

School of Media Arts and Technology

BSc Software Engineering

**Matthew Dear**

Q10232711

**My Approach to Developing the Places to Stay Mobile Application**

Assignment 1

**Mobile Application Development**

Tutor: Craig Gallen 19th May 2020

Contents

[XML layout files 3](#_Toc40362916)

[findViewById method 3](#_Toc40362917)

[Communication between activities 4](#_Toc40362918)

[Preferences 4](#_Toc40362919)

[File I/O 5](#_Toc40362920)

[Network communication 5](#_Toc40362921)

[CSV parsing 6](#_Toc40362922)

# XML layout files

XML layout files are used to define the objects that need to appear in the UI (User Interface) for a given activity and how those objects should be laid out on screen. This project contains two XML layout files activity\_main.xml and add\_new.xml. Activity\_main.xml is called when the default activity, main activity, is created. It starts with a linier layout that matches the width and height of the parent in this case the screen. It is also in the vertical orientation so that the items nested inside it will appear in a vertical stack. I did this to make it easier to add items in the future if required. This linier layout contains a MapView that is a custom osmdroid object, designed to display a map on screen. The hight and width are both set to match parent this means the map will be the same size as the linier layout as this is its parent. Resulting in the map filling the entire screen as the linier layout is set to match the size of the screen. The android clickable attribute is set to allow you to click on the map and move it around. Also, the MapView has been given and id using the Android id attribute allowing you to reference it from methods to update its centre point.

The add\_new.xml file is called when the add new activity is created. It starts with a LinierLayout, like the activity\_main.xml file, that contains five linier layouts in a horizontal orientation. Each has an id, fixed hight of 50dp, margin of 10dp, and a width of match parent making them the full width of the screen. The first linier layout contains a TextView that provide instructions to the user on how to add a new place to stay, this is pulled in from the strings.xml file using the text attribute and the @string/stringId command. The next three each contain a TextView displaying a label for the input pulled in from the strings.xml file. All the TextViews have the width attribute set to wrap\_content allowing the EditText also included in the LinierLayout to be displayed alongside it. Each of the edit texts have the width attribute set to fill parent forcing it to fill up all the remaining space not taken up by the TextView. They also have the input type attribute set to help validate the input from the user, for example the name and type inputs are set to TextCapWords allowing only upper and lowercase letters to be entered while capitalising the first letter of each word automatically. The price input is set to NumberDecimal changing the keyboard displayed to the user so that it is easier to enter numbers and blocking letters from being entered. The last linierLayout has the gravity attribute set to center the contained button that has the width attribute set to 350dp so it does not fill the screen width.

# findViewById method

This method is used to retrieve objects from the UI so that you can interact with them from methods. This is done by calling the global R object, then calling the ID method and adding the UI object ID, defined in the layout.xml file, and passing it as an argument to the findByViewId method. This method is used once in the MainActivity.java file to retrieve the MapView from the UI so the maps centre point can be updated by the GPS. It has also been used five times in the AddNewActivity.java file, the first two times are in the onCreate method. The first time it is being used to retrieve the AutoCompleteTextView UI element so a custom auto complete list can be added to it. The second time it is used to retrieve the add place button so that the setOnClickListener method can be called on it, allowing the onClick method to be called when the button is pressed. It is then used three times in the AddNewActivitys onClick method to retrieve the name and price EditTexts and the AutoCompleteTextView UI elements.

# Communication between activities

To facilitate communication between the MainActivity and the AddNewActivity I have used intents and bundles. Intents are used to call other activities allowing a response code to be attached so the MainActivity can process the response on the activity’s completion. A bundle is an object that stores string value pairs allowing data to be sent to the parent activity. When the add new place option is selected in the menu a new intent is created taking the current activity and the AddNewActivity class as arguments. This allows the system to know what activity should be started and where it should return once completed. It then uses the MainActivitie’s startActivityForResult method inputting the intent and a numeric value as the request code.

The addNewActivity is then called allowing the user to input the name, type, and price details for the new place to stay. When the user clicks the add place button, input validation is done before, a new bundle and intent are created. Then the name, type, and price values are passed in as arguments to the putString or putDouble method, along with a string key, called on the new bundle. Then the putExtras method is called on the new intent passing the bundle as an argument, before calling the setResult method passing RESULT\_OK and the intent as arguments. Then the finish method is called closing the activity returning the intent to the MainActivity. Once returned the onActivityResult method is called passing in the request code, result code, and the returned intent. This method will then us an if statement to check the request code if it is 0 it will then check if the result code is RESULT\_OK. If it is it will then check that the bundle is not null if it is not null the name, type, and price will be extracted. This will be done by calling the getString or getDouble method passing the corresponding string key as an argument allowing a new place object to be created.

# Preferences

Preferences are used to persist information like usernames for the next time an application is run. In Android this is done using the savedInstanceState bundle object. It can be accessed by any activity if it is passed as an argument into its onCreate method. It can also be update by any activity using its onSaveInstantState method. This bundle is then stored in the local file system by the operating system. In this project preferences have been used to allow the user to turn on/off automatic saving of places to stay. To do this I created a PreferencesActivity.java file and in its onCreate method I passed the savedInstanceState bundle as an argument. It then uses the addPreferencesFromResource method taking the preferences.xml file as an argument allowing the user to see the current status and update the preference. In the onResume method of MainActivity the getDefaultSharedPreferences method is called on the PreferenceManager object, passing the getApplicationContext method as an argument, and saving the returned bundle in a variable. Then the getBoolean method is called on the bundle passing a string key and default value, for if nothing is returned, as arguments. The returned value is then stored in a variable and used in the process of adding a new place if true the place will be saved automatically.

# File I/O

For file I/O Android uses four different types of streams: input, output, low-level, and wrapper. They can be used in many different combinations to achieve the required outcome. An example of an input stream is data coming from you keyboard and an example of an output stream is data going to your screen. A low-level stream reads or writes individual bytes to files or hardware components and a wrapper stream can convert files between types, work with ASCII characters, and convert RAW data into data types or objects. However, to do this it will need to contain a low-level stream that is passed in as an argument to the constructor.

In this project I have used file I/O to read and write a list of places to stay to the local file system so that they can be used when the device is offline. I have a method called savePTSLocally this uses the low-level stream FileWritter to write to the file system. This FileWritter takes the Environment.getExternalStorageDirectory allowing it to know the root directory of the device, it also takes the custom file name as arguments in the constructor. The FileWritter is then passed in as an argument to the constructor of the wrapper stream PrintWriter object used to convert ASCII characters to and from RAW bytes of data. Once the data has been manipulated and is ready to be written to file the printLn() method is called passing the data string as an argument. Finally, once all data has been written the close method is called on the PrintWritter object closing the output stream down completely.

Strems have also been used in the loadLocalPTS method this time the low-level stream is the FileReader object taking the Environment.getExternalStorageDirectory method and custom file name as the arguments for the constructor. The FileReader object is then passed in as an argument to the constructor for the BufferedReader wrapper steam. These streams are then used to read in ASCII characters from the local file and are converted into place objects using the BufferedReader read line method. Once the file has been read fully the close method is called on the BufferedReader object closing both the wrapper and low-level streams down completely. The InnerRemoteLoad and InnerRemoteSave methods both use the InputStreamReader object as the low-level stream when receiving network communications and the BufferedReader object to convert the input data to an object or string, respectively. Using the BufferedReader read line and close methods in the same way as the loadLocalPTS method.

# Network communication

In Android, all methods that do network communication should be in a separate inner class that extends the functionality of asyncTask. This allows the network communication to run in another thread alongside the UI thread and stops it from hanging the UI thread and being for quit. Also, making it an inner class allows it to interact with the variables and methods of the parent class. All network communication is done wrapped in a try catch block to stop the application from crashing if the server does not respond fast enough or is completely unavailable. In this project I have implemented two classes the first being InnerRemoteLoad used to retrieve data from a remote server using GET requests. The second being InnerRemoteSave used to save data to the remote server using POST requests.

InnerRemoteLoad

This method uses a URL object to store the address and GET query string. In this application this is hard coded however in the real world you would use variables to store the elements of the query string and pass them into the URL object. The openConnection method is then called on the URL object and it is cast as a HttpURLConnection to the variable conn. The getInputStream method is then called on conn storing it in an input stream variable. An if statement is then used to check the get request code is 200 if it is not the error is returned as the result of the method. If it is 200 then the input steam is passed in as an argument to the InputStreamReader object spoken about in the file I/O section. To make sure that the async task can be closed down correctly I have added the finally block to my try catch block this is called weather the code completes or errors and calls the disconnect method on the conn variable fully closing down the network stream.

InnerRemoteSave

This works in exactly the same way as the InnerRemoteSave method with one difference when the conn variable is created the getOutputStream method is called on it saving the returned OutputStream obeject as a variable before calling the write method on it and passing the post data sting as an argument. This send the pre generated postData string variable to the remote server allowing the place to be saved.

# CSV parsing

In this application I have used a CSV (Comma Separated Values) file to store the places locally on the device. To be able to read places from the file they will need to be converted from a string to a place object and to be able to write places to the file they will need to be converted from a place object to a string this is called CSV parsing. The savePTSLocally method parses place objects and converts the to a line of comma separated values ready to be saved to a file and the LoadPTSLocally method parses a line of comma separated values and converts them into a place.

Parsing objects to CSV

A for each loop is used to iterate thought the placesToStay array. For each place it then calls the: getName, getType, getPrice, getLatitude, and getLongitude methods storing the returned values in variables with the same names. These variables are then passed into the printLn method called on the PrintWritter wrapper stream, with commas added in between each one to separate the data, saving them to the file. This means that one object is stored on each line of the CSV file with commas in between the values allowing them to be individually pulled out when parsed back to an object.

Parsing CSV to objects

This uses a while loop that runs until the readLine method called on the BufferedReader stream returns null saving each line to a string variable. The split method is then called on that variable passing in a comma as the argument. This means the next piece of data will be extracted until a comma is read, then it will then be stored in the indexed array variable. An if statement is then used to check the size of the indexed array once the size is 5 the if statement is triggered. Then two string variables are created name and type and three double variables are created: price, latitude, and longitude. Then the corresponding components extracted from the csv file are stored in these variables. Finally, a new place is created by passing the name, type, price, latitude, and longitude variables in as arguments to the constructor method before, calling the add function on the placesToStay array passing the newly created place as an argument adding it to the array.