Introduzione alla piattaforma Android

Pulsanti, caselle di testo, Layout e Log

La classe Log

- Una utile API per le nostre operazioni di Logging
- Un po' come il System.out… che usavamo nei nostri primi programmi Java
- Mette a disposizione del programmatore diversi metodi secondo le nostre necessità di voler fornire più o meno informazioni

La classe Log

- Di solito si usano i metodi
  1. Log.v()
  2. Log.d()
  3. Log.i()
  4. Log.w()
  5. Log.e()
- E' convenzione dichiarare una costante TAG all'interno della nostra classe ed utilizzarla nelle successive invocazioni dei nostri metodi di Logging

```
private static final String TAG = "MyActivity";
```


La classe Log – Un semplice esempio

- Utilizzeremo il metodo Log.v
- Cerchiamo la firma e le info nella documentazone ufficiale :

static int	v (String tag, String msg) Send a VERBOSE log message.

- Msg è chiaramente l'effettivo messaggio di log che vogliamo stampare

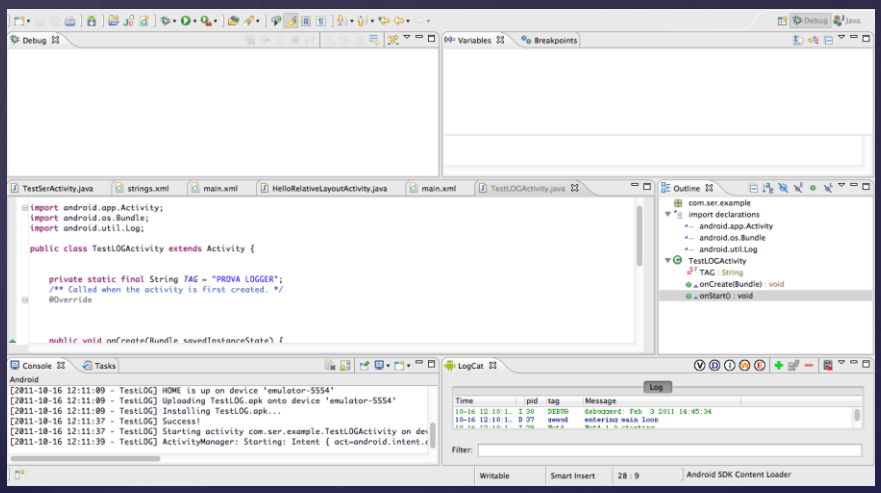
La classe Log – Un semplice esempio

```
import android.app.Activity;
import android.os.Bundle;
import android.util.Log;

public class TestLOGActivity extends Activity {


    private static final String TAG = "PROVA LOGGER";
    /** Called when the activity is first created. */
    @Override



    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        Log.v(TAG, "Sono nel metodo onCreate");

    }

    public void onStart() {
        super.onStart();
        Log.v(TAG,"Sono nel metodo onStart");


    }
```

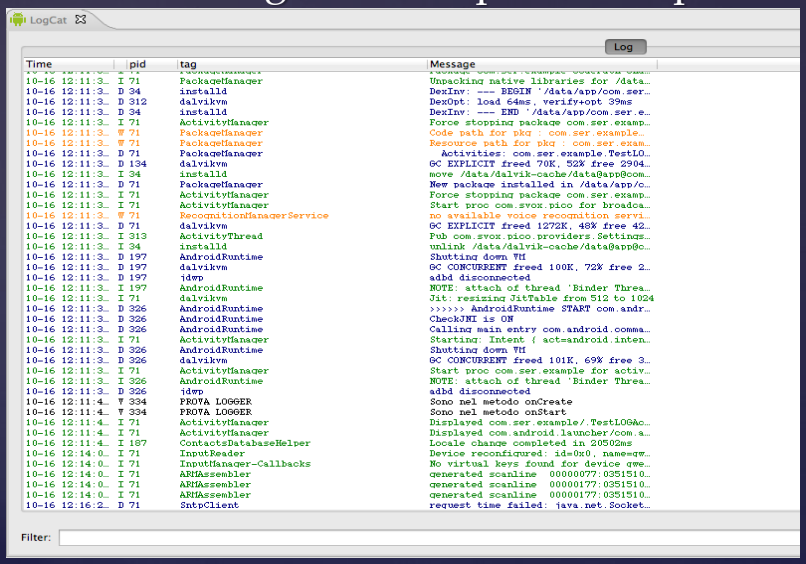
La classe Log – Un semplice esempio

- In questo semplice esempio vogliamo essere informati non appena venga invocato il metodo onCreate o il metodo onStart durante il ciclo di vita dell'Activity
- Dove viene stampato l'output del Log ?
- Se usiamo il plugin per Eclipse basata selezionare la "prospettiva" Debug (Clicchiamo sul menu Window -> Open Perspectve -> Debug) e dare una occhiata a cosa succede in basso a destra….

La classe Log – Un semplice esempio

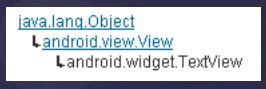


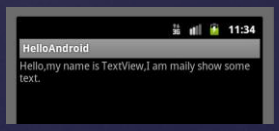
La classe Log – Un semplice esempio



Caselle di testo

- Le TEXTVIEW :
  - Visualizzano del testo all'utente e (opzionalmente) ne permettono la modifica
  - Rappresentano editor di testi completi, TUTTAVIA la classe base è configurata per non consentire la modifica del testo
  - Per le modifiche possiamo utilizzare la sottoclasse EditText



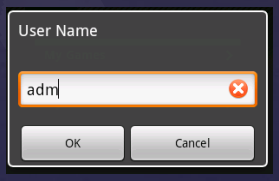


Caselle di testo

- Alcuni metodi utili
  - setText() Cosa farà mai?
  - setTextColor()
  - length() Restituisce la lunghezza, in caratteri, del testo
  - getText()
  - append() Aggiunge del testo, accodandolo all'eventuale testo già esistente

Caselle di testo - EditText

- Una piccola aggiunta alle TextView per renderle editabili



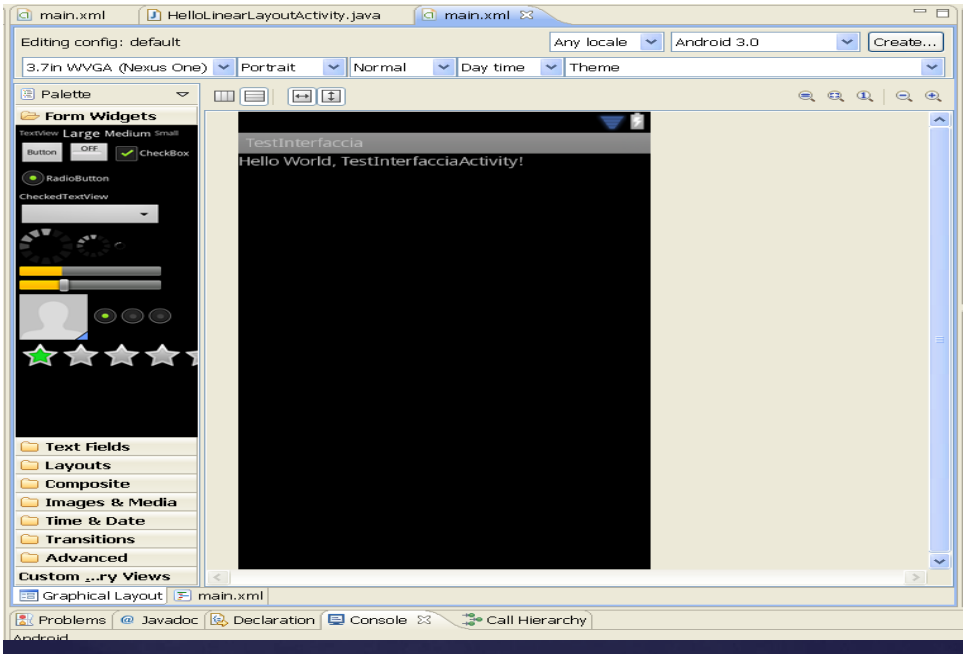
java.lang.Object
↳android.view.View
↳android.widget.TextView
↳android.widget.EditText

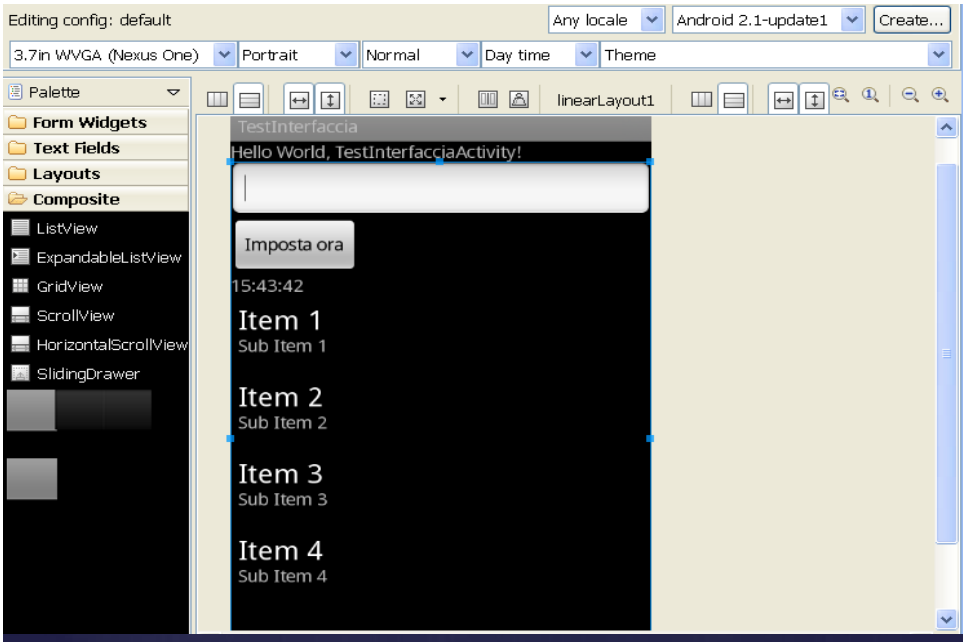
I Layout grafici in Android

I Layout grafici in Android

PREMESSA

- A partire dalle ultime versioni dell' SDK è disponibile un tool che consente di ''disegnare'' la nostra interfaccia con semplici operazioni di drag 'n drop (Un po' come avviene in tutti gli IDE moderni…)
- Dietro le quinte questo tool non fa altro che ''riempire'' per noi i file XML a cui abbiamo accennato nella prima lezione…
- TUTTAVIA ,per poter sfruttare a pieno tutte le potenzialità che Android mette a disposizione è comunque necessario conoscere il significato del codice XML che il tool scrive per noi…





La classe ViewGroup

- Nella scorsa lezione abbiamo accennato alla classe View
- Una ViewGroup è una View ''speciale'' che può contenere al suo interno altre View
- E' la classe alla base dei vari layout grafici che possiamo utilizzare in Android
- Definisce anche la classe ViewGroup.LayoutParams utilizzata per impostare i parametri relativi ai Layout

I layout di Android - LinearLayout

- Il tipo di layout più semplice
- Una ViewGroup che visualizza gli elementi al proprio interno seguendo una direzione ''lineare''
- Gli elementi possono essere allineati in orizzontale o in verticale

I layout di Android - LinearLayout

- Un LinearLayout può avere come proprio ''figlio'' un altro LinearLayout
- TUTTAVIA se ci accorgiamo che per realizzare la nostra interfaccia stiamo innestando troppi LinearLayout forse è il caso di utilizzare qualcosa di diverso ad esempio un RelativeLayout (vedremo tra un po' di cosa si tratta)

I layout di Android - LinearLayout

- Diamo uno sguardo al tutorial ufficiale…
- Ricordate cosa abbiamo detto nella scorsa lezione sulle differenze tra Programmatic UI e XML Layout ?
- Creiamo un nuovo progetto e andiamo a modificare il file main.xml contenuto nella sottodirectory /res/layout

I layout di Android - LinearLayout

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

  <LinearLayout
      android:orientation="horizontal"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:layout_weight="1">
      <TextView
          android:text="red"
          android:gravity="center_horizontal"
          android:background="#aa0000"
          android:layout_width="wrap_content"
          android:layout_height="fill_parent"
          android:layout_weight="1"/>
      <TextView
          android:text="green"
          android:gravity="center_horizontal"
          android:background="#00aa00"
          android:layout_width="wrap_content"
          android:layout_height="fill_parent"
          android:layout_weight="1"/>
      <TextView
          android:text="blue"
          android:gravity="center_horizontal"
          android:background="#0000aa"
          android:layout_width="wrap_content"
          android:layout_height="fill_parent"
          android:layout_weight="1"/>
      <TextView
          android:text="yellow"
          android:gravity="center_horizontal"
          android:background="#aaaa00"
          android:layout_width="wrap_content"
          android:layout_height="fill_parent"
          android:layout_weight="1"/>
  </LinearLayout>
```

I layout di Android - LinearLayout

```
<LinearLayout
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:layout_weight="1">
  <TextView
      android:text="row one"
      android:textSize="15pt"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:layout_weight="1"/>
  <TextView
      android:text="row two"
      android:textSize="15pt"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:layout_weight="1"/>
  <TextView
      android:text="row three"
      android:textSize="15pt"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:layout_weight="1"/>
  <TextView
      android:text="row four"
      android:textSize="15pt"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:layout_weight="1"/>
</LinearLayout>

</LinearLayout>
```

I layout di Android - LinearLayout

I layout di Android - LinearLayout

```
<LinearLayout
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_weight="1">
```

orientation : con ''horizontal'' tutti gli oggetti verranno allineati ''per riga''
(in particolare ogni oggetto occuperà una riga nel layout)
con ''vertical'' tutti gli oggetti verranno allineati ''per colonna''
(in particolare ogni oggetto occuperà una colonna nel layout)

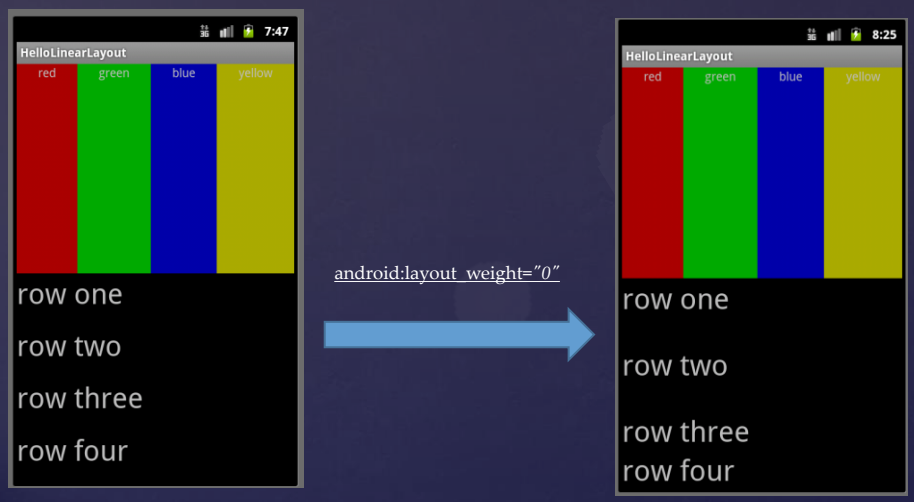
layout_width : Rappresenta l'estensione in ''ampiezza''.
Con il valore ''fill_parent'' facciamo si che l'ampiezza del nostro layout sia pari all'ampiezza del suo genitore

layout_height : Rappresenta l'estensione in ''altezza''.
Con il valore ''fill_parent'' facciamo si che l'altezza del nostro layout sia pari all'ampiezza del suo genitore

I layout di Android - LinearLayout

- weight : E' un valore che descrive come gestire lo spazio extra presente nel layout
- Se il valore è 0 non verrà intrapresa alcuna azione, altrimenti lo spazio extra verrà distribuito alle view che hanno un peso (weight) maggiore di 0
- Vediamo un esempio : Settiamo a 0 il valore weight delle textview ''row_three'' e ''row_four'' e osserviamo come viene modificato il nostro layout

I layout di Android – LinearLayout



I layout di Android - LinearLayout

```
<TextView
    android:text="red"
    android:gravity="center_horizontal"
    android:background="#aa0000"
    android:layout_width="wrap_content"
    android:layout_height="fill_parent"
    android:layout_weight="1"/>
```

- gravity : Specifica come allineare il testo, rispetto alle assi X e Y della casella di testo, quando è più piccolo della casella stessa.
- Può assumere i seguenti valori :

I layout di Android – LinearLayout

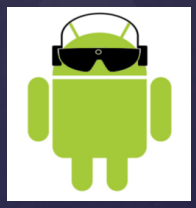
```
<TextView
    android:text="red"
    android:gravity="center_horizontal"
    android:background="#aa0000"
    android:layout_width="wrap_content"
    android:layout_height="fill_parent"
    android:layout_weight="1"/>
```

- layout_width : con ''wrap_content'' la nostra view è grande ''quanto basta'' per adattarsi al suo contenuto

I layout di Android - LinearLayout

- Abbiamo progettato la nostra interfaccia
- Le nostre caselle di testo sono visualizzate correttamente con i colori che desideriamo
- E se ora vogliamo modificarne il contenuto ?

  Vediamo un possibile metodo…



Ottenere il riferimento per una View

- Consideriamo il seguente problema :

Vogliamo modificare il testo di una TextView quando nella nostra Activity viene invocato il metodo onResume()

Ottenere il riferimento per una View

- Creiamo un nuovo progetto e impostiamo un semplicissimo layout…

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">


      <TextView
          android:id="@+id/mia_caselladitesto"
          android:text="blue"
          android:gravity="center_horizontal"
          android:background="#0000aa"
          android:layout_width="fill_parent"
          android:layout_height="fill_parent"
          />

</LinearLayout>
```

Notate qualche differenza con gli esempi precedenti?

```java
package com.ser.example;

import android.app.Activity;

public class HelloLinearLayoutActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    public void onRestart() {
        super.onRestart();

        TextView myText = (TextView) findViewById(R.id.mia_caselladitesto);
        myText.setText("Invocato onRestart");



    }

}
```


Ottenere il riferimento per una View – Gli IDs

- Le View possono avere un id (di tipo intero) ad esse associato
- Di solito, questi id sono a loro assegnati direttamente nei file XML che definiscono il layout e sono utilizzati per identificare una view specifica
- Vediamo un altro esempio che ci sarà utile anche per introdurre un nuovo componente per le nostre interfacce grafiche…

I Pulsanti

java.lang.Object
↳ android.view.View
↳ android.widget.TextView
↳ android.widget.Button


```
<Button
    android:id="@+id/my_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/my_button_text"/>
```

```
Button myButton = (Button) findViewById(R.id.my_button);
```


I Pulsanti

- Un Button è l'equivalente del il classico pulsante presenti nelle tradizionali interfacce desktop.
- Viene ''tappato'' da un utente per avviare una azione
- Vediamo un esempio di utilizzo…

I Pulsanti

```
public class MyActivity extends Activity {
    protected void onCreate(Bundle icicle) {
        super.onCreate(icicle);

        setContentView(R.layout.content_layout_id);

        final Button button = (Button) findViewById(R.id.button_id);
        button.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                // Perform action on click
            }
        });
    }
}
```

- OnClickListener è la definizione di una interfaccia che ci fornisce un ''callback'' da invocare quando clicchiamo su un oggetto di tipo View
- Il metodo onClick () viene invocato al momento del ''click'' effettivo

I Pulsanti

- Un metodo alternativo per far compiere una azione ad un pulsante consiste nell'assegnargli un metodo direttamente nel codice XML usando l'attributo android: onClick …

```
<Button
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:text="@string/self_destruct"
    android:onClick="selfDestruct" />
```

… e implementare poi il codice effettivo del metodo nel nostro file sorgente :

```
public void selfDestruct(View view) {
    // Kabloey
}
```

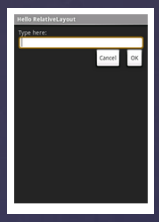
I Layout di Android – RelativeLayout

- Un componente dell'interfaccia grafica viene visualizzato ed allineato ''relativamente'' alla posizione di un altro componente
- Esempio : Un pulsante può essere allineato a sinistra ''relativamente'' ad una casella di testo, oppure ''relativamente'' all'area di schermo gestita dal RelativeLayout Stesso

I Layout di Android - RelativeLayout

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView
        android:id="@+id/label"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Type here:"/>
    <EditText
        android:id="@+id/entry"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:background="@android:drawable/editbox_background"
        android:layout_below="@id/label"/>
    <Button
        android:id="@+id/ok"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/entry"
        android:layout_alignParentRight="true"
        android:layout_marginLeft="10dip"
        android:text="OK" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_toLeftOf="@id/ok"
        android:layout_alignTop="@id/ok"
        android:text="Cancel" />
</RelativeLayout>
```

I Layout di Android – RelativeLayout



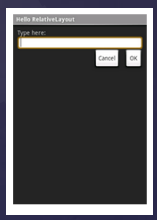
Notate la posizione dei due pulsanti ?
Quanti LinearLayout innestati avremmo dovuto usare per ottenere
lo stesso risultato?

Analizziamo il file XML della slide precedente….

I Layout di Android - RelativeLayout

```
<TextView
    android:id="@+id/label"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Type here:"/>
```

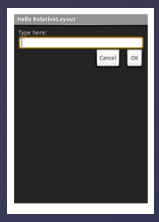
Tutto chiaro, giusto ? Prestiamo solo
Un minimo di attenzione al campo ID




```
<EditText
    android:id="@+id/entry"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:background="@android:drawable/editbox_background"
    android:layout_below="@id/label"/>
```

Layout_below : Diciamo alla nostra EditText di ''posizionarsi'' sotto
la View il cui id è ''label''

I Layout di Android - RelativeLayout




```
<Button
    android:id="@+id/ok"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@id/entry"
    android:layout_alignParentRight="true"
    android:layout_marginLeft="10dip"
    android:text="OK" />
```

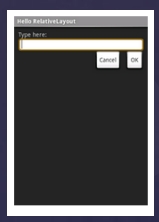
Layout_below : Diciamo al nostro Button di ''posizionarsi'' sotto la EditText con id ''entry''

Layout_alignParentRight : Se il valore è ''true'' il bordo destro del componente verrà ''allineato'' al bordo destro del genitore (in questo caso il nostro RelativeLayout)

Layout_marginLeft : Specifichiamo lo ''spazio'' che deve essere lasciato libero alla sinistra del componente

I Layout di Android – RelativeLayout

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_toLeftOf="@id/ok"
    android:layout_alignTop="@id/ok"
    android:text="Cancel" />
```



Layout_toLeftOf : Diciamo al nostro ''bottone'' di posizionarsi in modo tale che il proprio bordo destro sia alla sinistra della View con id ''ok''

Layout_alignTop : Diciamo al nostro ''bottone'' di posizionare il proprio margine superiore allo stesso livello del margine superiore della View con id ''ok''

I Layout di Android – DIP, DP, SP,PX ecc ecc

```
android:layout_marginLeft="10dip"
```

- Avrete sicuramente notato, nelle slide precedenti, la stringa ''10dip'' ; cerchiamo di saperne di più

- DIP (o DP) è un acronimo per Density Independet Pixel.
- Utilizzare una misura espressa in ''dip'' ci permette di sviluppare facilmente layout per dispositivi multipli
- Se una risorsa è espressa in ''dip'' essa verrà automaticamente adattata alla risoluzione disponibile sul nostro dispositivo Android

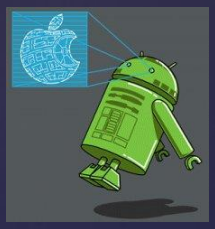
I Layout di Android – DIP, DP, SP,PX ecc ecc

- PX corrisponde ai pixel ''reali'' presenti sullo schermo del dispositivo
- PT, acronimo per ''Points'' , corrisponde a 1/72 di pollice (1 pollice = 2,54 cm)
- SP, acronimo di Scale Independent Pixels, è simile ad DP ma viene anche adattato in base alle preferenze espresse dall'utente relativamente alla dimensione dei font.

Quando specifichiamo la dimensione di un font può essere molto utile utilizzare SP come ''unità di misura''

I Layout di Android – DIP, DP, SP,PX ecc ecc

- Cerchiamo sempre di usare DP o SP
- Le nostre applicazioni saranno maggiormente compatibili con le diverse risoluzioni presenti nei dispositivi attualmente in commercio



I Layout di Android - TableLayout

- Gli elementi vengono visualizzati raggruppati in righe e colonne
- Sintassi HTML like
- All' interno di una cella della tabella possiamo visualizzare qualsiasi cosa

I Layout di Android - TableLayout

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:stretchColumns="1">

    <TableRow>
        <TextView
            android:layout_column="1"
            android:text="Open..."
            android:padding="3dip" />
        <TextView
            android:text="Ctrl-O"
            android:gravity="right"
            android:padding="3dip" />
    </TableRow>

    <TableRow>
        <TextView
            android:layout_column="1"
            android:text="Save..."
            android:padding="3dip" />
        <TextView
            android:text="Ctrl-S"
            android:gravity="right"
            android:padding="3dip" />
    </TableRow>
```


I Layout di Android - TableLayout

```
    <TableRow>
        <TextView
            android:layout_column="1"
            android:text="Save As..."
            android:padding="3dip" />
        <TextView
            android:text="Ctrl-Shift-S"
            android:gravity="right"
            android:padding="3dip" />
    </TableRow>

    <View
        android:layout_height="2dip"
        android:background="#FF909090" />

    <TableRow>
        <TextView
            android:text="X"
            android:padding="3dip" />
        <TextView
            android:text="Import..."
            android:padding="3dip" />
    </TableRow>

    <TableRow>
        <TextView
            android:text="X"
            android:padding="3dip" />
        <TextView
            android:text="Export..."
            android:padding="3dip" />
        <TextView
            android:text="Ctrl-E"
            android:gravity="right"
            android:padding="3dip" />
    </TableRow>
```

I Layout di Android - TableLayout

```
    <TableRow>
        <TextView
            android:layout_column="1"
            android:text="Quit"
            android:padding="3dip" />
    </TableRow>
</TableLayout>
```

Il Layout è abbastanza intuitivo
Approfondiamo giusto qualche dettaglio…

I Layout di Android - TableLayout

```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:stretchColumns="1">
```

- L'ampiezza globale di una colonna della tabella è definita dalla riga che ha la cella, relativamente a quella colonna, più ampia
- stretchColumns : Specifichiamo se le colonne possono ''allungarsi'' fino ad occupare anche spazio ''extra''

I Layout di Android - TableLayout

```
<TableRow>
    <TextView
        android:layout_column="1"
        android:text="Open..."
        android:padding="3dip" />
    <TextView
        android:text="Ctrl-O"
        android:gravity="right"
        android:padding="3dip" />
</TableRow>
```

- <TableRow> : Un layout che 'dispone' in orizzontale i propri figli
- Dovrebbe essere utilizzato sempre un figlio di TableLayout
- Se non lo è si comporterà sempre allo stesso modo di un LinearLayout che dispone in orizzontale i propri elementi

I Layout di Android - TableLayout

```
<TableRow>
    <TextView
        android:layout_column="1"
        android:text="Open..."
        android:padding="3dip" />
    <TextView
        android:text="Ctrl-O"
        android:gravity="right"
        android:padding="3dip" />
</TableRow>
```

- Gli elementi contenuti in un <TableRow> non hanno bisogno di specificare gli attributi ''layout_width'' o ''layout_height''
- TableRow forza, in automatico, questi due attributi ad assumere rispettivamente i valori MATCH_PARENT e WRAP_CONTENT

I Layout di Android - TableLayout

- E layout_column ? A che serve ?
- Definisce lo ''Zero based index'' della colonna ove vogliamo visualizzare il nostro oggetto…
- Un esempio ci aiuterà immediatamente a capire…

```
<TableRow>
    <TextView
        android:layout_column="1"
        android:text="Open..."
        android:padding="3dip" />
    <TextView
        android:text="Ctrl-O"
        android:gravity="right"
        android:padding="3dip" />
</TableRow>
```

Modifichiamo il valore dell'attributo

```
<TableRow>
    <TextView
        android:layout_column="0"
        android:text="Open..."
        android:padding="3dip" />
    <TextView
        android:text="Ctrl-O"
        android:gravity="right"
        android:padding="3dip" />
</TableRow>
```


I Layout di Android - TableLayout

Layout_column = ''1''

Layout_column = ''0''

I Layout di Android - TableLayout

```
<View
    android:layout_height="2dip"
    android:background="#FF909090" />
```

- E' sempre possibile utilizzare una View di base per ''simulare'' una linea di separazione tra una riga e l'altra

E se non voglio usare l'XML ?

- I vantaggi derivanti dal definire la nostra interfaccia mediante file XML dovrebbero essere oramai chiari…
- TUTTAVIA può sempre presentarsi la necessità di dover definire la nostra interfaccia utente totalmente da codice..

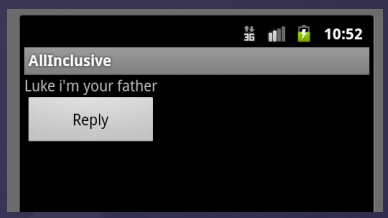
E se non voglio usare l'XML ?

```
import android.app.Activity;

public class AllInclusiveActivity extends Activity {

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        TableLayout tl=new TableLayout(this);

        TableRow trI=new TableRow(this);
        TableRow trII = new TableRow(this);

        // Attenzione a non dimenticare final
        final TextView myText = new TextView(this);
        myText.setText("Luke i'm your father");

        Button btn=new Button(this);
        btn.setText("Reply");

        trI.addView(myText);
        trII.addView(btn);

        tl.addView(trI);
        tl.addView(trII);
        setContentView(tl);
```

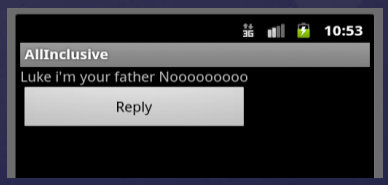

E se non voglio usare l'XML ?

```
        btn.setOnClickListener(new View.OnClickListener() {

            public void onClick(View v) {
                myText.append(" Nooooooooo");
            }
        });

    }
}
```

E se non voglio usare l' XML ?



Cliccando sul pulsante…



Riferimenti

ANDROID DEVELOPERS
http://developer.android.com/index.html

Reto Meier
Professional Android 2 Application Development
Wrox Press