

Android by Example

v4.2 JellyBean



Rossi Pietro Alberto

Premessa

Tanto tempo è passato dalla prima guida, troppo.

L'intento era quello di costruire una guida solida e completa nel corso del tempo, senza fretta, con la consapevolezza di dover fare qualcosa di grande per un pubblico vasto come i programmatori Android.

La prima guida ha avuto un successo abnorme, fuori le aspettative iniziali, si parla di circa trentamila download.

Era molto basilare e mancavano varie cose ma soprattutto gratuita! La prima guida gratuita!

Per chi non mi conosce o non ha letto la guida precedente mi ripresento: mi chiamo Alberto ed attualmente sono uno sviluppatore .NET per una azienda che si occupa di pubblica amministrazione.

In passato sono stato anche programmatore su sistemi POS con il buon linguaggio C, mai abbandonato e sempre in voga.

Sono stato relatore a vari LinuxDay, LinuxMeeting, JavaDay a Palermo, in attesa di qualche conferenza fuori.

La mia passione per Android nasce molti anni fa, quando, leggendo molti libri, mi sono chiesto se valeva la pena fare un ebook gratuito, da donare alla community.

Presto fatto l'ebook è uscito fuori, Dicembre 2010, ed adesso dopo 2 anni è pronta la seconda versione, completa ed aggiornata.

A metà 2012, sono stato anche docente per un corso Android all'Università di Palermo, dove non finirò mai di ringraziare Vincenzo Virgilio dell'associazione Sputnix di avermi dato fiducia ed aver partorito un corso eccezionale.

Ribadisco, che non metterò mai in vendita questa guida, ma purtroppo sarà l'ultima riguardo Android.

Spero in qualche casa editrice per scrivere qualcosa di più serio e più concreto.

Anche questa guida si basa sulla brevità e concisione. Non ci sarà molta teoria, quella la lasciamo ai libri veri, ma solo lo stretto e necessario per capire le fondamenta della programmazione su sistemi Android.

Stavolta la guida sarà pubblicata sul mio sito www.spruk.it, non per atto di egoismo, ma perchè ibasblog.it ha cessato di esistere.

Non finirò mai di ringraziare i miei ex collaboratori, che hanno creduto fin dall'inizio in questo progetto, **pinosiciliano** e **cd125**.

Anche se le nostre strade si sono ormai separate, per motivi lavorativi, non si dimenticano certe cose: incoraggiamenti, le nostre lunghe riunioni e tutto il resto.

Menu Generale

Introduzione di base	4
Installazione	9
UserControls	26
Notification	49
LayoutManager	55
Hardware	62
Miscellanea, parte 1	69
SMS e chiamate	82
Concetti avanzati	90
Miscellanea, parte 2	96
Conclusione	104

Introduzione di base

- Programmazione ad oggetti
- Programmazione visuale
- Programmazione ad eventi

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 al 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 deselection 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 deselection 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.

Installazione

- Eclipse, Android SDK e configurazione AVD
- Hello world

Eclipse, Android SDK e configurazione AVD

Il principale **IDE** (Integrated Development Environment) per lo sviluppo su piattaforma Android è **Eclipse**.

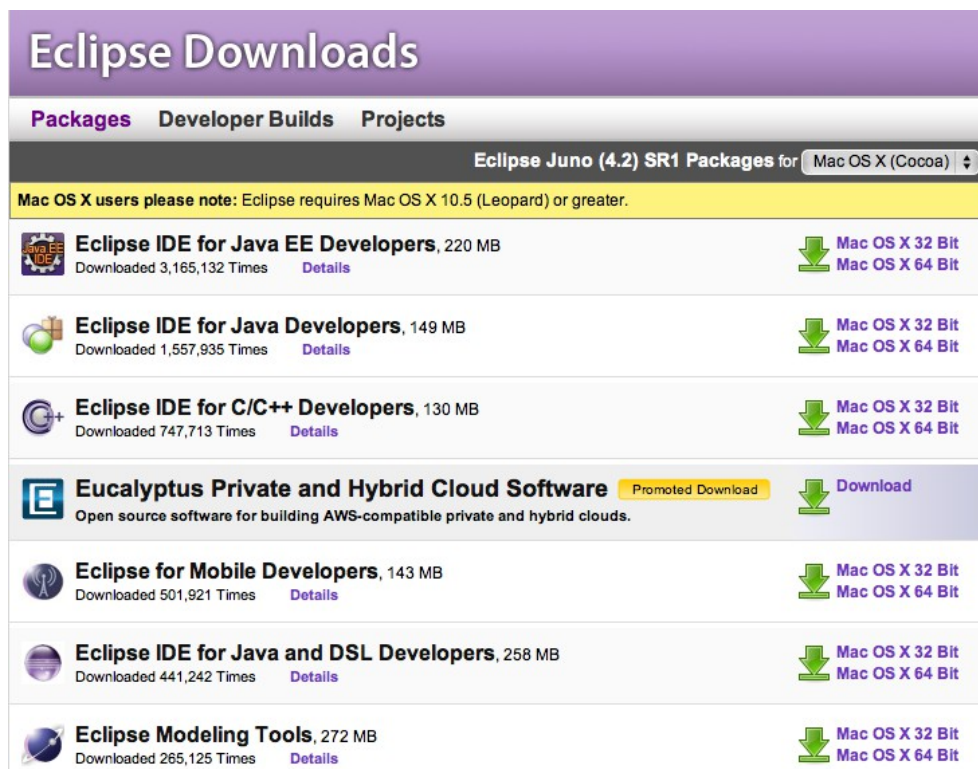
Esso è un ambiente di sviluppo che integra al suo interno vari strumenti per facilitare e centralizzare la costruzione di una applicazione.

Molto famoso per essere **cross-platform** (può essere installato su Windows, Linux e Mac) e per la sua capacità di integrare vari **plugin** per renderlo compatibile con diversi linguaggi di programmazione.

Per installarlo basta andare nella sezione download del sito :

<http://www.eclipse.org/downloads/>

In automatico verrà riconosciuta la versione del sistema operativo e noi possiamo scegliere se scaricare la versione a 32bit o 64bit.



The screenshot shows the 'Eclipse Downloads' page with a purple header. Below the header are tabs for 'Packages', 'Developer Builds', and 'Projects'. A dropdown menu shows 'Eclipse Juno (4.2) SR1 Packages for Mac OS X (Cocoa)'. A yellow note states: 'Mac OS X users please note: Eclipse requires Mac OS X 10.5 (Leopard) or greater.' The main content lists several packages:

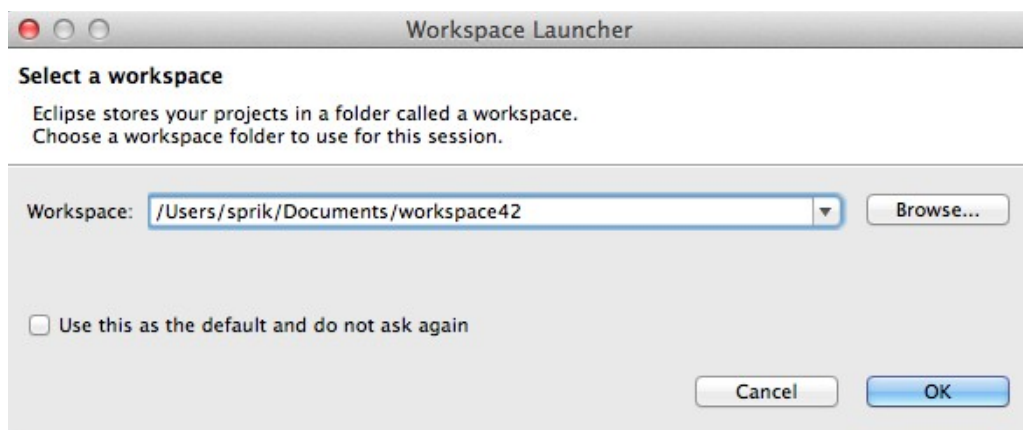
Package Name	Size	Download Count	Details	Download Link
Eclipse IDE for Java EE Developers	220 MB	Downloaded 3,165,132 Times	Details	Mac OS X 32 Bit Mac OS X 64 Bit
Eclipse IDE for Java Developers	149 MB	Downloaded 1,557,935 Times	Details	Mac OS X 32 Bit Mac OS X 64 Bit
Eclipse IDE for C/C++ Developers	130 MB	Downloaded 747,713 Times	Details	Mac OS X 32 Bit Mac OS X 64 Bit
Eucalyptus Private and Hybrid Cloud Software			Promoted Download	Download
Eclipse for Mobile Developers	143 MB	Downloaded 501,921 Times	Details	Mac OS X 32 Bit Mac OS X 64 Bit
Eclipse IDE for Java and DSL Developers	258 MB	Downloaded 441,242 Times	Details	Mac OS X 32 Bit Mac OS X 64 Bit
Eclipse Modeling Tools	272 MB	Downloaded 265,125 Times	Details	Mac OS X 32 Bit Mac OS X 64 Bit

Per comodità ho installato la versione “**Java EE Developers**” in quanto contiene tutte le librerie Java di base ed avanzati per lavorare anche con HTML/JSP/JSF, che possono rivelarsi utili per particolari scopi di sviluppo.

Dopo aver effettuato il download possiamo scompattare l'archivio in una cartella a nostra scelta ed avviare l'ambiente.



Alla prima apertura, ci chiederà la cartella per il **workspace**, il nostro contenitore di progetti.

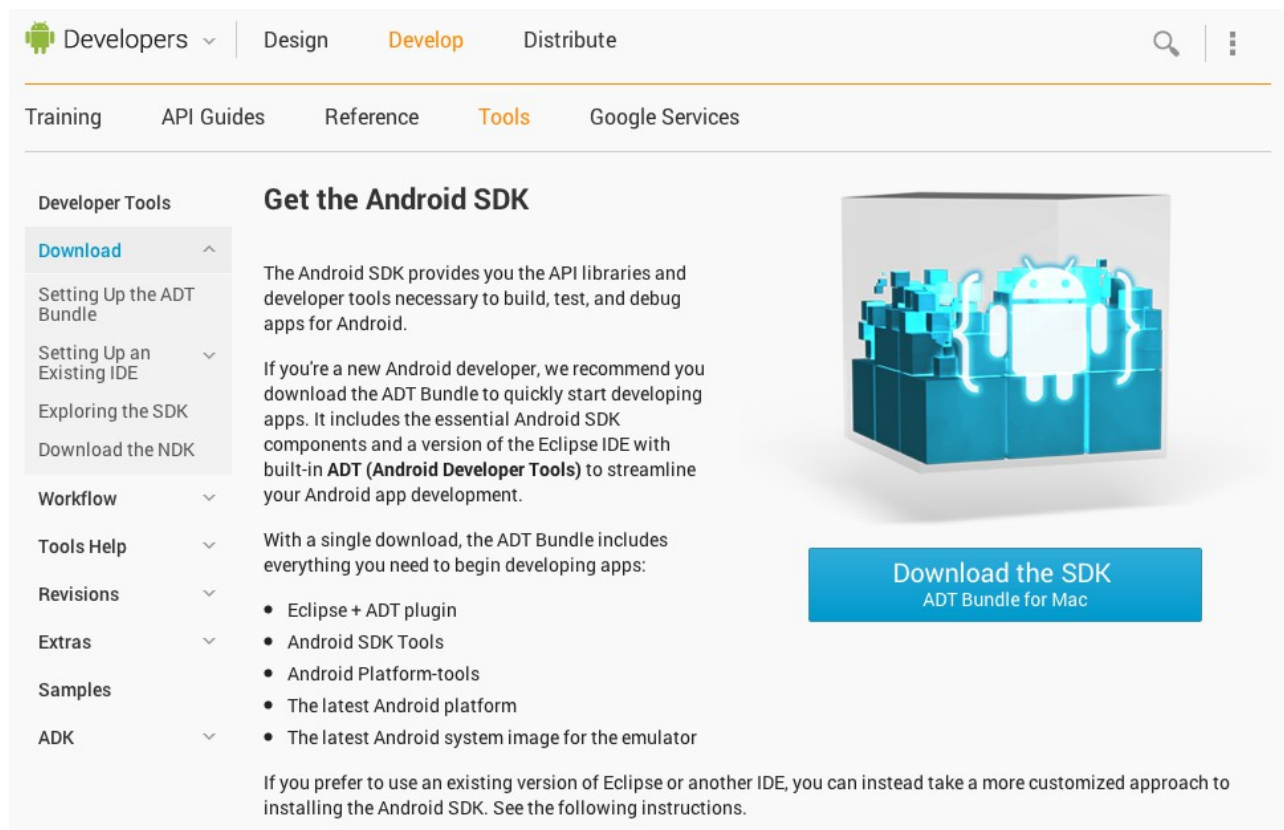


Mettendo la spunta su “Use this as the default and do not ask again”, al successivo avvio non ci chiederà più il workspace e renderà quello scelto precedentemente il predefinito.

Adesso effettuiamo il download del SDK di base per Android dal sito :

<http://developer.android.com/sdk/index.html>

Anche qui verrà riconosciuta la piattaforma di sviluppo e vi mostrerà il file da scaricare più appropriato.



The screenshot shows the 'Developer Tools' section of the Android Developer website. The main heading is 'Get the Android SDK'. The page explains that the Android SDK provides API libraries and developer tools for building, testing, and debugging apps. It recommends downloading the ADT Bundle for new developers. A list of included components is provided: Eclipse + ADT plugin, Android SDK Tools, Android Platform-tools, The latest Android platform, and The latest Android system image for the emulator. A large blue button labeled 'Download the SDK' is prominently displayed, with the subtitle 'ADT Bundle for Mac' below it. The left sidebar contains a navigation menu with options like 'Download', 'Setting Up the ADT Bundle', 'Setting Up an Existing IDE', 'Exploring the SDK', 'Download the NDK', 'Workflow', 'Tools Help', 'Revisions', 'Extras', 'Samples', and 'ADK'.

Developer Tools ▾ | Design | **Develop** | Distribute

Training | API Guides | Reference | **Tools** | Google Services

Developer Tools

- Download** ▴
 - Setting Up the ADT Bundle
 - Setting Up an Existing IDE ▾
 - Exploring the SDK
 - Download the NDK
- Workflow ▾
- Tools Help ▾
- Revisions ▾
- Extras ▾
- Samples
- ADK ▾

Get the Android SDK

The Android SDK provides you the API libraries and developer tools necessary to build, test, and debug apps for Android.

If you're a new Android developer, we recommend you download the ADT Bundle to quickly start developing apps. It includes the essential Android SDK components and a version of the Eclipse IDE with built-in **ADT (Android Developer Tools)** to streamline your Android app development.

With a single download, the ADT Bundle includes everything you need to begin developing apps:

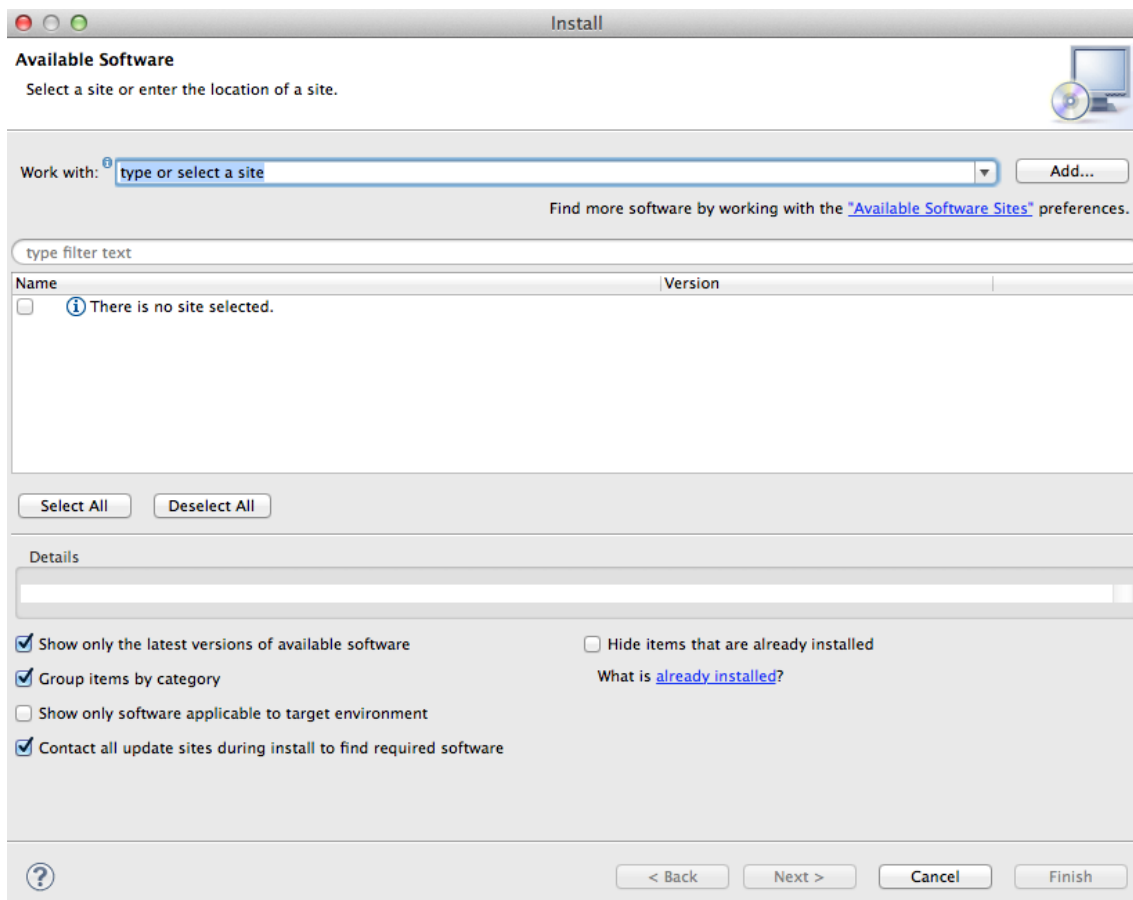
- Eclipse + ADT plugin
- Android SDK Tools
- Android Platform-tools
- The latest Android platform
- The latest Android system image for the emulator

If you prefer to use an existing version of Eclipse or another IDE, you can instead take a more customized approach to installing the Android SDK. See the following instructions.

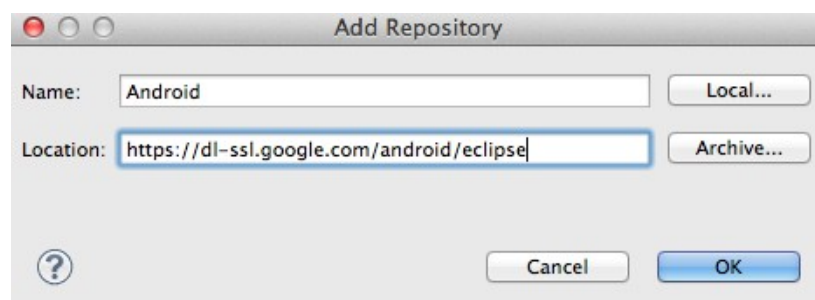
Download the SDK
ADT Bundle for Mac

Anche qui, dobbiamo scompattare l'archivio in una cartella, che servirà per la configurazione.

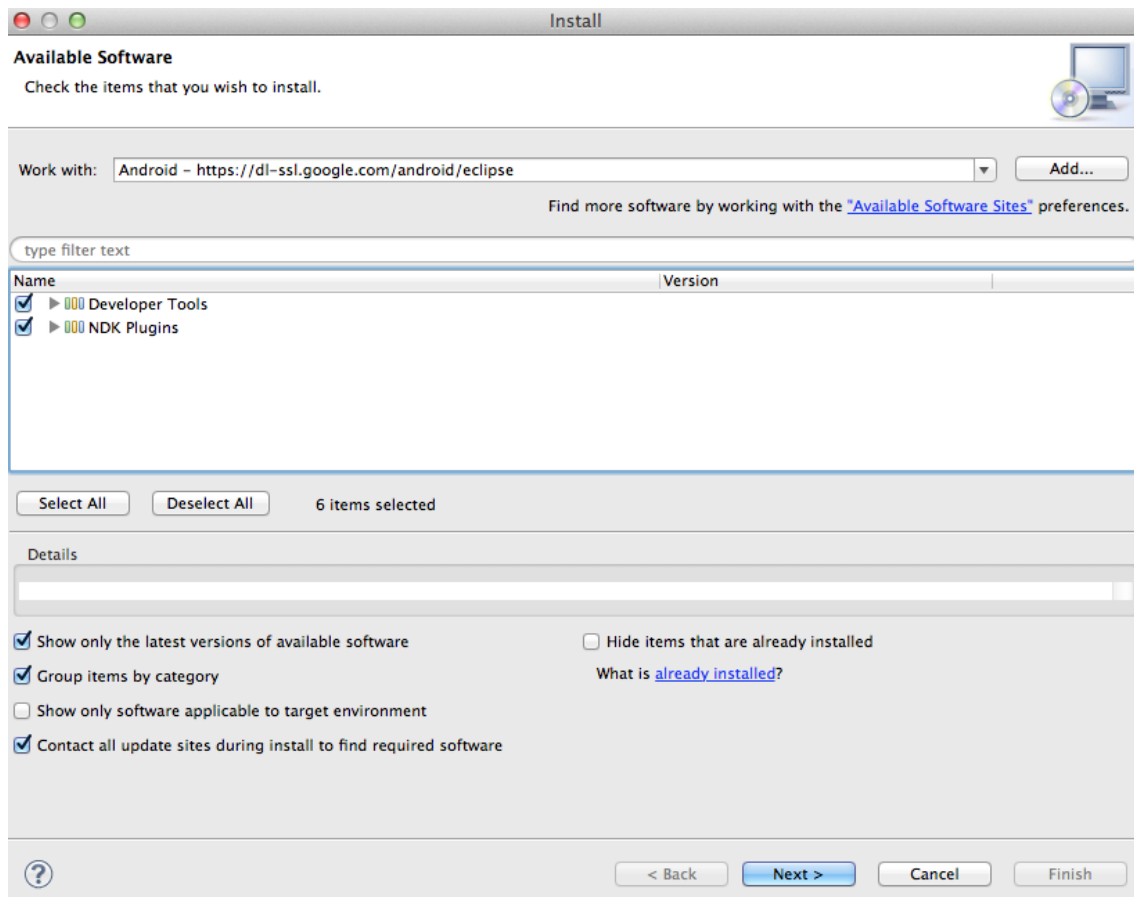
Successivamente dobbiamo installare il plugin di Android per Eclipse andando su **Help** → **Install New Software** e cliccando sul pulsante **Add** della finestra che vi viene proposta



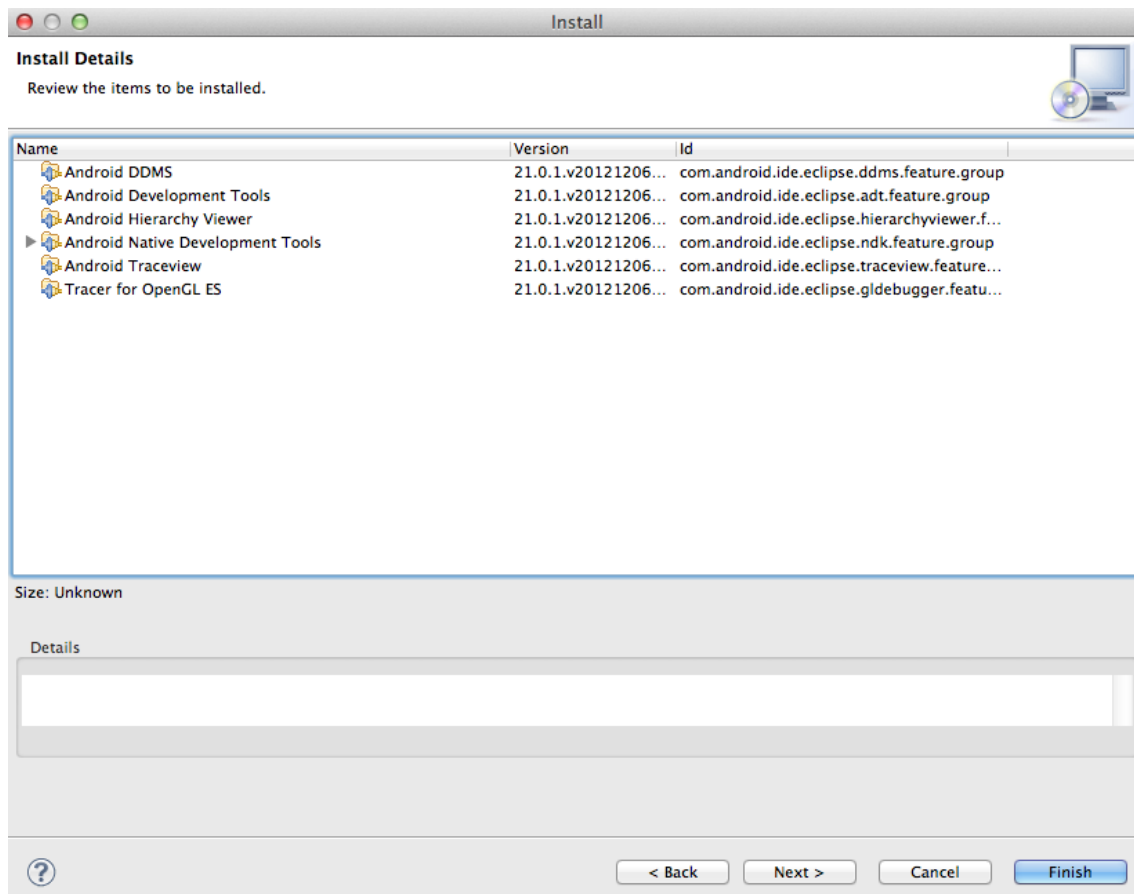
Come nuovo **repository** metteremo un nome a nostra scelta e come indirizzo quello specifico di google <https://dl-ssl.google.com/android/eclipse>



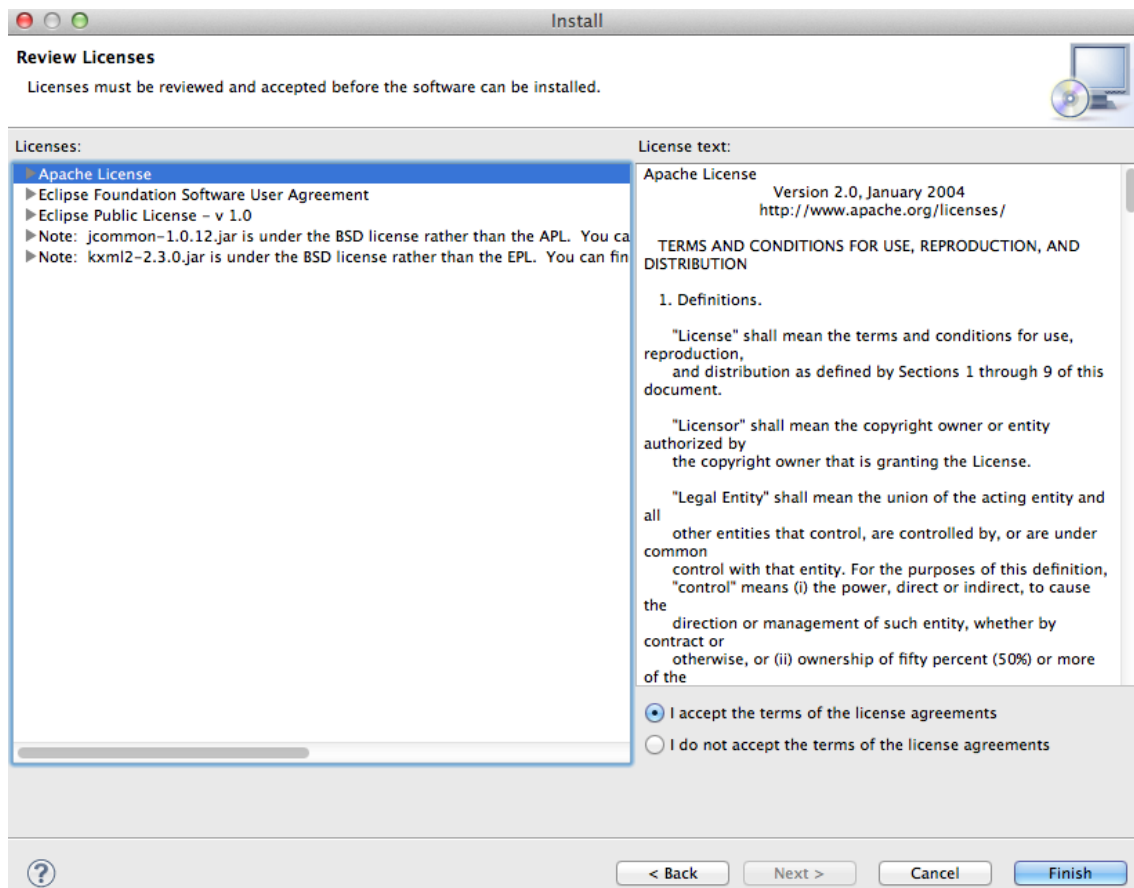
Cliccando su OK ci verrà restituita la lista dei tools da poter scaricare.



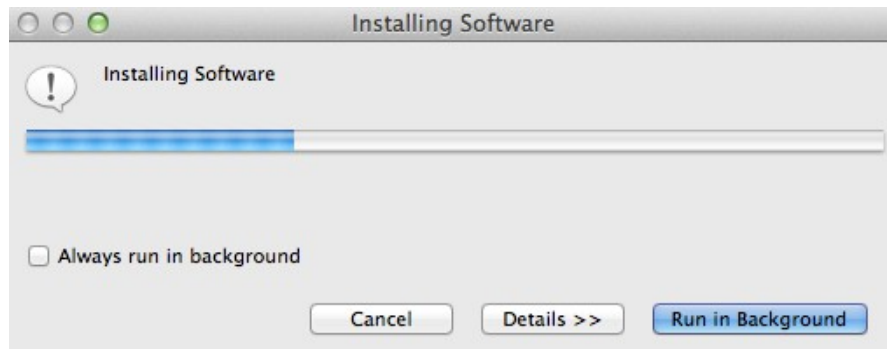
Nel passo successivo ci verranno mostrati i componenti che verranno scaricati.



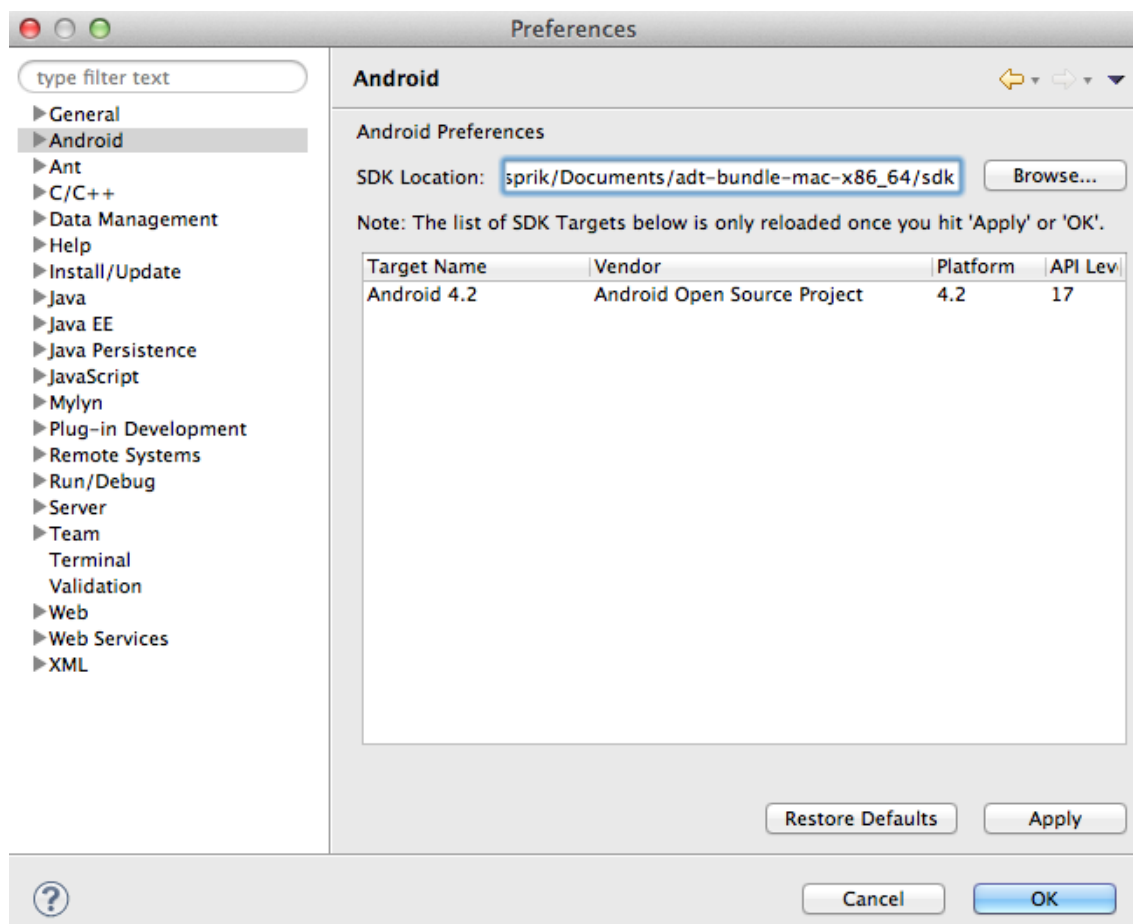
Accettiamo le licenze.



Il software si installa in maniera autonoma ed in poco tempo, secondo sempre la connessione disponibile, dato che deve scaricare una buona quantità di dati da internet.

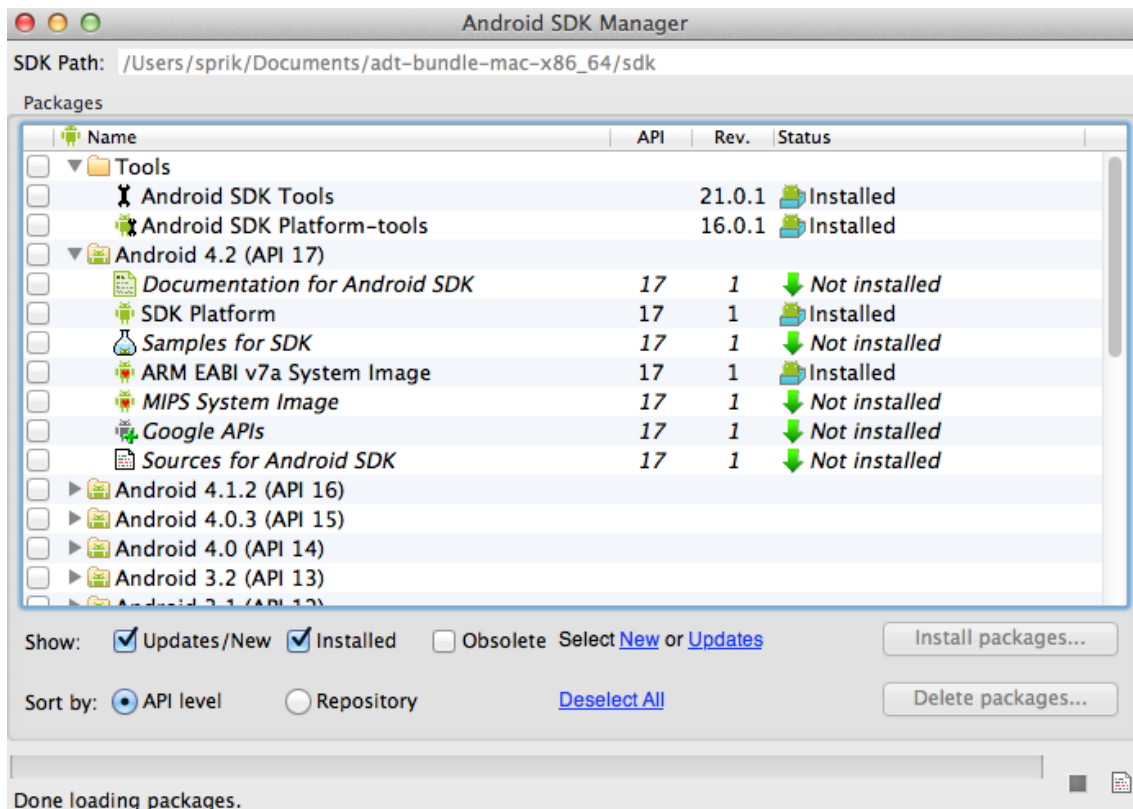


Quando tutto è completato passiamo alla configurazione del SDK scaricato precedentemente su **Preferences** → **Android**



Da notare che, in questa ultima versione per MAC, è già presente SDK relativo alla versione 4.2 di Android.

Lo stesso si può notare dall'**Android SDK Manager**, che si trova sotto il menu Window, dove sarà possibile scaricare altre versioni del SDK, varie librerie ed aggiornare le presenti.

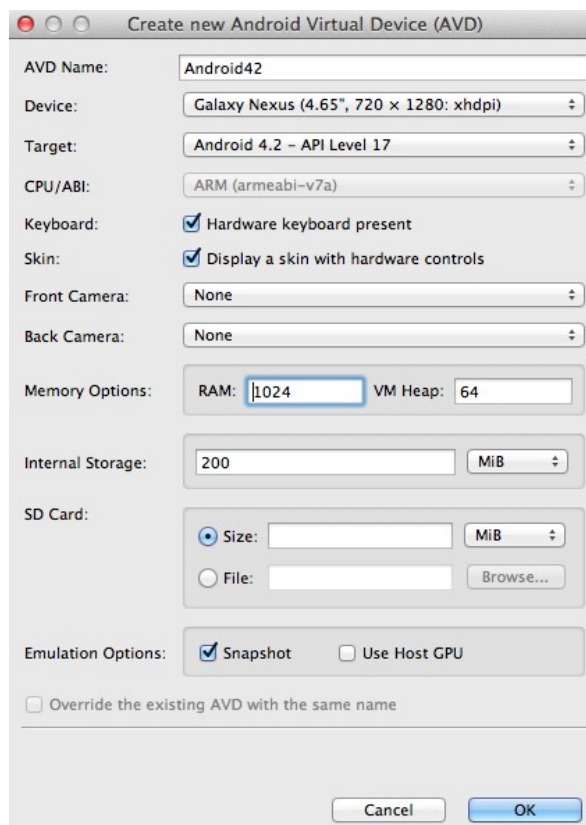


Scorrendo la lista, troviamo negli extra la **Android Support Library**, che ci servirà per gestire le vecchie versioni di Android in modalità compatibile.

Se non la selezioniamo ci verrà richiesta alla configurazione del primo progetto.

Per creare una nuova **AVD** (Android Virtual Device), partiamo dal **Android Virtual Device Manager**, presente sempre sotto il menu Window, dove ci verrà presentata una lista delle AVD create.

Clicchiamo sul tasto New per crearne una nuova.

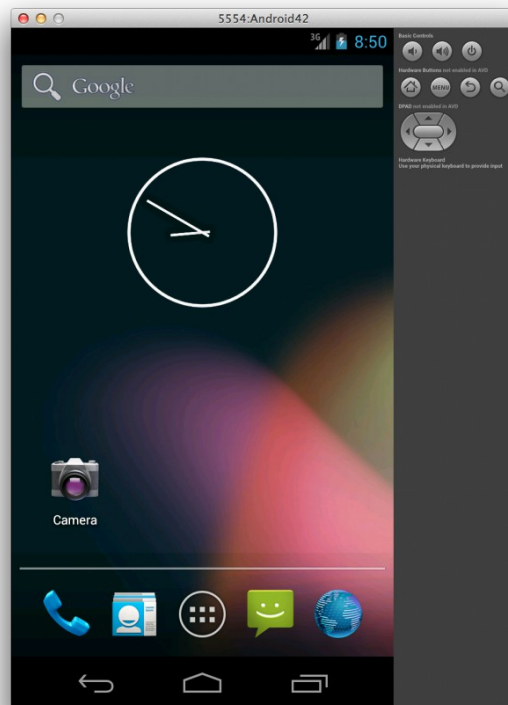


La configurazione è abbastanza semplice e non implica particolari azioni.

Nel mio caso ho selezionato che il dispositivo da emulare è un Galaxy Nexus, in automatico il programma preparerà tutto il necessario.

Sole impostazioni che potremmo modificare, in questo caso, è la selezione delle Camere, dove sarà possibile selezionare una eventuale webcam del nostro PC o notebook, la quantità di memoria di una eventuale scheda SD e lo **Snapshot**, dove verrà ibernato l'ultimo stato della macchina in modo da non doverla riavviare da zero.

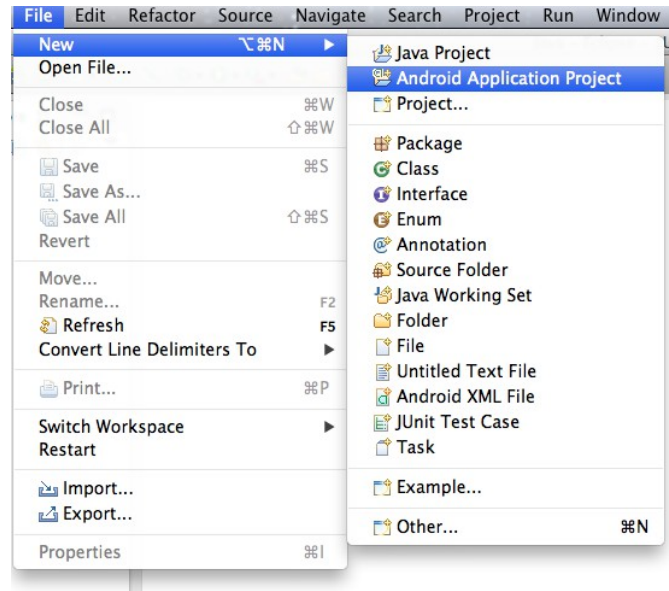
Il risultato sarà una perfetta macchina virtuale Android.



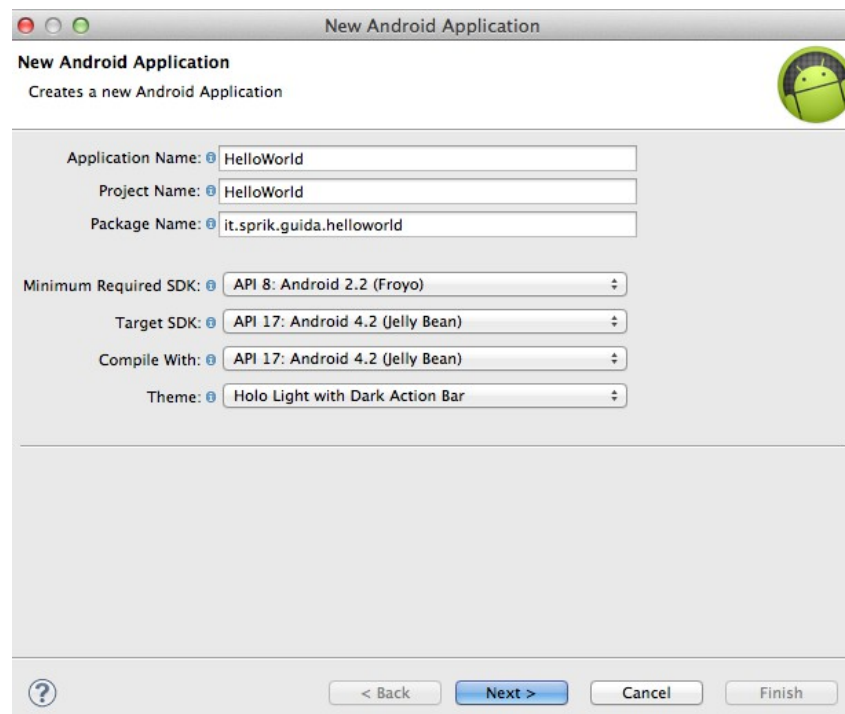
HelloWorld

Per aprire un nuovo progetto andiamo su

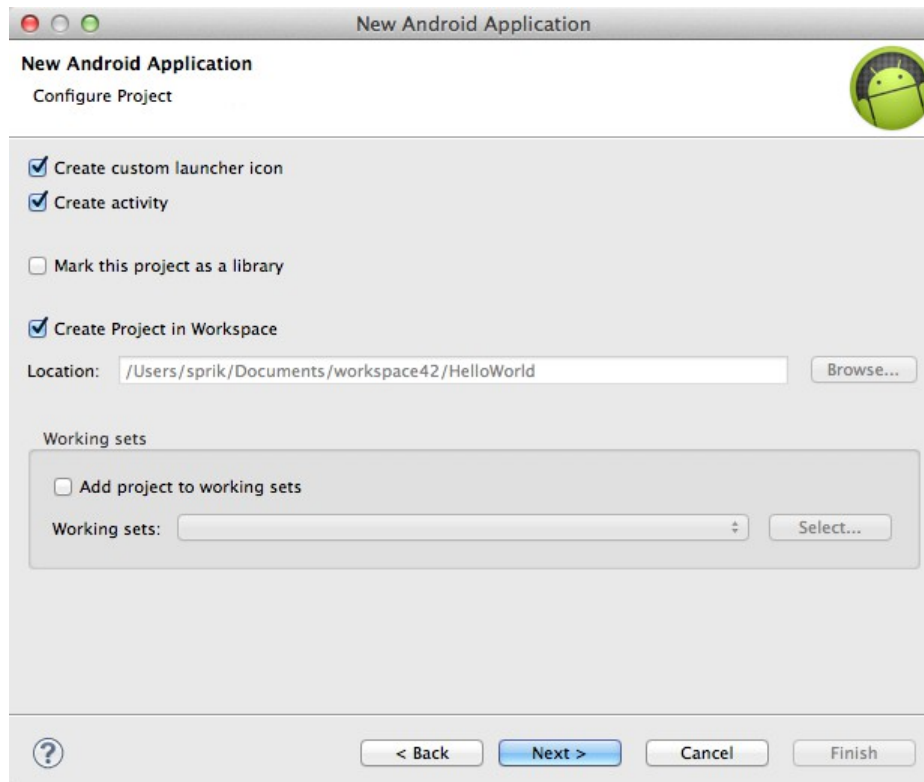
File → **New** e dalla lista che compare selezioniamo **Android Application Project**.



Come primo passo dobbiamo indicare il nome dell'applicazione, del progetto e del pacchetto. Quest'ultimo solitamente è composto dal nome del sito seguito dal nome del progetto. In questa prima parte, sarà possibile selezionare la versione di android su cui è destinata l'applicazione e il tema principale.



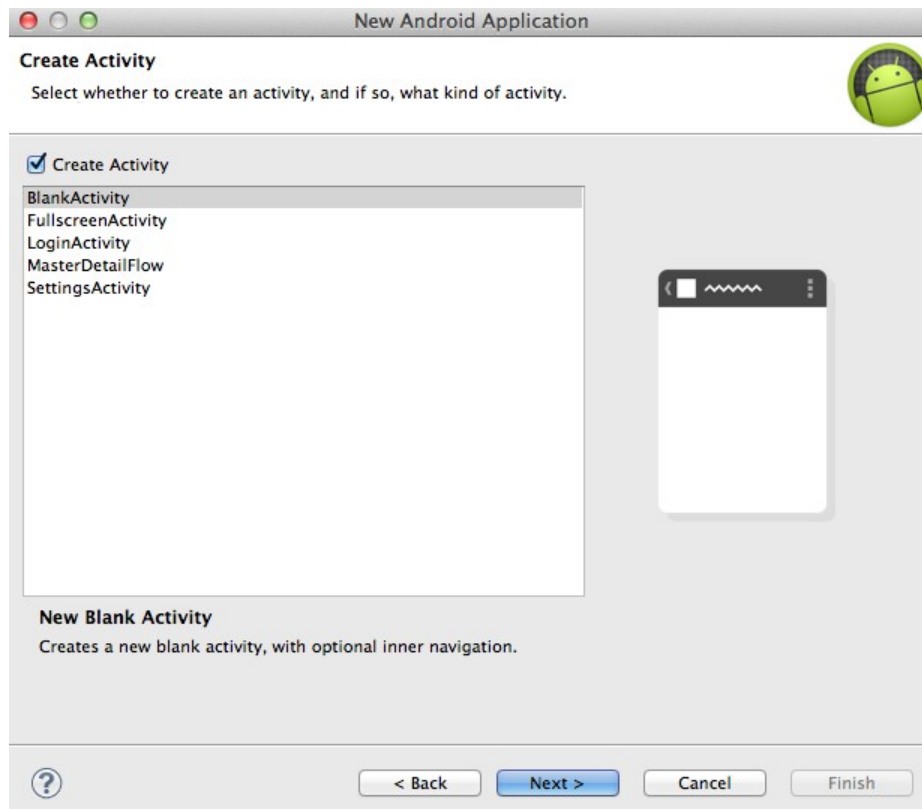
Il passo successivo prevede impostazioni di base, come la directory del progetto e la possibilità di pre-creare o meno activity (i vecchi form, detto in maniera basilare).



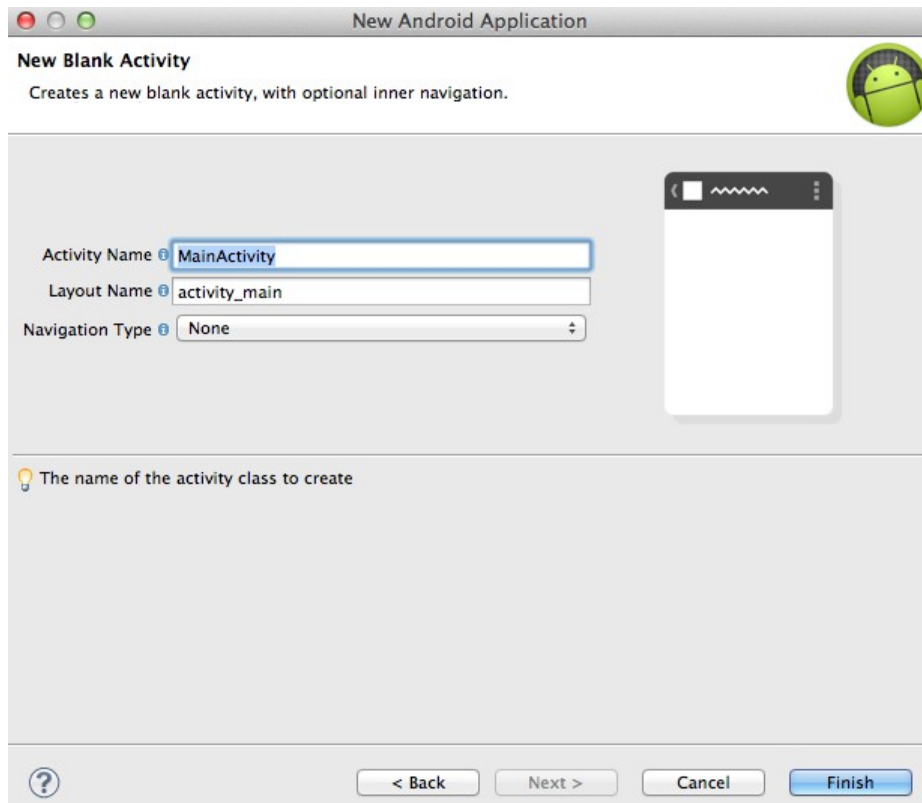
Poi si provvede alla configurazione dell'icona dell'applicazione.



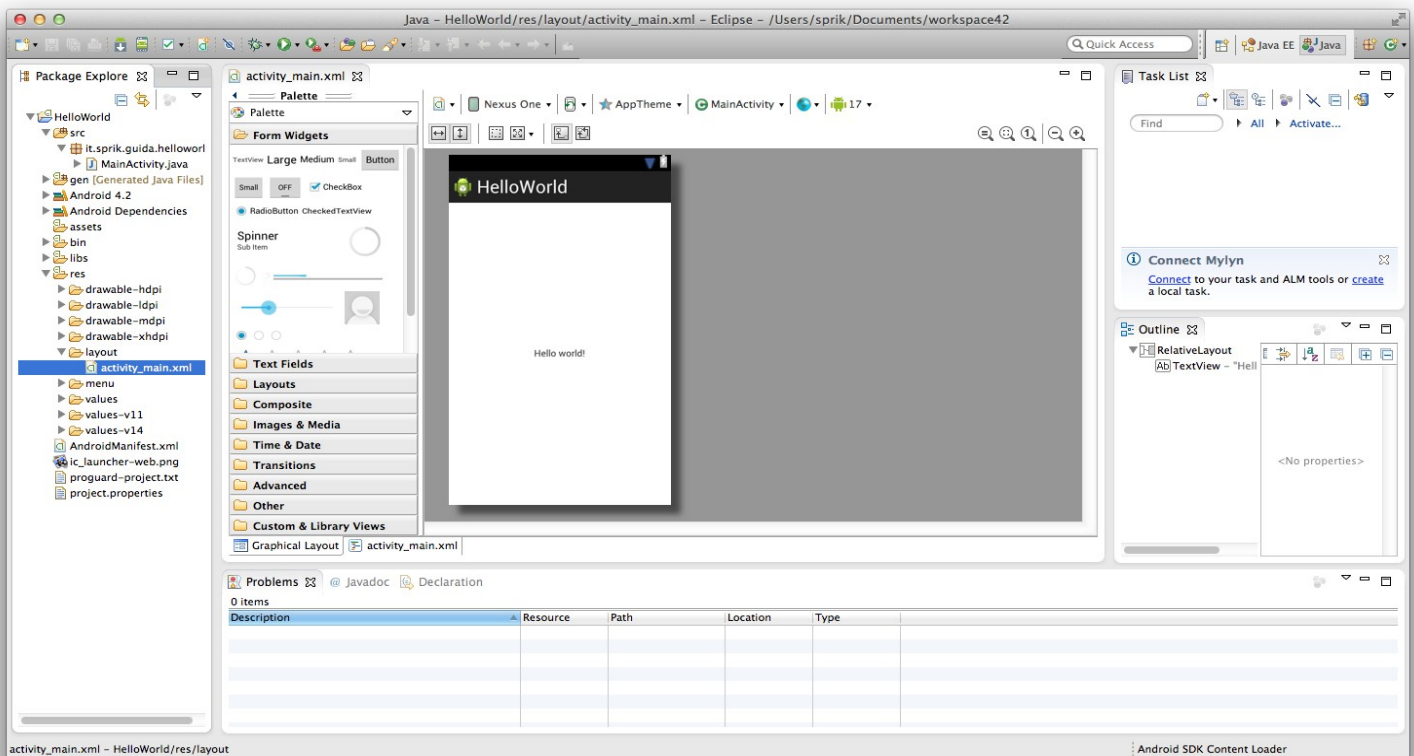
Lo step successivo ci mostra che possiamo selezionare, per la nostra activity preimpostata, il tipo di activity: vuota, a tutto schermo, impostazioni, ecc.



L'ultima schermata ci chiede il nome dell'activity creata, il nome del layout associato e il tipo di navigazione.



Il risultato sarà il seguente



Nella parte sinistra troveremo i file di progetto.

Le directory android di base sono divise in

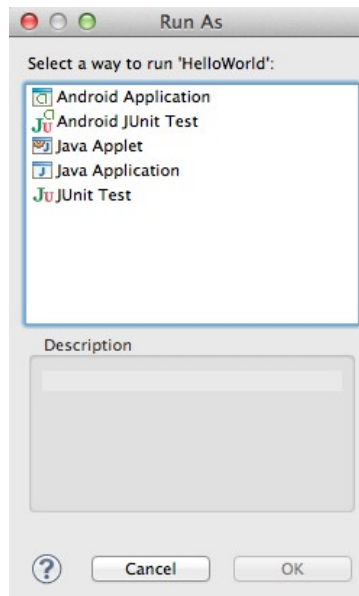
- **src**, contiene i sorgenti java
- **gen**, contiene i file generati dal ADT
- **assets**, contiene i file raw ed è possibile esplorarla tramite URI.
- **bin**, contiene i binari generati, compreso il file apk (eseguibile android)
- **libs**, contiene le varie librerie aggiuntive
- **res**, contiene le varie risorse: layout, valori ed xml di impostazione vari.

In un progetto android è presente anche un file **AndroidManifest.xml** che contiene le varie impostazioni dell'applicazione.

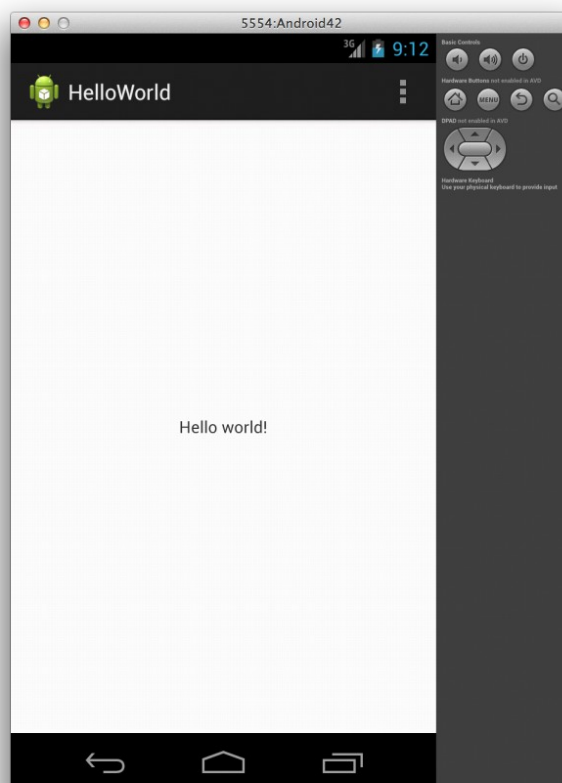
Nella parte centrale visualizzeremo tutti i sorgenti e, come nel caso del file xml, la sua possibile configurazione pratica, come la gestione manuale del posizionamento dei controlli o l'aggiunta di stringhe, impostazioni, ecc.

Al primo avvio di esecuzione ci verrà richiesto che tipo di applicazione è, noi sceglieremo **Android Application**.

E' anche possibile effettuare una configurazione manuale andando su Run → Run Configurations.



Il risultato sarà il seguente: il nostro emulatore ci mostrerà la nostra applicazione



I file principali sono due, in questo caso:
Il file

MainActivity.java, così composto

```
// package di base
package it.sprik.guida.helloworld;

// librerie da importare
import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;

// classe activity dove inseriremo le nostre operazioni da fare
public class MainActivity extends Activity {

    // intercettiamo l'evento onCreate che viene eseguito quando l'activity viene creata
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    // evento per la creazione di un menu, lo analizzeremo successivamente
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate menu
        getMenuInflater().inflate(R.menu.activity_main, menu);
        return true;
    }
}
```

activity_main.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:text="@string/hello_world" />

</RelativeLayout>
```

Nel primo file con la riga `setContentView(R.layout.activity_main);` associamo il file layout `activity_main` all'activity corrente.

Quindi quando verrà creata l'activity `MainActivity` verrà utilizzato il layout presente in `activity_main.xml`.

Nel secondo file è utile notare `android:text="@string/hello_world"` questo comporta che, invece di scrivere direttamente il testo da assegnare, lo conserviamo in una variabile di risorsa di tipo stringa.

Questi valori sono conservati sotto la cartella `/res/values/strings.xml` così composto:

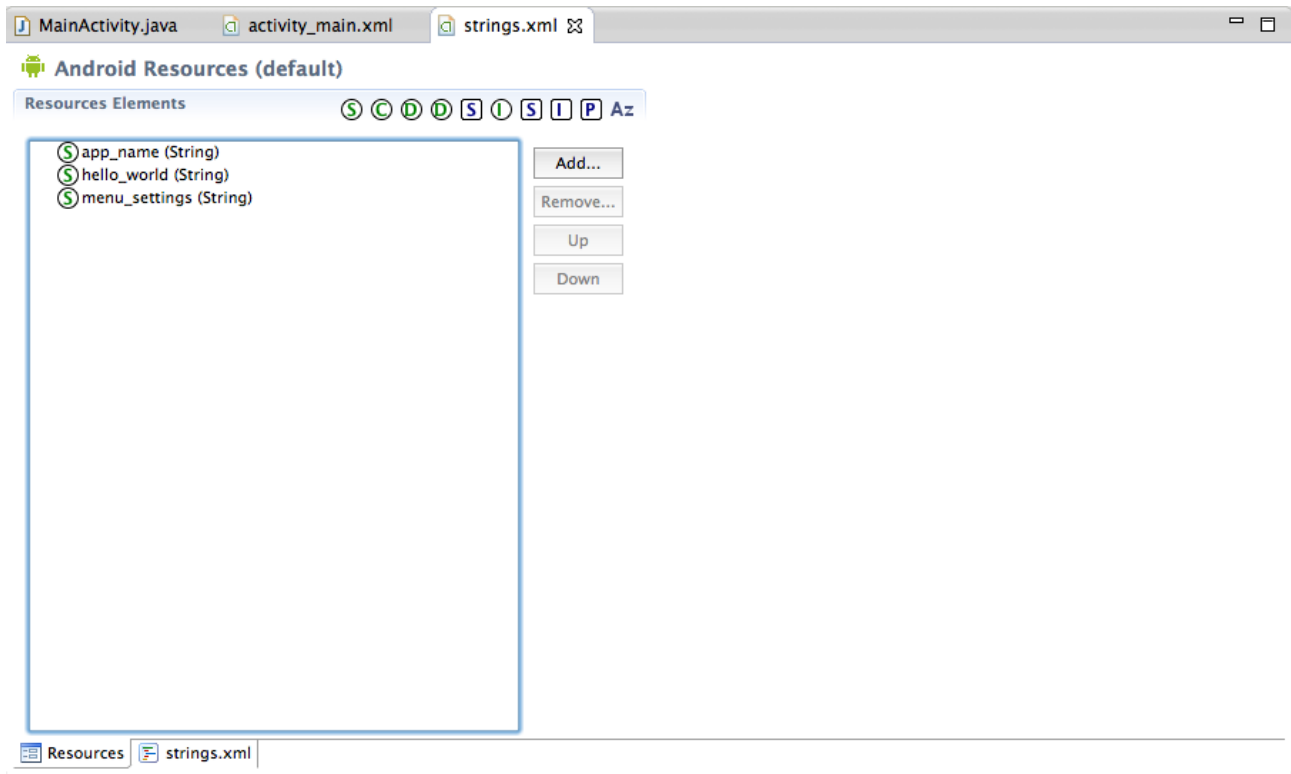
```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">HelloWorld</string>
    <string name="hello_world">Hello world!</string>
    <string name="menu_settings">Settings</string>
</resources>
```


L'editor di eclipse ci aiuterà ad editare questo file, senza metterci materialmente le mani, cliccandoci sopra.

Naturalmente, quando noi andremo a richiamare `android:text="@string/hello_world"` si farà riferimento alla stringa impostata `<string name="hello_world">Hello world!</string>`

Lo stesso procedimento viene utilizzato per editare altre risorse.

Esempio



User Controls

- TextView, AutoCompleteTextView, MultiAutoCompleteTextView e CheckedTextView
- EditText
- Button, ImageButton e ToggleButton
- RadioButton e CheckBox
- ListView
- GridView
- DatePicker e TimePicker
- AnalogClock e DigitalClock
- Chronometer
- ImageView
- ProgressBar, RatingBar e SeekBar
- Spinner
- ExpandableListView
- WebView

TextView, AutoCompleteTextView, MultiAutoCompleteTextView e CheckedTextView

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

Il controllo **AutoCompleteTextView** è una casella di testo con delle possibili parole di default che possono essere inserite.

Quando iniziamo a scrivere il controllo mostrerà le possibili parole che possono essere scelte da 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 di possibili parole che ci viene proposta, funziona una sola volta, cioè verrà visualizzata solo per una parola.

La **MultiAutoCompleteTextView** è praticamente una **AutoCompleteTextView** con l'aggiunta che l'intellinsense opera su più parole, opportunamente separate da un carattere speciale.

Quindi la lista delle parole verrà visualizzata più volte e non solo una volta come nell'**AutoCompleteTextView**.

L'esempio di codice mostra con semplicità come utilizzare questi tre controlli

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello" />
    <AutoCompleteTextView android:id="@+id/txtauto"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content" />
    <MultiAutoCompleteTextView android:id="@+id/txtmulti"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content" />
    <CheckedTextView android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/check"
        android:gravity="center_horizontal"
        android:textColor="#0000FF"
        android:checked="false"
        android:checkMark="@drawable/ic_launcher" />
</LinearLayout>
```

E' facile notare che, caso di **TextView** e **CheckedTextView**, per modificare il testo di base basta editare la proprietà `text`.

Le proprietà in comune, come `layout_width` e `layout_height` servono per impostare la grandezza del componente:

`fill_parent` sta per riempi il contenitore e `wrap_content` è la grandezza di default del componente.

Sia per **AutoCompleteTextView** sia per **MultiAutoCompleteTextView** bisogna caricare le parole via codice, ed utilizzeremo un **ArrayAdapter** per fare ciò.

```

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

        ArrayAdapter<String> aa = new ArrayAdapter<String>(this,
        android.R.layout.simple_dropdown_item_1line, new String[] { "Inglese", "Spagnolo", "Tedesco",
        "Italiano" });

        AutoCompleteTextView actv = (AutoCompleteTextView) findViewById(R.id.txtauto);
        actv.setAdapter(aa);

        MultiAutoCompleteTextView mactv =
        (MultiAutoCompleteTextView) this.findViewById(R.id.txtmulti);
        mactv.setAdapter(aa);
        mactv.setTokenizer(new MultiAutoCompleteTextView.CommaTokenizer());
    }

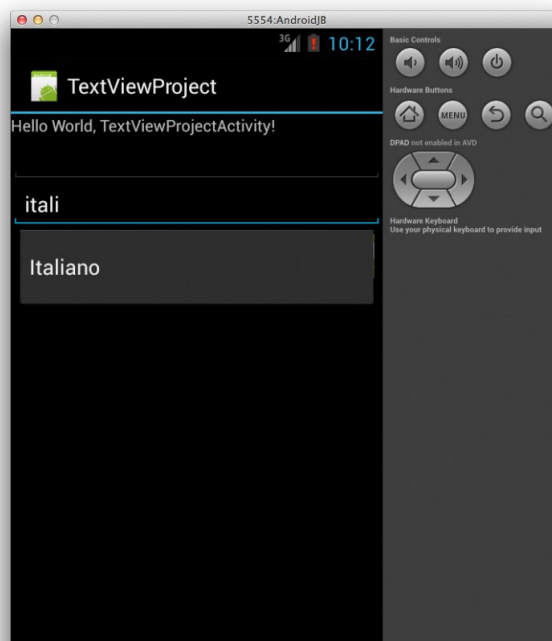
```

Il codice è abbastanza semplice: vengono ricavati i due componenti tramite l'id definito nel layout e viene utilizzato lo stesso ArrayAdapter per caricare le parole.

Un ArrayAdapter è una specie di vettore, nel caso nostro tipizzato a String (quindi un vettore di stringhe), dove le stringhe passate come parametri vengono adattate ad elementi del componente identificato, in questo caso simple_dropdown_item_1line, cioè semplice lista ad una colonna.

Nel caso della MultiAutoCompleteTextView da notare è il metodo **setTokenizer**, dove viene specificato che il carattere speciale che deve fare da separatore è una virgola.

Il risultato finale è questo



EditText

L'oggetto **EditText** è la comune TextBox di qualsiasi altro linguaggio di programmazione. Permette l'inserimento di testo e di gestire gli eventi relativi al suo contenuto.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
    <EditText android:id="@+id/edit1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:hint="Scrivi qualcosa..." />
    <TextView android:id="@+id/testo"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="" />
</LinearLayout>
```

La proprietà **hint** serve a definire un messaggio iniziale, solitamente come suggerito al testo da inserire, viene visualizzato di colore grigio chiaro e quando si comincia a scrivere scompare.

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    text = (TextView) this.findViewById(R.id.testo);
    EditText edit1 = (EditText) this.findViewById(R.id.edit1);
    edit1.addTextChangedListener(new TextWatcher() {

        @Override
        public void beforeTextChanged(CharSequence arg0, int arg1, int arg2, int arg3) {
        }

        @Override
        public void onTextChanged(CharSequence arg0, int arg1, int arg2, int arg3) {
            text.setText(arg0);
        }

        @Override
        public void afterTextChanged(Editable s) {
        }

    });
}
```

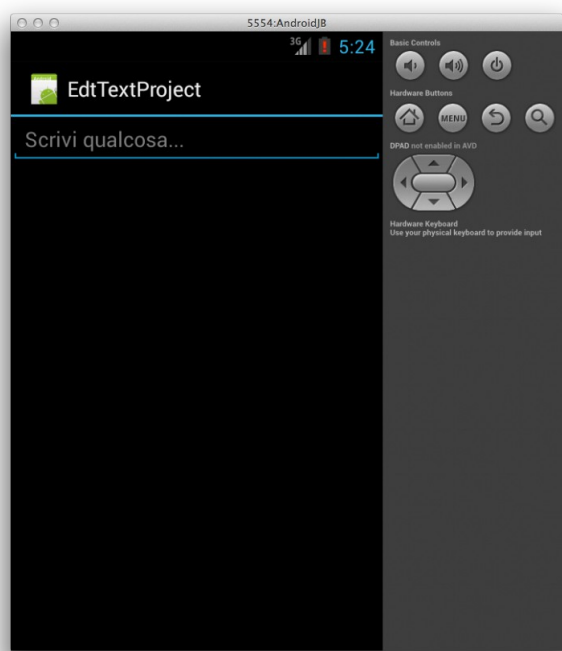
Nell'esempio proposto, tramite l'evento **onTextChanged**, controllo se è cambiato del testo nella EditText e l'assegno tramite il metodo **setText** alla TextView.

Da notare che il testo modificato è possibile trovarlo come primo argomento dell'evento, nel nostro caso era la variabile arg0.

E' possibile ricavare il testo dalla EditText, come nella TextView ed in molti controlli di tipo testo, con il metodo **getText**, che restituisce la sequenza di caratteri contenuta nel controllo.

Gli altri due eventi, **beforeTextChanged** e **afterTextChanged**, servono rispettivamente ad intercettare prima e dopo la modifica del testo contenuto nel controllo.

Esempio



Button, ImageButton e ToggleButton

Il **Button** (pulsante) è uno dei più classici componenti, utilizzato spesso per dare via a delle operazioni.

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

Il **ToggleButton** è il classico pulsante di On/Off.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
    <Button android:id="@+id/btn1"
        android:text="@string/pulsante"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <ImageButton android:id="@+id/btnimg1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:contentDescription="@string/contentimg"
        android:text="@string/immagine"
        android:src="@drawable/ic_launcher"
        android:onClick="TestClick" />
    <ToggleButton android:id="@+id/btntoogle1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textOn="Acceso"
        android:textOff="Spento" />
</LinearLayout>
```

Nel componente ImageButton notiamo che è presente un attributo **contentDescription**. Quest'ultimo serve a descrivere il tipo di contenuto visivo ed è presente in ogni componente di tipo immagine.

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

    Button btn1 = (Button) findViewById(R.id.btn1);
    btn1.setOnClickListener(new OnClickListener() {

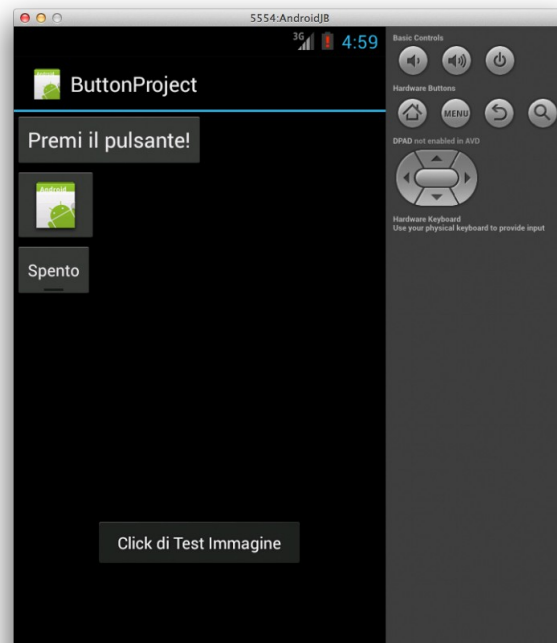
        @Override
        public void onClick(View arg0) {
            Toast.makeText(getApplicationContext(), "Click di Test",
            Toast.LENGTH_LONG).show();
        }

    });
}

public void TestClick(View view) {
    Toast.makeText(getApplicationContext(), "Click di Test Immagine", Toast.LENGTH_LONG).show();
}
```

Dal codice andiamo a gestire gli eventi di click dei componenti in due diversi modi: il primo è associare un listener (gestore eventi) che intercetta la pressione del tasto (evento onClick), in questo caso **OnClickListener** associato al componente Button; il secondo metodo è associare un metodo direttamente nella gestione del layout, nel nostro caso nel layout del componente ImageButton è stato configurato l'attributo onClick sul metodo TestClick.

Risultato



Per quanto riguarda il `ToggleButton` è possibile intercettare l'evento **`onCheckedChange`** oppure direttamente tramite il metodo **`isChecked`** per verificare se è impostato su On o su Off.

RadioButton e CheckBox

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

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

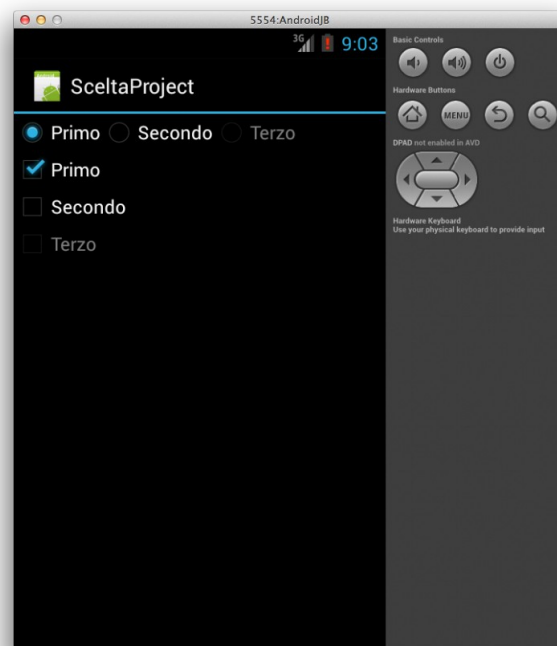
In altre parole, si inseriscono una serie di **RadioButton** in un contenitore **RadioGroup** (gruppi di **RadioButton**) e se ne potrà scegliere solo una delle varie disponibili.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
    <RadioGroup android:id="@+id/rdbgroup1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="horizontal" >
        <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>
    <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>
```

Sia per il componente **CheckBox** sia per il **RadioButton** gli attributi utilizzati per attivare e selezionare sono rispettivamente **enabled** e **checked**.

Sono entrambi di tipo booleano, quindi se per esempio **enabled** è impostato su **false**, il componente sarà disattivato e di conseguenza non selezionabile, invece se **checked** è impostato su **true**, il componente sarà preselezionato.

Risultato



Per entrambi si può usare il metodo **isChecked** per controllare quale componente è stato selezionato.

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

Qui al posto di un **ArrayAdapter** abbiamo utilizzato un **ListAdapter** che si adatta meglio al componente **ListView**.

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

    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, nel nostro adapter verranno inserite le stringhe caricate sul vettore **cols** e sarà di tipo **simple_list_item_1**, ovvero lista semplice ad un elemento.

Risultato



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:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
    <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 caricati.

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

    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);
}
```

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

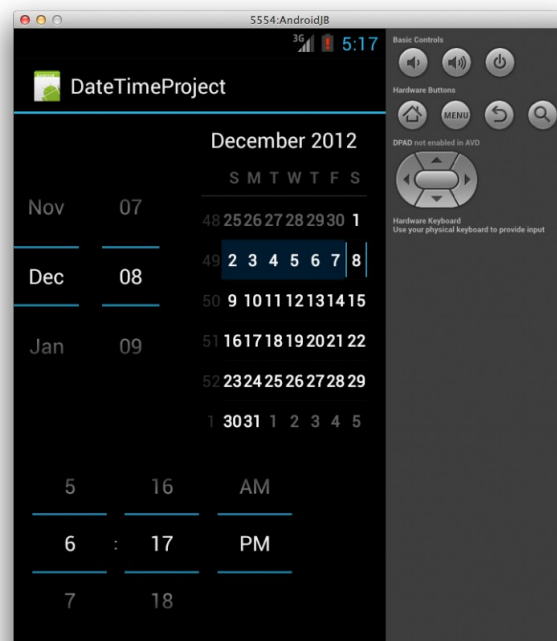


DatePicker e TimePicker

Il **DatePicker** viene utilizzato per selezionare una data.

Il **TimePicker**, invece, viene utilizzato per selezionare un orario.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
    <DatePicker android:id="@+id/datePicker1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <TimePicker android:id="@+id/timePicker1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</LinearLayout>
```



AnalogClock e DigitalClock

AnalogClock è il classico orologio analogico.

DigitalClock rappresenta l'orologio digitale.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
    <AnalogClock android:id="@+id/analog1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal" />
    <DigitalClock android:id="@+id/digit1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</LinearLayout>
```



Chronometer

Il **Chronometer** rappresenta il 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:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <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="@string/start" />
    <Button android:id="@+id/btnstop"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/stop" />

</LinearLayout>
```

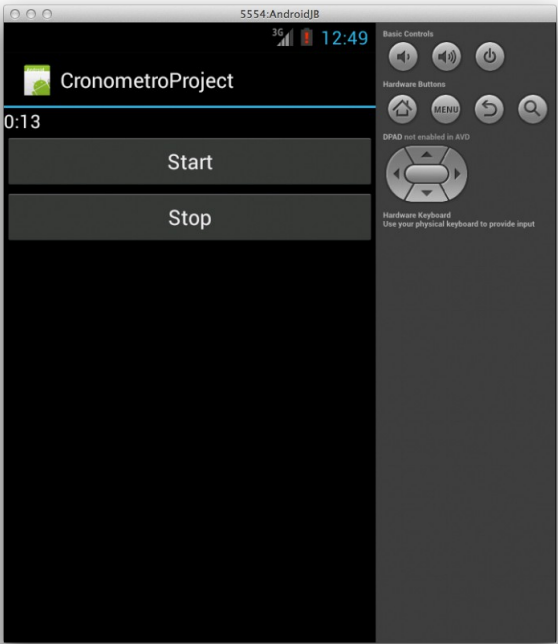
Nell'esempio abbiamo utilizzato due pulsanti, start e stop, rispettivamente per iniziare e fermare il cronometro.

```
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.setBase(SystemClock.elapsedRealtime());
            cr1.start();
        }
    });
    Button btnstop = (Button) this.findViewById(R.id.btnstop);
    btnstop.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View arg0) {
            cr1.stop();
        }
    });
}
```

Da notare il metodo **setBase** che serve a resettare il cronometro, in questo caso, prima di farlo ripartire.

Risultato

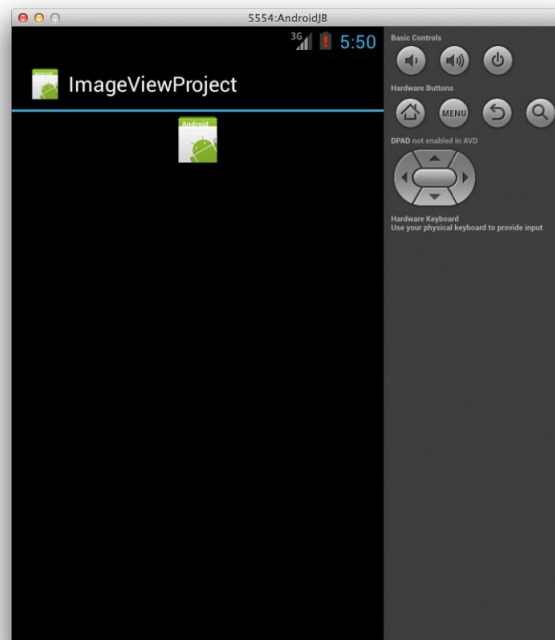


ImageView

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:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
    <ImageView android:src="@drawable/ic_launcher"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:contentDescription="@string/img" />
</LinearLayout>
```

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



ProgressBar, RatingBar e SeekBar

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

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

La **SeekBar** è una barra che consente di selezionare un valore in un certo range di valori discreti.

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

    <ProgressBar android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:max="100"
        android:progress="65"
        style="@android:style/Widget.ProgressBar.Horizontal" />

    <RatingBar android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:max="100"
        android:rating="65" />

    <SeekBar android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:max="100"
        android:progress="65" />

</LinearLayout>
```

Per tutti e tre i controlli sono stati utilizzati gli attributi **max** per indicare il valore massimo che può assumere la barra e **progress** o **rating** (nel caso della RatingBar) per definire il valore di default.

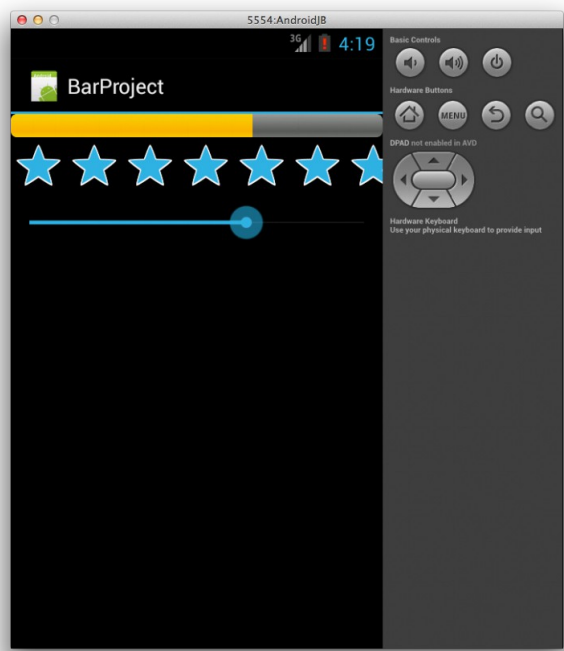
Da notare è che nella ProgressBar è stato utilizzato l'attributo **style** che permette di definire il tipo di ProgressBar da visualizzare.

Nel nostro caso si tratta della classica barra orizzontale ma potremmo impostare l'attributo in `style="@android:style/Widget.ProgressBar.Large"` per visualizzare una ProgressBar circolare.

Per la ProgressBar e per la SeekBar è possibile utilizzare i metodi **getProgress** e **setProgress** per avere o modificare il valore della barra.

Invece, per la RatingBar, i rispettivi metodi sono **getRating** e **setRating**.

Esempio



Spinner

Lo **Spinner** è un controllo per visualizzare e scegliere una opzione su una lista di dati.

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

Nell'esempio proposto abbiamo utilizzato una lista di lingue memorizzate nel file delle risorse (strings.xml) che durante l'evento onCreate verranno caricate dall'applicazione.

Il file strings.xml è così composto

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string name="hello">Hello World, SpinnerProjectActivity!</string>
    <string name="app_name">SpinnerProject</string>
    <string name="lingua">Scegli la lingua</string>
    <string-array name="lingue">
        <item>Italiano</item>
        <item>Inglese</item>
        <item>Francese</item>
        <item>Spagnolo</item>
        <item>Tedesco</item>
    </string-array>

</resources>
```

Come per una `AutoCompleteTextView` utilizziamo un `ArrayAdapter` per caricare gli elementi, ma stavolta invece di caricarli dal codice li prendiamo dalle risorse, come accennato precedentemente.

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

    Spinner s = (Spinner) findViewById(R.id.spinner);
    final ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(this,
R.array.lingue, android.R.layout.simple_spinner_item);
    adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
    s.setAdapter(adapter);
    s.setOnItemSelectedListener(new OnItemSelectedListener() {

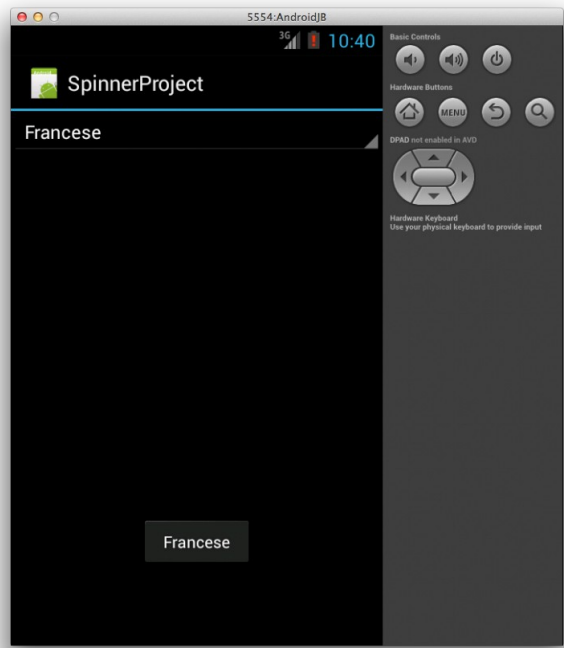
        @Override
        public void onItemSelected(AdapterView<?> arg0, View arg1, int arg2, long arg3) {
            Toast.makeText(getApplicationContext(), adapter.getItem(arg2).toString(),
Toast.LENGTH_LONG).show();
        }
        @Override
        public void onNothingSelected(AdapterView<?> arg0) {

        }
    });
}
```

Per lo **Spinner** è previsto gestire il listener **OnItemSelectedListener** che contiene due eventi: **onItemSelected** e **onNothingSelected**.

Il primo viene scatenato quando selezioniamo un elemento della lista, il secondo quando non viene selezionato nulla, per esempio quando premiamo il tasto Back e la lista si chiude.

Esempio



ExpandableListView

Una **ExpandableListView** è una **ListView** a due livelli.

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

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
    <ExpandableListView android:id="@+id/explist1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content" />
</LinearLayout>
```

Di seguito la classe di esempio:

```
public class ExpandableListViewProjectActivity extends Activity {

    ExpandableListAdapter mAdapter;

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

        mAdapter = new MyExpandableListAdapter();
        ((ExpandableListView) findViewById(R.id.explist1)).setAdapter(mAdapter);
    }

    public class MyExpandableListAdapter extends BaseExpandableListAdapter {
        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() {
            AbsListView.LayoutParams lp = new AbsListView.LayoutParams(
                ViewGroup.LayoutParams.MATCH_PARENT, 64);
            TextView textView = new TextView(
                ExpandableListViewProjectActivity.this);
            textView.setLayoutParams(lp);
            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 getView(int groupPosition, boolean isExpanded,
        View convertView, ViewGroup parent) {
        TextView textView = getGenericView();
        textView.setText(getGroup(groupPosition).toString());
        return textView;
    }

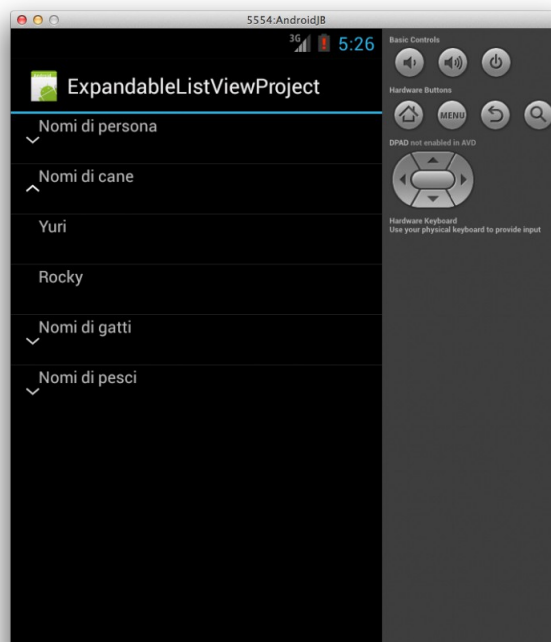
    public boolean isChildSelectable(int groupPosition, int childPosition) {
        return true;
    }

    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.

Esempio



WebView

Componente 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="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
    <WebView android:id="@+id/webview"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent" />
</LinearLayout>
```

L'uso è semplice e nell'esempio mostriamo come visualizzare la pagina principale di Google. Da notare che attiviamo **JavaScript**, cosa che non è molta consigliata per motivi di sicurezza.

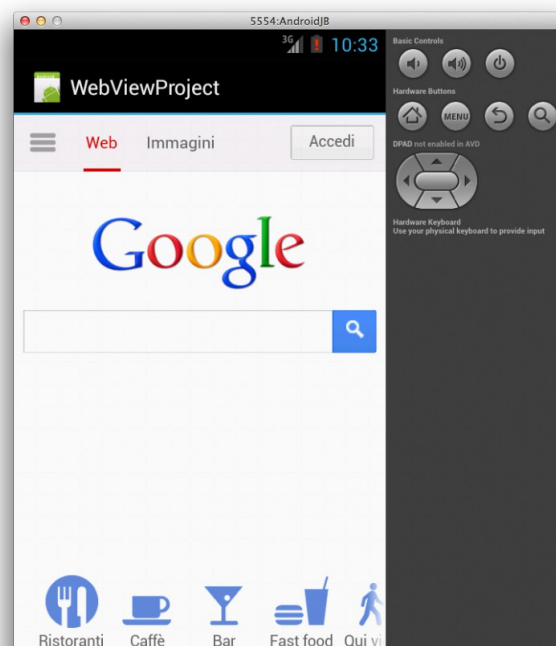
```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

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

Da non dimenticare di inserire nel **AndroidManifest.xml** il permesso ad accedere alla connessione ad internet, altrimenti l'applicazione andrà in errore.

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

Risultato



Notification

- Toast
- Notification
- MessageBox (AlertDialog)

Toast

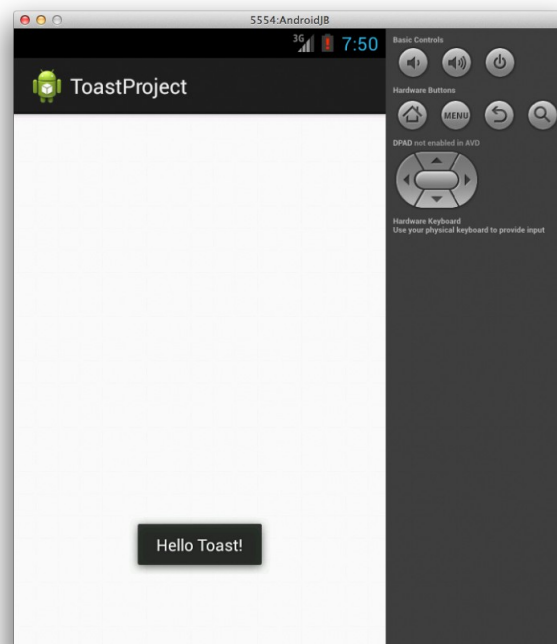
Il controllo **Toast** è un piccolo contenitore che visualizza un messaggio di testo. Adoperato spesso per notificare piccoli messaggi o per le segnalazioni di errori.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    Toast.makeText(getApplicationContext(), "Hello Toast!", Toast.LENGTH_LONG).show();
}
```

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

Esempio



Notification

In questo paragrafo vedremo come visualizzare una notifica nella **ActionBar**.

Il primo layout mostra l'activity principale, con un pulsante che, una volta cliccato, genererà la notifica.

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

    <Button android:id="@+id/btnnotifica"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/visualizza" />

</LinearLayout>
```

Nel codice, la parte importante è quella relativa al metodo `displayNotifica`, richiamato quando viene generato un click dal pulsante.

Il **PendingIntent** ci consentirà di comunicare con il servizio di notifica caricato su un oggetto **NotificationManager**.

Quest'ultimo, sarà utilizzato per inviare la notifica creata con il **Notification.Builder**, settando opportuni attributi come il titolo, l'oggetto, l'icona, ecc.

Al click sulla notifica verrà avviata l'Activity indicata nella prima Intent del metodo `displayNotifica`.

```
public class NotificationProjectActivity extends Activity {

    int ID_Notifica = 1;

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

        Button btnnotifica = (Button) findViewById(R.id.btnnotifica);
        btnnotifica.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                displayNotifica();
            }
        });
    }

    public void displayNotifica() {
        Intent Int_Notifica = new Intent(this, NotificationView.class);
        Int_Notifica.putExtra("ID_Notifica", ID_Notifica);
        PendingIntent pint = PendingIntent.getActivity(this, 0, Int_Notifica,
        PendingIntent.FLAG_CANCEL_CURRENT);
        NotificationManager nmanager = (NotificationManager) getSystemService(NOTIFICATION_SERVICE);
        Notification noti = new Notification.Builder(this)
            .setContentIntent(pint)
            .setWhen(System.currentTimeMillis())
            .setContentTitle("Notifica di test")
            .setContentText("Oggetto")
            .setAutoCancel(true)
            .setSmallIcon(R.drawable.ic_launcher)
            .build();
        nmanager.notify(ID_Notifica, noti);
    }
}
```

Il layout successivo rappresenta come si deve presentare la notifica.

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

```

<TextView android:id="@+id/txtnot"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" />
</LinearLayout>

```

L'Activity seguente, che viene avviata dopo l'azione sulla notifica, ricava l'id della stessa e tramite NotificationManager la cancella.

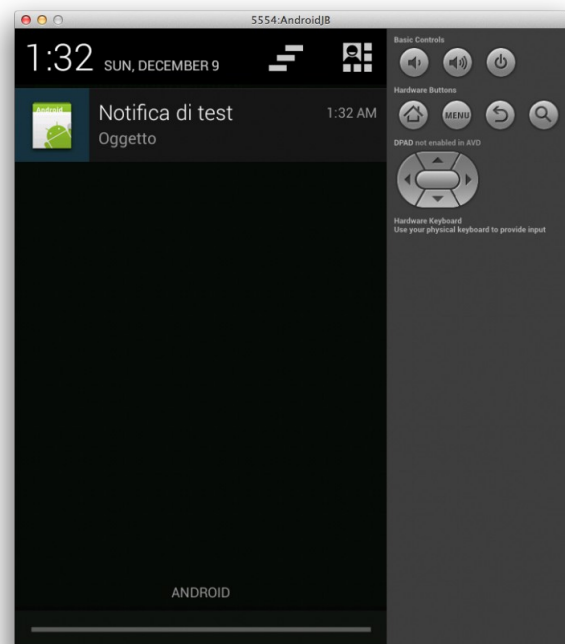
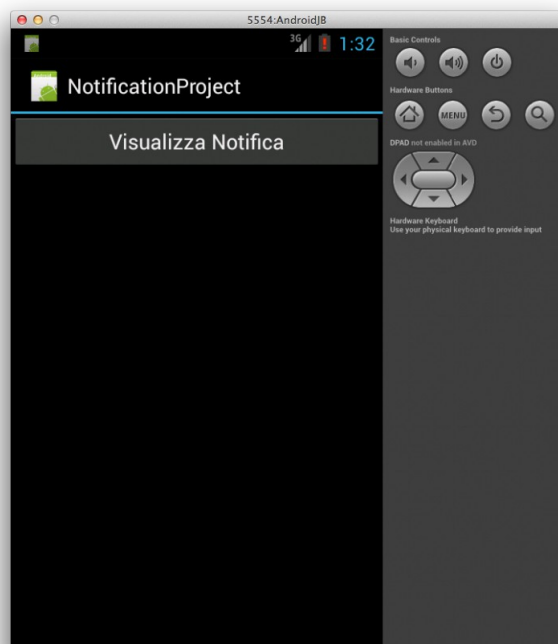
```

public class NotificationView extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.notification);

        NotificationManager nmanager = (NotificationManager)
getSystemService(NOTIFICATION_SERVICE);
        TextView txtres = (TextView) findViewById(R.id.txtnot);
        txtres.setText("ID Notifica " +
Integer.toString(getIntent().getExtras().getInt("ID_Notifica")));
        nmanager.cancel(getIntent().getExtras().getInt("ID_Notifica"));
    }
}

```

Risultato



MessageBox (AlertDialog)

Per creare la nostra buon vecchia **MessageBox** utilizzeremo il controllo **AlertDialog**. Il layout mostra solo quello che sarà l'Activity che lancerà la MessageBox e non il suo layout, che rimarrà quello standard (è anche possibile modificarlo).

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity" >

    <Button
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/clicca"
        android:onClick="Visualizza" />

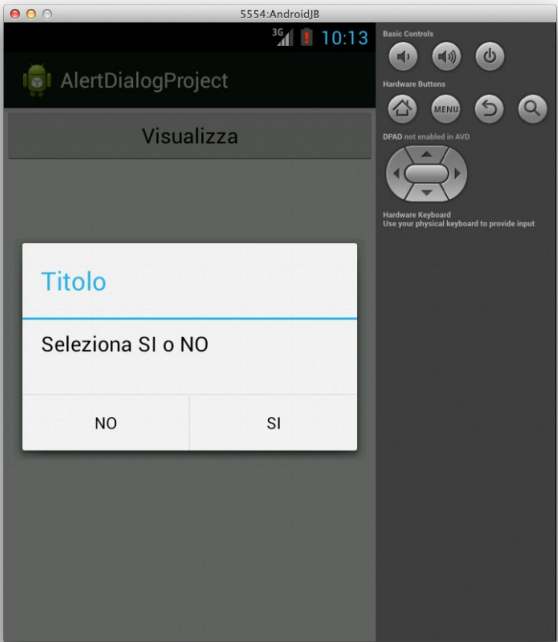
</RelativeLayout>
```

Alla pressione del pulsante, verrà richiamato il metodo Visualizza che provvederà a creare la MessageBox tramite **AlertDialog.Builder**, con le opportune proprietà, messaggio, titolo, pulsanti, ecc.

Successivamente tramite AlertDialog istanziamo e visualizziamo la MessageBox.

```
public void Visualizza(View view) {
    AlertDialog.Builder builder = new AlertDialog.Builder(context);
    builder.setTitle("Titolo")
        .setMessage("Seleziona SI o NO")
        .setCancelable(false)
        .setPositiveButton("SI", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                Toast.makeText(context, "Hai cliccato SI", Toast.LENGTH_SHORT).show();
            }
        })
        .setNegativeButton("NO", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                Toast.makeText(context, "Hai cliccato NO", Toast.LENGTH_SHORT).show();
            }
        });
    AlertDialog alert = builder.create();
    alert.show();
}
```

Risultato



LayoutManager

- LinearLayout
- TableLayout
- RelativeLayout
- FrameLayout
- Layout misto

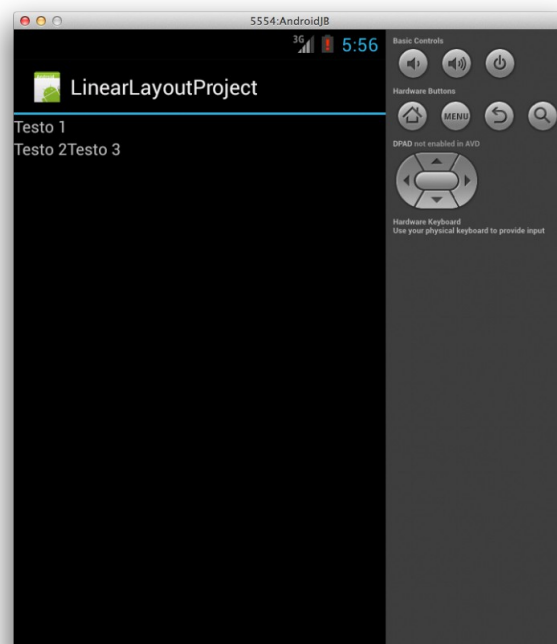
LinearLayout

Il **LinearLayout** è il layout manager più utilizzato, anche se non è più il layout di default per un progetto Android.

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

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
    <TextView android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/testo1" />
    <LinearLayout android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="horizontal" >
        <TextView android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/testo2" />
        <TextView android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/testo3" />
    </LinearLayout>
</LinearLayout>
```

Nell'esempio viene proposto un insieme di etichette di testo disposto su orientation vertical e horizontal annidate



Si può vedere facilmente che testo1 è disposta sulla prima riga e testo2 e testo3 (horizontal) sulla seconda riga.

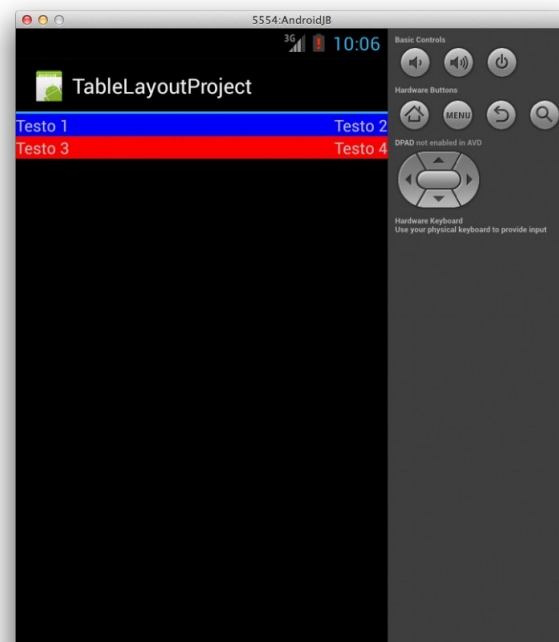
Dove testo1, testo2 e testo3 sono le etichette di testo.

TableLayout

Questo tipo di layout manager dispone i componenti in forma tabellare, dividendoli per 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>
```

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

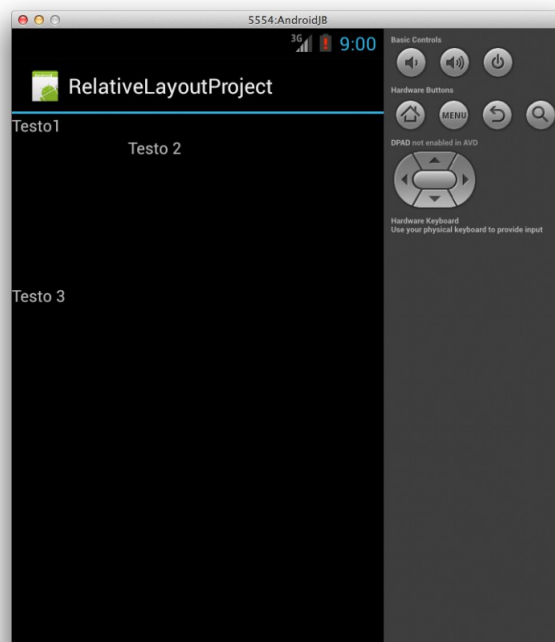


RelativeLayout

Dalla stessa parola, possiamo capire il funzionamento di questo layout manager. Esso predispone i controlli rispetto ad un controllo precedentemente segnalato. E' diventato il layout manager predefinito nei progetti Android, data la sua versatilità e maneggevolezza.

```
<?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="@string/testo1" />
    <TextView android:id="@+id/txt2"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/testo2"
        android:layout_below="@id/txt1"
        android:layout_marginLeft="100dp" />
    <TextView
        android:id="@+id/txt3"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_below="@id/txt2"
        android:layout_marginTop="108dp"
        android:text="@string/testo3" />
</RelativeLayout>
```

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 **leftMargin** e **marginTop** per spostare il controllo rispetto al riferimento prefissato.

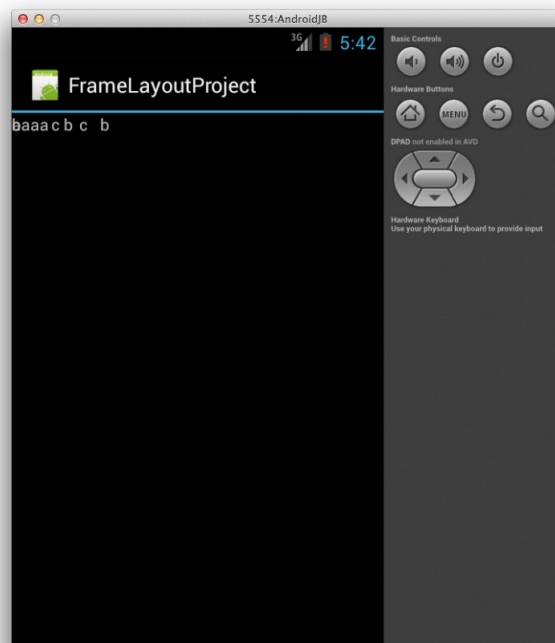


FrameLayout

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

```
<?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" />
</FrameLayout>
```

Nell'esempio visuale si nota che le varie etichette di testo vengono disposte una sopra l'altra rendendo il tutto illeggibile.



Layout misto

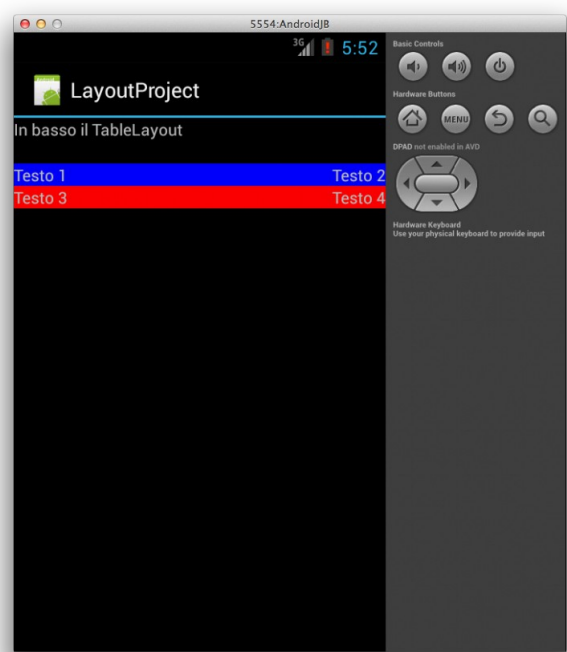
In questo esempio, ho unito quasi tutti i layout manager precedentemente illustrati, dando visione del fatto che è possibile annidarli anche se di natura diversa.

```
<?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" >
    <LinearLayout android:id="@+id/layoutlinear"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal" >
        <TextView android:text="In basso il TableLayout"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content" />
    </LinearLayout>
    <TableLayout android:id="@+id/layouttable"
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:stretchColumns="1"
        android:layout_below="@id/layoutlinear"
        android:layout_marginTop="30px" >
        <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>
```

Il codice non è molto complesso, anche se per layout ben più grandi di questo proposto, si possono utilizzare altre metodologie non illustrate in questa guida.

Dando una occhiata al codice, si nota che il layout manager principale è di tipo RelativeLayout ed al suo interno sono stati inglobati consecutivamente LinearLayout e TableLayout, quindi infondo non è poi così complicato questo esempio!

Risultato



Hardware

- Sensor
- Touch
- Multitouch

Sensor

Il sensore è un dispositivo hardware che converte una grandezza fisica, come la temperatura o la gravità, in un'informazione digitale.

Quello che faremo noi in questo esempio, è caricare la lista dei sensori e filtrarne solo uno e leggerne le informazioni.

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

    <TextView android:id = "@+id/txtsensore"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content" />

</LinearLayout>
```

Il layout contiene solo una TextView dove visualizzeremo le coordinate dell'accelerometro. Il codice non farà altro che visualizzare la lista dei sensori presenti sul dispositivo e successivamente filtrare l'accelerometro.

```
public class SensorProjectActivity extends Activity implements SensorEventListener {
    private SensorManager manager;
    private Sensor s_acc = null;
    private TextView txtsens;

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

        txtsens = (TextView) findViewById(R.id.txtsensore);

        manager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
        List<Sensor> listSensor = manager.getSensorList(Sensor.TYPE_ALL);
        for (Sensor stemp : listSensor) {
            Toast.makeText(this, stemp.getName(), Toast.LENGTH_LONG).show();
        }

        for (Sensor stemp : listSensor) {
            switch(stemp.getType()) {
                case Sensor.TYPE_ACCELEROMETER:
                    s_acc = stemp;
                    break;
            }
        }

        if(s_acc != null) {
            manager.registerListener(this, s_acc, SensorManager.SENSOR_DELAY_NORMAL);
        }
    }

    @Override
    public void onAccuracyChanged(Sensor arg0, int arg1) {
    }

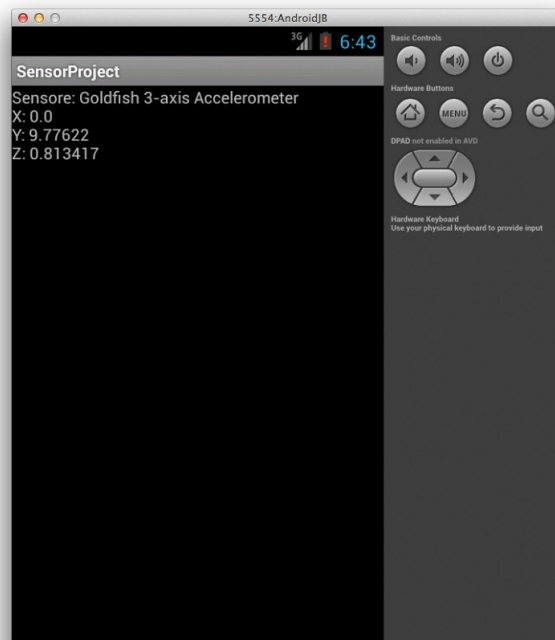
    @Override
    public void onSensorChanged(SensorEvent arg0) {
        float[] valori = arg0.values;
        String testo = "Sensore: " + arg0.sensor.getName() + "\nX: " + valori[0] +
            "\nY: " + valori[1] + "\nZ: " + valori[2];
        switch(arg0.sensor.getType()) {
            case Sensor.TYPE_ACCELEROMETER:
                txtsens.setText(testo);
                break;
        }
    }
}
```

Tramite l'oggetto **SensorManager** andiamo a richiamare tutti i sensori

`manager.getSensorList(Sensor.TYPE_ALL)` , è possibile anche richiamare solo un tipo di sensori cambiando l'argomento:

```
Sensor.TYPE_ACCELEROMETER, accelerometro  
Sensor.TYPE_GYROSCOPE, giroscopio  
Sensor.TYPE_LIGHT, luce  
Sensor.TYPE_MAGNETIC_FIELD, magnetico  
Sensor.TYPE_PRESSURE, pressione  
Sensor.TYPE_PROXIMITY, prossimità  
Sensor.TYPE_TEMPERATURE, temperatura  
Sensor.TYPE_ALL, tutti
```

Durante l'evento **onSensorChanged** andremo a filtrare il nostro sensore e visualizzeremo nella TextView le coordinate registrate.



Touch

Il touchscreen è un vero e proprio sensore che rileva la pressione di un oggetto sulla sua superficie. Senza perderci troppo in differenze quali resistivo e capacitivo, andiamo a vedere il nostro esempio, dove ci focalizziamo nell'intercettare la pressione di un oggetto su una porzione di schermo.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical"
    android:id="@+id/layout1" >

    <TextView android:id="@+id/txttdown"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/down" />
    <TextView android:id="@+id/txtup"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/up" />
    <TextView android:id="@+id/txtmove"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/move" />
    <LinearLayout android:id="@+id/layout2"
        android:layout_width="fill_parent"
        android:layout_height="100dp"
        android:background="#0000FF" ></LinearLayout>
</LinearLayout>
```

Nel layout andiamo a creare due zone con layout manager di tipo linear: la prima servirà a mostrarci le informazioni riguardo gli eventi di pressione, rilascio e movimento del sensore touch, la seconda sarà la porzione di schermo dove verranno intercettati gli eventi.

```
public class TouchProjectActivity extends Activity implements OnTouchListener {
    /** Called when the activity is first created. */
    TextView txttdown, txtup, txtmove;
    LinearLayout layout1, layout2;

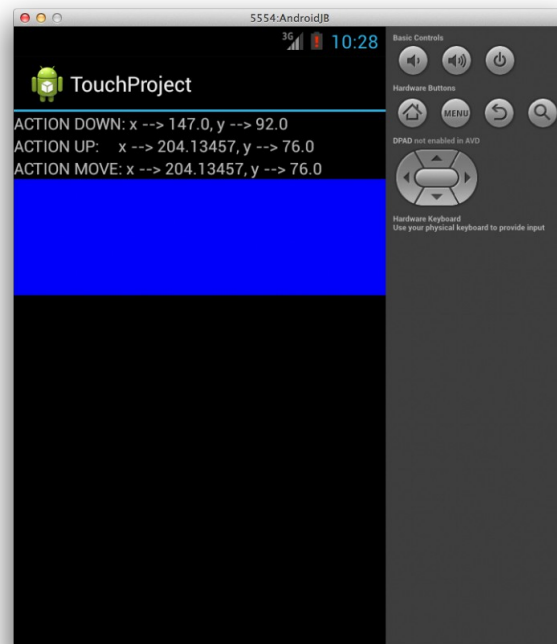
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        txttdown = (TextView) findViewById(R.id.txttdown);
        txtup = (TextView) findViewById(R.id.txtup);
        txtmove = (TextView) findViewById(R.id.txtmove);
        layout2 = (LinearLayout) findViewById(R.id.layout2);
        layout2.setOnTouchListener(this);
    }

    @Override
    public boolean onTouch(View v, MotionEvent event) {
        switch(event.getAction()) {
            case MotionEvent.ACTION_DOWN:
                txttdown.setText("ACTION DOWN: x --> " + event.getX() + ", y --> " +
event.getY());
                break;
            case MotionEvent.ACTION_UP:
                txtup.setText("ACTION UP:      x --> " + event.getX() + ", y --> " +
event.getY());
                break;
            case MotionEvent.ACTION_MOVE:
                txtmove.setText("ACTION MOVE: x --> " + event.getX() + ", y --> " +
event.getY());
                break;
        }
        return true;
    }
}
```

Come vediamo dall'intestazione della Activity, andiamo ad implementare il listener **OnTouchListener**, che ci permetterà di gestire gli eventi **ACTION_UP** (rilascio), **ACTION_DOWN** (pressione) e **ACTION_MOVE** (movimento) generati. Il listener sarà assegnato solo al secondo layout, come si vede dalla riga `layout2.setOnTouchListener(this);`

In sostanza, intercetteremo l'evento **onTouch** ed andremo a filtrare, secondo le azioni fatte, e visualizzare le relative coordinate dello schermo dove è avvenuto l'evento.

I metodi **getX()** e **getY()** ci serviranno per ricavare le coordinate.



Multitouch

La differenza nel rilevare uno o più tocchi sullo schermo è quasi irrisoria.
Basta ricavare il numero di punti e richiamare le coordinate tramite indice.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical"
    android:id="@+id/layout1" >

    <TextView android:id="@+id/txtmove1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="ACTION MOVE 1: " />
    <TextView android:id="@+id/txtmove2"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="ACTION MOVE 2: " />

</LinearLayout>
```

Il layout, come al solito, ci serve solo per mostrare informazioni.
Per semplicità l'evento onTouch è rilevato in tutto lo schermo.

```
public class MultiTouchProjectActivity extends Activity implements OnTouchListener {
    /** Called when the activity is first created. */
    TextView txtmove1, txtmove2;
    LinearLayout layout1;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        txtmove1 = (TextView) findViewById(R.id.txtmove1);
        txtmove2 = (TextView) findViewById(R.id.txtmove2);
        layout1 = (LinearLayout) findViewById(R.id.layout1);
        layout1.setOnTouchListener(this);
    }

    @Override
    public boolean onTouch(View v, MotionEvent event) {
        switch(event.getAction()) {
            case MotionEvent.ACTION_MOVE:
                txtmove1.setText("ACTION MOVE 1: x --> " + event.getX(0) + ", y --> " +
event.getY(0));
                if(event.getPointerCount() > 1)
                    txtmove2.setText("ACTION MOVE 2: x --> " + event.getX(1) + ", y --> " +
+ event.getY(1));
                break;
        }
        return true;
    }
}
```

Le parti di codice interessanti sono `event.getPointerCount()` che ci dice quanti tocchi ha lo schermo e le solite `getX()` e `getY()` ci serviranno a ricavare le coordinate ma, stavolta, dobbiamo passare l'indice del punto dello schermo da ricavare.

Risultato



Miscellanea, parte 1

- File
- Mail
- Menu
- Tab ActionBar
- MediaPlayer

File

Su Android, la gestione del file system è la stessa di una classica applicazione Java. Quindi, verranno utilizzate gli oggetti **FileOutputStream** e **OutputStreamWriter** per scrivere, **FileInputStream** e **InputStreamReader** per leggere.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/scrivi" />
    <EditText android:id="@+id/txtFile"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:inputType="text" />
    <Button android:id="@+id/btnCarica"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/carica" />
    <Button android:id="@+id/btnSalva"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/salva" />
    <Button android:id="@+id/btnCaricaSD"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/caricaSD" />
    <Button android:id="@+id/btnSalvaSD"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/salvaSD" />
</LinearLayout>
```

Il layout è composto da una casella di testo dove andremo ad inserire una stringa da memorizzare o che viene letta da file, ed i pulsanti relativi al caricamento e salvataggio del file su memoria interna ed esterna (SD).

```
public class FileProjectActivity extends Activity implements OnClickListener {
    /** Called when the activity is first created. */
    final String FILENAME = "fileprova.txt";
    final String MY_PATH = "/FileProject";
    final int MAX_BUF = 1024;
    EditText txtFile;

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

        txtFile = (EditText) findViewById(R.id.txtFile);
        Button btnCarica = (Button) findViewById(R.id.btnCarica);
        btnCarica.setOnClickListener(this);
        Button btnSalva = (Button) findViewById(R.id.btnSalva);
        btnSalva.setOnClickListener(this);
        Button btnCaricaSD = (Button) findViewById(R.id.btnCaricaSD);
        btnCaricaSD.setOnClickListener(this);
        Button btnSalvaSD = (Button) findViewById(R.id.btnSalvaSD);
        btnSalvaSD.setOnClickListener(this);
    }

    public boolean caricaFile(String pathname) {
        try {
            int read;
            String readS = "";
            FileInputStream fis = openFileInput(pathname);
```

```

        InputStreamReader isr = new InputStreamReader(fis);
        char[] buffer = new char[MAX_BUF];
        while((read = isr.read(buffer)) > 0) {
            readS += String.valueOf(buffer, 0, read);
            buffer = new char[MAX_BUF];
        }
        txtFile.setText(readS);
        isr.close();
    } catch (Exception e) {
        Toast.makeText(this, e.getMessage(), Toast.LENGTH_SHORT).show();
        return false;
    }
}

return true;
}

public boolean salvaFile(String pathname) {
    try {
        FileOutputStream fos = openFileOutput(pathname, MODE_PRIVATE);
        OutputStreamWriter osw = new OutputStreamWriter(fos);
        osw.write(txtFile.getText().toString());
        osw.flush();
        osw.close();
        txtFile.setText("");
    } catch (Exception e) {
        Toast.makeText(this, e.getMessage(), Toast.LENGTH_SHORT).show();
        return false;
    }
    return true;
}

public boolean caricaFileSD(String pathname) {
    try {
        int read;
        String readS = "";
        File pathsd = Environment.getExternalStorageDirectory();
        File directory = new File(pathsd.getAbsolutePath() + MY_PATH);
        File file = new File(directory, pathname);
        FileInputStream fis = new FileInputStream(file);
        InputStreamReader isr = new InputStreamReader(fis);
        char[] buffer = new char[MAX_BUF];
        while((read = isr.read(buffer)) > 0) {
            readS += String.valueOf(buffer, 0, read);
            buffer = new char[MAX_BUF];
        }
        txtFile.setText(readS);
        isr.close();
    } catch (Exception e) {
        Toast.makeText(this, e.getMessage(), Toast.LENGTH_SHORT).show();
        return false;
    }
    return true;
}

public boolean salvaFileSD(String pathname) {
    try {
        File pathsd = Environment.getExternalStorageDirectory();
        File directory = new File(pathsd.getAbsolutePath() + MY_PATH);
        if(!directory.exists())
            directory.mkdir();
        File file = new File(directory, pathname);
        FileOutputStream fos = new FileOutputStream(file);
        OutputStreamWriter osw = new OutputStreamWriter(fos);
        osw.write(txtFile.getText().toString());
        osw.flush();
        osw.close();
        txtFile.setText("");
    } catch (Exception e) {
        Toast.makeText(this, e.getMessage(), Toast.LENGTH_SHORT).show();
        return false;
    }
    return true;
}

@Override
public void onClick(View arg0) {
    switch(arg0.getId()) {
        case R.id.btnCarica:
            caricaFile(FILENAME);
            break;
        case R.id.btnSalva:

```

```

        salvaFile(FILENAME);
        break;
    case R.id.btnCaricaSD:
        caricaFileSD(FILENAME);
        break;
    case R.id.btnSalvaSD:
        salvaFileSD(FILENAME);
        break;
    }
}
}

```

I metodi cruciali sono **caricaFile**, **salvaFile**, **caricaFileSD** e **salvaFileSD**.

Scrivere o leggere da memoria interna o esterno non cambia il codice, tranne per il fatto di andarsi a ricavare il percorso della memoria esterna (`Environment.getExternalStorageDirectory()`).

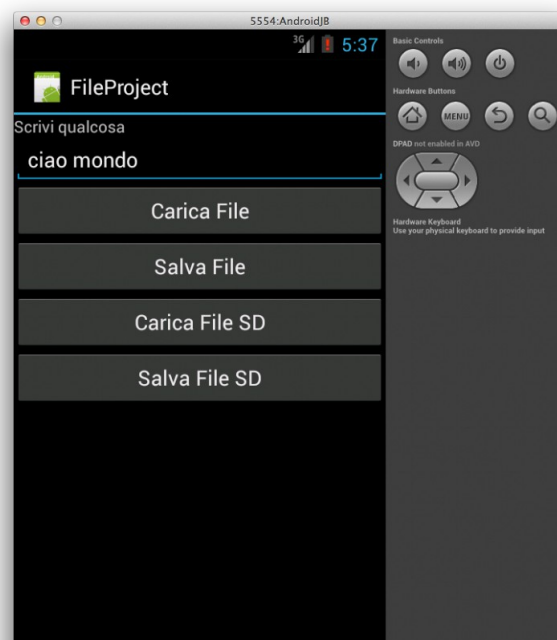
`caricaFile` e `caricaFileSD` tramite `InputStreamReader` leggono il contenuto del file (byte) e lo convertono in formato stringa, così lo possiamo visualizzare senza problemi nella nostra casella di testo.

`salvaFile` e `salvaFileSD` fanno l'operazione inversa rispetto le precedenti, ovvero tramite `OutputStreamWriter` scrivono la stringa presente nella casella di testo su file.

Necessitiamo sempre di un permesso, in questo caso `android.permission.WRITE_EXTERNAL_STORAGE` per scrivere su memoria esterna.

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

Risultato



Mail

Analizziamo come inviare una semplice mail, sfruttando i gestori di posta elettronica interni al nostro sistema operativo.

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

    <EditText android:id="@+id/txtdest"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:hint="@string/dest" />
    <EditText android:id="@+id/txtogg"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:hint="@string/ogg" />
    <EditText android:id="@+id/txttesto"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:hint="@string/mess" />
    <Button android:id="@+id/btninvia"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/invia" />

</LinearLayout>
```

Il layout è composto dal minimo indispensabile per inviare una mail: destinatario, oggetto e testo.

```
public class MailProjectActivity extends Activity implements OnClickListener {
    /** Called when the activity is first created. */
    EditText txtdest, txtogg, txttesto;

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

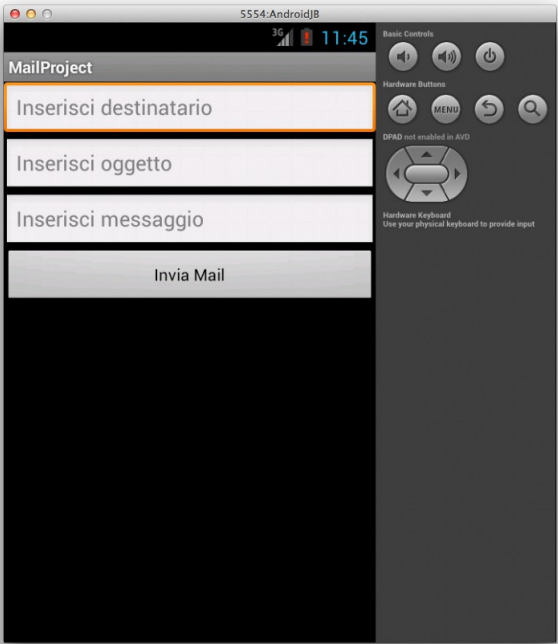
        txtdest = (EditText) findViewById(R.id.txtdest);
        txtogg = (EditText) findViewById(R.id.txtogg);
        txttesto = (EditText) findViewById(R.id.txttesto);
        Button btninvia = (Button) findViewById(R.id.btninvia);
        btninvia.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        Intent imail = new Intent(Intent.ACTION_SEND);
        imail.putExtra(Intent.EXTRA_EMAIL, new String[] {txtdest.getText().toString()});
        imail.putExtra(Intent.EXTRA_SUBJECT, txtogg.getText().toString());
        imail.putExtra(Intent.EXTRA_TEXT, txttesto.getText().toString());
        imail.setType("message/rfc822");
        startActivity(Intent.createChooser(imail, "Email"));
    }
}
```

Il metodo onClick, generato alla pressione del tasto di invio, non farà altro che prendere i dati inseriti dall'utente e passarli come parametri ad una applicazione esterna.

Quest'ultima non è altro che un gestore di posta elettronica e che provvederà in automatico a caricare i dati ed a darci la possibilità di inviare la mail.

Risultato



Menu

Nel sistema operativo Android è possibile utilizzare due tipi di menu: **Context Menu** e **Option Menu**.

L'Option Menu non è altro che il menu dell'applicazione ed è possibile utilizzarlo cliccando il tasto in alto a destra del nostro sistema o il tasto menu.

Il Context Menu è il menu generato da un controllo facendo pressione continua per qualche secondo su di esso.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
    <TextView android:id="@+id/txtmenuoption"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/menu1" />
    <TextView android:id="@+id/txtmenucontext"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/scegli" />
</LinearLayout>
```

Nel layout sono presenti due etichette di testo che serviranno a visualizzare la scelta fatta tramite i due tipi di menu.

```
public class MenuProjectActivity extends Activity {
    TextView txtop;
    TextView txtco;

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

        txtop = (TextView) findViewById(R.id.txtmenuoption);
        txtco = (TextView) findViewById(R.id.txtmenucontext);
        txtco.setOnCreateContextMenuListener(this);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        MenuInflater inflater = getMenuInflater();
        inflater.inflate(R.menu.menu_op, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        txtop.setText(item.getTitle());
        return true;
    }

    @Override
    public void onCreateContextMenu(ContextMenu menu, View view, ContextMenuInfo menuInfo) {
        MenuInflater inflater = getMenuInflater();
        inflater.inflate(R.menu.menu_co, menu);
    }

    @Override
    public boolean onContextItemSelected(MenuItem item) {
        txtco.setText(item.getTitle());
        return true;
    }
}
```

Per creare i menu basta interagire con i metodi **onCreateContextMenu** e **onCreateOptionsMenu**. Tramite l'oggetto **MenuInflater**, andremo a caricare il nostro menu creato tramite risorse e lo

assegnerà all'oggetto menu predestinato tramite il metodo **inflate** (contesto o controllo).

Quando viene richiamato onCreateOptionsMenu il menu viene assegnato indirettamente all'activity, invece, per il menu del controllo, bisogna indicarlo esplicitamente durante il metodo onCreate:

```
txtco.setOnCreateContextMenuListener(this);
```

La precedente riga non fa altro che assegnare al controllo etichetta di testo il menu di tipo Context. Si è scelto il listener interno (this), nel caso di più elementi con diversi menu, bisognerà filtrare secondo i controlli, sia per la creazione sia per gli elementi.

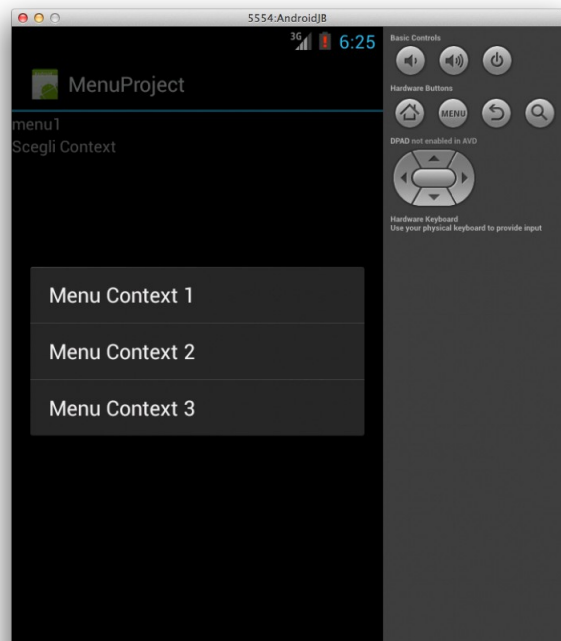
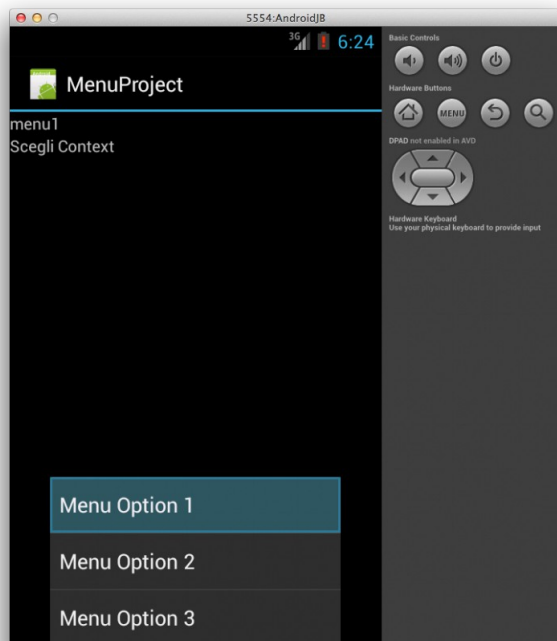
In alternativa è possibile utilizzare dei listener anonimi creati ad-hoc.

Tramite i file di risorse posti su /res/menu/ andiamo a creare gli elementi contenuti nei due menu. Il risultato è qualcosa di simile:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
    <item android:id="@+id/item1" android:title="Menu Context 1"></item>
    <item android:id="@+id/item2" android:title="Menu Context 2"></item>
    <item android:id="@+id/item3" android:title="Menu Context 3"></item>
</menu>
```

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
    <item android:id="@+id/item1" android:title="Menu Option 1"></item>
    <item android:id="@+id/item2" android:title="Menu Option 2"></item>
    <item android:id="@+id/item3" android:title="Menu Option 3"></item>
</menu>
```

Risultato



Tab ActionBar

La nostra ActionBar, oltre alla possibilità di inserire pulsanti, etichette, menu e varie altri componenti, possiamo impostarla per proporre un aspetto tabellare alla nostra applicazione. Questo tipo di impostazione, sostituisce quella precedente basata sulle **TabActivity** ormai deprecate dalla versione 13 delle API.

Lasciamo vuoto il layout dell'activity principale

```
<?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>
```

Impostiamo, invece, due layout per le nostre pagine, ecco il primo...

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <TextView android:id="@+id/txttab1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:text="@string/ttab1" />
</LinearLayout>
```

...ed il secondo

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <TextView android:id="@+id/txttab2"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:text="@string/ttab2" />
</LinearLayout>
```

Durante l'evento onCreate della nostra activity, andremo a caricare la nostra ActionBar e la imposteremo per la navigazione a pagine.

Successivamente, inseriremo le nostre pagine caricandole attraverso i **Fragment**, ovvero porzioni di interfaccia grafica in una Activity.

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

        ActionBar bar = getActionBar();
        bar.setNavigationMode(ActionBar.NAVIGATION_MODE_TABS);

        Tab tab1 = bar.newTab();
        tab1.setText("Tab 1");
        tab1.setTabListener(new TabListener<Tab1Fragment>(this, "Tag Tab 1", Tab1Fragment.class));
        bar.addTab(tab1);
        Tab tab2 = bar.newTab();
```

```

        tab2.setText("Tab 2");
        tab2.setTabListener(new TabListener<Tab2Fragment>(this, "Tag Tab 2", Tab2Fragment.class));
        bar.addTab(tab2);
    }

    private class TabListener<T extends Fragment> implements ActionBar.TabListener {

        private Fragment mFrag;
        private Activity mAct;
        private String mTag;
        private Class<T> mClass;

        public TabListener(Activity activity, String tag, Class<T> cls) {
            mAct = activity;
            mTag = tag;
            mClass = cls;
        }

        @Override
        public void onTabReselected(Tab tab, FragmentTransaction ft) {

        }

        @Override
        public void onTabSelected(Tab tab, FragmentTransaction ft) {
            if(mFrag == null) {
                mFrag = Fragment.instantiate(mAct, mClass.getName());
                ft.add(android.R.id.content, mFrag, mTag);
            } else {
                ft.attach(mFrag);
            }
        }

        @Override
        public void onTabUnselected(Tab tab, FragmentTransaction ft) {
            if(mFrag != null) {
                ft.detach(mFrag);
            }
        }

    }
}

```

Il TabListener serve ad intercettare gli eventi relativi alla selezione, riselezione e deselezione di una pagina e serviranno per caricare i nostri Fragment.

Le due successive classi rappresentano le due pagine vere e proprie, ciascuna con il proprio layout opportunamente caricato tramite **LayoutInflater**.

```

public class Tab1Fragment extends Fragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
        return inflater.inflate(R.layout.tab1, container, false);
    }
}

public class Tab2Fragment extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
        return inflater.inflate(R.layout.tab2, container, false);
    }
}

```

Risultato



MediaPlayer

Il controllo **MediaPlayer** permette di manipolare flussi audio/video.

Nel nostro esempio ci occuperemo solo di avviare e stoppare un file audio di tipo MP3.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity" >
    <Button android:id="@+id/btnstart"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/start"
        android:onClick="startMusic" />
    <Button android:id="@+id/btnstop"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/stop"
        android:onClick="stopMusic" />
</LinearLayout>
```

Il layout si compone semplicemente di due pulsanti, atti allo start e stop del file audio.

```
public class MainActivity extends Activity {

    MediaPlayer player;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        player = null;

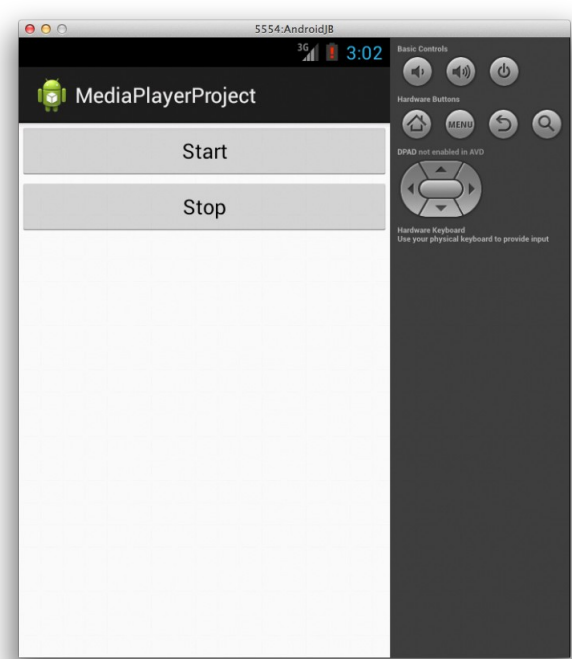
        public void startMusic(View view) {
            stopMusic(view);
            player = MediaPlayer.create(this, R.raw.capetown);
            player.start();
        }

        public void stopMusic(View view) {
            if(player != null)
                player.stop();
        }

    }
}
```

Il codice è abbastanza semplice e non si fa altro che, nel metodo **startMusic**, creare il controllo **MediaPlayer**, passando come parametro il file da riprodurre, ed avviare la riproduzione, e nel metodo **stopMusic** di fermare la riproduzione.

Esempio



SMS e chiamate

- Invio SMS
- Lista cartelle SMS
- Invio chiamata
- Intercettare chiamata

Invio SMS

In questo esempio, vedremo come inviare un semplice sms.

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

    <EditText android:id="@+id/txtnumber"
        android:layout_width="fill_parent"
        android:inputType="number"
        android:layout_height="wrap_content"
        android:hint="@string/dest" />
    <EditText android:id="@+id/txtttesto"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:hint="@string/testo" />
    <Button android:id="@+id/btninvia"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/invia" />

</LinearLayout>
```

Il layout servirà per inserire destinatario, un messaggio e il pulsante di invio.

```
public class SendSMSProjectActivity extends Activity implements OnClickListener {

    EditText txtnumber;
    EditText txtttesto;

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

        txtnumber = (EditText) findViewById(R.id.txtnumber);
        txtttesto = (EditText) findViewById(R.id.txtttesto);
        Button btninvia = (Button) findViewById(R.id.btninvia);
        btninvia.setOnClickListener(this);
    }

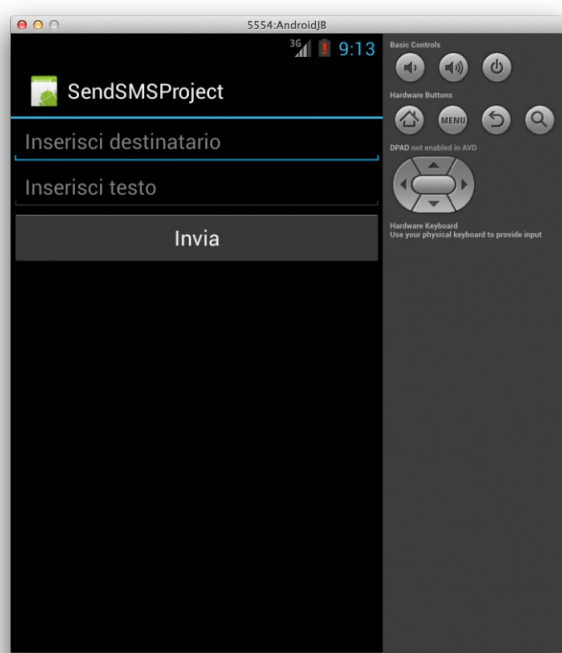
    @Override
    public void onClick(View v) {
        SmsManager smanager = SmsManager.getDefault();
        smanager.sendTextMessage(txtnumber.getText().toString(), null,
txtttesto.getText().toString(), null, null);
    }
}
```

Alla pressione del pulsante, caricheremo **SmsManager** del nostro dispositivo ed invieremo un sms tramite il metodo **sendTextMessage**, passando gli opportuni parametri.

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

Da non dimenticare di inserire il permesso di inviare sms nel file AndroidManifest.xml.

Esempio



Lista cartelle SMS

In questo esempio, vedremo come elencare i nostri sms contenuti in memoria.

Di seguito il layout dell'activity, che conterrà una lista di cartelle sms selezionabili e la lista degli sms letti.

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

Il successivo layout servirà a mostrare gli elementi della lista dell'activity. Ogni elemento conterrà il numero come intestazione seguito dal testo del sms.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <TextView android:id="@+id/txtnumero"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:textColor="#FFFF00" />
    <TextView android:id="@+id/txtsms"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content" />
</LinearLayout>
```

La parte cruciale del codice è il metodo **AggiornaLista** che provvederà, tramite un cursore, ad effettuare un'interrogazione nel database degli sms e recuperare quelli solo che sono stati selezionati.

```
public class SMSFolderProjectActivity extends Activity {

    String[] valori_folder = {"content://sms/", "content://sms/inbox", "content://sms/sent"};

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

        Spinner folder = (Spinner) findViewById(R.id.folder);
        ArrayAdapter<CharSequence> aadapter = ArrayAdapter.createFromResource(this,
R.array.foldersms, android.R.layout.simple_spinner_item);
        aadapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
        folder.setAdapter(aadapter);
        folder.setOnItemClickListener(new OnItemSelectedListener() {

            @Override
            public void onItemSelected(AdapterView<?> arg0, View arg1,
                int arg2, long arg3) {
                AggiornaLista(arg2);
            }

            @Override
            public void onNothingSelected(AdapterView<?> arg0) {

            }

        });
    }
}
```

```

private void AggiornaLista(int id) {
    ListView lista = (ListView) findViewById(R.id.listsms);
    lista.setAdapter(null);
    String[] colonne = new String[] {"address", "body", "_id"};
    int[] nomi = new int[] {R.id.txtnumero, R.id.txtsms};

    Cursor c;

    ContentResolver cr = this.getContentResolver();
    c = cr.query(Uri.parse(valori_folder[id]), colonne, null, null, null);

    ListAdapter adapter = new SimpleCursorAdapter(this, R.layout.sms_layout, c, colonne, nomi,
Adapter.NO_SELECTION);
    lista.setAdapter(adapter);

    c.close();
}
}

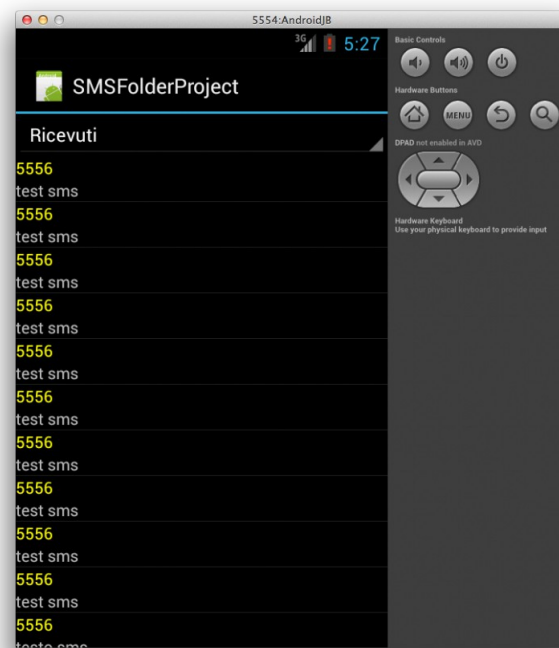
```

Il **Cursor** è un controllo che serve per manipolare una base di dati, come quella degli sms o della rubrica.

Ricordarsi di includere i permessi per leggere gli sms.

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

Risultato



Effettuare Chiamata

Per effettuare una chiamata utilizzeremo banalmente l'applicazione interna offerta dal nostro dispositivo.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity" >
    <EditText android:id="@+id/txtnumero"
        android:layout_height="wrap_content"
        android:layout_width="fill_parent"
        android:hint="@string/numero" />
    <Button android:id="@+id/btncall"
        android:layout_height="wrap_content"
        android:layout_width="fill_parent"
        android:text="@string/call"
        android:onClick="Chiama" />
</LinearLayout>
```

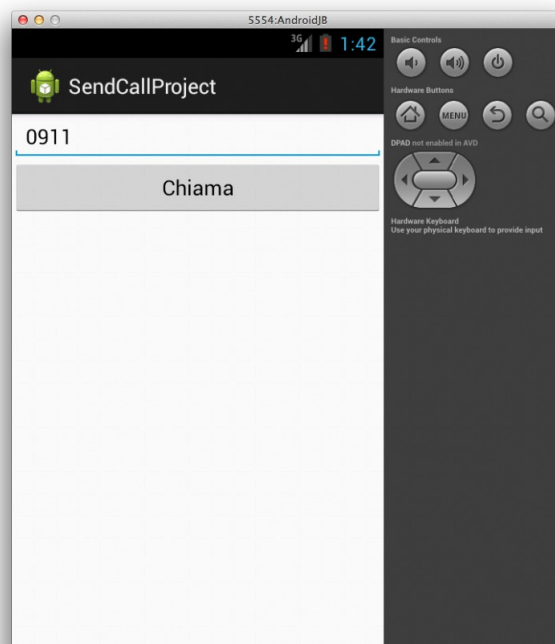
La parte importante dell'esempio è il metodo **Chiama**: ricava il numero dal campo di testo ed effettua la chiamata tramite l'applicazione interna del sistema.

```
public void Chiama(View view) {
    EditText txtnumero = (EditText) this.findViewById(R.id.txtnumero);
    Intent intent = new Intent(Intent.ACTION_CALL);
    intent.setData(Uri.parse("tel:" + txtnumero.getText().toString()));
    startActivity(intent);
}
```

Inserire, naturalmente, il permesso per poter effettuare la chiamata.

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

Esempio



Intercettare chiamata

Per intercettare una chiamata dobbiamo utilizzare sempre il solito listener, in questo caso

PhoneStateListener

Come dice il nome, servirà ad intercettare un cambiamento nello stato del telefono.

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

    <TextView android:id="@+id/txtcall"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="" />

</LinearLayout>
```

Il layout, sempre di appoggio, è composto da una etichetta di testo per visualizzare il numero della chiamata ricevuta.

```
public class CallProjectActivity extends Activity {

    TextView txtcall;

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

        txtcall = (TextView) findViewById(R.id.txtcall);
        TelephonyManager tmanager = (TelephonyManager) getSystemService(Context.TELEPHONY_SERVICE);
        tmanager.listen(new CallListener(), CallListener.LISTEN_CALL_STATE);
    }

    class CallListener extends PhoneStateListener {
        @Override
        public void onCallStateChanged(int state, String number) {
            super.onCallStateChanged(state, number);
            switch(state) {
                case TelephonyManager.CALL_STATE_RINGING:
                    txtcall.setText("Chiamata ricevuta da " + number);
            }
        }
    }
}
```

La parte di codice di analizzare è quella relativa al **CallListener**, un oggetto **PhoneStateListener**, che servirà ad intercettare, tramite l'evento **onCallStateChanged**, se avviene un cambiamento di stato.

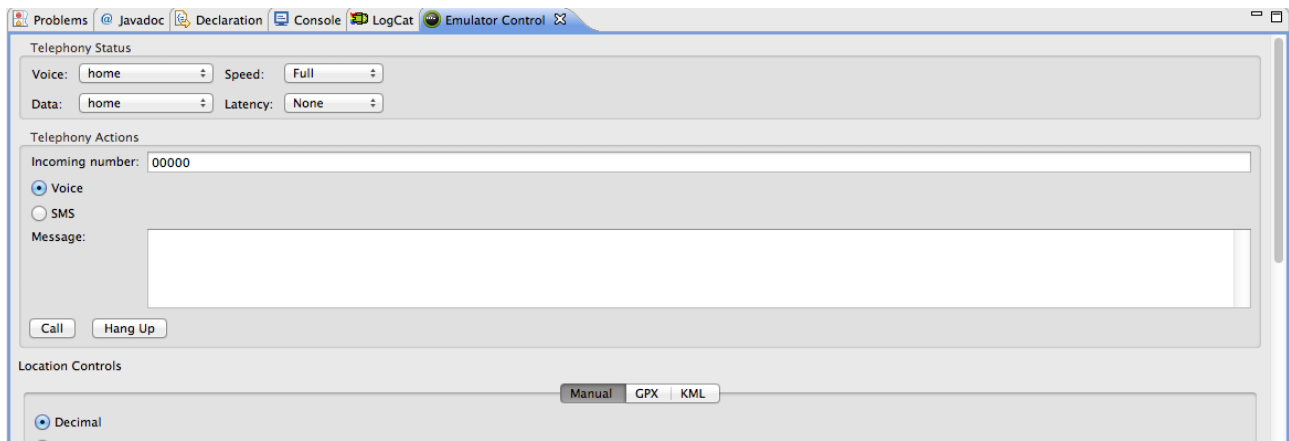
Quando verrà intercettato uno stato **CALL_STATE_RINGING**, significa che stiamo ricevendo una chiamata.

Il numero della chiamata è argomento dell'evento **onCallStateChanged** e lo visualizzeremo a video.

Anche qui ci vorrà un permesso per poter leggere lo stato del telefono.

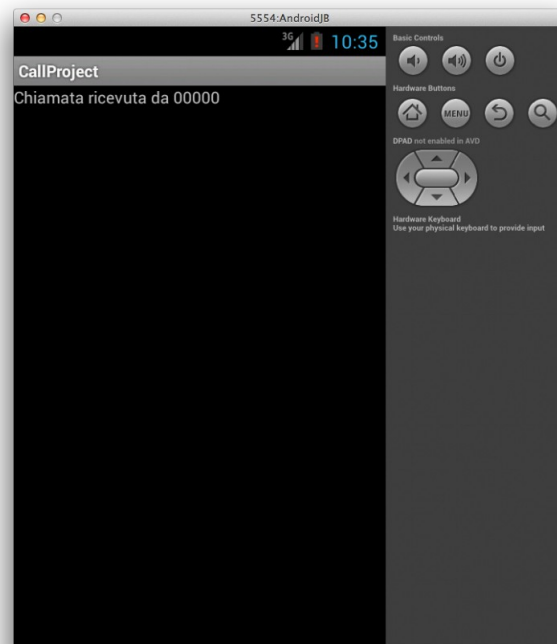
```
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
```

Una particolarità del plugin Android dell'ambiente Eclipse è quella di poter effettuare finte chiamate o inviare finti sms, con **EmulatorControl**, in modo da poter testare con facilità le applicazioni sulla macchina virtuale Android.



Se non è presente, basta andarlo a selezionare da Windows → Show View → Other.
Nella lista che comparirà ci sarà il suddetto componente.

Esempio



Concetti avanzati

- Service
- Threading

Service

Un **Service** è un insieme di operazioni, a corta o lunga durata, che vengono eseguite da una applicazione senza necessità dell'iterazione con l'utente.

Utilizzato spesso per espletare funzioni autonome con la possibilità di iterazione attraverso un applicativo di appoggio.

Nel nostro layout sono stati predisposti due pulsanti che serviranno ad avviare e arrestare il servizio.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
    <Button android:id="@+id/btnstart"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/start" />
    <Button android:id="@+id/btnstop"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/stop" />
</LinearLayout>
```

I metodi per avviare e arrestare un servizio, fanno parte della classe Activity e sono rispettivamente **startService** e **stopService**.

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

        Button btnstart = (Button) findViewById(R.id.btnstart);
        btnstart.setOnClickListener(this);
        Button btnstop = (Button) findViewById(R.id.btnstop);
        btnstop.setOnClickListener(this);
    }

    @Override
    public void onClick(View arg0) {
        switch(arg0.getId()) {
            case R.id.btnstart:
                startService(new Intent(getApplicationContext(), ServiceTest.class));
                break;
            case R.id.btnstop:
                stopService(new Intent(getApplicationContext(), ServiceTest.class));
                break;
        }
    }
}
```

L'oggetto servizio che andremo a creare è molto banale ed il suo compito è solo indicarci con una notifica, quando il servizio viene avviato e quando viene distrutto.

L'evento che viene generato quando viene avviato il servizio è **onStartCommand**, quando viene distrutto è **onDestroy**.

```
public class ServiceTest extends Service {

    public ServiceTest() {

    }

    @Override
```

```

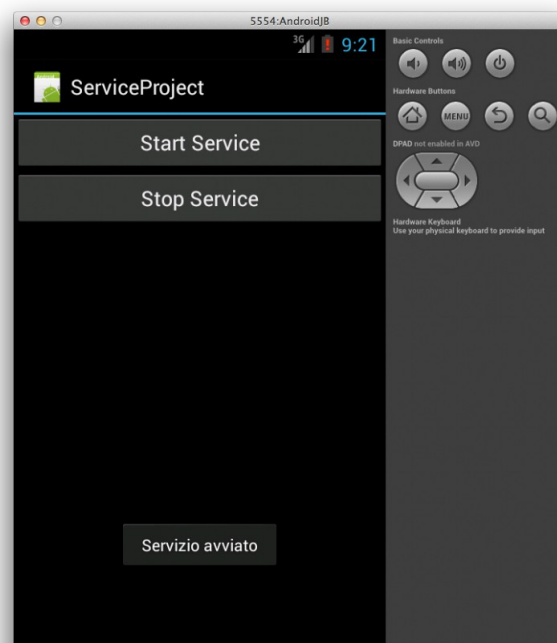
    public IBinder onBind(Intent intent) {
        return null;
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startID) {
        Toast.makeText(this, "Servizio avviato", Toast.LENGTH_LONG).show();
        return START_STICKY;
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        Toast.makeText(this, "Servizio distrutto", Toast.LENGTH_LONG).show();
    }
}

```

Risultato



Threading

L'argomento **Threading** è molto vasto, qui ci occuperemo di vedere in maniera superficiale alcuni dei costrutti che ci offre il nostro sistema Android.

Utilizzeremo tre modi per effettuare thread: **AsyncTask**, **Runnable** e **Handle**.

La nostra Activity sarà composta da tre pulsanti, ciascuno eseguirà uno dei tre metodi sopracitati.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
    <Button android:onClick="onClickAsyncTask"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/async" />

    <Button
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:onClick="onClickRunnable"
        android:text="@string/run" />

    <Button android:onClick="onClickHandle"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/han"
        android:id="@+id/btnh" />
</LinearLayout>
```

Quando sarà selezionato AsyncTask verrà creata una istanza della nostra classe **TestAsyncTask**, eseguirà un processo parallelo che provvederà ad aggiornare il suo stato di avanzamento ed a visualizzarlo a video.

Alla selezione di Runnable, verrà creata una classe anonima interna (classe senza identificatore) che provvederà anch'essa ad aggiornare il suo stato ed a visualizzarlo a video.

L'oggetto Handler provvederà a lanciare un messaggio, nella coda dei messaggi del thread corrente, che modificherà lo stato del pulsante.

Utilizzato solitamente in combinazione con oggetti Runnable, in quanto potrebbe bloccare il thread principale.

```
public class ThreadingProjectActivity extends Activity {
    Button btnh;

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

        btnh = (Button) findViewById(R.id.btnh);
    }

    public void onClickAsyncTask(View view) {
        TestAsyncTask task = new TestAsyncTask(getApplicationContext());
        task.execute("Valore 1", "Valore 2", "Valore 3");
    }

    public void onClickRunnable(View view) {
        final Button btr = (Button)view;
        new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    btr.post(new Runnable() {
```

```

        @Override
        public void run() {
            btnr.setText("Esecuzione...");
        }
    });
    Thread.sleep(3000);
    btnr.post(new Runnable() {
        @Override
        public void run() {
            btnr.setText("Runnable");
        }
    });
} catch (InterruptedException e) {
}
}

}).start();
}

Handler handler = new Handler() {
    @Override
    public void handleMessage(Message msg) {
        String testo = new String((byte[])msg.obj);
        btnh.setText(testo);
        Log.d("Testo", testo);
    }
};

public void onClickHandle(View view) {
    Message.obtain(handler, view.getId(),
String.valueOf("Esecuzione...").getBytes()).sendToTarget();
}
}

```

Da notare che vengono utilizzati ulteriori Runnable per aggiornare lo stato dei pulsanti, ciò è dovuto a problemi di **cross thread safe**, un determinato thread non può modificare valori riferiti ad un altro thread.

La nostra classe TestAsyncTask non farà altro che eseguire un processo in background tramite il metodo `doInBackground(String... arg0)` e, nel nostro esempio, l'abbiamo utilizzato passandogli delle stringhe come parametri, ogni stringa pubblicherà lo stato di avanzamento (`publishProgress`) rispetto al numero totale delle stringhe.

Ogni qualvolta viene aggiornato lo stato di avanzamento verrà stampato un messaggio a video con la relativa percentuale (metodo `onProgressUpdate(Integer... progress)`).

```

public class TestAsyncTask extends AsyncTask<String, Integer, Long> {

    private Context context;

    public TestAsyncTask(Context context) {
        this.context = context;
    }

    @Override
    protected Long doInBackground(String... arg0) {
        try {
            for(int i=0; i<arg0.length; i++) {
                publishProgress((int) ((float) (i+1) / (float) arg0.length) * 100);
            }
        } catch (Exception e) {
        }
        return null;
    }

    @Override
    protected void onProgressUpdate(Integer... progress) {
        StampaMessaggio(progress[0].toString() + "%");
    }

    @Override

```

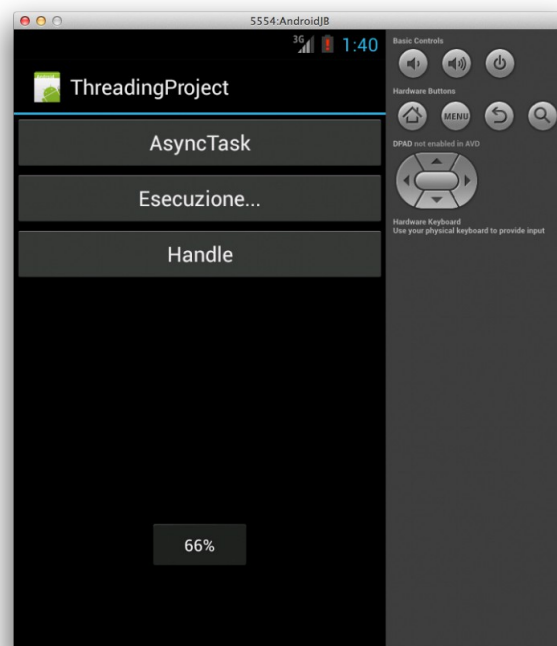
```

protected void onPostExecute(Long result) {
    StampaMessaggio("Finito!");
}

private void StampaMessaggio(String str) {
    Toast.makeText(context, str, Toast.LENGTH_SHORT).show();
}
}

```

Risultato



Miscellanea, parte 2

- Passaggio di parametri
- GridViewIcon
- Widget

Passaggio di parametri

In questo esempio analizzeremo un metodo per passare parametri a diverse activity di uno stesso progetto.

Solitamente, si utilizza `Bundle` che ci permette di mappare tramite chiave/valore per conservare e recuperare dati.

In questo caso utilizzeremo direttamente gli **Intent** delle Activity per recuperare dei parametri extra.

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

    <Button android:id="@+id/btndati"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/dati" />

</LinearLayout>
```

Nel layout della prima Activity è presente un pulsante che, una volta cliccato, provvederà a richiamare la seconda Activity tramite Intent, passando una stringa come parametro extra.

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

        Button btntesto = (Button) findViewById(R.id.btndati);
        btntesto.setOnClickListener(this);
    }

    @Override
    public void onClick(View arg0) {
        if(arg0.getId() == R.id.btndati) {
            Intent intent = new Intent(this, Activity2.class);
            intent.putExtra("Stringa1", "Testo 1");
            startActivity(intent);
        }
    }
}
```

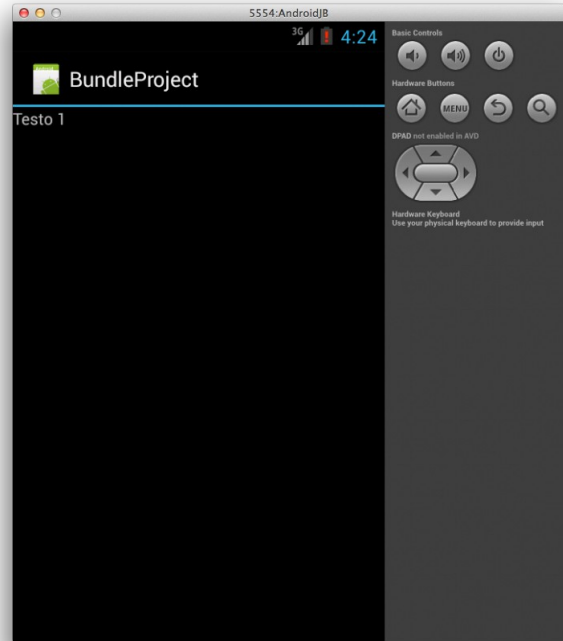
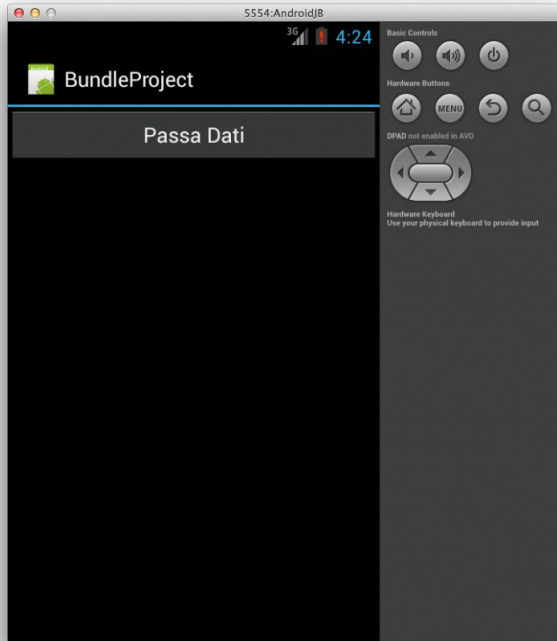
Il layout della seconda activity conterrà una etichetta di testo che servirà a visualizzare la stringa passata dall'activity precedente.

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

public class Activity2 extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main2);
        TextView txtTesto = (TextView) findViewById(R.id.txtдати);
        Intent intent = getIntent();
        txtTesto.setText(intent.getStringExtra("Stringa1"));
    }
}
```

}

Richiamando l'Intent della seconda Activity, è possibile recuperare il valore passato come parametro extra tramite la sua chiave, nel nostro caso “Stringa1”



GridViewIcon

Giusto per mettere un po' di pepe, proponiamo un esempio di GridView un po' più complesso. Nella nostra griglia, invece di visualizzare solo delle etichette di testo, creeremo un layout custom composto da una icona e da un testo.

Nel nostro layout principale piazzeremo la nostra GridView.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/sel" />
    <GridView android:id="@+id/grid1"
        android:layout_width="fill_parent"
        android:layout_height="match_parent"
        android:numColumns="2"
        android:columnWidth="50dp"
        android:gravity="center_horizontal" />
</LinearLayout>
```

Quando verrà generato l'evento onCreate della nostra Activity, assoceremo il nostro layout custom tramite un **Adapter** creato ad-hoc (ImageAdapter).

```
public class GridViewIconActivity extends Activity {
    /** Called when the activity is first created. */

    private GridView griglia;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        griglia = (GridView) findViewById(R.id.grid1);
        griglia.setAdapter(new ImageAdapter(this));
    }
}
```

Di seguito mostriamo il nostro layout personalizzato nella quale è presente una immagine ed una etichetta di testo.

Ogni elemento della GridView avrà questo tipo di layout.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/itemgrid"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    android:gravity="center_horizontal"
    android:background="#0066FF" >
    <ImageView android:id="@+id/imgitem"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <TextView android:id="@+id/txtitem"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="testo di prova"
        android:gravity="center_horizontal"
        android:textColor="#000000" />
</LinearLayout>
```

Il nostro Adapter sarà un'estensione di un **BaseAdapter** (Adapter di base) e servirà a costruire il

layout personalizzato ed assegnare anche un evento onClick quando viene selezionato l'elemento.

```
public class ImageAdapter extends BaseAdapter {

    private Activity sActivity;

    public ImageAdapter(Activity sActivity) {
        this.sActivity = sActivity;
    }

    @Override
    public int getCount() {
        return 8;
    }

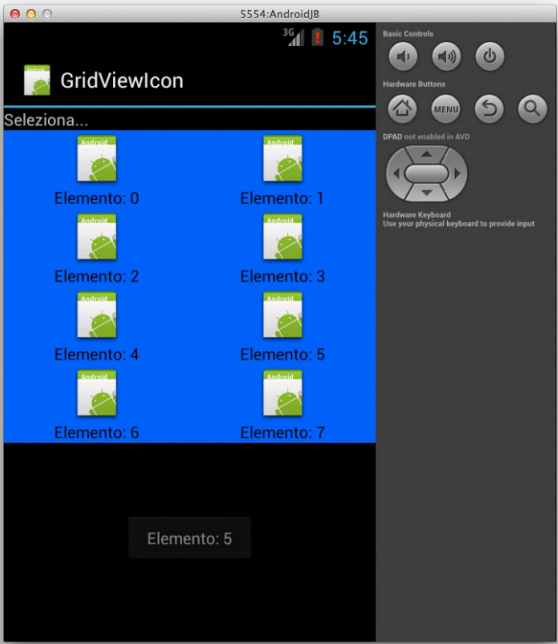
    @Override
    public Object getItem(int arg0) {
        return null;
    }

    @Override
    public long getItemId(int arg0) {
        return 0;
    }

    public View getView(int arg0, View arg1, ViewGroup arg2) {
        View view = arg1;
        if(view == null) {
            LayoutInflater li = sActivity.getLayoutInflater();
            view = li.inflate(R.layout.itemgrid, null);
            final TextView txt = (TextView)view.findViewById(R.id.txtitem);
            txt.setText("Elemento: " + arg0);
            ImageView img = (ImageView)view.findViewById(R.id.imgitem);
            img.setImageResource(R.drawable.ic_launcher);
            view.setOnClickListener(new OnClickListener() {
                @Override
                public void onClick(View arg0) {
                    Toast.makeText(sActivity, txt.getText(),
Toast.LENGTH_SHORT).show();
                }
            });
        }
        return view;
    }
}
```

Quando sarà selezionato un elemento della griglia, verrà visualizzata su schermo una notifica con l'etichetta di testo dell'elemento.

Risultato



Widget

I **Widget** sono dei contenitori grafici che vengono posizionati ed utilizzati sulla home o su una delle sue pagine.

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

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

```
public class WidgetProjectActivity extends AppWidgetProvider {  
}
```

Primissima differenza con un'applicazione normale è la classe a cui si fa riferimento: invece della solita Activity si utilizzerà 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="#00CCFF" >  
    <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/.

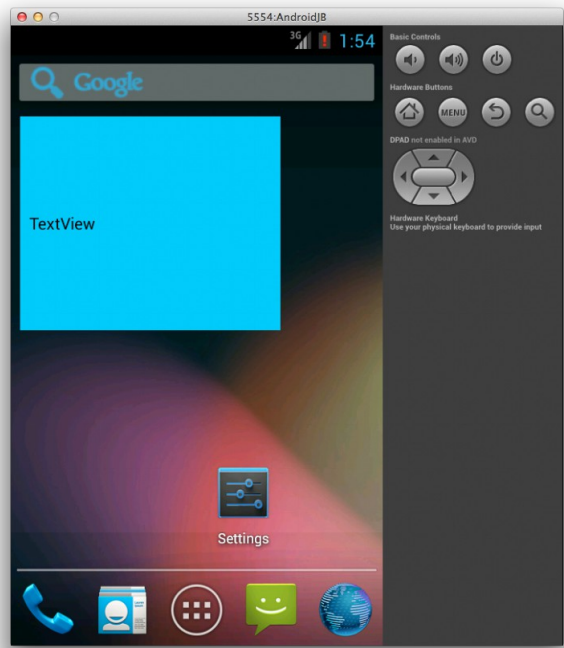
Nel mio caso l'ho chiamato widget_provider.xml .

```
<?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 74dp, e solitamente le misure si decidono come multipli di 74dp-2dp, per permettere di fare bordi particolari o margini, come gli angoli arrotondati.

Esempio



Conclusione

Cosa aspettarmi da questa guida?

Beh, sinceramente, spero in molti commenti sia positivi sia negativi.

E' importante sapere cosa la comunità di sviluppatori pensa del tuo lavoro: **migliorarsi! sempre tendere a migliorarsi!**

Ribadisco che questa guida è solo una introduzione, se qualcosa non è chiaro sarò disponibile al mio indirizzo di posta elettronica pietroalberto.rossi@gmail.com .