Resources

Android è progettato per essere eseguito su un innumerevole insieme di dispositivi. In questo ambito ogni applicazione sviluppata dovrebbe attuare *feature* dinamiche, rendendo adattiva la *user interface* rispetto alle configurazioni dello schermo.

Come già accennato un'applicazione Android dovrebbe essere eseguita su una eterogenità di dispositivi, ognuno di essi con le proprie caratteristiche. Per riuscire nell'intento possono essere adeguati due approcci principali:

- **Soluzione tradizionale**, adottare tutte le molteplici alternative. Tuttavia questo si traduce in un abuso di costrutti condizionali, rendendo il codice di difficile lettura; inoltre occorre ricompilare il progetto ogni qualvolta dovesse presentarsi la necessità di variare il *Layout* oppure qualora aggiunti nuovi *package*
- **Soluzione Android**, separare il codice dalle *risorse*. Rispetto al paradigma descritto si usufruisce dell'approccio dichiarativo XML affinchè possano essere definite le risorse

Prima di addentrarsi all'interno della composizione delle cartelle che caratterizzino un progetto è bene stabilire cosa sia una *risorsa*.

Definizione
Una *risorsa* è tutto ciò che non sia codice.

Pur di rispettare la definizione precedente è necessario che si susseguano tre condizioni:

- Separare gli aspetti visivi dagli aspetti gestionali. In tal senso si prosegue mediante il paradigma di "buon codice", ossia separare ciò che possa essere ritenuto associabile alla *business logic* da ciò che contraddistingue la *user interface*
- Provvedere a garantire un numero elevato di *features* che possano coprire il più ampio insieme di dispositivi
- Ricompilare solamente quando strettamente necessario

Solitamente è preferita la soluzione Android, data soprattutto dalla capacità del framework di rilevare tutte le caratteristiche configurative del dispositivo, associandone la risorse appropiate, e dalla facilità con cui sia possibile aggiungere nuove resources per supportare un'ulteriore categoria di device.

All'interno di un qualsiasi progetto le risorse sono destinate all'interno della directory **res**; si tratta di una gerarchia già predisposta per i molteplici package inseriti.

Resources Definition

Le risorse sono inizializzate mediante il metodo **dichiarativo** espresso dai file **XML**, in cui ognuna di esse possiede un proprio nome e identificativo.

```
<resources>
    <string name="hello">Hello World!</string>
    <string name="labelButton">Insert your name</string>
</resources>
```

Dichiarazione delle risorse mediante file XML

Tali risorse possono essere acquisite tramite la **classe R**, il cui funzionamento simula un *collante* tra il codice e il documento XML; si tratta di un file generato e gestito automaticamente, ricreato qualora dovessero essere riportate delle modifiche.

Applica il design pattern *Singleton*, pertanto l'intero progetto sviluppato fonda sulla singola istanza della classe. Quindi, ricapitolando, **R** contiene tutti gli identificativi delle resources poste all'interno della directory *res*.

```
public finale class R {
    public static final class string {
        public static final int hello=0x7040001;
        public static final int labelButton=0x7040005;
    }
}
```

Definizione della classe R

L'*ID* definito precedentemente è composto da due sezioni esplicative, definite come segue:

- Il **type**, ossia il tipo della risorsa, ad esempio string, menu, layout ...
- Il **name**, a sua volta contraddistinto dal *filename* e dal *value* posto all'interno del documento XML

Proseguendo, esistono due modalità di accesso al contenuto delle risorse:

- Accesso tramite XML, in cui è attuata una sintassi simile a

  $$@[<package\_name>:]<resource\_type>/<resource\_name>$$

  Il *@[<package_ name>:]*, oltre ad essere una sezione opzionale, definisce il nome del *package* in cui la risorsa è allocata. Il *<resource_ type>* è il nome del *tipo* della risorsa. Il *<resource_ name>* rappresenta il *filename* della risorsa.

  Di seguito è proposto un titolo esemplificativo, in maniera tale che possa essere definito nel migliore dei modi il concetto espresso prima.

```
<LinearLayout>
    <TextView android:id="@+id/label1"
        android:text="@string/labelText"
            android:textcolor="@android:color/black"/>
    <Button android:id="@+id/button1" android:text="@string/labelButton"
        android:background="@color/my_red"/>
</LinearLayout>
```

Accesso al contenuto delle risorse mediante XML

Da esempio sono differenti i punti salienti in cui occorre soffermarsi. Nella dichiarazione della *TextView* si nota l'utilizzo della sintassi *<resource_ type>/<resource_ name>*, definita in *android:text="@string/labelText"*, mentre, sempre nella medesima sezione, si evidenzia la presenza di un value in cui è espresso il *@[<package_ name>:]*, in *android:textcolor="@android:color/black"*. Contrariamente, all'interno del tag *Button* è visualizzata la denominazione di un ulteriore *package name*, in questo case ritrae un pacchetto ideato dallo sviluppatore e non dato built-in da Android, come da nomenclatura precedente.

- Accesso tramite *codice Java/Kotlin*, dove la sintassi adeguata promuove

  $$[package\_name]R.resource\_type.resource\_name$$

  Nuovamente, il *package_name* definisce il *package* in cui la risorsa è allocata. Il *resource_type* è il nome del *tipo* della risorsa. Il *resource_name* è la denominazione del *filename*.

```
val hello:String = this.getResource().getString(R.string.hello)
val myRed:Color = getResource().getString(R.color.my_red)
val msgTextView = findViewById(R.id.label1) as TextView
msgTextView.setText(R.string.labelText)
```

Accesso al contenuto delle risorse mediante Kotlin

  In relazione allo sketch di codice proposto, possono essere evidenziate alcuna peculiarità. Alla variabile *hello* è assegnata una stringa contenuta all'interno del file *string.xml*; l'accesso è definito esattamente come stabilito dalla sintassi precedente, in cui è dichiarata la classe *R*, il *tipo* ed infine il *nome* della risorsa. Si sottolinea come in questa casistica affinchè possa essere garantito l'assegnamento, occorre stabilire preventivamente il *contesto* e la *funzione* che ne permettano il corretto svolgimento, espresso come *context.getResource()*; contrariamente in casi in cui non sia necessario allocare il contesto può essere omesso.

La sezione successiva definisce come utilizzare tipi e strutture dati, in maniera tale che sia fornita una prima panoramica.

Values

```
<resources>
    <string name="label">Example application</string>
    <integer name="value">58</integer>
    <string-array name="nameArray">
        <item>Ciao</item>
        <item>Hello</item>
        <item>Hola</item>
    </string-array>
    <integer-array name="valArray">
        <item>1</item>
        <item>2</item>
        <item>3</item>
    </integer-array>
</resources>
```

Manipolazione di tipi primitivi e strutture dati in XML

```
val label:String = resources.getString(R.string.label)
val value:Int = resources.getInteger(R.integer.value)
val nameArray:Array<Integer> = resources.getStringArray(R.array.nameArray)
val valArray:Array<Integer> = resources.getIntegerArray(R.array.valArray)
```

Manipolazione di tipi primitivi e strutture dati in Kotlin

```xml
<resources>
    <color name="coin_yellow">#FF9800</color>
    <color name="coin_yellow">#FF3131</color>
</resources>
```

Manipolazione di risorse color in XML

```kotlin
val coinColor:Int = resources.getColor(R.color.coin_yellow, null)
```

Manipolazione di risorse color in Kotlin

```xml
<resources>
    <dimen name="textview_height">25dp</dimen>
    <dimen name="textview_width">150dp</dimen>
    <dimen name="font_size">20sp</dimen>
</resources>
```

Manipolazione di risorse dimension in XML

```xml
<TextView
    android:layout_height="@dimen/textview_height"
    android:layout_width="@dimen/textview_width"
    android:textSize="@dimen/font_size"/>
```

Assegnamento delle dimen agli attributi del XMl layout

Uno **style** è un insieme di attributi, che possono essere applicati ad uno specifico elemento grafico della *GUI* oppure ad intere schermate dell'applicazione. Generalmente si tratta di un file XML, referenziato mediante l'utilizzo dell'identificativo riportato all'interno dell'attributo designato.

Un'ulteriore peculiarità prevede la possibilità di attuare il paradigma di *inheritance*, ossia organizzare gli stili mediante una gerarchia, dove ognuno di essi può acquisire proprietà grafiche da *style* corrispondenti. La sintassi adottata per la dichiarazione di uno stile prevede l'uso dei seguenti tag *<style>...</style>*, tipicamente creato all'interno della directory */res*.

```xml
<resources>
    <style name="MyTheme" parent="Theme.Material3.DayNight.NoActionBar">
        <item name="colorPrimary">@color/coin_yellow</item>
        <item name="colorSecondary">@color/hound_grey</item>
    </style>
</resources>
```

Dichiarazione di uno style

```xml
<Button style="@style/MyTheme"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:text="Push me!" />
```

Assegnamento dello stile ad un solo elemento grafico

```
<application
    ...
    android:theme="@style/MyTheme"
>...</ application>
```

Assegnamento dello stile all'intera applicazione

Gli sketch di codice definiti precedentemente testimoniano la volontà di realizzare applicazioni legate alla scrittura di "buon" codice, non solo a livello di operatività, ma anche sotto l'aspetto architetturale delle molteplici directories, in cui si evidenzia come ogni componente abbia la sua locazione apposita, dove lo strumento degli identificativi ne facilita l'accesso.

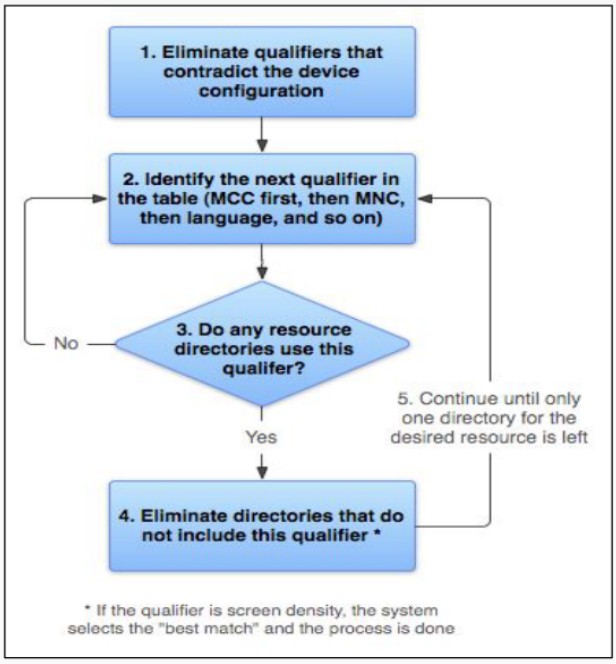
Resource Alternatives

Le applicazioni *Android* dovrebbero provvedere a risorse alternative in base alla tipologia e configurazione del device di riferimento. Durante *runtime* Android rileva le caratteristiche correnti del dispositivo e carica le *resources* appropriate per l'applicazione.

Per definire appropiatamente le *resource alternatives* occorre attuare due step fondamentali:

- Creare una nuova *directory* all'interno di */res*, denominata *<resource_ name>-<qualifier>*
- Salvare la *risorsa alternativa* all'interno della cartella

Di seguito è proposto il principio di funzionameno secondo cui Android riesce a descriminare quali risorse siano associabili alla configurazione del dispositivo. Quando l'applicazione richiede una risorsa, Android seleziona l'*alternative resource* da attuare durante *runtime*, tramite l'algoritmo mostrato in figura.



Come da algoritmo si notano cinque azioni principali, suddivise in:

- *Step 1*, eliminare tutti i *qualifiers* che contraddicono la configurazione del dispositivo

- *Step 2*, identificare il prossimo *qualifier* all'interno della tabella
- *Step 3*, stabilire se qualsiasi cartella presente all'interno di *resource* utilizzi il *qualifier* giudicato
- *Step 4a*, eliminare tutte le *directories* che non includono il *qualifier*
- *Step 4b*, il processo continua fino a quando non sia rimasta una singola cartella

Ricapitolando elimina tutti i *qualifiers* che contraddicono le specifiche del device.