CALIFORNIA STATE UNIVERSITY SAN BERNARDINO WiN GPS

Francisco A. Ron

CALIFORNIA STATE UNIVERSITY

WIN GPS

A Project

Presented to the

Faculty of

California State University,

San Bernardino

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

in

Computer Science

---

by

Francisco Anibal Ron

December 2015

CALIFORNIA STATE UNIVERSITY

WIN GPS

A Project

Presented to the

Faculty of

California State University,

San Bernardino

---

by

Francisco Anibal Ron

December 2015

Approved by:

David Turner, Advisor, Computer Science	Date
Kerstin Voigt	
Ernesto Gomez	

ABSTRACT

The objective of this masters project is to develop a working application for Android devices. This is an application intended to be used by CSUSB. It has its own database, which has information about most of the facilities on campus. There are many GPS applications on the market, however I chose to design and implement WiN GPS, short for Walking GPS, because it will allow the possibility of a personalized GPS for the school and for users should they choose to do so.

In order to develop WiN GPS it was necessary to research the available tools, and to become familiar with the ones that were selected. These tools such as map application providers, i.e. Google-maps, integrated development environments, database managers, software development kits, and mobile device emulators were analyzed and compared.

Once the tools were selected, it was necessary to study, to become familiar with, and to learn how to use them. Finally an app is developed and its main functions/code will be explained. This masters project will allow potentially Android developers to evaluate possible barriers, such as price and limitations of map application providers, so they can make an informed decision.

Keywords: GoogleMaps, Android, app, SQLite3, Eclipse, Java, Database, GPS, mobile development

ACKNOWLEDGEMENTS

Upon reflecting on the journey traveled to put this masters project together, I have come to realize that without the assistance of the following persons there would not be such a project.

My committee members Dr. Turner, Dr. Voigt, and Dr. Gomez. I want to thank especially Dr. David Turner, my project advisor: he has been very helpful, not only through the development of this project, but also on every class I took with him. His knowledge in up-to-date technologies has been helpful. I also want to thank Dr. Schubert. It was an honor to be one of his students. Finally, it's my pleasure to mention Mrs. Monica Latimer always committed to help students.

TABLE OF CONTENTS

LIST OF FIGURES

1. PROJECT SUMMARY

Mobile applications are quite popular nowadays. It's possible to find apps pretty much for anything, and yet, there is still so much to be done. One particular area of mobile apps deals with maps and their management. Multiple applications have been designed using maps, such as GPS, transit information, bike routes, and so on[3]. This masters project's purpose is to create a walking navigation GPS app for the CSUSB campus.

It must be noted also that mobile devices come in different shapes, brands, and operating systems. This particular project will focus on mobile devices using the Android Operating System.

1.1 Purpose

This is a masters degree project, therefore the main purpose is to deliver an application according to the masters program level of knowledge. The application might be used by CSUSB visitors unfamiliar with the facilities, or students that are familiar with them, but they want to point out their 'preferred' places on a map. Another purpose, not less important, is to provide a document that might be useful to students interested in Android applications.

1.2 Scope

The scope of this project is limited to maps on Android devices. Several tools are available toward this kind of development, therefore I'll mention the most popular and I'll explain the reasons I chose, or not, to use some of them.

Subjects such as publication, usage, distribution, or modifications of this app are out of scope of this report. Also out of the scope is image designing.

1.3 Development Tools

This project deals with several tools:

- Programming language: Java version 1 (1.7.0 51) for Android
- Programming tools: Java JDK, Java SDK, Android Debug Bridge
- Database: Sqlite3
- IDE: Eclipse
- Maps: Google maps, Google play services, Google API client
- Graphic design: Photoshop CS4

1.4 Definitions, Acronyms, and Abbreviations

- ANDROID: Operating System. Open-source operating system for mobile devices such as cell phones, and tablets. Android is Linux based.
- API: It stands for Application Programming Interface. This is a particular set of previously written classes and interfaces for the developer to use. Example: a calendar, Google Maps, etc.

- WiN-GPS: The CSUSB walking navigation application (deliverable from this project).
- IDE: It stands for Integrated Development Environment. It is a software application that provides the appropriate environment to develop a software application i.e. Eclipse for Android, JavaBeans for java, etc.
- MOBILE APPLICATION: It refers to any software or application that runs on current mobile phone and/or smart technology such as tablets.
- SDK: It stands for Software Development Kit. A library of previously written functions that aids a programmer to re-use code[3].
- TAP/TAPPING/PRESSING/CLICK/DOUBLE-CLICK/LONG-CLICK: Ways of input used on a mobile device through a touch screen interface. Tapping is quite equivalent to a click on a standard desktop computer.
- ADB: Android Debug Bridge is a tool that allows commands to be sent from a terminal to an Android device Zigurd, Laird (2010)[3].
- GEO CODING: worldwide location translator.
- WALKING: A path from point A to point B with everything in between.
- POI: points of interest.
- PHOTOSHOP: An image editing software developed by Adobe Corporation.

2. OVERVIEW OF DEVELOPMENT TOOLS

2.1 Maps

It's long gone the time when static GPS devices were the only way to manage maps. Most map software based applications have been moved to smart devices. This project deals with Android devices, so I'll mention some of the main map applications available for them.

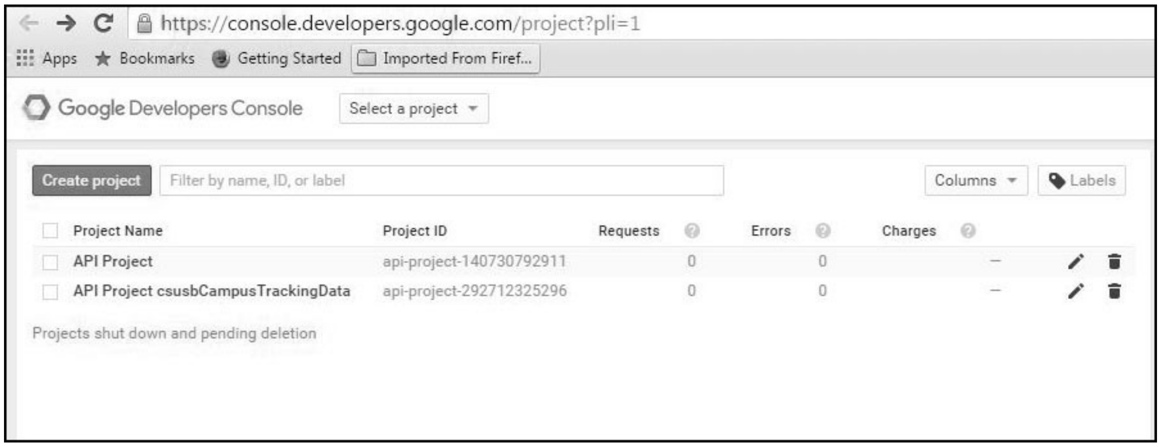
Map Quest: It has been in the industry for a while now. They have the background expertise working with map software development. I personally used it on the web even before I knew about the existence of Google maps. Map Quest offers a developer account for free. The only prerequisite is to create an account with them. The basic services are offered for free for the first 15,000 transactions per month. Technical support is restricted to paid accounts, and developer forums are not quite common on the Internet. All in all, Map Quest is definitely a good choice to be considered when it comes to map development[7].

Yahoo Boss Geo Services: It offers location and search services worldwide. It is a paid service, which begins at six dollars for up to 10,000 queries per day. In the present case a paid service is a limitation[9].

Google Maps: This is nowadays the most known web mapping service for developers. Among other benefits, it offers unlimited free usage of maps through the Android APIs[6]. There's plenty of technical information on the Internet, plus there are lots of independent developers sharing their experiences on websites such as Stack Overflow; where I happened to post most of my novice questions at the beginning of this development[2].

This development does not contemplate a budget for using a mapping service, therefore Yahoo maps in not an option. On the other hand, Map Quest has limited technical service, so in conclusion Google Maps is the adequate choice for this particular project.

In order to develop a google map based app, we must acquire a Google Maps API key. For that purpose it is necessary to have a google account. A shortened description on how to generate an API key is as follows. Sign in with your account and create a project in the google console developers. Figure 2.1 shows the website address as well as the project created for WiN GPS[6].


https://console.developers.google.com/project?pli=1
Apps
Bookmarks
Getting Started
Imported From Firef...
Google Developers Console
Select a project
Create project
Filter by name, ID, or label
Columns
Labels

Project Name	Project ID	Requests	Errors	Charges
API Project	api-project-140730792911	0	0	–
API Project csusbCampusTrackingData	api-project-292712325296	0	0	–

Projects shut down and pending deletion

Fig. 2.1: Project Defined in Google Console

Select the project; on the next screen select "use Google API"[6]. See Figure 2.2:

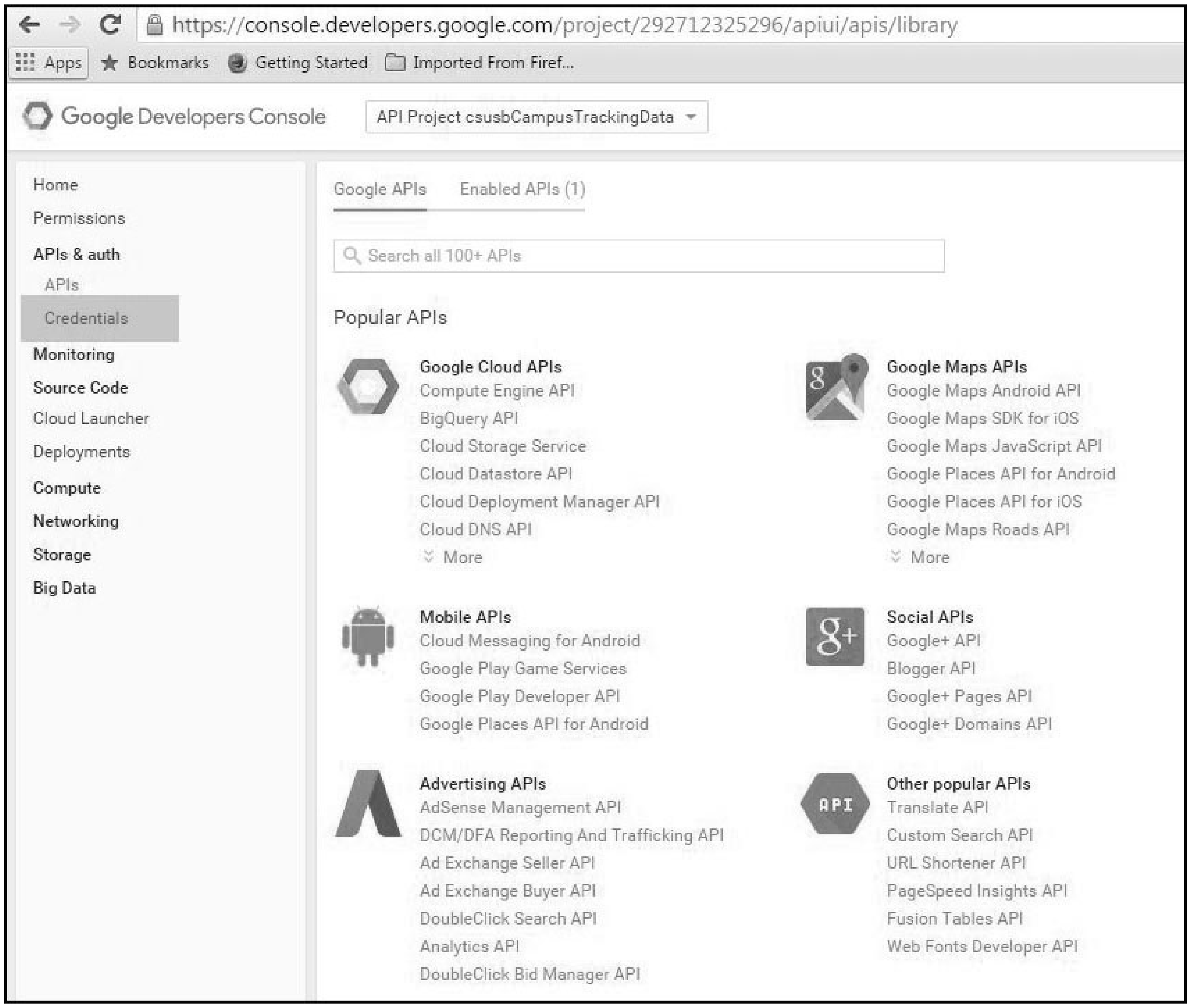
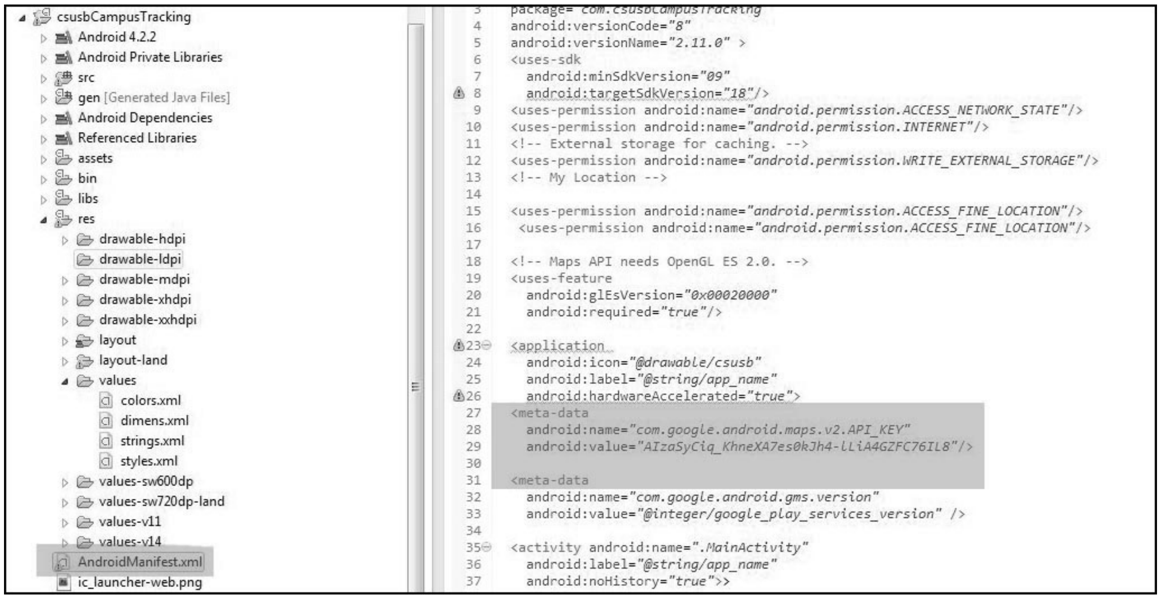


Fig. 2.2: Credentials for the Project

Follow the instructions and an API key will be generated. This API key must be part of the AndroidManifest.xml file of the project or it won't even start. This is part of the AndroidManifest. xml where the API key is present[4]. See Figure 2.3



```xml
package="com.csusbCampusTracking"
android:versionCode="8"
android:versionName="2.11.0" >
<uses-sdk
  android:minSdkVersion="09"
  android:targetSdkVersion="18"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.INTERNET"/>
<!-- External storage for caching. -->
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<!-- My Location -->

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

<!-- Maps API needs OpenGL ES 2.0. -->
<uses-feature
  android:glEsVersion="0x00020000"
  android:required="true"/>

<application
  android:icon="@drawable/csusb"
  android:label="@string/app_name"
  android:hardwareAccelerated="true">
<meta-data
  android:name="com.google.android.maps.v2.API_KEY"
  android:value="AIzaSyCiq_KhneXA7es0kJh4-lLiA4GZFC76IL8"/>

<meta-data
  android:name="com.google.android.gms.version"
  android:value="@integer/google_play_services_version" />

<activity android:name=".MainActivity"
  android:label="@string/app_name"
  android:noHistory="true">>
```

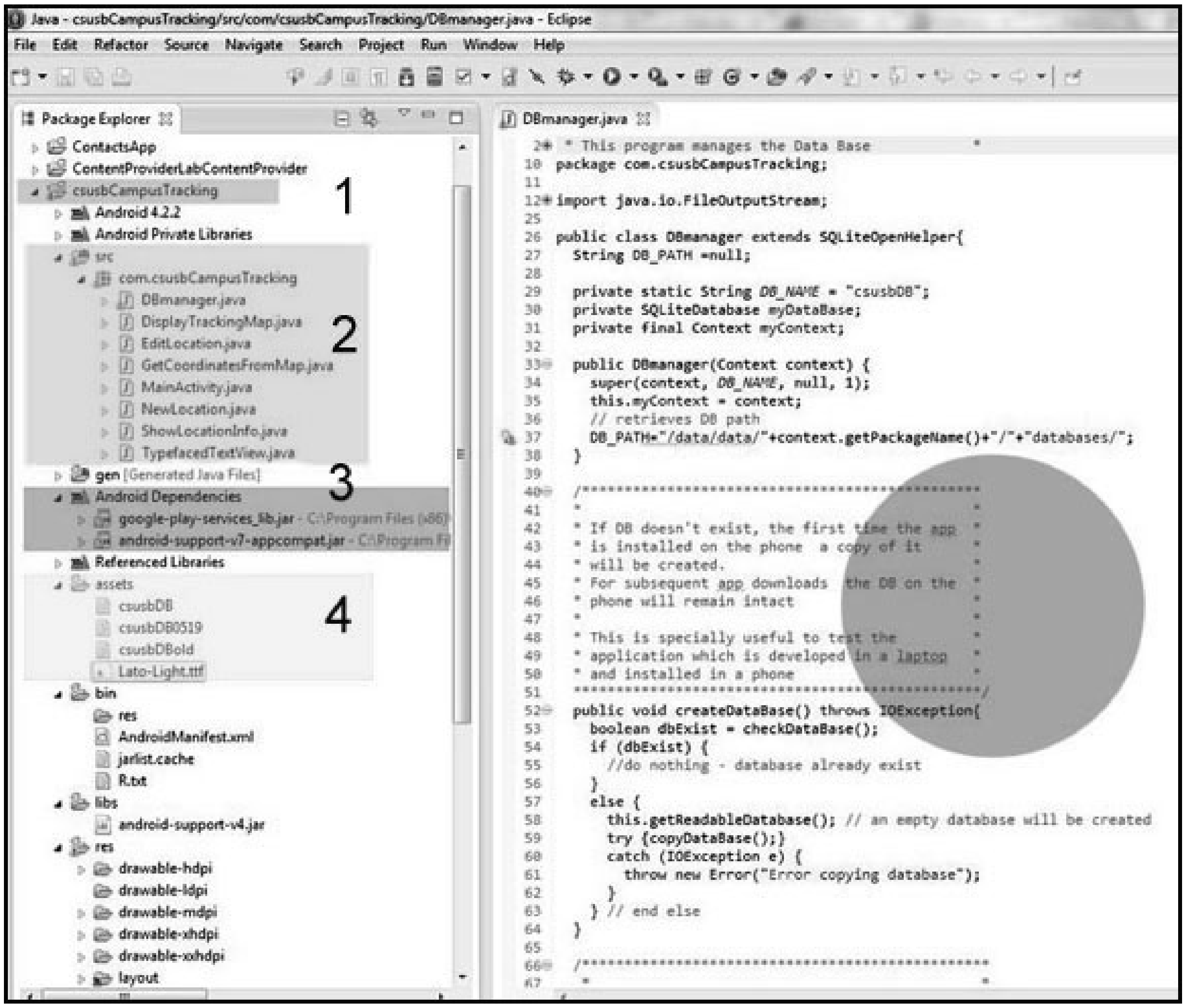
Fig. 2.3: API Key in the Manifest

More detailed instructions on how to generate an API key can be found at the following site: https://developers.google.com/maps/documentation/android-api/config

2.2 IDE

By the time this project started two IDE were available for Android development: Eclipse and Android Studio. Eclipse was released in the industry before Android Studio. I have used Eclipse for a couple of projects so I gained some expertise on it. That is the only reason I chose Eclipse over Android Studio on this particular task[4].

ECLIPSE the IDE of choice for this task may be used for different kinds of development i.e. Java, or Android Development. Using the figures 2.4 and 2.5, I will explain the components for an Android development[3].


Java - csusbCampusTracking/src/com/csusbCampusTracking/DBmanager.java - Eclipse
File Edit Refactor Source Navigate Search Project Run Window Help
Package Explorer
DBmanager.java
1
2
3
4

```
ContactsApp
ContentProviderLabContentProvider
csusbCampusTracking
  Android 4.2.2
  Android Private Libraries
  src
    com.csusbCampusTracking
      DBmanager.java
      DisplayTrackingMap.java
      EditLocation.java
      GetCoordinatesFromMap.java
      MainActivity.java
      NewLocation.java
      ShowLocationInfo.java
      TypefacedTextView.java
  gen [Generated Java Files]
  Android Dependencies
    google-play-services_lib.jar - C:\Program Files (x86)
    android-support-v7-appcompat.jar - C:\Program Fi
  Referenced Libraries
  assets
    csusbDB
    csusbDB0519
    csusbDBold
    Lato-Light.ttf
  bin
    res
    AndroidManifest.xml
    jarlist.cache
    R.txt
  libs
    android-support-v4.jar
  res
    drawable-hdpi
    drawable-ldpi
    drawable-mdpi
    drawable-xhdpi
    drawable-xxhdpi
    layout
```

```java
 * This program manages the Data Base                      *
package com.csusbCampusTracking;

import java.io.FileOutputStream;

public class DBmanager extends SQLiteOpenHelper{
  String DB_PATH =null;

  private static String DB_NAME = "csusbDB";
  private SQLiteDatabase myDataBase;
  private final Context myContext;

  public DBmanager(Context context) {
    super(context, DB_NAME, null, 1);
    this.myContext = context;
    // retrieves DB path
    DB_PATH="/data/data/"+context.getPackageName()+"/"+"databases/";
  }

  /***********************************************************
  *                                                          *
  * If DB doesn't exist, the first time the app              *
  * is installed on the phone  a copy of it                  *
  * will be created.                                         *
  * For subsequent app downloads  the DB on the              *
  * phone will remain intact                                 *
  *                                                          *
  * This is specially useful to test the                     *
  * application which is developed in a laptop               *
  * and installed in a phone                                 *
  ***********************************************************/
  public void createDataBase() throws IOException{
    boolean dbExist = checkDataBase();
    if (dbExist) {
      //do nothing - database already exist
    }
    else {
      this.getReadableDatabase(); // an empty database will be created
      try {copyDataBase();}
      catch (IOException e) {
        throw new Error("Error copying database");
      }
    } // end else
  }

  /***********************************************************
   *                                                         *
```

Fig. 2.4: Eclipse Features I

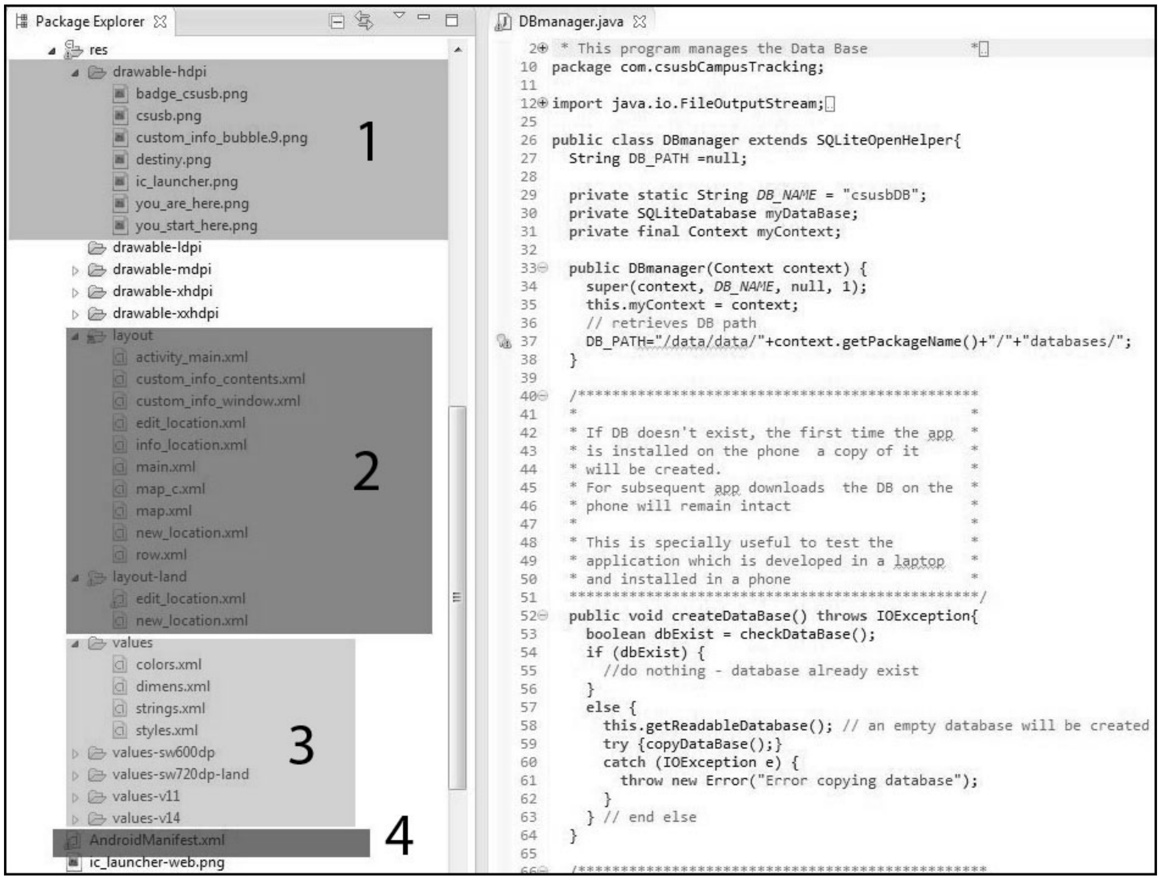
As follows:

1: name of the project

2: Java code source

3: libraries imported for this project

4: database, database backups, and imported font



```
res
  drawable-hdpi
    badge_csusb.png
    csusb.png
    custom_info_bubble.9.png
    destiny.png
    ic_launcher.png
    you_are_here.png
    you_start_here.png
  drawable-ldpi
  drawable-mdpi
  drawable-xhdpi
  drawable-xxhdpi
  layout
    activity_main.xml
    custom_info_contents.xml
    custom_info_window.xml
    edit_location.xml
    info_location.xml
    main.xml
    map_c.xml
    map.xml
    new_location.xml
    row.xml
  layout-land
    edit_location.xml
    new_location.xml
  values
    colors.xml
    dimens.xml
    strings.xml
    styles.xml
  values-sw600dp
  values-sw720dp-land
  values-v11
  values-v14
AndroidManifest.xml
ic_launcher-web.png
```

```java
 * This program manages the Data Base
package com.csusbCampusTracking;

import java.io.FileOutputStream;

public class DBmanager extends SQLiteOpenHelper{
  String DB_PATH =null;

  private static String DB_NAME = "csusbDB";
  private SQLiteDatabase myDataBase;
  private final Context myContext;

  public DBmanager(Context context) {
    super(context, DB_NAME, null, 1);
    this.myContext = context;
    // retrieves DB path
    DB_PATH="/data/data/"+context.getPackageName()+"/"+"databases/";
  }

  /***********************************************
  *                                              *
  * If DB doesn't exist, the first time the app  *
  * is installed on the phone  a copy of it      *
  * will be created.                             *
  * For subsequent app downloads  the DB on the  *
  * phone will remain intact                     *
  *                                              *
  * This is specially useful to test the         *
  * application which is developed in a laptop   *
  * and installed in a phone                     *
  ***********************************************/
  public void createDataBase() throws IOException{
    boolean dbExist = checkDataBase();
    if (dbExist) {
      //do nothing - database already exist
    }
    else {
      this.getReadableDatabase(); // an empty database will be created
      try {copyDataBase();}
      catch (IOException e) {
        throw new Error("Error copying database");
      }
    } // end else
  }

  /************************************************
```

Fig. 2.5: Eclipse Features II

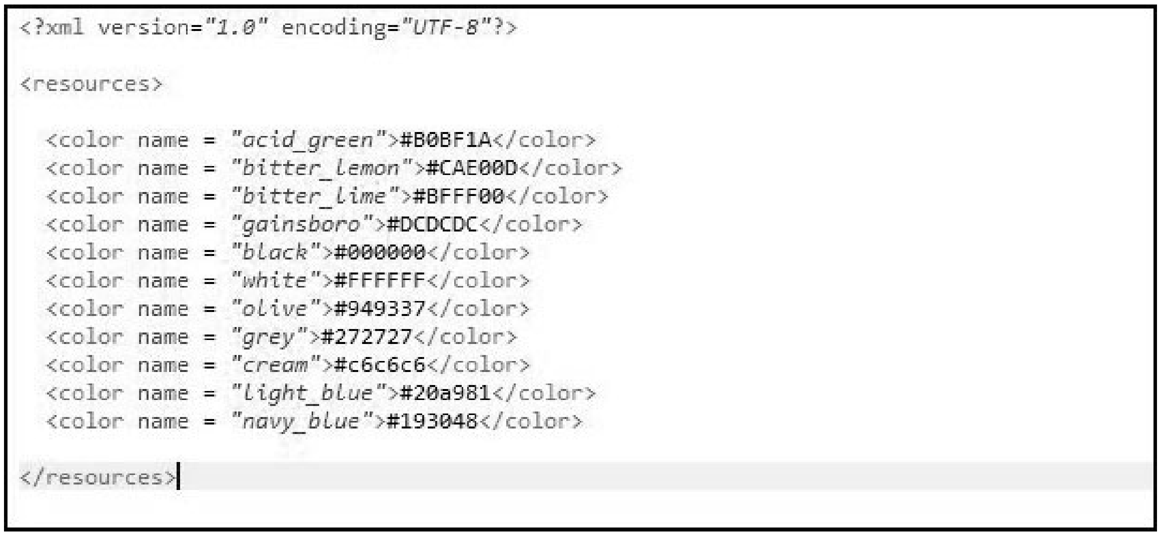
Following:
On 1: images used for the project. PNG format keeps the accuracy of images in the smaller size if they don't have motion incorporated, which is the case for the present project. All the images for this project were manipulated with Photo-shop. For an Android project it is possible to use high, low, middle, extra high, and extra extra high density files. There is a tab for each of them so the operating system might automatically find them.

On 2: Layouts, a layout is nothing but a screen that a program will use to interact with the user. It is possible, sometimes it becomes mandatory though, to have more than one layout for the same program. When the user shifts a device from vertical to horizontal the operating system will handle to shift the information from a vertical layout to an horizontal one. If there's a XML file in the layout-land it will use it, if not it will just crop some items so they fit in the current screen position: either vertical or landscape. A layout that is used by the same program must have the same name either vertical or landscape.

On 3: Values. All the system values to be used.

On 4: the package's definition file. This is the first file the application will look for before anything else.

Values, in light blue, might be defined globally, or they can be determined every time they are gonna be used in a program. For example, instead of typing an hexadecimal number every time we need an item to be black, we can simply define it as a value and use it by its name when necessary. The following figure 2.6 depicts this process:



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

<resources>

  <color name = "acid_green">#B0BF1A</color>
  <color name = "bitter_lemon">#CAE00D</color>
  <color name = "bitter_lime">#BFFF00</color>
  <color name = "gainsboro">#DCDCDC</color>
  <color name = "black">#000000</color>
  <color name = "white">#FFFFFF</color>
  <color name = "olive">#949337</color>
  <color name = "grey">#272727</color>
  <color name = "cream">#c6c6c6</color>
  <color name = "light_blue">#20a981</color>
  <color name = "navy_blue">#193048</color>

</resources>
```

Fig. 2.6: Values Definition

2.3 Java

Java is a programming language, its virtues and capacities are beyond the study of this project. Eclipse, the IDE used for this project is Java based. The supporting libraries for eclipse are Java based as well. Naturally Java is the programming language of choice for this project. Java and Java for Android have quite similar capabilities, however there are some dissimilarities basically due to the devices that will host the applications[3]. A desktop, for instance, won't have some of the problems a mobile device might have, such as: a discharged battery , an incoming call interrupting a process, etc. Besides, the hardware capabilities a desktop has are quite different from the ones of a mobile device. For instance: memory restrictions plays an important role in mobile development. Due to these differences a Java program for Android has a different life cycle than a Java source for a desktop has. The following graphic was presented in the course: "Programming Mobile Services for Android Handheld Systems". It is a self explanatory Android application life cycle. [8]

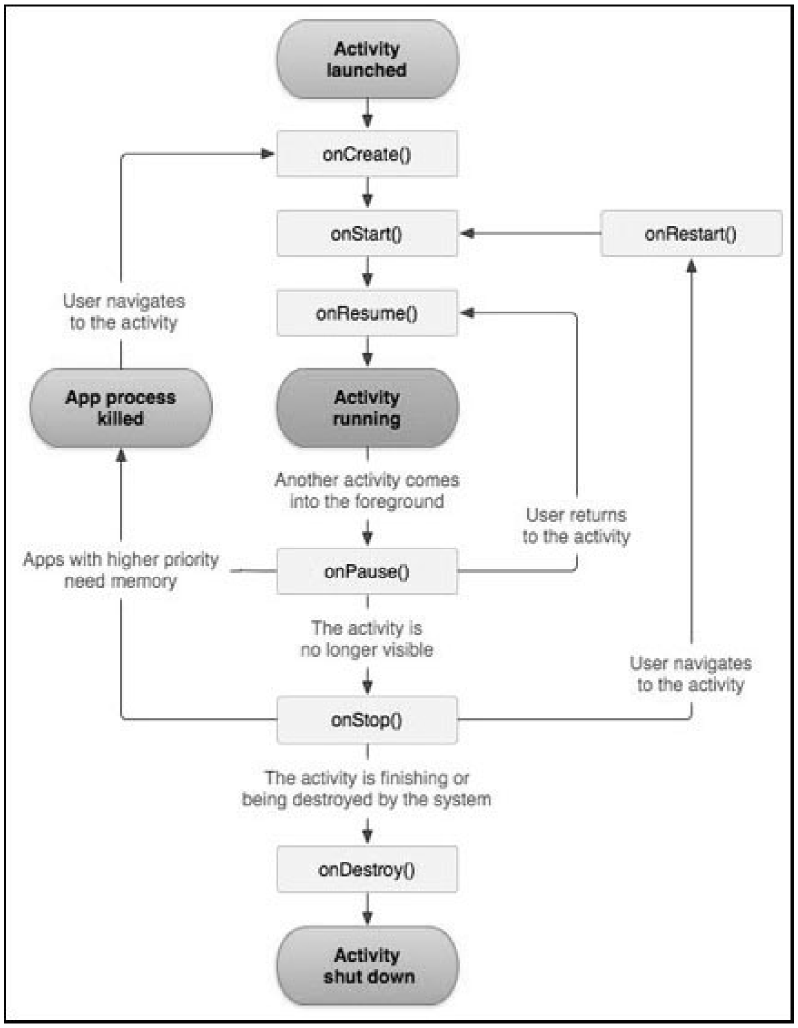

Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

Fig. 2.7: Android Application Life Cycle[8]

2.4 SDK and Google Packages

The Software Development Kit has to be installed along with the IDE. However, there are some additional packages that must be installed, such as Google APIs, and Google Play Services. These packages can be obtained free of charge from the Google developers website: https://developers.google.com/maps/documentation/android-api/config

These are typical installations that won't present a problem if the instructions are followed properly[6].

2.5 ADB

Android Debug Bridge is the terminal for android. It becomes handy when a malfunction occurs with the project. It happens sometimes that a session ends abnormally for any reason. For instance, if things are not working properly on the next connection, the command 'adb kill-server' from a terminal will fix the issue[3].

2.6 Emulators

An emulator is a piece of software that allows us to test an app if we don't have an actual device or several devices for that mater. It is also useful to emulate the actual device behavior, such as a low battery[8]. Eclipse comes with its own emulator system, however it takes quite a while to load an app. This makes it cumbersome to test an application in the process of development. In order to overcome this problem, Blue Stacks is a solution.

Blue Stacks Is an Android emulator service. It's fast and reliable. Once it's installed it's just a matter to locate the file applicationName.apk double click on it and the emulator will open it[5]. Blue stacks will open the application in a matter of seconds compared to minutes that the Android emulator takes.

Other than emulators, it is always safe to test an application over the actual device it's supposed to work on. For this project I used Blue Stacks to test the app on a emulated tablet. I also tested the app on a phone LG Stylus with OS Android 5.1.1

2.7 Database

The native database for Android applications is SQLITE3. It is possible to work with other databases, however, according to its specifications a SQLITE3 database would easily manage up to 10,000 records. This application is intended to manage no more than 500 locations at a time, and that's quite optimistic, so SQLITE3 will handle with no problem WiN GPS database[8].

3. WIN-GPS MECHANICS

3.1 Summary

WiN-GPS is a personalized GPS Google maps based application. It comes with an installed database, in other words: it's a Google map with an address book incorporated that will help you to get to that place you still don't know the address to, or the location of it is not in a regular map.

3.2 Screens

WiN-GPS uses touch screen inputs, and GUI buttons to navigate between the different app screens. This is shown in the next figure 3.1:

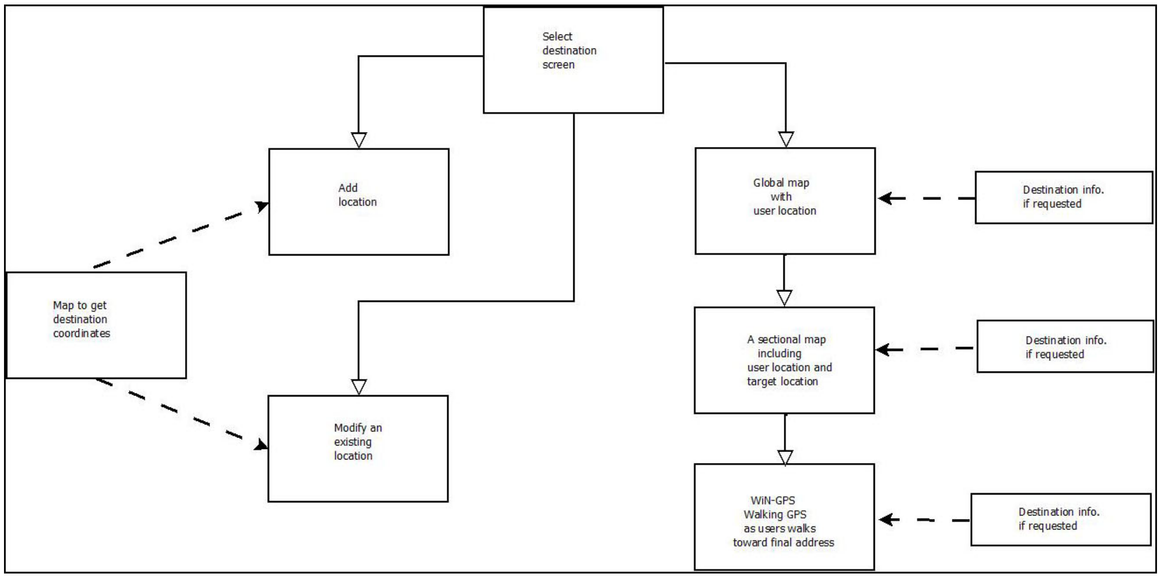
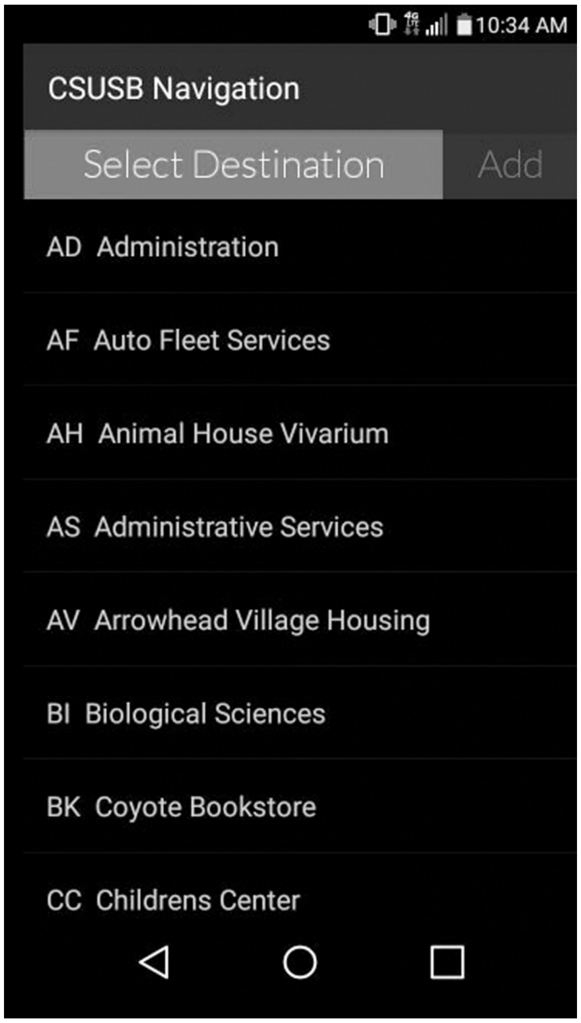

Select destination screen
Add location
Modify an existing location
Map to get destination coordinates
Global map with user location
A sectional map including user location and target location
WiN-GPS Walking GPS as users walks toward final address
Destination info. if requested
Destination info. if requested
Destination info. if requested

Fig. 3.1: Screen Flow Diagram

3.3 Walking Navigation

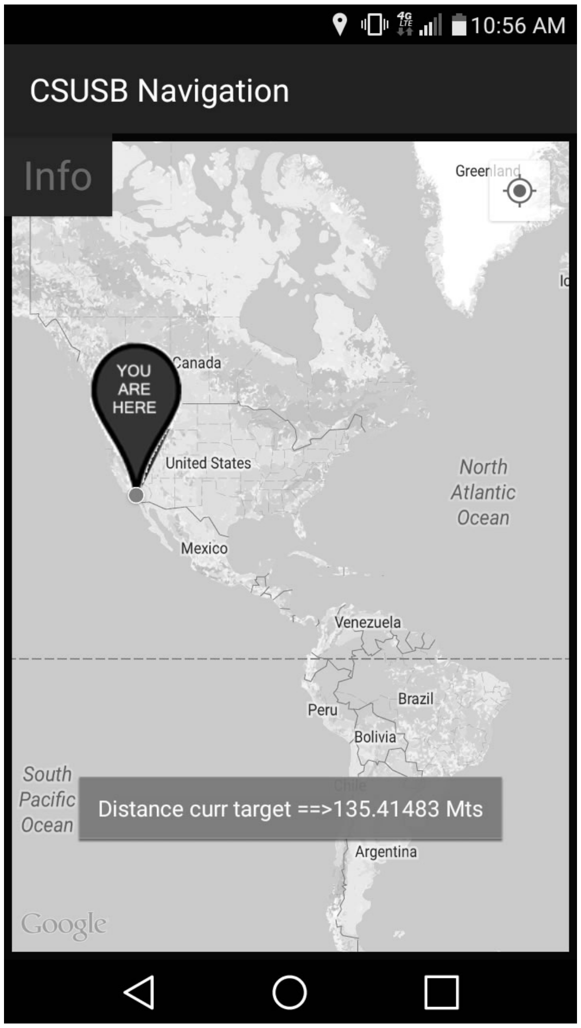
The initial screen lists all predetermined locations. The list consists of location code and description. From this screen the user might start a walking GPS, add a new location, or modify a location. See Figure 3.2.



AD Administration
AF Auto Fleet Services
AH Animal House Vivarium
AS Administrative Services
AV Arrowhead Village Housing
BI Biological Sciences
BK Coyote Bookstore
CC Childrens Center

Fig. 3.2: Initial Screen

By tapping on a location: the walking GPS will start. The following screen will show a map on a global scale with both user's current location and target destination. On this screen the distance between user location and target destination is shown in meters. However, it is necessary to zoom on the map in order to get a close up of the area. Figure 3.3 shows the global view of the map.



Distance curr target ==>135.41483 Mts

Fig. 3.3: Walking GPS Global Screen

Now, this figure shows a street level map where user will clearly see their starting location ('you started here mark'). A blue dot will show the user's current position. A red line will show their walking along the campus. Finally a red arrow indicates the selected destination. All the tools that are necessary to help the user to get to their destination are available on this screen. See Figure 3.4.

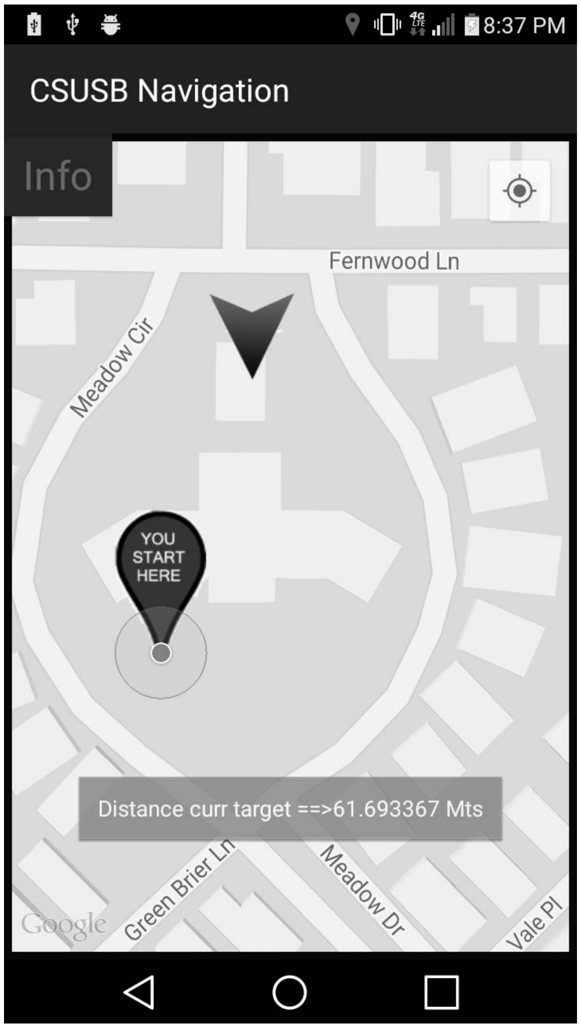


Fig. 3.4: Walking GPS Initial Screen

The following figures show a complete walking GPS, from the user's starting point to their target destination. This print screens were generated while walking from point A to point B in a test. See Figures 3.5 and 3.6.

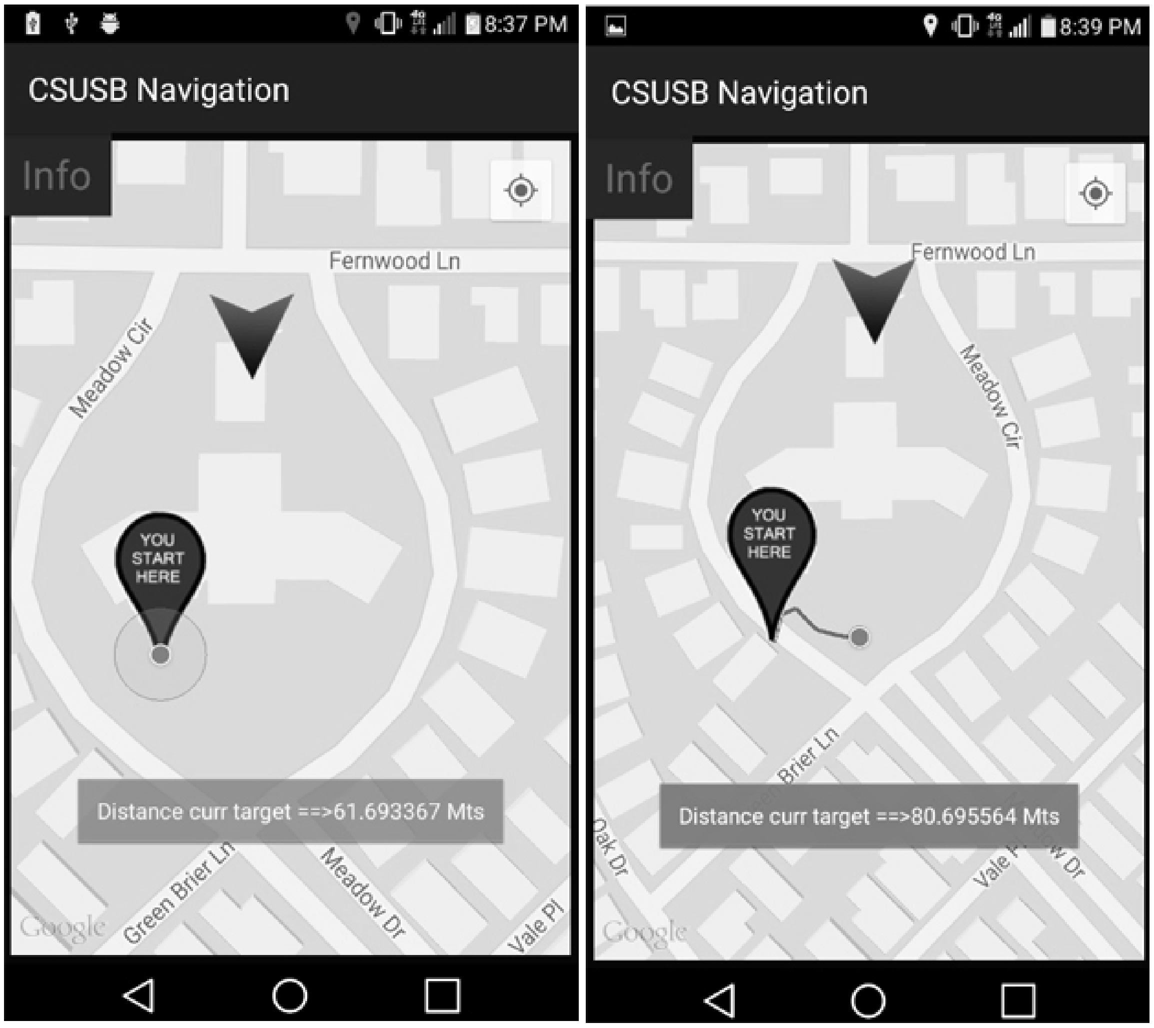


Fig. 3.5: Walking GPS I

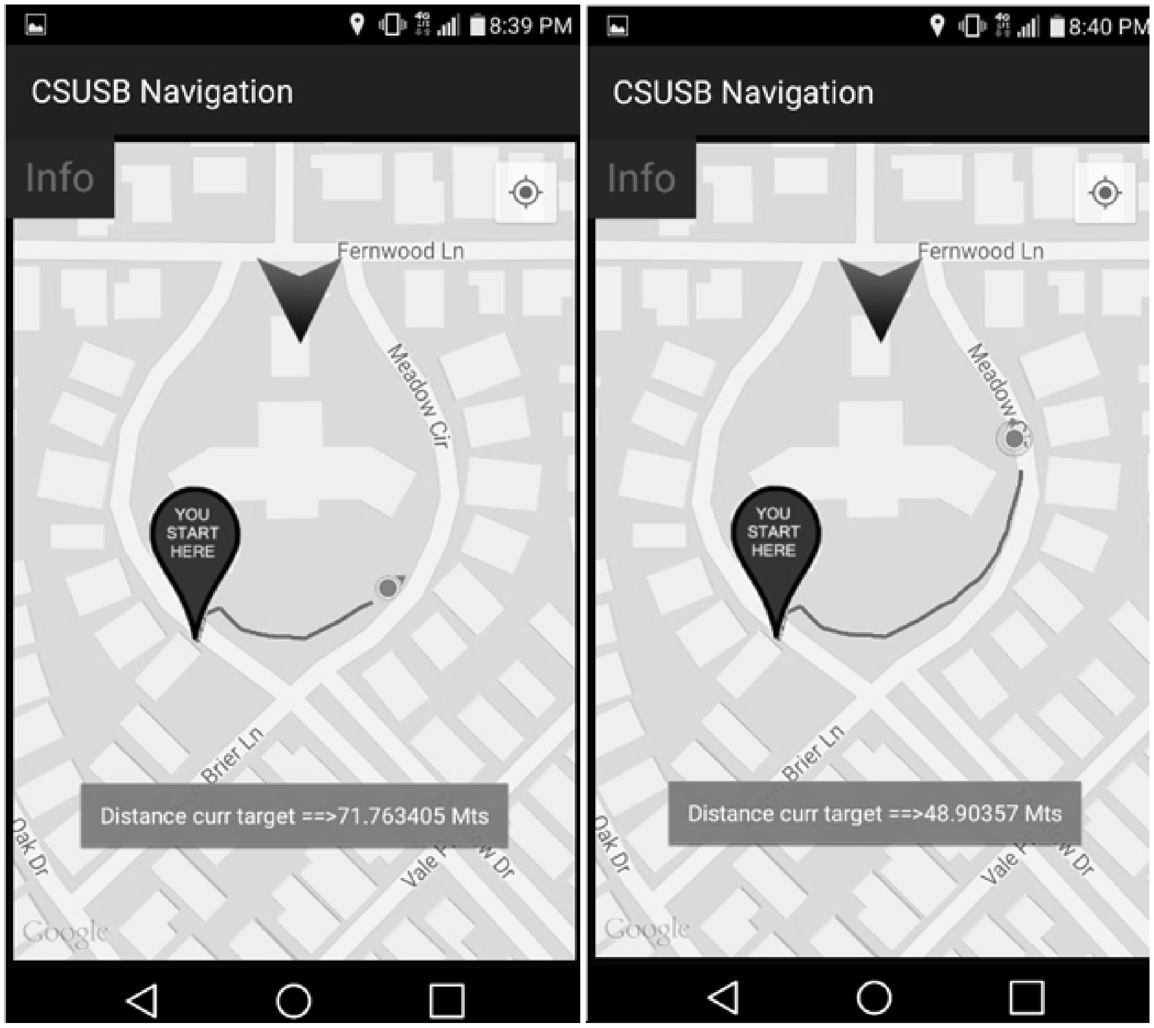
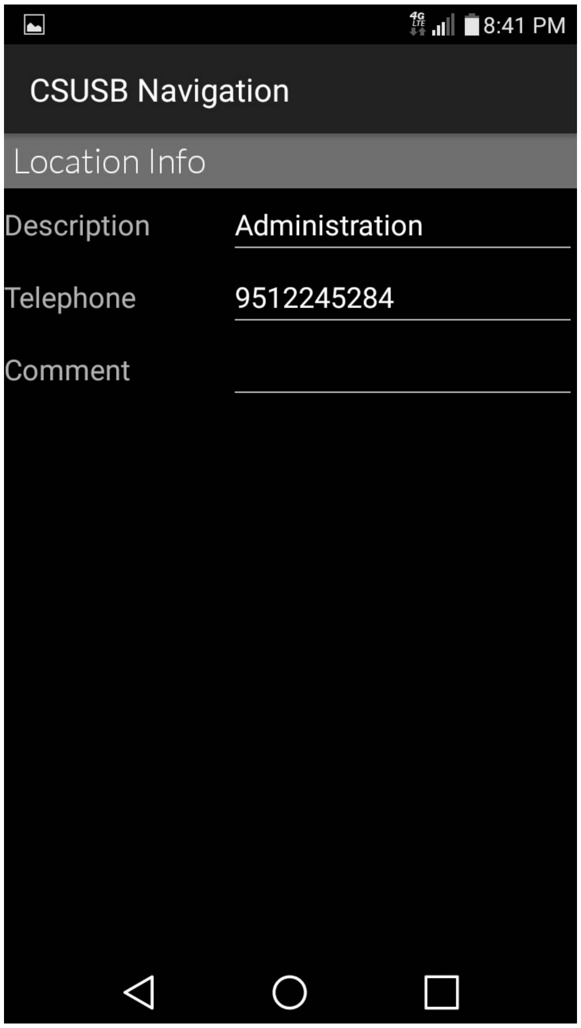


Fig. 3.6: Walking GPS II

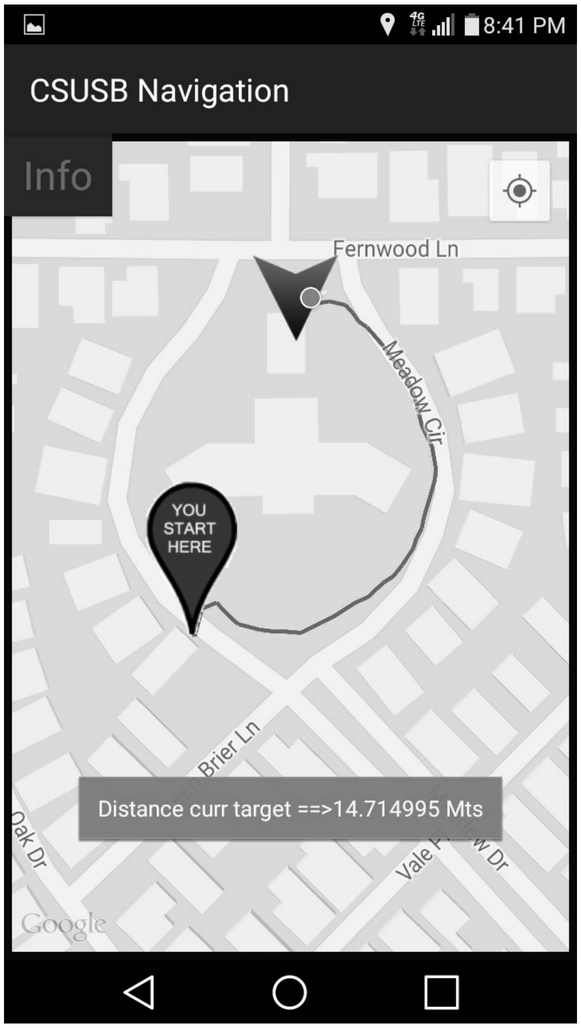
While using the map, the user can at any point, press the "Info" button. This will display some information like the telephone of the selected destination. Figure 3.7 shows is the information screen:



Description	Administration
Telephone	9512245284
Comment	

Fig. 3.7: Info Screen

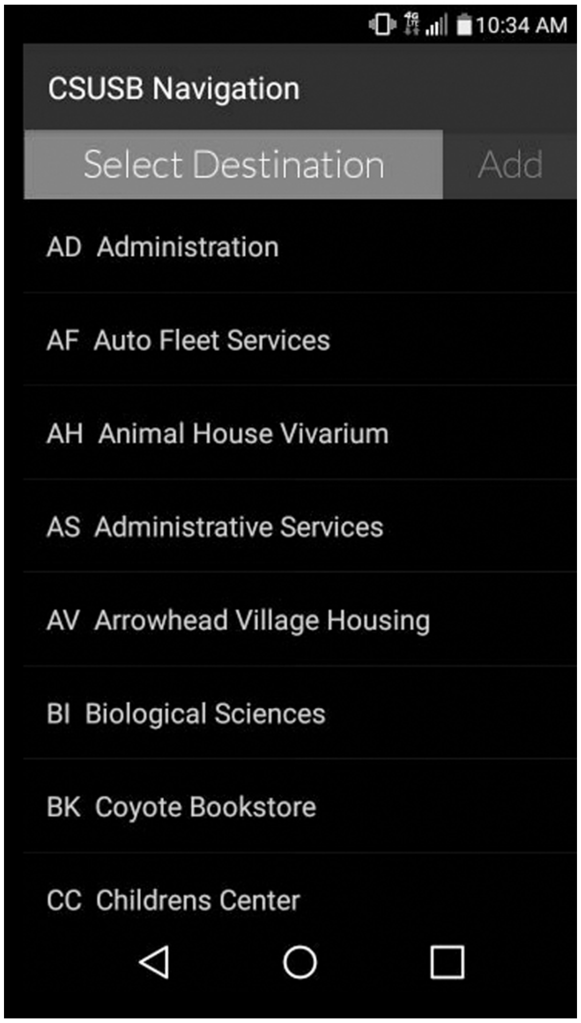
Finally the user arrives to their destination. The user will visually see the complete path. Figure 3.8 shows an entire walking GPS from point A to point B.



Distance curr target ==>14.714995 Mts

Fig. 3.8: Walking GPS III

Once the walking GPS has been reviewed, let's remember that in the initial screen, the user can also create a new location by pressing the "Add" button. See Figure 3.9.



10:34 AM

CSUSB Navigation

AD Administration

AF Auto Fleet Services

AH Animal House Vivarium

AS Administrative Services

AV Arrowhead Village Housing

BI Biological Sciences

BK Coyote Bookstore

CC Childrens Center

Fig. 3.9: Initial Screen

The add button will display a form in a new screen. Here the user can create the information about a preferred location, or a favorite spot. See Figure 3.10.

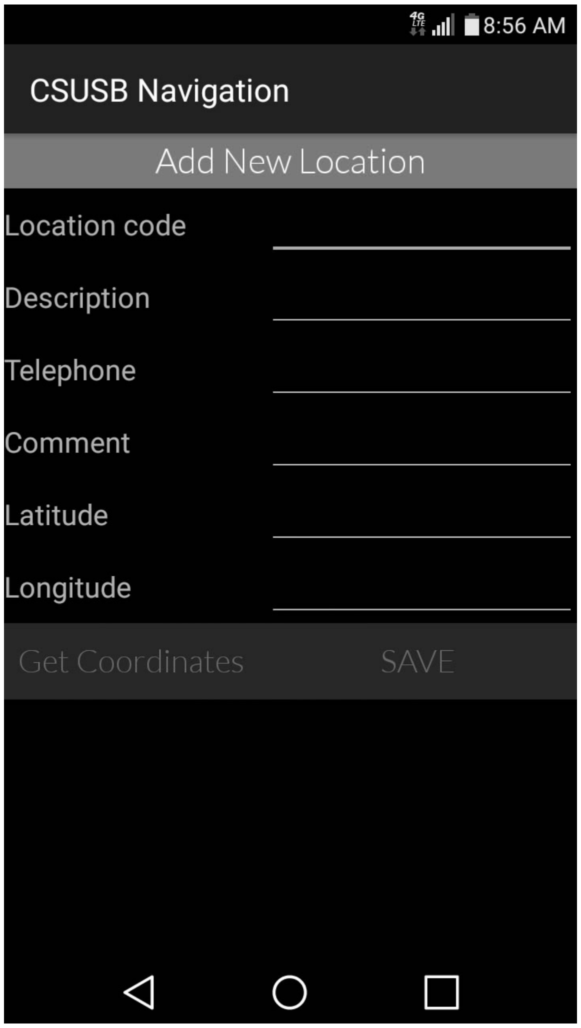
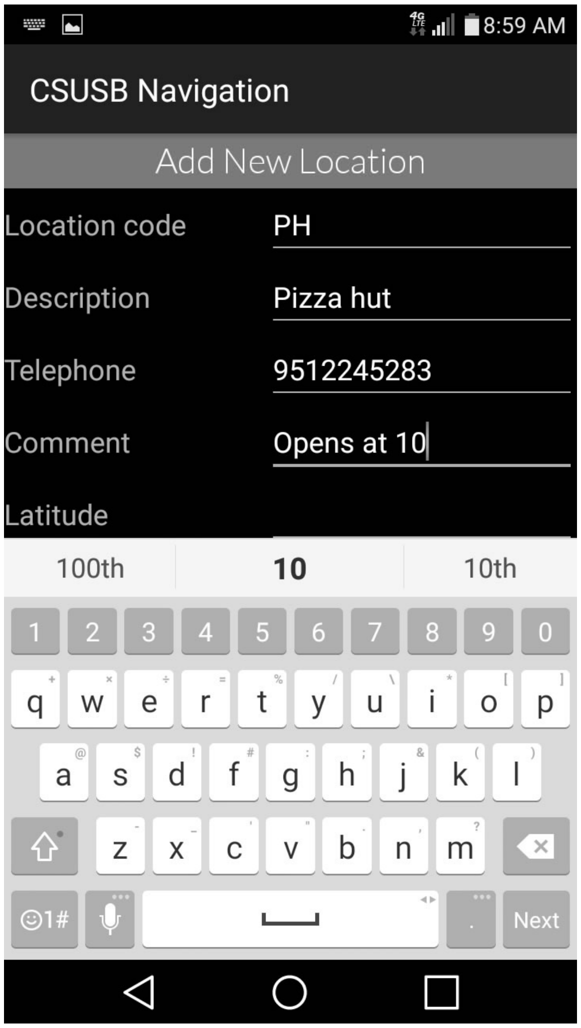


Fig. 3.10: Add a New Location Screen

Most of these attributes are pretty straight forward and the keyboard is part of the Android system. Most of the information to be entered here is pretty straight forward. See Figure 3.11.


CSUSB Navigation

Add New Location

Location code PH

Description Pizza hut

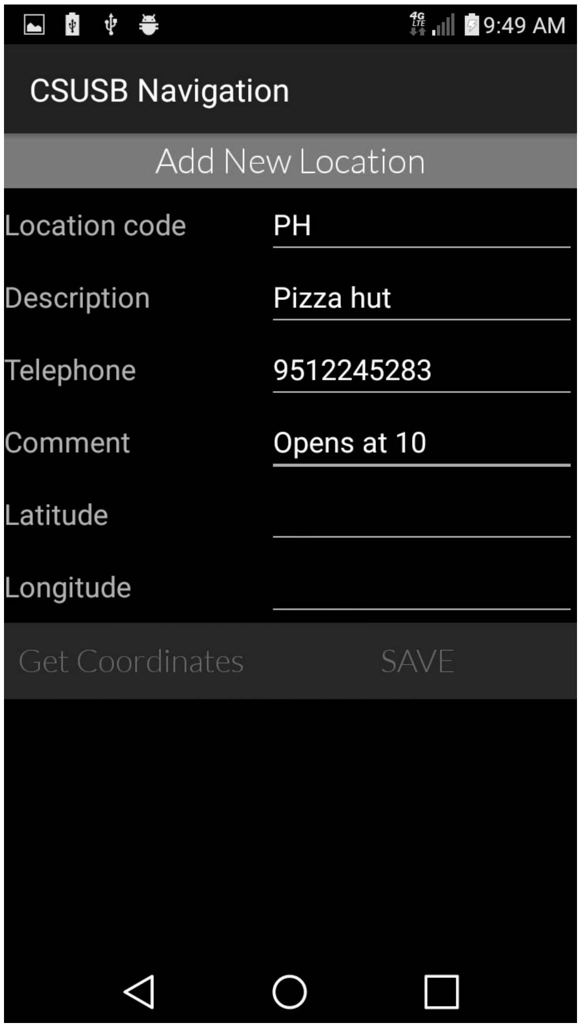
Telephone 9512245283

Comment Opens at 10

Latitude

Fig. 3.11: Add a New Location Partially Filled Screen

However, attributes Latitude and Longitude are values not easy to remember, neither they are handy without help. So, the application offers the possibility of selecting the location directly from a map. That's what the "Get Coordinates" button is for. See Figure 3.12.


9:49 AM

CSUSB Navigation

Add New Location

Location code	PH
Description	Pizza hut
Telephone	9512245283
Comment	Opens at 10
Latitude	
Longitude	

Fig. 3.12: Get Coordinates Screen

After pressing the "Get Coordinates" button, a screen with a map will be shown. A blue marker indicates the current user's location. In this street level map the user can select a destination by making a "Long Click" over the map. A red arrow will appear showing a location has been selected. By pressing "Save", latitude and longitude values are sent to the "Add location" or "Edit location" screens, depending on which one did the call. Figure 3.13 shows this:

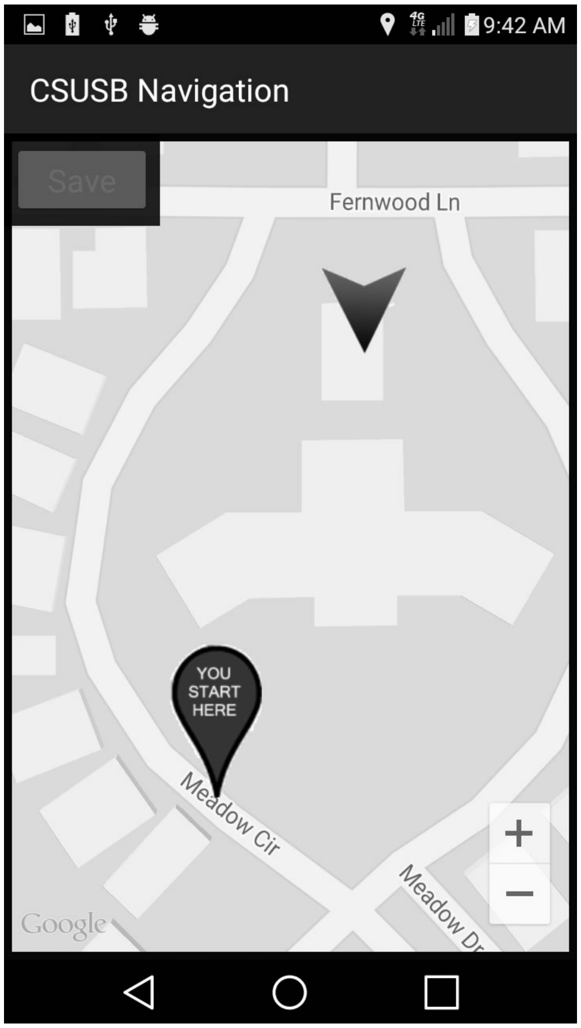
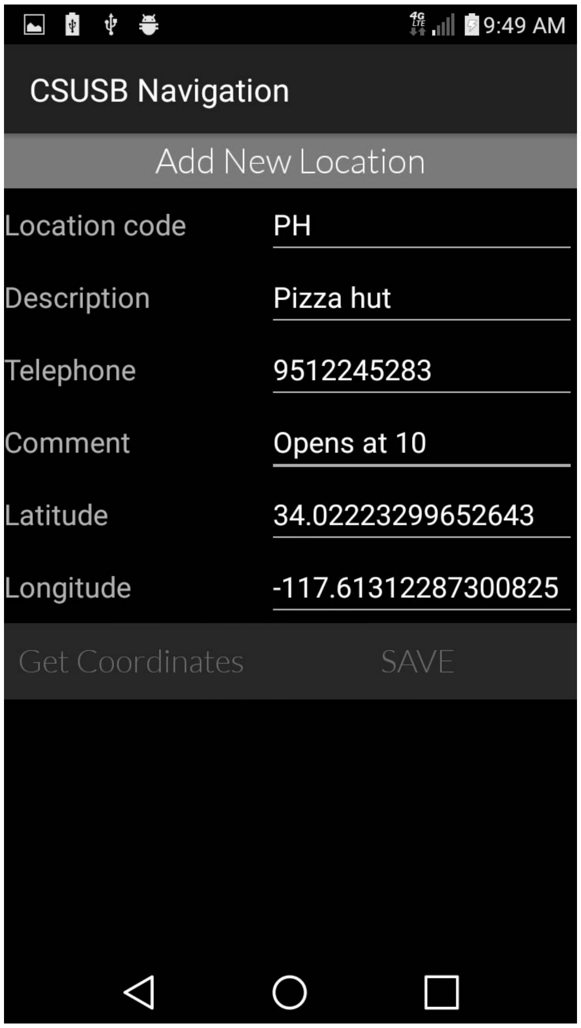


Fig. 3.13: Select a Location from Screen

In this case the ”New location” class called the ”Get Coordinates”. This is how the add a location screen will look like. Here the user may save a new location or discard the process. See Figure 3.14.

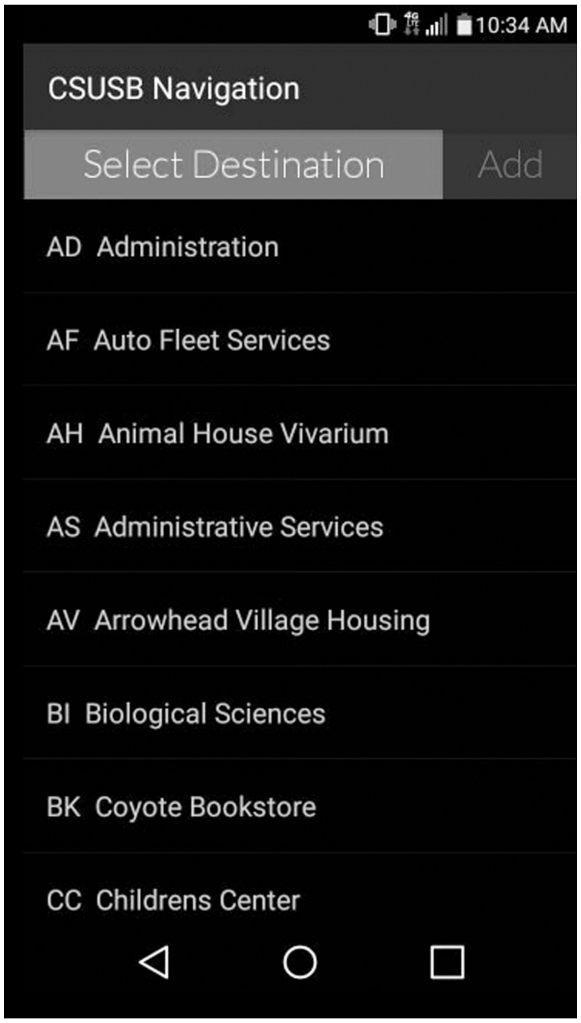

9:49 AM

CSUSB Navigation

Location code	PH
Description	Pizza hut
Telephone	9512245283
Comment	Opens at 10
Latitude	34.02223299652643
Longitude	-117.61312287300825

Fig. 3.14: Save a New Location Screen

Let's remember that in the initial screen the user can also make a "long click" in order to modify an existing location. See Figure 3.15.


10:34 AM

CSUSB Navigation

AD Administration

AF Auto Fleet Services

AH Animal House Vivarium

AS Administrative Services

AV Arrowhead Village Housing

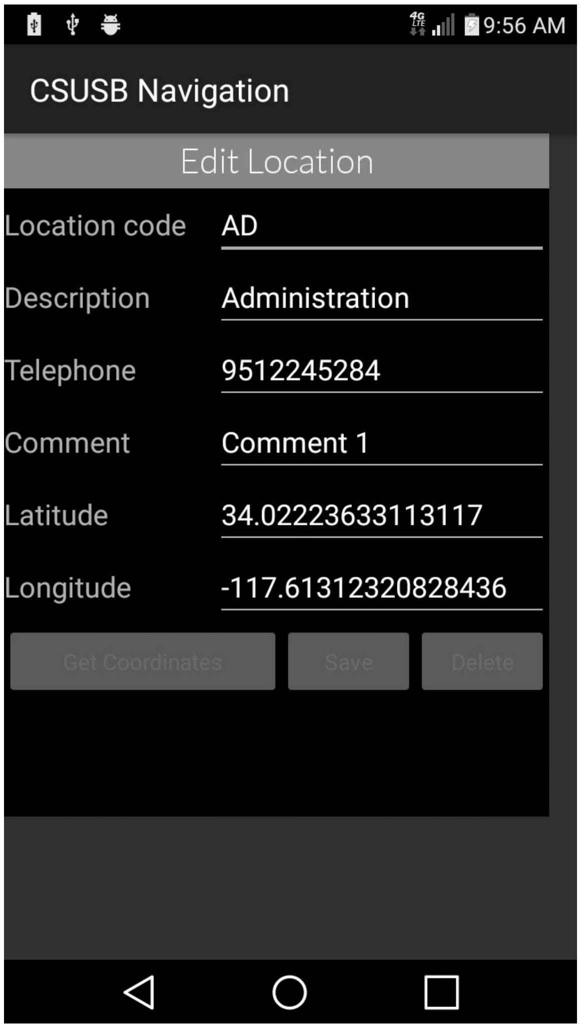
BI Biological Sciences

BK Coyote Bookstore

CC Childrens Center

Fig. 3.15: Initial Screen

We have just reviewed the walking GPS functionality of the app, and the ”Add” button. Now, let’s check the modification of a specific location. By making a long click over an specific location in the list the process will start. See Figure 3.16.



CSUSB Navigation

Edit Location

Location code	AD
Description	Administration
Telephone	9512245284
Comment	Comment 1
Latitude	34.02223633113117
Longitude	-117.61312320828436

Fig. 3.16: Edit Location Screen

The application shows in a screen the current information of a location. The user may edit any attribute on this screen. The functioning of the button "Get Coordinates" is just as it was described above. Something important to point out is the button "Delete". By pressing this button, that specific location will be deleted.

4. SOFTWARE ARCHITECTURE AND DESIGN

4.1 Main Activity Class

The entire app can be accessed from the main activity class. From this class the user can add, modify, or delete a location. The user also may use the walking GPS to navigate in the campus. The following UML package graphic explains it visually[1]. See Figure 4.1.

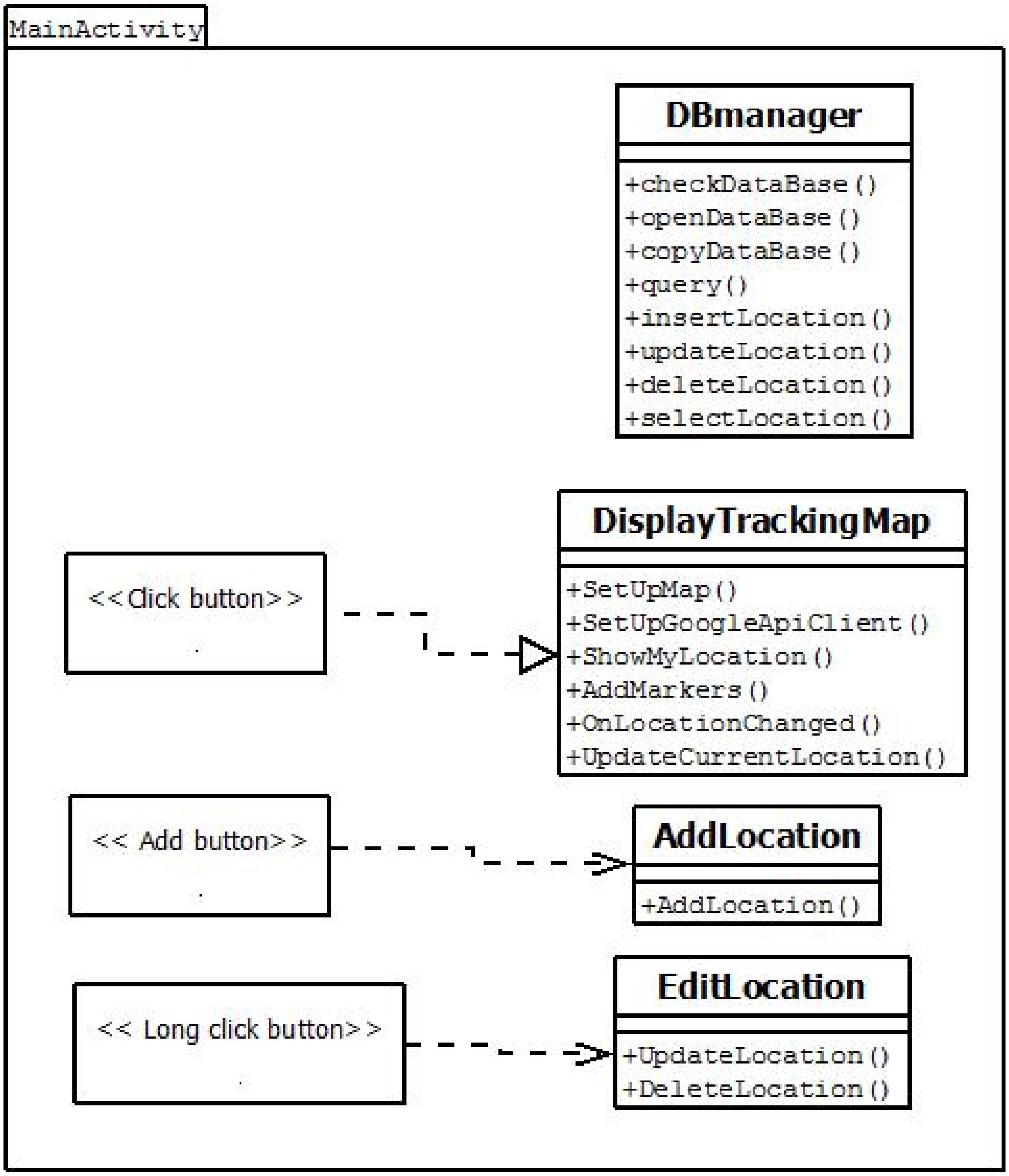


Fig. 4.1: UML Package Application

This UML graphic expresses the relationship between the classes DBmanager and MainActivity. See Figure 4.2.

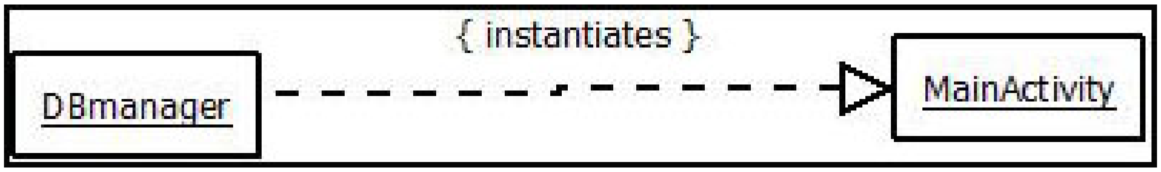
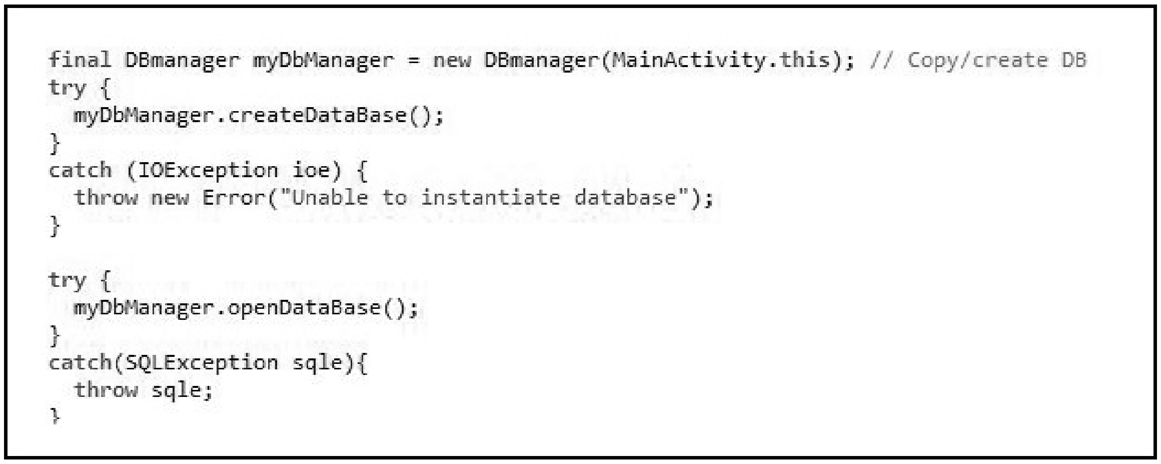


Fig. 4.2: UML Diagram of DBmanager and MainActivity Relationship

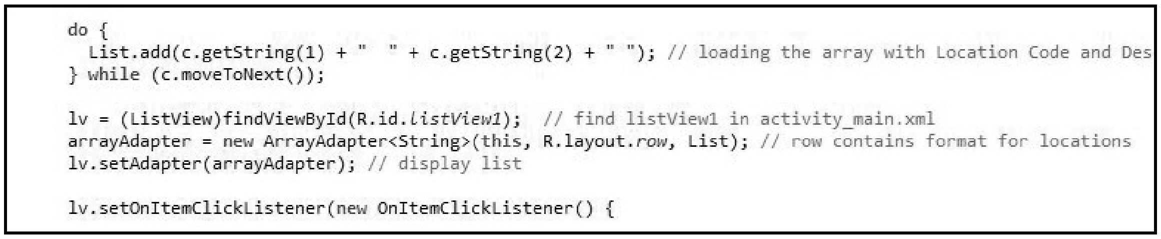
MainActivity class instantiates the DBmanager class. The MainActivity class checks the existence of a database and opens it. This piece of code shows it. See Figure 4.3.



```
final DBmanager myDbManager = new DBmanager(MainActivity.this); // Copy/create DB
try {
  myDbManager.createDataBase();
}
catch (IOException ioe) {
  throw new Error("Unable to instantiate database");
}

try {
  myDbManager.openDataBase();
}
catch(SQLException sqle){
  throw sqle;
}
```

Fig. 4.3: Main Activity Code (Database Opening)

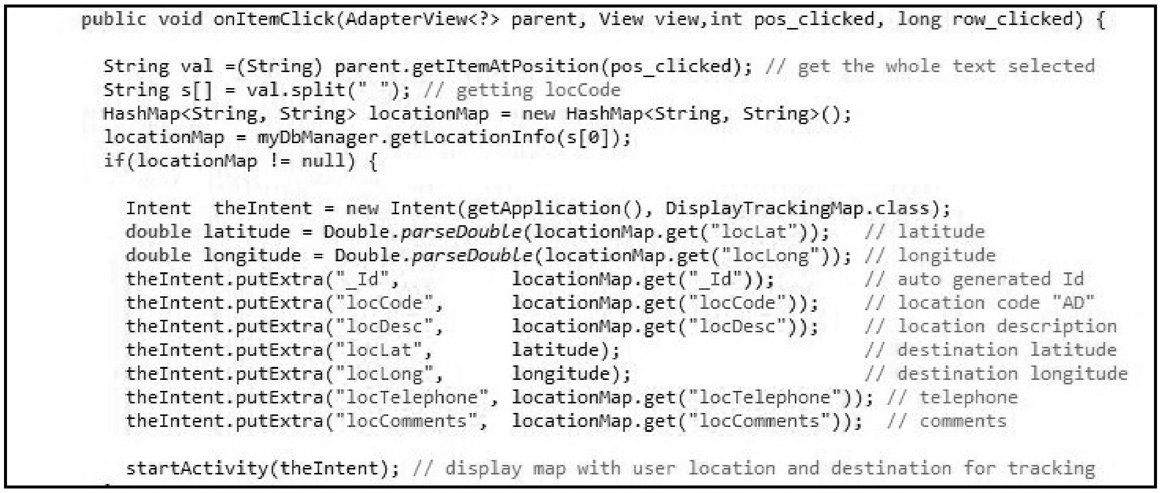
A cursor is defined as ”c”. The results of a locations query are assigned to this cursor. The content of the cursor is assigned to a list. Finally the list is displayed[2]. Figure 4.4 depicts this process.



```
do {
  List.add(c.getString(1) + "  " + c.getString(2) + " "); // loading the array with Location Code and Des
} while (c.moveToNext());

lv = (ListView)findViewById(R.id.listView1);  // find listView1 in activity_main.xml
arrayAdapter = new ArrayAdapter<String>(this, R.layout.row, List); // row contains format for locations
lv.setAdapter(arrayAdapter); // display list

lv.setOnItemClickListener(new OnItemClickListener() {
```

Fig. 4.4: Populating the List

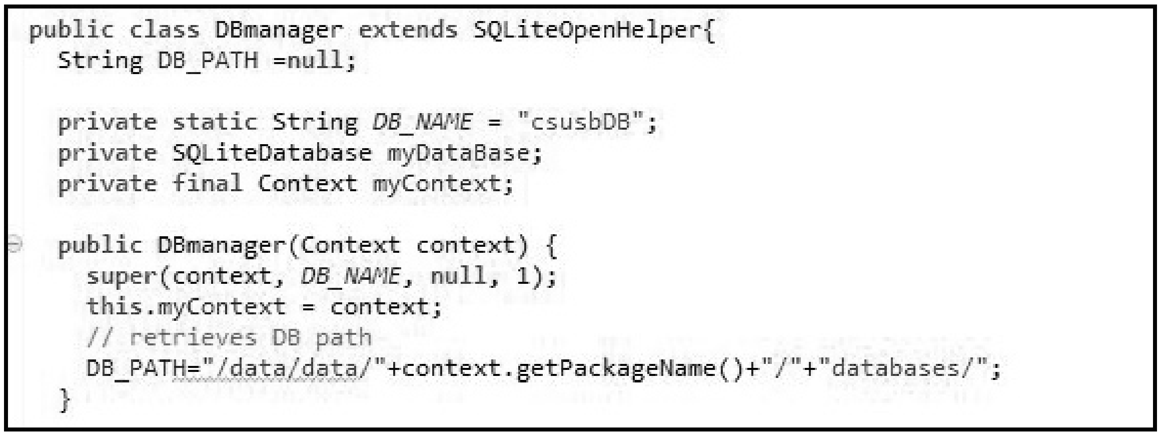
The application is listening to any input; When the user taps on a location, method OnItemClick activates. It will recover the string clicked and its position. The location code is recovered from the string. A query for that location is submitted in order to recover its info. With all that information a walking GPS is started. See Figure 4.5.



```
public void onItemClick(AdapterView<?> parent, View view,int pos_clicked, long row_clicked) {

  String val =(String) parent.getItemAtPosition(pos_clicked); // get the whole text selected
  String s[] = val.split(" "); // getting locCode
  HashMap<String, String> locationMap = new HashMap<String, String>();
  locationMap = myDbManager.getLocationInfo(s[0]);
  if(locationMap != null) {

    Intent  theIntent = new Intent(getApplication(), DisplayTrackingMap.class);
    double latitude = Double.parseDouble(locationMap.get("locLat"));   // latitude
    double longitude = Double.parseDouble(locationMap.get("locLong")); // longitude
    theIntent.putExtra("_Id",         locationMap.get("_Id"));        // auto generated Id
    theIntent.putExtra("locCode",     locationMap.get("locCode"));    // location code "AD"
    theIntent.putExtra("locDesc",     locationMap.get("locDesc"));    // location description
    theIntent.putExtra("locLat",      latitude);                      // destination latitude
    theIntent.putExtra("locLong",     longitude);                     // destination longitude
    theIntent.putExtra("locTelephone", locationMap.get("locTelephone")); // telephone
    theIntent.putExtra("locComments",  locationMap.get("locComments"));  // comments

    startActivity(theIntent); // display map with user location and destination for tracking
```

Fig. 4.5: When a Location is Clicked

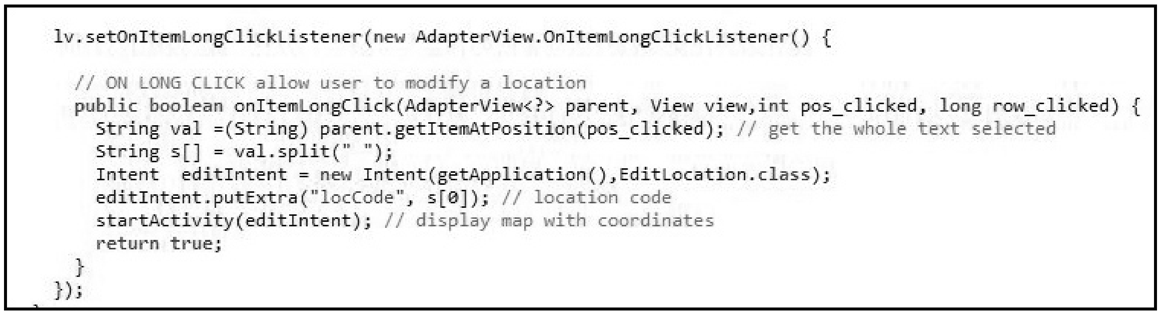
Should the user tap on the "Add" button, MainActivity will start the new location feature of the app. This method doesn't need any parameters since we are creating a location from scratch. See Figure 4.6.



```
public class DBmanager extends SQLiteOpenHelper{
  String DB_PATH =null;

  private static String DB_NAME = "csusbDB";
  private SQLiteDatabase myDataBase;
  private final Context myContext;

  public DBmanager(Context context) {
    super(context, DB_NAME, null, 1);
    this.myContext = context;
    // retrieves DB path
    DB_PATH="/data/data/"+context.getPackageName()+"/"+"databases/";
  }
```

Fig. 4.6: Calling the Methods to "Add" a New Location

If the user "long clicked" on a location from the list, the class EditLocation, will be executed. The only parameter sent to the called activity is the location code. See Figure 4.7.



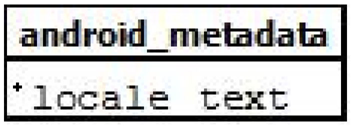
```
  lv.setOnItemLongClickListener(new AdapterView.OnItemLongClickListener() {

    // ON LONG CLICK allow user to modify a location
    public boolean onItemLongClick(AdapterView<?> parent, View view,int pos_clicked, long row_clicked) {
      String val =(String) parent.getItemAtPosition(pos_clicked); // get the whole text selected
      String s[] = val.split(" ");
      Intent  editIntent = new Intent(getApplication(),EditLocation.class);
      editIntent.putExtra("locCode", s[0]); // location code
      startActivity(editIntent); // display map with coordinates
      return true;
    }
  });
```

Fig. 4.7: On Long Click: Edit a Location

Those are the main functions on the MainActivity class.

4.2 DBmanager Class

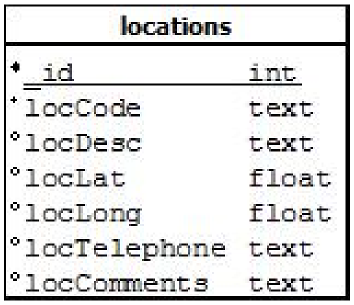
WiN GPS has a database incorporated. It is handled with SQLITE3. The database name is csusbDB and it has two tables: android meta-data and locations. Table android meta-data is a system required table, which holds information used by Android. It has one record indicating language and country i.e. en-US. The process to design this database was quite straight forward since there are no relationship among tables involved. There was no need to normalize the database. Figure 4.8 shows table meta-data structure:



android_metadata	
* locale	text

Fig. 4.8: Table Android Meta-Data: Structure

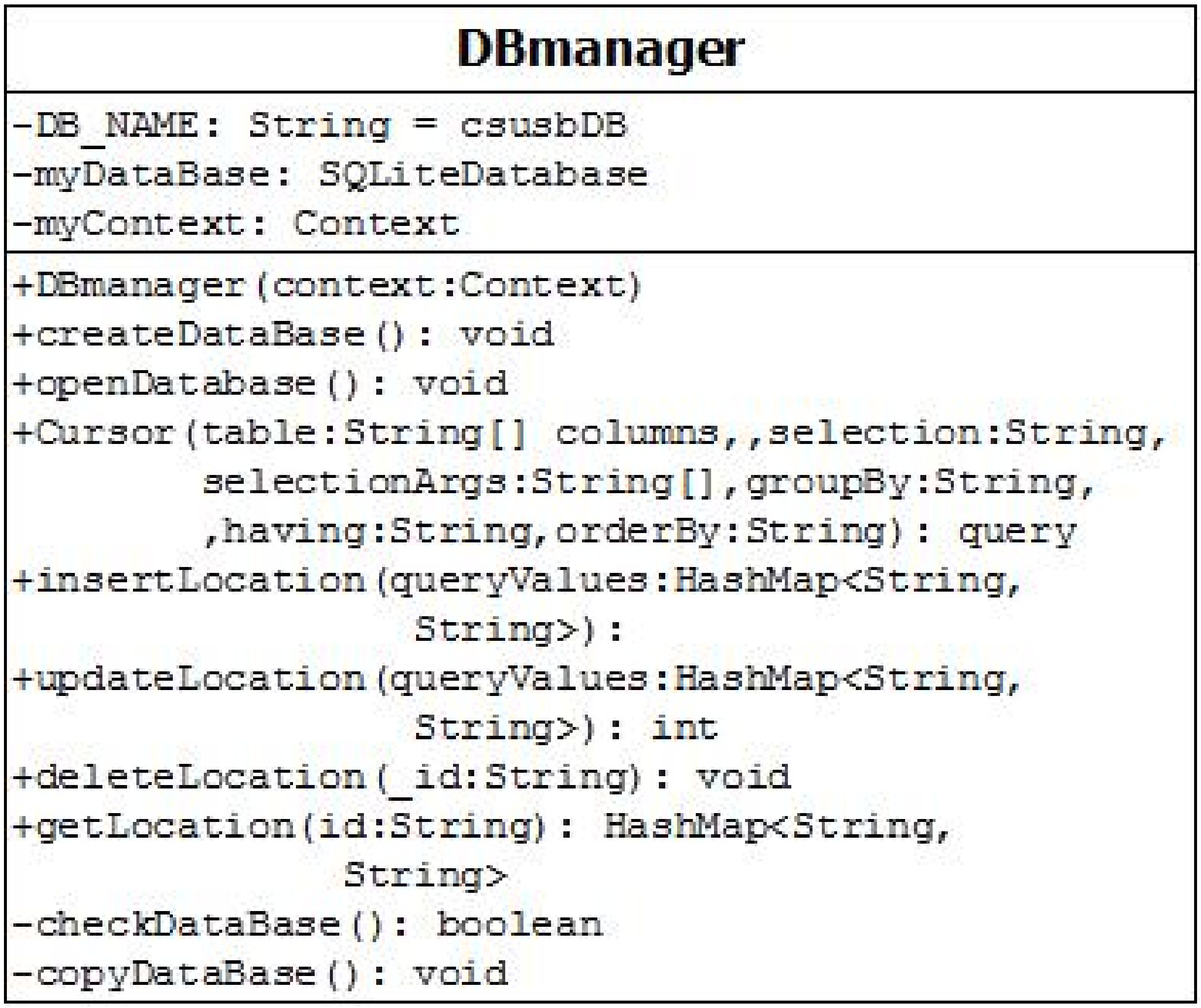
There's just one main table used by the entire application, which is Location. This is its structure. See Figure 4.9.



locations	
◆ _id	int
* locCode	text
° locDesc	text
° locLat	float
° locLong	float
° locTelephone	text
° locComments	text

Fig. 4.9: Table Locations: Structure

Class DBmanager handles all the methods involving the locations information. Its tasks are to create a new location, to locate information related to them, to edit, or to delete a specific location. The following UML diagram of the DBmanager class reflects that[1]. See Figure 4.10.



```
DBmanager
-DB_NAME: String = csusbDB
-myDataBase: SQLiteDatabase
-myContext: Context
+DBmanager(context:Context)
+createDataBase(): void
+openDatabase(): void
+Cursor(table:String[] columns,,selection:String,
        selectionArgs:String[],groupBy:String,
        ,having:String,orderBy:String): query
+insertLocation(queryValues:HashMap<String,
                String>):
+updateLocation(queryValues:HashMap<String,
                String>): int
+deleteLocation(_id:String): void
+getLocation(id:String): HashMap<String,
              String>
-checkDataBase(): boolean
-copyDataBase(): void
```

Fig. 4.10: UML DBmanager Class

Dbmanager instantiates MainActivity, NewLocation, and EditLocation classes. These classes control all the information related to locations. The following UML graphic depicts that. See Figure 4.11.

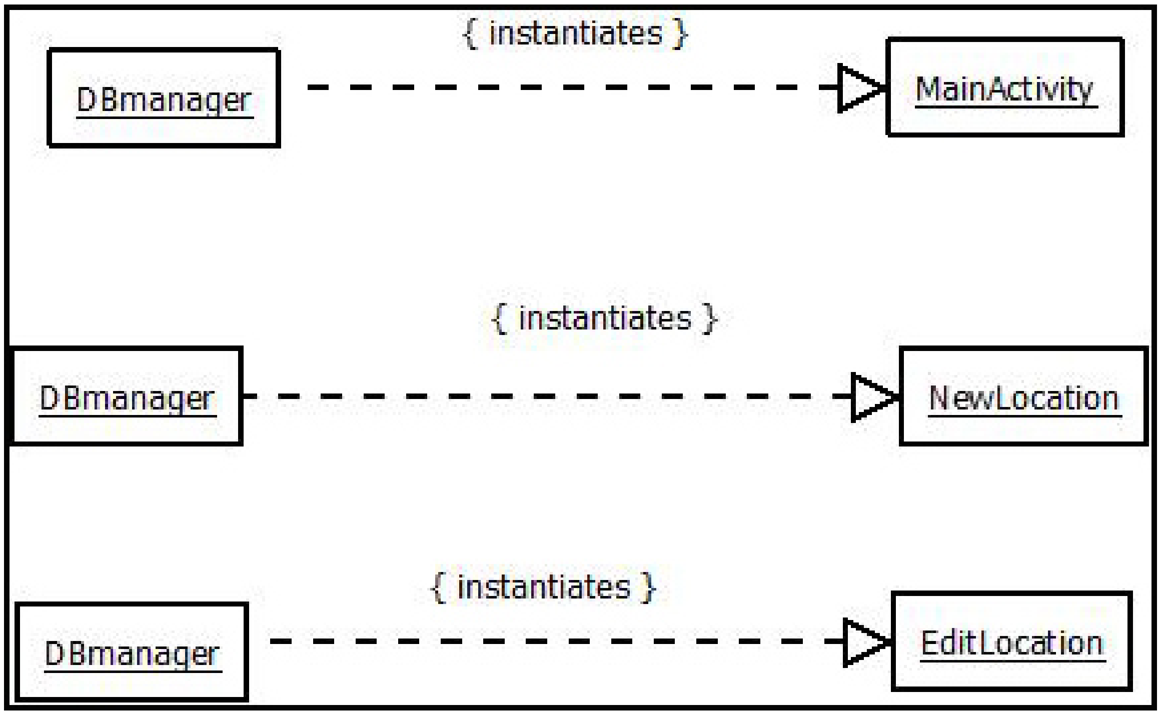
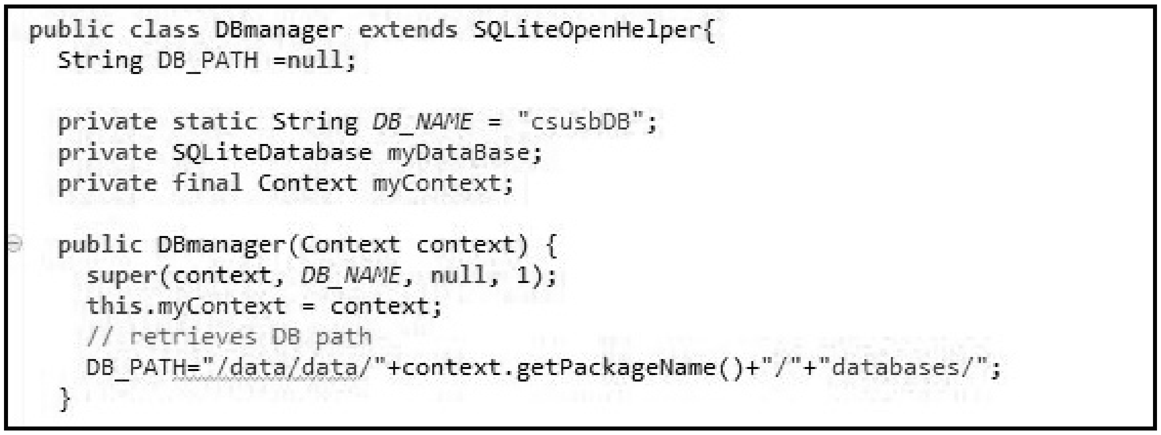


Fig. 4.11: UML DBmanager Instantiations

DBmanager inherits SQLiteOpenHelper, which is part of SQLite3. The database's name is defined and its path retrieved from the project assets as shown in Figure 4.12.

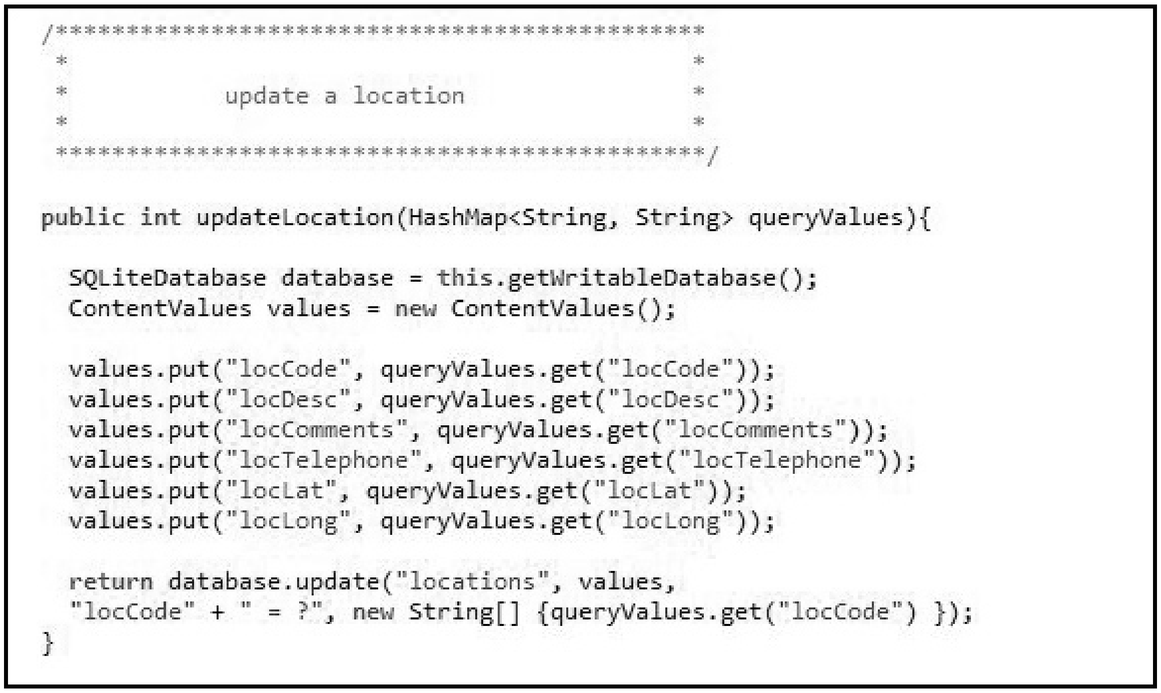


```
public class DBmanager extends SQLiteOpenHelper{
  String DB_PATH =null;

  private static String DB_NAME = "csusbDB";
  private SQLiteDatabase myDataBase;
  private final Context myContext;

  public DBmanager(Context context) {
    super(context, DB_NAME, null, 1);
    this.myContext = context;
    // retrieves DB path
    DB_PATH="/data/data/"+context.getPackageName()+"/"+"databases/";
  }
```

Fig. 4.12: Database Definition

DBmanager has several methods that will be instantiated by MainActivity, EditLocation, and NewLocation: checkDataBase, openDataBase, Cursor, insertLocation, updateLocation, delete location, and getLocation among the most important.

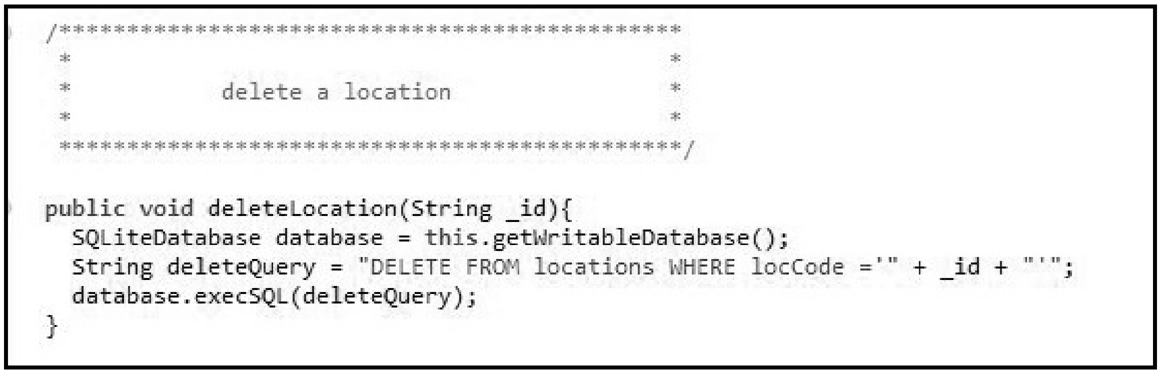
For instance, the method update location will receive values for the attributes modified and it will update the locations table using a SQL sentence pretty straight forward as figure 4.3 depicts.



```
/*******************************************
 *                                         *
 *           update a location             *
 *                                         *
 *******************************************/

public int updateLocation(HashMap<String, String> queryValues){

  SQLiteDatabase database = this.getWritableDatabase();
  ContentValues values = new ContentValues();

  values.put("locCode", queryValues.get("locCode"));
  values.put("locDesc", queryValues.get("locDesc"));
  values.put("locComments", queryValues.get("locComments"));
  values.put("locTelephone", queryValues.get("locTelephone"));
  values.put("locLat", queryValues.get("locLat"));
  values.put("locLong", queryValues.get("locLong"));

  return database.update("locations", values,
  "locCode" + " = ?", new String[] {queryValues.get("locCode") });
}
```

Fig. 4.13: Edit Location, Code

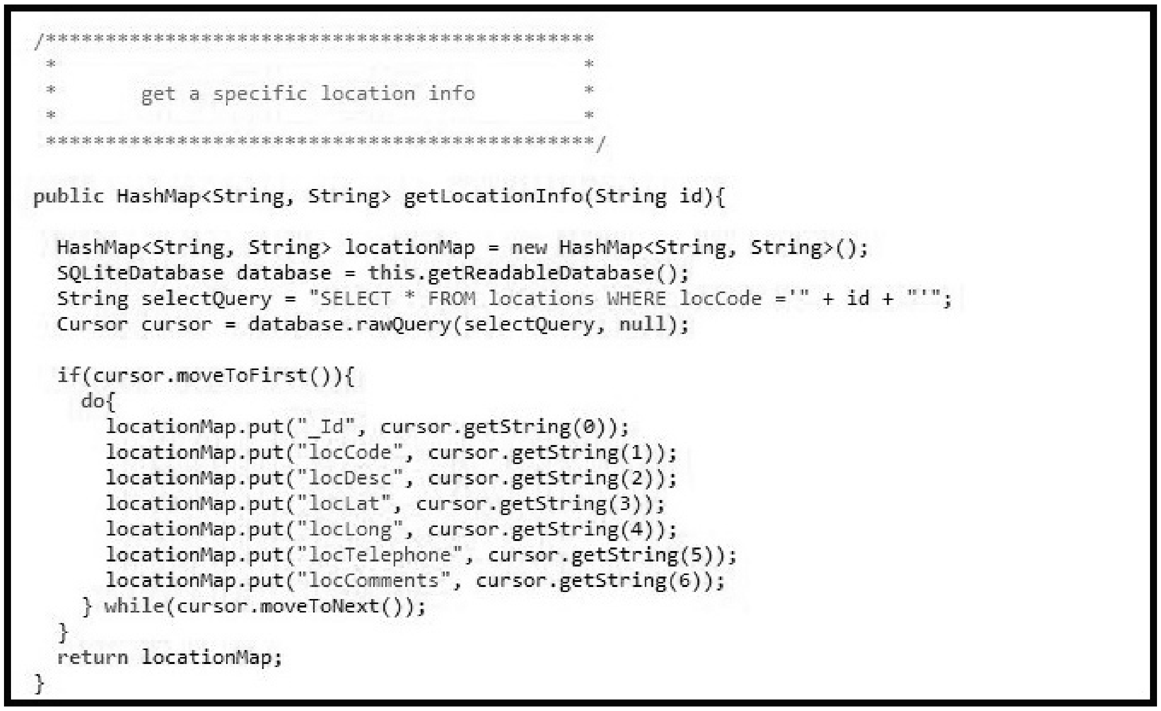
Method deleteLocation receives location code as a parameter. That location is deleted from table locations using the method deleteLocation from the DBmanager class. See Figure 4.14.



```
/**********************************************
 *                                            *
 *           delete a location                *
 *                                            *
 **********************************************/

public void deleteLocation(String _id){
  SQLiteDatabase database = this.getWritableDatabase();
  String deleteQuery = "DELETE FROM locations WHERE locCode ='" + _id + "'";
  database.execSQL(deleteQuery);
}
```

Fig. 4.14: Deleting a Location

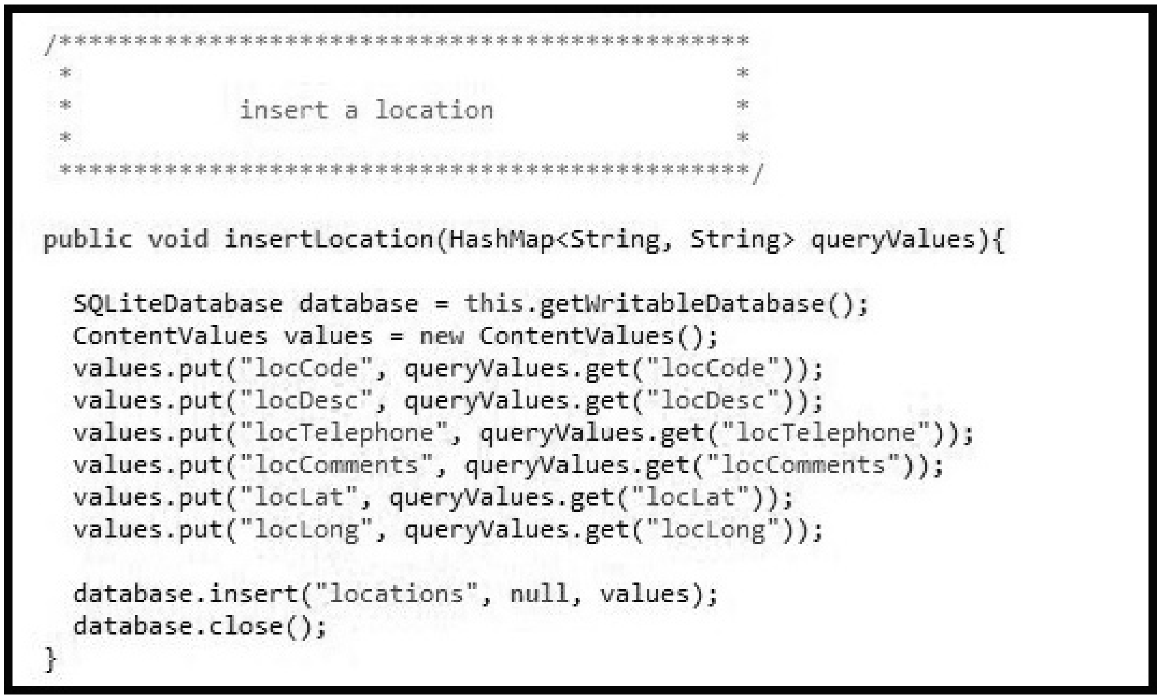
Another functionality from the DBmanager class is the getLocationInfoMethod. This method receives as a parameter the location code. It uses a hashmap that will be populated with a query for that specific location. See Figure 4.15.



```
/**********************************************
 *                                            *
 *      get a specific location info          *
 *                                            *
 **********************************************/

public HashMap<String, String> getLocationInfo(String id){

  HashMap<String, String> locationMap = new HashMap<String, String>();
  SQLiteDatabase database = this.getReadableDatabase();
  String selectQuery = "SELECT * FROM locations WHERE locCode ='" + id + "'";
  Cursor cursor = database.rawQuery(selectQuery, null);

  if(cursor.moveToFirst()){
    do{
      locationMap.put("_Id", cursor.getString(0));
      locationMap.put("locCode", cursor.getString(1));
      locationMap.put("locDesc", cursor.getString(2));
      locationMap.put("locLat", cursor.getString(3));
      locationMap.put("locLong", cursor.getString(4));
      locationMap.put("locTelephone", cursor.getString(5));
      locationMap.put("locComments", cursor.getString(6));
    } while(cursor.moveToNext());
  }
  return locationMap;
}
```

Fig. 4.15: Get Info For a Location

The last method to mention from DBmanager class is the insertLocation method. It receives as a parameter a string with the values for the new location. These individual values are recovered from the string and then used in an insert query command. See Figure 4.16.



```
/*******************************************
 *                                         *
 *          insert a location              *
 *                                         *
 *******************************************/

public void insertLocation(HashMap<String, String> queryValues){

  SQLiteDatabase database = this.getWritableDatabase();
  ContentValues values = new ContentValues();
  values.put("locCode", queryValues.get("locCode"));
  values.put("locDesc", queryValues.get("locDesc"));
  values.put("locTelephone", queryValues.get("locTelephone"));
  values.put("locComments", queryValues.get("locComments"));
  values.put("locLat", queryValues.get("locLat"));
  values.put("locLong", queryValues.get("locLong"));

  database.insert("locations", null, values);
  database.close();
}
```

Fig. 4.16: Creating a New Location.

4.3 DisplayTrackingMap Class

This class handles a street level map with the user's location, destination, and walking GPS. It is not possible to instantiate a Google Map object directly into the app. The way to do it is by adding a MapFragment and then instantiating the FragmentActivity object, which is part of the Google Play Services library I mentioned before. Figure 4.17 shows the UML instantiation[1]:

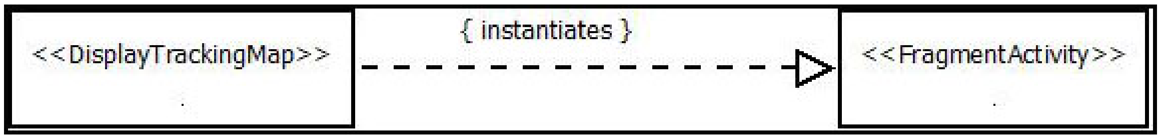
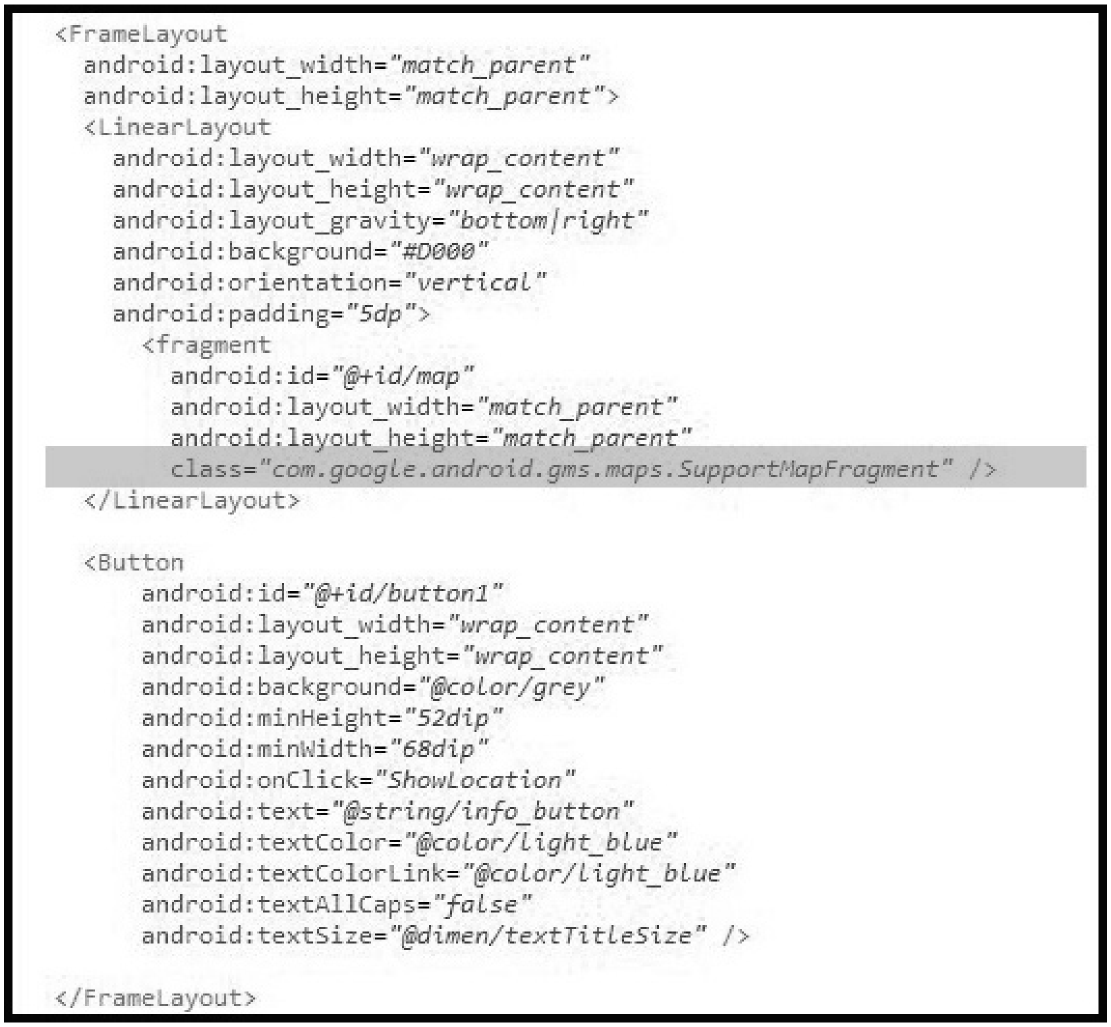


Fig. 4.17: UML Instantiation of Fragment Activity

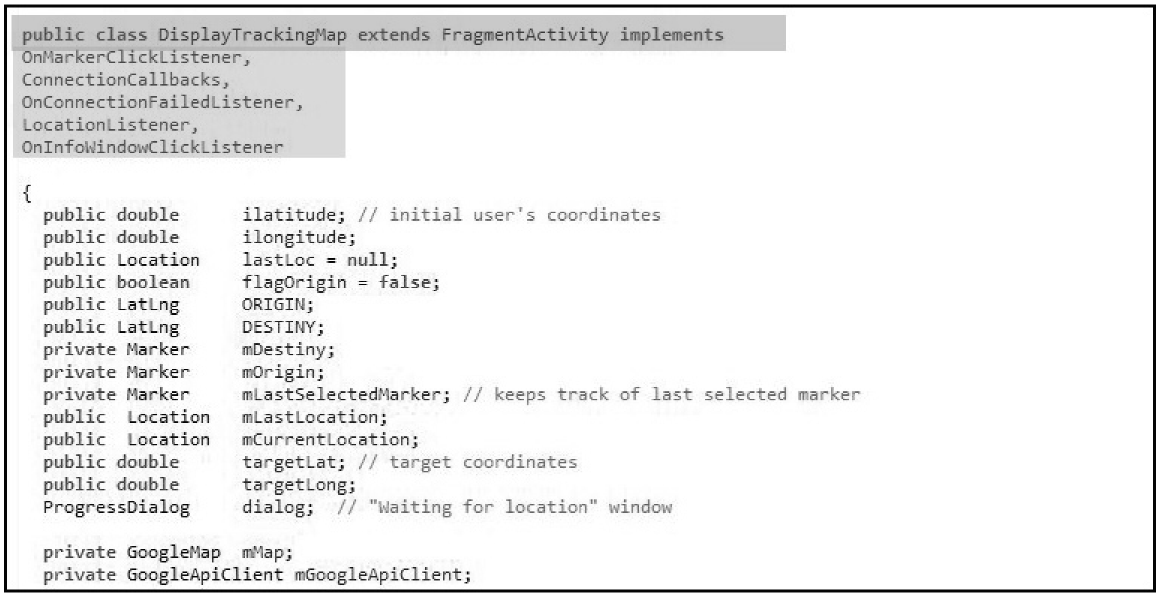
There's more than one way to instantiate a map fragment in an Android app. It could be done directly in the activity or in the activity's layout[2]. The approach I used is the second one. The following code shows the fragment definition in the Map layout. See Figure 4.18.



```xml
<FrameLayout
  android:layout_width="match_parent"
  android:layout_height="match_parent">
  <LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="bottom|right"
    android:background="#D000"
    android:orientation="vertical"
    android:padding="5dp">
      <fragment
        android:id="@+id/map"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        class="com.google.android.gms.maps.SupportMapFragment" />
  </LinearLayout>

  <Button
      android:id="@+id/button1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:background="@color/grey"
      android:minHeight="52dip"
      android:minWidth="68dip"
      android:onClick="ShowLocation"
      android:text="@string/info_button"
      android:textColor="@color/light_blue"
      android:textColorLink="@color/light_blue"
      android:textAllCaps="false"
      android:textSize="@dimen/textTitleSize" />

</FrameLayout>
```

Fig. 4.18: Fragment Definition in Map Layout

This is instantiated in DisplayTrackingMap class. See Figure 4.19.



```
public class DisplayTrackingMap extends FragmentActivity implements
OnMarkerClickListener,
ConnectionCallbacks,
OnConnectionFailedListener,
LocationListener,
OnInfoWindowClickListener

{
  public double      ilatitude; // initial user's coordinates
  public double      ilongitude;
  public Location    lastLoc = null;
  public boolean     flagOrigin = false;
  public LatLng      ORIGIN;
  public LatLng      DESTINY;
  private Marker     mDestiny;
  private Marker     mOrigin;
  private Marker     mLastSelectedMarker; // keeps track of last selected marker
  public  Location   mLastLocation;
  public  Location   mCurrentLocation;
  public double      targetLat; // target coordinates
  public double      targetLong;
  ProgressDialog     dialog;  // "Waiting for location" window

  private GoogleMap  mMap;
  private GoogleApiClient mGoogleApiClient;
```

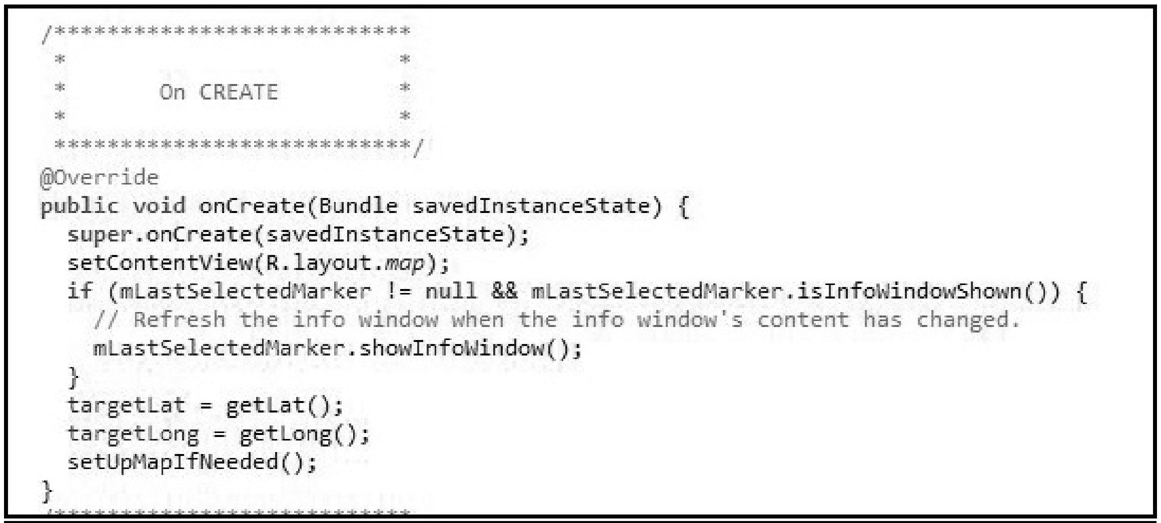
Fig. 4.19: Map Fragment Instantiation in DisplayTrackingMap

It also inherits methods:

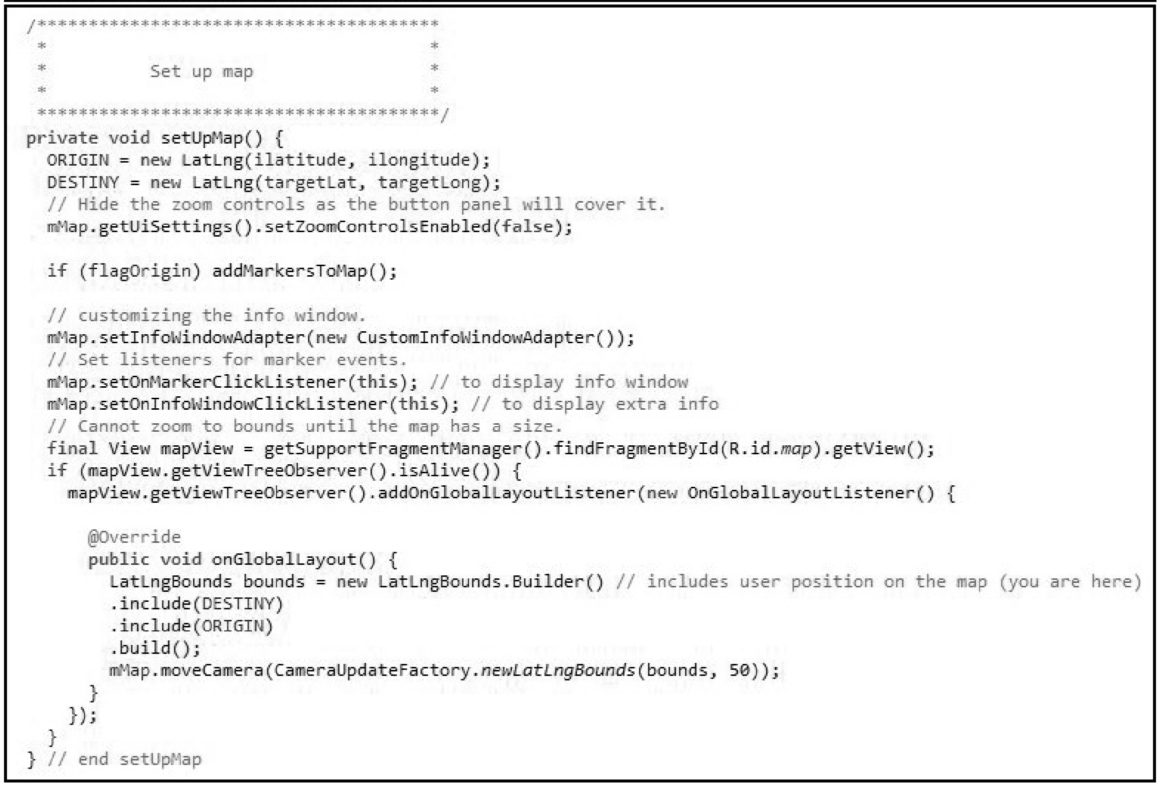
- OnMarkerClickListener,
- ConnectionCallbacks,
- OnConnectionFailedListener,
- LocationListener,
- OnInfoWindowClickListener

By using a map fragment Google maps will display the map, it will connect the activity to the API services, and it will manage the basic controls of the map such as zooming.

Activity DisplayTrackingMap on its create method defines the connection between the activity and its layout. It gets target latitude and longitude from table locations. It will set markers on user and target locations. Besides, it will turn off the control zoom and it will set up the camera[2]. See Figure 4.20.



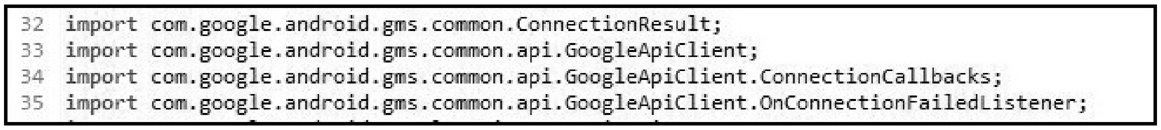
```
/***************************
 *                         *
 *       On CREATE         *
 *                         *
 ***************************/
@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.map);
  if (mLastSelectedMarker != null && mLastSelectedMarker.isInfoWindowShown()) {
    // Refresh the info window when the info window's content has changed.
    mLastSelectedMarker.showInfoWindow();
  }
  targetLat = getLat();
  targetLong = getLong();
  setUpMapIfNeeded();
}
/***************************
```



```
/**************************************
 *                                    *
 *         Set up map                 *
 *                                    *
 **************************************/
private void setUpMap() {
  ORIGIN = new LatLng(ilatitude, ilongitude);
  DESTINY = new LatLng(targetLat, targetLong);
  // Hide the zoom controls as the button panel will cover it.
  mMap.getUiSettings().setZoomControlsEnabled(false);

  if (flagOrigin) addMarkersToMap();

  // customizing the info window.
  mMap.setInfoWindowAdapter(new CustomInfoWindowAdapter());
  // Set listeners for marker events.
  mMap.setOnMarkerClickListener(this); // to display info window
  mMap.setOnInfoWindowClickListener(this); // to display extra info
  // Cannot zoom to bounds until the map has a size.
  final View mapView = getSupportFragmentManager().findFragmentById(R.id.map).getView();
  if (mapView.getViewTreeObserver().isAlive()) {
    mapView.getViewTreeObserver().addOnGlobalLayoutListener(new OnGlobalLayoutListener() {

      @Override
      public void onGlobalLayout() {
        LatLngBounds bounds = new LatLngBounds.Builder() // includes user position on the map (you are here)
        .include(DESTINY)
        .include(ORIGIN)
        .build();
        mMap.moveCamera(CameraUpdateFactory.newLatLngBounds(bounds, 50));
      }
    });
  }
} // end setUpMap
```

Fig. 4.20: Initializing the Map

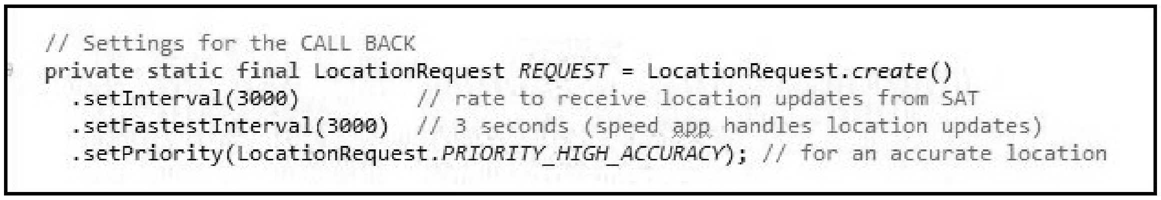
Once the map is settled, it is necessary to track the user's movements across the map. For this is necessary to import the library for the function callbacks. See Figure 4.21.



```java
import com.google.android.gms.common.ConnectionResult;
import com.google.android.gms.common.api.GoogleApiClient;
import com.google.android.gms.common.api.GoogleApiClient.ConnectionCallbacks;
import com.google.android.gms.common.api.GoogleApiClient.OnConnectionFailedListener;
```

Fig. 4.21: Importing ConnectionCallbacks

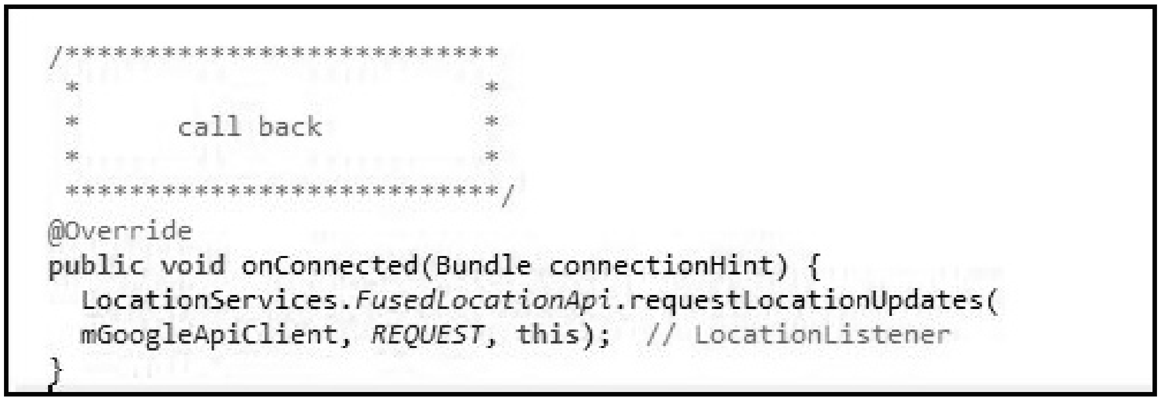
This library allows callbacks. A function to be called when the user connects or disconnects. In WiN GPS I implemented method onConnected to make requests about the user's location changes. First I define the REQUEST parameters that will be used by the callback[3]. See Figure 4.22.



```java
// Settings for the CALL BACK
private static final LocationRequest REQUEST = LocationRequest.create()
  .setInterval(3000)        // rate to receive location updates from SAT
  .setFastestInterval(3000)  // 3 seconds (speed app handles location updates)
  .setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY); // for an accurate location
```

Fig. 4.22: Request: Parameters for Callback

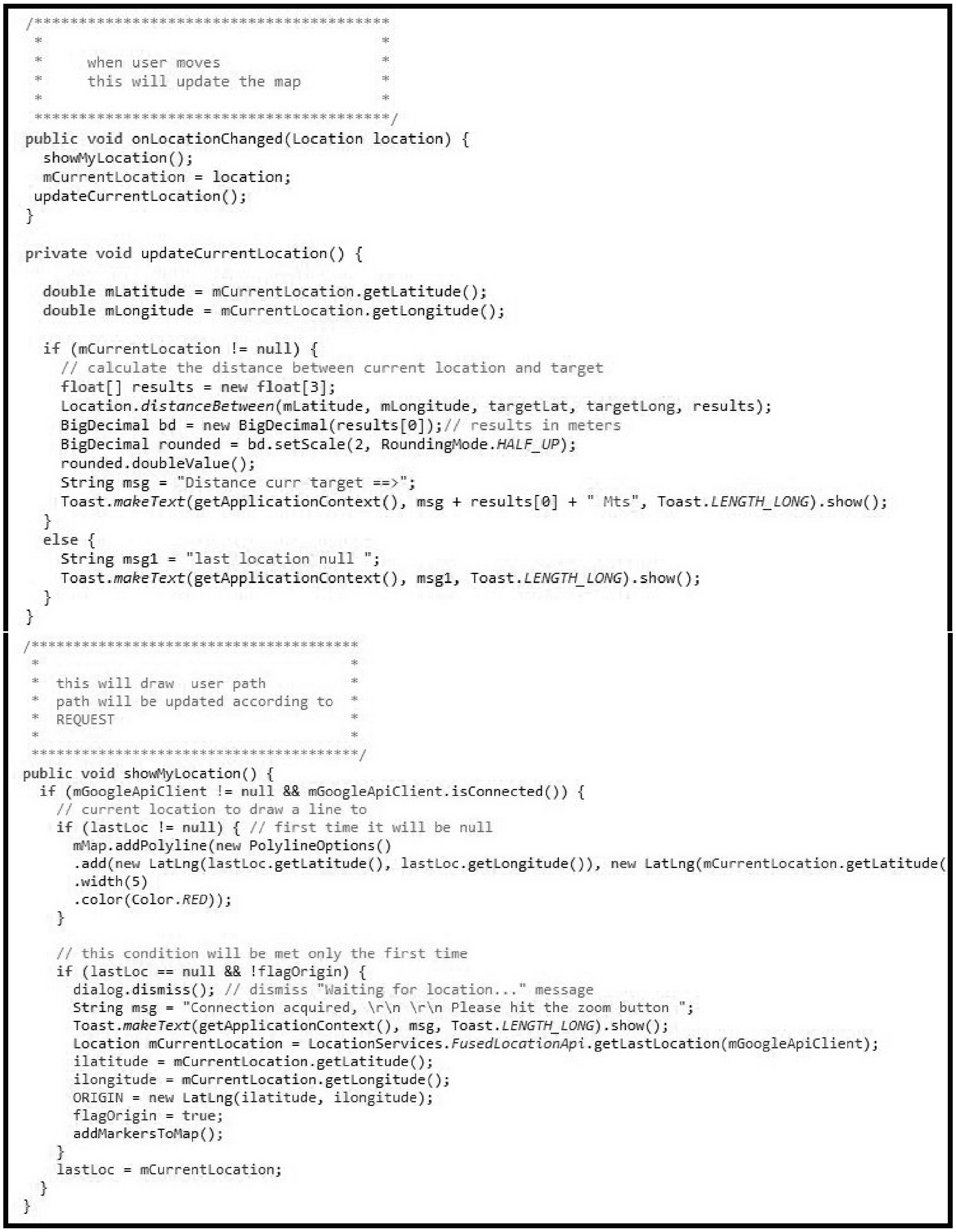
This request is used on method OnConnected. The requestLocationUpdates relates the callback with method onLocationChanged, which will be executed every time the user's location has changed[8]. See Figure 4.23.



```java
/**************************
 *                        *
 *      call back         *
 *                        *
 **************************/
@Override
public void onConnected(Bundle connectionHint) {
  LocationServices.FusedLocationApi.requestLocationUpdates(
  mGoogleApiClient, REQUEST, this);  // LocationListener
}
```

Fig. 4.23: OnConnected Definition

Method onLocationChanged will update continually the user's location. It will also calculate the distance between the current location of user and the selected destination. Method showMyLocation draws the user's path one line at a time. Figure 4.24 depicts that.



```
/***************************************
 *                                     *
 *     when user moves                 *
 *     this will update the map        *
 *                                     *
 ***************************************/
public void onLocationChanged(Location location) {
  showMyLocation();
  mCurrentLocation = location;
 updateCurrentLocation();
}

private void updateCurrentLocation() {

  double mLatitude = mCurrentLocation.getLatitude();
  double mLongitude = mCurrentLocation.getLongitude();

  if (mCurrentLocation != null) {
    // calculate the distance between current location and target
    float[] results = new float[3];
    Location.distanceBetween(mLatitude, mLongitude, targetLat, targetLong, results);
    BigDecimal bd = new BigDecimal(results[0]);// results in meters
    BigDecimal rounded = bd.setScale(2, RoundingMode.HALF_UP);
    rounded.doubleValue();
    String msg = "Distance curr target ==>";
    Toast.makeText(getApplicationContext(), msg + results[0] + " Mts", Toast.LENGTH_LONG).show();
  }
  else {
    String msg1 = "last location null ";
    Toast.makeText(getApplicationContext(), msg1, Toast.LENGTH_LONG).show();
  }
}
/*************************************
 *                                   *
 *  this will draw  user path        *
 *  path will be updated according to  *
 *  REQUEST                          *
 *                                   *
 *************************************/
public void showMyLocation() {
  if (mGoogleApiClient != null && mGoogleApiClient.isConnected()) {
    // current location to draw a line to
    if (lastLoc != null) { // first time it will be null
      mMap.addPolyline(new PolylineOptions()
      .add(new LatLng(lastLoc.getLatitude(), lastLoc.getLongitude()), new LatLng(mCurrentLocation.getLatitude(
      .width(5)
      .color(Color.RED));
    }

    // this condition will be met only the first time
    if (lastLoc == null && !flagOrigin) {
      dialog.dismiss(); // dismiss "Waiting for location..." message
      String msg = "Connection acquired, \r\n \r\n Please hit the zoom button ";
      Toast.makeText(getApplicationContext(), msg, Toast.LENGTH_LONG).show();
      Location mCurrentLocation = LocationServices.FusedLocationApi.getLastLocation(mGoogleApiClient);
      ilatitude = mCurrentLocation.getLatitude();
      ilongitude = mCurrentLocation.getLongitude();
      ORIGIN = new LatLng(ilatitude, ilongitude);
      flagOrigin = true;
      addMarkersToMap();
    }
    lastLoc = mCurrentLocation;
  }
}
```

Fig. 4.24: Updating User's Movement

This is a recursive process that will take while the user walks to the selected location.

4.4 New Location Class

NewLocation class is to create a new location in table locations. It uses a layout to input data. Eclipse allows to manage layouts graphically. This is a print screen of the New Location layout. See Figure 4.25.

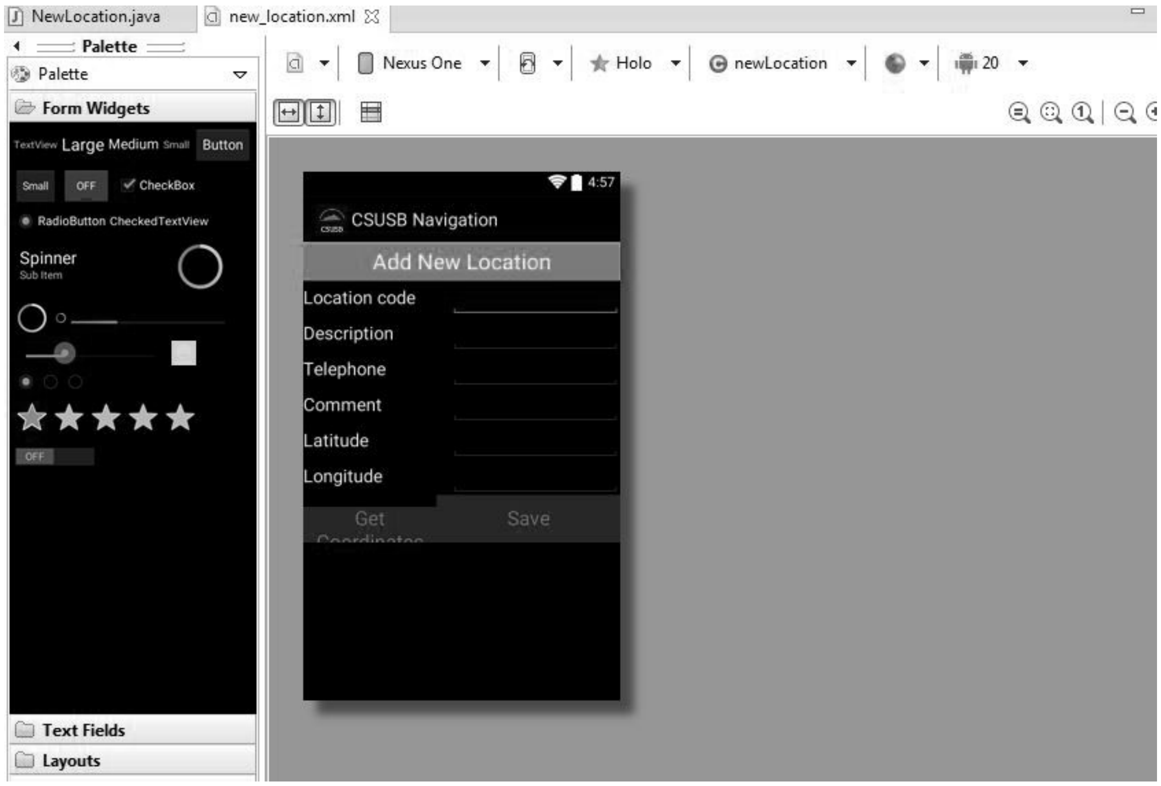
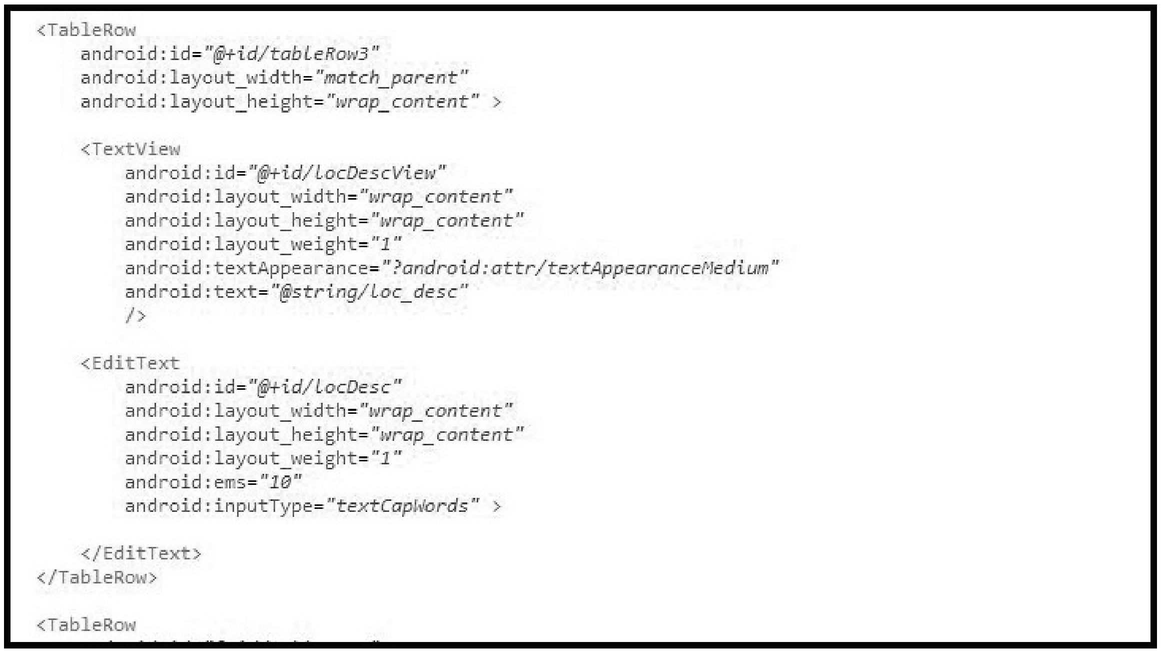


Fig. 4.25: New Location Graphic Layout in Eclipse

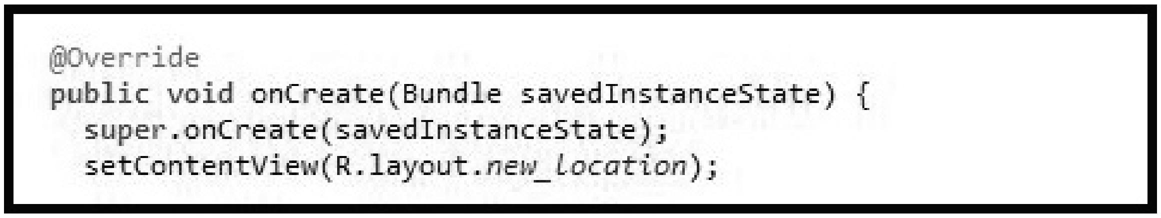
Internally this layout is constructed by one row tables of text views and edit fields as follows. See Figure 4.26.



```
<TableRow
    android:id="@+id/tableRow3"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" >

    <TextView
        android:id="@+id/locDescView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:text="@string/loc_desc"
        />

    <EditText
        android:id="@+id/locDesc"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:ems="10"
        android:inputType="textCapWords" >

    </EditText>
</TableRow>

<TableRow
```

Fig. 4.26: New Location Layout Code

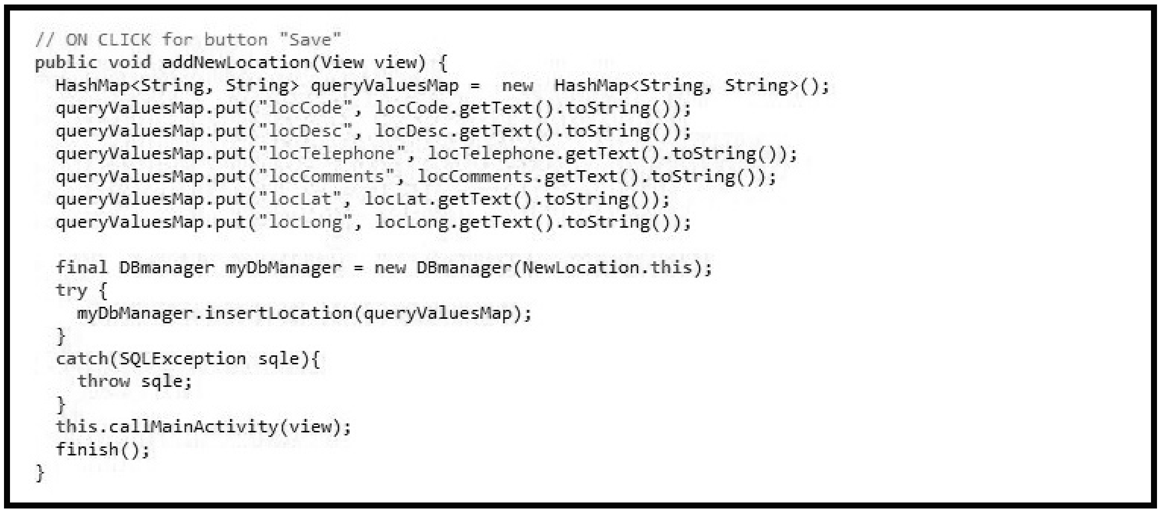
This is how a Java Android class relates to its layout, in this case NewLocation to layout new-location. See Figure 4.27.



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

Fig. 4.27: New Location Declaring Layout

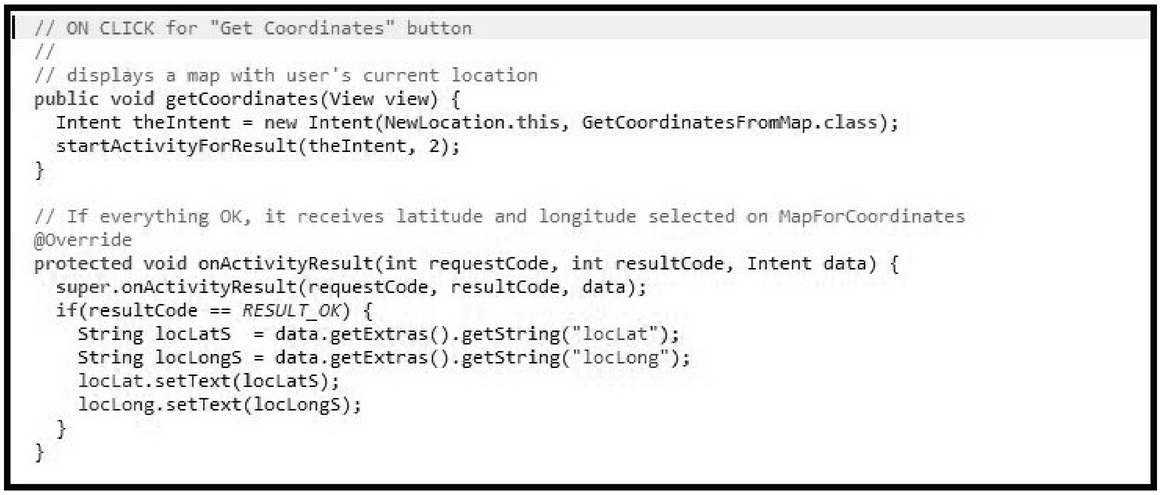
The user inputs the location information and press the Save button, the method addNewLocation calls the corresponding method on the DBmanager class that will insert a new location in table. See Figure 4.28.



```
// ON CLICK for button "Save"
public void addNewLocation(View view) {
  HashMap<String, String> queryValuesMap =  new  HashMap<String, String>();
  queryValuesMap.put("locCode", locCode.getText().toString());
  queryValuesMap.put("locDesc", locDesc.getText().toString());
  queryValuesMap.put("locTelephone", locTelephone.getText().toString());
  queryValuesMap.put("locComments", locComments.getText().toString());
  queryValuesMap.put("locLat", locLat.getText().toString());
  queryValuesMap.put("locLong", locLong.getText().toString());

  final DBmanager myDbManager = new DBmanager(NewLocation.this);
  try {
    myDbManager.insertLocation(queryValuesMap);
  }
  catch(SQLException sqle){
    throw sqle;
  }
  this.callMainActivity(view);
  finish();
}
```

Fig. 4.28: Saving a New Location on Locations Table

In case the user don't remember latitude and longitude of a location, then class Get Coordinates will solve this. If everything is correct, latitude and longitude will be retrieved. The following code shows that portion of the program. See Figure 4.29.



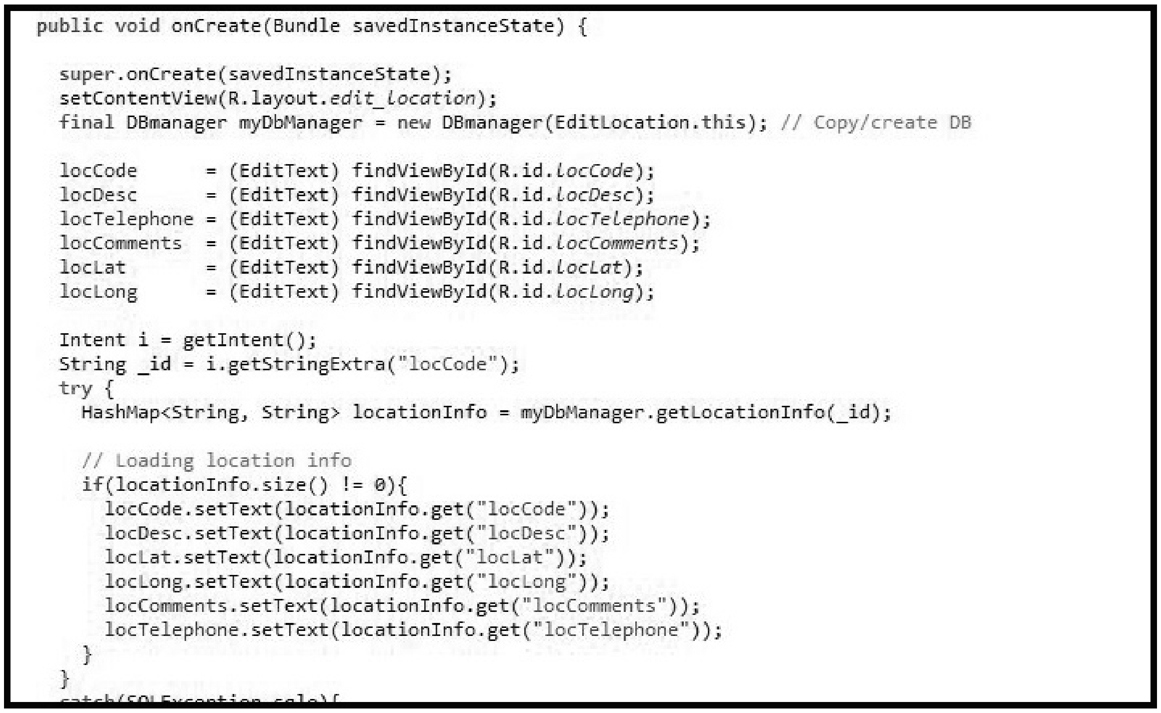
```
// ON CLICK for "Get Coordinates" button
//
// displays a map with user's current location
public void getCoordinates(View view) {
  Intent theIntent = new Intent(NewLocation.this, GetCoordinatesFromMap.class);
  startActivityForResult(theIntent, 2);
}

// If everything OK, it receives latitude and longitude selected on MapForCoordinates
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
  super.onActivityResult(requestCode, resultCode, data);
  if(resultCode == RESULT_OK) {
    String locLatS  = data.getExtras().getString("locLat");
    String locLongS = data.getExtras().getString("locLong");
    locLat.setText(locLatS);
    locLong.setText(locLongS);
  }
}
```

Fig. 4.29: Looking for Coordinates

That wraps up New Location class.

4.5 *Edit Location Class*

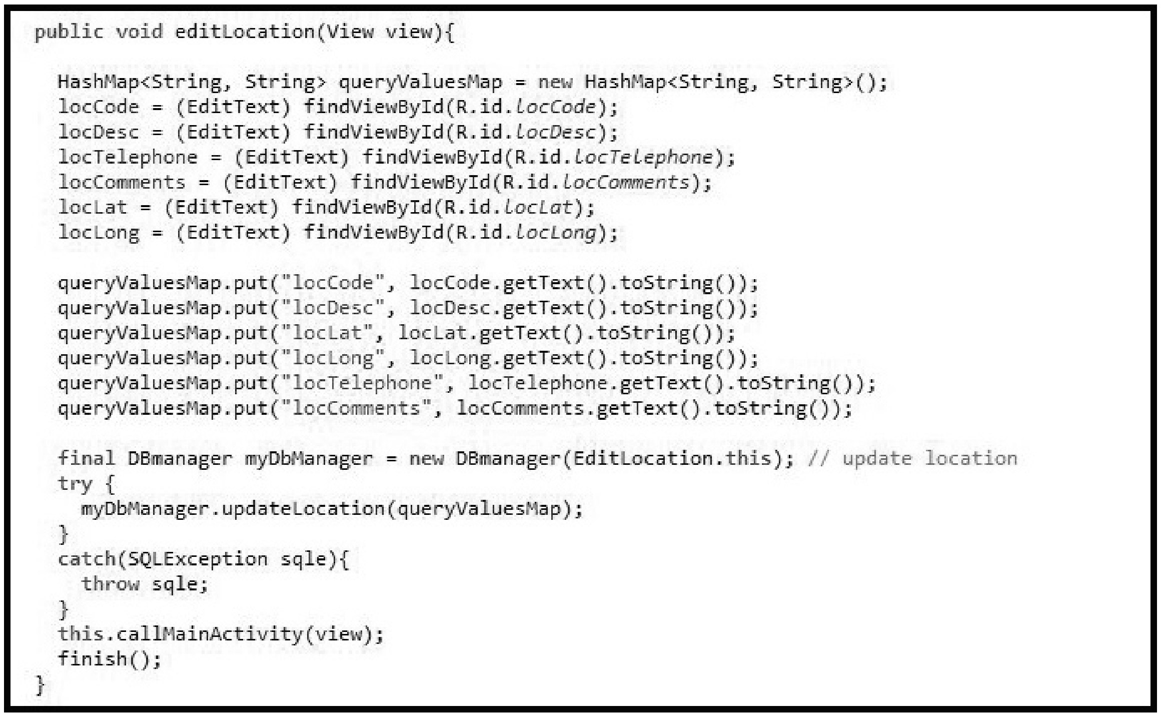
EditLocation class is to modify a specific location's data. It receives location ID as a parameter and looks up its information into table locations; then it uses a hashmap as a parameter for class DBmanager to get the location info. See Figure 4.30.



```
public void onCreate(Bundle savedInstanceState) {

  super.onCreate(savedInstanceState);
  setContentView(R.layout.edit_location);
  final DBmanager myDbManager = new DBmanager(EditLocation.this); // Copy/create DB

  locCode      = (EditText) findViewById(R.id.locCode);
  locDesc      = (EditText) findViewById(R.id.locDesc);
  locTelephone = (EditText) findViewById(R.id.locTelephone);
  locComments  = (EditText) findViewById(R.id.locComments);
  locLat       = (EditText) findViewById(R.id.locLat);
  locLong      = (EditText) findViewById(R.id.locLong);

  Intent i = getIntent();
  String _id = i.getStringExtra("locCode");
  try {
    HashMap<String, String> locationInfo = myDbManager.getLocationInfo(_id);

    // Loading location info
    if(locationInfo.size() != 0){
      locCode.setText(locationInfo.get("locCode"));
      locDesc.setText(locationInfo.get("locDesc"));
      locLat.setText(locationInfo.get("locLat"));
      locLong.setText(locationInfo.get("locLong"));
      locComments.setText(locationInfo.get("locComments"));
      locTelephone.setText(locationInfo.get("locTelephone"));
    }
  }
```

Fig. 4.30: Look Up for Location Information

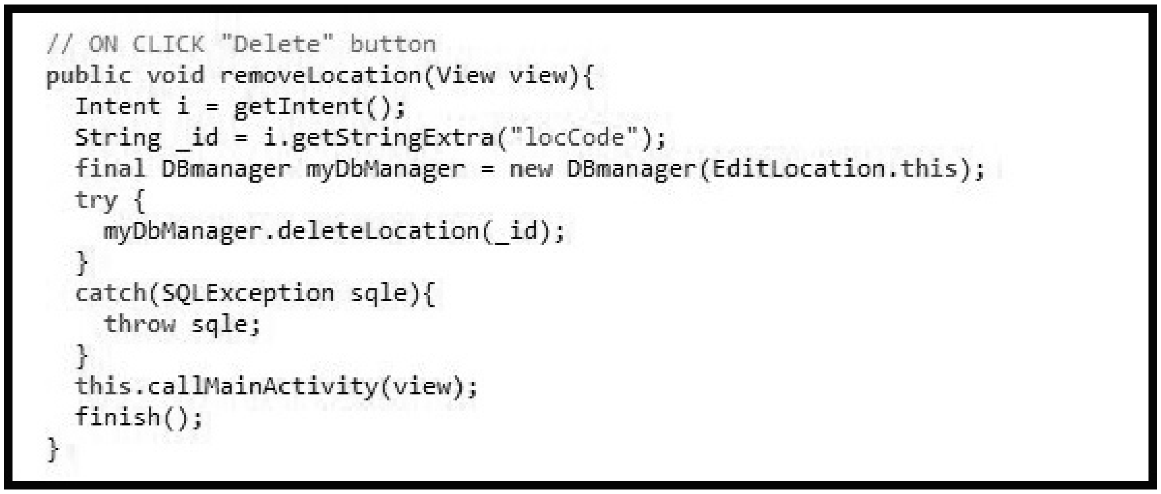
Any modified information will be saved by pressing the button "Save". All location information is handled to DBmanager; it will update location info. See Figure 4.31.



```
public void editLocation(View view){

  HashMap<String, String> queryValuesMap = new HashMap<String, String>();
  locCode = (EditText) findViewById(R.id.locCode);
  locDesc = (EditText) findViewById(R.id.locDesc);
  locTelephone = (EditText) findViewById(R.id.locTelephone);
  locComments = (EditText) findViewById(R.id.locComments);
  locLat = (EditText) findViewById(R.id.locLat);
  locLong = (EditText) findViewById(R.id.locLong);

  queryValuesMap.put("locCode", locCode.getText().toString());
  queryValuesMap.put("locDesc", locDesc.getText().toString());
  queryValuesMap.put("locLat", locLat.getText().toString());
  queryValuesMap.put("locLong", locLong.getText().toString());
  queryValuesMap.put("locTelephone", locTelephone.getText().toString());
  queryValuesMap.put("locComments", locComments.getText().toString());

  final DBmanager myDbManager = new DBmanager(EditLocation.this); // update location
  try {
    myDbManager.updateLocation(queryValuesMap);
  }
  catch(SQLException sqle){
    throw sqle;
  }
  this.callMainActivity(view);
  finish();
}
```

Fig. 4.31: Update a Location Information

It is also possible to eliminate a location. If the user decides to do this, button "Delete" should be pressed. The following code will delete selected location. See Figure 4.32.



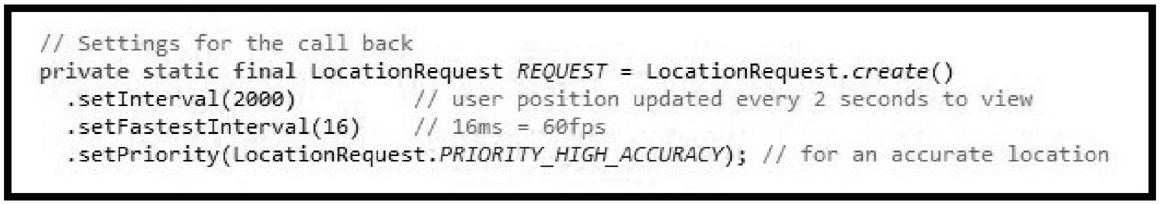
```
// ON CLICK "Delete" button
public void removeLocation(View view){
  Intent i = getIntent();
  String _id = i.getStringExtra("locCode");
  final DBmanager myDbManager = new DBmanager(EditLocation.this);
  try {
    myDbManager.deleteLocation(_id);
  }
  catch(SQLException sqle){
    throw sqle;
  }
  this.callMainActivity(view);
  finish();
}
```

Fig. 4.32: Deleting a Location

That is in general the structure of the Edit Location class.

4.6 Get Coordinates Class

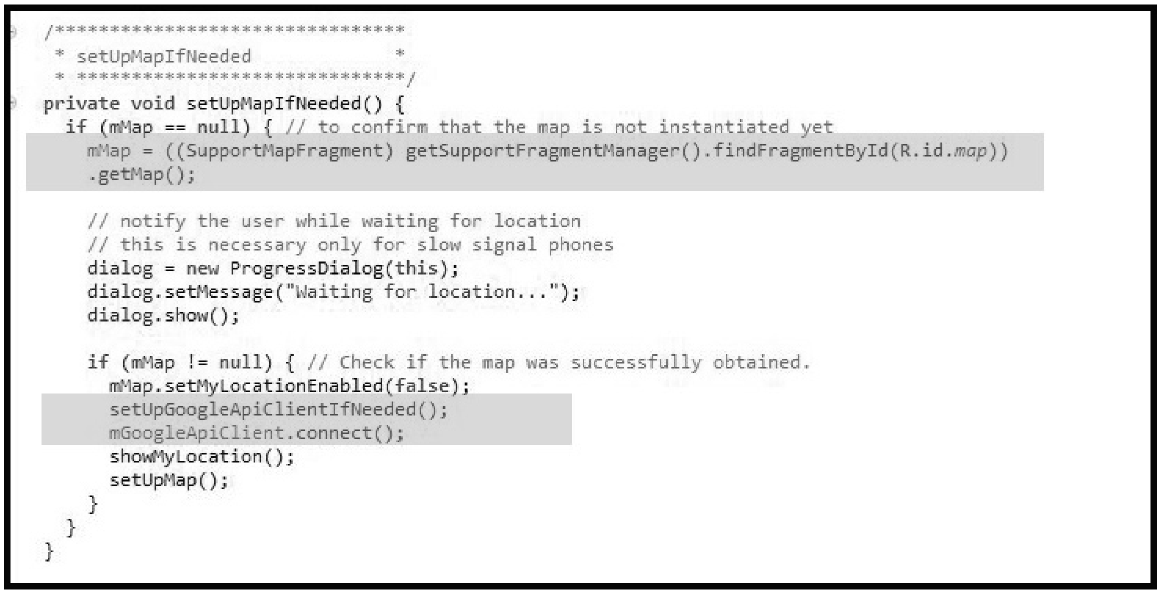
Get Coordinates class is used by the NewLocation and the EditLocation classes. Attributes Latitude and Longitude are decimal numbers with several digits, therefore it might be quite possible that user needs some help with them. Both, edit and insert location layouts, have a Get Coordinates button. By pressing this button a map will be displayed. From this map, the user can select a location by making a long click on the map. Coordinates latitude and longitude will be returned to the caller routine. This class has a similar structure than the DisplayTrackingMap class. It defines a fragment that will hold a map. A request for the callback is defined as well. The values for this request might be different though, since it's not necessary to check a location status changed frequently. It's just necessary to detect a long click and transform it into latitude and longitude values. These values will be returned them to the caller routine. See Figure 4.33.



```
// Settings for the call back
private static final LocationRequest REQUEST = LocationRequest.create()
  .setInterval(2000)         // user position updated every 2 seconds to view
  .setFastestInterval(16)    // 16ms = 60fps
  .setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY); // for an accurate location
```

Fig. 4.33: Defining Callback Parameters

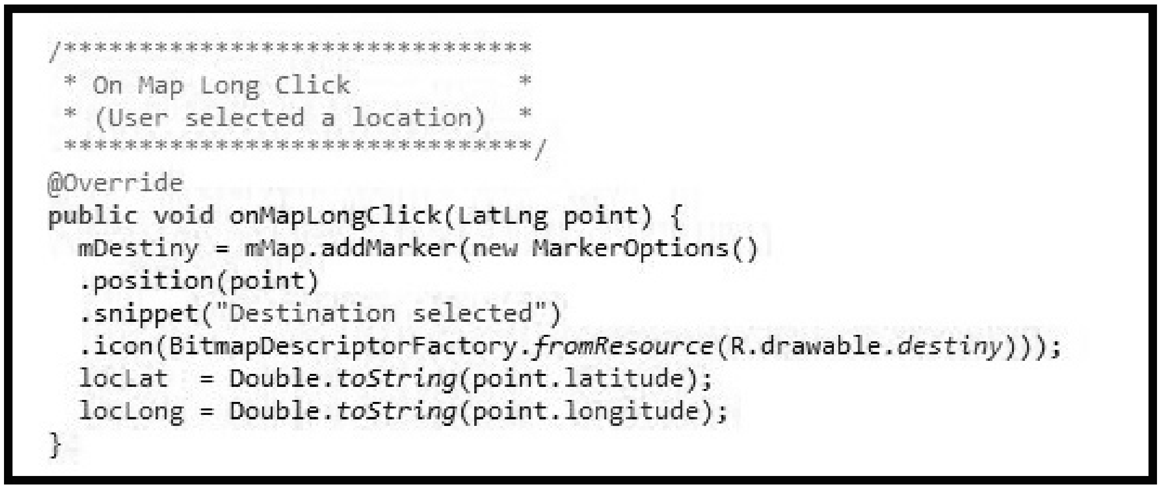
The parameters request for the callback function. Intervals to determine the frequency of time to check for any possible variation on the map, such as a long click over the map. A high priority request means that the most precise tool for locating a coordinate must be used. This accuracy is controlled by Google Maps[2]. See Figure 4.34.



```
/******************************
 * setUpMapIfNeeded            *
 * ****************************/
private void setUpMapIfNeeded() {
  if (mMap == null) { // to confirm that the map is not instantiated yet
    mMap = ((SupportMapFragment) getSupportFragmentManager().findFragmentById(R.id.map))
    .getMap();

    // notify the user while waiting for location
    // this is necessary only for slow signal phones
    dialog = new ProgressDialog(this);
    dialog.setMessage("Waiting for location...");
    dialog.show();

    if (mMap != null) { // Check if the map was successfully obtained.
      mMap.setMyLocationEnabled(false);
      setUpGoogleApiClientIfNeeded();
      mGoogleApiClient.connect();
      showMyLocation();
      setUpMap();
    }
  }
}
```

Fig. 4.34: Connecting Map Layout to Activity (Class)

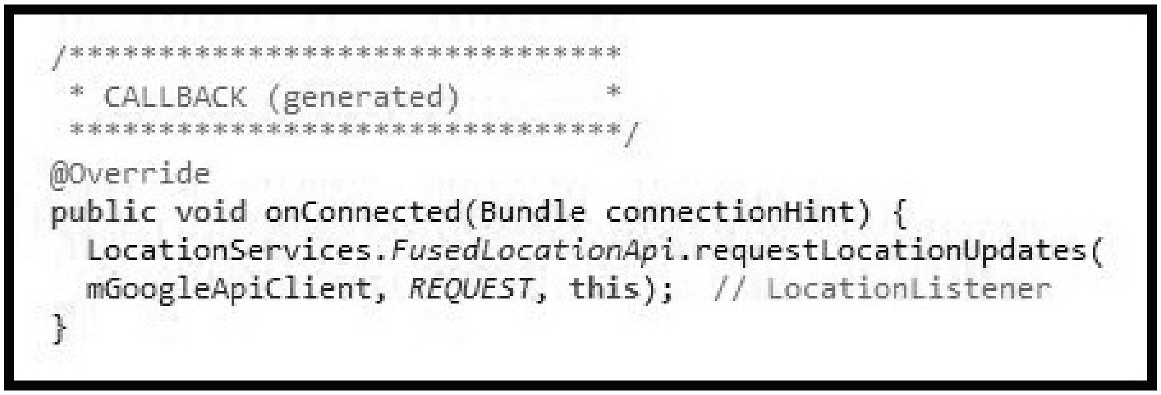
This is how the class relates to its layout in order to use a fragment that will hold the map, and it connects to Google-maps[2]. See Figure 4.35.



```
/******************************
 * On Map Long Click          *
 * (User selected a location)  *
 ******************************/
@Override
public void onMapLongClick(LatLng point) {
  mDestiny = mMap.addMarker(new MarkerOptions()
  .position(point)
  .snippet("Destination selected")
  .icon(BitmapDescriptorFactory.fromResource(R.drawable.destiny)));
  locLat  = Double.toString(point.latitude);
  locLong = Double.toString(point.longitude);
}
```

Fig. 4.35: On Long Click Select Lat. and Long

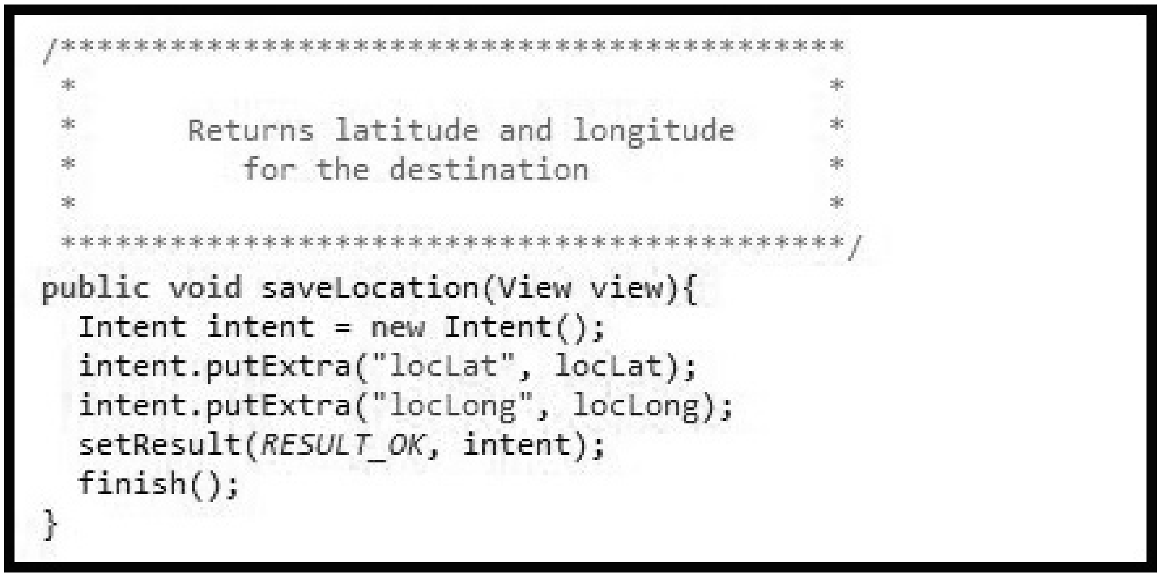
When a long click over the map is detected a red arrow will be drawn on the map to indicate that a location has been detected. A Google method: addMarker is used for that. See Figure 4.36.



```
/*******************************
 * CALLBACK (generated)        *
 *******************************/
@Override
public void onConnected(Bundle connectionHint) {
  LocationServices.FusedLocationApi.requestLocationUpdates(
  mGoogleApiClient, REQUEST, this);  // LocationListener
}
```

Fig. 4.36: onConnected Method requestLocationUpdate

Method requestLocationUpdate is to connect the callback function with the on-LocationChanged method. See Figure 4.37.



```
/****************************************
 *                                      *
 *      Returns latitude and longitude  *
 *         for the destination          *
 *                                      *
 ****************************************/
public void saveLocation(View view){
  Intent intent = new Intent();
  intent.putExtra("locLat", locLat);
  intent.putExtra("locLong", locLong);
  setResult(RESULT_OK, intent);
  finish();
}
```

Fig. 4.37: Return Latitude and Longitude

4.7 Show Location Info Class

The locationInfo class is a window with an specific location information that can be reached from classes newLocation and editLocation. Figure 4.38 shows its graphic layout:

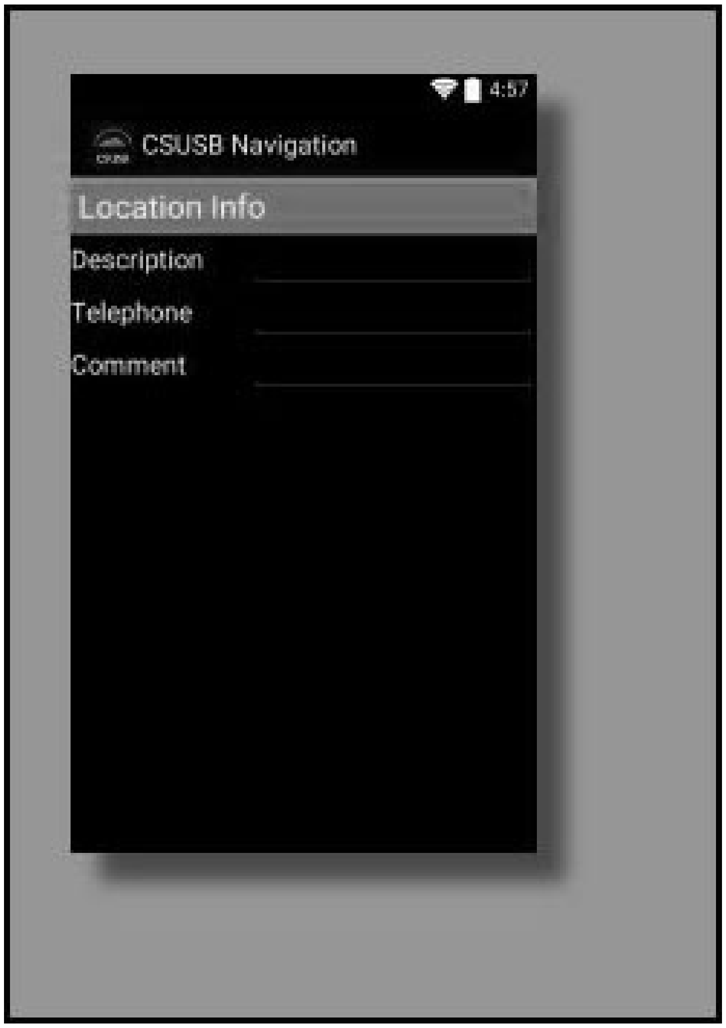
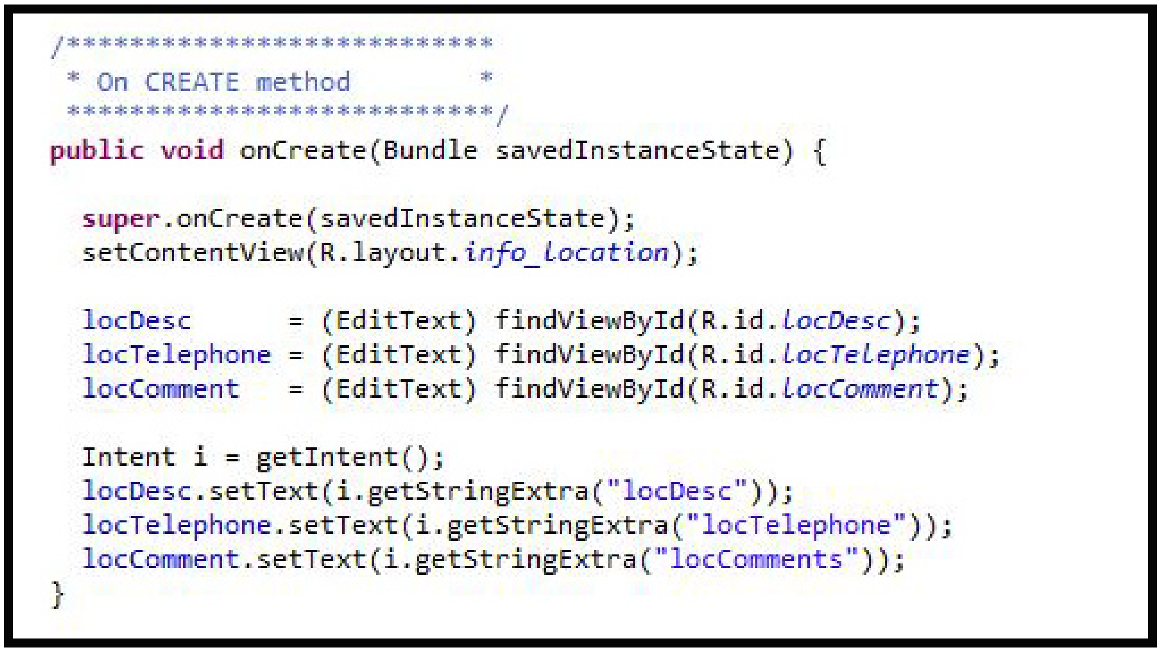


Fig. 4.38: Location Info Layout

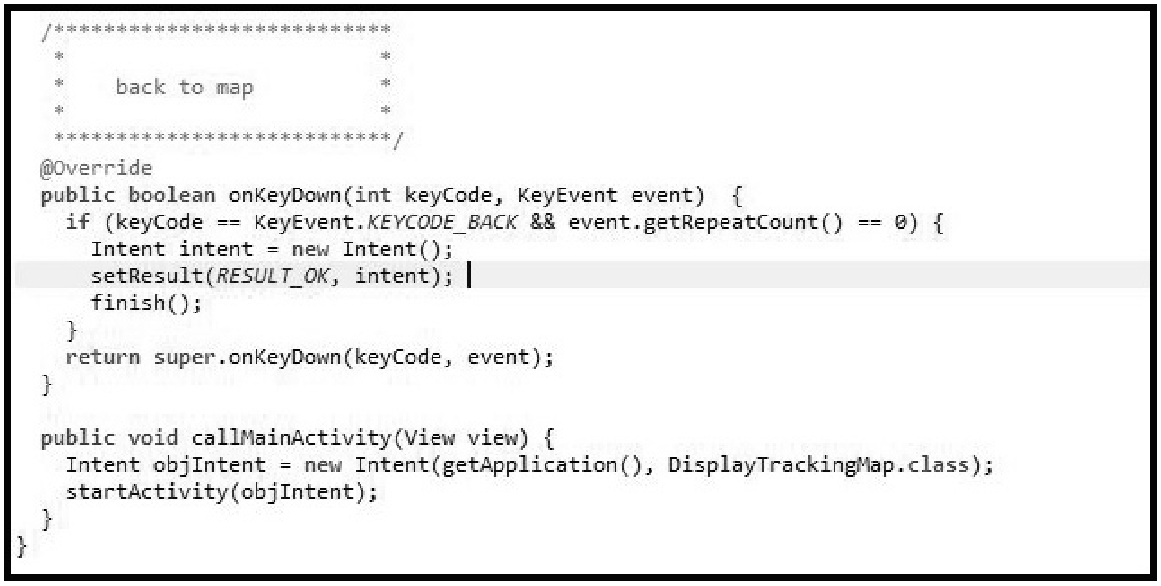
Class ShowLocationInfo define the layout used to display the info. See Figure 4.39.



```
/***************************
 * On CREATE method        *
 ***************************/
public void onCreate(Bundle savedInstanceState) {

  super.onCreate(savedInstanceState);
  setContentView(R.layout.info_location);

  locDesc      = (EditText) findViewById(R.id.locDesc);
  locTelephone = (EditText) findViewById(R.id.locTelephone);
  locComment   = (EditText) findViewById(R.id.locComment);

  Intent i = getIntent();
  locDesc.setText(i.getStringExtra("locDesc"));
  locTelephone.setText(i.getStringExtra("locTelephone"));
  locComment.setText(i.getStringExtra("locComments"));
}
```

Fig. 4.39: Displaying Location Information

When the user is ready to leave the location information window the go back button control will return control to the last active screen seen. See Figure 4.40.

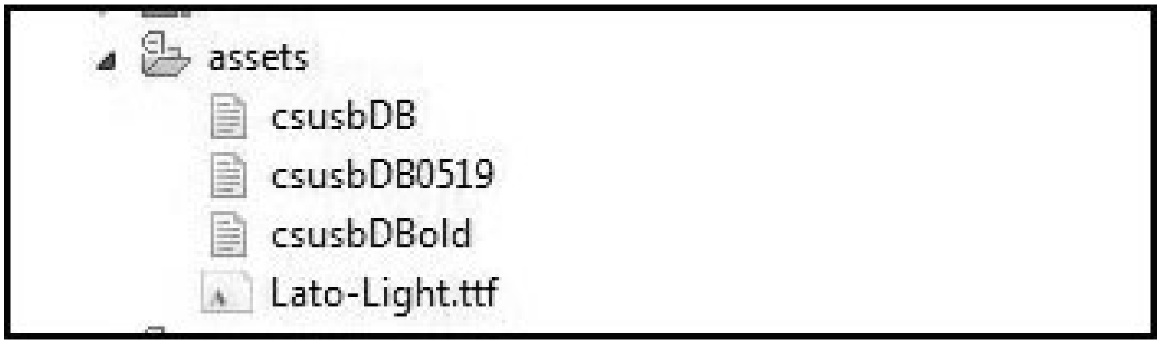


```
/**************************
 *                        *
 *    back to map         *
 *                        *
 **************************/
@Override
public boolean onKeyDown(int keyCode, KeyEvent event)  {
  if (keyCode == KeyEvent.KEYCODE_BACK && event.getRepeatCount() == 0) {
    Intent intent = new Intent();
    setResult(RESULT_OK, intent);
    finish();
  }
  return super.onKeyDown(keyCode, event);
}

public void callMainActivity(View view) {
  Intent objIntent = new Intent(getApplication(), DisplayTrackingMap.class);
  startActivity(objIntent);
}
}
```

Fig. 4.40: Returning to Caller

4.8 *Changing Font for the Entire Application*

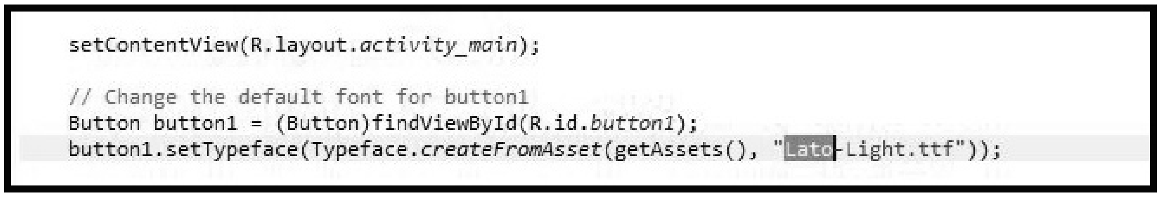
This should not be treated as a class itself, however I believe it is important to be mentioned because of the particular process to change the font for an Android application. This is the way to do it under Android development[2]. First, import the font into the assets folder of the project, Lato-Light.ttf in this case. See Figure 4.41.



```
assets
    csusbDB
    csusbDB0519
    csusbDBold
    Lato-Light.ttf
```

Fig. 4.41: Font Imported Into Project Assets

Then the font must be defined in the activity you need to use it. See Figure 4.42.



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

// Change the default font for button1
Button button1 = (Button)findViewById(R.id.button1);
button1.setTypeface(Typeface.createFromAsset(getAssets(), "Lato-Light.ttf"));
```

Fig. 4.42: Font Defined to be Used in a Button

That wraps up the app software architecture.

5. CONCLUSION AND FUTURE DIRECTION

5.1 Conclusion

It has been very interesting to learn about Android development. I read the book "Programming Android" by Zigurd Mednieks, and Laird Dornin[3]. I also took the on line class: "Programming Mobile Services for Android Handheld Systems" offered by The University of Maryland USA in order to get acquainted with the environment of Android, and specifically with Google maps[8]. In the end, one of the deliverables is an app, which is developed and it's ready to be deployed. Another deliverable is this document, which explains the most relevant details of this specific development. The application is fully functional on Android mobile devices and it can be implemented if CSUSB chooses to. On a personal level I acquired some knowledge on a very marketable field as Android development is. It's a good starting point before entering in the development of more challenging applications.

5.2 Future Direction

Android development and Google maps offer a lot of functionality that could be additionally implemented on the application. The database can be manipulated as well according to new requirements if needed. Of course, it can also be used as it is right now.

Besides, it is also possible to standardize the application so it could be used in places other than CSUSB. I, for instance, tested it on the community I live in.

REFERENCES

[1] B. Bernd and D. Allen. *Object-Oriented Software Engineering.* NJ Pearson., 2010.

[2] Stack Overflow for Android Developers Web Page. (undated). [Online]. Viewed 2015 January 5. Available: http://stackoverflow.com/.

[3] D Laird M. Zigurd and M.Nakamura. *Programming Android.* O Reilly., 2011.

[4] Android Developers API Guides Web Page. (undated). [Online]. Viewed 2015 February. Available: http://developer.android.com/guide/index.html.

[5] Blue Stacks Home Page. (undated). [Online]. Viewed 2015 February. Available: http://www.bluestacks.com/.

[6] Google Maps Api Developers Home Page. (undated). [Online]. Viewed 2015 February. Available: https://developers.google.com/maps/android/.

[7] Map Quest Web Page. (undated). [Online]. Viewed 2015 February. Available: http://https://developer.mapquest.com/.

[8] Online Course by the University of Maryland Programming Mobile Services for Android Handheld Systems: Concurrency. (undated). [Online]. Viewed 2015 February. Available: https://www.coursera.org/course/posaconcurrency.

[9] Yahoo Boss Geo Services web Page. (undated). [Online]. Viewed 2015 February. Available: https://developer.yahoo.com/boss/geo/.