

[This question paper contains *Sixteen* printed pages]

BSc(IT)/2021/IV/F/ITW202/I

Student No:

Semester End Examination, Spring 2021
Royal University of Bhutan
Gyalpozhing College of Information Technology
BSc in Information Technology - Year II, Semester II
ITW202 : Mobile Application

Time: 3 Hours

Max. Marks: 50

Write your Student number on the top immediately on receipt of this question paper. All questions are compulsory, and marks are given at the end of each question. Parts of a question should be answered together. Spend the first 10 minutes in reading the questions.

PART A

5

Multiple Choice Questions

5 x 0.5 = 2.5

Q1) What is a system broadcast?

- A. A message that your app sends and receives when an event of interest occurs in the app.
- B. A message that is sent from an app to a different component of the same app.
- C. A message that the Android system sends when a system event occurs. ✓
- D. A message that the Android system receives when an event of interest occurs in your app.

Q2) When the user rotates the device, how do AsyncTask and AsyncTaskLoader behave differently if they are in the process of running a task in the background?

- A. A running AsyncTask becomes disconnected from the activity, but keeps running. A running AsyncTaskLoader becomes disconnected from the activity and stops running, preserving system resources.
- B. A running AsyncTask becomes disconnected from the activity and stops running, preserving system resources. A running AsyncTaskLoader automatically restarts execution of its task from the beginning. The activity displays the results.
- C. A running AsyncTask becomes disconnected from the activity, but keeps running. A running AsyncTaskLoader automatically reconnects to the activity after the device rotation. The activity displays the results. ✓
- D. A running AsyncTask becomes disconnected from the activity, but keeps running. A running AsyncTaskLoader automatically reconnects to the activity after the device rotation. The activity displays the results.

Q3) Which steps do you perform to test a View interaction, and in what order? Choose one:

- A. Match a View, assert and verify the result, and perform an action.
- B. Match a View, perform an action, and assert and verify the result. ✓
- C. Perform an action, match a view, and assert and verify the result.
- D. Perform an action, and assert and verify the result.

Q4) Which folder would hold the strings.xml file for translation into French for Canada? Choose one:

- A. res/values-fr-rFR/
- B. res/values-ca-rFR/ ✓
- C. res/values-fr-rCA/
- D. res/values-en-rFR/

Q5) Which of the following statements about a RecyclerView is false? Choose one.

- A. A RecyclerView is a more resource-efficient way to display scrollable lists.
- B. You need to provide a layout for just one item of the list.
- C. All list items look the same. ✓
- D. You don't need a layout manager with a RecyclerView to handle the hierarchy and layout of View elements.

Fill in the blanks

5 x 0.5 = 2.5

Q6)clear()... method in a SharedPreferences.Editor object is used to remove all the data stored in the preferences.

Q7) To process the incoming Intent associated with a broadcast, you subclass theBroadcastReceiver..... class and implementonReceive().....

Q8)onPostExecute(Result.....) runs on the UI thread after the background computation is finished.

Q9)@Test..... annotation is used to mark a method as an actual test

PART B

5 x 3 = 15

Q1) If an AsyncTask is defined as follows:

3 x 1 = 3

private class DownloadFilesTask extends AsyncTask<URL, Integer, Long>

A. What is the type of the value that is passed to **doInBackground()** in the AsyncTask?

Answer: URL

1

B. What is the type of the value that is passed to the callback that reports the progress of the task?

Answer: Integer

1

C. What is the type of the value that is passed to the callback that is executed when the task completes?

Answer: Long

1

Q2) Explain **three** reasons on why Mobile Applications?

3 x 1 = 3

Answer:

A. Transformative Devices: Exaplanation

0.5 + 0.5

B. Reaching customers: Exaplanation

0.5 + 0.5

C.Changing business process: Exaplanation

0.5 + 0.5

D. Making Money: Exaplanation

Q3) **Operating system, Screen size and orientation, and Connectivity** issues are three of the app design issues. Explain them

3 x 1 = 3

Answer:

Explanation of operating System design issue:

1

Explanation of screen size and orientation design issue: 1

Explanation of connectivity issue: 1

Q4) Write down the **differences** between Native apps and Cross-platform apps with an example? 3

Answer:

- The term native app development refers to building a mobile app exclusively for a single platform. The app is built with programming languages and tools that are specific to a single platform. For example, you can develop a native Android app with Java or Kotlin and choose Swift and Objective-C for iOS apps. 1.5
- Cross-platform development points to the process of creating an app that works on several platforms. This is done by using tools like React Native, Xamarin, and Flutter, where the apps created can be deployed on both Android and iOS. 1.5

Q5) Explain **why** size of the APK matters? 3 x 1 = 3

Answer: The size of the APK matters as:

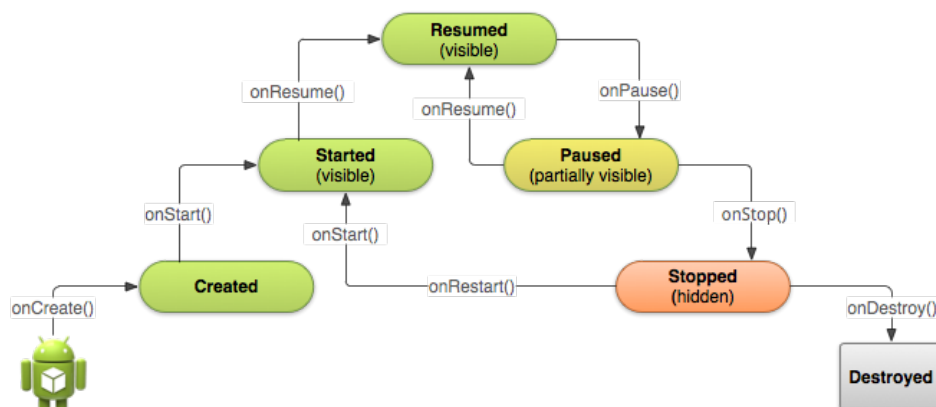
- how fast your app loads: Explanation 0.5 + 0.5
- how much memory it uses: Explanation 0.5 + 0.5
- how much power it consumes: Explanation 0.5 + 0.5

PART C

5 x 6 = 30

Q1) Draw Activity state and callback graph in android. 5

Answer:



Q2) Two methods **size** and **toString** are implemented in an ArrayList class(Figure 1). Create a **Unit test** class named **ArrayListUnitTesting.java** and implement the testing. **5**

```
ArrayList.java x ArrayListUnitTesting.java x
1 public class ArrayList{
2     int[] arr;
3     int size;
4     public ArrayList(){
5         arr = {1,5,6,7};
6         size = 4;
7     }
8     public int size(){
9         return size;
10    }
11    public String toString(){
12        String str="";
13        for (int i=0;i<size ;i++ ) {
14            str+=arr[i]+" ";
15        }
16        str=str.substring(0,str.length()-2);
17        str="["+str+"]";
18        return str;
19    }
20 }
```

Figure 1: ArrayList.java

Answer:

```
ArrayListUnitTesting.java x ArrayList.java x
1 @RunWith(JUnit4.class)//.....Marks: 0.5
2 public class ArrayListUnitTesting{
3     private ArrayList mArrayList;
4
5     @Before//.....Line 5 to Line 8....Marks: 0.5
6     public void setUp() {
7         mArrayList = new ArrayList();
8     }
9     // test for size
10    @Test//.....Line 10 - Line 14....Marks: 2
11    public void displaySize() {
12        int result = mArrayList.size();
13        assertEquals(resultAdd, is(equalTo(4)));
14    }
15    // test for string display
16    @Test//.....Line 16 - Line 19....Marks: 2
17    public void displayString() {
18        String result = mArrayList.toString();
19        assertEquals(resultAdd, is(equalTo([1,5,6,7])));
20    }
21 }
```

Figure 2: Solution: ArrayListUnitTesting.java

Q3) Implement the implicit intent for sharing the text in android. The layout sample is attached in Figure 2. You can have one xml file and one java file for this. Hardcode the strings. **5**

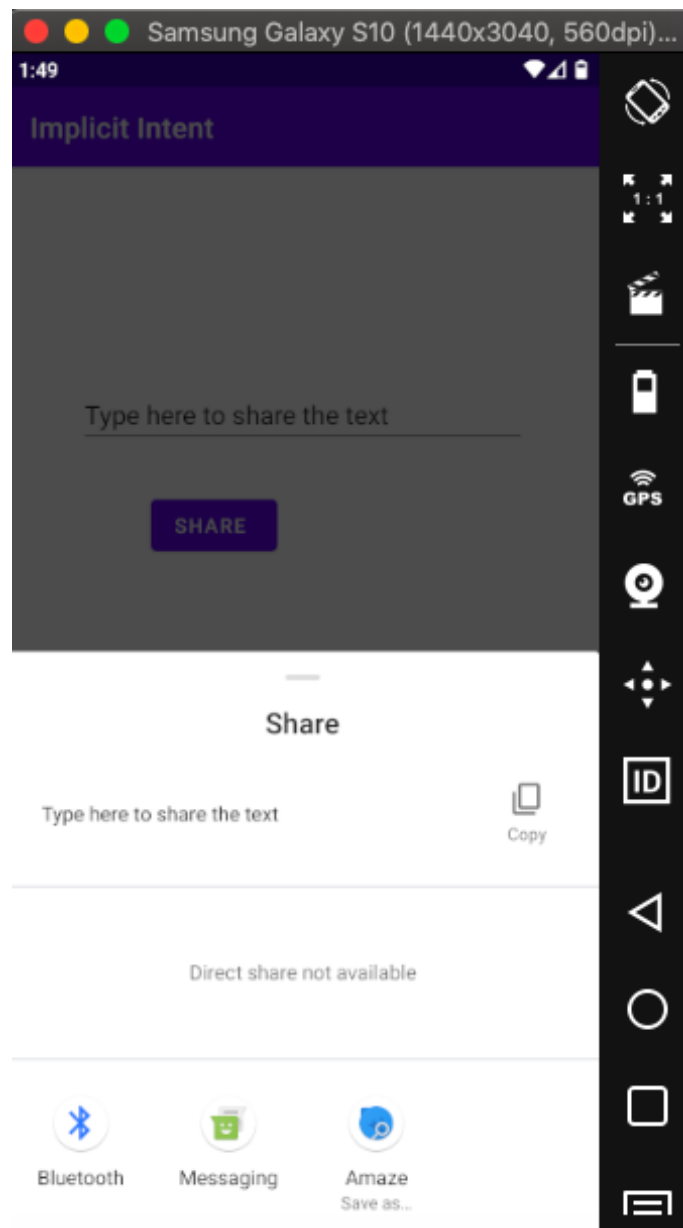
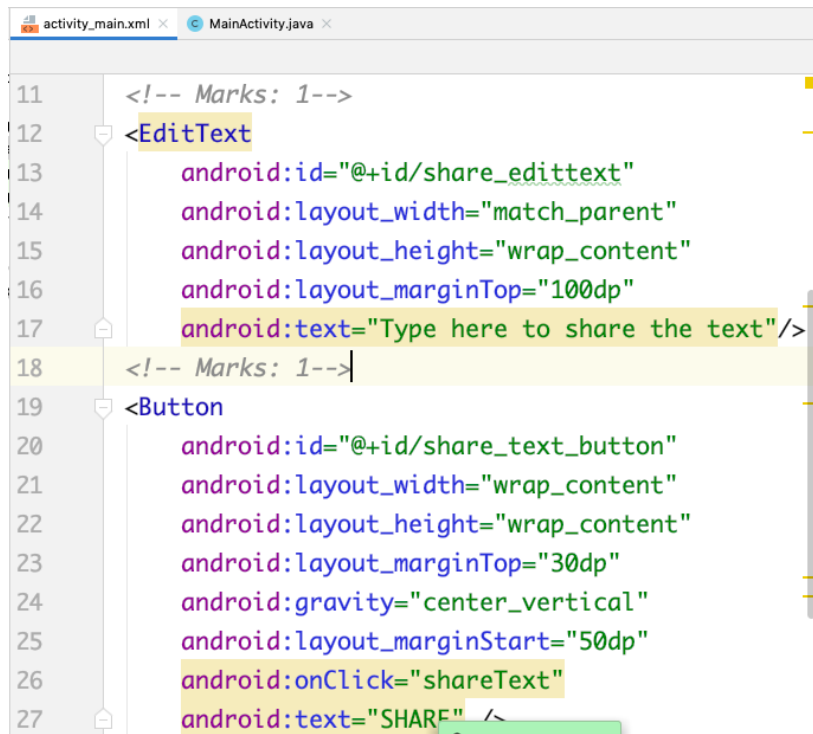



Figure 3: Implicit Intent: Sharing

Answer:



```
11 <!-- Marks: 1-->
12 <EditText
13     android:id="@+id/share_edittext"
14     android:layout_width="match_parent"
15     android:layout_height="wrap_content"
16     android:layout_marginTop="100dp"
17     android:text="Type here to share the text"/>
18 <!-- Marks: 1-->
19 <Button
20     android:id="@+id/share_text_button"
21     android:layout_width="wrap_content"
22     android:layout_height="wrap_content"
23     android:layout_marginTop="30dp"
24     android:gravity="center_vertical"
25     android:layout_marginStart="50dp"
26     android:onClick="shareText"
27     android:text="SHARE"/>
```

Figure 4: activity_main.xml



```

main.xml x MainActivity.java x
public class MainActivity extends AppCompatActivity {
    private EditText mShareEditText; //--- Marks: 0.5---
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        //--- Marks: 0.5 ---
        mShareEditText = findViewById(R.id.share_edittext);
    } //----- Marks: 2 -----
    public void shareText(View view) {
        String txt = mShareEditText.getText().toString();
        String mimeType = "text/plain";
        ShareCompat.IntentBuilder
            .from(this)
            .setType(mimeType)
            .setChooserTitle("Share the text with...")
            .setText(txt)
            .startChooser();
    }
}

```

Figure 5: MainActivity.java

Q4) The Overflow Optional Menu is shown in the given figure 3. Implement only the **menu_main.xml** to show the menu item with its properties. **5**

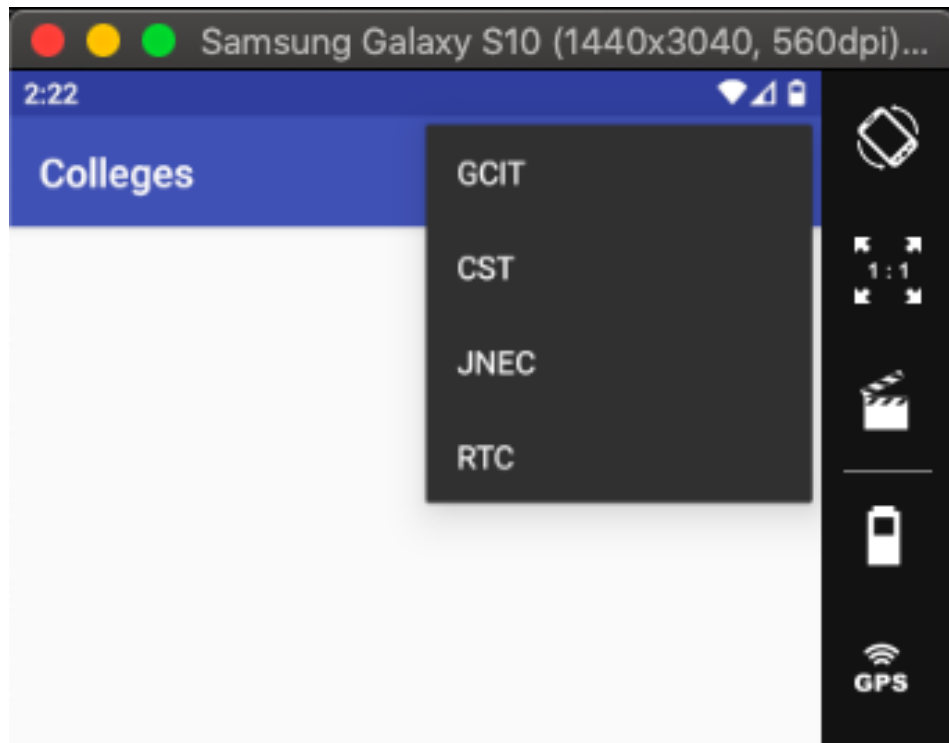


Figure 6: Overflow optional menu

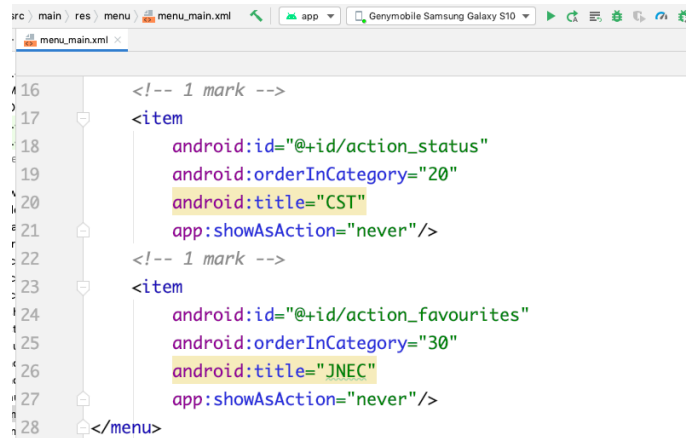
Answer:

```

1  <?xml version="1.0" encoding="utf-8"?><!-- 1 mark -->
2  <menu xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:app="http://schemas.android.com/apk/res-auto">
4      <!-- 1 mark -->
5      <item
6          android:id="@+id/action_contact"
7          android:orderInCategory="100"
8          android:title="RTC"
9          app:showAsAction="never"/>
10     <!-- 1 mark -->
11     <item
12         android:id="@+id/action_order"
13         android:orderInCategory="10"
14         android:title="GCIT"
15         app:showAsAction="never"/>

```

Figure 7: menu_main.xml



```

16 <!-- 1 mark -->
17 <item
18     android:id="@+id/action_status"
19     android:orderInCategory="20"
20     android:title="CST"
21     app:showAsAction="never"/>
22 <!-- 1 mark -->
23 <item
24     android:id="@+id/action_favourites"
25     android:orderInCategory="30"
26     android:title="JNEC"
27     app:showAsAction="never"/>
28 </menu>

```

Figure 8: menu_main.xml_partii

Q5) You will build an app(Figure 4 and Figure 5) that contains an EditText and a Button.

10

- The user enters the name of the book in the EditText and taps the button.
- The button executes an AsyncTask that queries the Google Books API to find the author and title of the book the user is looking for.
- The results are retrieved and displayed in a TextView below the button

Implement the internet connection in worker thread. The details of bookAPI are given below:

- **BOOK_BASE_URL** = "https://www.googleapis.com/books/v1/volumes?";
- **QUERY_PARAM** = "q";
- **MAX_RESULTS** = "maxResults";
- **PRINT_TYPE** = "printType";
- **Note:** No need to check network availability.

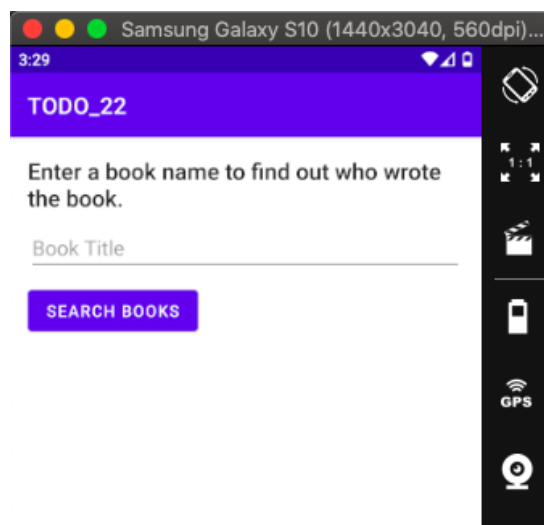


Figure 9: BookAPI_I

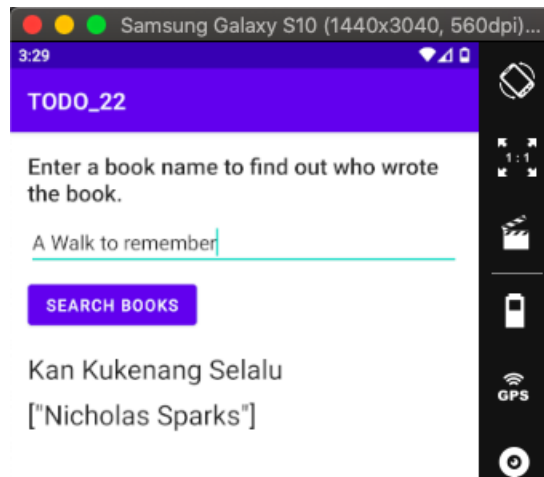


Figure 10: BookAPI_ii

Answer:

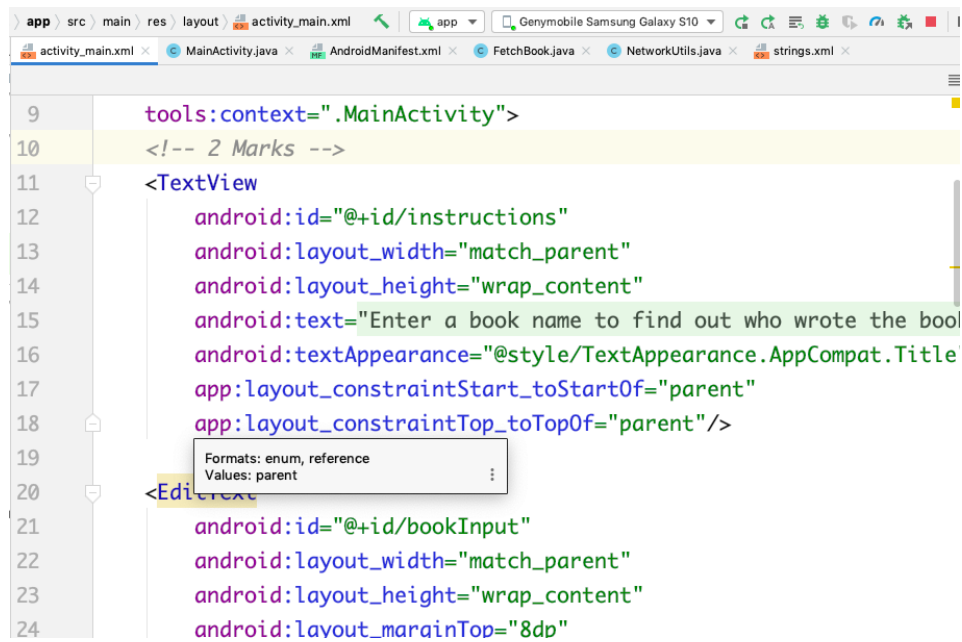


Figure 11: activity_mainPartI.xml

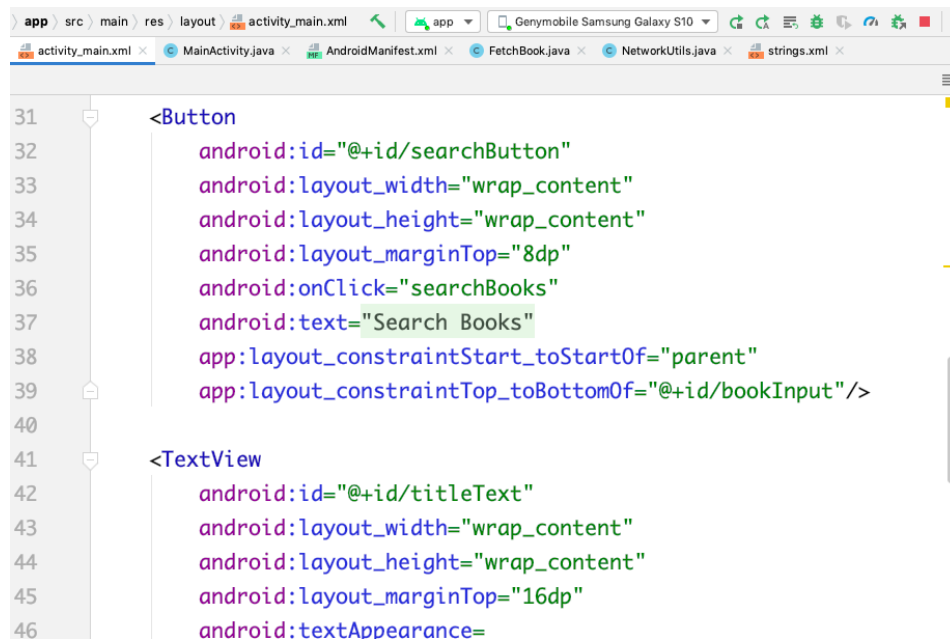


Figure 12: activity_mainPartII.xml

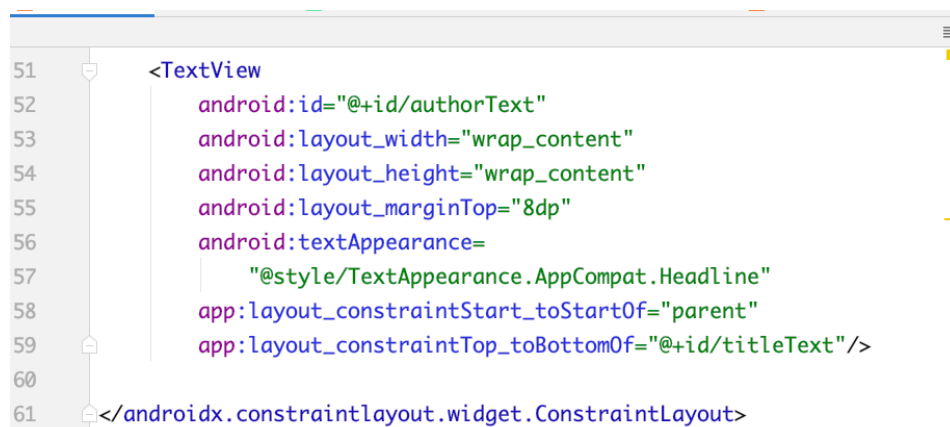
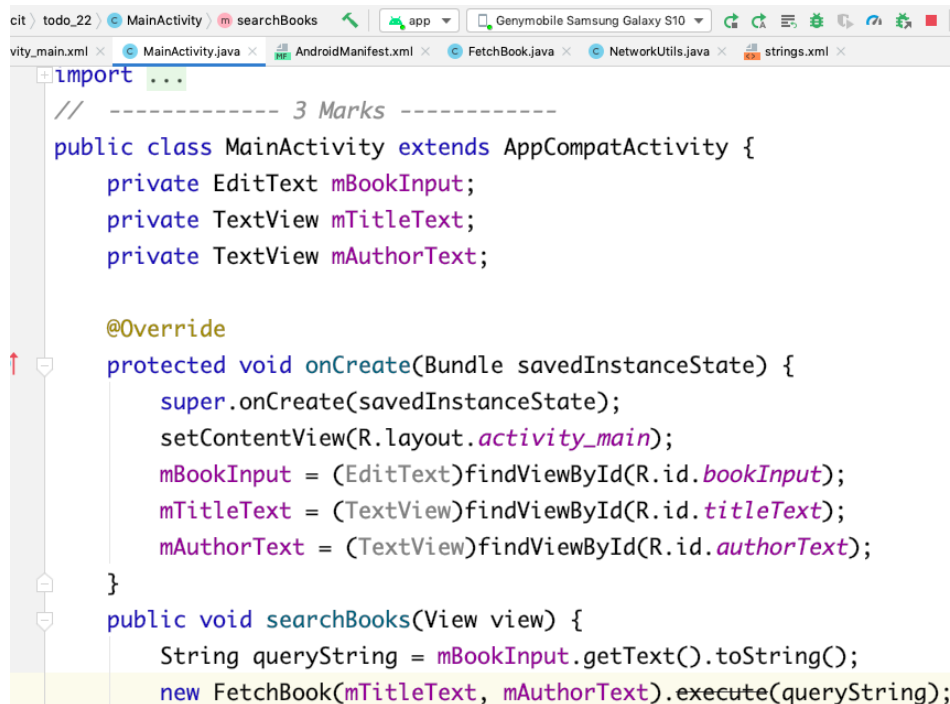


Figure 13: activity_mainPartIII.xml



```

import ...

// ----- 3 Marks -----

public class MainActivity extends AppCompatActivity {
    private EditText mBookInput;
    private TextView mTitleText;
    private TextView mAuthorText;

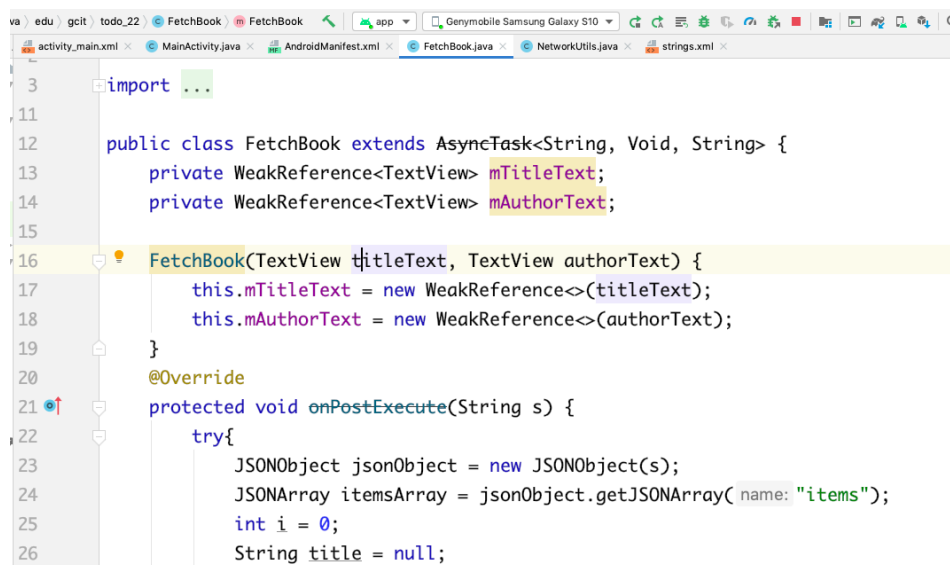
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        mBookInput = (EditText)findViewById(R.id.bookInput);
        mTitleText = (TextView)findViewById(R.id.titleText);
        mAuthorText = (TextView)findViewById(R.id.authorText);
    }

    public void searchBooks(View view) {
        String queryString = mBookInput.getText().toString();
        new FetchBook(mTitleText, mAuthorText).execute(queryString);
    }
}

```

Figure 14: MainActivity.java

3



```

import ...

public class FetchBook extends AsyncTask<String, Void, String> {
    private WeakReference<TextView> mTitleText;
    private WeakReference<TextView> mAuthorText;

    FetchBook(TextView titleText, TextView authorText) {
        this.mTitleText = new WeakReference<>(titleText);
        this.mAuthorText = new WeakReference<>(authorText);
    }

    @Override
    protected void onPostExecute(String s) {
        try{
            JSONObject jsonObject = new JSONObject(s);
            JSONArray itemsArray = jsonObject.getJSONArray( name: "items");
            int i = 0;
            String title = null;

```

Figure 15: FetchBookPartI.java

2.5

```

27 String authors = null;
28 while (i < itemsArray.length() &&
29       (authors == null && title == null)) {
30     // Get the current item information.
31     JSONObject book = itemsArray.getJSONObject(i);
32     JSONObject volumeInfo = book.getJSONObject("volumeInfo");
33
34     // Try to get the author and title from the current item,
35     // catch if either field is empty and move on.
36     try {
37         title = volumeInfo.getString( name: "title");
38         authors = volumeInfo.getString( name: "authors");
39     } catch (Exception e) {
40         e.printStackTrace();
41     }
42
43     // Move to the next item.
44     i++;

```

Figure 16: FetchBookPartII.java

```

45 }
46 if (title != null && authors != null) {
47     mTitleText.get().setText(title);
48     mAuthorText.get().setText(authors);
49 }
50 else {
51     mTitleText.get().setText("No Result Found");
52     mAuthorText.get().setText("");
53 }
54
55 } catch (JSONException e) {
56     mTitleText.get().setText("No Result Found");
57     mAuthorText.get().setText("");
58     e.printStackTrace();
59 }
60
61

```

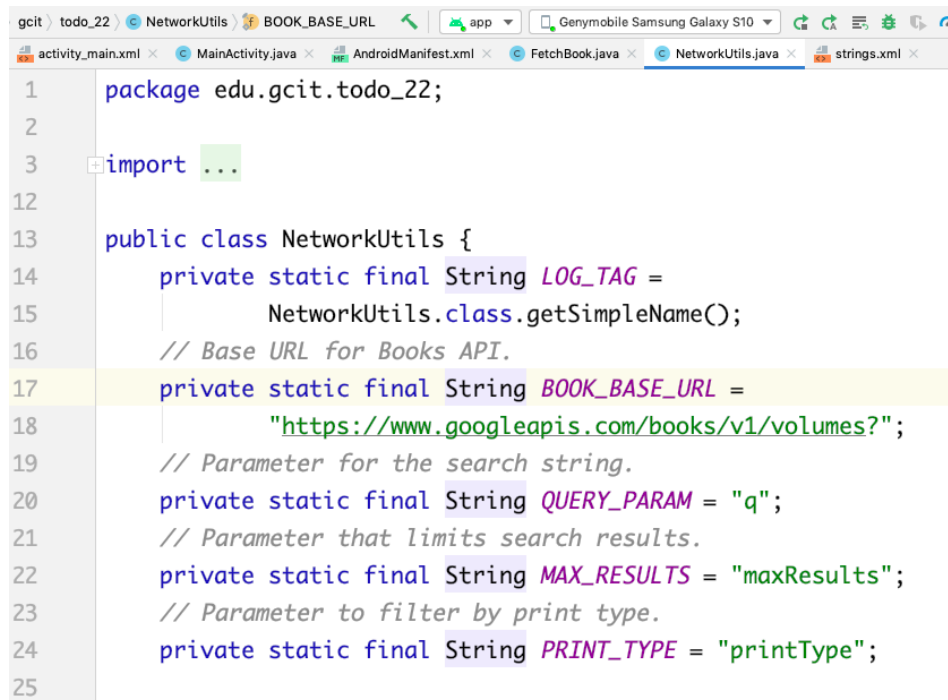
Figure 17: FetchBookPartIII.java

```

62 super.onPostExecute(s);
63 }
64
65 @Override
66 protected String doInBackground(String... strings) {
67     return NetworkUtils.getBookInfo(strings[0]);
68 }
69 }
70

```

Figure 18: FetchBookPartIV.java



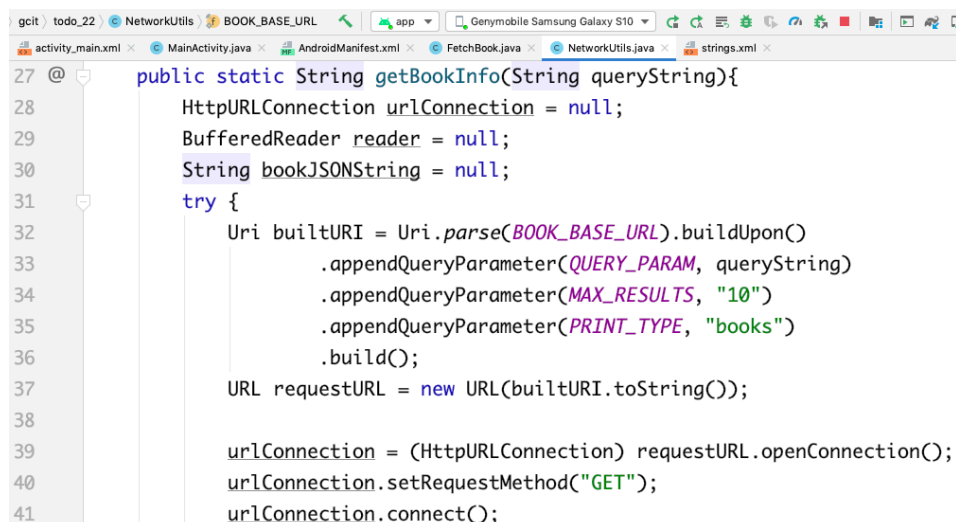
```

1 package edu.gcit.todo_22;
2
3 import ...
4
12
13 public class NetworkUtils {
14     private static final String LOG_TAG =
15         NetworkUtils.class.getSimpleName();
16     // Base URL for Books API.
17     private static final String BOOK_BASE_URL =
18         "https://www.googleapis.com/books/v1/volumes?";
19     // Parameter for the search string.
20     private static final String QUERY_PARAM = "q";
21     // Parameter that limits search results.
22     private static final String MAX_RESULTS = "maxResults";
23     // Parameter to filter by print type.
24     private static final String PRINT_TYPE = "printType";
25

```

Figure 19: NetworkUtilsPartI.java

2.5

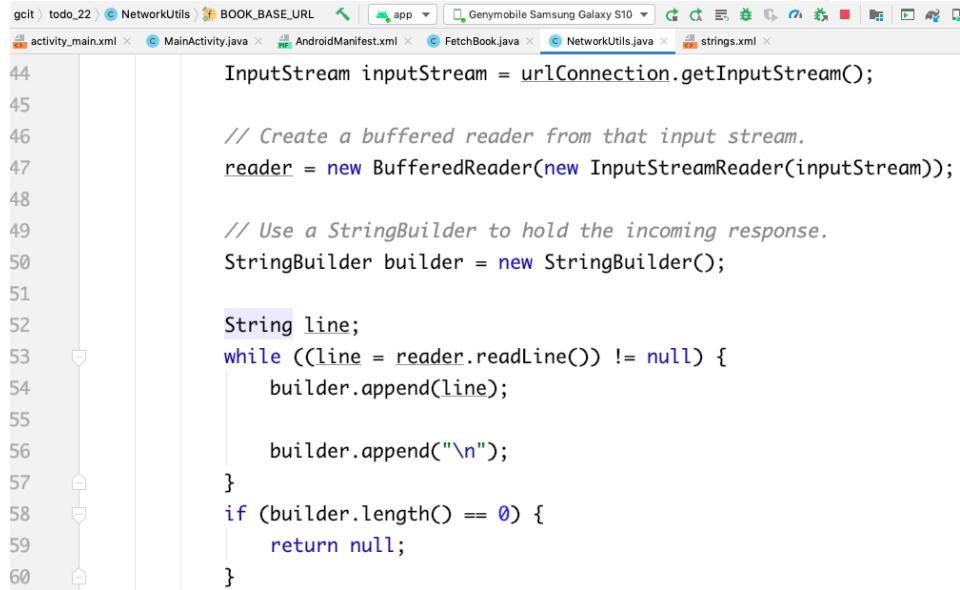


```

27 @ public static String getBookInfo(String queryString){
28     HttpURLConnection urlConnection = null;
29     BufferedReader reader = null;
30     String bookJSONString = null;
31     try {
32         Uri builtURI = Uri.parse(BOOK_BASE_URL).buildUpon()
33             .appendQueryParameter(QUERY_PARAM, queryString)
34             .appendQueryParameter(MAX_RESULTS, "10")
35             .appendQueryParameter(PRINT_TYPE, "books")
36             .build();
37         URL requestURL = new URL(builtURI.toString());
38
39         urlConnection = (HttpURLConnection) requestURL.openConnection();
40         urlConnection.setRequestMethod("GET");
41         urlConnection.connect();

```

Figure 20: NetworkUtilsPartII.java

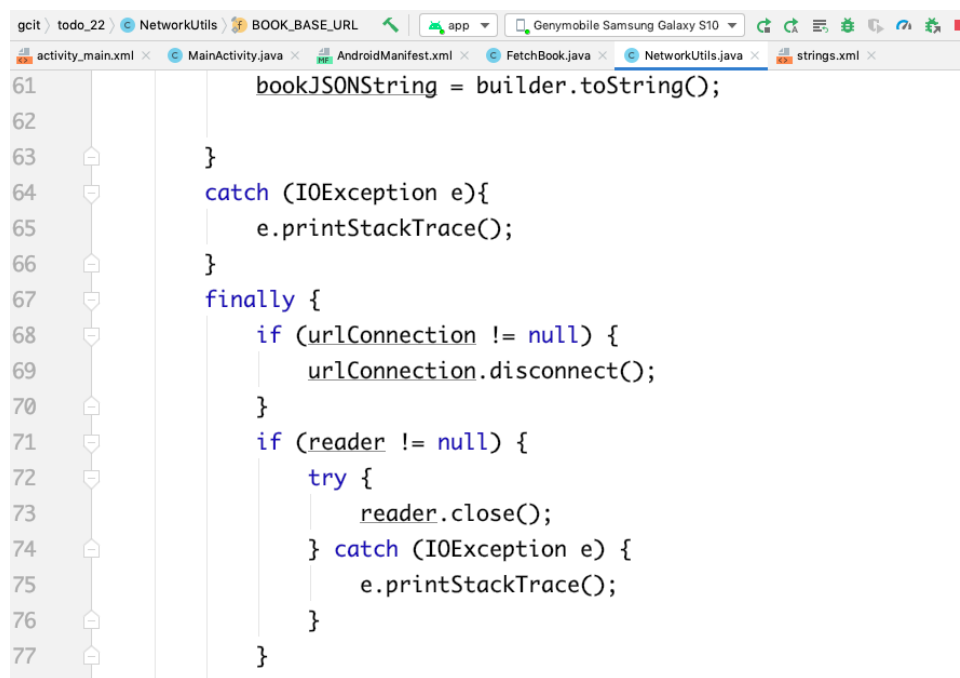


```

44      InputStream inputStream = urlConnection.getInputStream();
45
46      // Create a buffered reader from that input stream.
47      reader = new BufferedReader(new InputStreamReader(inputStream));
48
49      // Use a StringBuilder to hold the incoming response.
50      StringBuilder builder = new StringBuilder();
51
52      String line;
53      while ((line = reader.readLine()) != null) {
54          builder.append(line);
55
56          builder.append("\n");
57      }
58      if (builder.length() == 0) {
59          return null;
60      }

```

Figure 21: NetworkUtilsPartIII.java

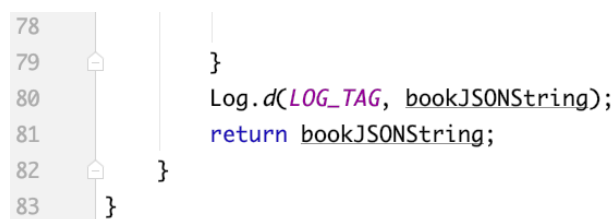


```

61      bookJSONString = builder.toString();
62
63      }
64      catch (IOException e){
65          e.printStackTrace();
66      }
67      finally {
68          if (urlConnection != null) {
69              urlConnection.disconnect();
70          }
71          if (reader != null) {
72              try {
73                  reader.close();
74              } catch (IOException e) {
75                  e.printStackTrace();
76              }
77          }

```

Figure 22: NetworkUtilsPartIV.java



```

78
79      }
80      Log.d(LOG_TAG, bookJSONString);
81      return bookJSONString;
82  }
83  }

```

Figure 23: NetworkUtilsPartV.java