

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](#_Toc40262944)

[findViewById() method 4](#_Toc40262945)

[Communication between activities 4](#_Toc40262946)

[Preferences 4](#_Toc40262947)

[File I/O 5](#_Toc40262948)

[Network communication 6](#_Toc40262949)

[CSV parsing 6](#_Toc40262950)

# XML layout files

This project contains four XML layout files: activity\_main.xml, add\_new.xml, menu.xml, and preferences.xml. All are used to define the items that should be on each screen and how they should be laid out. Activity\_main.xml is called when the default activity, main activity, is created. It starts with a linier layout in the vertical orientation that matches the width and height of the parent, in this case the screen. I have done this to make it easier to add more to this screen later if needed. This linier layout then contains a MapView that is a custom osmdroid object that allows a map to be displayed. This is set to match the parent, in this case the linier layout as this is the only content needed on this screen. The android clickable attribute is set to allow you to click on the map and move it around. Finally, I have used the id attribute to give the MapView an id so I can reference it in other files later.

The add\_new.xml starts with a linier layout in a vertical orientation that contains 5 nested linier layers in horizontal orientation. These linier layouts each have an id, a fixed hight of 50dp, margin of 10dp, and a width of match parent. In the first linier layout I have a TextView containing instructions for the user that are pulled in from the strings.xml file using the text attribute and the @string/stringId command. The next three then each contain a TextView displaying a label for the input pulled 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 them. Each of the edit texts have an input type attribute set to make it so only certain characters can be entered. For the name and type inputs it is set to TextCapWord and for the price it is set to NumberDecimal. Finally, in the last one there is a Button with a fixed width of 350dp to stop the button from filling the entire width of the screen.

The preferences.xml layout file contains a CheckBoxPreference object. This has four attributes set. The first being the default value this is set to false meaning that unless overridden by the user this will remain false. The second being the key attribute, this is the string reference that can be used to access the stored value in another file e.g. MainActivity. Then the summary attribute is set this is sets the value of the small line of text that will appear under the title and the title attribute that sets the value for the bold heading of the preference. These both use the @string/stringId method to pull in the value from the Strings.xml file.

Menu.xml

This start with the open and close menu tags and then nested inside that are 6 item objects. All items have two attributes set an id and title. The title is set using the @string method that pulls the value in from the strings.xml file.

# findViewById() method

The findViewById method is used to retrieve a user interface object that you would like to interact with in your code. E.g. an EditText. To do this you will need to pass in the items ID as an argument, this is done by calling the global R object, then calling the ID method before giving the UI items ID. This would look something like this

mv = findViewById(R.id.map1);

This has been taken from the MainActivity.java for this application and is used to retrieve the MapView from the layout so the maps centre point can be updated by the GPS.

It has also been used a couple of times in the AddNewActivity.java file. The first two times it is in the activities 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 UI element 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 onClick method to retrieve the name and price edit texts and the AutoCompleteTextView UI elements.

# Communication between activities

To allow for 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 its return. A bundle is an object that stores string keys and maps them to values to allow them to be sent to other activities. When the add new place, button is pressed in the menu a new intent is created taking the current activity as and the class of the activity you would like to open. You then use the startActivityForResult method passing the intent and a request code in this case 0. The AddNew activity is then created allowing a user to input a name, type, and price for the new place to stay they would like to add to the map. When the user clicks add place validation checks are then run on the input information. If all information is correct, a new bundle and new intent are created. Then the input name and type are stored in the bundle using the putString method passing a string key and the respective variable. The same is then done for price but the putDouble function is used instead. Once complete the bundle is added to the intent using the putExtras method and passing it as an argument. The setResult method is then used to set the result message to OK on the intent. Then the finish function is called closing the activity and 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 then uses three if statement. The first is to check the request code is 0 if it is then it will check if the result code is RESULT\_OK. Then finally it will extract the extras and check they are not null before extracting the name, type, and price allowing it to create a new place object.

# Preferences

Preferences are used to persist certain information (e.g. username) for the next time an application is run using the built in Android savedInstanceState bundle object. This bundle can be accessed by an activity if it is passed in as an argument to its onCreate method. It can also be saved by all activities using there built in onSaveInstantState method that can be overridden if needed. The bundle is then stored in the file system automatically by the OS. (Operating System) In this project preferences have been used to allow a user to turn on/off the automatic saving of places. To achieve this, I have created a PreferencesActivity.java file. In the on create method I pass savedInstanceState in as an argument this will allow the activity to read or write the currently stored preferences. I then use the addPreferencesFromResource method to update the preferences based on the users input. Once the preferences activity is closed the bundle is saved using the activities default onSaveInstantState method.

In the main activity the getDefaultSharedPreferences method is called on the preference manager object passing getApplicationContext as the argument. This returnes the bundle and stores it in a variable. Then the getBoolean method is called on that variable passing the string key for the auto save preference and the returned value is stored in another variable. This variable is then used in the onActivityResult method when adding a new place to decide if the place should be automatically saved or not.

# 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 OS 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.