TESTARE E CREARE APPLICAZIONI TESTUALI JAVA PER ANDROID CON ANDROID

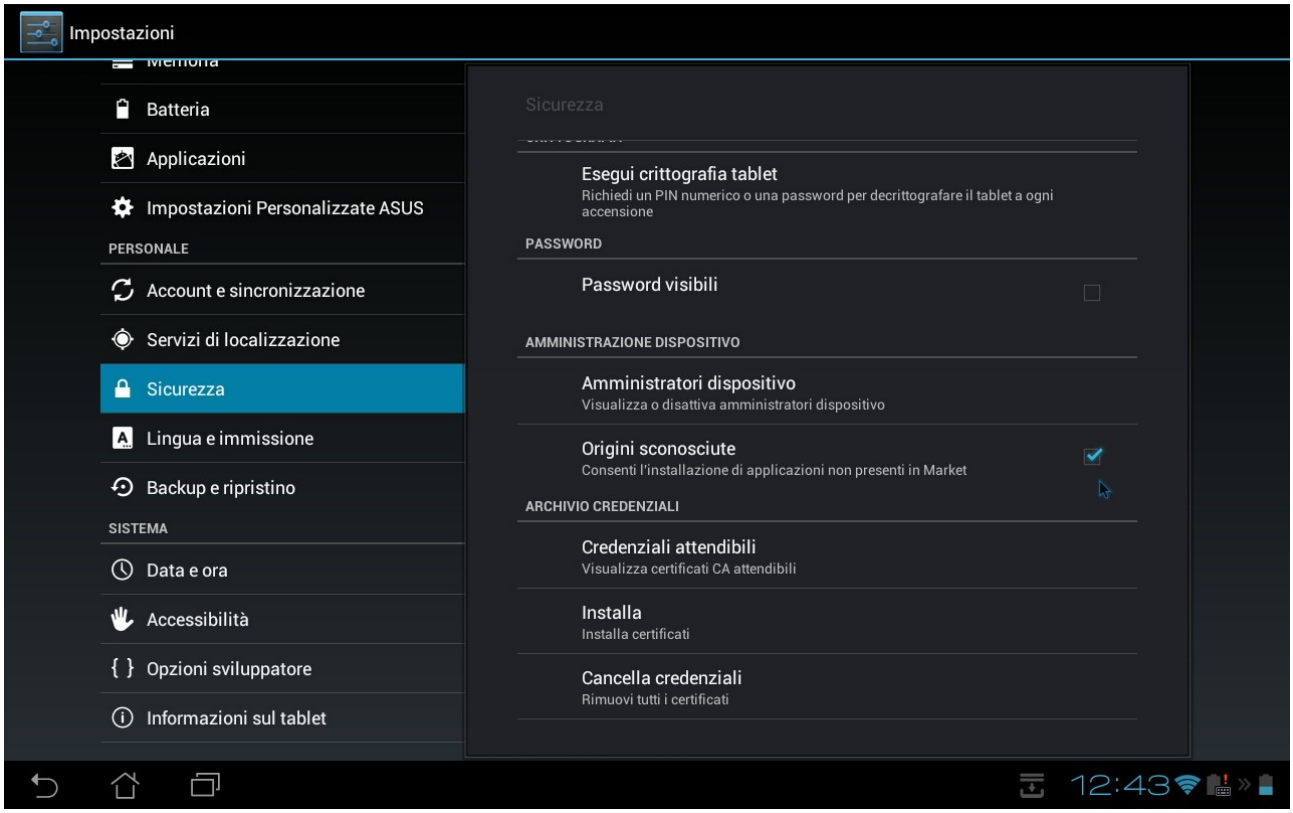
Ho deciso di scrivere questa guida per tre motivi principali:

1) Avendo un tablet che ha la possibilità di essere usato per programmare, mi sembra improponibile non poter creare delle applicazioni testuali in Java in Android.
2) Se devo testare delle classi in Java, mi pare un po' troppo dover creare un'applicazione grafica.
3) Rendere Android il più possibile separato dal pc, perché mi sembra cosa buona e giusta poter creare applicazioni per Android direttamente su Android.

Bado alle ciance, cominciamo.

REQUISITI MINIMI:

Per prima cosa assicuratevi di avere abilitato la spunta *Origini sconosciute* da *Impostazioni* → *Sicurezza* in Android.



Dopo aver fatto questo, installate dal *PlayStore* le seguenti applicazioni:

AIDE – Android Java Editor
cmdConsole

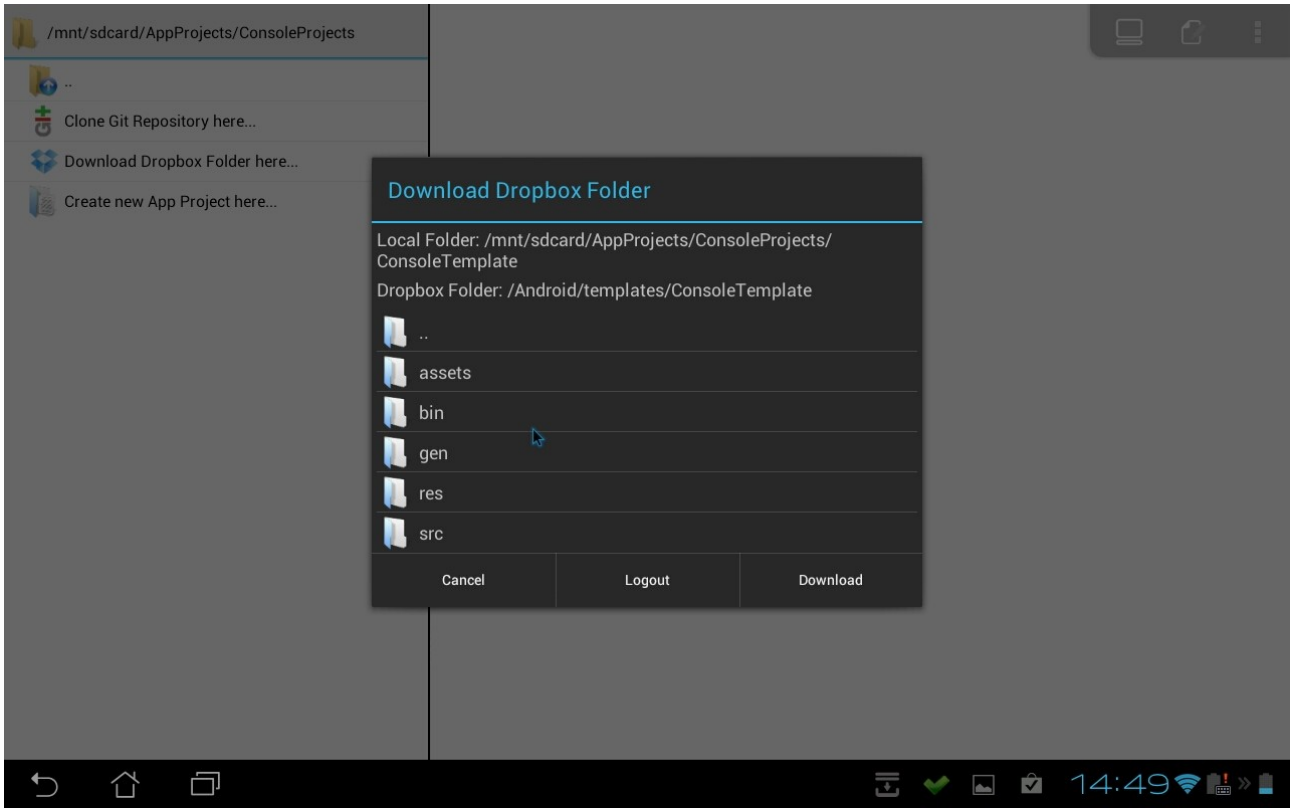
ESEMPIO: JCalc

Fingiamo che vogliamo fare una semplice calcolatrice, che scelta un'operazione, dati due numeri in input, ne sputa fuori il risultato, il tutto come programma testuale.

Prima però scaricate questo .zip: https://dl.dropbox.com/u/15801118/ConsoleTemplate.zip

Potete scegliere di estrarre i dati da qualche parte della vostra */sdcard* e poi di aprire direttamente il progetto con **AIDE** posizionandovi al suo interno (quando sarete all'interno del progetto, comparirà una voce chiamata *"Open this App Project"*).

Io vi consiglio un altro metodo. Ovvero, estraete il contenuto dello zip da qualche parte in **Dropbox**, e poi aprite **AIDE**. Posizionatevi dove volete salvare il vostro progetto e cliccate su *"Download Dropbox Folder here..."*, entrate nella directory che volete scaricare e cliccate *"Download"*.



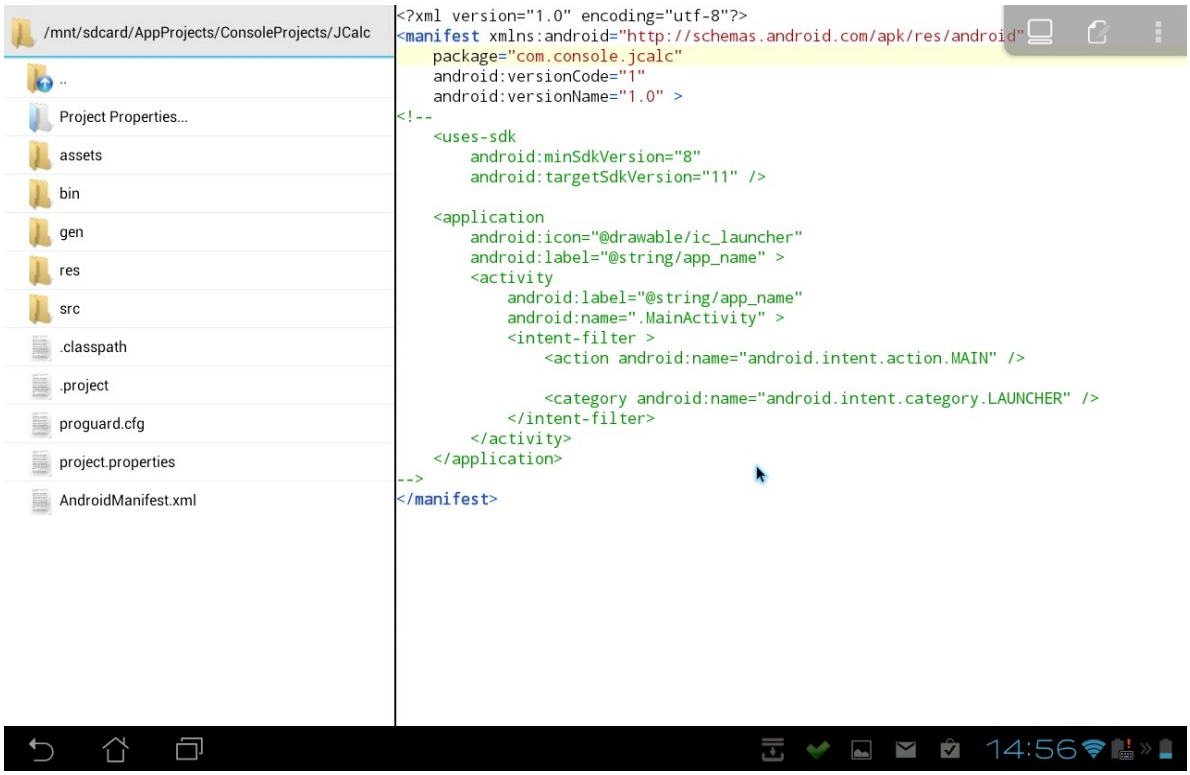
Dopo aver fatto ciò cancellate il file *.aidedropbox* così da rimuovere la sincronizzazione con **Dropbox**. In questo modo potrete avere un template per la vostra App sempre a portata di mano.

Ora dobbiamo "personalizzare" il nostro progetto, ovvero:
a) cambiare il nome del progetto da *ConsoleTemplate* in *JCalc*
b) rinonimare la directory *template* in *jcalc* (e quindi tutti i suoi riferimenti)

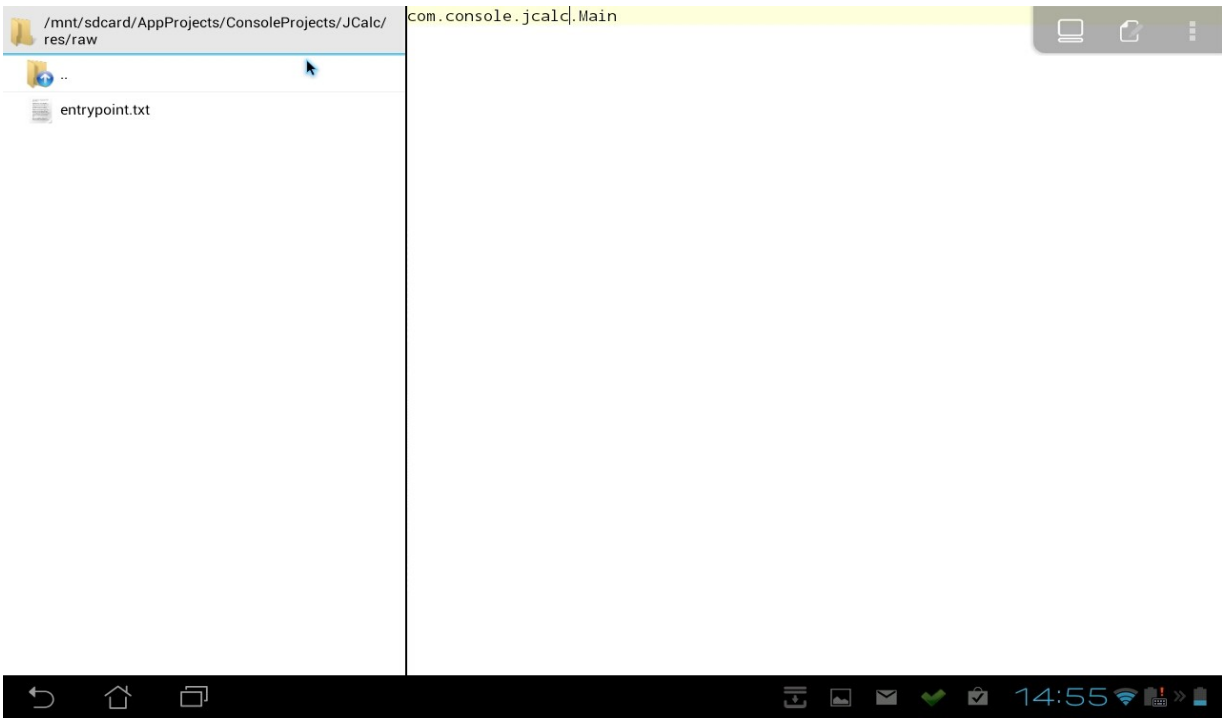
Per il primo punto basta rinonimare la directory principale.

Per il secondo punto invece, dobbiamo fare un paio di modifiche e rinonimazioni.

Aprite il file *AndroidManifest.xml.*
Modificate la seguente riga: *package=”com.console.template”*
in: *package=”com.console.jcalc”*



Fatto questo entrare in *res/raw*, da qui aprite il file *entrypoint.txt*.
Dopo aver aperto il file, cambiate l'unica riga: *com.console.template.Main*
in: *com.console.jcalc.Main*



Torniamo all'inizio del progetto (che ora si chiama *JCalc*) ed entriamo in *src/com/console*. Da qui cambiamo il nome della directory *template* in *jcalc*.

Entriamo nella rinonimata *jcalc* e vedremo che al suo interno di sono due file *.java*.
Il primo, *CmdApp.java* è la classe che ci darà i metodi che ci permetteranno la lettura e scrittura nella console, mentre la seconda è il *Main.java*, ovvero il programma principale che verrà lanciato all'avvio dell'applicazione.

Prima di cominciare a scrivere il codice, dobbiamo vedere i metodi di input e output.
Il metodo di input è *console.readln()* e restituisce un oggetto di tipo *String* (ovvero la stringa letta).
I metodi di output sono richiamabili attraverso *console.stdOut*, e sono tali e quali a quelli di *System.out*.

Main.java
/mnt/sdcard/AppProjects/ConsoleProjects/JCalc/src/com/console/jcalc

```
import java.util.*;
public class Main
{
    public static void main(HashMap<Integer, Object> args)
    {
        CmdApp console = new CmdApp(args);
        int choice, num1, num2, ris=0;
        String input;

        console.stdOut.println("1- somma; 2- sottrai; 3- moltiplica; 4- dividi");
        console.stdOut.print(">>");
        input=console.readln();
        choice=Integer.parseInt(input);
        console.stdOut.print("Inserisci il primo numero >>");
        input=console.readln();
        num1=Integer.parseInt(input);
        console.stdOut.print("Inserisci il secondo numero >>");
        input=console.readln();
        num2=Integer.parseInt(input);
        switch(choice)
        {
            case 1: ris=num1+num2;
            break;
            case 2: ris=num1-num2;
            break;
            case 3: ris=num1*num2;
            break;
            case 4:
                if(num2==0) console.stdOut.println("Divisione per 0 impossibile!");
                else ris=num1/num2;
            break;
        }
        if(!(choice==4 && num2==0)) console.stdOut.printf("Risultato: %d\n",ris);
    }
}
```

Come vedete da voi, l'applicazione è di una semplicitià triviale.
Stamperà a video una riga con gli output, quindi mostrerà il prompt, ed infine richiederà l'input. Letto l'input sarà trasformato in un valore intero. Stessa operazione per i due numeri. Infine, a variazione del caso scelto, stamperà il risultato (o non lo stamperà, nel caso della divisione per 0).

Ora è tempo di compilare e lanciare l'applicazione. Dal menù a destra cliccate *Run*.
Alla fine della compilazione (se non sono presenti errori di vario tipo) vi verrà chiesto se volete installare l'app. Ovviamente dite di no. Come no direte? Semplicemente perché a noi basta avere l'apk, non l'app installata nel nostro dispositivo.
È quindi tempo di aprire **cmdConsole**.

Come potete vedere **cmdConsole** si comporta come una qualunque console, tipo bash di Linux (per maggiori informazioni sui comandi vedere la sezione ***Approfondimenti***).
Il nostro file *.apk* si troverà dentro *JCalc/bin/*. Raggiunta quella posizione, per lanciarla ci basterà digitare il seguente comando:

> run JCalc.apk

E tadà, come vedete, la nostra applicazione funziona meravigliosamente.
Ora mi chiederete, perché non possiamo usare *System.out*? Il motivo è semplice, perché *System.out* non reindirizza l'output e l'input da **cmdConsole**, ma bensì nella shell *sh* di Android. Per visualizzare ciò bisognerebbe lanciare le nostre applicazioni con la **dalvikvm**, e vi posso assicurare che è diventa un pelino più complicato. ;)

Se lo ritenete necessario, è possibile chiamare la classe *Main.java* con nome differente, l'importante è che il nome della classe di lancio (ovvero quella che contiene lo *static void main(HashMap<Integer, Object> args)*) sia referenziata nel file *entrypoint.txt* (senza l'estensione *.java*!).

RIASSUNTO

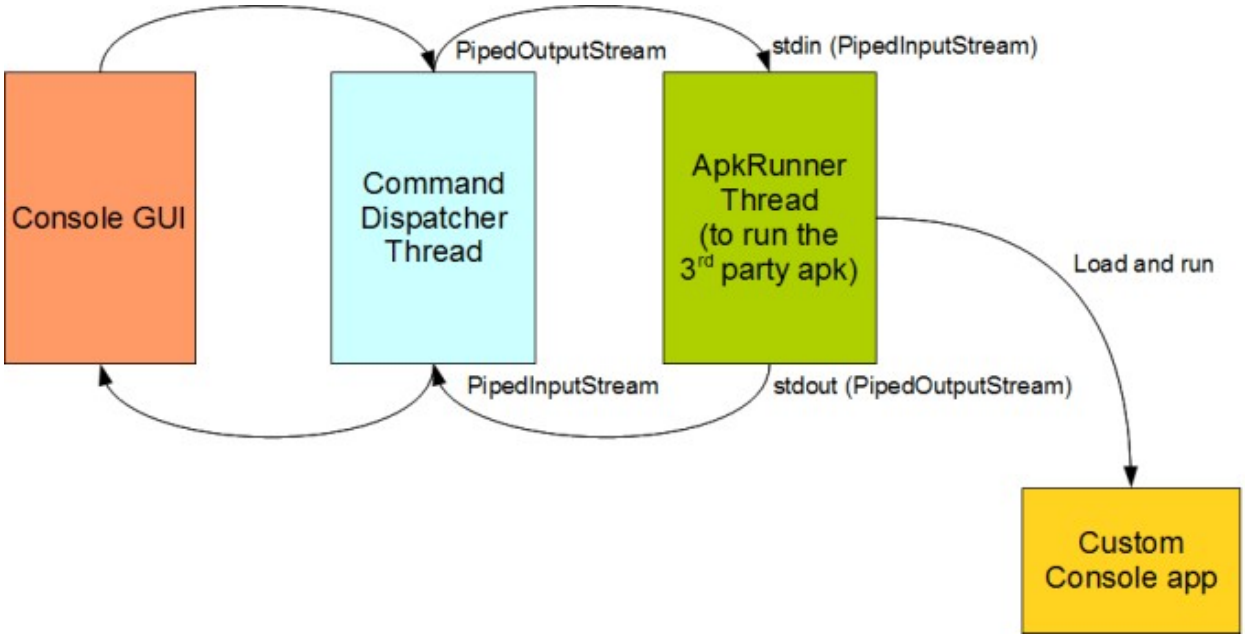
Legenda:
<ProjectName> nome del progetto
<projectname> nome della cartella contenente i sorgenti del progetto
<main> il nome della classe che lancia *static void main(HashMap<Integer, Object> args)*

Per la creazione di una applicazione testuale dobbiamo:

1) scaricare ed estrarre i file del template
2) rinonimare *ConsoleTemplate* in ***<ProjectName>***
3) rinonimare *template* in ***<projectname>***
4) modificare i riferimenti a template in *AndroidManifest.xml* ed *entrypoint.txt*:
   AndroidManifest.xml :
   package="com.console.template" → *package="com.console.****<projectname>****"*
   entrypoint.txt:
   *com.console.template.****<main>*** → *com.console.****<projectname>.<main>***
5) usare *console.stdOut* invece di *System.out* per l'output. Usare *console.readln()* invece per l'input (dove *console* è l'oggetto di tipo *CmdApp*).
6) ???
7) Profitto!

APPROFONDIMENTI

Come funziona **cmdConsole**.
Il modo più semplice per capirlo è vedere questo schema:



La GUI di **cmdConsole** genera un thread a variazione del comando digitato. Se il comando digitato è run, allora viene lanciata la nostra applicazione testuale.

Ora vediamo il contenuto di *HashMap<Integer, Object> args* che viene passato al main da parte di **cmdConsole**:

Indice	**Tipo**	**Contenuto**
args.get(0)	*android.app.Application*	Dati dell'applicazione
args.get(1)	*String[ ]*	Lista di comandi passati dopo il nome dell'applicazione. Lo zeresimo elemento non è il nome dell'applicazione!
args.get(2)	*java.io.InputStream*	Si comporta come *stdin* per **cmdConsole**
args.get(3)	*java.io.PrintStream*	Si comporta come *stdout* per **cmdConsole**
args.get(4)	*String*	Specifica il tipo di encoding dell'input.

È preferibile accedere ai contenuti di *args* attraverso l'oggetto di tipo *CmdApp* (*console* nell'esempio).

Ora vediamo i comandi di **cmdConsole** (i principali):

Comando	**Descrizione**	**Sintassi**
help	Mostra una descrizione accurata dei vari comandi	*help*
ls	Mostra una lista dei file e della directory	*ls*
pwd	Mostra la posizione attuale	*pwd*
clear	Pulisce lo schermo della **cmdConsole**	*clear*
cd	Passa ad un'altra directory	*cd <dir>*
run	Lancia un'applicazione testuale *.apk*	*run <app>.apk*
history	Mostra gli ultimi comandi dati	*history*
del	Elimina un file	*del <file>*
mkdir	Crea una directory	*mkdir <dir>*
cp	Copia un file/directory nella posizone dopo	*cp <src> <dest>*
fontsize	Aumenta la dimesione del font di **cmdConsole**	*fontsize <size>*
exit	Esce da **cmdConsole**	*exit*

Qual'ora la vostra applicazione vada in loop, basta premere il tasto menù e scegliere "*Kill Running App*".

DOWNLOAD

Qui troverete i link dove scaricare il software necessario:

AIDE: https://play.google.com/store/apps/details?id=com.aide.ui
cmdConsole: https://play.google.com/store/apps/details?id=com.sss.cmdconsole
ConsoleTemplate.zip: https://dl.dropbox.com/u/15801118/ConsoleTemplate.zip

BIBLIOGRAFIA

Guida alla creazione di una app testuale in Java scritta dall'autore di **cmdConsole**:
http://www.codeproject.com/Articles/202996/Write-a-console-app-on-Android-using-Java

Tutorial testato su:
Dispositivo: *ASUS Transformer Prime TF201*
Sistema Operativo: *Android Ice Cream Sandwich 4.0.3*

Autore: Claudio Borsato
Data: 12/9/2012