 100, 20)
    finestra.add(pulsante)
```

```
    m_i_1.addActionListener{
        var t = input_1.getText()
        var tt = t.toDouble()
        var u = input_2.getText()
        var uu = u.toDouble()
        var r = pr(tt, uu)
        var rr = (r*10).toInt()
        var rrr = rr.toFloat()/10
        risultato.setText("Punto di rugiada:  " + rrr.toString() + " gradi")
    }
    m_i_2.addActionListener{
        var t = input_1.getText()
        var tt = t.toDouble()
        var u = input_2.getText()
        var uu = u.toDouble()
        var r = tp(tt, uu)
        var rr = (r*10).toInt()
        var rrr = rr.toFloat()/10
        risultato.setText("Temperatura percepita:  " + rrr.toString() + " gradi")
    }
    pulsante.addActionListener{exitProcess(0)}
    finestra.setVisible(true)
}
```

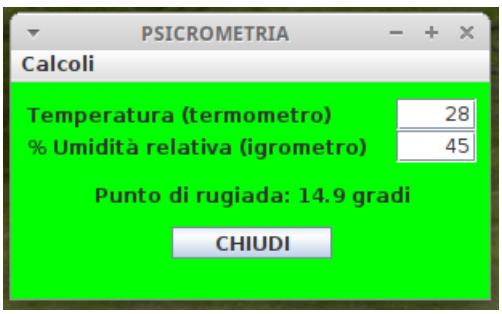
Lo script produce questa finestra

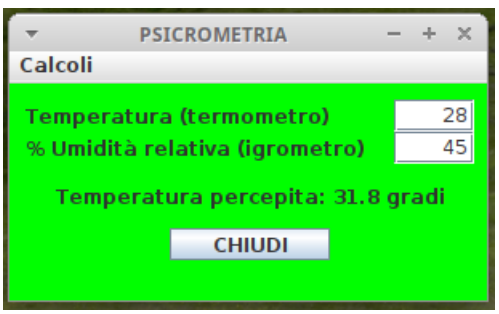


Una volta inseriti i dati della temperatura e dell'umidità relativa in percentuale, da menu scegliamo il calcolo da effettuare



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.*
import kotlin.system.*
fun main()
{
    var colore_1 = Color(200, 200, 150)
    var colore_2 = Color(100, 100, 150)
    var colore_3 = Color(255, 30, 80)
    var colore_4 = Color(160, 190, 80)
    var colore_5 = Color(30, 30, 255)
    var font_1 = Font("Chilanka", Font.PLAIN, 22)
    var font_2 = Font("Learning Curve", Font.BOLD, 40)
    var finestra = JFrame("SALUTO")
    finestra.setLayout(null)
    finestra.setBounds(300, 200, 300, 190)
    finestra.getContentPane().setBackground(colore_1)
    var scritta = JLabel("Come ti chiami?")
    scritta.setForeground(colore_2)
    scritta.setFont(font_1)
    scritta.setBounds(10, 20, 180, 25)
    finestra.add(scritta)
    var input = JTextField()
    input.setBounds(190, 15, 100, 25)
    finestra.add(input)
    var saluto = JLabel()
    saluto.setBounds(10, 60, 280, 40)
    saluto.setForeground(colore_3)
    saluto.setFont(font_2)
    saluto.setHorizontalAlignment(JLabel.CENTER)
    finestra.add(saluto)
    input.addActionListener{
        var nome = input.getText()
        saluto.setText("Ciao, " + nome + "!")
    }
    var pulsante = JButton("ESCI")
    pulsante.setBounds(110, 110, 80, 30)
    pulsante.setBackground(colore_4)
    pulsante.setForeground(colore_5)
    finestra.add(pulsante)
    pulsante.addActionListener{exitProcess(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.