School of Media Arts and Technology

BSc Software Engineering

Matthew Dear

Q10232711

Engineering the Places to Stay Mobile Application

Assignment 1

Mobile Application Development

Tutor: Craig Gallen

19th May 2020

Contents

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 object has been given an 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 linier layout, 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 object 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 object displaying a label for the input pulled in from the strings.xml file. All the TextView objects have the width attribute set to `wrap_content` allowing the EditText objects also included in the linier layout 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 object. 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 linier layout has the gravity attribute set to centre 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 object 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 object 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 more times in the onClick method to retrieve the name and price EditText and the AutoCompleteTextView UI objects.

Communication between activities

To facilitate communication between the MainActivity.xml and the AddNewActivity.xml files 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 between activities. 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 to where it should return once completed. It then uses the MainActivity 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 methods, 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 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 methods passing the corresponding string keys as arguments 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 updated 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 object as an argument. It then uses the addPreferencesFromResource method taking the preferences.xml file as an argument allowing the user to see the current preferences and update them. 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

To facilitate file I/O Android uses either input or output streams; an input stream being the keyboard and an output stream being the screen. These streams are then divided again into low-level and wrapper streams. A low-level stream reads or writes individual bytes to and from files or hardware devices. In comparison a wrapper stream takes data from or provides it to a low-level stream while adding additional features such as, converting RAW data into other data types or objects, converting files between file types, and most importantly converting ASCII characters to and from individual bytes. In this project file I/O has been used to read and write a local copy of places to stay so that the application can be used offline. Below I will talk about the two methods in MainActivity that do this the first being savePTSLocally and the second being loadLocalPTS.

SavePTSLocally method

This method uses a low-level stream object called FileWriter to write to the file system using the Environment.getExternalStorageDirectory.getAbsolutePath methods, used to return a devices root directory, with a custom file name as an argument in the constructor. The FileWriter object is then passed as an argument to the constructor of the PrintWriter object. This wrapper stream is then used to convert ASCII characters to RAW bytes of data to be saved to the file system. The printLn method is called on the PrintWriter object, passing a data string as an argument, saving one object to the file each time around the loop. Once all data has been written to the file the close method is called on the PrintWriter object closing both the high and low-level streams.

LoadLocalPTS method

This method uses a low-level stream object called FileReader to read from the file system using the Environment.getExternalStorageDirectory.getAbsolutePath methods with a custom file name as an argument in the constructor. The FileReader object is then passed as an argument to the constructor of the BufferedReader object. This wrapper stream is then used to convert RAW bytes of data into ASCII characters to be used in the constructor method for a new place object. The readLine method is called on the BufferedReader object, extracting the individual components of the object separated with a comma, creating one new place object each time around the loop. Once all the data has been read from the file the close method is called on the BufferedReader object closing both the high and low-level streams.

Network communication

In Android, all methods that might fail or take a long time to run, like network communication, should be implemented in a separate inner class that extends the functionality of the AsyncTask class. This puts it in a separate thread alongside the UI thread, stopping it from hanging the UI thread and crashing the application. These async classes are called from inside an activity so, to give it access to the parent activities variables and methods you will need to change its scope. This is done by prefixing the class name with the inner key word allowing it to access a variable by prefixing the variable name with the name of the parent activity. In this project network communication has been used to save and load places to stay from a remote server. All network communications are done wrapped in a try catch block, to stop the application from crashing if the server is unresponsive, with a finally block added. This block will run if the code completes successfully or errors to stop the remote server connection from being orphaned. Below I will talk about the two methods in the main activity that do this.

InnerRemoteSave class

This method uses a URL object that has the remote servers POST address passed into the constructor. In this application the address is hard coded but, in the real world this would be stored in a variable. A POST request is being used as data being sent from the application is being used to update the database. The openConnection method is then called on the URL object and it is cast as a HttpURLConnection object to the variable conn. Then the setDoOutput method is called on conn allowing the POST data to be transmitted. Also, the setFixedLengthStreamingMode method is called on conn passing the length of the post data stream in as an argument to tell it how much data will be sent. The getOutputStream method is then called on conn storing the returned OuputStream object in a variable. The write method is then called on the OutputStream object, passing the POST data string as an argument, saving one object to the remote server each time around the loop. An if statement is then used to check the connection response code equals 200, if it is then the getInputStream method is called on conn storing the returned InputStream object in a variable. Then the input steam is passed in as an argument to the constructor of an InputStreamReader object. That is itself passed into the constructor of a BufferedReader object. This BufferedReader object then uses the readLine method to store the remote server's response in a string variable, that is returned by this method to the onPostExecute method to be displayed in an alert dialog to the user.

InnerRemoteLoad class

This method uses a URL object that has the remote servers address and GET query string passed into the constructor. A GET request is being used as data being sent from the application is not being used to update the database. The openConnection method is then called on the URL object and it is cast as a HttpURLConnection object to the variable conn. The getInputStream method is then called on conn storing the returned InputStream object in a variable. An if statement is then used to check the connection response code equals 200, if it is then the InputSteam object is passed in as an argument to the constructor of an InputStreamReader object. That is itself passed into the constructor of a BufferedReader

object. This BufferedReader object is then used in the same way as the one in the LoadLocalPTS method of MainActivity. A string value is then returned by this method to the onPostExecute method to be displayed in an alert dialog to the user.

CSV parsing

In this application I have chosen to use a CSV (Comma Separated Values) file to store the places to stay locally on the device. To be able to save places to the file they will need to be converted from a place object into a string of comma separated values and, this happens in the savePTSLocally method of MainActivity. Also, to be able to read places from the file the comma separated values will need to be converted to a place object and, this happens in the loadLocalPTS method of MainActivity.

Parsing objects to CSV

A foreach 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 PrintWriter wrapper stream, with commas added in between each one to separate the values, before 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 extracted when parsed back into an object.

Parsing CSV to objects

A while loop is used that runs until the readLine method called on the BufferedReader wrapper 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 results in the next value being extracted until a comma is read, then it is stored in an indexed array variable. An if statement is then used to check the length of the indexed array if it is equal to 5 then the statement is triggered. Then two string variables are created, name and type, and three double variables are created: price, latitude, and longitude. The corresponding values in the indexed array are then stored in these variables. Finally, a new place object is created by passing the name, type, price, latitude, and longitude variables in as arguments to the constructor method.