

# CREARE APP PER ANDROID





## L'ambiente di lavoro per progettare applicazioni per Android

**TUTORIAL 1** - In questo capitolo vedremo tutti i passaggi per scaricare il software *Android Developer Tools*, necessario a creare l'ambiente di lavoro per iniziare a progettare le tue prime applicazioni Android. In meno di 10 minuti, il tuo computer si trasformerà in un vero banco di lavoro con simulatore virtuale del tuo Android, senza necessariamente aver acquistato uno smartphone.

In questa prima lezione del "Creare app su Android", vedremo il **primo passo necessario** per mettersi nelle condizioni di iniziare a creare semplici applicazioni o come si dice in gergo "app" basate su Android e Java, ossia come dotarsi degli strumenti software che **agevolino la scrittura del codice Java**.

Per chi non ha mai programmato infatti, iniziare l'avventura con Java, potrebbe stroncare definitivamente ogni velleità di progettazione di una app in Android.

E' bene quindi affidarsi a degli strumenti che **agevolino l'inserimento di righe di codice** che, nella maggioranza dei casi, sono costituite da un insieme di parole o stringhe standard che è **noioso imparare a memoria**. Una volta scritte in automatico una decina di volte, sono sicuro riuscirai già a capire a cosa si riferiscono se seguirai le prossime lezioni.

Alla fine di questa prima lezione, troverai anche un breve video che **spiega tutti i passaggi**, nel caso tu abbia già un software di sviluppo come Eclipse, in modo che tu possa ricordare il tutto con più semplicità.

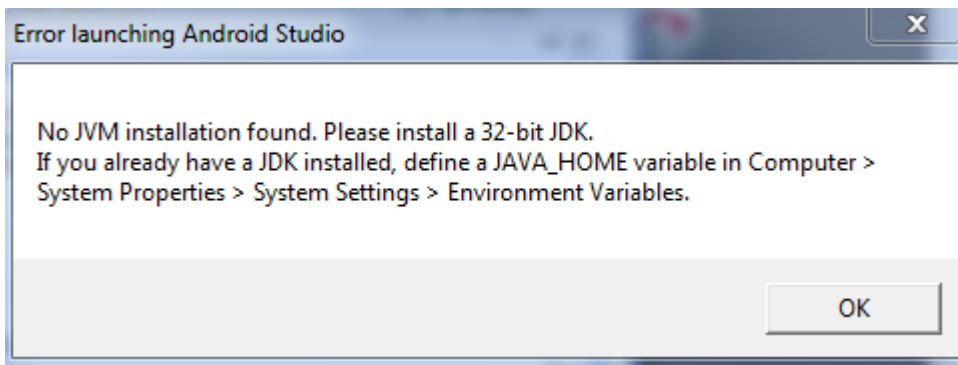
### STEP 1 : SCARICARE il software SDK sul tuo computer

Esistono diversi software che permettono la scrittura di codice Java per Android, ma la scelta consigliata è quella di usare il popolare **software open source Eclipse** con abbinato un plug-in dedicato per lo sviluppo su Android. Il **primo passo** quindi consiste nello scaricare il file .zip qui sotto a seconda del tuo sistema operativo:

- Se hai **Windows 32 bit** - Scarica SDK+ADT Bundle - [Clicca qui](#) (circa 400Mb)
- Se hai **Windows 64 bit** - Scarica SDK+ADT Bundle - [Clicca qui](#) (circa 400 Mb)
- Se hai **MAC, Linux** usa lo stesso link qui sopra e scorri verso il basso della pagina.

Dovrai accettare le condizioni e poi salvare il file nel tuo computer.

**NB:** Da maggio 2013, è disponibile una **nuova versione** del software di sviluppo chiamata **Android Studio**, che puoi scaricare con lo stesso procedimento a questo link: [Android Studio \(SDK\)](#) e che presto potrai usare al posto di Eclipse come SDK.



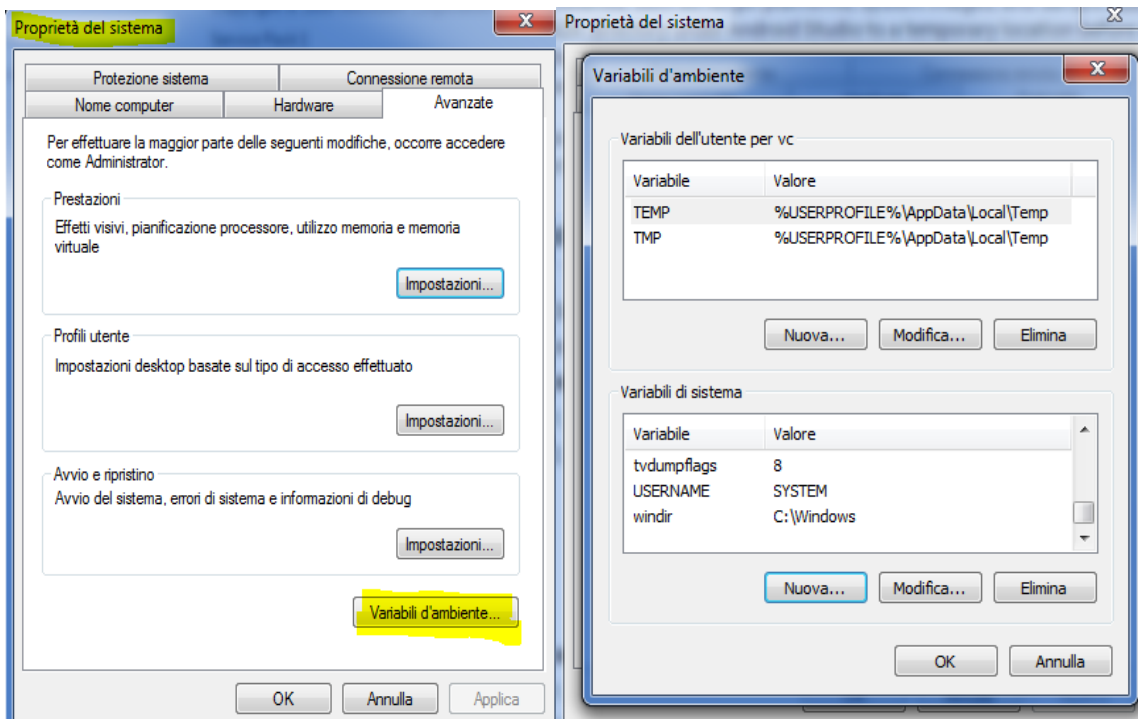
**ATTENZIONE:** Alcuni utenti hanno segnalato l'impossibilità di installare sia Android Studio che ADT in seguito alla mancanza di Java.

Per installare l'occorrente è necessario collegarsi al sito di oracle e scaricare l'ultimo JDK (Java Development Kit) adatto per il proprio sistema operativo.

A questo link <http://www.oracle.com/technetwork/java/javase/downloads/>

troverai le ultime versioni. Dovrai selezionare tra le opzioni "Java JDK" e poi il tuo sistema operativo.

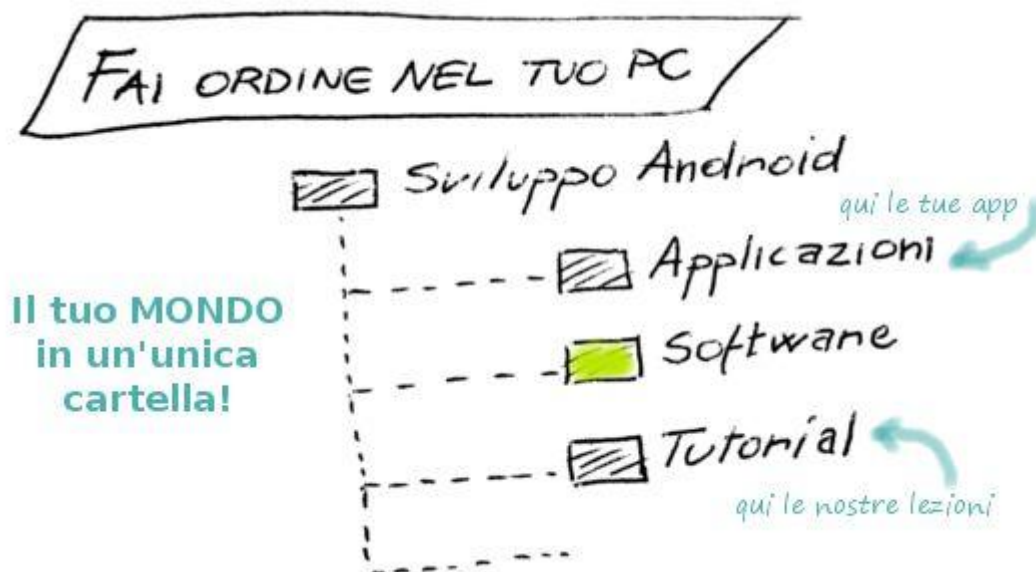
Una volta installato il software Java, procedi con l'installazione come indicato qui sotto. Se ancora non riesci, allora inserisci nelle variabili ambiente di window il percorso di installazione di JDK, aprendo il pannello di controllo, andando su sistema, poi su Impostazioni di sistema Avanzate, poi sulla scheda Avanzate e infine su Variabili Ambiente.



All'interno della sezione "variabili di sistema", aggiungi una nuova variabile con il nome "JAVA\_HOME" e come percorso, il percorso in cui hai installato Java JDK del tipo: C:\Program Files\Java\jdk1.7.0\_21 dove dovrai sostituire ai numeri qui a fianco, quelli effettivi che trovi nel tuo computer.

## STEP 2: INSTALLARE il software sul tuo computer. Consigli

Prepara una cartella nel tuo computer, in cui salvare tutto ciò legato ad Android, comprese le lezioni che scaricherai da questo sito web. Potresti ad esempio chiamarla "**Sviluppo Android**".



Poi decomprimi qui dentro il file appena scaricato, ed eventualmente sposta tutti i file e le cartelle in una **sottocartella**, chiamata ad esempio "**software**" al fine di avere tutto più ordinato.

In questo modo, ti sarà più semplice capire dove si trovano certe informazioni, e potrai usare la stessa cartella per memorizzare via via tutte le applicazioni che di volta in volta creerai. Le stesse lezioni di questo corso, le potrai salvare all'interno della sottocartella "**tutorial**". In più, se un giorno decidessi di usare un nuovo pc, ti basterà copiare tutta questa cartella, e sarai sicuro di non aver dimenticato nulla.

## STEP 3: Fai PARTIRE il motore

Ora che hai tutta la cartella ipotetica "**Sviluppo Android**" in ordine, entra nella sottocartella dove hai decompresso il file e cerca la cartella "Eclipse". Clicca sull'eseguibile eclipse.exe e ..... BOOOM....ecco che hai fatto partire l'interfaccia del software che inizieremo ad approfondire dalle prossime lezioni.

Se usi già Eclipse, nell'area privata del corso Base Android, troverai un breve video, che ti spiega come installare il plug-in ADT necessario allo sviluppo per Android.

Se invece hai installato Android Studio, dopo averlo lanciato e confermato che non hai creato progetti precedenti, ecco la schermata che vedrai e che inizieremo ad approfondire dalle prossime lezioni.



# Welcome to Android Studio

## Recent Projects

No Project Open Yet

## Quick Start



New Project...



Import Project



Open Project



Check out from Version Control



Configure



Docs and How-Tos

© createapp.com



## Creare la tua prima app usando Eclipse o ADT

*TUTORIAL 2: In questo tutorial vedremo tutti i passi per creare il tuo primo documento di lavoro in Android, sfruttando come software di sviluppo il popolare Eclipse o l'equivalente interfaccia di Android Developer Tools (ADT).*

Ora che abbiamo tutti gli strumenti correttamente impostati (vedi TUTORIAL 1), possiamo iniziare ad avventurarci nel mondo della **creazione della nostra prima applicazione**.

Non prendere paura se all'inizio alcune cose non ti rimarranno nella testa. E' del tutto normale. Cercheremo di focalizzarci man mano sugli aspetti proprio della **programmazione in Java**, senza però entrare nei minimi dettagli inutili per chi si avvicina per la prima volta a questo fantastico mondo.

Segui passo dopo passo questo Tutorial, **senza farti troppe domande** all'inizio. E' il risultato finale che conta.

Al termine, non avrai sicuramente imparato molto, ma come il bambino che impara ad andare in bici per la prima volta, anche fare un metro senza cascare, e' un **primo importante risultato**.

### Cosa imparerai

I punti salienti che imparerai in questa seconda lezione sono:

- Saper usare **Android Developer Tools (ADT)** o Eclipse per creare un nuovo progetto.
- Conoscere le diverse **impostazioni** da settare per ogni progetto.
- Conoscere il significato delle **cartelle** di cui è costituito un progetto Android
- Imparare alcune **terminologie** di uso comune in un progetto Android (Activity, Layout, Risorse, User Interface (UI))

### STEP 1: Alcune terminologie UTILI. Le Activity

Quando si deve sviluppare un'applicazione Android, il **primo passo** è quello di creare il "foglio" su cui andare a scrivere, così come accade quando si inizia a scrivere una lettera o come quando si inizia a progettare un sito web costituito da più pagine.

Così come un **progetto di un sito web**, è costituito da più pagine html o pagine dinamiche, anche un progetto Android, può essere costituito da più pagine. Anche se **non è corretto al 100%**, per semplicità possiamo dire che l'equivalente di una **pagina web VUOTA, o finestra nel mondo desktop**, nel mondo Android si chiama "**Activity**".

Un'applicazione Android costituita da più **schermate o pagine**, come quella evidenziata qui sotto, sarà costituita allora da diverse Activity e ogni Activity sarà caratterizzata dal **mostrare al suo interno** del testo, delle immagini, dei bottoni, esattamente come avviene nel corpo (tag body) di una pagina web.

E' chiaro quindi che per **passare da una pagina ad un'altra**, significa passare da una Activity all'altra. Nell'esempio qui sotto, l'applicazione è costituita da 2 Activity NON VUOTE (A-B), che sono state riempite con del contenuto.

Per visualizzare la pagina A dovremo richiamare l'Activity A, mentre per visualizzare la pagina B dovremo richiamare l'Activity B. Nulla di complesso per ora, tranne la nuova terminologia.



Nel mondo Android, il **corpo, ossia il layout** di ogni Activity, viene progettato non usando dei tag html, come avviene nel mondo delle pagine web, ma tipicamente progettando un **file xml**, al cui interno saranno definiti una serie di elementi quali testo, immagini, bottoni, campi di un form.

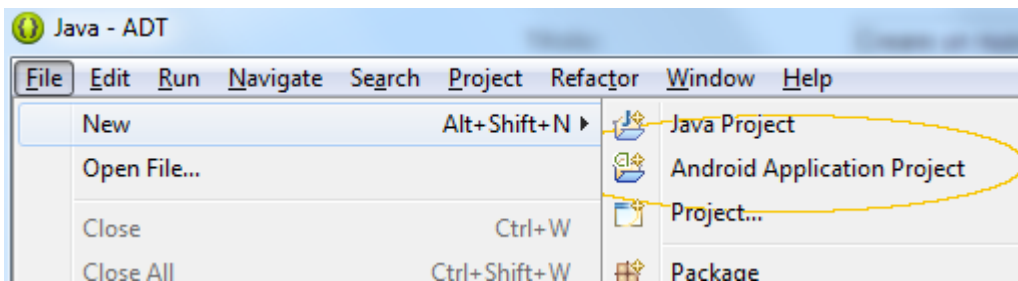
Pertanto per poter creare un'applicazione come quella sopra, non solo avrò bisogno di specificare diverse Activity, ma dovrò anche progettare 2 diversi **layout**, ossia contenuti, che vorrò mostrare all'interno. E ciascun **layout** sarà un semplice file .xml, che avrà all'interno gli elementi classici di una pagina web, come foto, testo, link.

## STEP 2: Le impostazioni di un progetto ANDROID

Allora, giusto per partire, una volta aperto Eclipse o Android Developer Tools (ADT), dovrai allora:

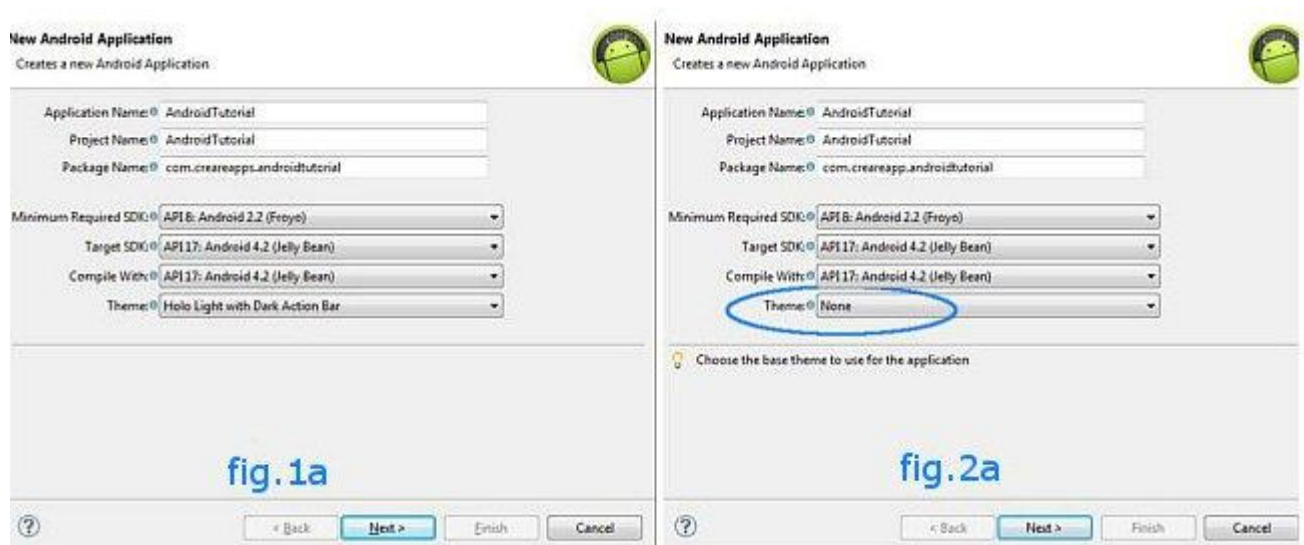
1. Selezionare File ► New
2. Selezionare "Android Application Project" dalla menu a lato che compare (vedi figura sotto)





**NB:** Se usi Eclipse, e la voce "Android Project" non viene visualizzata, puoi cliccare su: New ► Other ► Android ► Android Project.

A questo punto compariranno una serie di finestre di dialogo, apparentemente complesse, ma che ti **diventeranno famigliari**, dopo aver ripetuto 10 volte sempre questa procedura.



In questa finestra (fig.1a) in sostanza è necessario specificare il **nome da dare** al tuo progetto e qualche altro parametro che ti permetterà di **identificare la versione** del sistema operativo, con cui intendiamo sviluppare la nostra applicazione, più informazioni sulle icone da usare e sul nome della prima pagina che si vuole creare.

Come ben sai, ad ogni nuova versione di Android, vengono introdotte nuove funzionalità, quindi e' consigliabile sviluppare per versione **non troppo recenti**, al fine di rendere fruibile l'applicazione anche a chi non ha un dispositivo super aggiornato. In questa fase però non preoccuparti più di tanto di cosa scegliere.

Dovrai pertanto completare le seguenti voci:

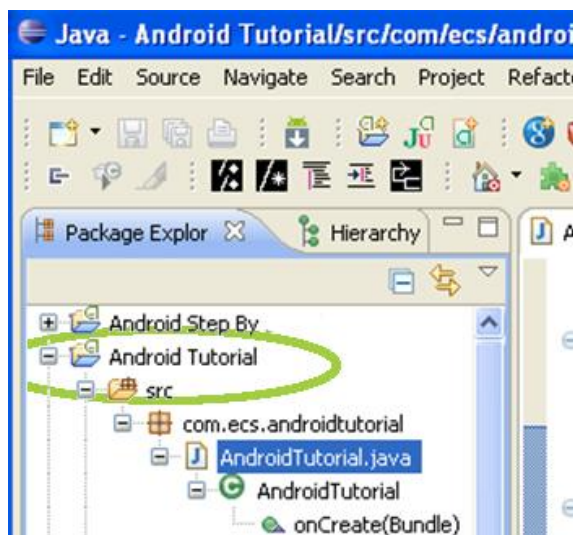
1. (Application name) - inserire il **nome della tua Applicazione** ossia quella che apparirà nello Store di Google. Comparirà anche come titolo quando l'utente finale andrà ad installarla sul proprio dispositivo mobile. Usa un nome non troppo lungo e accattivante.
2. (Project name) - il **nome del tuo progetto** usato internamente al tuo computer per diversificarlo dagli altri.

3. (Package name) - il **nome del pacchetto**, con una notazione nella forma com.XXX.YYY. E' in sostanza il nome del "contenitore" java che identifichera' tutti i pezzi di codice con cui sara' sviluppata la tua applicazione. E' una convenzione quindi quello che ti consiglio è di inserire ad esempio il tuo ipotetico nome di dominio, seguito dal nome della tua applicazione senza usare spazi o caratteri non alfanumerico. Esempio: "com.miosito.nomeapplicazione"
4. I parametri legati alla versione dell'SDK etc lasciali pure impostati in automatico per ora, con l'unica avvertenza (fig.2a) di non selezionare alcune tema (theme) quindi impostandolo a None.

Nelle successive finestre:

- Seleziona la posizione in cui verranno salvati i documenti (workspace). Puoi impostare la cartella "applicazioni" vista nel Tutorial 1
- Spunta la casella per **creare un'Activity** con il nome predefinito (vedremo in maggior dettaglio cos'è in una guida apposita)
- Seleziona la tipologia di icone da associare alla tua applicazione.
- Imposta la tipologia di Activity iniziale (vedi tutorial successivi), predefinita (VUOTA) ossia la tua ipotetica prima pagina.
- E infine dai un nome a questa Activity (lascia pure il nome predefinito pari al nome della tua applicazione) compreso il layout da usare e senza alcun tipo di navigazione.

Terminate queste impostazioni, puoi proseguire cliccando su Next e magicamente si aprira' una finestra che all'interno di "Package Explorer" (simile a Esplora risorse di Windows), ti mostrera' la cartella del tuo progetto appena creata (vedi sotto).



All'interno di questa cartella progetto, vi saranno **diverse cartelle e file**, che via via impareremo a conoscere. Queste cartelle sono create in automatico dal software e

servono per contenere una serie di risorse necessarie al funzionamento e alla personalizzazione grafica della nostra applicazione.

### STEP 3: Entrare nel CUORE della tua prima applicazione

Giusto per esaudire la tua voglia di vedere "il volto" dell'applicazione che in automatico si crea con questo processo, puoi espandere la relativa cartella src e sotto al pacchetto java con il nome che hai scelto del tipo com.xxx.yyy, troverai proprio il file che termina con estensione .java!

Facendo il classico doppio click sopra al file, si aprirà il tuo primo "foglio" con all'interno una serie di righe apparentemente incomprensibili, ma che impareremo a "decifrare" nel corso delle prossime lezioni. Il tutto senza che tu abbia scritto una sola riga di codice a mano!

```
package com.ecs.androidtutorial;

import android.app.Activity;
import android.os.Bundle;

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

Capisci allora la **potenza di avere un IDE** che ti aiuti nella scrittura delle tue applicazioni in java per Android.

Ovviamente se provi a leggere le diverse righe prenderai paura, ma è normale che sia così visto che non hai ancora alcuna conoscenza di java!

Nella prossima lezione cercheremo di andare oltre e vedere **cosa produce** questo primo file se lo volessimo testare grazie all'emulatore installato nella prima lezione.

## Le pagine web in Android: Activity

*TUTORIAL 3: In questo breve tutorial, vedremo le analogie possibili tra una pagina web e l'equivalente pagina nel mondo di Android. Impareremo in modo definitivo cosa sia una Activity e come devono essere riempite al fine di visualizzarne del contenuto all'interno.*

Così come il pittore, parte da una parete bianca, da dove deve partire lo sviluppatore di applicazioni per Android?

Lo sviluppo di un **sito web**, presuppone la creazione di un certo numero di **pagine web**. Se analizziamo un qualsiasi sito web, questo sarà costituito da decine di pagina .html o pagine dinamiche .php o di altri linguaggi. Non appena creo una nuova pagina .html, questa tipicamente risulta essere priva di contenuto, ossia VUOTA.

Quello che creo se vogliamo è un **contenitore** che ha un certo nome, e che verrà popolato in una fase successiva da testo, immagini etc.

Partendo da questa analogia, possiamo dire che lo **sviluppo di una app in Android**, presuppone la creazione di un certo numero di "Activity" ossia (semplificando) un certo numero di pagine VUOTE, che avranno uno specifico nome e che dovranno, in una fase successiva essere popolate con del **contenuto quindi con del testo, delle immagini, dei video, dei campi di un modulo web**.

Ogni **Activity** dovrà prelevare un **layout (vedi tutorial successivo)**, ossia una rappresentazione grafica degli elementi che si vogliono rappresentare all'interno di questa pagina (testo, link, immagini, bottoni)

Riassumendo, possiamo quindi dire che un'**applicazione completa in Android**, conterrà diverse activity e generalmente ognuna di queste richiamerà un layout ben definito.

### Esempio: ideare una app in Android

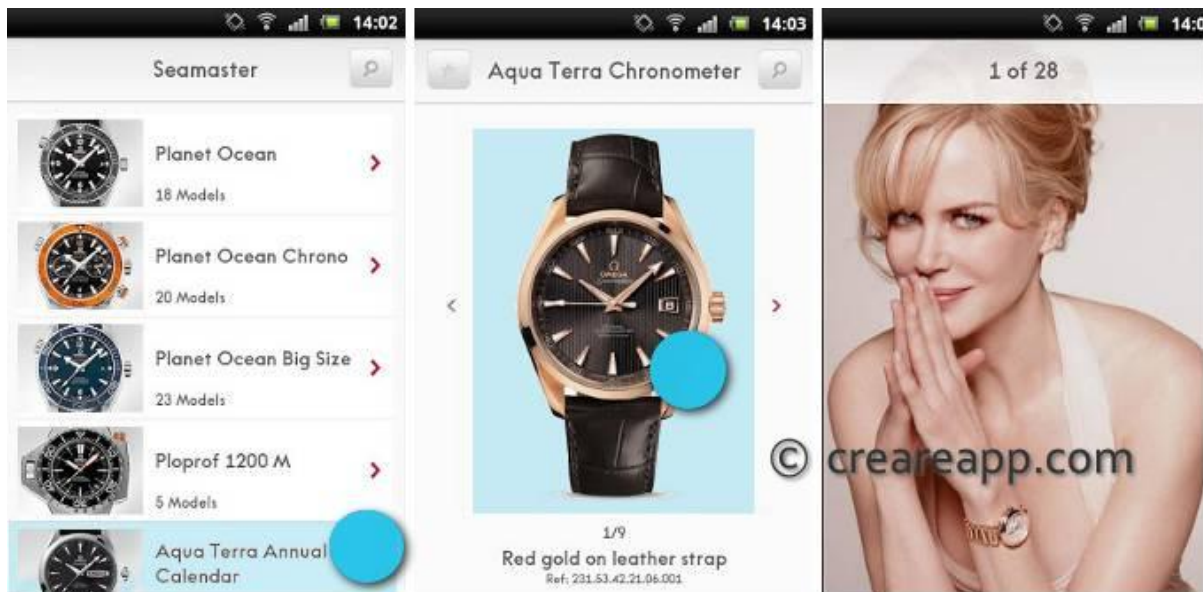
Come ogni progetto sul web, anche la "**nascita**" di una **app in Android**, richiede l'importante passo di decidere come **suddividere** la visualizzazione di tutte le informazioni che si vogliono mostrare all'utente.

Ad esempio se un tuo cliente che produce orologi, ti chiedesse di creare una app che mostri la lista di tutti gli orologi in produzione, con descrizione delle caratteristiche per ognuno e una foto di un personaggio famoso che lo indossa, potresti pensare ad una app base di 3 pagine.

- Pagina 1: elenco orologi (activity: orologi)
- Pagina 2: dettaglio orologio (activity: orologio\_dettaglio)

- Pagina 3: foto personaggio famoso (activity: orologio\_vip)

L'**ossatura** della tua app quindi sarà costituita da **3 ACTIVITY**, con relativi nomi per identificarle in modo univoco, che dovranno essere **riempite con del contenuto**. Il passaggio da una pagina all'altra, quindi il **passaggio da una activity ad un'altra**, dovrà anch'esso essere pensato in fase di ideazione dell'applicazione, al fine di rendere facile la navigazione tra pagine diverse.



Ad esempio, se il navigatore fosse nella prima pagina e cliccasse sulla freccia a fianco di ogni orologio, dovrò prevedere delle istruzioni che **recuperino la seconda pagina/Activity** (orologio\_dettaglio) e mostrino il **contenuto** presente all'interno. E così via per ogni collegamento ipertestuale presente nel contenuto delle diverse pagine.

Purtroppo, mentre per una pagina web, rendere **clickabile del testo** o un'immagine è un gioco da ragazzi perchè basta inserire il classico **tag HTML href**, nel mondo Android questo richiede **parecchio codice**, ma niente paura, in quanto, grazie ai software installati in precedenza, questo verrà creato quasi in automatico con pochi click.

L'altro importante passaggio in fase di ideazione di una app, è decidere **COSA** inserire all'interno di ogni pagina, e soprattutto **COME** disporre a livello di **layout** i diversi elementi.

Ad esempio nell'Activity (orologi), le foto voglio siano visualizzate a sinistra o a destra? A livello di descrizione il mio cliente vuole visualizzare il nome dell'orologio e il numero di modelli o devo inserire altre informazioni?

Insomma gli stessi passaggi che si fanno nel mondo dello sviluppo di siti web, dovranno essere fatti nel mondo dello sviluppo di app in Android, partendo sempre dall'**obiettivo finale** che si vuole raggiungere e cercando di **semplificare al massimo** la struttura di navigazione, rispetto ai classici siti web, come evidenziato qui sotto.



Nel prossimo tutorial base, vedremo i passaggi per **creare il contenuto (layout)** da inserire all'interno di ogni activity.

## Progettare un semplice Layout in Android

*TUTORIAL 4: Vedremo alcuni concetti chiave per realizzare semplici interfacce grafiche, basandosi su analogie con il mondo della creazione di pagine web, con l'uso del classico codice html.*

Dopo aver visto come si crea un **foglio bianco** contenitore di qualcosa, ossia la nostra **Activity**, il secondo passo è quello di riempire con del **contenuto** questo contenitore.

Come già detto, così come avviene per le pagine di un sito web, non appena inserisco del testo, delle immagini, dei link, anche per creare una **pagina di una app**, è necessario inserire del testo, delle immagini, dei link etc.

La differenza rispetto al mondo dei siti web, dove, per definire la **struttura (layout) e il contenuto** di una pagina web, si usa il **linguaggio HTML e i CSS**, è che nel mondo Android, la struttura e il contenuto di una pagina si definiscono con il **linguaggio XML**.

Niente paura perchè l'XML è per certi versi più semplice dell'HTML.

XML non è altro che un linguaggio con cui si **rappresentano** dei dati. Pertanto, invece di creare il contenuto della pagina, usando del codice html, quindi iniziando con doctype, body etc etc, e salvando la pagina con estensione .htm, dovremo procedere creando un file con **estensione .xml**, che conterra' all'interno del **codice xml e opportuni tag**.

### Cosa imparerai

Giusto per scaldare un pò i muscoli, procediamo creando la nostra prima struttura di pagina android: un layout semplice semplice:

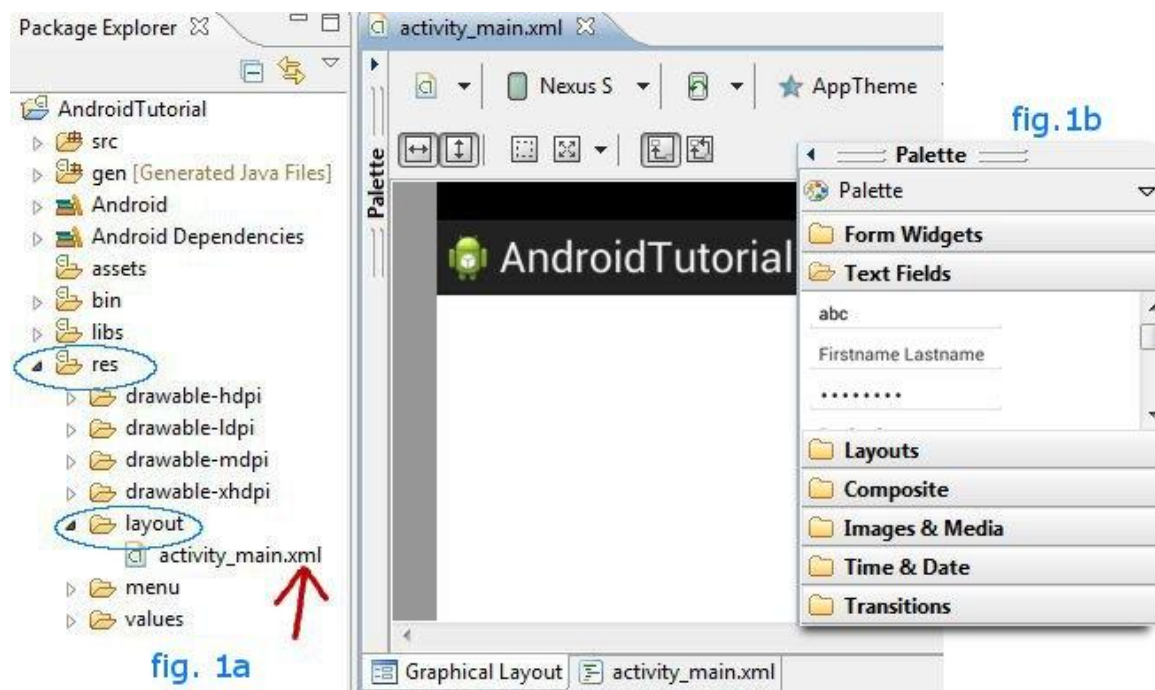
- Imparerai ad usare Eclipse per creare la tua prima 'interfaccia grafica di una pagina
- Vedrai l'ossatura di una interfaccia grafica, ossia il codice XML
- Imparerai le terminologie *Linear Layout, Relative Layout, FrameLayout, TextView, Button, ImageView*

### STEP 1: Come si crea il VESTITO (layout grafico) di una pagina

Grazie all'uso di ADT o Eclipse (vedi tutorial 1), diventa abbastanza immediato vedere come è fatto un file di questo tipo, perchè all'atto di creazione di un nuovo progetto, viene creato ex novo anche un semplice layout, che possiamo recuperare andando all'interno della cartella "res", e cliccando sopra a "layout".

Troveremo un **file con estensione .xml** che all'interno conterra' proprio i tag necessari a creare un layout che si sviluppa dall'alto verso il basso e da sinistra verso destra. Non prendere paura se per ora ti sembra qualcosa di quasi incomprensibile: è del tutto normale.

Grazie ad ADT o Eclipse, e all'**interfaccia WYSIWYG**, possiamo progettare il **contenuto** della nostra pagina esattamente come facciamo con i popolari software Dreamweaver o simili. All'interno di Eclipse, troveremo una serie di **strumenti** (schermata palette), come testo, bottoni etc, che grazie alla semplice operazione di **trascinamento**, ci permetteranno di creare delle interfacce utente (UI) in modo facile e veloce (vedi fig.1b)



## STEP 2: Visualizzare l'ossatura di un layout: il codice XML

Se invece vogliamo vedere proprio il **codice XML** della pagina come faccio? E' sufficiente selezionare la scheda a fianco della voce Graphical Layout. Vedi immagini sotto.

```
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>
```

Che caratteristiche ha questo layout? Per prima cosa **NON devi imparare a memoria** tutto il codice xml che vedi. Questa è una prima assicurazione importante per chi devi iniziare. La seconda cosa che puoi notare è che nel codice xml compaiono delle scritte in colore verde che



assomigliano molto a tag **html**. Nel caso della figura qui sopra, puoi notare il tag "RelativeLayout" che identifica una tipologia di contenitore che permette di creare uno specifico layout.

### Tipologie di Layout predefinite

Possiamo dire che ogniqualvolta devi inserire del testo, dei bottoni, delle pagine web, è necessario prima di tutto creare il **CONTENITORI** principale di questi elementi, che in Android vengono chiamati anche **View**.

In android è possibile usare diversi contenitori, ognuno con diverse **caratteristiche** e con **nomi** diversi. I più noti sono:

- **Linear Layout (verticale o orizzontale)** - Permette di inserire i diversi elementi della pagina (View) ossia bottoni, testo, immagini, in righe orizzontali o verticali, dall'alto verso il basso e da sinistra verso destra.
- **Relative Layout** - E' forse il contenitore maggiormente impiegato perchè versatile, in quanto permette di definire la posizione di ogni elemento, relativamente ad altri elementi precedentemente inseriti nel contenitore.
- **Frame Layout** - Forse il più semplice contenitore, ma meno flessibile, in quanto permette di allineare ogni elemento all'angolo superiore sinistro quindi sovrapponendoli uno all'altro.

A grandi linee possono essere pensati come a delle **scatole indipendenti**, che al loro interno possono contenere altre scatole, quindi altri contenitori, così come avviene con le matrici. Il contenitore principale prende il nome di root o radice ed è l'equivalente del classico tag body delle pagine web.

Che caratteristiche hanno queste scatole e dove si devono creare?

### L'importanza della cartella predefinita "res" : Resources

Come sicuramente avrai notato, quando hai creato il tuo primo progetto, in automatico si sono generate **numerose cartelle** dai nomi più strani. Ebbene ognuna di queste cartelle contiene una serie di file che, in parte sono di sistema e non devono essere toccati, in parte sono da personalizzare a seconda della tipologia di applicazione che si vuole creare.

Impareremo nel corso dei prossimi tutorial il significato di ognuna di queste cartelle, ma giusto per non perderci, ogni interfaccia grafica (layout) che andrai a progettare, quindi

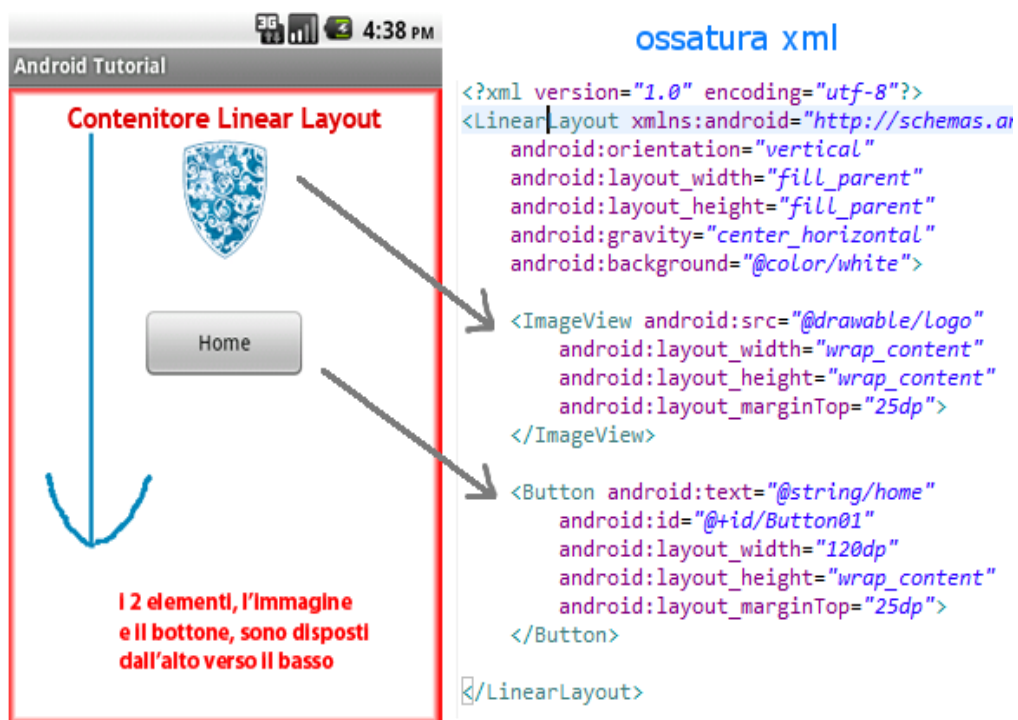
ogni file .xml, dovrà essere memorizzata, all'interno di una sottocartella di "res" che rappresenta l'armadio dove custodire il **vestito e gli accessori** di ogni applicazione.

Pertanto un'applicazione costituita da 5 pagine (activity) dovrà contenere almeno 5 layout (vestiti) e questi dovranno essere memorizzati all'interno della sottocartella "Layout" interna a "res".

### Scatola "LinearLayout"

La prima scatola che devi imparare ad usare è quella che prende il nome di **LinearLayout**.

Il nome è abbastanza eloquente, perchè non è altro che quella scatola che permette di contenere elementi che devono essere **visualizzati dall'altro verso il basso**, esattamente come avviene per le pagine web quando si usa una tabella a singola colonna.



The image shows a screenshot of an Android emulator window titled "Android Tutorial". The emulator displays a simple layout with a blue shield-shaped logo at the top and a grey button labeled "Home" below it. A blue arrow points downwards from the logo, indicating the vertical arrangement. Below the emulator, a red-bordered box contains the text: "I 2 elementi, l'immagine e il bottone, sono disposti dall'alto verso il basso". To the right of the emulator, the XML code for the layout is shown, titled "ossatura xml". The code defines a LinearLayout with a vertical orientation, containing an ImageView for the logo and a Button for the "Home" text. Arrows point from the XML code back to the corresponding elements in the emulator.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.a
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:gravity="center_horizontal"
  android:background="@color/white">
  <ImageView android:src="@drawable/Logo"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="25dp">
  </ImageView>
  <Button android:text="@string/home"
    android:id="@+id/Button01"
    android:layout_width="120dp"
    android:layout_height="wrap_content"
    android:layout_marginTop="25dp">
  </Button>
</LinearLayout>
```

Nell'esempio qui a lato, abbiamo creato un semplice **layout** costituito da 3 elementi:

1. un contenitore principale o nodo radice
2. un'immagine
3. un bottone con del testo.

A livello di codice xml, ognuno di questi elementi è rappresentato con una notazione ben precisa, che NON è necessario imparare a memoria, visto che ADT o Eclipse permette, con semplici operazioni di Drag&Drop di inserire questi elementi.

Analizzando in dettaglio ogni singolo elemento possiamo dire che:

- L'elemento radice ossia il body della nostra pagina, è costituito da un contenitore **LinearLayout**.
- Questo tipo di contenitore, può contenere **elementi figli** sia disposti in righe che in colonne. nel nostro caso i due figli sono un testo (TextView) e un bottone (Button)
- Per poterli disporre in colonne, è necessario specificare l'attributo "orientation" del contenitore **LinearLayout** a "vertical". Nel caso volessimo disporre gli elementi su di una riga, è necessario specificare l'orientamento "horizontal".

Nel nostro caso, i due elementi immagine e bottone sono disposti su una colonna, e questo è il motivo per cui abbiamo usato l'orientamento verticale. Nella fig. 2b puoi vedere il relativo codice xml, in cui sono evidenziati il contenitore LinearLayout, il campo di testo (**TextView**) e l'immagine (**ImageView**).

### Scatola "RelativeLayout"

Quando inizierai a creare delle interfacce più complesse, avrai la necessità di disporre gli elementi non più uno sotto l'altro o di fianco, ma con un ordine vario.

In questo caso dovrai allora usare una struttura che ti dia più autonomia nella disposizione degli elementi della tua pagina (bottoni, testo, immagini) ossia le nostre View: questa struttura prende il nome di "**RelativeLayout**", che come dice lo stesso nome, permette di posizionare i diversi elementi in funzione della posizione di ognuno, o in funzione della posizione rispetto al padre contenitore.

Un tipico esempio è quando si vuole centrare nella pagina un singolo elemento, e solo quello e disporre invece gli altri su una riga affiancati..

Nell'esempio precedente, siamo riusciti a **centrare TUTTI** gli elementi View, ma se vogliamo agire **solo su uno di questi**, indipendentemente dagli altri, allora dobbiamo necessariamente partire con il creare un "RelativeLayout", e poi aggiungere i diversi elementi trascinandoli con Eclipse, dalla Palette degli strumenti. Magicamente, compaiono delle frecce che ci daranno delle indicazioni su come posizionare i diversi elementi l'uno rispetto all'altro, cosa non possibile se invece partiamo con il classico LinearLayout.

Lo schema xml di un Relative\_Layout è identico a quello del precedente, cambia solo il nome.

Per centrare allora solo un elemento, basterà aggiungerlo all'interno di un contenitore "RelativeLayout" e impostare l'attributo **android:layout\_centerInParent** al valore true.

**NB:** Come sempre **non è necessario impararlo a memoria**, ma giusto per sapere dove mettere le mani nel caso si voglia modificare manualmente la posizione di alcuni elementi.

La lista completa di questi attributi la puoi trovare [cliccando qui](#).

Nell' esempio qui sotto, abbiamo proprio usato un semplice relative layout, che va ad occupare in larghezza e altezza, la dimensione dello schermo (**layout\_width=match\_parent**), e con un orientamento verticale. All'interno di questa scatola, abbiamo inserito rispettivamente un testo, un campo di input, e un bottone.

Come vedi il testo è allineato al centro, mentre il campo di input e il bottone sono disposti uno a fianco dell'altro, e la loro posizione è determinata andando ad agire proprio sugli attributi che iniziano con *layout\_* (es. **layout\_centerInParent=true; layout\_centerHorizontal, layout\_above, layout\_to RightOf, layout\_alignParentLeft** etc)

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.     android:layout_width="match_parent"
4.     android:layout_height="match_parent" >
5.
6.     <TextView
7.         android:id="@+id/textView1"
8.         android:layout_width="wrap_content"
9.         android:layout_height="wrap_content"
10.        android:layout_above="@+id/button1"
11.        android:layout_centerHorizontal="true"
12.        android:text="Esempio di Relative Layout"
13.        android:textAppearance="?android:attr/textAppearanceMedium"
14.    />
```

15. **<Button**

16. `android:id="@+id/button1"`

17. `android:layout_width="wrap_content"`

18. `android:layout_height="wrap_content"`

19. `android:layout_alignBaseline="@+id/editText1"`

20. `android:layout_alignBottom="@+id/editText1"`

21. `android:layout_toRightOf="@+id/editText1"`

22. `android:text="Button" />`

23.

24. **<EditText**

25. `android:id="@+id/editText1"`

26. `android:layout_width="wrap_content"`

27. `android:layout_height="wrap_content"`

28. `android:layout_alignParentLeft="true"`

29. `android:layout_alignParentTop="true"`

30. `android:layout_marginLeft="16dp"`

31. `android:layout_marginTop="54dp"`

32. `android:ems="10" />`

33.

34. **</RelativeLayout>**

## Il primo passo prima di chiamare una Activity

*TUTORIAL 5: In questo articolo/video vedremo i passi per creare la nostra prima activity ossia la nostra equivalente prima pagina web di un normale sito. Capiremo come, usando Eclipse, il meccanismo di creazione sia quasi automatizzato. L'unica cosa da sapere è quale layout prelevare per mostrarlo sullo schermo dello smartphone del tuo navigatore.*

Ora che sappiamo come **creare un nuovo progetto** con Eclipse, come costruire un'**interfaccia grafica** da inserire all'interno della cartella Layout, e infine cosa sia un'**Activity**, vediamo di chiudere il cerchio e analizzare in dettaglio il procedimento grazie al quale il nostro utente, dopo aver aperto l'applicazione, visualizzi la sua **prima pagina in Android**.

### Cosa imparerai?

Al termine di questo tutorial avrai imparato:

- Come usare Eclipse per **aprire una activity** e visualizzarne il codice
- Significato della **cartella "src"** presente in ogni progetto
- Significato dell'istruzione java **setContentView()**;

Riassumendo i precedenti tutorial, abbiamo imparato che il metodo con cui si definiscono queste pagine in Android a livello grafico, e' principalmente grazie alla definizione di un **file xml** che stabilisce **cosa visualizzare e come**, all'interno della pagina.

L'anello mancante è capire come comandare la visualizzazione di questa pagina e di altre pagine che avrò la necessità di creare in seguito.

Per fare questo dobbiamo servirci di Eclipse, in quanto, molto del codice che ci servira' per fare questa banale operazione, viene proprio creato in automatico.

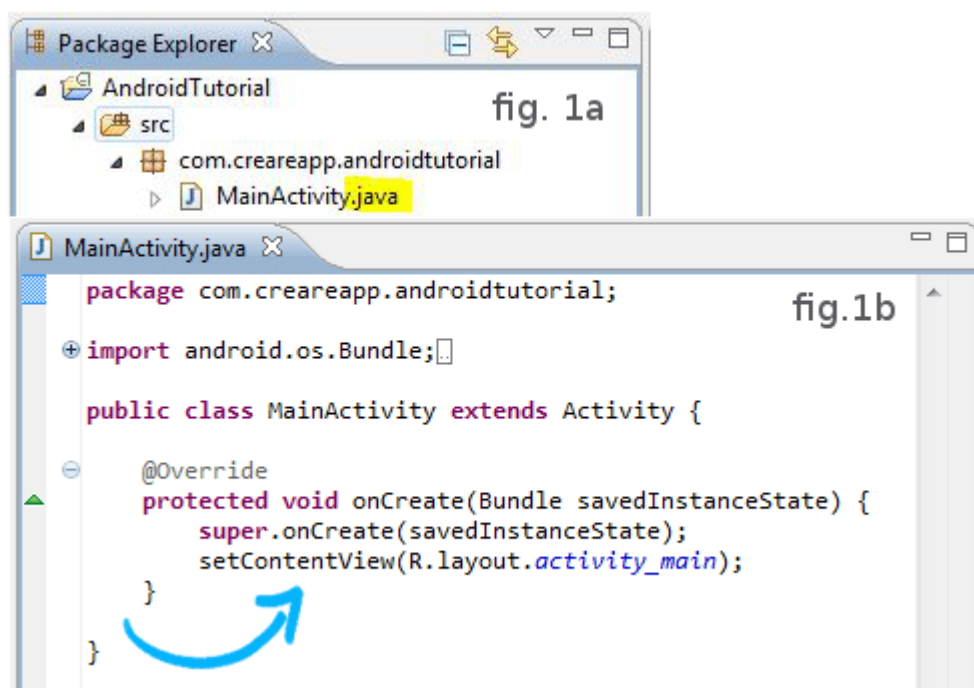
Se difatti, proviamo ad aprire un qualsiasi progetto Android con Eclipse, quello che ci viene mostrata dal software e' proprio la prima Activity. Ma come avviene questa magia?

### STEP1: dove si trova il cuore di ogni app?

Abbiamo già introdotto il significato della cartella di sistema "res", e abbiamo visto anche il significa e l'uso della cartella di sistema "layout". Se non ti ricordi cosa sono e a cosa servono, ti invito a rileggere il tutorial "[Progettare un semplice Layout in Android](#)". La terza cartella che impareremo a conoscere come le nostre tasche, è quella che prende il nome di "**SRC**" che sta per "source" ossia sorgente del codice java.

All'interno della cartella src, troveremo di volta in volta i file java, che rappresentano il cuore logico della nostra applicazione. Sarà proprio grazie alle righe scritte all'interno di ognuno di questi file che la nostra applicazione sarà in grado di rispondere alle richieste fatte dal navigatore.

Allora se proviamo ad espandere il contenuto di questa cartella, troveremo, come nostro primo file, proprio la prima Activity creata, che avrà il nome scelto in fase di impostazione del progetto (vedi tutorial "[Creare la tua prima app usando Eclipse o ADT](#)"). Puoi notare subito che l'estensione del file è proprio .java (fig. 1a)



Quello che invece visualizzi nella (fig. 1b), non è altro che il codice java (non prendere paura), che Eclipse ha creato al volo per noi e che imparerai a decifrare seguendo i prossimi tutorial.

### STEP 2: la mia prima istruzione java: setContentView()

Senza spendere troppe parole, la **prima cosa** che vogliamo fare quando si crea una applicazione, è mostrare la **prima pagina**. Ora, mentre nel caso di un sito web, basta creare una pagina index.htm o simili, caricarla nel tuo spazio web, e digitare il nome di dominio per visualizzarla, nel mondo android, questo viene fatto con una singola istruzione che prende il nome di **setContentView()**

### STEP 3: Dialogo semiserio tra un utente e una giovane app

**Utente** - Buongiorno *PrimaPagina*, mi faresti vedere la tua prima pagina. Sono proprio curioso...

**APP** - Ok, aspetta... mi preparo. Prendo dal mio zaino... dovrebbe essere questa.. eccola .. l'istruzione: setContentView()

**APP** - Ehm, non sono ancora molto esperta. Potresti dirmi quale prima pagina?

**Utente** - Certo, hai ragione. Mostrami quella interfaccia grafica - quel file .xml - che tuo padre ha creato e che ha chiamato *pagina1*.

**APP:** Volentieri! Potresti dirmi per favore dove si trova questa interfaccia grafica, questo layout?

**Utente:** Che palle!

**APP:** ..? dove scusa...

**Utente:** ohps...Certo si trova dentro alla cartella Layout di risorse.

**APP:** E come faccio a indicare il percorso dove recuperarla?

**Utente:** Ma insomma ... sei proprio alle prime armi... Non sai ... una maz.... Ti spiego: si adotta una sintassi simile a quella per recuperare un'immagine, solo che al posto di indicare un percorso stile "images/nomeimmagine.png", si usano delle abbreviazioni e una notazione con il punto al posto delle /

**APP:** Ah ho capito. Semplice. Potrei allora indicare il percorso così: layout.pagina1

**Utente:** Mi sembra vada bene, ma non funziona. Non è che bisogna indicare un percorso completo?

**APP:** ehm... fammi pensare... Ci sono: mi sono dimenticata di aggiungere una R. Prova con: R.layout.pagina1

**Utente:** Funziona! ... che bella pagina che hai!

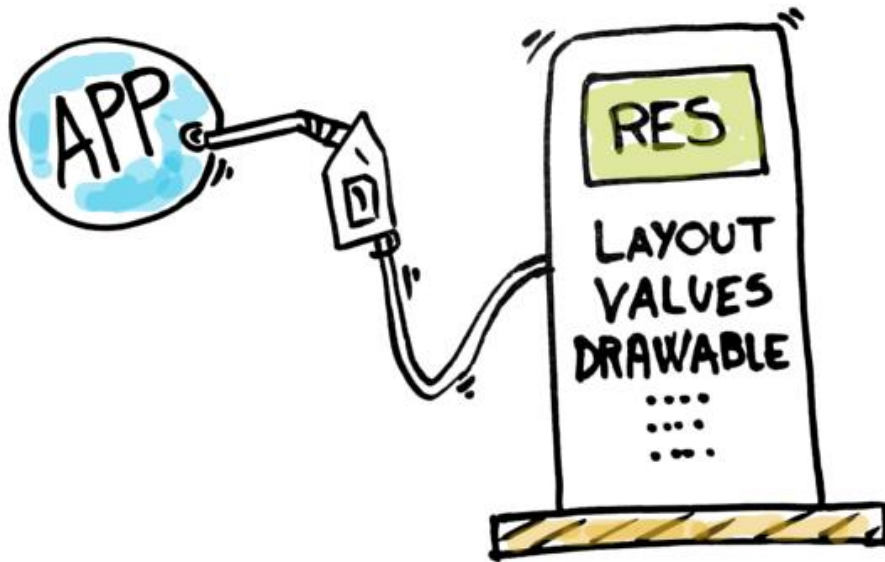
#### STEP 4: Il mio primo file java

Tutto questo dialogo si traduce nel codice qui sotto, che è lo stesso codice che trovi quando crei un nuovo progetto da zero in android.

```
1. public class MainActivity extends Activity {  
2.  
3.     @Override  
4.     protected void onCreate(Bundle savedInstanceState) {  
5.         super.onCreate(savedInstanceState);  
6.         // imposto il layout da visualizzare  
7.         setContentView(R.layout.pagina1);  
8.     ....
```

Non prendere paura dal codice, visto che gran parte viene generato in automatico da Eclipse. L'unica riga che devi conoscere per ora, è quella in cui viene richiamato il metodo **setContentView()** (capiremo cos'è un metodo in una delle prossime lezioni)





All'interno delle parentesi, si ha una notazione, come detto, diversa da quella classica del mondo delle pagine web, che, per chi ha già esperienza di programmazione in altri linguaggi ad oggetti, è sicuramente familiare, ma per tutti gli altri, serve a **identificare** una particolare **risorsa** interna alla cartella "res", in particolare la risorsa che si trova dentro alla cartella "layout" che vogliamo mostrare al navigatore.

#### **R.cartella.nomerisorsa**

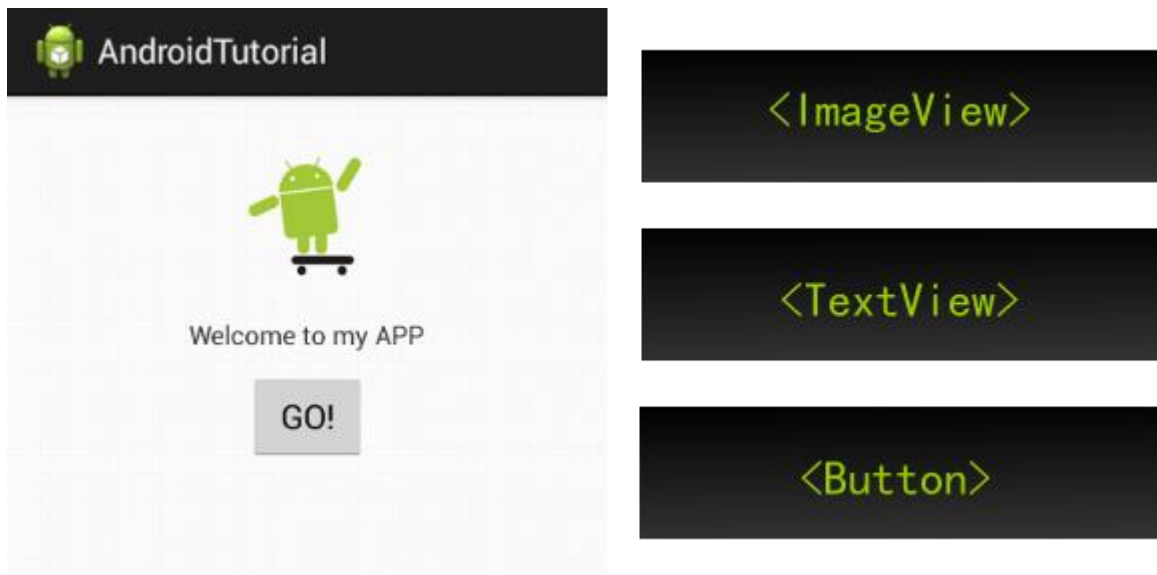
Nella prossima lezione vedremo come creare una seconda pagina per la nostra app e come richiamarla grazie al tocco su di un bottone o link.

## Anatomia di un linear layout con testo immagine e bottone cliccabile

*TUTORIAL 6: In questo tutorial vedremo come aggiungere TESTO, IMMAGINI, BOTTONI per creare un semplice linear layout. Imparerai ad impostare alcune delle proprietà di questi elementi (View) e altre caratteristiche della cartella risorse "res", che abbiamo iniziato ad esplorare nei precedenti tutorial.*

Abbiamo già visto come si progetta un semplice layout da usare come rappresentazione grafica per le nostre Activity. Ora che abbiamo qualche conoscenza in più sulle terminologie tipiche del mondo Android, come **file xml**, **cartella risorse**, **linear layout**, cerchiamo di creare la **home page** della nostra applicazione, che avrai discusso con il tuo cliente ipotetico.

Dopo decine di incontri, finalmente decidi di creare questa "complessa" pagina home per l'applicazione, costituita da un'immagine, da un testo e da un pulsante per accedere alla seconda Activity.



Se hai dei dubbi su come si possa creare una pagina come questa, ti rimando al tutorial "[Progettare un semplice Layout in Android](#)". Se proviamo ad analizzare il codice XML di tale pagina, scopriremo delle interessanti caratteristiche.

### STEP 1: Il codice xml della pagina

Dopo aver trascinato un elemento immagine, un elemento testo, e un bottone, il codice xml risultante sarà quello visualizzato qui sotto.

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

```
5. android:gravity="center | top"
6. android:orientation="vertical" >
7.
8. <ImageView
9. android:id="@+id/imageView1"
10. android:layout_width="wrap_content"
11. android:layout_height="wrap_content"
12. android:contentDescription="myapp"
13. android:paddingBottom="20dp"
14. android:paddingTop="30dp"
15. android:src="@drawable/logo" />
16.
17. <TextView
18. android:id="@+id/textView1"
19. android:layout_width="wrap_content"
20. android:layout_height="wrap_content"
21. android:clickable="false"
22. android:text="@string/welcome_main" />
23.
24. <Button
25. android:id="@+id/button1"
26. android:layout_width="wrap_content"
27. android:layout_height="wrap_content"
28. android:layout_marginTop="10dp"
29. android:text="GO!!" />
30. </LinearLayout>
```

Analizziamo in dettaglio ogni elemento della nostra pagina, per avere una panoramica dei primi attributi che è possibile impostare per qualsiasi layout

### Box Linear Layout <LinearLayout>

L'elemento root è di tipo Linear Layout. Come già detto un linear layout può contenere i propri "figli" sia a livello di singola riga, uno a fianco all'altro, sia a livello di colonna, uno sotto l'altro. Per fare questo è necessario specificare un particolare attributo "*orientation*" che ti diventerà familiare.

1. `<?xml version="1.0" encoding="utf-8"?>`
2. `<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"`
3. `android:layout_width="match_parent"`
4. `android:layout_height="match_parent"`
5. `android:gravity="center | top"`
6. `android:orientation="vertical"`
7. `android:background:"@color/white" >`
- 8...

Ad esempio per disporre ogni elemento, **uno sotto l'altro**, quindi con un orientamento a **colonna**, dovrai impostare l'orientamento a `"vertical"`. Il valore predefinito è sempre `"horizontal"`

Come larghezza `"layout_width"` e altezza `"layout_height"` del nostro layout, abbiamo usato il valore `"fill_parent"`, per ottenere il massimo spazio consentito dall'altezza del padre, che nel nostro caso è proprio il contenitore linear layout. È possibile utilizzare `"wrap_content"`, per usare la minima quantità di spazio necessario a rappresentare ogni elemento nel layout.

Come unità di misura per l'altezza e la larghezza, possiamo usare dp, px, unità.

### Come mantenere l'elemento centrato orizzontalmente?

Per mantenere ogni elemento figlio nel centro orizzontalmente nel layout abbiamo usato l'attributo `"gravity"` impostato a `"center_horizontal"`

### Come impostare il colore dello sfondo?

Per impostare lo sfondo del layout (es. grigio), abbiamo usato questa notazione `"@color/grigio"` apparentemente complessa, se vista la prima volta. Niente paura.

In Android, così come visto per il layout, è possibile memorizzare su un **file esterno XML**, tutte le risorse stringa, colore, ecc. Questo offre numerosi vantaggi, il più evidente è che se volessi modificare al volo decine di pagine, non dovrei entrare in ognuna e modificare il dato, ma semplicemente accedere al file xml comune, e cambiare solo lì il dato, per automaticamente riflettersi su tutte le pagine.

Se analizziamo ancora una volta la cartella `"res"`, scopriremo proprio la sottocartella cartella `"string"`, al cui interno è appunto possibile memorizzare un file XML con nome **string.xml**.

Per aggiungere allora un colore con nome `gray`, basta aprire il file `string.xml`, fare clic su `Aggiungi>colori>OK`. Ora, nel campo nome scriveremo `"gray"` e nel campo Valore inserisci `#CCC` (`#` è introduttivo, quindi è necessario mettere il mettere un valore esadecimale del colore in formato ARGB)

Il logo: `<ImageView>`

Per **inserire il logo** sulla parte superiore abbiamo usato l'elemento (View) che prende il nome di **ImageView**. Come valore per l'attributo `layout_width` e `layout_height`, abbiamo usato `"wrap-content"`, proprio per avvolgere il contenuto nello lo spazio minimo necessario alla visualizzazione del logo.

```
1.<ImageView android:id="@+id/imageView1"  
2. android:layout_width="wrap_content"  
3. android:layout_height="wrap_content"  
4. android:contentDescription="myapp"  
5. android:paddingBottom="20dp"  
6. android:paddingTop="30dp"  
7. android:src="@drawable/logo" />
```

### Dove e come vado a prelevare l'immagine?

L'origine dell'immagine viene selezionata con l'attributo `"src"` esattamente come avviene nel mondo web. Come sorgente dell'immagine abbiamo usato la notazione `"@drawable/logo"` che immagino inizi a diventare familiare.

Si riferisce ad un file di immagine denominato `"logo"` situato in cartella `drawable` del progetto. La domanda che potrebbe sorgere è: ma in quale delle diverse cartelle `"drawable"`?

Per i dispositivi con alta risoluzione (HDPI) dovrai inserire l'immagine di risoluzione più elevata nella cartella `drawable-hdpi`, per il file di risoluzione inferiore sceglierai `drawable-ldpi` e per file di risoluzione inferiore posto in `drawable-ldpi`. Quindi, copiare un file chiamato `logo.jpg` o `logo.png` (manipolazione. Gestire file GIF in Android è difficile)

Per mantenere un certo margine sulla parte superiore della nostra immagine, il nostro elemento `ImageView`, abbiamo usato l'attributo `"layout_marginTop"`. È possibile in alternativa utilizzare `"layout_margin"` per mantenere un margine su tutti i lati. In sostanza si ottiene lo stesso effetto che si ha con i CSS e la proprietà `margin`.

### Il bottone: <Button>

Per inserire un pulsante, inserire un componente **Button**. Fornire le informazioni di layout. Ecco un nuovo attributo `android:id`. Questo è utilizzato per stabilire una connessione tra il layout e codice java che andremo a scrivere, esattamente come avviene per l'attributo `id` nell'html e javascript.

```
1.<Button android:id="@+id/button1"  
2. android:layout_width="wrap_content"  
3. android:layout_height="wrap_content"  
4. android:layout_marginTop="10dp"
```

5. `android:text="GO!" />`

Nei prossimi tutorial vedremo come usare questo pulsante per entrare proprio nella seconda Activity della nostra applicazione. Quindi dovremo avere un sistema per **referenziarlo in modo univoco**, senza sbagliare. Per impostare l'id per ogni componente, è necessario seguire il formato "**@+id/nomeComponente**" (non prendere paura se non lo ricordi)

Per impostare l'etichetta del pulsante, usiamo l'attributo "text". In teoria potremmo scrivere qui il testo, ma non è una buona pratica e te lo sconsiglio, sebbene apparentemente è più veloce del metodo indicato qui sotto.

Qual è il metodo consigliato? Si deve mettere il testo nel file xml visto in precedenza, e poi referenziarlo in questo attributo "text" con la solita notazione @string/nomestr. Quindi dovrai aprire il file string.xml, e fare clic su Aggiungi>Stringa>OK. Ora, nel campo Name, inserisci "home\_btn" e nel campo Value, scrivi il testo "GO!"

Il risultato che si otterra' equivale ad aver scritto direttamente `android:text="GO!"` come il codice sopra.

A questo punto, sarebbe bello poter dare la possibilità al navigatore di **clickare sul bottone** e accedere ad un'altra pagina, ma questo lo vedremo nel prossimo tutorial.

## Gestire il tocco su di un bottone

**TUTORIAL 7:** In questo tutorial vedremo le prime importanti istruzioni java da usare, per rendere cliccabile un pulsante precedentemente inserito all'interno di un layout.

Ora che abbiamo progettato il nostro primo layout, con alcuni degli elementi tipici di una app, come il testo, le immagini e i bottoni, vediamo come modificare il codice del file java associato alla nostra Activity, in modo da inserire una delle azioni più semplice ma al tempo stesso più importanti per una applicazione: il passaggio **da una pagina ad un'altra** sulla base del **click o tocco** effettuato su di un un elemento della nostra app, tipicamente associato ad un bottone, immagine o riga di un elenco di informazioni (vedi Liste)

### Cosa imparerai

Vedremo due importanti metodi di java:

- Imparerai ad **individuare** nel codice, **le risorse inserite nei file xml** tramite l'uso del metodo: **findViewById()**
- Imparerai a scrivere nel codice java, le righe per aggiungere un **gestore di evento Click** su di un elemento del nostro layout.

Come visto nel tutorial "[Il primo passo prima di chiamare una Activity](#)", l'istruzione che permette di far apparire magicamente il nostro primo layout, è il metodo `setContentView(pagina_da_mostrare)`, dove pagina non è altro che un riferimento alla risorsa che si vuole prelevare, ossia il nostro layout o file .xml

```
1. public class PrimaPagina extends Activity {  
2.     @Override  
3.     protected void onCreate(Bundle savedInstanceState) {  
4.         super.onCreate(savedInstanceState);  
5.         setContentView(R.layout.main_page);  
6.     ....
```

Tutto questo codice ti ricordo è creato in automatico da Eclipse, non appena creiamo una nuova Activity. Il passo successivo è quello di scrivere le istruzioni necessarie a **rendere "intelligente"** il bottone inserito nel layout sviluppato nel precedente tutorial.

La prima domanda da farsi quindi è come faccio a identificare il bottone con delle istruzioni java inserite nel codice, per poi aggiungergli delle istruzioni che lo rendano sensibile al tocco.

## Identificare risorse tramite id: findViewById()

Ogni elemento inserito all'interno del layout della nostra pagina, ha un importante attributo, che è possibile visualizzare aprendo direttamente il file .xml. Questo attributo è l'id, un po' come accade con le pagine web, quando vado a definire l'attributo di id per un tag, al fine di poterlo referenziare con javascript o jQuery.

Se infatti rivediamo il codice xml del nostro bottone inserito:

1. `<Button android:id="@+id/button1"`
2. `android:layout_width="wrap_content"`
3. `android:layout_height="wrap_content"`
4. `android:layout_marginTop="10dp"`
5. `android:text="GO!" />`

noteremo nella prima riga che esiste l'attributo id e che ha uno specifico valore pari a "**button1**".

Questo cosa significa? Significa che per identificare nel codice java, questo bottone, potrò usare proprio il suo **nome definito nell'attributo id**. Questa è una regola che si applica non solo ai bottoni, ma a qualsiasi elemento che andrai ad inserire nel tuo layout.

La prima regola importante che è necessario imparare quindi è: **assegna sempre un nome** all'attributo id di ogni elemento che andrai ad inserire nel tuo layout.

Ma a livello di codice come posso riferirmi a questo pulsante? Si può usare un metodo **findViewById()**, il quale risponde sostanzialmente alla domanda: cerca l'elemento con attributo id specificato tra parentesi. Ma noi già sappiamo come identificare le risorse nella cartella "res", quindi basterà scrivere:

1. `findViewById(R.id.button1);`

Nota il punto e virgola finale: questa è un'altra caratteristica del linguaggio java, che imparerai a conoscere bene .

## Trasportare le risorse in più punti del codice: le VARIABILI

Anche se presumo tu abbia già delle basi di programmazione, vediamo per i meno esperti un altro concetto importante, ossia quello di variabile.

Così come usiamo le bottiglie per trasportare il vino o l'olio, anche in java è possibile usare delle bottiglie. È una semplificazione, ma giusto per capire il concetto. Il nostro vino, è il bottone che siamo riusciti ad individuare. Per poterlo spostare e usare in più punti del programma, dobbiamo inserirlo all'interno di una bottiglia, che nel linguaggio della programmazione, prende il nome di **variabile**.



Le variabili posso avere i nomi che decidi tu, ma devi rispettare alcune regole base, come non iniziare con numeri, non usare caratteri non alfanumerici, spazi etc. Ad esempio: indirizzo, potrebbe essere il nome di una variabile.

Così come il bravo imbottigliatore, prende l'imbuto per travasare il vino da una botte alla bottiglia, allo stesso modo, il programmatore usa il simbolo = per "travasare" del contenuto (in realtà non è proprio così ma giusto per semplificare)

Pertanto se la mia bottiglia si chiamasse, btnHome, per "travasare" il bottone identificato in precedenza, si userà:

1. // Identifico il bottone Button1 e lo travaso in una variabile
2. btnHome = findViewById(R.id.button1);

Questo il codice che abbiamo scritto fino ad ora:

1. **public class** AndroidTutorial **extends** Activity {
2.     @Override
3.     **public void** onCreate(Bundle savedInstanceState) {
4.         **super**.onCreate(savedInstanceState);
5.         setContentView(R.layout.main);
6.         // Identifico il bottone tramite il proprio id, Button1
7.         Button btnHome=(Button)findViewById(R.id.button1);

### Gestire il CLICK/TOCCO su di un bottone: `setOnClickListener()`

A questo punto non ci rimane che trovare un sistema per **rendere questo bottone "intelligente"** e fare in modo che si accorga se qualcuno lo tocca. Per fare questo si devono introdurre i **gestori di evento** in Android. In sostanza funzionano come degli allarmi: non appena scattano, significa che l'evento che stavo monitorando si è verificato e posso prendere delle decisioni.

Per rendere sensibile al click il nostro bottone identificato con la variabile di nome btnHome, dovremo allora scrivere:

1. //aggiungo una "sentinella" al bottone - gestore di evento
2. btnHome.setOnClickListener(...)

dove all'interno delle parentesi dovremo inserire una serie di altre istruzioni che verranno eseguite solo al verificarsi dell'evento monitorato.

1. btnHome.setOnClickListener(**new** OnClickListener){
2.     @Override

```
3. public void onClick(View arg0) {  
4.     // azioni da fare  
5. }  
6. });
```

Non prendere paura per il codice. Ti diventerà familiare molto velocemente e soprattutto non dovrai ricordarlo a memoria non appena lo inseriremo all'interno della nostra lista di snippet di codice da usare velocemente (vedi tutorial sulla creazione di snippet di codice).

E cosa dobbiamo fare non appena si verifica questo evento? Dovremo **aprire una nuova pagina**, quindi una nuova Activity, che avremo cura di identificare in modo opportuno. Pertanto il codice completo che dovremo scrivere sarà:

```
1. package com.creareapp.androidtutorial;  
2. ...  
3.  
4. public class AndroidTutorial extends Activity {  
5.     @Override  
6.     public void onCreate(Bundle savedInstanceState) {  
7.         super.onCreate(savedInstanceState);  
8.         setContentView(R.layout.main);  
9.         Button btnHome=(Button)findViewById(R.id.button1);  
10.        btnHome.setOnClickListener(new OnClickListener(){  
11.            @Override  
12.            public void onClick(View arg0) {  
13.                startActivity(...);  
14.            }  
15.        });  
16.    }  
17. }
```

Le righe che vedi qui sopra non contengono elementi personalizzati nel senso che sono righe standard, quindi le potrai copiare e incollare in ogni tuo progetto per ottenere lo stesso risultato. La nuova istruzione che non abbiamo ancora visto è quello che ci permette di **avviare una nuova Activity**, quindi di lanciare il file java, grazie al quale verrà prelevato il layout da mostrare al navigatore.

Tutto questo si fa usando il metodo **startActivity()** che inizieremo ad approfondire dal prossimo tutorial.

## Richiamare un'Activity da un'altra pagina con Intent

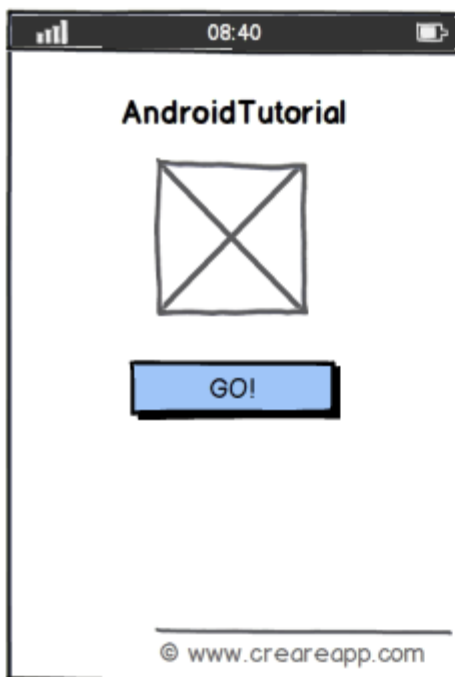
*Tutorial 8: In questo tutorial vedremo uno dei primi importanti meccanismi per animare un pò le nostre app e riuscire così a passare da una pagina ad un'altra, sulla base di un click o tocco fatto su di un bottone.*

Nel tutorial precedente, abbiamo visto come rendere "intelligente" un bottone inserito in un apposito layout, in modo che al verificarsi di un certo evento, il CLICK o tocco, accada qualcosa. Questo qualcosa potrebbe essere ad esempio l'apertura di una nuova pagina, quindi l'attivazione di una nuova Activity.

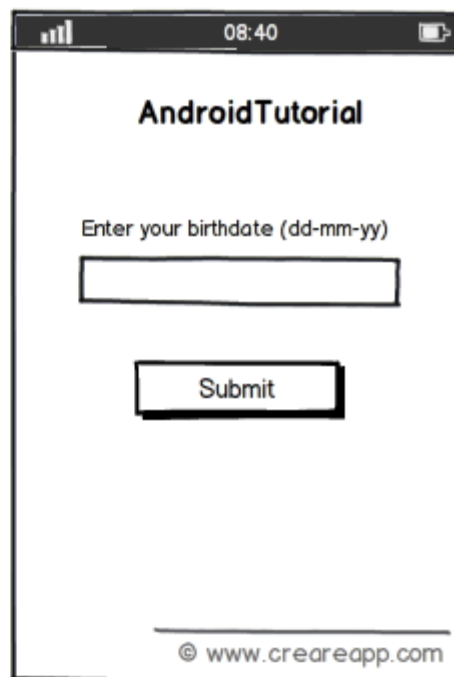
### Cosa imparerai

- Imparerai uno dei concetti più importanti di Android ossia l'uso di Intent
- Imparerai a passare da una Activity ad un'altra e a gestirle in modo corretto nel file manifest.xml

Ipotizzando allora di aver già creato questa seconda Activity e di averla chiamata "Page1.java" (vedi il tutorial: Le pagine web in Android: Activity) e ipotizzando che questa Activity mostri, tramite setContentView() un layout progettato in precedenza con all'interno un testo, un campo di input e un bottone, il passo successivo sarà quello di completare il codice da inserire all'interno del metodo startActivity() visto nel tutorial precedente.



MainActivity.java



Page1.java

Definire la seconda Activity Page1.java

Giusto per avere del codice da usare per ricreare l'esempio, potremmo definire il layout che mostrerà la nostra Activity di nome Page1.java, in questo modo:

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.     android:layout_width="match_parent"
4.     android:layout_height="match_parent"
5.     android:gravity="center | top"
6.     android:orientation="vertical" >
7.
8. <TextView
9.     android:id="@+id/textView1"
10.    android:layout_width="wrap_content"
11.    android:layout_height="wrap_content"
12.    android:layout_marginTop="20dp"
13.    android:clickable="false"
14.    android:text="@string/page_top"
15.    android:textSize="20dp" />
16.
17. <EditText
18.     android:id="@+id/editText1"
19.     android:layout_width="234dp"
20.     android:layout_height="wrap_content"
21.     android:layout_marginTop="30dp"
22.     android:ems="10"
23.     android:hint="Enter your Birthdate" >
24. <requestFocus />
25. </EditText>
26.
27. <Button
28.     android:id="@+id/button1"
29.     android:layout_width="wrap_content"
30.     android:layout_height="wrap_content"
```

```
31. android:layout_marginTop="10dp"
```

```
32. android:text="Submit" />
```

```
33. </LinearLayout>
```

mentre a livello di codice java, non faremo altro che richiamare il solito metodo setContentView()

```
1. package com.creareapp.androidtutorial;
```

```
2.
```

```
3. import android.os.Bundle;
```

```
4. import android.view.View;
```

```
5. import android.view.View.OnClickListener;
```

```
6. import android.widget.Button;
```

```
7. import android.app.Activity;
```

```
8. import android.content.Intent;
```

```
9.
```

```
10. public class Page1 extends Activity {
```

```
11.
```

```
12. @Override
```

```
13. protected void onCreate(Bundle savedInstanceState) {
```

```
14.     super.onCreate(savedInstanceState);
```

```
15.     setContentView(R.layout.page1);
```

```
16. }
```

```
17. }
```

### Attivare elementi della nostra app con Intent()

Una delle operazioni che spesso dovrai fare all'interno di una app in Android, è richiamare una nuova Activity. Ebbene in Android hanno sviluppato un meccanismo molto semplice, che non solo permette di attivare ogni componente di una applicazione, quindi anche una Activity, ma in generale di **trasmettere delle informazioni** tra componenti e di farli comunicare l'uno con l'altro o con altre applicazioni.

Basti pensare ad esempio a quello che succede non appena si riceve una chiamata al telefono. Il sistema attiva una applicazione che mostra all'utente un layout con il numero del chiamante e una serie di pulsanti per rispondere.

Questo meccanismo base prende il nome di **Intent()** e possiamo pensarlo come ad una "Intenzione" di fare qualcosa.

### Attivare e collegare una seconda Activity

Ora che sappiamo che istruzioni usare per richiamare una Activity dal codice, ossia usando `startActivity()`, vediamo come usare gli Intent per attivare a tutti gli effetti la nostra Activity.

```
1. // definisco l'intenzione di aprire l'Activity "Page1.java"
2. Intent openPage1 = new Intent(MainActivity.this,Page1.class);
```

Ho definito una variabile di tipo Intent con nome `openPage1`, che mi servira' poi all'interno di `startActivity()`. All'interno delle parentesi di Intent ho inserito due parametri, il primo che identifica l'attuale Activity (nota la presenza della parola `.this` finale), mentre il secondo che identifica l'activity che vorremmo richiamare (nota la presenza della parola `.class` finale).

Questo tipo di chiamata al nostro Intent si dice **esplicita**, in quanto diciamo proprio noi che tipo di activity lanciare.

```
1. // definisco l'intenzione di aprire l'Activity "Page1.java"
2. Intent openPage1 = new Intent(MainActivity.this,Page1.class);
3. // passo all'attivazione dell'activity page1.java
4. startActivity(openPage1);
```

Pertanto riprendendo il codice visto nel precedente tutorial, potremmo concludere l'esempio:

```
1. package com.creareapp.androidtutorial;
2. ...
3.
4. public class MainActivity extends Activity {
5.     @Override
6.     public void onCreate(Bundle savedInstanceState) {
7.         super.onCreate(savedInstanceState);
8.         setContentView(R.layout.main);
9.         Button btnHome=(Button)findViewById(R.id.Button1);
10.        btnHome.setOnClickListener(new OnClickListener(){
11.            @Override
12.            public void onClick(View arg0) {
13.                // definisco l'intenzione
```

```
14.         Intent openPage1 = new Intent(MainActivity.this,Page1.class);
15.         // passo all'attivazione dell'activity page1.java
16.         startActivity(openPage1);
17.     }
18. });
19. }
20. }
```

Cosa succederà allora non appena l'utente cliccherà sul bottone identificato dall'id "Button1"?

- Il gestore di evento riconosce il **verificarsi di un click**
- Fa partire le istruzioni presenti **dentro** al gestore onClick
- Esprimo la mia **intenzione di attivare una Activity** di nome "Page1"
- Faccio partire l'activity con **startActivity()**
- "Magicamente" appare sullo schermo il nuovo layout, che oscura il precedente.

Se tutto è corretto, in teoria l'utente dovrebbe visualizzare sullo schermo proprio il layout definito nell'istruzione setContentView() dell'activity di nome "Page1".

In realtà se testiamo il tutto, potrebbe verificarsi un errore. E questo perchè ogni Activity che andiamo a definire all'interno della cartella "src" deve essere anche "registrata" all'interno del file AndroidManifest.xml

In teoria questo avviene in automatico non appena aggiungiamo una nuova activity, ma se non dovesse accadere dobbiamo farlo manualmente andando all'interno del file con il doppio click, poi selezionando la scheda "Application" e tra i nodi presenti aggiungere "Add", cliccare sulla scheda "Application" e poi selezionare il nome dell'Activity aggiunta, completa di percorso, e un'etichetta.