Sviluppo di applicazioni integrate con il canale di lettura ottica per piattaforma Android

Laureando: Marco Ciacco 578487
Relatore: Ch.mo Prof. Federico Filira
Corso di laurea: Ingegneria Informatica

Data Laurea 26 Luglio 2012
Anno accademico: 2011/2012

Indice

Capitolo 1

Introduzione

1.1 Obiettivi

L'obiettivo dello stage é creare un'applicazione per cellulari in grado di semplificare i pagamenti di: Bollette, Bollettini, Multe cercando il piú possibile di automatizzare il processo.

L'idea é quella di scattare una foto con il cellulare al documento (bolletta o multa che sia) e in automatico generare un SMS o un testo che ne consenta il pagamento.

1.2 Primo Approccio

La prima decisione da affrontare é scegliere come realizzare l'Applicazione, ovvero se realizzarla con il **linguaggio nativo (java)** oppure con uno strumento per lo sviluppo multi piattaforma.

1.3 PhoneGap

1.3.1 Cos'è phonegap?

PhoneGap é una Piattaforma di sviluppo applicazioni in HTML5 che permette di creare una sola applicazione da pubblicare su 6 piattaforme diverse: iOS, Android, Blackberry, Palm, WinMobile e Symbian.

1.3.2 I vantaggi di PhoneGap

I vantaggi di questo tipo di piattaforma sono lampanti:

Write Once Come già anticipato consente di realizzare un applicazione che può funzionare su praticamente tutti i tipi di cellulare piú diffusi in accordo con la filosofia *write once run anywhere*.

Html, css e javascript Semplicità nella creazione di un interfaccia grafica con la possibilità di utilizzare gli editor html per la creazione/visualizzazione/test dell'applicazione prima che questa venga inviata al cellulare.

Api Nonostante sia multi piattaforma si può accedere in maniera relativamente facile alle periferiche del cellulare come fotocamera, connessione di rete, GPS, invio SMS ecc...

1.3.3 Gli svantaggi di PhoneGap

Pesantezza e Inefficienza Il codice sorgente inserito in html non viene ricodificato/compilato per funzionare sulla piattaforma, ma semplicemente interpretato al momento[1], quindi l'efficienza é notevolmente ridotta. Nell'applicazione, oltre ai codici sorgenti scritti in html, va inviato anche tutto il software di phonegap che consente l'utilizzo delle API[2] del cellulare.

Difficoltà di debugging Nonostante phonegap metta a disposizione un debugger molto valido, quando il numero di righe di codice inizia a diventare consistente si fa difficile l'individuazione di eventuali errori. Inoltre, per alcune piattaforme, il testing su macchina virtuale può dare risultati molto diversi da quelli che si ottengono su una macchina fisica.

1.4 Codice nativo Android (Java)

Le applicazioni che girano in Android vengono scritte in Java, ovviamente con alcuni accorgimenti per quanto riguarda la grafica (che viene definita in xml, come descritto in seguito).

[1] un po' come fa il browser del computer durante con il codice html o javascript

[2] Con Application Programming Interface (API) si indica ogni insieme di procedure disponibili al programmatore, di solito raggruppate a formare un set di strumenti specifici per l'espletamento di un determinato compito all'interno di un certo programma.

1.4.1 I vantaggi

Efficienza Scrivere codice nel linguaggio nativo del OS Android ci garantisce la massima efficienza, inoltre non é necessario inviare pacchetti aggiuntivi affinché l'applicazione funzioni.

Supporto Il supporto e le guida per la creazione di applicazioni é decisamente piú ampio di quello di PhoneGap. La presenza di un IDE [3] avanzato come Eclipse [4] aiuta decisamente lo sviluppo dell'applicazione; vedremo in seguito come.

Disponibilità completa API Le API[5] del cellulare sono accessibili in maniera relativamente semplificata e allo stesso tempo approfondita, vedremo meglio nel resto della tesi come ciò sia possibile.

1.4.2 Gli svantaggi

Portabilità Il codice scritto per Android gira ovviamente solo su cellulari Android. Per poter essere eseguito su un tipo diverso di cellulare necessita una riscrittura completa.

Linguaggio É necessario imparare come funziona nello specifico un'applicazione Android e la filosofia con cui dev'essere strutturata.

[3]Integrated Development Environmentin, in italiano: ambiente di sviluppo integrato, (conosciuto anche come integrated design environment o integrated debugging environment, rispettivamente ambiente integrato di progettazione e ambiente integrato di debugging) é un software che aiuta i programmatori nello sviluppo del codice.

[4]Eclipse é un ambiente di sviluppo integrato multi-linguaggio e multi piattaforma. Ideato da un consorzio di grandi società quali Ericsson, HP, IBM, Intel, MontaVista Software, QNX, SAP e Serena Software, chiamato Eclipse Foundation sullo stile dell'open source.

[5]Application Programming Interface API (Interfaccia di Programmazione di una applicazione) si indica ogni insieme di procedure disponibili al programmatore, di solito raggruppate a formare un set di strumenti specifici per l'espletamento di un determinato compito all'interno di un certo programma.

Capitolo 2

Sviluppo di un Hello World

Che si scelga PhoneGap oppure l'applicazione nativa, il modo piú facile per testare l'applicazione é sicuramente quello di utilizzare Eclipse, quindi vediamo come configuralo affinché sviluppi per Android.

2.1 Installazione e configurazione Eclipse e ADT

Poniamo che sul computer sia installato l'ultimo JDK[1]. Come prima cosa dobbiamo scaricare *Eclipse IDE for Java Developers*[2] estrarlo ed eseguirlo. Al primo avvio ci chiede il path del nostro spazio di lavoro (*workspace*), scegliamo e clicchiamo su avanti. A questo punto dobbiamo scaricare l'SDK[3]. A seconda del sistema operativo usato si scarica e estrae il programma.

A questo punto dobbiamo installare l'ADT [4]: apriamo Eclipse e andiamo su Help > Install New Software. Si aprirá una finestra in cui dobbiamo inserire il server da cui vogliamo scaricare il plugin. Inseriamo questo indirizzo https://dl-ssl.google.com/android/eclipse[5] e selezioniamo tutti e 4 i pacchetti da installare.

Ora riavviamo Eclipse e selezioniamo Window ⇒ Preferences... (Mac OS X: Eclipse ⇒ Preferences), nelle schede di sinistra clicchiamo su Android.

[1] Java Development Kit, Il compilatore java, disponibile su http://www.oracle.com/technetwork/java/javase/downloads/index.html

[2] Il programma è disponibile sul sito http://www.eclipse.org/downloads/

[3] *Software Development Kit (piú brevemente SDK) é un termine che in italiano si può tradurre come pacchetto di sviluppo per applicazioni, e indica un insieme di strumenti per lo sviluppo e la documentazione di software. Scaricabile a questo indirizzo http://developer.android.com/sdk/index.html*

[4] *Android Development Tools (ADT) é un plugin per Eclipse IDE che é stato progettato per fornire un potente e integrato strumento con cui sviluppare applicazioni Android*

[5] in caso il download non dovesse partire, rimuovere pure la s da https

Impostiamo il percorso dove abbiamo estratto l'SDK, premiamo OK e quindi Salva.

2.1.1 Installare AVD

AVD é l'acronimo di Android Virtual Device, necessario per testare agevolmente le applicazioni. Per installarlo andiamo da eclipse, Window ⇒ Android Sdk Manager, andiamo a mettere la spunta sul pacchetto che ci interessa, nel nostro caso Android 2.2 (API 8) e clicchiamo su install package. Dopo l'installazione andiamo in Window ⇒ AVD Manager e clicchiamo su New. A questo punto decidiamo che macchina virtuale creare -noi scegliamo Android 2.2- digitiamo un nome a piacere e impostiamo la dimensione della SD (io ho impostato 100Mb e mi sono bastati). A questo punto per avviare la macchina clicchiamo su start e attendiamo il tempo di caricamento (circa 1-2 minuti): abbiamo un cellulare virtuale sul computer a nostra disposizione.

2.2 Sviluppo con Java

Adesso che abbiamo la macchina virtuale installata e tutto il resto collegato possiamo iniziare a creare la prima applicazione di prova. Crearla con eclipse é immediato e veloce. Andiamo su File ⇒ New Project... e selezioniamo Android Project, scegliamo il nome, la versione di android per cui lo si vuole sviluppare (nel nostro caso Android 2.2). Infine scegliamo il nome del package[6] composto da almeno due nomi [7] (nel nostro caso telerete.software) e

[6]Un Package rappresenta una collezione di classi ed interfacce che possono essere raggruppate in base alla funzione comune da esse svolta.

[7]I Package sono di solito definiti usando una struttura gerarchica, indicando i livelli di gerarchia con dei punti. Anche se i package più in basso nella gerarchia sono spesso chiamati sotto-package di altri package, non c'è nessuna relazione semantica. Il documento Java Language Specification stabilisce le convenzioni da adottare nei nomi dei package, così da evitare di pubblicarne due con lo stesso nome. Le convenzioni descrivono come creare nomi unici, in modo da assicurare che package di uso generale e di larga distribuzione non abbiano nomi che possano generare ambiguità.

In generale, un nome comincia con il dominio di primo livello dell'organizzazione che lo produce, seguito dal dominio e da altri eventuali sottodomini, elencati in ordine inverso. L'organizzazione può infine scegliere un nome specifico per quel particolare package. Inoltre, sempre per convenzione, i nomi dei package dovrebbero contenere solo lettere minuscole.

Ad esempio, se un'organizzazione canadese chiamata MySoft crea un package che si occupa di frazioni, chiamare il package ca.mysoft.fractions lo distingue da un altro package simile creato da un'altra compagnia. Infatti se una compagnia americana omonima crea un package sulle frazioni, ma lo chiama com.mysoft.fractions, le classi nei due package saranno tutte definite in namespace separati ed unici.

premiamo su fine.

Versione di Android Ho scelto la versione di Android 2.2 perché é la versione minima compatibile con ció che dovremo fare. Ci tengo a precisare che un applicazione creata per una versione é sicuramente compatibile con le versioni successive, ma non necessariamente con quelle precedenti. Android 2.1 e precedenti non supportano alcune API di cui avremo bisogno, ma vedremo in dettaglio piú avanti.

Abbiamo creato un progetto vuoto già funzionante, adesso andiamo a provarlo sulla nostra brava macchina virtuale. Come prima cosa andiamo su run ⇒ Run configurations. Nella scheda Android nel campo Project clicchiamo su Browse e scegliamo il nome della nostra applicazione, nel campo sottostante Launch Activity selezioniamo launch, clicchiamo sul menú a tendina e selezioniamo l'unica cosa selezionabile; nella scheda target selezioniamo Automatic,infine mettiamo la spunta alla nostra macchina virtuale creata e clicchiamo su run. A questo punto dopo qualche secondo sulla macchina virtuale apparirá la nostra prima applicazione.

2.3 Sviluppo con PhoneGap

Abbiamo visto nel paragrafo precedente come creare un Hello World in Java. Ora vediamo di crearne uno con PhoneGap. Possiamo scegliere di creare un nuovo progetto oppure utilizzare lo stesso che abbiamo usato in precedenza. Nel caso scegliessimo di crearne uno nuovo bisogna ripetere ció che abbiamo fatto in 2.2, a questo punto modifichiamo il metodo OnCreate nel modo seguente.

```
1 public class app extends DroidGap {
      /** Called when the activity is first created. */
3     @Override
      public void onCreate(Bundle savedInstanceState) {
5         super.onCreate(savedInstanceState);
          super.loadUrl("file:///android_asset/www/index.html");
7     }
}
```

Altre convenzioni per evitare le ambiguità e regole per dare nomi ai package quando in dominio Internet non può essere direttamente usato nel nome sono descritte nella sezione 7.7 della Java Language Specification.

Eclipse non accetta meno di due nomi separati da punto. Questo vuol dire che è impossibile scrivere codice per un package di primo livello. Questa scelta progettuale è probabilmente stata fatta per evitare che dei package abbiano lo stesso nome e vadano in conflitto

Adesso dobbiamo scaricare PhoneGap dal sito http://phonegap.com. Seguiamo la guida che troviamo su http://phonegap.com/start#android per generare l'applicazione.

Capitolo 3

Sviluppo dell'applicazione Bollettino

3.1 Progettazione

Come spiegato in 1.1, dovremmo essere in grado di:

- Scattare la foto
- Analizzare la foto estrapolando le informazioni (OCR[1])
- Produrre una stringa da inviare via SMS

3.1.1 La scelta tra PhoneGap e Java

Ora che abbiamo provato entrambe le strade abbiamo gli elementi per capire quale strada prendere.

Innanzitutto per produrre un'applicazione che utilizzi l'OCR é necessario trovare una libreria che ci consenta di farlo. Dopo una breve ricerca in internet ci si rende conto che una libreria scritta in javascript non esiste e realizzarla non é una soluzione accettabile considerato il tempo a disposizione. Il fatto che non esista libreria javascript non é un ostacolo dato che con PhoneGap tramite il metodo PhoneGap.exec() é possibile eseguire metodi Java. Il problema é il seguente: per ogni libreria non javascript che si utilizza bisogna trovarne una compatibile per ogni tipo di telefono. Quindi si perderebbe la portabilità dell'applicazione.

[1]I sistemi di riconoscimento ottico dei caratteri, detti anche OCR (dall'inglese optical character recognition) sono programmi dedicati alla conversione di un'immagine contenente testo, solitamente acquisita tramite scanner, in testo digitale modificabile con un normale editor.

In secondo luogo, questa applicazione deve riconoscere delle immagini, scansionarle e dare un output in termini di posizioni. Questo richiede una grande velocità di calcolo se si vuole che il processo termini in tempi utili, e abbiamo spiegato prima che la velocità di esecuzione di javascript é parecchio inferiore a Java.

Per queste ragioni ho scelto di utilizzare Java.

3.1.2 La ricerca di un OCR: Tesseract

Il primo obiettivo per la ricerca di un OCR era quello di trovare una libreria OCR Java. La prima che poteva fare al caso nostro è Java Object Oriented Neural Engine: il suo problema é che usa delle librerie di Java puro non supportate da Android.

Dopo numerose ricerche ho trovato Tesseract, un progetto OCR open source. Sviluppato originariamente come software proprietario dalla Hewlett-Packard tra il 1985 e il 1995, non venne piú aggiornato nel decennio successivo. Fu poi rilasciato come open source nel 2005 da Hewlett Packard e dall'Università del Nevada, Las Vegas, e rilasciato con la licenza Apache, versione 2.0. Lo sviluppo di Tesseract é attualmente sponsorizzato da Google [Tesseract - Wikipedia].

3.2 Inserimento di Tesseract nel progetto

Il problema di Tesseract é che non é scritto in java, bensì in C++. Sappiamo che in java è possibile l'inserimento di applicazioni native (già compilate) che in Java normalmente servono per eseguire operazioni non supportate dalle API native di Java (vedi ad esempio ridurre a icona una figura). Per android sarà lo stesso? Sì, grazie allo strumento messo a disposizione dagli sviluppatori android: L'NDK.

3.2.1 L'NDK

Cos'è l'NDK L'Android NDK[2] (native development kit) é uno strumento che consente di inserire del codice nativo in un'applicazione Android. La miglior guida che ho trovato per imparare ad utilizzare questo strumento é disponibile nella bibliografia sotto la voce [Marakana - Using NDK to Call C code from Androi
Leggerla non é fondamentale ai fini della creazione del progetto ma aiuta di molto a comprendere come lavora L'NDK. La cosa piú interessante da capire

[2]Scaricabile all'indirizzo http://developer.android.com/sdk/ndk/index.html

é la variabile env che viene passata dalla JVM [3] . Questa permette di trasformare oggetti, stringhe e valori dal C/C++ al Java e vice versa. Questo passaggio è necessario perché Java gestisce gli oggetti in maniera diversa da C/C++.

La variabile env) Poniamo ad esempio di creare un metodo molto semplice in C++ a cui venga passata una stringa.

```
String str= "hello";
chiamataMetodoNativo(str);
```

Quando Java crea la stringa hello la salverà con la sua codifica in uno spazio in memoria atto a contenerla, il cui indirizzo verrà salvato nella variabile *str*. Questo indirizzo viene infine passato al metodo nativo C/C++.

```
//C++ code
extern "C"
JNIEXPORT void JNICALL Java_ClassName_chiamataMetodoNativo
  (JNIEnv *env, jobject obj, jstring javaString)
{
    //Get the native string from javaString
    const char *nativeString =
        env->GetStringUTFChars(javaString, 0);

    //Do something with the nativeString

    //DON'T FORGET THIS LINE!!!
    env->ReleaseStringUTFChars(javaString, nativeString);
}
/*-------------------------------------------------------------------------
/*C code*/
JNIEXPORT void JNICALL Java_ClassName_chiamataMetodoNativo
  (JNIEnv *env, jobject obj, jstring javaString)
{
    /*Get the native string from javaString*/
    const char *nativeString = (*env)->GetStringUTFChars(env,
        javaString, 0);

    /*Do something with the nativeString*/

    /*DON'T FORGET THIS LINE!!!*/
    (*env)->ReleaseStringUTFChars(env, javaString,
        nativeString);
}
```

[3] Java Virtual Machine

N.B.: Il codice appena visto sarà l'implementazione di un'interfaccia C/C++ generata dalla compilazione con **javah**.

La variabile **env** è in grado di convertire la stringa in modo che C/C++ sia in grado di elaborarla. (tratto da [Java Native Interface - Wikipedia])

Con queste informazioni si é in grado di portare qualsiasi applicazione C/C++ in java, anche tesseract. Fare un lavoro del genere di interfacciamento é un lavoro lungo e impegnativo, per fortuna qualcuno ci ha messo a disposizione tesseract-android-tools, un progetto java/c++ praticamente già pronto per essere compilato

3.2.2 tesseract-android-tools

Tesseract android tools, come ho già, detto semplifica l'importazione di tesseract nel progetto. La guida che ho utilizzato per realizzarlo è disponibile nella bibliografia [Using Tesseract Tools for Android to Create a Basic OCR App].

Nonostante sia ben fatta io ho trovato numerosi problemi nella compilazione del progetto. Ecco gli accorgimenti che ho dovuto fare per rendere il progetto compilabile.

Innanzitutto per avviare *ndk build* é necessario impostare il percorso nel path di sistema. Quando si scarica il progetto è preferibile posizionarlo nel workspace di Eclipse per evitare complicazioni.

Durante la compilazione mi dava errori, ecco come ho fatto per ovviare al problema in caso dovesse presentarsi:

Il file mk Il file con estensione .mk é un file, di solito, generato automaticamente che indica al compilatore come eseguire la compilazione. //

Vediamo come modificarlo per non fargli lanciare errori:

La revision 6 di tesseract-android-tools ha una piccola modifica che lo rende incompatibile con il file readme al suo interno. Per ovviare al problema navighiamo fino ad <percorsoWorkspaceEclipse>/android-tesseract-tools/jni/Android.mk e commentiamo (o se preferite cancelliamo) tutte le righe prima di

```
ifeq "$(TESSERACT_PATH)" ""
  $(error You must set the TESSERACT_PATH variable to the
      Tesseract source \
          directory. See README and jni/Android.mk for details)
endif
```

salviamo e chiudiamo.

Poi dobbiamo correggere i file *Android.mk* nelle due sottocartelle

1. **com_googlecode_leptonica_android**
2. **com_googlecode_tesseract_android**

e per ogni file Android.mk nelle 2 sottocartelle, modificare la riga REAL_LOCAL_PATH := $(call my-dir) in REAL_LOCAL_PATH := <percorsoWorkspaceEclipse>/tesseract-android-tools/$(call my-dir)

Una volta compilato tesseract con NDK, se non riusciamo a compilare il file con ant (come scritto nella guida), lo si può fare con Eclipse nel seguente modo: si apre Eclipse e si importando i file come indicato nella guida; sempre da Eclipse si clicca col tasto destro sulla cartella tesseract-android-tools ⇒ Properties, sulla barra di sinistra si clicca Android e si toglie la spunta da isLibrary e si salva. Si va su Run Configuration, click col destro su Android application e si preme new. A questo punto impostiamo come project tesseract-android-tools e impostiamo do nothing. Andiamo su start e controlliamo che nella consolle scriva tesseract-android-tools.apk installed on device. Ora rimettiamo la spunta a isLibrary e giunti a questo punto siamo riusciti a generare l'apk di tesseract-android-tools. Nel nostro progetto originale dobbiamo includere le librerie di tesseract: per fare ciò basta cliccare col destro sul nome della nostra applicazione a sinistra, andare in *Properties*, nella sezione Build Path e sulla scheda Project cliccare su add e mettere la spunta su tesseract-android-tools. Clicchiamo infine su ok e salva. A questo punto siamo quasi pronti per far partire il riconoscimento ma prima dobbiamo inviare il file eng.traineddata (o ita.traineddata) nella cartella /mnt/sdcard/tesseract/tessdata del telefono virtuale. Per fare questo abbiamo bisogno prima di tutto di aggiungere al path di linux o windows la cartella <vostro percorso>/android-sdks/platform-tools/. A questo punto potete lanciare i comandi:

adb shell mkdir /mnt/sdcard/tesseract

adb shell mkdir /mnt/sdcard/tesseract/tessdata

adb push eng.traineddata(ita.traineddata) /mnt/sdcard/tesseract/tessdata

I primi due comandi generano le cartelle tesseract e la sottocartella tessdata, il secondo invia il file eng.traineddata (ita.traindeddata) nell'ultima cartella creata.

Inseriamo nell'applicazione questo codice:

```
File myDir = getExternalFilesDir(Environment.MEDIA_MOUNTED);

TessBaseAPI baseApi = new TessBaseAPI();
```

```
baseApi.init(myDir.toString(), "eng"); // oppure ita al posto di
    eng
baseApi.setImage(myImage); //puo essere un percorso o un Bitmap
    codificato con ARGB_8888 altrimenti dar\'a un errore

String recognizedText = baseApi.getUTF8Text(); // Log or
    otherwise display this
Log.i("Testo Riconosciuto",recognizedText);
baseApi.end();
```

Eseguendo questo codice e passandogli un'immagine nel LogCat[4] si vedrà il testo riconosciuto. Adesso possiamo inviare un file di immagine jpg o png e vedere il suo riconoscimento sulla consolle LogCat

3.3 Scheletro di un applicazione Java

3.3.1 Gestione di base

Nell' Hello World che abbiamo ottenuto in 2.2 notiamo subito che al posto del Main c'è una classe che estende la classe Activity e ne viene sovrascritto il metodo onCreate(Bundle savedInstanceState). Questa é la prima cosa che viene eseguita quando l'applicazione viene lanciata. Ora esaminiamo il codice già inserito.

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

questa riga chiama il metodo onCreate della classe Activity. Una riga ben piú interessante é

```
setContentView(R.layout.main);
```

[4]Il termine Log indica la registrazione cronologica delle operazioni man mano che vengono eseguite [Log - Wikipedia]. Il termine Cat indica un comando dei sistemi operativi Unix e Unix-like che legge i file che gli sono specificati come parametri (o lo standard input) e produce sullo standard output la concatenazione del loro contenuto [Cat - Wikipedia].

I due termini uniti formano Logcat, che indica quindi il tracciamento cronologico delle operazioni con la relativa visualizzazione sullo standard output del sistema. Lo si visualizza in una scheda di Eclipse accanto alla consolle. Vengono visualizzati anche gli errori nel caso in cui ce ne siano e la relativa posizione nel codice.

Questo comando mostra a video la schermata main estratta dalla classe R.layout.main che viene generata automaticamente nella fase di build. Questa schermata grazie ai tool di Eclipse viene visualizzata in maniera grafica e si trova nella cartella res/layout/main.xml.

I Layout Nella cartella Layout vengono definiti tutti i layout delle schermate che si vogliono inserire nel programma. Vediamo nello specifico la struttura del file main.xml generato automaticamente in fase di creazione del progetto:

```
<?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" />

</LinearLayout>
```

Ogni comando di definizione é preceduto dalla stringa “*android:*”. Lasciando perdere il resto che verrà approfondito piú avanti notiamo cosa mostra a video la scritta Hello World <nome applicazione> che é la TextView. I primi due comandi definiscono la dimensione del campo di testo, la terza definisce cosa c'è scritto nel campo. Notiamo che c'è scritto @string/hello che fa riferimento alla stringa hello la quale é definita nel file string che si trova in res/strings

Le Stringhe Tutti i file che si trovano nella cartella res/strings contengono la definizione di tutte le stringhe che vengono usate nel programma. Ovviamente non siamo costretti a definire ogni stringa all'interno di questi file, possiamo inserirle direttamente nei file che ci interessano. Nel file precedente avremmo potuto scrivere:

```
android:text="CIAO MONDO"
```

e il compilatore ce lo avrebbe accettato senza problemi. É buona norma però inserire qui ogni stringa per due motivi:

1. É più facile modificarle per eventuali correzioni

2. Si possono fare dei supporti multilingua in modo da non dover creare diverse versioni del programma solo per la lingua

3.3.2 La schermata principale

Iniziamo subito con la creazione della schermata principale, che consiste nell'inserire i bottoni con funzioni che poi dovremo implementare. Con l'editor grafico risulta facile crearla. Ecco Il listato

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

     <TextView
          android:id="@+id/textView1"
          android:layout_width="fill_parent"
          android:layout_height="wrap_content"
          android:layout_gravity="right"
          android:text="@string/useCamera"
          android:textAppearance="?android:attr/textAppearanceLarge"
              />

     <Button
          android:id="@+id/videocamera"
          android:layout_width="match_parent"
          android:layout_height="wrap_content"
          android:text="@string/useCamera" />

     <Button
          android:id="@+id/ocr"
          android:layout_width="match_parent"
          android:layout_height="wrap_content"
          android:text="@string/ocr" />

     <Button
          android:id="@+id/bollettino"
          android:layout_width="match_parent"
          android:layout_height="wrap_content"
          android:text="@string/bollettino" />

     <Button
```

```
        android:id="@+id/visualizza"
39      android:layout_width="match_parent"
        android:layout_height="wrap_content"
41      android:text="@string/show" />

43  <Button
        android:id="@+id/scanButton"
45      android:layout_width="match_parent"
        android:layout_height="wrap_content"
47      android:onClick="onScan"
        android:text="@string/barre" />
49
    <TextView
51      android:id="@+id/scanResult"
        android:layout_width="wrap_content"
53      android:layout_height="wrap_content"
        android:text="@string/noBarCode"
55      android:textAppearance="?android:attr/textAppearanceMedium"
            />

57  <Button
        android:id="@+id/download"
59      android:layout_width="match_parent"
        android:layout_height="wrap_content"
61      android:onClick="onDownload"
        android:text="@string/download" />
63
    <Button
65      android:id="@+id/testCamera"
        android:layout_width="match_parent"
67      android:layout_height="wrap_content"
        android:onClick="onCamera"
69      android:text="@string/testCamera" />

71  <Button
        android:id="@+id/riconoscitarga"
73      android:layout_width="match_parent"
        android:layout_height="wrap_content"
75      android:onClick="onTarga"
        android:text="@string/riconoscitarga" />
77
    <Button
79      android:id="@+id/button1"
        android:layout_width="match_parent"
81      android:layout_height="wrap_content"
        android:onClick="onTestHtml"
83      android:text="@string/richiestaHtml" />

85   <Button
```

```
            android:id="@+id/test"
87          android:layout_width="match_parent"
            android:layout_height="wrap_content"
89          android:onClick="onTestActivity"
            android:text="@string/test" />
91
    </LinearLayout>
93  </ScrollView>
```

../software/res/layout/main.xml

Come possiamo notare ci sono due tipi di elementi, le *TextView* e i *Button*. Le TextView sono delle comuni aree testuali mentre i Button sono appunto i bottoni. Come potete notare ogni elemento é identificato dall' ID che viene definito così *android:id=@+id/textView1*[5]. Gli ID ci risulteranno utili piú avanti per assegnare azioni ai vari bottoni.

Dimensionamento I Campi *android:layout_ width* e *android:layout_ height* definiscono le dimensioni dei pulsanti. Il valore *match_ parent* attribuito alla lunghezza indica al bottone di allargarsi finché non raggiunge la lunghezza del contenitore padre (in questo caso lo schermo stesso), il valore *wrap_ content* invece indica al componente che può allungarsi quanto vuole (in altezza se é associato a layout_height) per far spazio al testo che ha al suo interno.

Assegnare Stringhe ai componenti Grazie all'editor grafico é molto semplice assegnare una stringa, basta cliccare col destro sul componente e cliccare su *edit text*, a questo punto vediamo tutte le stringhe già inserite -quella che vogliamo noi ovviamente non é presente- , quindi clicchiamo su *New String*. Nel campo *String:* scriviamo il valore della stringa che vogliamo aggiungere (ad esempio *Utilizza la Telecamera*), nel campo New R.string scriviamo un id che in qualche modo evochi la scritta, ad esempio *useCamera*. Tutte le stringhe vanno nella cartella *res/values*. Eclipse automaticamente assegna il nome *strings.xml* anche se, come vedremo anche in altre occasioni, il nome non é discriminante. Ecco il file con tutte le stringhe.

```
1   <?xml version="1.0" encoding="utf-8"?>
    <resources>
3
        <string name="hello">Hello World, SoftwareActivity!</string>
5       <string name="app_name">Software</string>
        <string name="wait">Per favore attendi</string>
7       <string name="useCamera">Usa la videocamera</string>
```

[5]Si possono editare anche via grafica i vari id semplicemente facendo click destro sul componente e cliccando su *Edit ID*

```
    <string name="back">Ritorna</string>
    <string name="expectedText">QuÃň dovrebbe comparire il testo
        riconosciuto</string>
    <string name="ocr">Riconoscimento Caratteri</string>
    <string name="bollettino">Riconosci Bollettino</string>
    <string name="show">Visualizza Immagine</string>
    <string name="cc">Conto Corrente</string>
    <string name="importo">Importo</string>
    <string name="intestatario">Intestatario</string>
    <string name="causale">Causale</string>
    <string name="send">Invia Sms</string>
    <string name="barre">Scan Codice a Barre</string>
    <string name="download">Scarica Cliccando Qui</string>
    <string name="noBarCode">Non hai installato nessuna
        applicazione per effettuare lo scan!!!</string>
    <string name="add">Aggiungi</string>
    <string name="verify">Verifica</string>
    <string name="showCode">Visualizza Codici</string>
    <string name="archivio">Archivio</string>
    <string name="delete">Cancella</string>
    <string name="testCamera">Testa la Telecamera</string>
    <string name="click">Scatta</string>
    <string name="riconoscitarga">Riconosci La Targa</string>
    <string name="richiestaHtml">Test Html</string>
    <string name="test">Test</string>

</resources>
```

../software/res/values/strings.xml

3.3.3 Assegnare le azioni ai bottoni

I vari bottoni in *main* così come sono non fanno nulla perché non é stato assegnato loro alcuna azione. Ora vediamo come assegnarne loro una. Nel metodo onCreate andiamo a inserire questo codice per richiamare l'oggetto bottone dal codice Java

```
OnClickListener click= new OnClickListener(){
  public void onClick(View arg0) {
        //Codice da eseguire al click
      }

    };
Button but=(Button)findViewById(R.id.videocamera);
but.setOnClickListener(click);
```

Spiego brevemente il codice. **OnClickListener** é un interfaccia java con un unico metodo da implementare: ❝onClick()❞. Questo metodo viene invocato appunto quando il tasto viene premuto. Ora che abbiamo l'oggetto *click* con un implementazione di OnClickListener dobbiamo dire in quale bottone impostare questa azione. Per farlo prima dobbiamo prelevare dalla grafica l'oggetto. Per fare questo dobbiamo usare il metodo *findViewById(R.id.<idComponente>)*. Questo ci restituisce un oggetto View che é superclasse rispetto a Button. Quindi con un cast esplicito lo trasformiamo in un Button. A questo punto con l'oggetto gli impostiamo l'azione che deve fare al click.

3.3.4 Activity e Cambio di Schermata

Ora che sappiamo assegnare azioni ai bottoni vogliamo creare altre schermate in modo tale che quando clicchiamo su un bottone si acceda ad altre. Vediamo come creare la piú semplice, la visualizzazione di un'immagine. Il metodo migliore per visualizzare schermate é senza dubbio quello di farle visualizzare a una nuova Activity

Cos'è un Activity? Un Activity rappresenta una possibile interazione dell'utente con l'applicazione e può essere associata al concetto di schermata. Essa potrà contenere componenti di sola visualizzazione, insieme ad altri che invece permettono l'interazione con l'utente.[6]

L'immagine che scegliamo di visualizzare é quella che poi useremo per farci l'Ocr, riconoscere il bollettino oppure la bolletta. Come prima cosa dobbiamo creare il Layout per questa nuova schermata, ecco il codice xml relativo:

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView
    xmlns:android="http://schemas.android.com/apk/res/android"
     android:layout_width="fill_parent"
     android:layout_height="fill_parent" >
<LinearLayout
     android:layout_width="fill_parent"
     android:layout_height="wrap_content"
     android:baselineAligned="true"
     android:orientation="vertical" >
<LinearLayout android:layout_width="fill_parent"
     android:layout_height="300dp"
     android:background="@color/white"
```

[6]Tratto dal libro: Android Guida per lo sviluppatore di Massimo Carli disponibile nella bibliografia [Android, Guida per lo sviluppatore]

```
    android:gravity="center">
    <ImageView
        android:id="@+id/image"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:contentDescription="@string/expectedText"
        android:src="@drawable/loading" />
</LinearLayout>
    <Button
        android:id="@+id/back"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/back" />

    <TextView
        android:id="@+id/testoRiconosciuto"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:scrollbars = "vertical"
        android:layout_weight="0.51"
        android:text="@string/expectedText"
        android:textAppearance="?android:attr/textAppearanceMedium"
            />



</LinearLayout>
</ScrollView>
```

../software/res/layout/visualizza.xml

Ci sono tre componenti: l'area dell'immagine (**ImageView**), l'area di testo (**TextView**) e il Bottone (**Button**). L'unico non visto fin ora é l'ImageView la cui unica particolarità é il campo *android:src* che corrisponde alla sorgente dell'immagine. Sorgente che inseriamo ma che, in realtà, non ci interessa in quanto l'indirizzo verrà cambiato sempre prima della visualizzazione della schermata. Ora che abbiamo la schermata procediamo con la creazione dell'Activity relativa alla visualizzazione della nuova schermata. Una nuova Activity ha la stessa struttura della precedente, quindi ricopiamo il codice dell'Hello World con l'unica differenza che al posto di visualizzare il layout *Main*, visualizziamo il layout *visualizza*. Vediamone il codice:

```
package android.telerete;

import android.app.Activity;
import android.graphics.Bitmap;
import android.os.Bundle;
import android.text.method.ScrollingMovementMethod;
```

```
7  import android.view.View;
   import android.view.View.OnClickListener;
9  import android.widget.Button;
   import android.widget.ImageView;
11 import android.widget.TextView;

13 public class visualizza extends Activity {
     public void onCreate(Bundle savedInstanceState) {
15     super.onCreate(savedInstanceState);
       Bitmap immagine=SoftwareActivity.img;
17     if(immagine==null){
         finish();
19     }
       setContentView(R.layout.visualizza);
21     ImageView img=(ImageView)findViewById(R.id.image);

23     img.setImageBitmap(immagine);
       //img.setImageURI(fileUri);
25     String recognizedText=SoftwareActivity.testoRiconosciuto;
       if(recognizedText!=null){
27       TextView t= (TextView)findViewById(R.id.testoRiconosciuto);
         t.setMovementMethod(new ScrollingMovementMethod());
29       t.setText(recognizedText);
       }
31     Button but= (Button)findViewById(R.id.back);
       but.setOnClickListener(new OnClickListener() {
33       public void onClick(View v) {
           finish();
35
         }
37     });

39   }
   }
```

../software/src/android/telerete/visualizza.java

Nel codice si nota come é avvenuto il passaggio dell'immagine dall'Activity principale all'Activity visualizza. In sostanza c'è una variabile statica nell'Activity principale che si chiama *immagine* di tipo Bitmap[7] e a cui bisogna necessariamente assegnare un valore prima di invocare l'Activity, altrimenti questa, appena creata, termina immediatamente (invocando il metodo *finish*). Lo stesso metodo viene usato per passare il testo da visualizzare sotto l'immagine[8] Notiamo inoltre che l'azione che fa il tasto Back non é esplicita-

[7] Tipo di oggetto che contiene la mappa dei pixel di un immagine

[8] Il passaggio dell'immagine e del testo é avvenuto tramite riferimento. Questo tipo di passaggio é possibile soltanto quando si vuole avviare un altra Activity interna all'applica-

mente di tornare alla schermata principale, ma semplicemente di terminare l'Activity. Questo perché all'attivazione di una nuova Activity quella vecchia viene messa in pausa[9] finché la nuova Activity non termina, lasciando il *foreground* a quella che l'ha generata. Ora non ci resta che far avviare l'Activity visualizza all'Activity principale. Per farlo eseguiamo il metodo schermataVisualizza

```
public void schermataVisualizza(Bitmap immagine,String
    recognizedText){
     SoftwareActivity.immagine=immagine;
     SoftwareActivity.testoRiconosciuto=recognizedText;
     Intent intent= new Intent(getApplicationContext(),
         visualizza.class);
     startActivity(intent);
  }
```

Come già detto in precedenza impostiamo le variabili statiche *immagine* e *testoRiconosciuto*, poi facciamo partire l'Activity selezionata. Per avviarla ci serviamo di un **Intent**.

Cos'è un Intent? Nel dizionario di Android, un intent é "la descrizione di un'operazione che dev'essere eseguita". (tratto da [Android, Guida per lo sviluppatore]) Più semplicemente, gli Intent sono dei messaggi che il sistema manda a un'applicazione quando si aspetta che questa faccia qualcosa. Ci sono due tipi di Intent:

- Gli Intent Espliciti
- Gli Intent non Espliciti

Gli **Intent Espliciti**, quelli usati per avviare l'Activity *visualizza*, vengono usati nel caso in cui si intenda far comunicare attività della stessa applicazione e già note in fase di sviluppo. Gli **Intent non Espliciti**, invece, si riferiscono ad Activity non necessariamente presenti nell'applicazione, ma all'interno del telefono, come ad esempio la rubrica e la fotocamera (che tra poco vedremo).

Se proviamo ora a far partire l'applicazione notiamo che, appena clicchiamo sul tasto *Visualizza Immagine*, si blocca. Dando uno sguardo al LogCat vediamo la comparsa di questo errore.

---

zione, vedremo successivamente che nel caso volessimo passare dei parametri a un activity esterna é necessario utilizzare dei metodi piú rigorosi.

[9] In alcuni casi, come ad esempio il bisogno urgente di memoria ram da parte del telefono, viene anche rimossa e successivamente ricaricata.

```
02-10 13:41:37.897: E/AndroidRuntime(308):
    android.content.ActivityNotFoundException: Unable to find
    explicit activity class
    {android.telerete/android.telerete.visualizza}; have you
    declared this activity in your AndroidManifest.xml?
```

L'errore ci sta dicendo che non é stata dichiarata alcuna attività esplicita col nome *visualizza*. Quindi per far partire l'Activity dobbiamo editare il file AndroidManifest.xml e aggiungere questa riga prima della chiusura del tag *</application>*

```
1 <activity android:name="visualizza"></activity>
```

Abbiamo così dichiarato l'Activity, quindi ora é autorizzata ad essere eseguita.

Sulla macchina virtuale vediamo che ci mostra correttamente la foto. Per tornare alla schermata principale possiamo usare il tasto *Ritorna* sullo schermo o semplicemente il tasto ritorna del cellulare.

Invio dati ad un'Activity Sia che si stia chiamando una nuova Activity in modo esplicito o la si stia chiamando in modo implicito, si possono passare dei parametri. Il metodo che ci consente di farlo è il **putExtra(String chiave,Object Valore)**. Questo è un metodo sovraccarico [10] che accetta come ingresso tutti i tipi di dato standard di java (byte, int, boolean, String ecc..) e anche array e liste concatenate contenenti dati di questo tipo. Accetta inoltre oggetti di tipo **Parcelable** e **Serializable** (e liste concatenate o ArrayList contenenti questi oggetti). Se si vuole inviare oggetti non standard in un array bisogna necessariamente che siano di tipo Parcelable, i dati di tipo Serializable in array non potranno essere inviati.

Vediamo ora come usare un'**Activity non esplicita**

[10] si dice sovraccarica(overloaded in inglese) una famiglia di funzioni/subroutine aventi lo stesso nome, ma con la possibilità di accettare un diverso set di argomenti (signature), ed eventualmente restituire un diverso valore di ritorno

3.4 Prime Implementazioni

3.4.1 Acquisire un'immagine dalla Fotocamera

Per fare un'azione del genere dobbiamo creare un Intent diverso da quello generato prima per tre ragioni.

- Il software per scattare la fotografia sappiamo che non é integrato nel nostro, quindi per fare ciò dobbiamo chiedere a un'Activity di un altra applicazione se può scattare la foto.
- Abbiamo bisogno che questa Activity ci comunichi se l'operazione é andata a buon fine e dove ha salvato l'immagine.
- Dobbiamo dire all'Activity dove salvare l'immagine scattata.

Ecco il listato del codice che fa partire la fotocamera.

```
1  public void onClick(View v) {
   Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
3  File dir = Environment.getExternalStorageDirectory();
   yourFile = new File(dir, "DCIM");
5  yourFile = new File(yourFile, "Camera");
   yourFile = new File(yourFile, "bau.jpg");
7  fileUri = Uri.fromFile(yourFile);// create a file to save the
       image
   Log.i("File Destination Foder", fileUri.getEncodedPath());
9  intent.putExtra(MediaStore.EXTRA_OUTPUT, fileUri); // set the
       image file name
   // start the image capture Intent
11 startActivityForResult(intent,
       CAPTURE_IMAGE_ACTIVITY_REQUEST_CODE);
}
```

Come possiamo notare l'Intent creato non é esplicito in quanto non gli é stato detto esplicitamente che classe eseguire. Le righe dalla **3** alla **7** servono a creare un oggetto File che punta a un oggetto di nome *bau.jpg* nella cartella /DCIM/Camera/ all'interno della sdcard; la riga **8** serve soltanto a mostrare nel LogCat l'indirizzo completo del file bau.jpg; la riga **9** serve a dire all'Intent che viene prodotto dalla fotocamera che deve essere salvato come bau.jpg nella cartella indicata prima; la riga **10** serve a far partire l'Intent. Notiamo peró che questa volta il metodo scrive *startActivityForResult*: questo sta a indicare che si fa partire l'Activity per avere un risultato. Una volta che termina una qualsiasi Activity, lanciata con questo metodo,

viene eseguito il metodo *protected void onActivityResult(int requestCode, int resultCode, Intent data)*. Vediamone i parametri:

- **requestCode**: é un intero contenente il codice identificativo di quella particolare Activity che noi abbiamo precedentemente assegnato quando abbiamo invocato il metodo *startActivityForResult*, in questo specifico caso il codice che restituirá la fotocamera é CAPTURE_IMAGE_ACTIVITY_REQUEST_CODE[11]

- **resultCode**: Il result code é un intero che ci informa se l'Activity é terminata correttamente, é stata annullata o se si sono verificati errori. Può assumere i valori RESULT_OK, RESULT_CANCELED o altri valori che corrispondono ai vari errori.

- **intentData**: Questo é un oggetto di tipo Intent che contiene, tra le altre cose, l'insieme dei dati che l'Activity lanciata ha prodotto.

Vediamo ora il metodo onActivityResult[12] della nostra applicazione

```
protected void onActivityResult(int requestCode, int resultCode,
    Intent data) {
     if (requestCode == CAPTURE_IMAGE_ACTIVITY_REQUEST_CODE) {//
        \'e la risposta relativa alla fotocamera?
       if (resultCode == RESULT_OK) {//l'acquisizione \'e andata
          a buon fine
         // Image captued and saved to fileUri specified in the
            Intent
         String au="destinazione non disponibile";
         if(data!=null)
           au=data.getData().getPath();
         else{
           Toast.makeText(this, "Errore durante
              l'acquisizione",Toast.LENGTH_LONG).show();
           return;
         }
         this.yourFile=new File(data.getData().getPath());
         Toast.makeText(this, "Image saved "+au,
             Toast.LENGTH_LONG).show();
         BitmapFactory.Options options = new
             BitmapFactory.Options();
         options.inSampleSize = 2;
```

[11] Questo codice viene definito come costante statica
private static final int CAPTURE_IMAGE_ACTIVITY_REQUEST_CODE = 100;

[12] Questo metodo è già presente nella superclasse Activity che estendiamo, e verrà dunque sovrascritto

```
16          Bitmap
                immagine=BitmapFactory.decodeFile(data.getData().getPath(),options);
            schermataVisualizza(immagine, null);
18
        } else if (resultCode == RESULT_CANCELED)
            {//l'acquisizione \'e stata cancellata
20          Toast.makeText(this, "L'acquisizione dell'immagine \'e
                stata cancellata",Toast.LENGTH_LONG).show();

22      } else {//E' avvenuto un errore
            Toast.makeText(this, "Errore durante
                l'acquisizione",Toast.LENGTH_LONG).show();
24
        }
26

    }
28  }
```

In questo frammento di codice, vediamo innanzitutto che controlla che il *requestCode* sia relativo a quello della chiamata alla videocamera. In secondo luogo controlla di ricevere il *RESULT_OK*, infine se il parametro *data* é diverso da null. Se tutte queste condizioni sono soddisfatte, esegue le seguenti operazioni:

- Notifica all'utente con un breve messaggio dove l'immagine é stata salvata.
- Salva il percorso dell'immagine nella variabile di istanza yourFile.
- Legge l'immagine e la salva in un Oggetto di tipo Bitmap.
- Infine visualizza l'immagine all'utente richiamando il metodo costriuito in precedenza.

3.4.2 Effettuare il riconoscimento dei caratteri

Ora che sappiamo come acquisire un'immagine e sappiamo come visualizzarla, possiamo benissimo effettuare l'OCR e visualizzare Immagine e Riconoscimento a video. Il codice per effettuare l'OCR é questo

```
TessBaseAPI baseApi = new TessBaseAPI();
2           if(immagine.getConfig().compareTo(Config.ARGB_8888)!=0){
              immagine=correttoreGrafico.dammiImmagine(immagine,
                  new
                  Dimension(0,0,immagine.getWidth(),immagine.getHeight()));
```

```
            }
            baseApi.init(myDir.toString(), "ita"); // myDir +
                "/tessdata/eng.traineddata" must be present
            baseApi.setImage(immagine);

            String recognizedText = baseApi.getUTF8Text(); //
                Log or otherwise display this string...
            Log.i("Testo Riconosciuto",recognizedText);

            baseApi.end();
            schermataVisualizza(immagine, recognizedText);
```

Se assegnamo questo codice all'evento click notiamo che, al momento della pressione del bottone, il programma sembra bloccarsi. In realtà sta effettuando l'OCR che richiede un po' di tempo a seconda delle dimensioni e del numero di caratteri nella figura. Ora é abbastanza naturale pensare che un utente medio pensi appunto che il programma abbia smesso di funzionare e cercherà di terminarlo se non si dice almeno di aspettare. Per farlo intanto creiamo la schermata che invita l'utente ad attendere.

Schermata di Attesa

```
<?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:baselineAligned="false"
     android:orientation="vertical" >

     <ImageView
          android:id="@+id/immagineDaRiconoscere"
          android:layout_width="match_parent"
          android:layout_height="400dp"
          android:src="@android:drawable/alert_dark_frame" />

     <RelativeLayout
          android:id="@+id/relativeLayout1"
          android:layout_width="match_parent"
          android:layout_height="wrap_content" >

          <ProgressBar
               android:id="@+id/progressBar1"
               style="?android:attr/progressBarStyleLarge"
               android:layout_width="wrap_content"
               android:layout_height="wrap_content"
               android:layout_alignParentLeft="true"
```

```
            android:layout_centerVertical="true" />

        <TextView
            android:id="@+id/textView2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_centerVertical="true"
            android:layout_toRightOf="@+id/progressBar1"
            android:text="@string/wait"
            android:textAppearance="?android:attr/textAppearanceLarge"
            />

    </RelativeLayout>

</LinearLayout>
```

../software/res/layout/caricamento.xml

Vediamo l'inserimento di un nuovo elemento che si chiama **RelativeLayout** in cui sono inclusi gli oggetti *progressBarStyleLarge*[13] e un *TextView* a cui é associata una stringa con scritto *Per favore attendi*. Il RelativeLayout serve solo per mettere di fianco il progressBarStyle e il TextView. Il tool grafico di Eclipse aiuta molto nella sistemazione degli elementi e genera lui automaticamente il codice. Per completezza spiego brevemente i comandi che per la prima volta vediamo nel codice: **android:layout_alignParentLeft= true**: Allineamento a sinistra rispetto al contenitore[14]. **android:layout_centerVertical= true**: allineamento centrato rispetto al contenitore. **android:layout_toRightOf= @+id/progressBar1**: posizionato alla destra di *progressBar1*.

L'Activity Caricamento Il codice dell'activity caricamento é abbastanza semplice

```
package android.telerete;

import android.app.Activity;
import android.graphics.Bitmap;
import android.os.Bundle;
import android.widget.ImageView;

public class caricamento extends Activity{
  public void onCreate(Bundle savedInstanceState){
    super.onCreate(savedInstanceState);
    Bitmap immagine=SoftwareActivity.img;
```

[13]Un oggetto di forma quadrata che mostra l'animazione di una rotella che gira. Serve a far capire all'utente che il telefono sta lavorando

[14]Che in questo caso é appunto il *RelativeLayout*

```
12      setContentView(R.layout.caricamento);
        ImageView
            img=(ImageView)findViewById(R.id.immagineDaRiconoscere);
14      if(immagine==null){
          finish();
16      }
        img.setImageBitmap(immagine);
18    }
}
```

../software/src/android/telerete/caricamento.java

Come possiamo notare é semplicemente la visualizzazione dell'immagine passata come riferimento nella variabile statica img della classe SoftwareActivity (Che é quella principale). La cosa interessante di questa Activity é come viene lanciata. Vediamo il codice che la lancia

```
1 public void schermataCaricamento(Bitmap immagine){
    SoftwareActivity.img=immagine;
3   Intent i= new Intent(getApplicationContext(),
        caricamento.class);
    i.addFlags(Intent.FLAG_ACTIVITY_NO_HISTORY);
5   startActivity(i);
}
```

La riga interessante é la quarta, in cui si aggiunge il **flag**: *FLAG_ACTIVITY_NO_HISTORY*.

Spieghiamo innanzitutto cos'è un **flag**: si tratta di una variabile che possiamo impostare al lancio di un'Activity per modificarne il modo in cui viene eseguita. Il flag che abbiamo usato noi serve a **non** far entrare l'Activity nello stack delle attività.

A livello pratico cosa succede? Siamo nel caso in cui l'Activity *SoftwareActivity* sta mostrando un caricamento lanciando l'Activity *caricamento*. Dopo qualche istante *SoftwareActivity* ha terminato il riconoscimento e vuole mostrare i risultati quindi lancerà l'Activity *visualizza*. Se non avessimo usato il **flag** indicato in precedenza alla pressione del tasto ritorna (nella schermata visualizza) si tornerebbe a visualizzare il caricamento e non é decisamente ció che si vuole ottenere. Con il flag impostato, l'Activity *caricamento* viene sostituita dalla nuova(*visualizza*) lanciata dal padre(*SoftwareActivity*). Così facendo, alla pressione del tasto ritorna, si fa ritorno direttamente alla schermata principale.

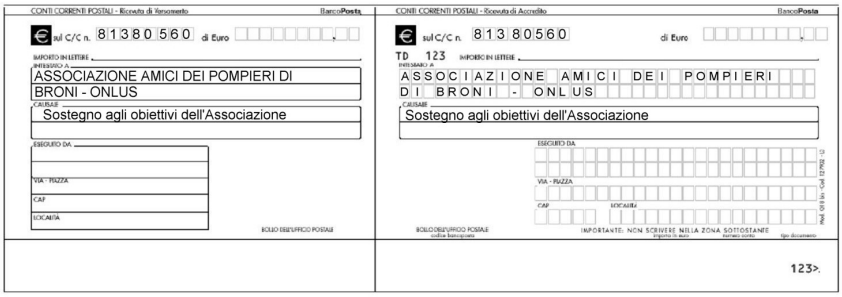

Figura 3.1: Bollettino

3.5 Riconoscimento del bollettino postale

3.5.1 Ideazione

Abbiamo ora tutti gli strumenti per preoccuparci di riconoscere il bollettino. La prima soluzione che mi é venuta in mente per risolvere il problema é di fare un riconoscimento OCR generalizzato a tutta l'immagine e dalla stringa ottenuta estrapolare i dati. Questo metodo peró é molto inefficiente oltre che impreciso. Provando l'OCR su una quantità di caratteri anche limitata a mezza pagina di un libro per bambini, il cellulare può impiegare anche un minuto per il riconoscimento completo.

Per queste ragioni ho ritenuto la mia prima idea da scartare, optando invece per trovare le parti interessanti del bollettino ed effettuare un riconoscimento mirato su queste.
Ovviamente anche una scansione dell'immagine richiede un certo tempo, ma significativamente inferiore a quello di un OCR generalizzato. In maniera tale da escludere alcune zone in cui effettuare la scansione per velocizzare il processo. L'idea é di usare dei punti facilmente individuabili e da quelli prelevare le informazioni scritte sul bollettino. I punti in questione, per via del loro colore e della loro posizione, sono sicuramente i simboli neri su cui é disegnato l'Euro.

É facile individuarli perché basta cercare nella foto due quadrati neri con le dimensioni simili a quelle indicate. Dopo aver individuato le posizioni e le dimensioni più o meno precise dei due €possiamo estrarre dal bollettino le parti interessanti, ovvero il C/C, gli Euro da pagare, l'intestatario e la causale. Per quanto riguarda le informazioni di chi esegue il bollettino non ci preoccupiamo di prelevarle per due ragioni:

- Perché nella maggior parte dei casi saranno i dati dell'utente proprietario del telefono, quindi daremo la possibilità di inserirli a mano al

primo utilizzo.

- Se le informazioni sono presenti saranno scritte inevitabilmente a mano e Tesseract funziona male con il riconoscimento di questa scrittura.

Vediamo ora di realizzare a livello di codice le nostre idee.

3.5.2 Individuazione posizione dei simboli con l'euro

Il problema si riduce a trovare due quadrati nero di dimensioni compatibili all'interno della foto. Come prima cosa cerchiamo di capire quali sono le dimensioni che stiamo cercando. Guardando la figura 3.1 possiamo intuire che oscillano tra il 2% e il 4% della figura. Quindi iniziamo a definire le costanti

```
private final static int expectedDimEMin=2;
private final static int expectedDimEMax=4;
```

Prima di iniziare la ricerca ci serve un metodo per capire se il pixel preso in esame é da considerarsi un pixel nero(o almeno grigio) oppure di un colore diverso. Creiamo allora il metodo *isGrigio(int argb)* per discriminare i pixel

```
private boolean isGrigio(int argb){
  int r = (argb >> 16) & 0xFF;
  int g = (argb >> 8) & 0xFF;
  int b = (argb >> 0) & 0xFF;
  long mediaGrigia=(r+g+b)/3;
  int colore=r;
  if(mediaGrigia>colore-range&&mediaGrigia<colore+range){
    colore=g;
    if(mediaGrigia>colore-range&&mediaGrigia<colore+range){
      colore=b;
      if(mediaGrigia>colore-range&&mediaGrigia<colore+range){
        int value=ingresso.getWidth()*2;
        mediaColore=(( mediaColore*value) +mediaGrigia)/
            (value+1);
        if(mediaGrigia<mediaColore)
          mediaBassa=Math.min(((mediaBassa*value)
              +mediaGrigia)/ (value+1), 110);
        else
          mediaAlta=Math.max(((mediaAlta*value)+ mediaGrigia)/
              (value+1), 150);
        return mediaGrigia<(mediaBassa+mediaAlta)/2;
      }
    }
  }
```

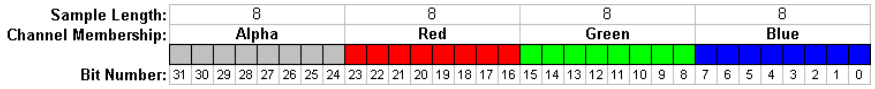

Figura 3.2: Raffigurazione ARGB

```
22
        return false;
24    }
```

Il primo pezzo di codice trasforma l'intero ARGB in un colore RGB.

Cos'è l'ARGB L'**ARGB** é l'acronimo di Alpha Red Green Blue che sono le quattro componenti necessarie a definire un colore. **Alpha** indica la trasparenza del colore (0=trasparente, 255 opaco), **Red** é l'intensità del rosso, **Green** quella del verde e **Blue** quella del blu. Ogni componente può assumere un valore compreso tra 0 e 255. Il valore in ARGB si ottiene così
ARGB=(((((Alpha*256)+Red)*256)+Green)*256)+Blue
Oppure apponendo i valori in esadecimale uno dietro l'altro
Ad esempio se volessimo rappresentare il rosso opaco si utilizzano i seguenti valori A=255 R=255 G=0 B=0
Che in esadecimale diventano A=FF R=FF G=00 B=00
Quindi ARGB= FFFF0000;

Per convertire un numero da ARGB nelle sue 3 componenti (Perché il valore di Alpha non ci interessa) facciamo uno shift dei bit verso destra, ad esempio per il rosso argb » 16. Questo corrisponde a fare una divisione per 2^{16} ovvero per 256*256. Così facendo otteniamo il valore che ci interessa nei primi 8 bit. Estraiamo il valore forzando gli altri bit al di fuori di quegli otto ad essere zero con un operazione di AND 0xFF.

Tornando ad esaminare il codice vediamo che viene calcolata la *mediaGrigia* che altro non é che la media dei tre colori, ovvero la rappresentazione in bianco e nero del colore proposto. I 3 if controllano semplicemente che il colore sia effettivamente una gradazione di grigio e non un rosso ad esempio[15]. All'interno dei tre if c'è un sistema che cerca di capire se un pixel é effettivamente da considerarsi nero oppure no cercando di adattarsi a tutte le condizioni di illuminazione in cui può essere scattata la foto.

[15]La variabile range indica con che tolleranza accettare le gradazioni di grigio; il valore che dà i migliori risultati é 10 (ottenuto sperimentalmente)

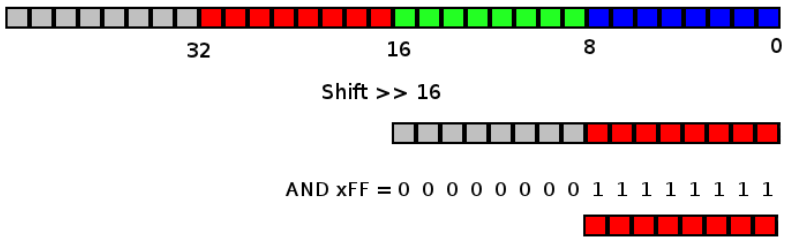

Figura 3.3: Estrazione del valore rosso da ARGB

Vediamo che é presente la variabile *mediaColore* che é la media dei valori degli ultimi[16] pixel. Le altre medie vengono calcolate col medesimo meccanismo badando che la *mediaAlta* é la media dei valori superiori alla *mediaColore*, mentre la *mediaBassa* é la media di quelli inferiori. Da notare inoltre che le medie hanno un massimo e un minimo imposto per evitare che assumano valori troppo sballati in determinate condizioni, ad esempio che la foto venga fatta su un tavolo nero e che sia eccessivamente scura.

Si decide infine se il pixel in esame é grigio eseguendo questo test.

```
return mediaGrigia<(mediaBassa+mediaAlta)/2;
```

Individuazione effettiva delle posizioni Ora che abbiamo il metodo per distinguere i pixel da considerare neri passiamo al metodo che restituisce le coordinate delle posizioni dei simboli dei 2 euro neri. Come prima cosa notiamo che é inutile fare la scansione dell'intera immagine per trovarli. Nella maggior parte dei casi (se non in tutti) gli euro si troveranno nella metà superiore della foto, quindi la scansione si limita a metà documento. In secondo luogo, dato che usiamo lo stesso metodo per individuare i due bollettini, dovremmo prevedere che ammetta un intero che definisca da quale posizione iniziare la scansione. La scansione, quindi, inizierà dal pixel più in alto della colonna indicata dal parametro in ingresso e procederà verticalmente verso il basso. Inizierà così a cercare delle strisce di pixel di lunghezza compatibile a quella aspettata. Non appena le trova, conta i pixel neri del quadrato che si origina utilizzando come lato sinistro la striscia appena trovata. Se questi sono almeno il doppio dei pixel non neri trovati nel medesimo quadrato, allora, con ogni probabilità, sarà il quadrato del bollettino che stiamo cercando. Restituiamo quindi le dimensioni e posizioni del simbolo tramite un oggetto

[16]Guardando il codice si nota che si dà peso soltanto agli ultimi ingresso.getWidth()*2 valori, quindi in un immagine di dimensioni 300*200 pixel saranno gli ultimi 600 valori

Dimension che contiene quattro interi, la posizione x e y e la dimensione x e y. Ecco il listato del metodo

```
public Dimension posizioneEuroBollettino(int da){
  int dimMin=(int)((ingresso.getWidth()*expectedDimEMin)/100);
  int dimMax=(int)((ingresso.getWidth()*expectedDimEMax)/100);
  Log.d("Min e MaxNeri",dimMin+" "+dimMax);
  int maxDist=dimMin/5;
  Log.i("Max Dist",""+maxDist);
  for(int x=da;x<ingresso.getWidth()-1;x++){
    int posizioneY=-1;
    int distanzaDaUltimo=0;
    int numeriNeri=0;
    for(int y=1;y<ingresso.getHeight()/2;y++){//cerca solo nella
        metà in alto
    int argb=ingresso.getPixel(x, y);
    if(isGrigio(argb)){
      if(numeriNeri==0)
      posizioneY=y;
      numeriNeri++;
      distanzaDaUltimo=0;
    }
    else{
      distanzaDaUltimo++;
      if(distanzaDaUltimo>maxDist){//Non è interrotto da un solo
          pixel bianco
         if(numeriNeri<dimMax&&numeriNeri>dimMin){//La
            dimensione è compatibile
          Log.i("NumeriNeriY accettabili trovati","Alla
              posizione "+x+" "+y);
          int numeriBianchi=0;
          int questiNumeriNeri=0;
          for(int
              questoY=posizioneY;questoY<y;questoY++){//conta i
              punti neri a destra della linea trovata
            for(int i=x;i<x+((numeriNeri*2)/3);i++){
            int c=ingresso.getPixel(i,questoY);
              if(!isGrigio(c)){
                numeriBianchi++;
            }
            else{
                questiNumeriNeri++;
            }
            }
          }
          if(questiNumeriNeri>numeriBianchi*2){
            return new
                Dimension(x,posizioneY,numeriNeri,numeriNeri);
          }
```

```
      }
      numeriNeri=0;
      }
     }
    }
   }
  return null;
}
```

3.5.3 Estrapolazione delle Informazioni

Servendosi del metodo precedente, si individuano le posizioni degli euro. Tramite proporzioni si generano i Dimension relativi alle varie posizioni, si ritagliano le figure e si fa l'OCR su quelle estratte. Infine si restituisce il bollettino.

```
private bollettino bollettino(Bitmap thumb){
 correttoreGrafico cor= new correttoreGrafico(thumb);
 Dimension d= cor.posizioneEuroBollettino(0);
 Bitmap immagine=correttoreGrafico.incornicia(thumb,d);
 schermataCaricamento(immagine);
 Dimension d2;
 cor= new correttoreGrafico(thumb);
 int dax=thumb.getWidth()/3;
 do{
  d2= cor.posizioneEuroBollettino(dax);
  dax=d2.posx+5;
  immagine=correttoreGrafico.incornicia(thumb,d2);
      schermataCaricamento(immagine);
 }
 while(d2.posy>d.posy+(thumb.getHeight()*0.02)||d2.posy<d.posy-(thumb.getHeight
 correttoreGrafico aa= new
     correttoreGrafico(correttoreGrafico.dammiImmagine(thumb,
     d2));
 immagine=aa.evidenzia();
 schermataCaricamento(immagine);
 try {
   Thread.sleep(2000);
  } catch (InterruptedException e) {

  }
 d.dimy=d.dimx=(d2.posx-d.posx)/19;
 Log.i("Dimensione x",""+d.dimx);
 int posx=d.posx+d.dimx;
 int dimx=d2.posx-(2*d2.dimx)-posx;
```

```
Dimension striscia CC=new
    Dimension(posx,(d.posy+d2.posy)/2,dimx-((dimx+d.dimx)/2),d2.dimy);
posx=posx+(dimx-((dimx+d.dimx)/2));
Dimension strisciaImporto=new
    Dimension(posx+3*d.dimx,(d.posy+d2.posy)/2,((dimx-d.dimx*3)/2),d2.dimy);
Bitmap contoCorrente=correttoreGrafico.dammiImmagine(thumb,
    striscia CC);
Bitmap importo=correttoreGrafico.dammiImmagine(thumb,
    strisciaImporto);
posx=d.posx+(d.dimx/4);
int posy=d.posy+(int)(d.dimy*2.3);
int dimy=(int)((d.dimy*2)*0.8);
Bitmap intestatoA=correttoreGrafico.dammiImmagine(thumb, new
    Dimension(d.posx,posy,d2.posx-(2*d2.dimx)-posx,dimy));
posx=d.posx+(d.dimx/4);
posy=d.posy+(int)(d.dimy*4.3);
dimy=(int)((d.dimy*2)*0.9);
Bitmap causale=correttoreGrafico.dammiImmagine(thumb, new
    Dimension(d.posx,posy,d2.posx-(2*d2.dimx)-posx,dimy));
String[] uscita=riconosciOCR(new
    Bitmap[]{contoCorrente,importo,intestatoA,causale});
String cau=uscita[3];
int da=-1;
uscita[1]=uscita[1].toLowerCase().replace(" ", "");
for(int i=0;i<uscita[1].length();i++){
 if(uscita[1].charAt(i)>='0'&&uscita[1].charAt(i)<='9'){
  da=i;
  break;
 }
}
float impo;
if(da==-1){
 impo=0;
}
else{
uscita[1]=uscita[1].substring(da, uscita[1].length());
uscita[1]=uscita[1].replace("o", "0");

if(uscita[1].charAt(uscita[1].length()-3)>'9'||uscita[1].charAt(uscita[1].length()-3)
 uscita[1]=uscita[1].replace(uscita[1].charAt(uscita[1].length()-3),
     '.');
Log.i("Valore Float",uscita[1]);
impo=Float.parseFloat(uscita[1]);
}
da=-1;
uscita[0]=uscita[0].toLowerCase().replace(" ", "");
for(int i=0;i<uscita[0].length();i++){
 if(uscita[0].charAt(i)>='0'&&uscita[0].charAt(i)<='9'){
  da=i;
```

```
69      break;
70     }
71    }
72    int CC=0;
73    if(da==-1){
74     CC=0;
75    }
76    else{
77     Log.v("Uscita[0]",uscita[0]);
78     while(uscita[0].charAt(da-1)=='b'||uscita[0].charAt(da-1)=='o')
79      da--;
80     uscita[0]=uscita[0].substring(da, uscita[0].length());
81     uscita[0]=uscita[0].replace("o", "0");
82     uscita[0]=uscita[0].replace("b", "8");
83     StringBuilder ret= new StringBuilder();
84     for(int i=0;i<uscita[0].length();i++){
85      if(uscita[0].charAt(i)>='0'&&uscita[0].charAt(i)<='9')
86       ret.append(uscita[0].charAt(i));
87     }
88     uscita[0]=ret.toString();
89     CC=Integer.parseInt(uscita[0]);
90    }
91    String intestatario=uscita[2];
92    return new bollettino(cau, impo, CC, intestatario);
93    }
```

Vediamo che nel codice postato ci sono dei metodi che non abbiamo visto in passato. Spiego il loro output senza mostrare il metodo poiché poco interessante:

- **correttoreGrafico.incornicia(Bitmap immagine,Dimension d)**: Questo metodo genera un immagine(Bitmap) a bassa risoluzione il cui quadrato definito da *d* viene colorato di rosso. Ciò facilita di molto le operazioni di debug in quanto rende palese dove ha trovato il bollettino.

- **dammiImmagine(Immagine thumb,Dimension d)**: Questo metodo restituisce un'immagine Bitmap ritagliata da *thumb* nelle dimensioni indicate da *d*.

- **evidenzia()**: Questo metodo restituisce un immagine bitmap uguale alla precedente con i pixel neri colorati di rosso, che mi è utile per capire se il metodo *isGrigio()* funziona correttamente.

- **riconosciOCR(Bitmap[] img)**: Effettua il riconoscimento OCR di piú immagini, così facendo si risparmia tempo di caricamento di Tes-

seract per ogni immagine. Durante il processo mostra a video le schermate caricamento.

Dopo il metodo *riconosciOCR* vediamo che ci sono dei metodi che cercano di correggere eventuali errori di riconoscimento dell' OCR (ad esempio lo scambio dello zero per la O) e filtraggio delle informazioni utili (rimozione ad esempio del testo sul C/C del bollettino). Infine le informazioni vengono salvate nell'oggetto Bollettino e restituite.

3.5.4 L'activity

Anche questo metodo viene racchiuso all'interno di un'Activity. Questa differisce dalle precedenti create per il fatto che deve restituire un valore. Vediamone il codice.

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

    questa=SoftwareActivity.img;

    schermataCaricamento(questa);
    Thread t= new Thread(new Runnable(){
      public void run() {
        bollettino b= bollettino(questa);
        Intent intent=getIntent();
        intent.putExtra("bollettino", b);
        setResult(RESULT_OK, intent);
        finish();
      }

    });
    t.start();
}
```

Intanto notiamo che l'esecuzione dell'Activity viene affidata ad un thread. Questo per un motivo ben preciso: il metodo *onCreate* é quello responsabile della creazione della nuova Activity, e finché questo non termina, l'Activity non viene creata. Conseguentemente a ció noi a video non la vediamo apparire fino al termine del metodo *onCreate()*. Quindi, se non fosse in un thread, alla pressione del tasto Riconosci bollettino, la grafica rimarrebbe bloccata fino all'uscita dei risultati.

Altra novità in questa Activity é il metodo *putExtra* applicato a un intent generico. Questo metodo serve ad aggiungere dei dati da allegare all'Intent di risposta che invieremo all'Activity padre. Con il metodo putExtra possiamo

inviare qualsiasi tipo di variabile standard (tipo int, char, double ecc..) in piú possiamo inviare qualsiasi Oggetto che implementi la classe Serializable.

Il metodo setResult(RESULT_OK,intent) impone che al termine dell'Activity venga notificato che l'esecuzione é andata a buon fine producendo i risultati attesi, e imposta l'intent da restituire.

Per ricevere le informazioni nella *SoftwareActivity* ovvero quella principale dobbiamo aggiungere questo pezzo di codice al metodo *onActivityResult*

```
if(requestCode== BOLLETTINO_REQUEST_CODE){
  if (resultCode == RESULT_OK) {
  bollettino
      b=(bollettino)data.getSerializableExtra("bollettino");
  schermataVisualizza(img,b.toString());
} else if (resultCode == RESULT_CANCELED) {
  Toast.makeText(this, "Il riconoscimento del bollettino non \'e
      andato a buon fine",Toast.LENGTH_LONG).show();
}
else {
  Toast.makeText(this, "Errore, ma non dovrebbe mai arrivare
      qui",Toast.LENGTH_LONG).show();
}
```

3.6 Invio delle informazioni

Adesso che abbiamo le informazioni incapsulate nell'Oggetto bollettino possiamo inviare le informazioni acquisite tramite SMS alla banca per effettuare il pagamento. Come prima cosa diamo la possibilità all'utente di correggere gli eventuali errori di riconoscimento di *tesseract*.

3.6.1 Layout per editare le informazioni

Creiamo quindi un editor visuale che ci consenta di editare le informazioni. Per farlo dobbiamo creare una nuova View e una nuova Activity. La nuova Activity verrà chiamata visBollettino. Quindi creiamo la classe che estende Activity e la dichiariamo nell'*AndroidManifest* come spiegato prima. Creiamo un nuovo layout che consenta la modifica delle informazioni come mostrato in figura 3.4 Vediamo ora il codice che lo genera.

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

 <ScrollView
```

h 

Figura 3.4: Bollettino

```
    xmlns:android="http://schemas.android.com/apk/res/android"
6   android:layout_width="fill_parent"
    android:layout_height="fill_parent">
8
    <LinearLayout
10      android:id="@+id/linearLayout2"
        android:layout_width="match_parent"
12      android:layout_height="wrap_content"
        android:orientation="vertical" >
14
        <TextView
16          android:id="@+id/Tcc"
            android:layout_width="wrap_content"
18          android:layout_height="wrap_content"
            android:text="@string/cc"
20          android:textAppearance="?android:attr/textAppearanceMedium"
                />
22
        <EditText
            android:id="@+id/contCont"
24          android:layout_width="match_parent"
            android:layout_height="wrap_content"
26          android:inputType="textPersonName" >
```

```
28              <requestFocus />
            </EditText>
30
            <TextView
32              android:id="@+id/Timporto"
                android:layout_width="wrap_content"
34              android:layout_height="wrap_content"
                android:text="@string/importo"
36              android:textAppearance="?android:attr/textAppearanceMedium"
                    />

38          <EditText
                android:id="@+id/importo"
40              android:layout_width="match_parent"
                android:layout_height="wrap_content"
42              android:inputType="textPersonName" />

44          <TextView
                android:id="@+id/Tintestatario"
46              android:layout_width="wrap_content"
                android:layout_height="wrap_content"
48              android:text="@string/intestatario"
                android:textAppearance="?android:attr/textAppearanceMedium"
                    />
50
            <EditText
52              android:id="@+id/intestatario"
                android:layout_width="match_parent"
54              android:layout_height="wrap_content"
                android:inputType="textPersonName" />
56
            <TextView
58              android:id="@+id/Tcausale"
                android:layout_width="wrap_content"
60              android:layout_height="wrap_content"
                android:text="@string/causale"
62              android:textAppearance="?android:attr/textAppearanceMedium"
                    />

64          <EditText
                android:id="@+id/causale"
66              android:layout_width="match_parent"
                android:layout_height="wrap_content"
68              android:inputType="textPersonName" />
            <RelativeLayout
70      android:id="@+id/linearLayout1"
        android:layout_width="match_parent"
72      android:layout_height="wrap_content">
```

```
74      <Button
            android:id="@+id/back"
76          android:layout_width="wrap_content"
            android:layout_height="wrap_content"
78          android:text="@string/back" />

80      <Button
            android:id="@+id/invia"
82          android:layout_width="wrap_content"
            android:layout_height="wrap_content"
84          android:layout_alignParentRight="true"
            android:layout_alignParentTop="true"
86          android:text="@string/send" />
    </RelativeLayout>
88  </LinearLayout>

90  </ScrollView>
```

../software/res/layout/bollettino.xml

Nel listato del codice emergono due nuovi elementi: **EditText** e **ScroolView**. **EditText** é semplicemente un campo che consente all'utente l'inserimento di dati. **ScroolView** invece é un tipo di *View* che consente lo scorrimento di tutti gli elementi che contiene. Quindi se lo schermo é troppo piccolo per visualizzare contemporaneamente tutti gli elementi, tramite scorrimento é possibile visualizzarli tutti comunque. Se non effettuiamo ulteriori modifiche notiamo che il simulatore e tutti i cellulari con tastiera fisica non hanno problemi a far funzionare correttamente l'app. Se invece utilizziamo un cellulare con tastiera virtuale[17] notiamo che questa può posizionarsi sopra i campi e rende difficile l'inserimento del testo. Per risolvere questo problema inseriamo nel manifest questa riga di codice

```
1 android:windowSoftInputMode="adjustResize">
```

tra i tag *activity* dell'activity principale. Grazie a questo inserimento quando compare la tastiera il programma sposta automaticamente il campo di testo sopra la tastiera in modo da vedere ció che si sta scrivendo. Ecco il metodo onCreate dell'Activity

```
1 public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
3     questaActivity=this;
      Intent i= getIntent();
```

[17] Ovvero una tastiera non fisica, che compare sul touch screen al momento dell'immissione di un testo

```
5       bollettino
            b=(bollettino)i.getSerializableExtra("bollettino");
        float a=(float) -1;
7       if (b==null){
          b=new bollettino("Errore", a, 0,"Errore");
9       }
        setContentView(R.layout.bollettino);
11
        impostaBollettino(b);
13      azioniPulsanti();

15    }
```

Possiamo notare il passaggio dell'informazione tramite il metodo *getSerializableExtra* e il caricamento del Layout. Se non viene passato alcun bollettino viene considerato il bollettino Errore. Questo metodo é stato utilizzato per testare la visualizzazione senza dover effettuare il riconoscimento ogni volta, ovviamente si potrebbe sostituire la riga con il seguente codice.

```
1 if (b==null){
    throw new IllegalArgumentException("Bollettino mancante");
3 }
```

Il metodo **impostaBollettino()** semplicemente compila i campi vuoti degli EditText. Non viene riportato perché lo giudico poco interessante. Il metodo *azioniPulsanti()* attribuisce al tasto back l'azione di terminare l'Activity. Anche questo non viene riportato per le medesime motivazioni.

A questo punto non rimane che inviare i dati alla banca rispettando un formato. Siccome il protocollo di invio non é stato ancora definito, deve poter essere modificato agevolmente anche dopo la creazione dell'applicazione, faccio in modo che lo si possa editare in un file xml. I file xml esterni al programma vanno inseriti nella cartella *xml* (sottocartella di *res*)[18].

```
1 <?xml version="1.0" encoding="utf-8"?>
<variabili>
3     <!-- $CC$ campo conto corrente -->
      <!-- $importo$ campo importo senza simboli di euro aggiunti,
          solo la cifra -->
5     <!-- $causale$ campo causale -->
```

[18]Per creare un nuovo file xml con Eclipse nella sezione *Package Explorer* clicchiamo col destro sul nostro progetto → new → Other... infine andiamo su Android Xml File, nel menù a tendina scegliamo values, scegliamo il nome e clicchiamo su OK. Dopo aver creato il file dobbiamo spostarlo nella cartella xml, se non esiste la creiamo

```
    <!-- $intestatario$ campo intestatario -->
    <sms type="bollettino">Conto Corrente $CC$, importo:
        $importo$ causale: $causale$ intestatario:
        $intestatario$ eseguito: $nome$ $indirizzo$ $CAP$
        $localita$</sms>
    <numero type="bollettino">3481064869</numero>
</variabili>
```

../software/res/xml/variabili.xml

Ho scelto di definire tra dollari le parole chiave che poi nel programma verranno sostituite con quelle reali estrapolate dal bollettino.

3.6.2 lettura del codice XML

Per leggere un documento XML ho dovuto creare una classe che mi consentisse agevolmente di prelevare valori.

```
public class XMLReader {
  public static String getTagKey(int contenitore,String
      chiave,String tipo, Activity a){
    try {
      XmlResourceParser p= a.getResources().getXml(contenitore);
      int eventType= p.getEventType();
      boolean giusto=false;
      while(eventType!=XmlResourceParser.END_DOCUMENT){

        if(eventType==XmlResourceParser.START_TAG){
          String tagName=p.getName();
          if(chiave.equals(tagName)){
            String Tipo=p.getAttributeValue(null,"type");
            Log.v("Codice type", Tipo);
            if(Tipo.equals(tipo)){
              giusto=true;
              Log.v("Codice Type","Era quello che cercavamo");
            }
            else{
              giusto=false;
            }
          }
          else{
            giusto=false;
          }
        }
        else if(eventType==XmlResourceParser.TEXT){
          if(giusto){
            String ret=p.getText();
            return ret;
          }
```

```
31            }
              eventType= p.next();
33          }
        } catch (Exception e) {
35          e.printStackTrace();
            return null;
37      }

39      return null;
    }
41  }
```

Questo metodo accetta in ingresso:

- **contenitore:** un *int* che contiene il codice del file xml da cui bisogna cercare le chiavi
- **chiave:** uno *String* che contiene il nome del tag che si vuole cercare
- **tipo:** uno *String* che contiene il nome del dell'attributo type associato al tag.
- **a:** Una variabile di tipo *Activity*.

3.6.3 Spedizione SMS

Adesso che sappiamo come prendere una stringa dall'xml possiamo inviare effettivamente l'sms.

Come prima cosa, onde evitare che l'sms venga inviato piú volte per pressioni multiple accidentali del tasto, con effetti non troppo piacevoli sul conto corrente, lo disabilitiamo dopo la pressione e lo manteniamo disabilitato finché non abbiamo la conferma che il messaggio non é stato inviato.

Come seconda cosa dobbiamo sostituire le parole chiave che otteniamo dall'XML con quelle che l'utente ha inserito nei campi di testo.

Infine dobbiamo inviare l'sms vero e proprio e dare una conferma all'utente che il messaggio é stato effettivamente inviato. Ecco il codice associato alla pressione del tasto invia SMS.

```
1  public void onClick(View v) {
     ((Button)findViewById(R.id.invia)).setEnabled(false);
3    String sms=XMLReader.getTagKey(R.xml.variabili, "sms",
         "bollettino", questaActivity);
     Log.v("Lettura XML SMS", sms);
5    String
         CC=((EditText)findViewById(R.id.contCont)).getText().toString();
```

```
    String
        importo=((EditText)findViewById(R.id.importo)).getText().toString();
    String
        intestatario=((EditText)findViewById(R.id.intestatario)).getText().toString();
    String
        causale=((EditText)findViewById(R.id.causale)).getText().toString();
    sms=sms.replace("$CC$", CC);
    sms=sms.replace("$importo$", importo);
    sms=sms.replace("$intestatario$", intestatario);
    sms=sms.replace("$causale$", causale);
    sms=sms.replace("$nome$", "nome");
    sms=sms.replace("$indirizzo$", "indirizzo");
    sms=sms.replace("$CAP$", "CAP");
    sms=sms.replace("$localita$", "localit\'a");
    Log.v("Lettura XML SMS", sms);
    Log.v("Lunghezza SMS", sms.length()+"");
    String numeroTel=XMLReader.getTagKey(R.xml.variabili,
        "numero", "bollettino", questaActivity);
     sendSMS(numeroTel,sms);
}
```

Il metodo *sendSMS* deve prendere il messaggio e controllare se questo è maggiore o minore di 160 caratteri: se é inferiore manda un messaggio singolo, altrimenti manda un messaggio composto (*multipartSms*). Inoltre deve verificare che l'invio sia effettivamente avvenuto oppure si sia verificato qualche errore (dovuto ad esempio alla mancanza di rete).

```
private void sendSMS(String phoneNumber, String message) {
  String SENT = "SMS_SENT";
  String DELIVERED = "SMS_DELIVERED";
  SmsManager sms = SmsManager.getDefault();
  final int numeroParti;
  ArrayList<String> parts=null;
  if(message.length()<=160){
    numeroParti=1;
  }
  else{
    parts = sms.divideMessage(message);
    numeroParti=parts.size();
  }
  PendingIntent sentPI = PendingIntent.getBroadcast(this, 0, new
      Intent(SENT), 0);
  BroadcastReceiver bro=new BroadcastReceiver(){
    int inviati=0;
    public void onReceive(Context arg0, Intent arg1) {
    if(inviati<0){
      return;
```

```
    }
    switch (getResultCode()){
      case Activity.RESULT_OK:
      inviati++;
      if(inviati>=numeroParti){
        Toast.makeText(getBaseContext(), "SMS inviato con
            successo",
                                        Toast.LENGTH_SHORT).show();
        unregisterReceiver(this);
        finish();
      }
      else{
        Toast.makeText(getBaseContext(),"Invio in Corso...
            "+inviati+"/"+numeroParti, Toast.LENGTH_SHORT
            ).show();
      }
      break;
      case SmsManager.RESULT_ERROR_GENERIC_FAILURE:
                            Toast.makeText(getBaseContext(),
                                "Errore invio",
      Toast.LENGTH_SHORT).show();
      inviati=-1;
      ((Button)findViewById(R.id.invia)).setEnabled(true);
      break;
      case SmsManager.RESULT_ERROR_NO_SERVICE:
        Toast.makeText(getBaseContext(), "SMS non inviato per
            assenza di servizio",
        Toast.LENGTH_SHORT).show();inviati=-1;
        ((Button)findViewById(R.id.invia)).setEnabled(true);
        break;
      case SmsManager.RESULT_ERROR_NULL_PDU:
        Toast.makeText(getBaseContext(), "Null PDU",
            Toast.LENGTH_SHORT).show();
        inviati=-1;
        ((Button)findViewById(R.id.invia)).setEnabled(true);
        break;
      case SmsManager.RESULT_ERROR_RADIO_OFF:
        Toast.makeText(getBaseContext(), "SMS non inviato
            perch\'e il cellulare \'e in modalit\'a OFFLINE o
            AEREO",
        Toast.LENGTH_LONG).show();
        inviati=-1;
        ((Button)findViewById(R.id.invia)).setEnabled(true);
        break;
      }
    }

  };
  if(message.length()<=160){
```

```
61      registerReceiver(bro, new IntentFilter(SENT));
        sms.sendTextMessage(phoneNumber, null, message, sentPI,
            null);
63    }
      else{
65      ArrayList<PendingIntent> PI = new ArrayList<PendingIntent>();
        registerReceiver(bro, new IntentFilter(SENT));
67      for(int i=0;i<numeroParti;i++){
          PI.add(PendingIntent.getBroadcast(this, 0, new
              Intent(SENT), 0));
69      }
        sms.sendMultipartTextMessage(phoneNumber, null, parts, PI,
            null);
71    }
  }
```

In questo caso non usiamo un Intent per mandare un messaggio, perché non andremo ad utilizzare un'Activity, ma direttamente le API native del cellulare.

L'**SmsManager** é un oggetto che ci consente di inviare un messaggio. Il **BroadcastReceiver** é un'interfaccia che ha il metodo *onReceive()* che viene richiamato al termine di un'operazione, in questo caso al termine dell'operazione di invio dell'SMS.

Come funziona il BroadcastReceiver? Il *BroadcastReceiver* viene creato implementando il metodo definito nell'interfaccia. In seguito viene associato a un IntentFilter (in questo caso associato alla stringa *SMS_SENT* contenuto nella variabile SENT). L'associazione viene fatta quando si usa il metodo **registerReceiver**. Così facendo si é impostato un intent filter, cioè quando viene chiamato un intent (non esplicito) associato alla stringa "*SMS_SENT*", risponderà il nostro broadcast receiver. Ovviamente questo é valido finché l'Activity non termina. Se si vuole creare un intent filter valido per tutte le applicazioni lo si deve definire all'interno del file *AndroidManifest.xml* come vedremo in seguito. Ora dobbiamo solo comunicare al metodo che invia l'SMS quale Intent chiamare al termine della sua esecuzione, che sarà *new Intent(SENT)*. Lo comunichiamo tramite l'oggetto **PendingIntent**. Per crearlo usiamo il metodo statico **getBroadcast** a cui chiediamo di darci il *pending intent* associato a questa Activity che risponde alla chiamata *SENT*. Infine inviamo il messaggio tramite il metodo **sendTextMessage** se il messaggio ha un numero massimo di 160 caratteri, altrimenti usiamo il metodo **sendMultipartTextMessage**.

Come si può notare, a differenza del primo, che accetta uno singolo, il secondo metodo accetta una lista concatenata di *pendingIntent*. Questo perché quando si invia un messaggio composto da piú parti, il telefono invia tutte le parti come messaggi effettivamente separati e, al termine di ogni invio, viene chiamato il pendingIntent associato. Nel nostro caso il pendingIntent é lo stesso ma in una situazione generale possiamo richiamare *pending intent* diversi per ogni parte del messaggio.

Diamo uno sguardo ora all'implementazione dell'interfaccia **BroadcastReceiver**. Tramite il metodo *getResultCode()* riusciamo a capire se il messaggio associato è stato inviato oppure no. L'intero che viene restituito può assumere i valori che vediamo visualizzati nel metodo stesso, i cui significati sono spiegati nei *Toast*. Al termine di ogni tipo di errore viene riabilitato il tasto *invia*, per consentire il rinvio del messaggio. Se tutti i messaggi sono stati inviati con successo l'Activity termina e notifica all'utente l'esito positivo.

Capitolo 4

Sviluppo dell'Applicazione Codice a Barre

Passiamo adesso alla creazione di un tool per il riconoscimento dei codici a barre nell'applicazione.

4.1 Obiettivi

L'obiettivo di questo tool è di riconoscere dei codici a barre di tipo EAN-13[1]. E associare ad ogni codice una descrizione in modo che, ogni volta che si ripresenti lo stesso codice, il programma mostri la descrizione definita al momento dell'inserimento. Questo tool è la base per poi (in futuro) usare un database online condiviso da più cellulari/pc che utilizzino quest'applicazione.

4.2 I codici a barre EAN-13

I codici a barre EAN-13 sono un tipo particolare di codice a barre. Sono composti da 13 cifre di cui 12 significative e la tredicesima di controllo.

4.2.1 Generazione della cifra di controllo

La cifra di controllo viene generata in funzione delle altre 12 e serve a limitare gli errori di lettura del codice stesso. In pratica il lettore legge il codice a barre (tutte e 13 le cifre) e poi partendo dalle prime 12, genera la tredicesima. Se questa combacia con quella letta allora possiamo accettare il codice. Così

[1]Spiegati nel capitolo seguente

Figura 4.1: Esempio di codice a barre EAN-13

facendo diminuisce sensibilmente la probabilità di avere un errore durante la lettura.
La tredicesima cifra si ottiene nel seguente modo:

- Definiamo **D** come la somma delle cifre nelle posizioni dispari del codice letto. (cioè la prima cifra + la terza + la quinta ecc...)
- Definiamo **P** come la somma delle cifre nelle posizioni pari del codice letto.
- Eseguiamo questa somma **N**=D+(P*3)
- Definiamo **S** come la decina superiore di N. (Cioè se N=11 allora S=20, se N=24 allora S=30, se N=40 allora S=40)
- La cifra di controllo **C** si ottiene facendo C=S-N.

Esempio Se il codice é **123456789012** per calcolare la tredicesima cifra facciamo:
D=1+3+5+7+9+1=26
P=(2+4+6+8+0+2)=22
N=26+(22*3)=26+66=92
S=100

C=100-92=**8**

La codifica del codice a barre vera e propria viene spiegata in maniera esauriente nella sezione 4.7.

4.3 La libreria Zxing e L'applicazione Barcode Scanner

4.3.1 Introduzione

Durante la ricerca di un'applicazione open source da poter inserire nel progetto mi sono imbattuto subito in Barcode Scanner che utilizza la libreria Zxing per riconoscere i codici a barre. Quest'applicazione, la cui documentazione é disponibile nella bibliografia [ZXing (Zebra Crossing)], non è solo in grado di riconoscere i codici a barre EAN-13, ma anche la maggior parte dei codici a barre presenti sul mercato e codici QR. Dobbiamo quindi tenere conto che potrebbe restituirci un codice non in formato EAN-13 e gestire l'evento.

4.4 Integrazione di Barcode Scanner col programma

Grazie all'utilizzo di Intent non espliciti si può utilizzare Activity non inclusa nell'applicazione. Quindi per integrare il software possiamo seguire due strade: includere i sorgenti nel software oppure chiedere all'utente di scaricare l'applicazione barcode scanner e utilizzarla tramite intent. Vediamole entrambe.

4.4.1 Utilizzo di barcode Scanner non incluso nell'applicazione

Per utilizzare il barcode scanner non incluso nell'applicazione dobbiamo necessariamente verificare che questo sia stato installato dall'utente e, in caso contrario farglielo installare. Per sapere se il Barcode Scanner é stato installato sul cellulare usiamo il PackageManager. Ecco le righe di codice di cui abbiamo bisogno.

```
PackageManager packageManager = this.getPackageManager();
ResolveInfo resolveInfo = packageManager.resolveActivity(
    SCAN_INTENT, PackageManager.GET_RESOLVED_FILTER);
```

dove *SCAN_INTENT* é un intent cosí definito

```
private static final Intent SCAN_INTENT = new
    Intent("com.google.zxing.client.android.SCAN");
```

cioè associato alla particolare Activity scansione di Android Scanner. Se la variabile *resolverInfo* é **null** allora non c'è nessun Intent Filter che risponde alla chiamata di quell'Intent, ovvero Barcode Scanner non è installato. In questo caso notifichiamo all'utente e lo indirizziamo verso il download dell'applicazione. In caso contrario lo facciamo accedere al sottomenù che consentirá poi di aggiungere, visualizzare e verificare i codici a barre.

4.4.2 Inserimento di barcode Scanner nel progetto

Se invece vogliamo inglobare barcode Scanner nel progetto dobbiamo operare come segue. Intanto procurarci i sorgenti (scaricandoli dal sito indicato a 4.3.1): lo zip di Zxing contiene tutti i progetti per tutte le piattaforme, a noi interessa soltanto android e core, per cui tutto il resto può essere eliminato. Creiamo un nuovo progetto con eclipse e importiamo tutti i file contenuti nella cartella Android. Dopo l'importazione ci saranno sicuramente degli errori perché dobbiamo anche inserire la libreria Zxing.

Per importarla clicchiamo col destro sul progetto appena creato -> build Path -> Configure build path. In librarys aggiungiamo la libreria *core.jar* che si trova nella cartella *Core* del file appena scaricato, aggiungiamo anche core.jar al nostro progetto originale e, infine, mettiamo la spunta su isLibrary, come abbiamo visto in 3.2.2. Andiamo nelle impostazioni del nostro progetto principale e nelle proprietà->(Scheda)Android oltre a tesseract aggiungiamo anche il progetto appena creato. Ora abbiamo aggiunto il Barcode Scanner al progetto .

4.4.3 Creazione di un intentFilter

Il problema ora è che il progetto non sa che può soddisfare gli intent di tipo *SCAN_INTENT*. Quindi dobbiamo andare dentro al file *AndroidManifest.xml* e dichiarare il nuovo **intent Filter**. Apriamo quindi il file AndroidManifest e inseriamo questo codice prima della fine del tag *application*

```
1 <activity
    android:name="com.google.zxing.client.android.CaptureActivity"
```

```
android:screenOrientation="landscape"
android:configChanges="orientation|keyboardHidden"
android:theme="@android:style/Theme.NoTitleBar.Fullscreen"
android:windowSoftInputMode="stateAlwaysHidden">
<intent-filter>
  <action android:name="com.google.zxing.client.android.SCAN"/>
  <category android:name="android.intent.category.DEFAULT"/>
</intent-filter>
</activity>
```

Abbiamo così inserito una nuova Activity: **CaptureActivity**. Tutti i campi prima di *<intent-filter>* servono a definire l'orientamento, la visualizzazione della tastiera, e impostano il full screen. all'interno dei tag *<intent-filter>* vediamo la definizione dell'Intent.

Abbiamo quindi detto che, ad ogni chiamata di Intent non esplicito con la stringa indicata in action, risponde l'Activity *CaptureActivity*.

4.5 Creazione e immagazzinamento dati in un DataBase SQLite

Creiamo un metodo statico che ci restituisca un dataBase SQLite funzionante. Il metodo crea un file chiamato database.db nella cartella tesseract (già creata nell'activity precendente) dove al suo interno viene scritto un database contenente una tabella (se non esiste) denominata codiciabarre la quale contiene due campi:

ID Un long contenente il codice a barre che é anche chiave primaria

DESCRIZIONE un varchar (una Stringa) contenente una descrizione di massimo 300 caratteri.

```
class dataBaseCreator{
  public static SQLiteDatabase getDatabase(Activity a){

    File Dir = Environment.getExternalStorageDirectory();
    Dir= new File(Dir,"tesseract");
    Dir= new File(Dir,"database.db");
    SQLiteDatabase data=SQLiteDatabase.openOrCreateDatabase(Dir,
        null);
    data.execSQL("CREATE TABLE IF NOT EXISTS
        \"codiciabarre\"(\"id\" LONG PRIMARY KEY,
        \"descrizione\" VARCHAR(300) )");
    return data;
```

```
10    }

12 }
```

Come si vede dal codice, il metodo ci restituisce il database sottoforma di un oggetto di tipo SQLiteDatabase; se non esiste perché è la prima volta che il metodo viene eseguito su una determinata macchina, lo crea. Vediamo come inserire un codice a barre all'interno del DataBase SQL

```
    data.execSQL("INSERT INTO codiciabarre (id,descrizione)
        VALUES ("+codiceABarre+",'"+descrizione+"')");
2   data.close();
    finish();
```

Vediamo così che stiamo inserendo una coppia id,descrizione nel DataBase mediante SQL nativo. Al termine dell'operazione chiudiamo il database e l'Activity. É necessario invocare il metodo data.close() per evitare che vada in errore alla chiusura dell'Activity o alla riapertura del database all'interno della stessa. Diamo un'occhiata al codice per verificare se un entry è già presente nel DataBase:

```
1 SQLiteDatabase data=dataBaseCreator.getDatabase();
    Cursor c=data.query("codiciabarre", new
        String[]{"descrizione"}, "id="+codiceABarre, null, null,
        null, null);
3   String descrizione;
    if(c.moveToNext())
5     descrizione=c.getString(0);
    else{
7     descrizione="Codice non registrato";
    }
9   c.close();
    data.close();
```

Creiamo quindi una variabile di tipo Cursor dove vengono passati i parametri:

- la tabella (*codiciabarre*) su cui vogliamo effettuare la Query
- i campi che ci interessano (*descrizione*)
- il WHERE della Query

Gli altri parametri, che nel nostro caso sono impostati a **null**, sono le altre componenti dell'SQL, come il groupBy, l'Having e l'OrderBy che, nel nostro particolare caso, non ci interessano. Il metodo *moveToNext()* chiede alla variabile Cursor se c'è almeno una riga che rispetta la query. Se non c'è nella variabile descrizione verrà impostata la stringa *Codice non registrato*. Per visualizzare tutti i codici a barre con relative descrizioni registrati nel database dobbiamo effettuare una Query piú semplice che peró risponde piú di un valore. Vediamo quindi come estrapolare tutte le righe della query.

```
SQLiteDatabase data= dataBaseCreator.getDatabase();
    Cursor c=data.query("codiciabarre", new String[] "id"
        ,"descrizione"}, null ,null, null, null, null);
    while(c.moveToNext()){
      sfondo.addView( creaVista( sfondo, c.getLong(0),
          c.getString(1)));
    }
    c.close();
    data.close();
```

In questo caso abbiamo lasciato a **null** il campo *SELECT* che equivale ad avere un *SELECT **. Finché ci sono elementi, viene creata una vista con la visualizzazione del codice a barre e della descrizione, infine si chiude il database. Il metodo **creaVista** lo vediamo nella prossima sezione.

4.6 Creare una lista di elementi grafici

Per creare una lista di elementi grafici dello stesso tipo abbiamo bisogno del **LayoutInflater**. Il *LayoutInflater* ci consente di duplicare i layout contenuti nei file xml. Per creare una lista di elementi uguali dobbiamo però prima creare l'elemento da copiare.

4.6.1 Creazione dell'elemento

Quindi creiamo un nuovo file Layout xml. Questo file conterrà l'immagine del codice a barre (che vedremo come generare dal codice nella sezione 4.7) a sinistra, e subito a destra la descrizione associata al codice, come mostrato in figura 4.2. Il codice xml per generare questo layout é il seguente:

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

    <RelativeLayout
```

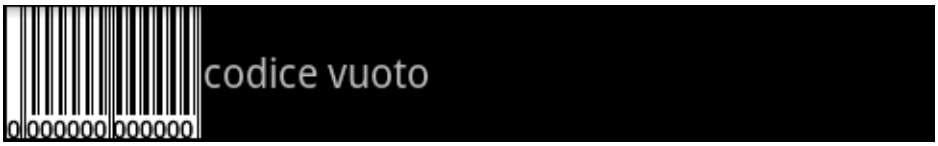

Figura 4.2: Creazione dell'elemento

```
5           android:id="@+id/vista"
                xmlns:android="http://schemas.android.com/apk/res/android"
            android:layout_width="match_parent"
7           android:layout_height="wrap_content"
            android:paddingTop="1dp"
9           android:paddingBottom="1dp">

11              <ImageView
                    android:id="@+id/immagineCodiceABarre"
13                  android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
15                  android:layout_alignParentLeft="true"
                    android:layout_centerVertical="true"
17                  android:contentDescription="@string/barre"
                    android:src="@drawable/loading" />
19
                <TextView
21                  android:id="@+id/descrizioneCodiceABarre"
                    android:layout_width="wrap_content"
23                  android:layout_height="wrap_content"
                    android:layout_centerVertical="true"
25                  android:paddingLeft="3dp"
                    android:layout_toRightOf="@+id/immagineCodiceABarre"
27                  android:text="@string/barre"
                   />
29
        </RelativeLayout>
```

../software/res/layout/elemento.xml

Il campo *android:padding* serve a definire quanto margine lasciare, in modo tale che gli elementi non siano spiacevolmente attaccati gli uni agli altri. Per il resto, i comandi sono tutti già conosciuti.

4.6.2 Duplicazione elementi grafici

Ora che abbiamo il nostro elemento.xml da usare per visualizzare ogni singolo codice a barre, dobbiamo duplicarlo tante volte quanti sono i codici nel database e assegnargli l'immagine relativa al codice e la sua descrizione.

Prima di tutto però dobbiamo creare un nuovo file xml con un layout a scorrimento dove inserire tutti questi elementi. Abbiamo già visto come si crea un layout a scorrimento tramite *ScroolView* in 3.6.1. Creiamo quindi uno ScroolView e, al suo interno, mettiamo un *LinearLayout*.

Vediamo ora il metodo *creaVista*:

```
private View creaVista(ViewGroup vista,final long codice,final
    String descrizione){
    LayoutInflater inflater = (LayoutInflater) getSystemService(
        Context.LAYOUT_INFLATER_SERVICE);
    View view = inflater.inflate( R.layout.elemento, vista,
        false);
    ((ImageView) view.findViewById(
        R.id.immagineCodiceABarre)).setImageBitmap(
        correttoreGrafico.creaCodiceEan13(codice, 1));
    ((ImageView)view.findViewById( R.id.immagineCodiceABarre)
        ).setContentDescription(codice+"");
    ((TextView)view.findViewById
        (R.id.descrizioneCodiceABarre)).setText( descrizione);
    view.setClickable(true);
    view.setOnClickListener(new OnClickListener() {
      public void onClick(View v) {
        schermataVisualizza( correttoreGrafico.creaCodiceEan13(
            codice, 2), descrizione);
      }
    });
    view.setLongClickable(true);
    registerForContextMenu(view);
    return view;
  }
```

Il *LayoutInflater* crea appunto dei duplicati del layout *elemento*, ai quali poi sostituiamo l'immagine e la descrizione con quelle passate dai due valori in ingresso (**codice** e **descrizione**). Modifichiamo inoltre anche il *ContentDescription* (vedremo in seguito a cosa ci serve modificarlo).

Ad ogni elemento associamo un particolare *OnClickListener* che gli fa visualizzare il codice in grande con la relativa descrizione.

Il metodo **regiterForContextMenu** serve a far comparire, alla pressione lunga dell'elemento, un menù, che vedremo dopo come creare.

4.6.3 ContextMenu

Il **ContextMenu** é un menù che compare sopra l'Activity e permette la scelta di uno o piú parametri.

Le varie opzioni del menú vengono definite in un file xml. Per crearlo andiamo su New->Other.. e selezioniamo Android XML file. Poi dal menù a tendina accanto a *resource Type* selezioniamo Menù. Ora possiamo inserire tutte le voci del menù che vogliamo. In questo caso solo una, ovvero *delete*. É sufficiente compilare i campi **id** e **title** per avere la voce del menú completa e funzionante.

Adesso dobbiamo comunicare all'Activity quale menù lanciare e lo definiamo sovrascrivendo il metodo **onCreateContextMenu**

```
public void onCreateContextMenu(ContextMenu menu, View
    v,ContextMenuInfo menuInfo) {
     super.onCreateContextMenu(menu, v, menuInfo);
     lanciata=v;
     MenuInflater inflater = getMenuInflater();

     inflater.inflate(R.menu.showmenu, menu);
  }
```

Alla pressione di ogni tasto viene invocato il metodo **onContextItemSelected** che dobbiamo sovrascrivere.

```
public boolean onContextItemSelected(MenuItem item) {
   switch (item.getItemId()) {
        case R.id.delete:
             rimuoviElemento(lanciata);
             return true;
        default:
             return super.onContextItemSelected(item);
   }
}
```

Il metodo **getItemId** ci restituisce l'ID del menù che abbiamo appena premuto (siccome è l'unico disponibile ci restituirà l'ID di delete, ma, in un caso generale, potrebbero essercene di più). Alla pressione del tasto Delete lanciamo il metodo *rimuoviElemento* che rimuove il codice a barre dal video e dal database.

4.7 Generazione dei codici a barre EAN-13

In questa sezione parleremo di come generare l'immagine Bitmap a partire dal codice a barre in ingresso.

4.7.1 La codifica EAN-13

Questa codifica trasforma i 13 numeri passati in ingresso in un codice binario, il quale viene poi trasformato in barrette secondo la logica: 1 corrisponde alla barra nera, 0 corrisponde a quella bianca. Le barre vanno raffigurate tutte attaccate le une alle altre senza spaziature, quindi la raffigurazione di 11 apparirà come un unica sbarra del doppio dello spessore.

Ogni cifra ha tre possibili codifiche, la codifica A,B e C secondo questa tabella:

cifra	*A*	*B*	*C*
0	0001101	0100111	1110010
1	0011001	0110011	1100110
2	0010011	0011011	1101100
3	0111101	0100001	1000010
4	0100011	0011101	1011100
5	0110001	0111001	1001110
6	0101111	0000101	1010000
7	0111011	0010001	1000100
8	0110111	0001001	1001000
9	0001011	0010111	1110100

Detto questo, possiamo iniziare a dare un'occhiata alla figura 4.1. Intanto notiamo che le prime due barrette, le due centrali e le due finali, sono piú lunghe rispetto alle altre. Questo perché sono dei caratteri standard che fanno capire al lettore di codice quanto larga é una barretta. Il codice quindi é composto da 5 parti

- **Parte Iniziale:** composta sempre da **101**
- **Prima Parte:** formata dalla conversione dei caratteri dal **2** al **7**, secondo la codifica **A** o **B** (vedremo poi come)
- **Stacco Centrale:** composto sempre da **01010**
- **Seconda Parte:** formata dalla conversione dei caratteri dall'**8** al **13**, secondo la codifica **C**
- **Parte Finale:** composta sempre dai caratteri **101**

Notiamo subito che la prima cifra non é presente nell'elenco precedente. Questo perché non viene direttamente codificato in forma di barre ma dedotto dal tipo di codifica delle prime 6 cifre. A decidere quali cifre codificare secondo la codifica A o la codifica B è appunto la prima cifra secondo la seguente tabella.

cifra	*codifica*
0	AAAAAA
1	AABABB
2	AABBAA
3	AABBBA
4	ABAABB
5	ABBAAB
6	ABBBAA
7	ABABAB
8	ABABBA
9	ABBABA

Ad esempio se la prima cifra é 2 vorrà dire che la seconda, la terza, la sesta e la settima cifra dovranno essere codificate secondo la codifica A, mentre la quarta e la quinta con la B.

4.7.2 La realizzazione dell'immagine

Per realizzare l'immagine ci serve innanzitutto un metodo che trasformi un ingresso di tipo long in un insieme di bit codificati, come spiegato in 4.7.1. Una volta trasformata in bit, un altro metodo deve creare un'immagine a partire dalla sequenza di bit generata. Ecco il codice piú interessante, ovvero quello che genera la prima e la seconda parte della sequenza:

```
private static final String[] sequenza={"AAAAAA",
    "AABABB","AABBAB","AABBBA","ABAABB"
    ,"ABBAAB","ABBBAA","ABABAB","ABABBA", "ABBABA"};
  private static byte[][] dammiCodifica(String codice){
    if(codice.length()!=13){
      throw new IllegalArgumentException("Il codice inviato non
          ha il numero di caratteri giusto. Quello inviato ne ha
          "+codice.length());
    }
    byte[] primaParte= new byte[42];
    String sequenzaAttuale=sequenza
        [Integer.parseInt(codice.charAt(0)+"")];
    for(int i=1;i<7;i++){
      int ora=(i-1)*7;
      byte[]
          questi=codifica(Integer.parseInt(codice.charAt(i)+""),
          sequenzaAttuale.charAt(i-1));
      for(int j=0;j<7;j++){
        primaParte[ora+j]=questi[j];
      }
    }
    byte[] secondaParte= new byte[42];
```

```
    for(int i=7;i<13;i++){
      int ora=(i-7)*7;
      byte[]
          questi=codifica(Integer.parseInt(codice.charAt(i)+""),
          'C');
      for(int j=0;j<7;j++){
        secondaParte[ora+j]=questi[j];
      }
    }
    return new byte[][]{primaParte,secondaParte};
  }
```

Innanzitutto si vede che non si accettano in ingresso sequenze con una lunghezza superiore o inferiore a 13. In base alla prima cifra, poi viene scelta la codifica da usare per i primi 6 caratteri. Il metodo **codifica** non viene riportato per la sua banalità, questo semplicemente restituisce una sequenza di 1 e di 0 in un array di lunghezza 7 secondo la codifica impostata.

Questo metodo infine ritorna la prima e la seconda parte del codice sotto forma di 2 array di byte.

Vediamo ora come viene generata l'immagine Bitmap del codice a barre a partire dal codice completo, ovvero comprensivo della parte iniziale, quella centrale e quella finale. L'iniziale e la finale, dovendo avere una lunghezza superiore rispetto alla prima e alla seconda parte, avranno valore 2 anziché 1.

```
int lunghezza=(codiceCompleto.length+7)*dimBarra;
Bitmap barre=
    Bitmap.createBitmap(lunghezza,(lunghezza*2)/3,Config.ARGB_8888);
int[] barraNera=new int[barre.getHeight()*dimBarra];
for(int i=0;i<barraNera.length;i++){
  barraNera[i]=Color.BLACK;
}
int[] barraBianca=new int[barre.getHeight()*dimBarra];
for(int i=0;i<barraBianca.length;i++){
  barraBianca[i]=Color.WHITE;
}
for(int i=0;i<7;i++){\\colora di bianco l'immagine per i primi 7
    posti

  barre.setPixels(barraBianca, 0, dimBarra, (i)*dimBarra , 0,
      dimBarra, barre.getHeight());


}
for(int i=0;i<codiceCompleto.length;i++){\\crea le barre
  if(codiceCompleto[i]==1){
```

```
    barre.setPixels(barraNera, 0, dimBarra, (7+i)*dimBarra , 0,
        dimBarra, barre.getHeight());
    int lung=(barre.getHeight()*5)/6;
    barre.setPixels(barraBianca, 0, dimBarra, (7+i)*dimBarra ,
        lung, dimBarra, barre.getHeight()-lung);
  }
  else if(codiceCompleto[i]==0){
    barre.setPixels(barraBianca, 0, dimBarra, (7+i)*dimBarra ,
        0, dimBarra, barre.getHeight());
  }
  else if(codiceCompleto[i]==2){
    barre.setPixels(barraNera, 0, dimBarra, (7+i)*dimBarra , 0,
        dimBarra, barre.getHeight());


  }
}
Canvas c = new Canvas(barre);
c.setDensity(Bitmap.DENSITY_NONE);
Paint p = new Paint();
p.setTextSize(dimBarra*13);
p.setAntiAlias(true);
p.setFakeBoldText(true);
p.setColor(Color.BLACK);
s.insert(1, ' ');
s.insert(8, ' ');
c.drawText(s.toString(), 0, (barre.getHeight()), p);
return barre;
```

La variabile **dimBarra** è la larghezza in pixel della sbarra del codice a barre. **N.B.** Da ora in poi userò come unità di misura la barra, che misurerà tanti pixel quanti impostati nella variabile *dimBarra*. Andiamo a creare un'immagine Bitmap di larghezza pari al numero di barrette da inserire + 7. Il +7 serve a lasciare lo spazio per la prima cifra del codice che deve essere inserito in basso a sinistra[2]. **BarraNera** e **barraBianca** sono la rappresentazione su array di interi delle le barrette, di colore rispettivamente nero e bianco, che poi andremo a inserire nell'immagine.

Il metodo *createBitmap* ci restituisce un'immagine vuota, completamente nera. Siccome vogliamo avere lo spazio riservato al primo carattere bianco, effettuiamo un ciclo **for** che colora di bianco le prime 7 barre.

Il secondo **ciclo for**, partendo dall'ottava barra, ne crea una bianca se nell'array il valore é 0, di nero se é 1 lasciando peró $\frac{1}{5}$ di spazio libero a

[2]sono 7 barrette perché ogni cifra viene rappresentata da 7 bit, quindi 7 barrette

fondo immagine riservato alle cifre. Se il valore è 2, invece, crea una sbarra continua fino in fondo senza lasciare alcuno spazio.

L'ultima parte di codice inserisce in fondo all'immagine le cifre del codice a barre.

Capitolo 5

Sviluppo dell'applicazione per l'acquisto di biglietti

5.1 Obiettivi

Realizzare e sviluppare un applicazione che consenta agevolmente di acquistare,tramite cellulare, biglietti e abbonamenti per autobus, treni o traghetti.

5.2 Ideazione

Per realizzare un progetto del genere abbiamo bisogno di un'applicazione che faccia scegliere all'utente che tipo di biglietto acquistare, da quale compagnia, per quale tratta e in quale quantità. Infine produrre una richiesta di acquisto e notificare l'utente dell'esito dell'operazione.

5.2.1 La località

Per l'acquisto del biglietto è necessario conoscere la località in cui si trova l'utente, in maniera tale da fargli scegliere soltanto tra le compagnie e le tratte che operano localmente semplificando e velocizzando di molto l'acquisto. Inoltre, è possibile che anche stesse compagnie in città diverse offrano dei servizi differenti e con prezzi diversi.

5.2.2 Scelte Successive

Una volta decisa la località si farà scegliere all'utente la **compagnia** presso la quale vuole acquistare il biglietto. In seguito, verrà chiesto di selezionare la modalità di acquisto.

Modalità di acquisto Le modalità di acquisto sono 3: **Biglietto Singolo, Biglietto Multiplo** e **Abbonamento**.

Biglietto Singolo É il modo più veloce per acquistare un biglietto, utile se si è alla fermata e si ha necessità di comprare in fretta il titolo di viaggio.

Biglietto Multiplo Utile se si intende comprare più biglietti dello stesso tipo (comitive o carnet), oppure anche biglietti diversi della stessa compagnia in un unico passaggio (ad esempio tram e autobus).

Abbonamento Per acquistare un abbonamento ad una determinata compagnia di trasporti. Verrà chiesta la data di quando si ha intenzione di attivare l'abbonamento e il numero di tessera.

5.2.3 Pagamento

Il pagamento avverrà attraverso SMS, come spiegato nel paragrafo 3.6. Se l'importo è sufficientemente elevato, come appunto nel caso di stipulazione di abbonamenti, sarà richiesto il codice di controllo (vedi 5.3.6).

In breve, vorremmo ottenere una sequenza di schermate mostrata nella figura 5.1

5.3 Realizzazione

Creiamo una nuova applicazione come descritto in 2, e la chiamiamo BiglietiAutobus.

5.3.1 Disclaimer

Prima di tutto, bisogna far accettare al cliente alcune condizioni di utilizzo del programma e, possibilmente, dargli la possibilità di non visualizzarlo la prossima volta che lo si avvia. Affidiamo la visualizzazione del disclaimer ad un'Activity e creiamo un nuovo Layout da visualizzare.

CheckBox L'unico aspetto interessante in questo layout rispetto a quelli già visti precedentemente è la presenza di un **CheckBox** (Figura 5.2). A differenza dei bottoni, non gli attribuiremo un listener, poiché questo tipo di elemento grafico conserva lo stato, che andremo a leggere dopo che l'utente

avrà premuto il bottone Accetto le condizioni. Riporto qui il pezzo di codice che viene eseguito alla pressione del tasto:

```
public void OnContinua(View view){
    CheckBox c= (CheckBox)findViewById(R.id.nonMostrare);
    if(c.isChecked()){
      preferenze.setNonMostareDisclaimer(this, true);
    }
    setResult(RESULT_OK);
    finish();
}
```

Come si può notare, se il *CheckBox* è stato spuntato, chiama il metodo statico **preferenze.setNonMostareDisclaimer(this, true);**. Questo metodo fa in modo che si salvi la preferenza anche quando termina l'applicazione. Ma vediamone ora il listato:

5.3.2 Salvataggio Preferenze

```
public static void setNonMostareDisclaimer(Activity a,boolean
    bol){
  SharedPreferences prefs=
      a.getSharedPreferences(Const.MY_PREFERENCES,
      Context.MODE_PRIVATE);
  Editor e=prefs.edit();
  e.putBoolean(Const.NON_MOSTRARE, bol);
  e.commit();
}
public static boolean getNonMostareDisclaimer(Activity a){
  SharedPreferences prefs=
      a.getSharedPreferences(Const.MY_PREFERENCES,
      Context.MODE_PRIVATE);

  return prefs.getBoolean(Const.NON_MOSTRARE, false);
}
```

Le informazioni vengono salvate in un data-base interno all'applicazione in cui è possibile salvare dati come in un dizionario, ovvero si può attribuire a una determinata chiave uno specifico valore.

Come si vede dal listato, utilizziamo il metodo **getSharedPreferences** per ottenere l'accesso alle impostazioni salvate, associate alla stringa contenuta in *MY_REFERENCES*. La variabile *MODE_PRIVATE* indica che le

informazioni a cui accedo non sono condivise con altre applicazioni. Il metodo **edit** mi restituisce un oggetto di tipo *Editor* che mi consente di inserire e modificare le impostazioni salvate. Con il metodo **putBoolean** scrivo il valore della variabile booleana *bol* all'interno del DataBase associandolo alla stringa *NON_MOSTRARE*. I tipi di variabile che si possono salvare sono di tipo *boolean*, *float*, *int*, *long* e *String*. Nel nostro esempio abbiamo salvato solo una preferenza ma nel caso in cui volessimo salvarne diverse, basterà attribuire loro una chiave diversa. Prima dell'esecuzione del metodo **commit()** i nuovi dati o modifiche rimangono nell'oggetto "*e*", e quindi in RAM; dopo l'esecuzione del comando verrano fisicamente scritti nella memoria del telefono. La mancata esecuzione del metodo comporta un mancato salvataggio delle impostazioni.

Per reperire le informazioni salvate non si dovrà fare altro che richiamare il metodo get<variabile> (nel nostro caso getBoolean dato che è una variabile booleana) sull'oggetto *prefs*, come si evince analizzando il secondo metodo del listato.

5.3.3 Selezione

Una volta accettate le condizioni, il programma dovrà far scegliere all'utente la città, la compagnia, la modalità ecc... In breve dovrà far compiere all'utente tante scelte multiple. Creiamo quindi un layout standard per la scelta multipla.

Layout scelta multipla Voglio creare un oggetto che mi mostri una lista di elementi selezionabili, come ad esempio un elenco di città. Appena tocco uno degli oggetti della lista questo si deve evidenziare, e alla pressione del tasto conferma devo essere in grado di capire quale o quali elementi sono stati selezionati.

Come prima cosa creo un nuovo tipo di oggetto, l'oggetto **vista<E>** che implementa l'interfaccia Serializable [1]. La <*E*> indica che è un oggetto generico, ovvero uno o più dei suoi campi sarà un oggetto non definito a priori. Vediamone il listato.

```
1 class vista<E> implements Serializable{
  /**
3  *
   */
5 private static final long serialVersionUID = 1L;
  public String testo,descrizione;
```

[1] Il motivo per cui l'oggetto estende l'interfaccia **Serializable** è che in tal modo è possibile inviarlo attraverso le varie Activity

```
7   public E ogg;
    public Bitmap img;
9   public boolean freccia;
    public int quantita;
11
    public vista(String testo,String descrizione, E ogg, Bitmap
        image,boolean freccia,int quantita){
13    this.testo=testo;
      this.ogg=ogg;
15    img=image;
      this.freccia=freccia;
17    this.descrizione=descrizione;
      this.quantita=quantita;
19  }
    public vista(String testo, E ogg, Bitmap image,boolean
        freccia){
21    this(testo,null,ogg,image,freccia,-1);

23  }

25 }
```

Nelle variabili di istanza ne abbiamo due di tipo String *testo* e *descrizione* che conterranno rispettivamente il testo che verrà visualizzato nella lista che poi creeremo e un eventuale descrizione che comparirà in basso sotto al testo. La variabile **ogg** è l'oggetto generico. Vedremo in seguito perché ci sarà utile. **img** è l'eventuale immagine che si accompagna al testo per avere l'interfaccia più accattivante. **freccia** è un flag che ci dirà se aggiungere o no l'immagine di una freccia nella visualizzazione dell'elemento. **quantità** è un intero in cui verrà salvata un'eventuale quantità. Ad esempio, se volessimo prendere 2 biglietti dello stesso tipo la variabile sarà impostata su 2.

Ora che abbiamo gli oggetti che definiscono la lista non ci resta che creare gli oggetti grafici a partire dall'oggetto vista. Ad assolvere questo compito sarà l'oggetto **oggettoLista<E>**.

```
1 abstract class oggettoLista<E>{
    public static String VISTA_RETURN="strBack";
3   public final vista<E> vista;
    protected Activity act;
5   protected RelativeLayout questo;
    private boolean selected;
7   public final OnClickListener click;
    public oggettoLista(vista<E> vista){
9     this.vista=vista;

11    click=getOnClickListener();
```

```
13    }
      protected RelativeLayout getRelativeLayout( ViewGroup
          overview,Activity a){
15
        LayoutInflater inflater =
            (LayoutInflater)a.getSystemService(
            Context.LAYOUT_INFLATER_SERVICE);
17      RelativeLayout view = (RelativeLayout)inflater.inflate(
            R.layout.bottonefreccia, overview, false);

19      return view;

21  }

23    public OnClickListener getOnClickListener(){
        OnClickListener r= new OnClickListener() {
25        public void onClick(View v) {
            OnClickListener(v);
27        }
        };
29      return r;
      }
31    public abstract void OnClickListener(View v);
      public void finish(){
33      act.finish();
      }
35    public void ritornaVista(vista<E> s){
        Intent intent=act.getIntent();
37      intent.putExtra(VISTA_RETURN, s);
        act.setResult(Activity.RESULT_OK,intent);
39      finish();
      }
41    protected Intent getIntent(){
        return act.getIntent();
43    }
      public View creaVista(Activity a,ViewGroup overview){
45
        act=a;
47
        RelativeLayout view = getRelativeLayout(overview, a);
49
        questo=view;
51      aggiornaVista();
        return view;
53    }
      public void aggiornaVista(){
55      RelativeLayout view=questo;
        if(vista.img==null){
```

```
57          view.removeView( view.findViewById(
                R.id.immagineElemento));
        }
59      else{
          ((ImageView)view.findViewById(R.id.immagineElemento
              )).setImageBitmap(vista.img);
61      }
        if(!vista.freccia){
63        view.removeView(view.findViewById(R.id.immagineFreccia));
        }
65      ((TextView)view.findViewById(R.id.testo)).setText(vista.testo);
        view.setClickable(true);
67      view.setOnClickListener(new OnClickListener() {
          public void onClick(View v) {
69
            OnClickListener(v);
71
          }
73      });
    }
75  private Drawable prev;
    public void select(boolean select){
77    this.selected=select;
      if(select){
79      prev=questo.getBackground();
        questo.setBackgroundColor(Color.YELLOW);
81    }
      else{
83      questo.setBackgroundDrawable(prev);

85    }
    }
87  public boolean isSelected(){
      return selected;
89  }
public LinkedList<vista<E>> getLista(){
91    Iterator<oggettoLista<E>> i= selected.iterator();
      LinkedList<vista<E>> ret= new LinkedList<vista<E>>();
93    while(i.hasNext()){
        oggettoLista<E> r=i.next();
95      ret.add(r.vista);
      }
97    return ret;

99  }
}
```

Come vediamo nel listato ho creato una classe astratta [2], lasciando come unico metodo da implementare **onClickListener**.

Lo scopo di questo oggetto è di creare una view che rappresenti graficamente l'oggetto *Vista* passato nel costruttore(metodo **creaVista(Activity, ViewGroup)**), che sia possibile selezionarlo (metodo **select()**) e che sia possibile capire se è stato selezionato (metodo **isSelected()**).

Il metodo **aggiornaVista()** consente di aggiornare la grafica della vista stessa in base all'oggetto passato.

RitornaVista() invece termina l'Activity restituendo l'oggetto *vista* passato al costruttore.

Ora che abbiamo gli oggetti per creare un elemento della lista possiamo creare la lista vera e propria.

Ogni lista selezionabile avrà le sue caratteristiche. Procedendo con ordine, vediamo come realizzare la prima, ovvero quella di scelta della città. La schermata prevede che l'utente possa scegliere, tramite tocco, una e una sola città dall'elenco.

Prima di realizzare la lista selezionabile creiamo una classe che mostri semplicemente un elenco di *oggettoLista* che verrà poi esteso per realizzare le liste di cui abbiamo bisogno.

Il codice, davvero molto semplice, è il seguente

```
public class listaView<E> extends ScrollView{
  protected oggettoLista<E>[] lista;
  protected listaView(Activity activity){
    super(activity.getApplicationContext());
  }
  public listaView( oggettoLista<E>[] listas, Activity activity)
      {
    super(activity.getApplicationContext());
    if(listas==null){
      throw new IllegalArgumentException(" La variabile listas
          non può essere null");
    }
    creaLista(listas, activity);
  }
  protected void creaLista(oggettoLista<E>[] listas, Activity
      activity){
    lista=listas;
    LinearLayout l= new
        LinearLayout(activity.getApplicationContext());
```

[2] Si dice classe astratta una classe che definisce un'interfaccia senza implementarla completamente. Ciò serve come base di partenza per generare una o più classi specializzate aventi tutte la stessa interfaccia di base. Queste potranno poi essere utilizzate indifferentemente da applicazioni che conoscono l'interfaccia base della classe astratta, senza sapere niente delle specializzate.(Wikipedia)

```
16      l.setOrientation(LinearLayout.VERTICAL);
        for(oggettoLista<E> lis:lista){
18        l.addView(lis.creaVista(activity, this));
        }
20      addView(l);
        l.getLayoutParams().height=LayoutParams.WRAP_CONTENT;
22      l.getLayoutParams().width=LayoutParams.FILL_PARENT;
    }
24  }
```

Cominciamo notando che questa classe estende ScroolView (già visto in 3.6.1), dunque ci aspettiamo un layout a scorrimento.

Come si evince dal codice, il costruttore non si limita a controllare che la variabile **listas** sia diversa da null e chiama il metodo **creaLista**. Quest'ultimo inserisce all'interno del layout le viste generate dal comando *creaVista* della classe *oggettoLista*. Le ultime due righe del metodo servono a dimensionare la lista view e hanno la stessa funzionalità dei comandi *android:layout_ width* e *android:layout_ width* nel file xml spiegati in 3.3.2.

Realizziamo ora la lista selezionabile di cui avevamo bisogno. Ovviamente la classe estenderà quella appena vista. Eccone il listato:

```
class selectableListaView<E> extends listaView<E>{
2   private int maxSelectable;
    private LinkedList<oggettoLista<E>> selected;
4   protected selectableListaView(Activity activity){
      super(activity);
6   }
    public selectableListaView(vista<E>[] viste,Activity
        activity,int maxSelecteabl) {
8     super(activity);
      selected= new LinkedList<oggettoLista<E>>();
10    this.maxSelectable=maxSelecteabl;
      lista = extracted2(viste.lenght);
12    for(int i=0;i<lista.length;i++){
        lista[i]= new oggettoLista<E>(viste[i]) {
14        public void OnClickListener(View v) {
            if(!isSelected()){
16            if(selected.size()>=maxSelectable){
                selected.remove().select(false);
18            }
              select(true);
20            selected.add(this);
            }
22          else{
              select(false);
24            selected.remove(this);
```

```
            }
          }
        };
      }
      creaLista(lista, activity);


    }
    public LinkedList<vista<E>> getLista(){
      Iterator<oggettoLista<E>> i= selected.iterator();
      LinkedList<vista<E>> ret= new LinkedList<vista<E>>();
      while(i.hasNext()){
        oggettoLista<E> r=i.next();
        ret.add(r.vista);
      }
      return ret;

    }
    public void setSelectedLista(LinkedList<vista<E>> v){
      Iterator<vista<E>> i= v.iterator();
      while(i.hasNext()){
        vista<E> n=i.next();
        oggettoLista<E> o= trovamiOggettoLista(n);
        o.select(true);
        selected.add(o);
      }
    }
    private oggettoLista<E> trovamiOggettoLista(vista<E> n) {
      for(int i=0;i<lista.length;i++){
        if(lista[i].vista.equals(n)){
          return lista[i];

        }
      }
      return null;
    }
    public selectableListaView(vista<E>[] viste,Activity
        activity,int maxSelecteabl,boolean riepilogo) {
      super(activity);
      selected= new LinkedList<oggettoLista<E>>();
      this.maxSelectable=maxSelecteabl;
      oggettoRiepilogo<E>[] lista = extracted(viste);

      for(int i=0;i<lista.length;i++){
        lista[i]= new oggettoRiepilogo<E>(viste[i]) {
          public void OnClickListener(View v) {
            if(!isSelected()){
              if(selected.size()>=maxSelectable){
                selected.remove().select(false);
```

```
            }
74          select(true);
            selected.add(this);
76        }
          else{
78          select(false);
            selected.remove(this);
80        }

82      }

84    };
    }
86  creaLista(lista, activity);
  }
88  @SuppressWarnings("unchecked")
  private oggettoRiepilogo<E>[] extracted(vista<E>[] viste) {
90    return (oggettoRiepilogo<E>[])new
        oggettoRiepilogo[viste.length];
  }
92  @SuppressWarnings("unchecked")
  private oggettoLista<E>[] extracted2(vista<E>[] viste) {
94    return (oggettoLista<E>[])new oggettoLista[viste.length];
  }
96  public LinkedList<oggettoLista<E>> getSelectedOggettoLista(){
    LinkedList<oggettoLista<E>> ret=new
        LinkedList<oggettoLista<E>>();
98    Iterator<oggettoLista<E>> i=selected.iterator();
    while(i.hasNext()){
100   ret.add(i.next());
    }
102   return ret;
  }
104 }
```

Questa classe ha due costruttori pubblici, ma cominciamo analizzando solo il primo. Per ogni oggetto di tipo *vista* passato nel costruttore vediamo che viene creato un oggetto di tipo *oggettoLista*. Siccome oggettoLista è un oggetto di tipo **astratto**, devo implementare i metodi non implementati, in questo caso: *OnClickListener*. In questa maniera se l'oggetto non è già selezionato, ad ogni click, si aggiunge alla LinkedList **selected**. Se la lista ha già raggiunto il numero massimo di elementi viene rimosso il primo per fare spazio al nuovo arrivato. Se invece viene cliccato un elemento già selezionato, questo viene deselezionato e rimosso da *selected*.

I metodi **setSelectedLista** e **getLista** servono fare in modo che si possa

salvare lo 'stato' della lista (ovvero fare in modo di ricordare cosa è stato selezionato e ripristinarlo). Questi metodi sono importanti per non perdere la selezione mentre si ruota lo schermo[3]. Il metodo **getLista** inoltre è utile per ottenere la lista degli elementi selezionati.

Il metodo **getSelectedOggettoLista()** ci restituisce una copia della lista concatenata contenente gli *oggettoLista* selezionati.

Il secondo costruttore è praticamente identico al primo, con l'unica differenza avere *oggettoLista* al posto di *oggettoRiepilogo*. Vedremo in seguito a cosa servirà.

Adesso che abbiamo i vari metodi per generare il layout a scelta multipla non ci resta che utilizzarlo: partendo da un array di String -che vedremo in seguito come ottenere- creiamo un array di *vista<String>* per ogni città impostando le variabili *nome* e *ogg*[4] con il nome della città.

A questo punto invochiamo il costruttore di *selectableListaView* passandogli come parametro l'array di viste e aggiungendo la vista al layout, si ottiene una schermata come quella della figura 5.3.

Alla pressione del tasto *continua* viene invocato il metodo *getLista* per ottenere l'oggetto *vista* in cui è scritto quale città è stata selezionata; quest'ultimo infine viene inviato all'Activity successiva (in questo caso scegliCompagniaActivity) come illustrato in 3.3.4

5.3.4 Reperimento dati

Il reperimento dati è affidato alla classe **htmlReader**. Attualmente ho sviluppato il metodo che consente di aprire pagine html leggendone il codice, contando di estrapolare informazioni servendomi dei metodi della classe String. Tuttavia, non avendo ancora un server a cui allacciarci, siamo costretti a simulare l'arrivo di dati semplicemente ritornando dei dati verosimili.

Benché la connessione a internet non venga effettivamente utilizzata, è importante inserire nell'android manifest i permessi di accesso.

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

[3] N.B.: Riusciamo a salvare la LinkedList ottenuta da metodo getSelected perché vista<E> è di tipo Serializable

[4] Che, in questo caso, è una stringa

5.3.5 Layout biglietto multiplo

Come si vede nella figura 5.1, nel caso in cui l'utente scelga il biglietto multiplo, gli verrà data la possibilità di comprare biglietti di tipo e quantità diverse. Siccome abbiamo bisogno di passare degli oggetti sottoforma di array attraverso Activity è necessario creare un nuovo oggetto di tipo **Parcelable** (come spiegato in 3.3.4)

Gli oggetti Parcelable A differenza di Serializable, Parcelable è un'interfaccia, quindi non ci limiteremo ad estendere la classe, bisognerà anche implementare tutti i metodi di interfaccia. Nella documentazione internet [Parcelable - Android Developers], si può trovare un esempio di implementazione, da cui è facile partire per poi creare la nostra.

Come realizzarla? É più complesso rispetto all'interfaccia Serialiable, infatti verrà dato direttamente a noi il compito di inviare i vari tipi di dato contenuti nell'oggetto.

Quello che andremo a realizzare sarà l'oggetto **codicePrezzo**. Per realizzarlo copiamo il codice della documentazione e sostituiamo opportunamente il nome *MyParcelable* con il nome della classe *codicePrezzo* dovunque sia richiesto. Le variabili di istanza saranno:

```
1  public String nome;
   public int codice;
3  public int prezzo;
   public int quantita;
5  public String descrizione;
```

Queste sono le variabili che dobbiamo **"**salvare**"**. I metodi da implementare per far sì che questi dati vengano inviati e ricevuti sono rispettivamente **writeToParcel** e il costruttore **private codicePrezzo(Parcel in)**. Vediamo come sono implementati:

```
1  public void writeToParcel(Parcel out, int flags) {
       out.writeString(nome);
3      out.writeInt(codice);
       out.writeInt(prezzo);
5      out.writeInt(quantita);
       out.writeString(descrizione);
7    }
     private codicePrezzo(Parcel in) {
9      nome= in.readString();
       codice= in.readInt();
11     prezzo=in.readInt();
       quantita=in.readInt();
```

```
13        descrizione=in.readString();
    }
```

L'oggetto out di tipo *Parcel* ci consente di salvare tutti i tipi di dato standard di Java e qualsiasi oggetto di tipo Parcelable. Notiamo inoltre che il metodo **write(ogg)** accetta come parametro soltanto l'oggetto da salvare (e non chiave valore come abbiamo visto fin ora), quindi è importante l'ordine in cui i metodi vanno invocati. Come si nota nel listato, la sequenza con cui vengono salvati i dati nella variabile out è la stessa di quella usata per prelevarli dalla variabile in.

Dall'oggetto codicePrezzo è facile ottenere un oggetto di tipo vista<codicePrezzo> utilizzando il seguente metodo:

```
  public vista<codicePrezzo> getVista() {
2   return new vista<codicePrezzo>(nome, descrizione, this,
        null, false, quantita);
  }
```

Grazie a questo metodo possiamo velocemente creare un layout a video per permettere all'utente di effettuare la scelta multipla. Quando creeremo la nostra *selectableListaView* dovremo invocare il secondo costruttore del metodo (vedi listato di 5.3.3) in cui al posto di creare array di *oggettoVista* dal db verranno creati *oggettoRiepilogo*.

oggettoRiepilogo oggettoRiepilogo estende la classe oggettoVista. Questo, come si può notare dalla figura 5.4, oltre allo spazio per il testo ha anche un contatore e una descrizione sotto al nome.
N.B. vedremo in seguito che questo layout ci sarà utile non solo come selezione multipla, ma anche nel riepilogo.

Ora che abbiamo creato la lista, dobbiamo dare la possibilità all'utente di selezionare uno o più biglietti dalla lista. Quello che vogliamo ottenere è una schermata come quella illustrata sotto al numero 2 in figura 5.1. Quindi oltre alla lista centrale dovremo predisporre anche i tasti + e -. Per scegliere di comprare un certo numero di biglietti, l'utente dovrà toccare sul nome del biglietto che vuole acquistare, poi toccare il tasto +. A questo punto il numero scritto a destra (inizialmente zero) dovrà crescere al pari delle volte in cui viene schiacciato il tasto + (e diminuire se l'utente tocca il tasto -). Per far si che ciò accada dobbiamo implementare in questo modo i tasti azione del pulsante:

```
public void onPiu(View view){
      LinkedList<oggettoLista<codicePrezzo>> sce =
          sel.getSelectedOggettoLista();
      if(sce.isEmpty()){
        Toast.makeText(getApplicationContext(), "Devi
            selezionare un tipo di biglietto per aumentarlo",
            Toast.LENGTH_SHORT);
        return;
      }
      oggettoLista<codicePrezzo> a= sce.remove();
      a.vista.quantita++;
      a.vista.ogg.quantita=a.vista.quantita;
      a.aggiornaVista();
    }
    public void onMeno(View view){
      LinkedList<oggettoLista<codicePrezzo>>
          sce=sel.getSelectedOggettoLista();
      if(sce.isEmpty()){
        Toast.makeText(getApplicationContext(), "Devi
            selezionare un tipo di biglietto per aumentarlo",
            Toast.LENGTH_SHORT);
        return;
      }
      oggettoLista<codicePrezzo> a= sce.remove();
      a.vista.quantita=Math.max(0, a.vista.quantita-1);
      a.vista.ogg.quantita=a.vista.quantita;
      a.aggiornaVista();
    }
```

In breve ci facciamo dare l'elemento selezionato tramite il metodo *getSelectedOggettoLista()*, verifichiamo che effettivamente almeno uno sia selezionato e, infine, aggiorniamo la variabile *quantita*. Nel codice vediamo che viene aggiornata non solo quella dell'*oggettoVista*, ma anche quella dell'oggetto *codicePrezzo* contenuto nella variabile ogg. Questo perché quando andremo a comunicare le scelte effettuate in questa schermata all'activity successiva, potremo inviare soltanto oggetti di tipo parcelable. Inoltre ricordiamo che ogni volta che cambia l'orientamento del telefono bisogna esplicitamente salvare lo stato della schermata, e anche in questo caso andremo a salvare un array di *codicePrezzo*.

Alla pressione del tasto ❛continua❜ non si dovrà fare altro che inviare un array (o un ArrayList) di codicePrezzo contenente le scelte dell'utente all'activity successiva.

Layout Abbonamento Questa schermata non è particolarmente complicata, non prevede selezioni elaborate quindi le *view* standard di android sono più che sufficienti. Nella figura 5.1 sotto il 3 infatti, notiamo che ci servono 2 selettori a tendina (**Spinner**) per il tipo di abbonamento e il tipo di sconto che si preferisce utilizzare, un selettore di data (**datePicker**) per determinare la data di inizio dell'abbonamento, e un campo di testo (**EditText**) per inserire il numero di tessera dell'abbonato.

Per gestire gli sconti utilizzerò comunque l'oggetto codicePrezzo salvando nella variabile di istanza *prezzo* la percentuale di sconto.

5.3.6 Layout Riepilogo

La schermata di riepilogo degli acquisti consente all'utente di visualizzare cosa sta per acquistare e il prezzo totale. Il layout lo si può vedere nella figura 5.1. Notiamo la forte somiglianza con la schermata di Biglietto Multiplo. Per realizzare la grafica utilizziamo gli stessi metodi già visti in precedenza con l'unica differenza che non saranno selezionabili; quindi per creare la lista useremo la classe *listaView* vista in 5.3.3. Per calcolare il prezzo totale non dovremo fare altro che sommare i prezzi dei vari biglietti moltiplicati per le rispettive quantità- nel caso in cui sia stato selezionato un tipo di sconto modificare in proporzione il risultato ottenuto- e infine mostrare il totale in fondo alla pagina.

La schermata è provvista di un tasto continua alla posizionato in fondo. Ricordiamo che se l'importo è superiore ad una certa quantità si dovrà richiedere un codice di controllo.

Richiesta codice di controllo Come notiamo subito dalla figura 5.1 (ultima foto in basso) notiamo che questo non è un layout normale, infatti non copre l'intero schermo. Quindi non creeremo una nuova Activity bensì un **Dialog** [5]. Per crearlo è necessario innanzitutto un layout in xml in cui metteremo un textView, un editText e un Button. Possiamo ora creare l'oggetto Dialog in questo modo:

```
    private Dialog creaRichiestaCodice(){
      Dialog ric= new Dialog(this);
      ric.setTitle("Richiesta Codice");
      ric.setContentView(R.layout.richiesta_codice);
      return ric;
    }
```

[5] la documentazione completa di Dialog è disponibile nella bibliografia alla voce [Documentazione di Dialog | Android developer]

A questo punto non dobbiamo fare altro che attribuire un'azione al Button del layout che abbiamo inserito e gestire il fatto che l'utente possa premere il tasto *back*:

```
private void richiediCodice(final OnCodeReceived lis){
      final Dialog d=creaRichiestaCodice();
      ((Button)d.findViewById(R.id.bottoneCodice
          )).setOnClickListener( new OnClickListener() {

      @Override
      public void onClick(View v) {
        long code=Long.parseLong(((EditText
            )d.findViewById(R.id.codice )).getText().toString());
        lis.onCodeReceived(code);
        d.setOnDismissListener(null);
        d.dismiss();
      }
    });
      d.setOnDismissListener(new OnDismissListener() {
      public void onDismiss(DialogInterface dialog) {
        lis.onDismiss();
      }
    });
      d.show();
    }
```

Le righe di codice all'interno del metodo **onDismiss(DialogInterface dialog)** vengono eseguite se l'utente preme il tasto back quando il Dialog è mostrato a video.

Nel caso in cui l'utente tocchi il tasto continua, invece, viene eseguito il metodo *onClick*. Vediamo che all'interno del metodo viene prelevato il codice all'interno dell' editText e lo viene salvato nella variabile *code*, viene chiamato il metodo onCodeReceived (che spiegheremo in seguito), viene rimosso il DismissListener dal Dialog, e infine viene chiusa la schermata.

La variabile **lis** è di tipo OnCodeReceived. Il quale è una interfaccia, creata per l'occasione composta da 2 metodi:

```
interface OnCodeReceived{
  public void onCodeReceived(long code);
  public void onDismiss();
}
```

L'interfaccia viene così implementata:

```
if(costoTotale>Const.costoMassimo){
  richiediCodice(new OnCodeReceived() {
    public void onDismiss() {
      ((Button)view).setEnabled(true);
      return;

    }
    @Override
    public void onCodeReceived(long code) {
      a.append("CODICE: "+code);
      concludi(a);
    }
  });

}
```

Questo è il pezzo di codice che richiama il Dialog. Come si può notare ho lasciato l' if in modo da sottolineare che si deve richiedere il codice soltanto quando l'acquisto supera un costo massimo. Chiamo il metodo **richiediCodice** passandogli come parametro l' implementazione dell'interfaccia. Se l'utente preme il tasto back allora il bottone continua, che avevamo disabilitato in precedenza, torna attivo. Se invece l'utente inserisce il codice verrà eseguito all'interno del metodo **onCodeReceived**. Dato che non so ancora con precisione se il pagamento avverrà via SMS o in altro modo, per il momento la parte del pagamento è sostituita da un messaggio nel LogCat ed è facilmente modificabile editando il metodo **concludi()**. L'invio di messaggio è stato comunque già trattato in 3.6.

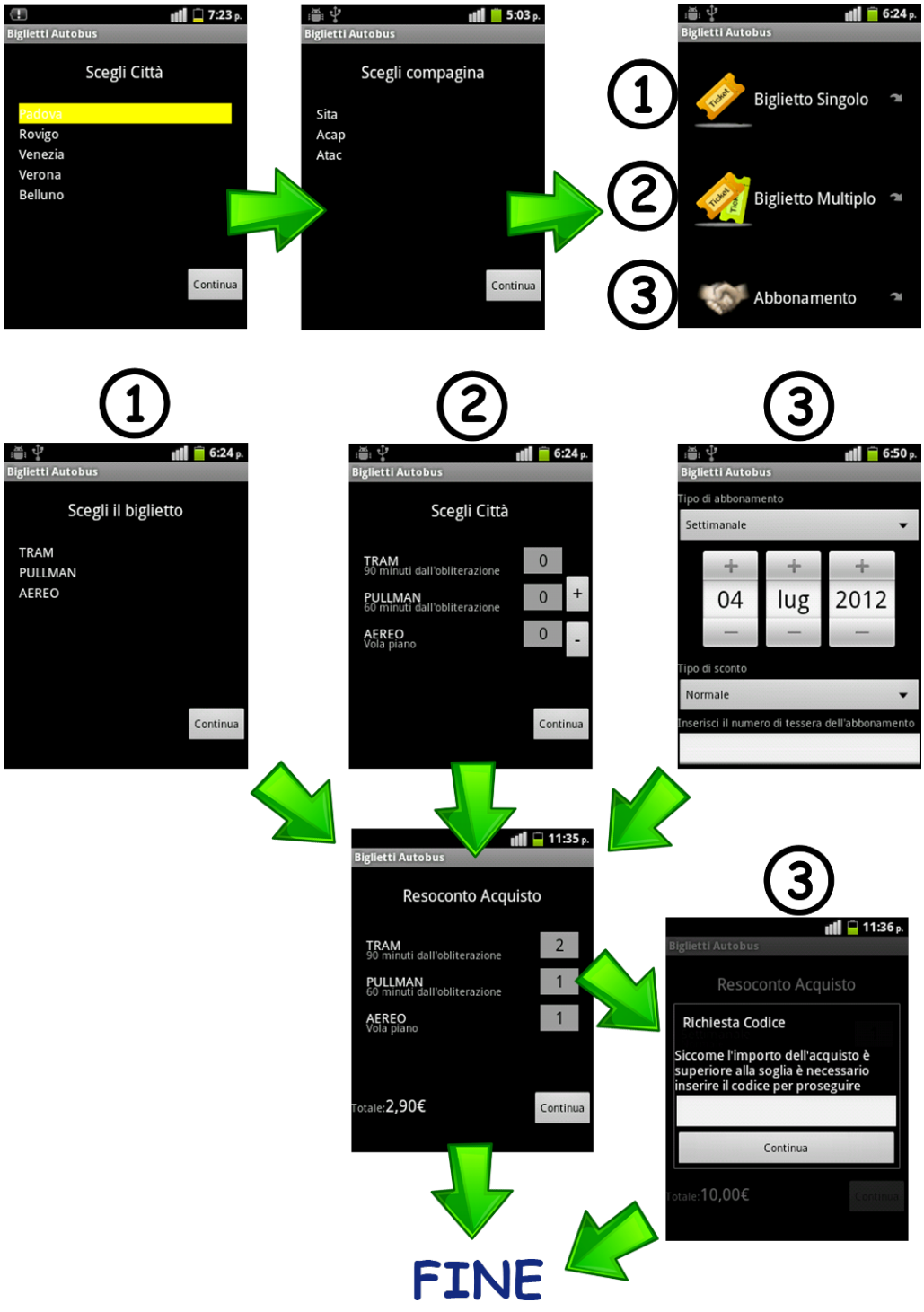

Figura 5.1: Albero Decisionale

Figura 5.2: CheckBox

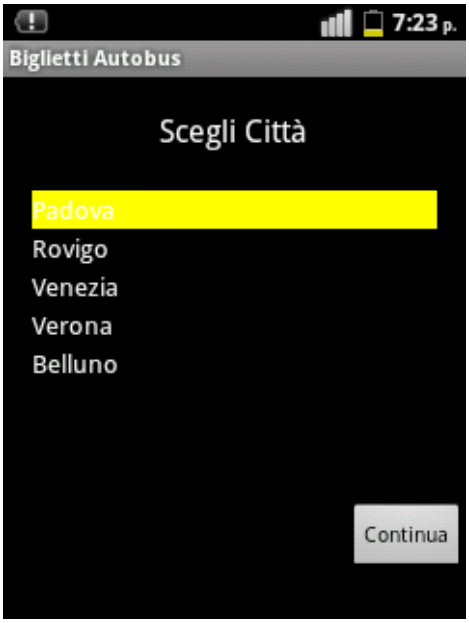

Figura 5.3: Layout seleziona città

Figura 5.4: Oggetto Riepilogo

Capitolo 6

Pubblicazione dell'applicazione

Adesso che abbiamo completato l'applicazione possiamo pubblicarla su Google Play Store (il successore dell'Android Market). Per procedere alla pubblicazione possiamo seguire la guida che Google ci mette a disposizione (disponibile nella bibliografia [Caricamento di applicazioni - Google Play]). Di cosa abbiamo bisogno?

- L'applicazione salvata in un file con estensione .apk di dimensione massima 50 MB.
- Essere registrati a Google
- Un paio di screenshots (almeno)
- Icona applicazione ad alta risoluzione

6.1 Creazione file .apk

Il file apk è un file che contiene tutti i dati della nostra applicazione, compresi quelli nella cartella res. Senza saperlo, per provare la nostra applicazione, abbiamo sempre inviato al cellulare (reale o virtuale) un file apk generato automaticamente, che, però, non è firmato digitalmente (o meglio è firmato con chiave di debug). Google Play Store non accetterà mai un file apk senza firma digitale, quindi vediamo come crearlo firmato. Con Eclipse generare il file apk firmato è semplice: clicchiamo col destro sulla nostra applicazione, e clicchiamo su esporta (vedi figura 6.1). Dall' esplora risorse selezionare la cartella **Android** e quindi la voce **Export Android Application**, infine clicchiamo su Next. Nella prossima schermata controllare che nel campo Project ci sia proprio l'applicazione che vogliamo esportare e andiamo avanti.

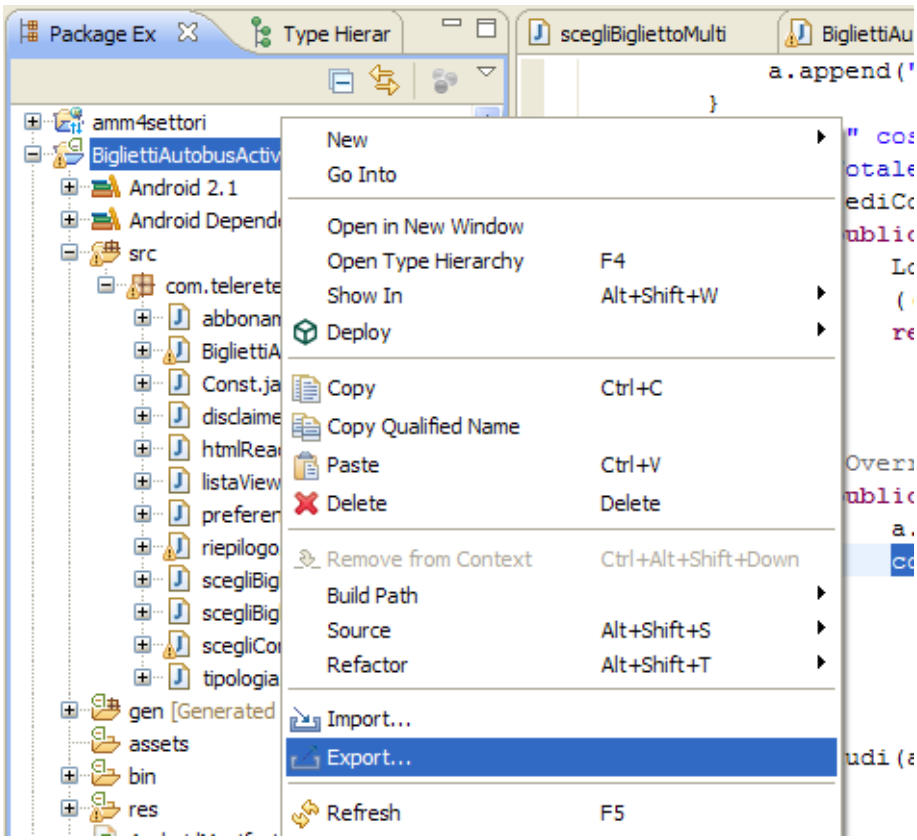

Figura 6.1: Eclipse - Esporta

A questo punto dobbiamo creare un portachiavi dove salvare tutte le nostre firme digitali (chiavi). Mettiamo il pallino su *Create new keystore*, scegliamo un percorso a nostro piacimento nel campo *location*, una password di almeno 6 caratteri, inseriamola in entrambi i campi e clicchiamo su Next. A questo punto compiliamo i campi della firma digitale inserendo un alias, una password, la validità della chiave (minimo 25 anni) e il nostro nome e cognome. Gli altri campi sono facoltativi ma è bene compilarli. Finita questa fase scegliamo il percorso dove vogliamo che il nostro file apk venga generato e clicchiamo su Finish. Abbiamo così ottenuto il nostro file .apk firmato.

Un paio di note Un'applicazione può essere sostituita da un'altra soltanto se ha la stessa chiave, altrimenti saremmo costretti a disinstallare la predente e installare la nuova. Questa restrizione è ai fini della sicurezza e serve a impedire che l'utente venga truffato con applicazioni non autentiche.

Il file generato è già in grado di essere installato su qualsiasi cellulare anche senza essere inviato su Google Play Store[1].

6.2 Invio dell'applicazione

Prima di tutto bisogna collegarsi al sito https://play.google.com/apps/publish/signup dove inserire tutti i dati richiesti, accettare il contratto di licenza, pagare 25$ (USD) e registrarsi a Google (se non si è già registrati). Infine, seguendo la guida, si inviano tutti i file indicati al punto **6** e saremo in grado di pubblicare la nostra applicazione nel Market.

[1]Affinché sia possibile installare un file apk firmato su un cellulare senza passare da Google Play Store è necessario che l'utente abbia autorizzato il cellulare a installare le applicazioni di origine sconosciuta.

Capitolo 7

Conclusioni

Come si è potuto notare nel corso della tesi, i cellulari stanno diventando sempre più dei computer a tutti gli effetti. Con i mezzi che Android ci mette a disposizione siamo in grado di creare applicazioni di ogni tipo. Da quando Android, e più in generale lo smartphone, ha preso seriamente piede nel mercato, sempre più sviluppatori si sono cimentati nella creazione di App, incoraggiati da una documentazione ben fatta e un supporto su internet veramente ottimo.

Inoltre, come abbiamo visto in 3.2.1, si può unire alla facilità di Java, l'efficienza del C/C++.

7.1 Uno sguardo al futuro

L'utilizzo degli smartphone è in rapido aumento sia in Italia che nel resto del mondo. Molte ditte (come ad esempio la Net-t by Telerete, sede dove ho svolto lo stage) stanno investendo molto in questa nuova tecnologia.

7.1.1 Cellulare VS PC

C'è da chiedersi perché cellulari e tablet stiano avendo una diffusione così ampia dato che il computer può svolgere le stesse funzioni.

Il cellulare è sicuramente più portabile di qualsiasi PC. Molte delle applicazioni per smartphone, infatti, sono pensate per un utilizzo in movimento (vedi navigatori, orari dei treni ecc..).

In secondo luogo abbiamo la semplicità di utilizzo. A differenza del computer, il cellulare permette di fare le operazioni base in maniera semplice ed efficace, anche grazie al touchscreen. Toccare lo schermo, anziché utilizzare i tasti, lo rende molto più intuitivo oltre che più accattivante.

Altra nota importante sta nella facilità con cui è possibile installare e rimuovere programmi. Sia in Android che in iOS[1], infatti, si possono reperire facilmente grazie a un negozio online (Google play per Android e App store per iOS). Rimuoverli è altrettanto facile e sicuro. Basta trascinare l'applicazione nel cestino, oppure dalla lista programmi lanciare la rimozione. A differenza del pc, la rimozione del cellulare non lascia tracce residue all'interno del cellulare (come operazioni di background, file sparsi, librerie ecc..)[2].

Ad aumentare la facilità d'uso degli smartphone sarà l'assistente vocale. Ovvero un applicazione che consente all'utente di impartire ordini al cellulare e ricevere informazioni, semplicemente conversando col cellulare, ampliando così sempre più il range di persone in grado di utilizzarlo correttamente.

Un cellulare, oltretutto, è, in media, meno costoso di un pc ed è dotato di tutte le periferiche necessarie al suo funzionamento.

Ovviamente non potrà mai sostituire il PC, però l'utente medio non ha bisogno di potenza di calcolo elevata, tantomeno di programmi che facciano qualcosa di complesso.

7.2 Potenzialità

Nel corso della tesi abbiamo visto come sia possibile ricevere informazioni da videocamera, elaborarle a piacimento, e interagire con il mondo tramite SMS e internet. Non abbiamo sfruttato gli strumenti di geo-localizzazione (GPS[3]), la bussola interna, gli accelerometri ecc...

Sfruttando adeguatamente tutti gli strumenti di cui sono dotati smartphone, si possono creare applicazioni in grado di rivoluzionare il modo di fare informazione. In futuro potrebbero sparire code agli sportelli della posta, alle biglietterie dei cinema, agli sportelli automatici. Sempre meno persone avranno bisogno di chiedere indicazioni come raggiungere un determinato luogo, passando sempre per la via meno trafficata. Sarà sempre più facile ed intuitivo creare ritrovi, condividere la propria posizione ecc... Il tutto con una facilità estrema e sempre a portata di dito.

[1] iOS (iPhone Operative System) è il sistema operativo sviluppato da Apple per iPhone, iPod touch e iPad.

[2] In realtà i dati di configurazione di ogni applicazione rimangono salvati all'interno del cellulare anche dopo la rimozione, ma la quantità di memoria che occupano è veramente piccola e non influisce sulle prestazioni di sistema

[3] Global Positioning System

Bibliografia

[Android, Guida per lo sviluppatore] Titolo: Android, Guida per lo sviluppatore. Autore: Massimo Carli, Casa Editrice: Apogeo.

[Parcelable - Android Developers] Documentazione internet dell'interfaccia Parcelable, Disponibile all'indirizzo: http://developer.android.com/reference/android/os/Parcelable.html

[Marakana - Using NDK to Call C code from Android Apps] Guida per usare l'NDK per chiamare funzioni c in applicazioni Android, disponibile all'indirizzo: http://marakana.com/forums/android/examples/49.html, [4]

[Using Tesseract Tools for Android to Create a Basic OCR App] Usare tesseract tools per android per creare una semplice applicazione OCR per Android. La guida è disponibile a questo link: http://rmtheis.wordpress.com/2011/08/06/using-tesseract-tools-for-android-to-create-a-basic-ocr-app/

[Log - Wikipedia] Descrizione di Log, disponibile all'indirizzo: http://it.wikipedia.org/wiki/Log

[Cat - Wikipedia] Descrizione di Cat, disponibile all'indirizzo: http://it.wikipedia.org/wiki/Cat_(Unix)

[Documentazione di Dialog | Android developer] La documentazione di Dialog è disponibile all'indirizzo: http://developer.android.com/reference/android/app/Dialog.html

[4]Se non dovesse funzionare il programma della guida dovete modificare la riga return (*env)->NewStringUTF(env, Hello World!); togliendo l'asterisco a env e scriverla in questo modo (env)->NewStringUTF(Hello World!); Se non riuscite a trovare javah dovete installarlo a parte

[Caricamento di applicazioni - Google Play] La guida completa su come inviare un applicazione a google è disponibile all'indirizzo http://support.google.com/googleplay/android-developer/bin/answer.py?hl=it&answer=113469

[Tesseract - Wikipedia] La descrizione completa di tesseract è disponibile a questo indirizzo: http://it.wikipedia.org/wiki/Tesseract_

[Java Native Interface - Wikipedia] Java Native Interface From Wikipedia, the free encyclopedia. Link originale: http://en.wikipedia.org/wiki/Java_Native_Interface

[ZXing (Zebra Crossing)] Sito ufficiale di ZXing disponibile all'indirizzo: http://code.google.com/p/zxing/