

# Programmare per Android: User Interface v0.1beta



ANDROID



Rossi Pietro Alberto

# Indice

Introduzione alla programmazione ad oggetti.....	5
Introduzione alla programmazione visuale.....	6
Introduzione alla programmazione ad eventi.....	7
UserControls.....	
• TextView.....	10
• EditText.....	11
• AutoCompleteTextView.....	12
• MultiAutoCompleteTextView.....	14
• Button.....	15
• ImageButton.....	16
• ToggleButton.....	17
• CheckBox.....	18
• RadioButton.....	19
• ListView.....	21
• GridView.....	22
• DatePicker.....	23
• TimePicker.....	24
• AnalogClock.....	25
• DigitalClock.....	26
Altri Controlli.....	
• CheckedTextView.....	28
• Chronometer.....	29
• ImageView.....	31
• ProgressBar.....	32
• RatingBar.....	33
• SeekBar.....	34
• Toast.....	35
Controlli Avanzati.....	
• Spinner.....	38
• TabWidget.....	40
• WebView.....	42
• MapView.....	44
• PopupWindow.....	46
• ExpandableListView.....	49

Layout Manager.....	
• LinearLayout.....	53
• TableLayout.....	55
• RelativeLayout.....	57
• AbsoluteLayout.....	59
• FrameLayout.....	61
Widget.....	64

## Premessa

Dopo molto tempo che giriamo su Android e varie, finalmente abbiamo pubblicato questa prima bozza di una nostra guida!

Più che beta, dovremmo definirla in versione alpha, in quanto è ancora un cantiere ben aperto e speriamo di non chiuderlo mai!

Il nostro scopo è quello di portare conoscenze sulla programmazione Android dove non ci sono, ed arricchire le già molte presenti.

Passando alle presentazioni, mi chiamo Alberto e sono uno sviluppatore su sistemi POS in linguaggio C, recentemente passato anche alla programmazione su sistemi Android.

Sono stato relatore al Linux Day 2010 di Palermo, presentando Android e tutti i suoi aspetti migliori, non tralasciando nulla e posando la prima pietra di un grande castello, spero.

Questa mini guida non la definirei mia, ma nostra, di tutta la comunità.

Da buon guru di Linux, la guida è libera e scaricabile gratuitamente dal sito [www.ibasblog.it](http://www.ibasblog.it).

Non metterò mai in vendita questa guida, non è lo scopo per cui è stata fatta, anzi per chi vuole, può collaborare, postare esempi di codice, implementare argomenti.

Non c'è nessun problema, basta contattarmi al mio indirizzo di posta [pietroalberto.rossi@gmail.com](mailto:pietroalberto.rossi@gmail.com) e proporre quello che volete.

Nota dolente, è il mio italiano...mi dispiace non sono madrelingua ma sto imparando!

Spero mi perdoniate nel caso qualche verbo non vi vada giù.

Come detto prima, la guida è in costante sviluppo, già sono pronte altre parti che non sono state integrate in questa beta, e ci saranno molte novità sul futuro.

Se gli argomenti sono trattati troppo brevemente è perché ho pensato che leggere troppo annoia un programmatore, preferisco essere breve e conciso, il resto si aggiusterà dopo!

Ah, dimenticavo...metterò anche un po' di stile e qualche font diverso, tranquilli...

Ringrazio [ibasblog.it](http://ibasblog.it) per il tempo da loro dedicatomi, e per avermi mandato all'Android Developer Lab a Firenze (evento stupendo!), in particolare ringrazio:

Claudio, cd125

Pino, pinosiciliano

Da non dimenticare [Androidiani.com](http://Androidiani.com), la più grande community di riferimento per il mondo Android, hanno organizzato un Android Lab in Italia, più di così?

Ci hanno permesso di inseguire questo sogno e non smetteremo di dirgli 1000 volte grazie!

## Introduzione alla programmazione ad oggetti

Quando iniziamo a realizzare un programma, sia di 10 righe sia di 100 mila righe, a primo impatto andiamo a cercare una soluzione al nostro problema, pensando principalmente a non avere troppe complicazioni man mano che si scrive il codice.

Fra le considerazioni iniziali c'è sempre la famosa domanda “Che linguaggio utilizziamo?”. Quello che noi andremo a cercare è un paradigma che ci permette di risolvere il problema, nella maniera più semplice possibile. Da qui discende che un paradigma è un modello comportamentale di programmazione o, più semplicemente, uno stile di programmazione.

La scelta del linguaggio di programmazione è molto importante, soprattutto se un progetto è abbastanza grande. Essa può derivare anche dal tipo di problema da risolvere: molti linguaggi dispongono di librerie, già predisposte, che semplificano alcune procedure ritenute fondamentali o soltanto per il semplice scopo di semplificare la vita del programmatore.

Il modo di pensare come programmare un applicazione è fondamentale.

In programmazione imperativa ci mettiamo davanti il progetto e ci chiediamo cosa fare, fissando l'attenzione sull'obiettivo da raggiungere e di come raggiungerlo.

In programmazione ad oggetti, invece, dobbiamo cercare di non personalizzare l'oggetto ma continueremo a pensare l'oggetto così come è.

Penseremo ad un oggetto astratto, cercando di individuare quali sono le caratteristiche dell'oggetto e le sue potenzialità.

Gli elementi caratteristici sono chiamati Attributi dell'oggetto, le potenzialità sono invece chiamate Metodi.

Da notare che, molti programmatori, le potenzialità dell'oggetto li chiama erroneamente funzioni: dato che possono somigliare parecchio alle funzioni utilizzate in qualunque linguaggio imperativo.

Durante l'analisi del problema non importa cosa faccia il metodo, l'importante è rispecchiare le potenzialità e successivamente svilupparne il problema.

Altro errore che spesso si sente è l'utilizzo di Classe e Oggetto come se fossero la stessa cosa: un Oggetto è la rappresentazione reale della problematica presa in considerazione, la Classe invece è la rappresentazione astratta dell'Oggetto.

Per esempio, se consideriamo una penna, essa nella realtà rappresenta il nostro Oggetto vero e proprio, così come è fatto: sappiamo che è una penna, il colore, il tipo, ecc...

La Classe che rappresenterà la penna sarà, per definizione, l'astrazione dell'Oggetto. Qui ci possiamo chiedere “E cosa cambia? Non è la stessa cosa?”, la risposta è parzialmente negativa perché, attraverso questo processo di astrazione, noi andremo a determinare anche altri dettagli supplementari dell'Oggetto: se scrive o meno, livello di inchiostro, ecc.

## Introduzione alla programmazione visuale

In base alla modalità con cui l'utente interagisce con il sistema, le interfacce utente possono essere classificate in: interfacce testuali ed interfacce grafiche.

Le interfacce testuali, dette anche interfacce a caratteri, le ricordiamo più o meno tutti, come il buon vecchio MS-DOS e la shell di Linux. Hanno fatto la storia dei computer, all'inizio erano tutti ad interfaccia testuale, rendendo l'uso stesso del computer poco accessibile al pubblico.

Le interfacce grafiche sono le odierne interfacce con cui interagiamo. Tutti quei oggetti che si muovono sullo schermo, hanno migliorato l'uso del computer, rendendolo accessibile a tutti e diffusissimo in tutto il mondo.

La differenza principale fra interfaccia testuale e visuale è la modalità con cui l'utente interagisce con il computer.

Nel primo caso l'utente interagisce solo con la tastiera, le istruzioni proposte a video possono variare nell'aspetto in modo molto limitato, il monitor viene visto come una matrice di celle con un determinato numero di righe e di colonne e l'inserimento dei dati è controllato dal programma.

Nel secondo caso l'utente interagisce attraverso vari strumenti come il mouse e la tastiera, il monitor è visto come una matrice di pixel caratterizzata da un certo numero di righe e di colonne (risoluzione), ogni carattere visualizzato può avere una propria dimensione e un proprio aspetto, il controllo del flusso del programma è deciso dall'utente.

In un'interfaccia utente grafica, l'interazione tra utente e sistema avviene grazie ad un sistema di gestione degli eventi. Qualsiasi interazione tra la macchina e l'esterno genera un evento che viene intercettato e gestito dal sistema operativo.

L'interfaccia utente grafica, principalmente, viene vista come una collezione di oggetti annidati, nel senso che saranno presenti oggetti che conterranno degli altri.

Occorre subito distinguere due tipi di oggetti di una GUI (Graphics Unit Interface): oggetti contenitori ed oggetti componenti.

Gli oggetti contenitori sono quelli all'interno dei quali è possibile posizionare e dimensionare altri oggetti (Finestre e Pannelli).

La Finestra è costituita da un'area che può contenere vari elementi. Può essere presente una barra del titolo, all'interno della quale ci sono i pulsanti che gestiscono la sua chiusura ed il suo ridimensionamento.

Il Pannello è costituito da un rettangolo, il cui perimetro può essere reso visibile o meno, così come il suo contenuto.

Gli oggetti componenti sono quelli che vengono inseriti all'interno degli oggetti contenitori (Label, Button, CheckBox, ecc...).

Una Label (etichetta) è una stringa di testo utilizzata per etichettare altri componenti o visualizzare una descrizione visibile nella GUI.

I Button (pulsanti) sono semplici componenti che avviano un'azione quando vengono premuti.

Una TextField (campo di testo) è costituita da una o più righe e consente di immettere testo al suo interno.

## Introduzione alla programmazione ad eventi

Quando si verifica un qualsiasi evento, il sistema lo intercetta ed invia le informazioni (tipo di evento ed oggetto origine) ad uno speciale manipolatore degli eventi.

Gli eventi controllabili dal nostro sistema generalmente tre:

- Eventi del mouse
- Eventi della tastiera
- Eventi del sistema

Gli eventi del mouse sono quelli che più comunemente avvengono mentre utilizziamo un PC: basta muovere il mouse o cliccare su un punto dello schermo per generare un evento.

Gli eventi del mouse si dividono in tre sotto-categorie:

- Eventi generati dai pulsanti del mouse
- Eventi generati dal movimento del mouse
- Eventi generati dal trascinamento del mouse

La prima sotto-categoria definisce eventi quali:

- Click, che viene generato quando si clicca su un oggetto
- DoppioClick, che viene generato con un doppio click
- PulsanteMousePremuto, che viene generato quando si tiene premuto il pulsante sinistro del mouse
- PulsanteMouseRilasciato, che viene generato quando si rilascia il pulsante sinistro del mouse precedentemente premuto

La seconda definisce eventi quali:

- MouseSopra, che viene generato quando il mouse si muove su un oggetto
- MouseVia, che viene generato quando il mouse si sposta da un oggetto
- MovimentoMouse, che viene generato quando il mouse si muove

Infine, la terza definisce eventi quali:

- TrascinaMouse, che viene generato quando un utente trascina il mouse con il pulsante premuto
- MouseTrascinaOggetto, che viene generato quando un utente trascina un oggetto sulla finestra

Gli eventi della tastiera sono quelli classici che avvengono quando premiamo un tasto o una combinazione di essi.

Gli eventi della tastiera si dividono in:

- TastoAttivato, che viene generato quando un utente preme e rilascia un tasto o quando tiene premuto un tasto
- TastoPremuto, che viene generato quando viene premuto un tasto
- TastoRilasciato, che viene generato quando viene rilasciato un tasto precedentemente premuto

Gli eventi del sistema sono quei eventi generati con le iterazioni fra degli oggetti, come finestre, ed il sistema.

Gli eventi del sistema si suddividono in quattro sotto-categorie:

- Eventi generati dal caricamento degli oggetti
- Eventi generati dalle modifiche dell'utente
- Eventi legati al fuoco
- Eventi generati dai movimenti delle finestre

La prima sotto-categoria definisce eventi quali:

- Carica, che viene generato quando al caricamento degli oggetti
- Abbandona, funzione opposta a Carica

La seconda definisce eventi quali:

- Cambio, che viene generato quando viene modificato il valore di un oggetto

La terza definisce oggetti quali:

- PerdeFuoco, che viene generato quando si esce da un oggetto della GUI
- AcquistaFuoco, che viene generato quando si entra in un oggetto della GUI
- SelezionaTesto, che viene generato quando si seleziona del testo all'interno di un campo, di un'etichetta o di altri oggetti simili

La quarta definisce eventi quali:

- RidimensionaFinestra, che viene generato quando l'utente riduce o ingrandisce una finestra
- ScorriFinestra, che viene generato quando si effettua lo scorrimento della pagina sia con il mouse che con i tasti PgSu e PgGiù.

Un'altra categoria di eventi sono quelli generati dagli oggetti standard di una GUI e possono essere classificati nel seguente modo:

- Eventi d'azione: sono gli eventi generati quando un pulsante è stato premuto, quando si attiva una casella di testo o un pulsante di opzione, quando si seleziona una voce di un menu, quando si preme "Invio" dentro una casella di testo, ecc;
- Eventi di selezione o deselegione in un elenco: sono gli eventi legati alla selezione di una casella di controllo o di una voce di menu a scelta;
- Eventi di selezione o di deselegione per input: sono gli eventi generati quando un oggetto in risposta ad un click del mouse o all'utilizzo del tasto di tabulazione.

Questa piccola introduzione, teorica, agli eventi ci fa capire quanti eventi vengono generati anche solo premendo un tasto o muovendo il mouse.

A livello di linguaggio di programmazione non è difficile intercettare gli eventi e gestirli, molti linguaggi facilitano il lavoro, soprattutto quelli equipaggiati di un IDE visuale che ci mette a disposizione tutti gli strumenti per facilitare ed accelerare la programmazione.

# UserControls

- TextView..... 9
- EditText..... 10
- AutoCompleteTextView..... 11
- MultiAutoCompleteTextView..... 13
- Button..... 14
- ImageButton..... 15
- ToggleButton..... 16
- CheckBox..... 17
- RadioButton..... 18
- ListView..... 20
- GridView..... 21
- DatePicker..... 22
- TimePicker..... 23
- AnalogClock..... 24
- DigitalClock..... 25

## TextView

L'oggetto TextView è il classico componente label o etichetta di testo che serve per visualizzare del testo.

L'esempio di codice ne mostra la sua utilità più semplice, declinando aspetti come font e colore.

```
<?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:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Questa è una TextView"
    />
</LinearLayout>
```

E' facile notare che per modificare il testo basta editare la proprietà text.

Le altre proprietà, layout\_width e layout\_height servono per la grandezza del componente:

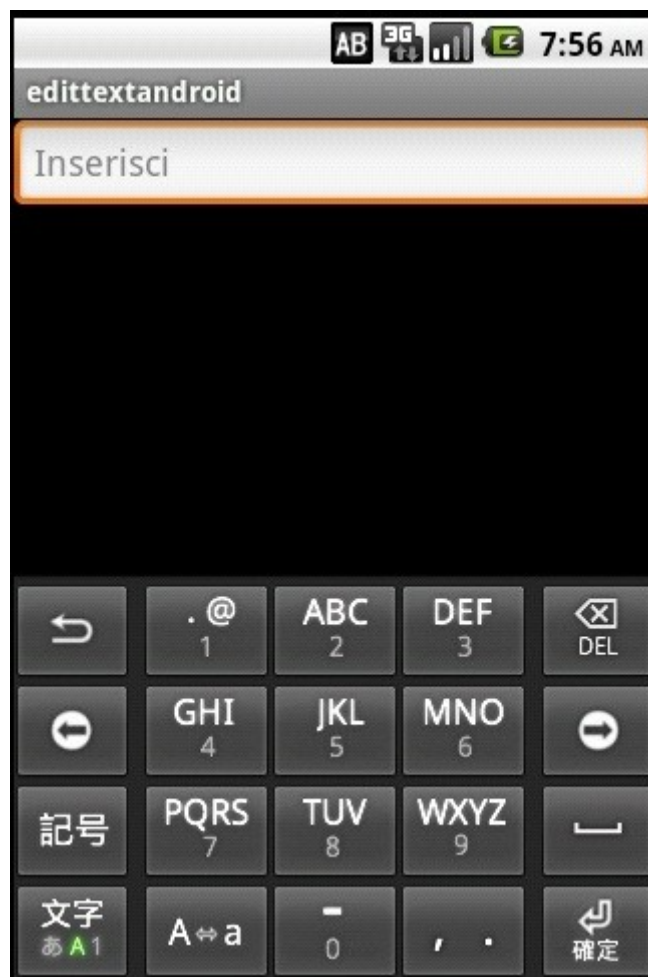
fill\_parent sta per riempi il contenitore e wrap\_content è la grandezza di default del componente.



## EditText

L'oggetto EditText è la comune TextBox di qualsiasi altro linguaggio di programmazione. La proprietà hint serve a definire un messaggio iniziale, viene visualizzato di colore grigio chiaro e quando si comincia a scrivere scompare. E' possibile sempre utilizzare la proprietà text per avere un testo di default ma solitamente si usa hint per far capire cosa bisogna scrivere.

```
<?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"
    >
<EditText android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:hint="Inserisci"
    />
</LinearLayout>
```



## AutoCompleteTextView

Il controllo AutoCompleteTextView è semplicemente una EditText con delle possibili parole di default che possono essere digitate.

Quando iniziamo a scrivere il controllo mostrerà le possibili parole che possono essere scelte in una lista.

Attenzione, con possibili parole non significa che possiamo inserire solo quelle, ma tutte.

La particolarità di questo componente è che l'intellinsense, ovvero la lista che viene visualizzata quando digitiamo, funziona solo una volta, ovvero verrà visualizzata solo per una parola.

```
<?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"
    >
<AutoCompleteTextView android:id="@+id/autotxt1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    />
</LinearLayout>
```

Oltre al layout bisogna caricare via codice le parole:

```
AutoCompleteTextView actv = (AutoCompleteTextView)
    this.findViewById(R.id.autotxt1);
ArrayAdapter<String> aa = new ArrayAdapter<String>(this,
    android.R.layout.simple_dropdown_item_1line, new String[] {"Inglese",
    "Spagnolo", "Tedesco", "Italiano"});
actv.setAdapter(aa);
```

Il codice è abbastanza semplice: viene ricavato il componente tramite l'id definito nel layout, e viene utilizzato un ArrayAdapter per inserire le parole.

Un ArrayAdapter è una specie di vettore, nel caso nostro tipizzato a String, dove le stringhe passate come parametri vengono adattate ad elementi del componente identificato, in questo caso `simple_dropdown_item_1line`.



## MultiAutoCompleteTextView

La MultiAutoCompleteTextView è praticamente una AutoCompleteTextView con l'aggiunta che l'intellinsense opera su più parole separate da un carattere speciale. Quindi la lista delle parole verrà visualizzata più volte e non una sola come nell'AutoCompleteTextView.

```
<?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"
    >
<MultiAutoCompleteTextView android:id="@+id/multiautotxt1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    />
</LinearLayout>
```

Nel caso dell'esempio, si nota nell'ultima riga che il separatore utilizzato è la virgola (comma).

```
MultiAutoCompleteTextView mactv =
    (MultiAutoCompleteTextView) this.findViewById(R.id.multiautotxt1);
ArrayAdapter<String> aa = new ArrayAdapter<String>(this,
    android.R.layout.simple_dropdown_item_1line, new String[] {"Inglese",
    "Spagnolo", "Tedesco", "Francese", "Italiano"});
mactv.setAdapter(aa);
mactv.setTokenizer(new MultiAutoCompleteTextView.CommaTokenizer());
```



## Button

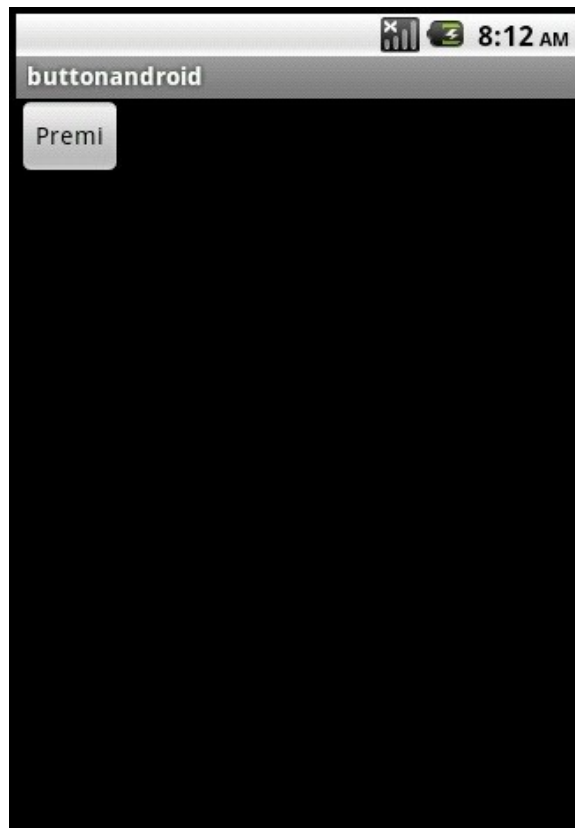
Uno dei più classici dei componenti è il pulsante, utilizzato spesso per dare via a delle operazioni.

```
<?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"
    >
<Button android:id="@+id/btnprova"
    android:text="Premi"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    />
</LinearLayout>
```

Mostriamo l'utilizzo pratico del suo evento onClick generato alla pressione del pulsante.

```
Button btnvis = (Button) this.findViewById(R.id.btnprova);
btnvis.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View arg0) {

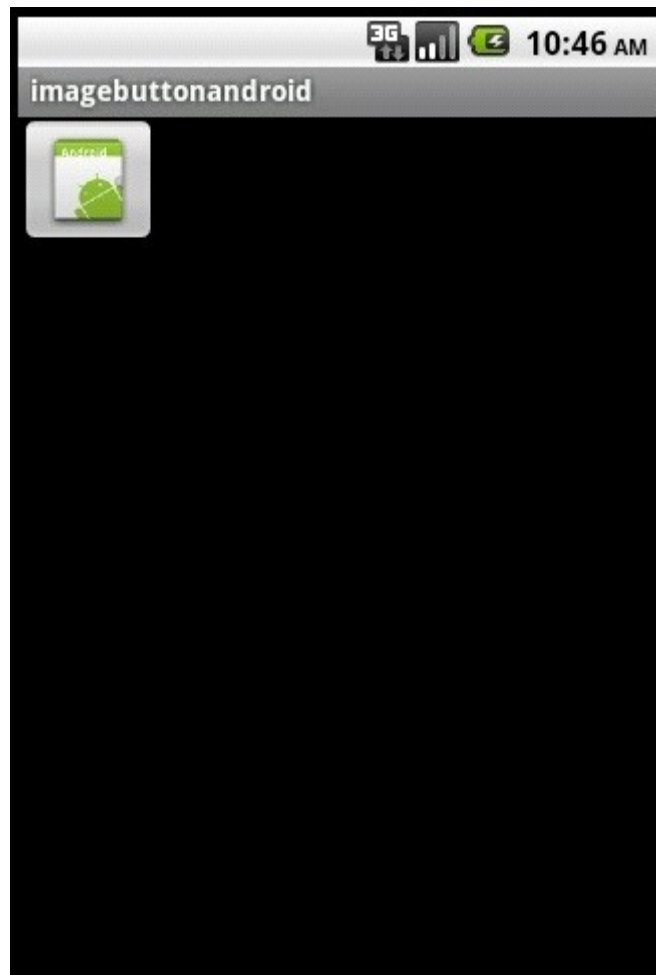
    }
});
```



## ImageButton

L'ImageButton non è altro che un Button, sia in comportamento sia come struttura, ma con una immagine di sfondo.

```
<?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"
    >
<ImageButton android:id="@+id/btnimg1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Clicca"
    android:src="@drawable/icon"
    />
</LinearLayout>
```



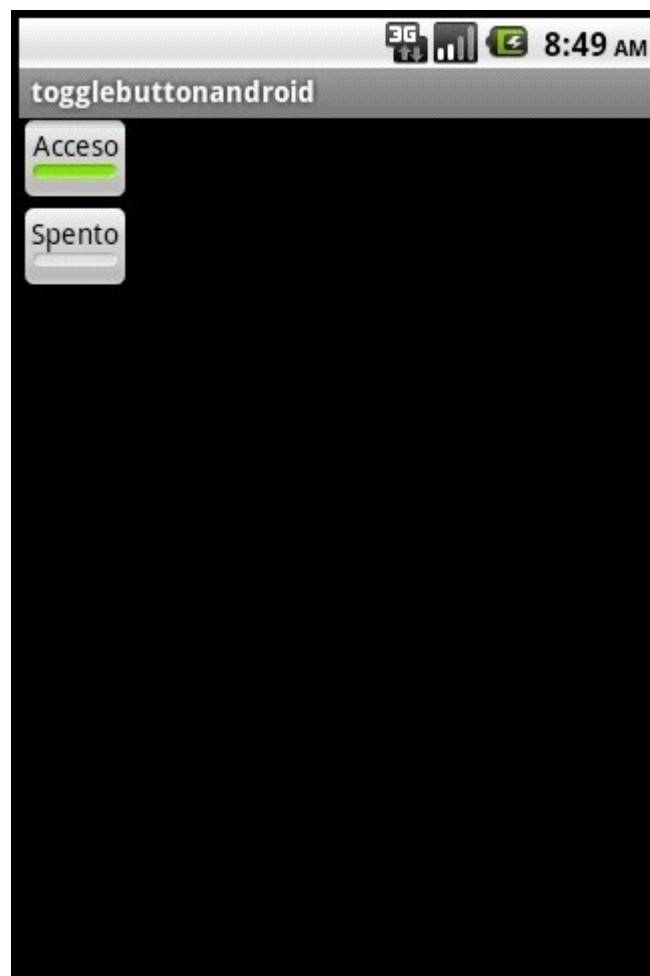
## ToggleButton

Il ToggleButton è il classico pulsante di On/Off.

Visivamente è un Button con un led sotto il testo che ne evidenzia lo stato.

```
<?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"
    >
<ToggleButton android:id="@+id/btntoogle1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textOn="Acceso"
    android:textOff="Spento"
    />

<ToggleButton android:id="@+id/btntoogle2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textOn="Acceso"
    android:textOff="Spento"
    />
</LinearLayout>
```



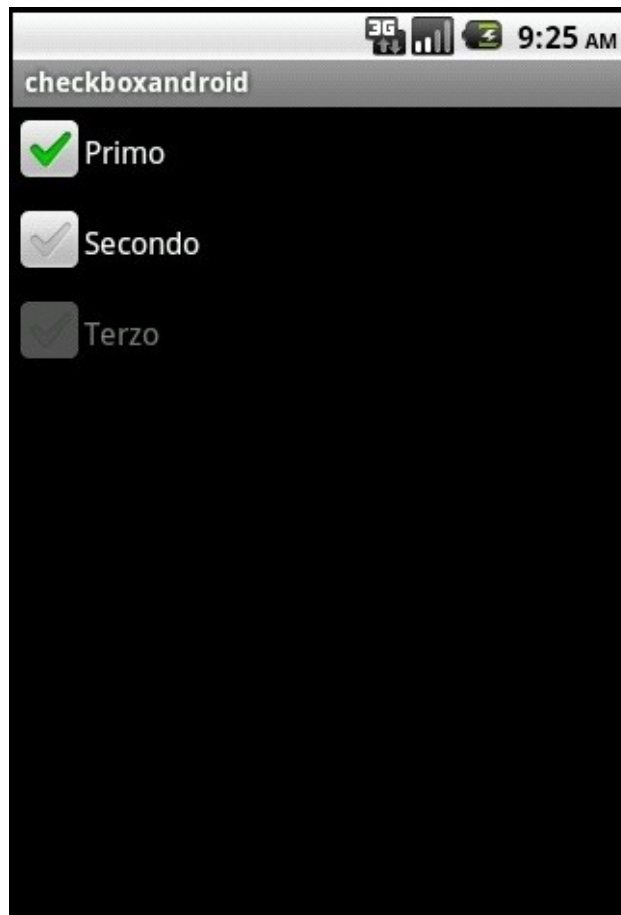
## CheckBox

Una CheckBox è il classico componente di spunta, utilizzato spesso da selettore per opzioni.

```
<?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"
    >
<CheckBox android:text="Primo"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:checked="true"
    />

<CheckBox android:text="Secondo"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:checked="false"
    />

<CheckBox android:text="Terzo"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:enabled="false"
    />
</LinearLayout>
```



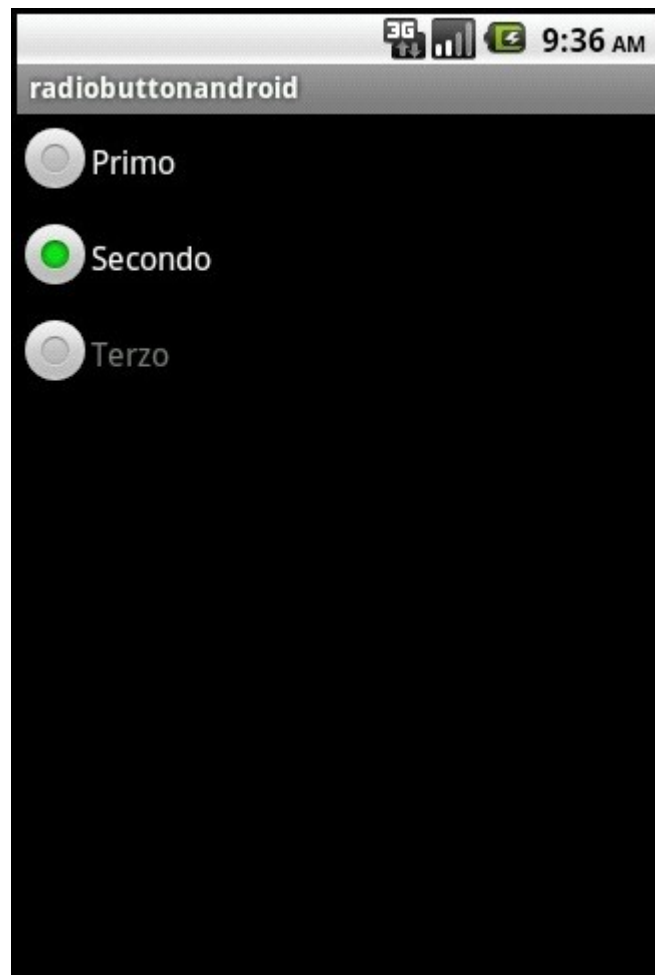
## RadioButton

La RadioButton (pulsante rotondo), come la CheckBox, permette di scegliere delle opzioni con la particolarità che sono esclusive.

In altre parole, si inseriscono una serie di RadioButton in un componente RadioGroup (gruppi di RadioButton) e se ne potrà scegliere solo una delle varie disponibili.

```
<?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"
    >
<RadioGroup android:id="@+id/rdbgroup1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="vertical" >
    <RadioButton android:id="@+id/rdb1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Primo"
        android:checked="true"
        />
    <RadioButton android:id="@+id/rdb2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Secondo"
        />
    <RadioButton android:id="@+id/rdb3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Terzo"
        android:enabled="false"
        />
</RadioGroup>
</LinearLayout>
```

Le proprietà utilizzate nell'esempio sono `enabled` per l'attivazione e disattivazione del componente e `checked` per il RadioButton selezionato di default.



## ListView

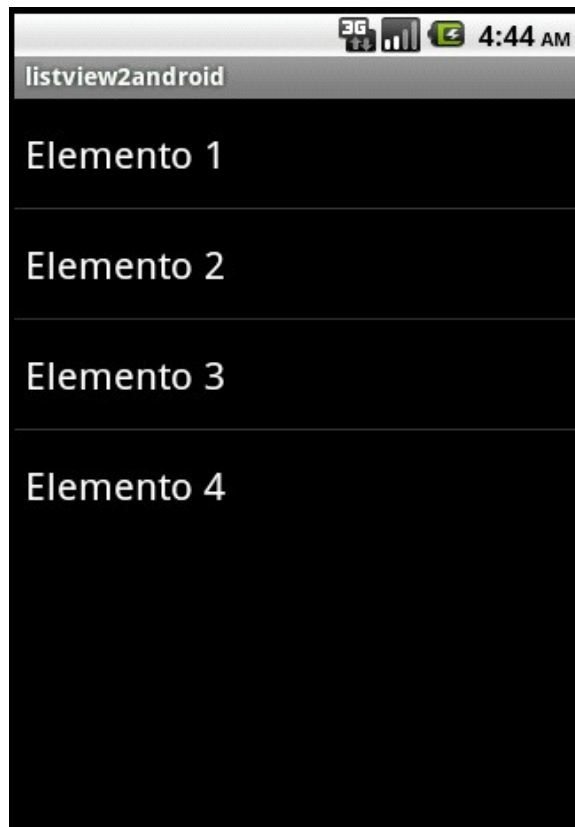
Una ListView è il basilare componente per la visualizzazione di una lista di elementi.

```
<?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"
    >
<ListView android:id="@+id/listview1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    />
</LinearLayout>
```

Qui al posto di un ArrayAdapter utilizziamo un ListAdapter che si adatta meglio al componente ListView.

```
String[] cols = new String[]{"Elemento 1", "Elemento 2", "Elemento 3",
    "Elemento 4"};
ListView list1 = (ListView) this.findViewById(R.id.listview1);
ListAdapter adapter = new ArrayAdapter<String>(this,
    android.R.layout.simple_list_item_1, cols);
list1.setAdapter(adapter);
```

Il codice è molto semplice e, inoltre, è possibile caricare gli elementi anche da risorse. Vedremo più avanti come.



## GridView

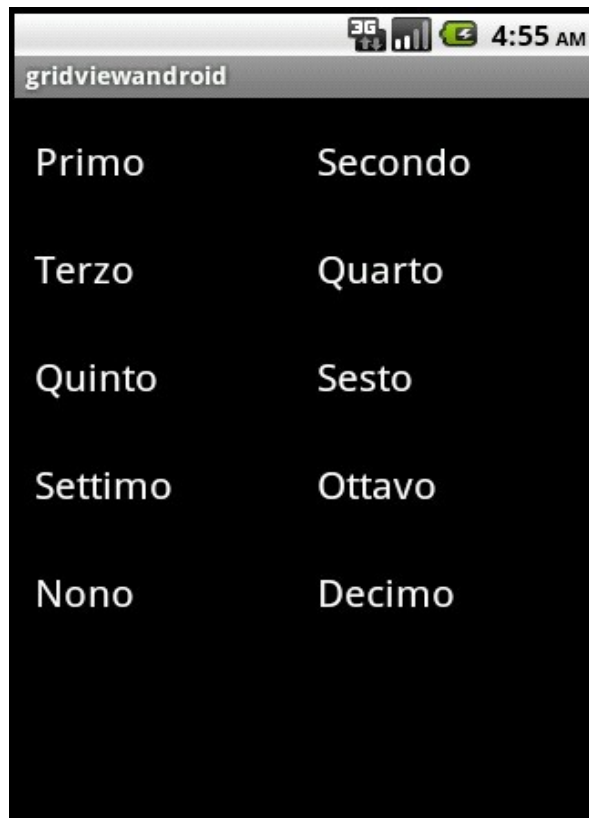
Come suggerisce il nome, una GridView è una griglia di elementi visualizzati a video.

```
<?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"
    >
<GridView android:id="@+id/grid1"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:numColumns="auto_fit"
    android:gravity="center"
    />
</LinearLayout>
```

La proprietà numColumns definisce il numero di colonne in cui deve essere divisa la griglia. Per semplicità, nell'esempio, è stato impostato ad auto\_fit in modo da dividere in automatico le colonne secondo il numero di elementi.

Come per la ListView abbiamo utilizzato un ListAdapter per definire gli elementi della griglia.

```
GridView grid = (GridView) this.findViewById(R.id.grid1);
String[] cols = new String[] {"Primo", "Secondo", "Terzo", "Quarto", "Quinto",
    "Sesto", "Settimo", "Ottavo", "Nono", "Decimo"};
ListAdapter adapter = new ArrayAdapter<String>( this,
    android.R.layout.simple_list_item_1, cols);
grid.setAdapter(adapter);
```



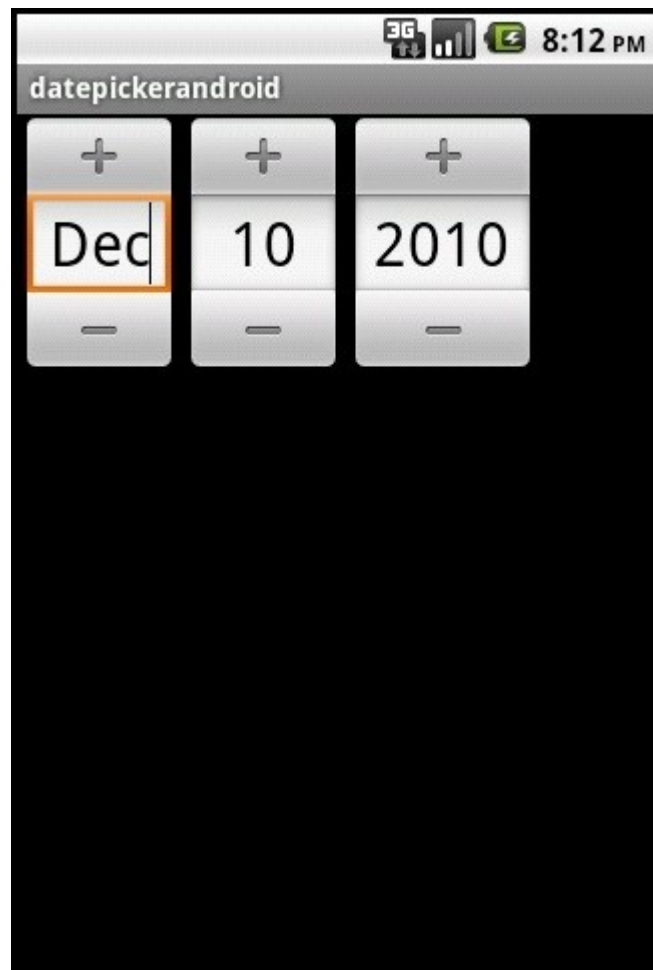
## DatePicker

Il componente DatePicker viene utilizzato per modificare una data.

```
<?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"
    >
<DatePicker android:id="@+id/datePicker1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
</LinearLayout>
```

Con il metodo `init` del componente possiamo impostare la data che il componente deve visualizzare per poi essere modificata.

```
DatePicker dp = (DatePicker) this.findViewById(R.id.datePicker1);
dp.init(2010, 11, 10, null);
```



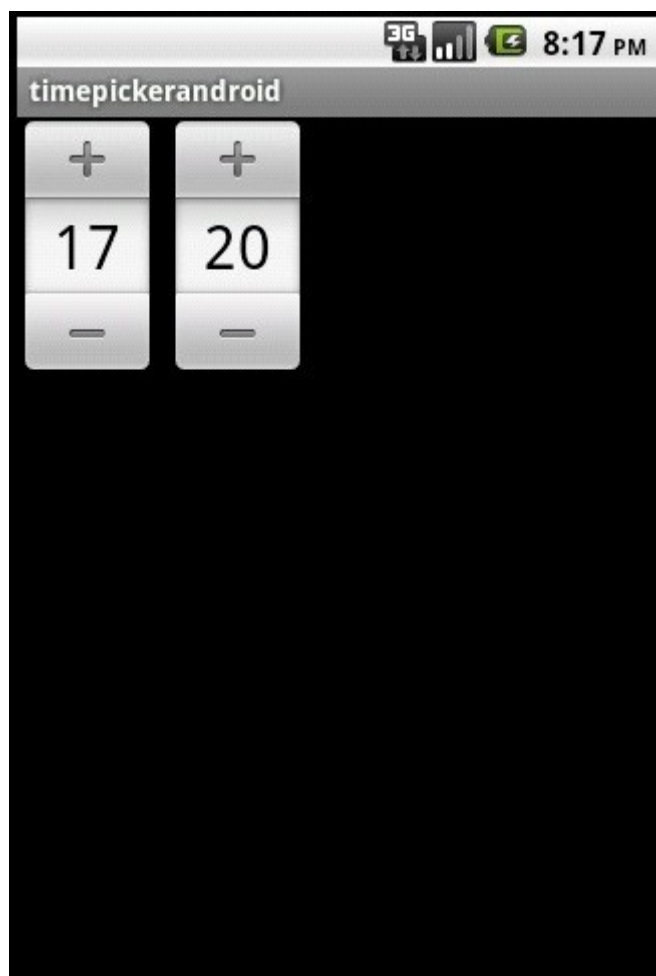
## TimePicker

Il controllo TimePicker ha lo stesso scopo di DatePicker, solo che cambia il tempo e non la data.

```
<?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"
    >
<TimePicker android:id="@+id/timePicker1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
</LinearLayout>
```

Come per il DatePicker, è possibile impostare il tempo di partenza da cui poi poter fare la modifica.

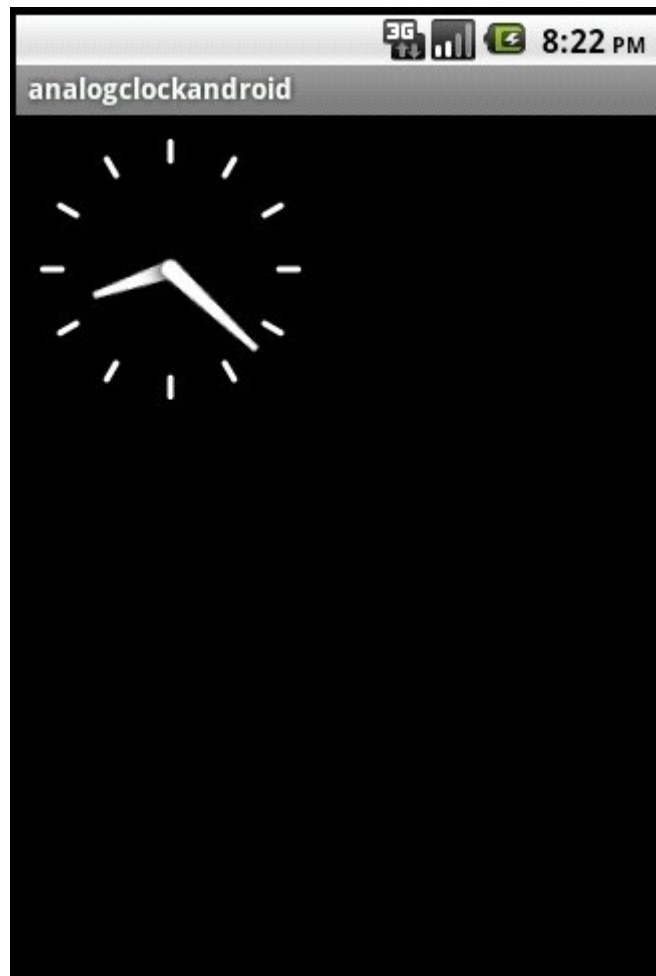
```
TimePicker timep = (TimePicker) this.findViewById(R.id.timePicker1);
timep.setIs24HourView(true);
timep.setCurrentHour(17);
timep.setCurrentMinute(20);
```



## AnalogClock

Orologio analogico per visualizzare l'orario.

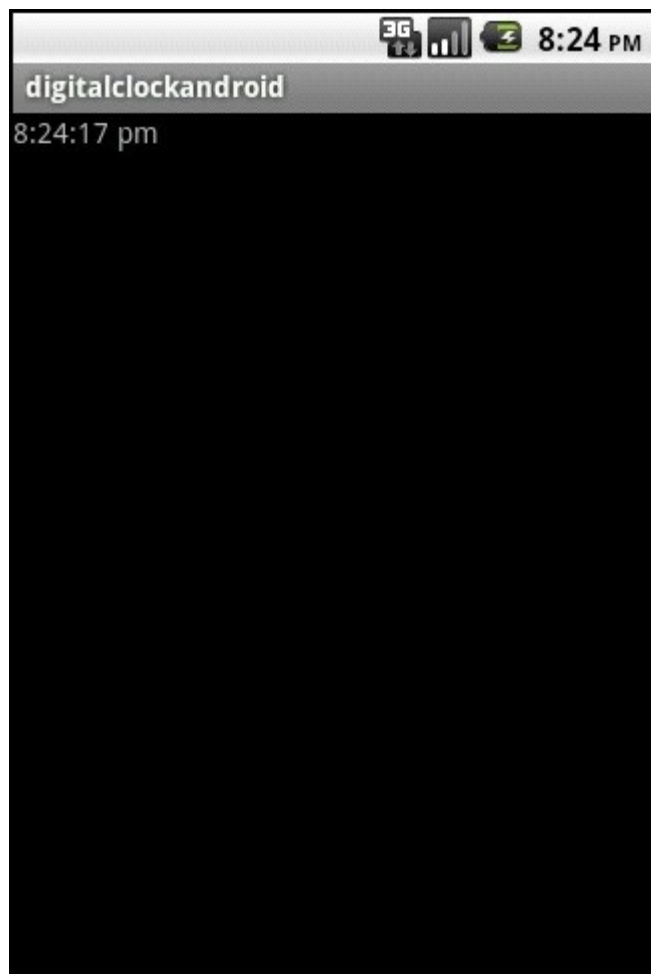
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<AnalogClock android:id="@+id/analog1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
</LinearLayout>
```



## DigitalClock

Orologio digitale per visualizzare l'orario.

```
<?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"
    >
<DigitalClock android:id="@+id/digit1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    />
</LinearLayout>
```



# Altri Controlli

- `CheckedTextView`.....28
- `Chronometer`.....29
- `ImageView`.....31
- `ProgressBar`.....32
- `RatingBar`.....33
- `SeekBar`.....34
- `Toast`.....35

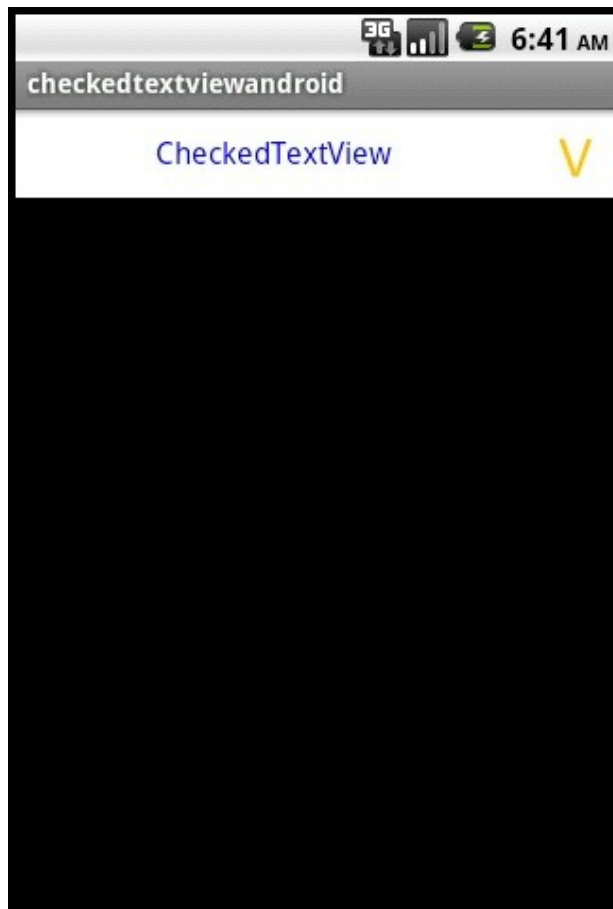
## CheckedTextView

Una CheckedTextView non è altro che una TextView con un'immagine che fa da spunta.

```
<?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"
    >
<CheckedTextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="CheckedTextView"
    android:background="#FFFFFF"
    android:textColor="#0000FF"
    android:gravity="center"
    android:checked="true"
    android:checkMark="@drawable/immagine"
    />
</LinearLayout>
```

Nel nostro esempio abbiamo utilizzato la proprietà checkMark per identificare l'immagine, ricavata dalle risorse.

La proprietà gravity è utilizzata per allineare il testo e la proprietà checked è utilizzata per visualizzare o meno l'immagine di spunta.



## Chronometer

Il Chronometer è il classico cronometro digitale, poco utile a prima vista, ma per applicazioni specifiche può essere un valido componente.

```
<?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"
    >
<Chronometer android:id="@+id/chr1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    />
<Button android:id="@+id/btnstart"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Start"
    />
<Button android:id="@+id/btnstop"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Stop"
    />
</LinearLayout>
```

Nell'esempio abbiamo utilizzato due pulsanti start e stop per iniziare e fermare il cronometro.

```
private Chronometer cr1;
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    cr1 = (Chronometer) this.findViewById(R.id.chr1);
    Button btnstart = (Button) this.findViewById(R.id.btnstart);
    btnstart.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View arg0) {
            cr1.start();
        }
    });
    Button btnstop = (Button) this.findViewById(R.id.btnstop);
    btnstop.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View arg0) {
            cr1.stop();
        }
    });
}
```

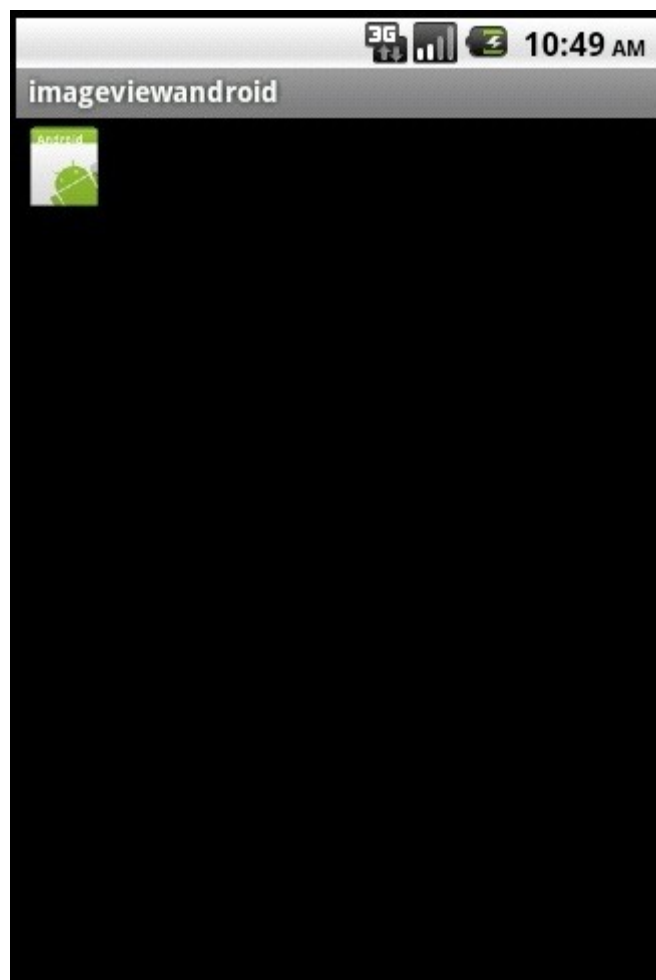


## ImageView

ImageView è il componente per la visualizzazione di immagini.

```
<?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"
    >
<ImageView android:src="@drawable/icon"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    />
</LinearLayout>
```

Tramite la proprietà src possiamo definire il percorso dell'immagine, in questo caso è stata presa dalle risorse.

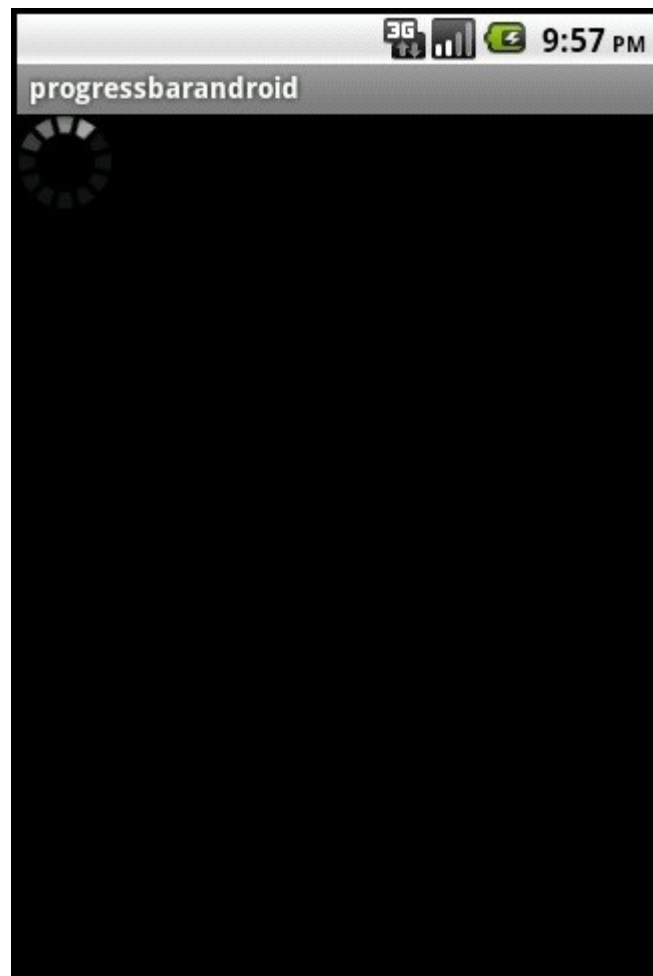


## ProgressBar

La ProgressBar, ovvero barra di progresso, è un componente utilizzato per visualizzare lo stato di un'operazione.

```
<?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"
    >
<ProgressBar
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:max="100"
    android:progress="95"
    />
</LinearLayout>
```

Tramite le proprietà max e progress definiamo il valore massimo che può raggiungere la ProgressBar ed il valore di default.



## RatingBar

La RatingBar è la consueta barra di gradimento, utilizzata molto per votazioni di video o immagini e rappresentata dalle classiche stelle.

```
<?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"
    >
<RatingBar
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:max="100"
    android:rating="65"
    />
</LinearLayout>
```

Anche qui è presente la proprietà max che identifica il massimo valore che può assumere la votazione.

La proprietà rating serve per impostare il valore di default.

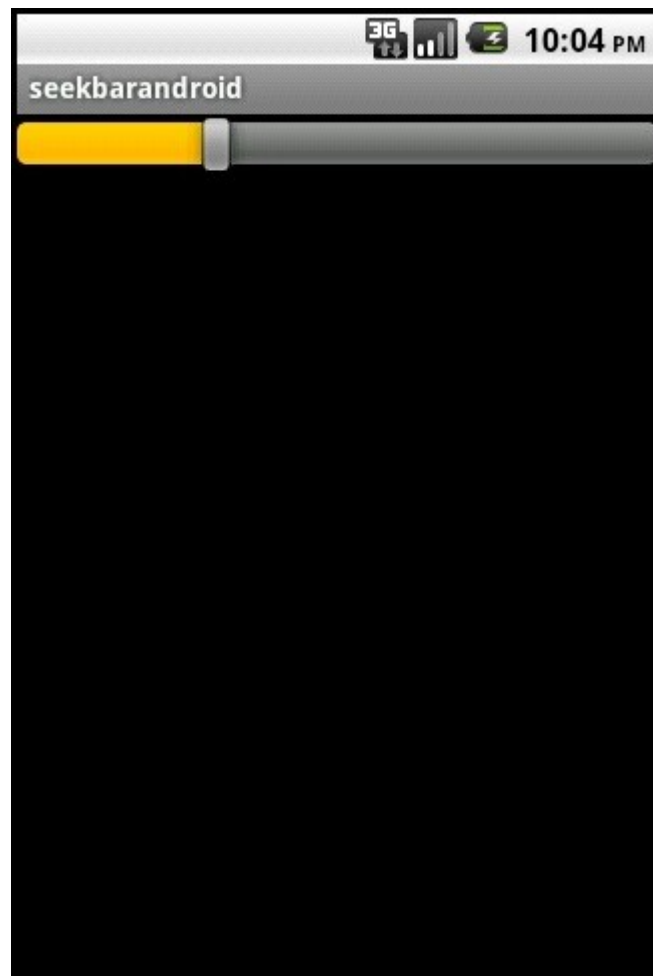


## SeekBar

La SeekBar è a tutti gli effetti una ProgressBar che può essere manovrata: possiede un cursore che è possibile spostare per modificare il valore del controllo.

```
<?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"
    >
<SeekBar
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:max="100"
    android:progress="37"
    />
</LinearLayout>
```

Gli attributi utilizzati sono gli stessi della ProgressBar, max e progress, per impostare il valore massimo ed il valore di default.



## Toast

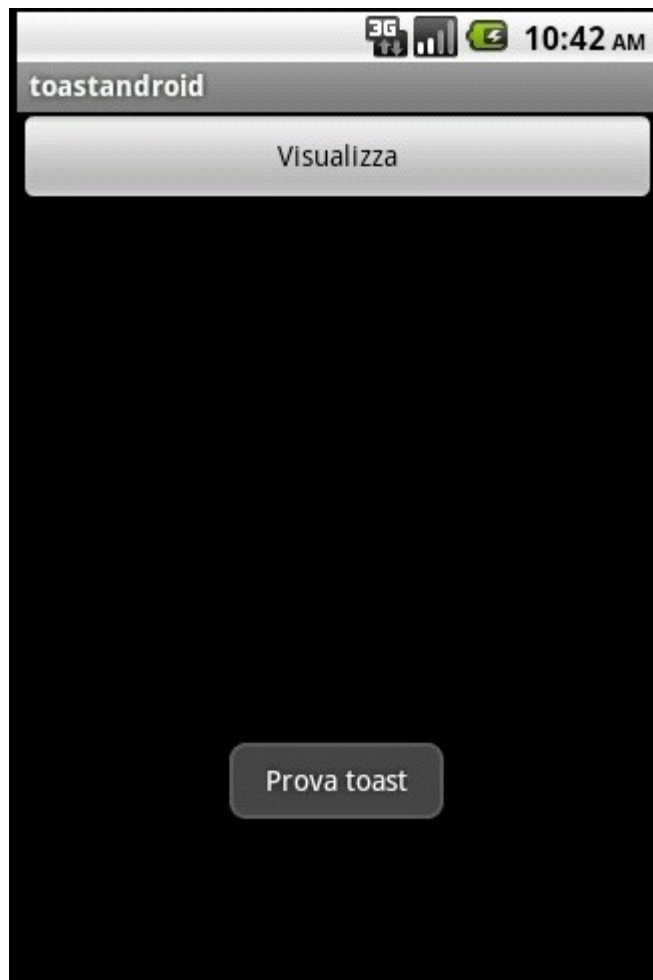
Il controllo Toast è un piccolo contenitore che visualizza un messaggio. Adoperato spesso per le notifiche e per varie segnalazioni di errori.

```
<?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"
    >
<Button        android:id="@+id/btnvisual"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Visualizza"
    />
</LinearLayout>
```

Nell'esempio abbiamo utilizzato un pulsante per visualizzare il controllo Toast.

```
Button btnvis = (Button) this.findViewById(R.id.btnvisual);
btnvis.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View arg0) {
        Toast t = Toast.makeText(toastandroid.this, "Prova toast",
Toast.LENGTH_LONG);
        t.show();
    }
});
```

Come si nota dall'esempio, il componente viene creato attraverso il metodo statico `makeText`, passando come parametri il contesto dove visualizzare, il messaggio e la lunghezza ( `Toast.LENGTH_LONG` o `Toast.LENGTH_SHORT`).



# Controlli Avanzati

- Spinner.....38
- TabWidget.....40
- WebView.....42
- MapView.....44
- PopupWindow.....46
- ExpandableListView.....49

## Spinner

Lo Spinner è un controllo per visualizzare e scegliere una opzione su un insieme.

E' composto da due parti: la prima è una semplice barra con la scelta corrente, la seconda è la lista delle opzioni disponibili.

```
<?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"
    >
<Spinner
    android:id="@+id/spinner"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:prompt="Scegli il nome"
    />
</LinearLayout>
```

Con la proprietà `prompt` andiamo a settare il messaggio che comparirà nella lista di elementi. Serve a far capire all'utente il perchè deve effettuare una scelta.

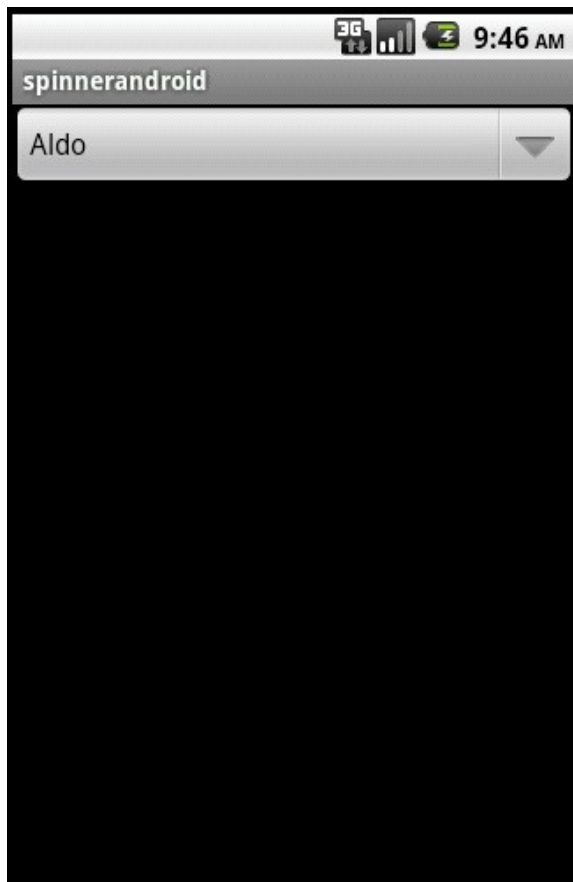
Nell'esempio abbiamo utilizzato una lista di nomi memorizzata nelle risorse (file `strings.xml`) che successivamente verranno caricate nel metodo `onCreate` dell'applicazione.

```
<string-array name="nomi">
    <item>Alberto</item>
    <item>Aldo</item>
    <item>Giovanni</item>
    <item>Stefano</item>
    <item>Luca</item>
    <item>Giuseppe</item>
    <item>Fabrizio</item>
</string-array>
```

Come per una `ListView`, utilizziamo un `ArrayAdapter` per caricare gli elementi.

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

    Spinner s = (Spinner) findViewById(R.id.spinner);
    ArrayAdapter adapter = ArrayAdapter.createFromResource(this,
R.array.nomi, android.R.layout.simple_spinner_item);
    adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown
_item);
    s.setAdapter(adapter);
}
```



## TabWidget

Il controllo TabWidget permette la visualizzazione di più contesti visuali in un unico contenitore. E' composto da un insieme di pagine, ognuna delle quali specifica un contesto visuale.

```
<?xml version="1.0" encoding="utf-8"?>
<TabHost xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@android:id/tabhost"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <LinearLayout
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent">
        <TabWidget
            android:id="@android:id/tabs"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content" />
        <FrameLayout
            android:id="@android:id/tabcontent"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent">
            <TextView
                android:id="@+id/textview1"
                android:layout_width="fill_parent"
                android:layout_height="fill_parent"
                android:text="Tab 1" />
            <TextView
                android:id="@+id/textview2"
                android:layout_width="fill_parent"
                android:layout_height="fill_parent"
                android:text="Tab 2" />
            <TextView
                android:id="@+id/textview3"
                android:layout_width="fill_parent"
                android:layout_height="fill_parent"
                android:text="Tab 3" />
        </FrameLayout>
    </LinearLayout>
</TabHost>
```

Il controllo TabHost serve da contenitore per il TabWidget.

Nell'esempio proposto utilizziamo tre TextView, ognuna delle quali sarà predisposta per pagina.

```

import android.app.TabActivity;
import android.os.Bundle;
import android.widget.TabHost;

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

        TabHost mTabHost = getTabHost();

        mTabHost.addTab(mTabHost.newTabSpec("tab_test1").setIndicator("TAB
1").setContent(R.id.textview1));
        mTabHost.addTab(mTabHost.newTabSpec("tab_test2").setIndicator("TAB
2").setContent(R.id.textview2));
        mTabHost.addTab(mTabHost.newTabSpec("tab_test3").setIndicator("TAB
3").setContent(R.id.textview3));

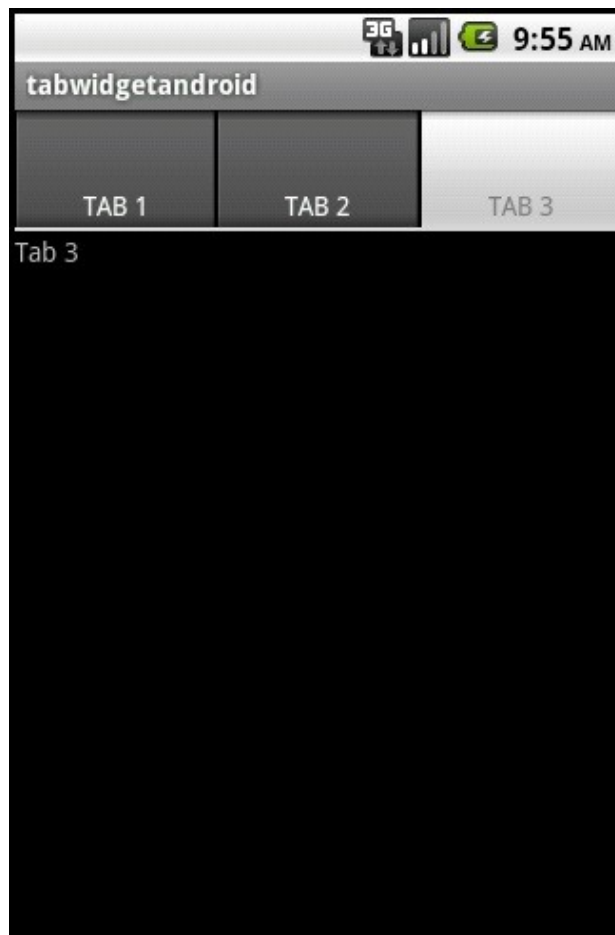
        mTabHost.setCurrentTab(0);
    }
}

```

E' chiaro che quando l'applicazione verrà creata, dobbiamo aggiungere manualmente le varie pagine ed impostarne il contesto di visualizzazione.

Nell'esempio si vede che vengono impostate le tre TextView per le tre pagine.

Alla fine viene selezionata la pagina predefinita.



## WebView

Componete non di poco conto è il WebView, un vero e proprio browser all'interno della nostra applicazione.

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

L'uso è semplice e non dobbiamo dimenticare di inserire il permesso per poter utilizzare la connessione ad internet.

```
<uses-permission android:name="android.permission.INTERNET"></uses-permission>
```

Nell'esempio mostriamo come visualizzare la pagina principale di Google.

```
import android.app.Activity;
import android.os.Bundle;
import android.webkit.WebView;

public class webviewandroid extends Activity {
    WebView webview;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        webview = (WebView) findViewById(R.id.webview);
        webview.getSettings().setJavaScriptEnabled(true);
        webview.loadUrl("http://www.google.com");
    }
}
```

Da notare come possiamo attivare e disattivare Javascript con il metodo `setJavaScriptEnabled` ed impostare la pagina con il metodo `loadUrl`.



## MapView

Controllo basilare per la visualizzazione delle mappe di Google.

È possibile attivare anche la modalità StreetView e varie opzioni, come lo zoom.

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

    <com.google.android.maps.MapView
        android:id="@+id/mapview1"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:enabled="true"
        android:clickable="true"
        android:apiKey="tuo codice apikey google"
    />
</RelativeLayout>
```

Per poter utilizzare il controllo MapView, dobbiamo importare la libreria nel file Manifest.xml e permettere la connessione ad internet.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="mapview.android"
    android:versionCode="1"
    android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <uses-library android:name="com.google.android.maps" />
        <activity android:name=".mapviewandroid"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
    <uses-permission android:name="android.permission.INTERNET" />
</manifest>
```

Le righe incriminate per l'uso della libreria e per i permessi alla connessione sono

```
<uses-library android:name="com.google.android.maps" />
<uses-permission android:name="android.permission.INTERNET" />
```

Nell'esempio, abbiamo utilizzato le coordinate della città di Palermo, disattivato la modalità StreetView e attivato la modalità Satellite.

Da notare che lo zoom è impostato a 12, che permette una visuale abbastanza gradevole di buona parte della provincia di Palermo.

Il metodo isRouteDisplayed impostato a true permette la visualizzazione delle direzioni di guida.

```

public class mapviewandroid extends MapActivity {
    private MapView map;
    private MapController controller;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        map = (MapView) this.findViewById(R.id.mapview1);
        controller = map.getController();
        map.setSatellite(true);
        map.setStreetView(false);
        map.displayZoomControls(true);
        controller.setZoom(12);
        Double lat = 38.115619*1E6;
        Double lng = 13.361376*1E6;
        GeoPoint point = new GeoPoint(lat.intValue(), lng.intValue());
        controller.setCenter(point);
        controller.animateTo(point);
    }

    @Override
    protected boolean isRouteDisplayed() {
        return false;
    }
}

```



## PopupWindow

Il PopupWindow è un oggetto contenitore per la visualizzazione di controlli a video su un livello differente rispetto alla vista principale.

Utilizzata spesso per visualizzazione di messaggi o da input per attributi.

Nell'esempio, naturalmente, abbiamo utilizzato due layout diversi per la vista principale e per la PopupWindow.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/main"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<Button android:id="@+id/btnvis"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Visualizza"
    android:gravity="center"
    />
<Button android:id="@+id/btnch"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Chiudi"
    android:gravity="center"
    />
</LinearLayout>
```

Di seguito il layout del PopupWindow.

```
<?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"
    android:background="#0066FF"
    >
<TextView android:id="@+id/txtpop"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Popup Window"
    android:textColor="#0000FF"
    />
</LinearLayout>
```

Il codice è abbastanza semplice e vengono eseguiti i controlli su due pulsanti per la visualizzazione e la chiusura del PopupWindow.

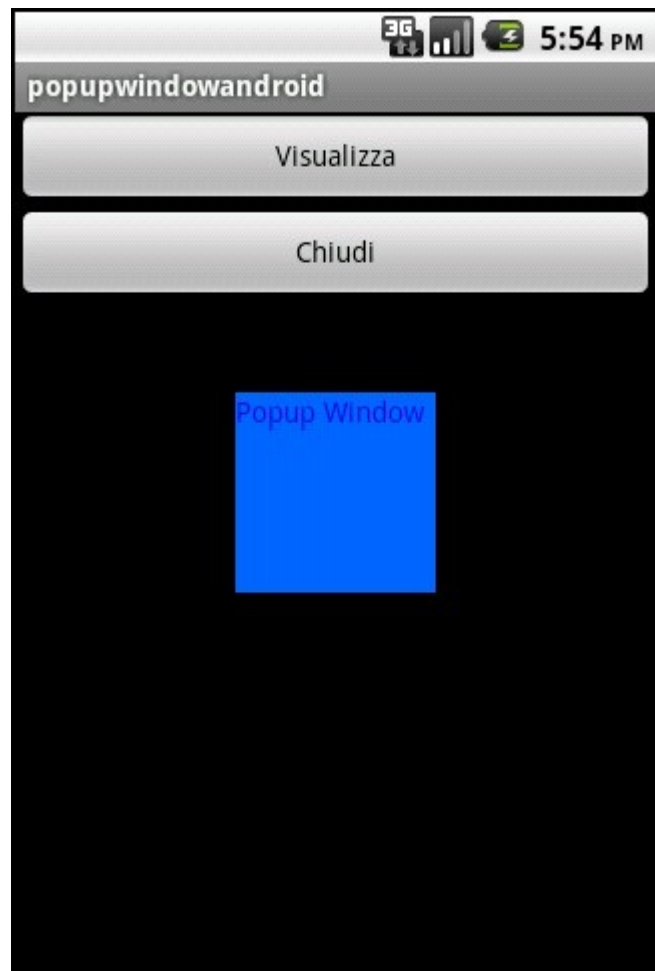
```

public class popupwindowandroid extends Activity {
    LinearLayout layout;
    LayoutInflater inflater;
    PopupWindow pw = null;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button btn = (Button) this.findViewById(R.id.btnvis);
        // Ricaviamo il layout corrente che servirà da base per a
visualizzazione del Popup
        layout = new LinearLayout(this);
        inflater =
(LayoutInflater) this.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
        btn.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View arg0) {
                if(pw == null || !pw.isShowing())
                {
                    pw = new
PopupWindow(inflater.inflate(R.layout.popupw, null, true), 100, 100, true);
                    pw.setFocusable(false);
                    // Verrà visualizzato al centro della vista di
base
                    pw.showAtLocation(layout, Gravity.CENTER, 0, 0);
                    pw.update();
                }
            }
        });
        Button btn2 = (Button) this.findViewById(R.id.btnch);
        btn2.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View arg0) {
                if(pw != null && pw.isShowing())
                {
                    // Chiusura Popup
                    pw.dismiss();
                }
            }
        });
    }
}

```

I passi per la visualizzazione di un PopupWindow sono semplici: ricavo del contesto corrente, creazione PopupWindow e visualizzazione.



## ExpandableListView

Una ExpandableListView è una ListView a due livelli.

Nel primo livello sono presenti i principali argomenti da rappresentare, nel secondo, invece, i vari elementi per ogni argomento.

```
<?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"
    >
<ExpandableListView android:id="@+id/explist1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    />
</LinearLayout>
```

Di seguito la classe di esempio:

```
public class expandablelistviewandroid extends ExpandableListActivity {
    ExpandableListAdapter mAdapter;

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

        // Set up our adapter
        mAdapter = new MyExpandableListAdapter();
        setListAdapter(mAdapter);
    }

    public class MyExpandableListAdapter extends BaseExpandableListAdapter {
        // Sample data set. children[i] contains the children (String[]) for
        groups[i].
        private String[] groups = { "Nomi di persona", "Nomi di cane", "Nomi di
        gatti", "Nomi di pesci" };
        private String[][] children = {
            { "Alberto", "Roberto", "Paolo" },
            { "Yuri", "Rocky" },
            { "Briciola", "Puccia" },
            { "Pallina" }
        };

        public Object getChild(int groupPosition, int childPosition) {
            return children[groupPosition][childPosition];
        }

        public long getChildId(int groupPosition, int childPosition) {
            return childPosition;
        }

        public int getChildrenCount(int groupPosition) {
            return children[groupPosition].length;
        }

        public TextView getGenericView() {
            // Layout parameters for the ExpandableListView
```

```

AbsListView.LayoutParams lp = new AbsListView.LayoutParams(
    ViewGroup.LayoutParams.MATCH_PARENT, 64);

TextView textView = new TextView(expandablelistviewandroid.this);
textView.setLayoutParams(lp);
// Center the text vertically
textView.setGravity(Gravity.CENTER_VERTICAL | Gravity.LEFT);
// Set the text starting position
textView.setPadding(36, 0, 0, 0);
return textView;
}

public View getChildView(int groupPosition, int childPosition, boolean
isLastChild,
    View convertView, ViewGroup parent) {
    TextView textView = getGenericView();
    textView.setText(getChild(groupPosition, childPosition).toString());
    return textView;
}

public Object getGroup(int groupPosition) {
    return groups[groupPosition];
}

public int getGroupCount() {
    return groups.length;
}

public long getGroupId(int groupPosition) {
    return groupPosition;
}

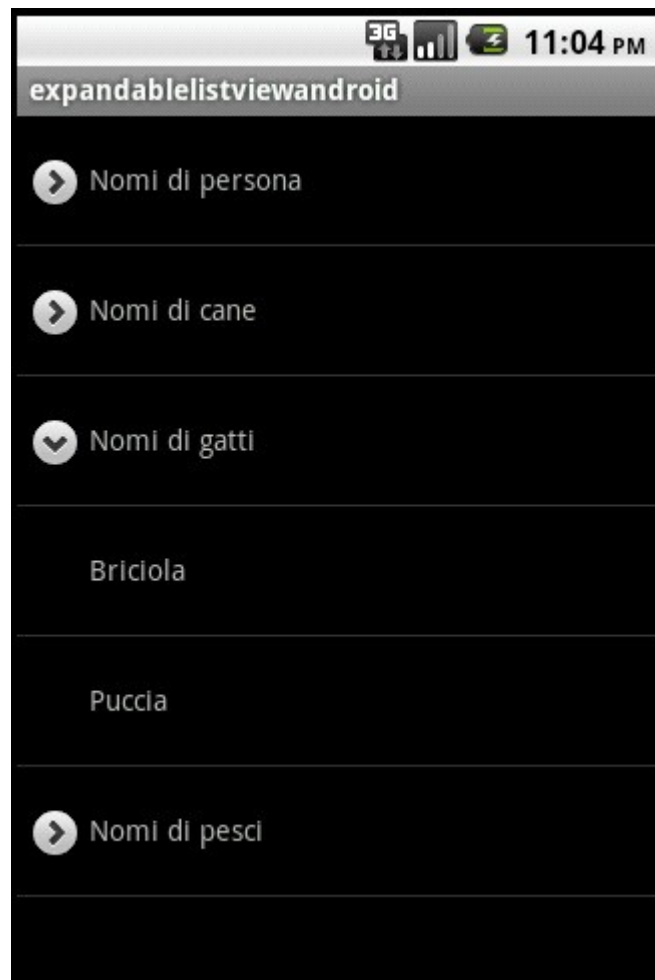
public View getGroupView(int groupPosition, boolean isExpanded, View
convertView,
    ViewGroup parent) {
    TextView textView = getGenericView();
    textView.setText(getGroup(groupPosition).toString());
    return textView;
}

public boolean isChildSelectable(int groupPosition, int childPosition) {
    return true;
}

public boolean hasStableIds() {
    return true;
}
}
}

```

La struttura della classe è abbastanza articolata: viene creato un nuovo oggetto di tipo `ExpandableListAdapter` e vengono scritti i metodi per ricavare la posizione degli elementi, numero elementi ed altri che possono risultare utili.



# Layout Manager

- LinearLayout..... 53
- TableLayout.....55
- RelativeLayout.....57
- AbsoluteLayout.....59
- FrameLayout.....61

## LinearLayout

Il LinearLayout è il layout manager più utilizzato e settato di default in un progetto Android con Eclipse.

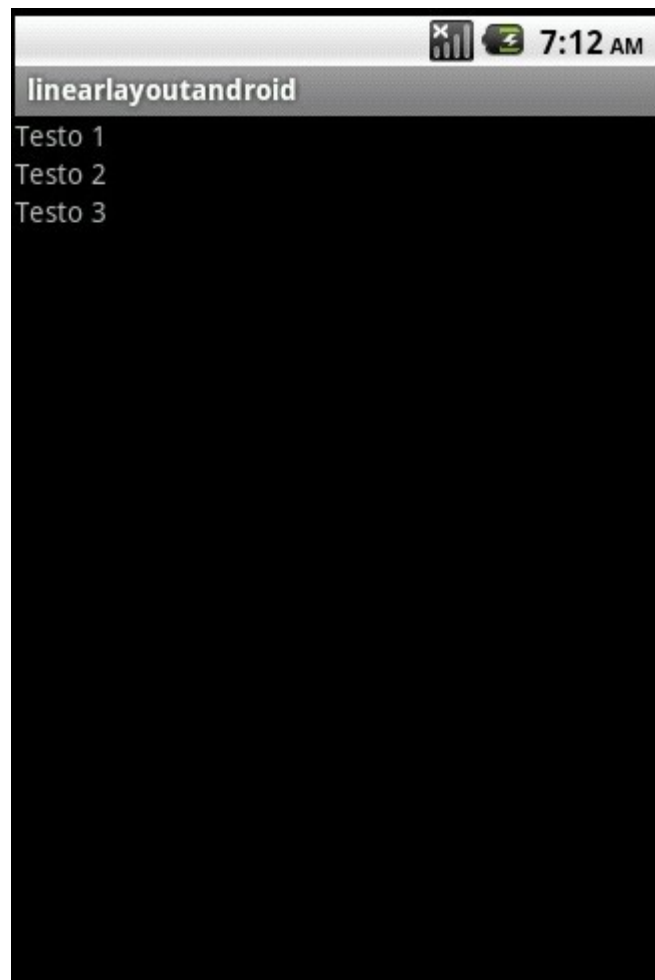
Dispone ogni elemento su ogni riga se l'attributo orientation è impostato su vertical e su colonne se è impostato su horizontal.

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

La prima parte di applicazione è abbastanza semplice e rappresenta il codice della classe Java per richiamare il layout.

Nell'esempio viene proposto un insieme di etichette di testo disposti con orientation vertical.

```
<?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:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Testo 1"
    />
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Testo 2"
    />
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Testo 3"
    />
</LinearLayout>
```



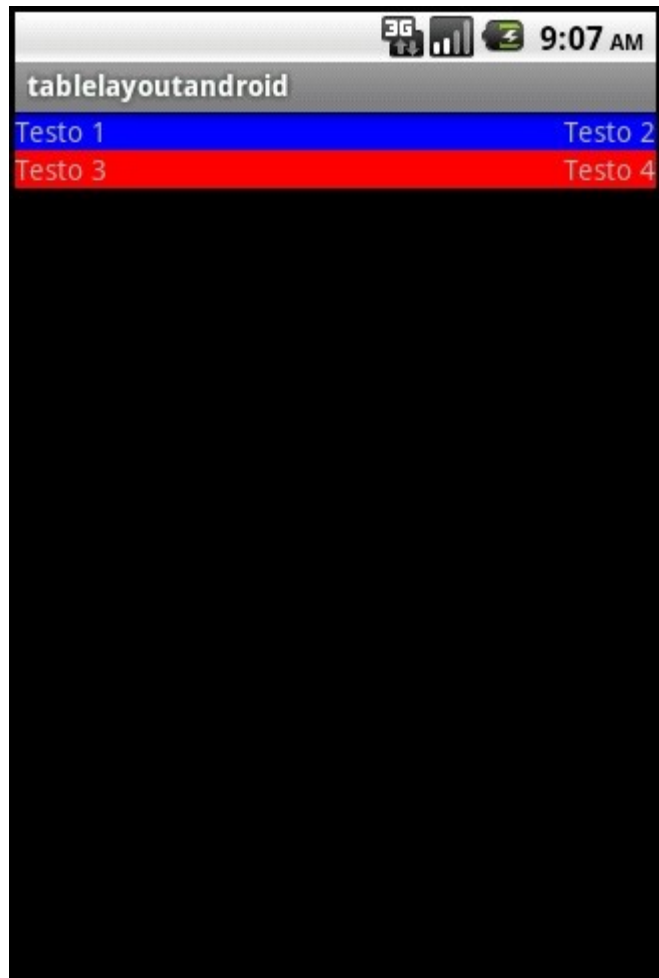
## TableLayout

Questo tipo di layout manager dispone i componenti in forma tabellare, dividendoli per riga.

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

Si può notare facilmente dal codice che oltre al contenitore TableLayout, viene utilizzato il contenitore TableRow, che, come dice il nome, rappresenta una riga.

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:stretchColumns="1"
    >
<TableRow android:background="#0000FF">
    <TextView
        android:layout_column="1"
        android:text="Testo 1"
        android:gravity="left"
    />
    <TextView
        android:text="Testo 2"
        android:gravity="right"
    />
</TableRow>
<TableRow android:background="#FF0000">
    <TextView
        android:layout_column="1"
        android:text="Testo 3"
        android:gravity="left"
    />
    <TextView
        android:text="Testo 4"
        android:gravity="right"
    />
</TableRow>
</TableLayout>
```



## RelativeLayout

Dalla stessa parola, possiamo capire il funzionamento di questo layout manager. Esso predispone i controlli rispetto ad un controllo precedentemente segnalato.

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

Notiamo subito che viene usato l'attributo `layout_below` per far capire a quale componente si deve fare riferimento per individuare la sua posizione.

In appoggio, si utilizzano gli attributi di tipo `margin`, come `marginLeft` e `marginTop` per spostare il controllo secondo il riferimento prefisso.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout 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/txt1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Testo 1"
    />
<TextView android:id="@+id/txt2"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Testo 2"
    android:layout_below="@id/txt1"
    android:layout_marginLeft="100px"
    />
<TextView android:id="@+id/txt3"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Testo 3"
    android:layout_below="@id/txt2"
    android:layout_marginTop="50px"
    />
</RelativeLayout>
```



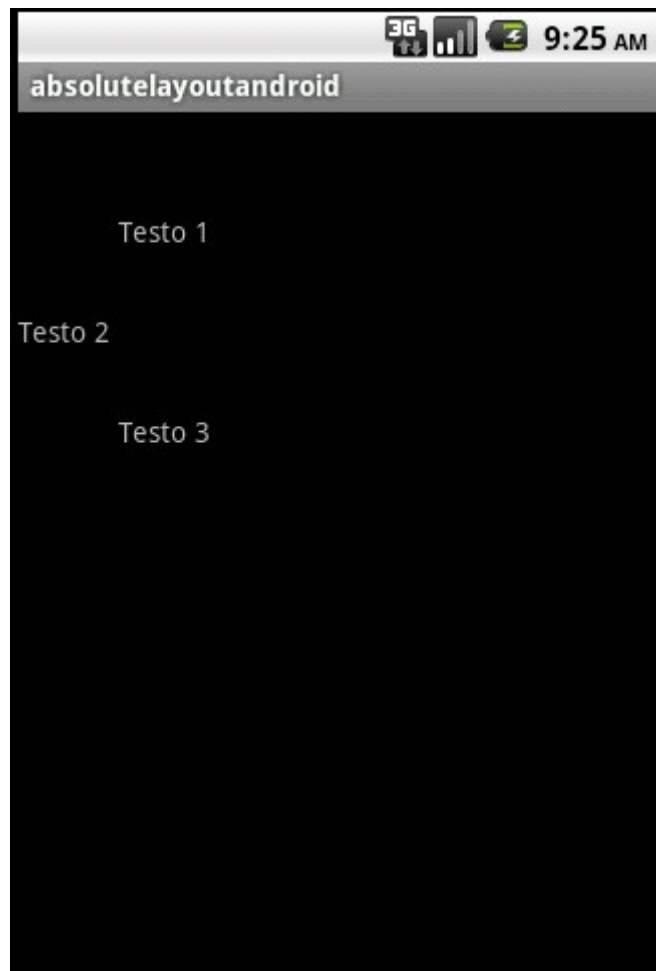
## AbsoluteLayout

Si può considerare un contenitore di comodo, in quanto, è utilizzato per tutti quei layout dove bisogna utilizzare misure precise al pixel.

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

Da notare, l'utilizzo degli attributi `layout_x` e `layout_y` che servono per decidere la posizione del componente rispetto all'angolo in alto a sinistra del display.

```
<?xml version="1.0" encoding="utf-8"?>
<AbsoluteLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Testo 1"
    android:layout_x="50px"
    android:layout_y="50px"
    />
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Testo 2"
    android:layout_x="0px"
    android:layout_y="100px"
    />
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Testo 3"
    android:layout_x="50px"
    android:layout_y="150px"
    />
</AbsoluteLayout>
```



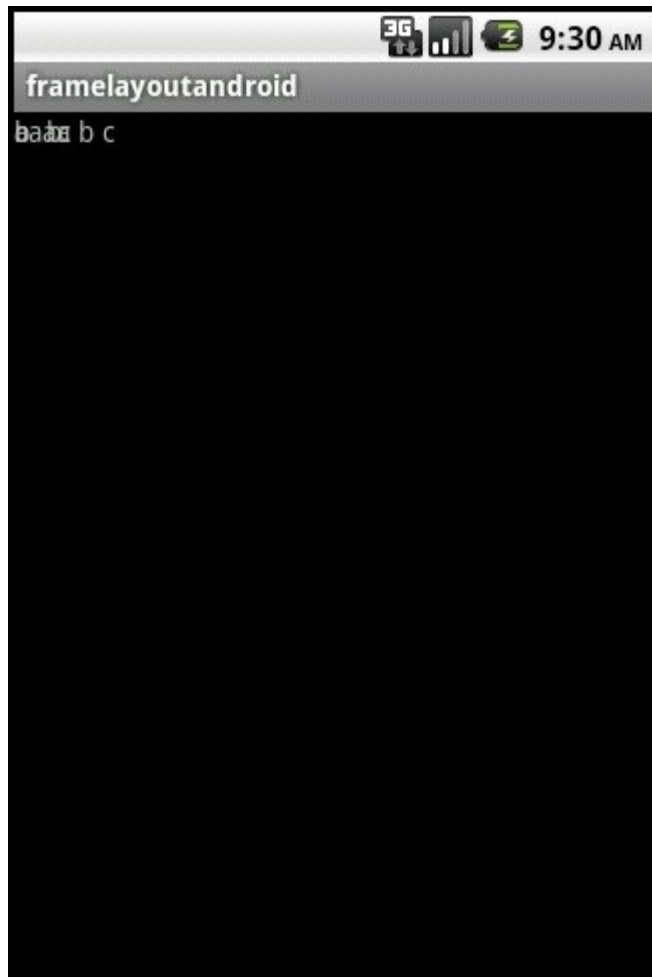
## FrameLayout

Questo tipo di layout manager dispone tutti gli elementi uno sopra l'altro. Utilizzato per creare piccoli popup o sovrapposizioni varie.

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

Nell'esempio visuale si nota che i vari campi delle TextView vengono disposti uno sopra l'altro rendendo il tutto illegibile.

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="aaaa "
    />
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="b b b"
    />
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="c c c"
    />
</FrameLayout>
```



# Widget

## Widget

I Widget sono dei componenti grafici che vengono visualizzati sulla home o su una delle sue pagine.

Utilizzati per mettere in rilievo informazioni importanti o comunicazioni di vario tipo, come il meteo, aggiornamenti mail, l'orologio o varie.

L'input è abbastanza limitato e i controlli che si possono utilizzare sono pochi, esclusi anche EditText et similia.

A che servono quindi? Beh, come elementi vetrina sono ottimi!

Vediamo un po' di codice:

```
import android.appwidget.AppWidgetProvider;

public class HelloWidget extends AppWidgetProvider {

}
```

Primissima differenza con un applicazione normale è la classe a cui si fa riferimento, invece della solita Activity, si utilizza AppWidgetProvider.

Per la costruzione del layout nulla cambia, eccetto la limitazione sul numero di controlli disponibili.

```
<?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"
    android:gravity="center"
    android:background="@drawable/backwid3"
    >
<TextView android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textColor="#000000"
    android:text="TextView"
    android:paddingLeft="12px" />
</LinearLayout>
```

Aspetto importante è dichiarare il provider per utilizzare il Widget, in un file separato nella cartella res/xml/.

Nel mio caso l'ho chiamato hello\_widget\_provider.xml.

```
<?xml version="1.0" encoding="utf-8" ?>
<appwidget-provider xmlns:android="http://schemas.android.com/apk/res/android"
    android:updatePeriodMillis="10000"
    android:initialLayout="@layout/main"
    android:minWidth="146dip" android:minHeight="72dip"/>
```

Gli attributi minWidth e minHeight definiscono le misure minime del Widget.

Da ricordare che lo schermo è diviso in blocchi da 74dip e solitamente le misure si decidono come multipli di 74dip – 2dip, per permettere di fare bordi particolari o margini, come gli angoli arrotondati.

