|  |
| --- |
| **My Modules** |

Contents

[Section A: Understanding Intent 2](#_Toc71711144)

[Create a New Project for Intent Example 2](#_Toc71711145)

[Design the UI Layout 3](#_Toc71711146)

[Create the Second Activity 4](#_Toc71711147)

[Design the Second Activity 7](#_Toc71711148)

[Event Handling in MainActivity Class 7](#_Toc71711149)

[Launching 2nd Activity 9](#_Toc71711150)

[Passing Data from Main Activity to the Second Activity 10](#_Toc71711151)

[Receiving Data in the Second Activity 10](#_Toc71711152)

[Writing Code to Launch the Third Activity 11](#_Toc71711153)

[Section B: Intent Data Passing 12](#_Toc71711154)

[Create Another Project on Intent Data Passing 12](#_Toc71711155)

[Design the Main Activity UI Layout 12](#_Toc71711156)

[Create the Second Activity and Update the UI Layout 13](#_Toc71711157)

[Additional Challenge 16](#_Toc71711158)

[Section C: Activity Lifecycle and Intent 17](#_Toc71711159)

[Learning Activity Lifecycle Using Logcat 18](#_Toc71711160)

[Back to the Problem Statement 23](#_Toc71711161)

[Create a New Project for the App 23](#_Toc71711162)

[Design the Layout 23](#_Toc71711163)

[Handle the User Events 23](#_Toc71711164)

[Additional Challenge 23](#_Toc71711165)

**Resources and References**

Managing Activity Lifecycle

<http://developer.android.com/training/basics/activity-lifecycle/index.html>

Activity and Intent

<http://developer.android.com/guide/components/fundamentals.html>

<http://developer.android.com/reference/android/app/Activity.html>

<http://developer.android.com/guide/components/intents-filters.html>

# Section A: Understanding Intent

For the past few weeks, the Android apps that we have developed usually comprise of only one Activity, and the Activity is associated to a layout file. However, for most apps in the Play Store, would they be that simple, with only one screen (Activity)?

Obviously, apps with a single Activity would have its limitations, either with limited functionalities or in adhering to the aesthetic integrity (representing how well an app’s UI and business logics work well in fulfilling the app’s objectives) of the app.

To address the limitations, you as a developer could implement more than one Activity class in the Android project, and we will be learning new APIs and coding approaches to "link" one Activity to another, as well as passing message between the Activities. The "linking" could be done via a message passing mechanism, known as **Intent** in Android platform.

Let’s try out some exercises to learn more about Intent and how to implement Intent in your Android project.

## Create a New Project for Intent Example

Let’s develop an app that displays riddles, in which the user can think out of the box by guessing the answer for a list of riddles. While doing so, do recall and apply what you have learnt in the previous problems, including event handling and layout implementation.

Following is the screen capture of the completed app.

|  |  |
| --- | --- |
| C:\Users\denise_quek\Desktop\Screenshot_1589908211.png | C:\Users\denise_quek\Desktop\Screenshot_1589971824.png  C:\Users\denise_quek\Desktop\Screenshot_1589972334.png |

Create the project based on the following requirements.

|  |  |
| --- | --- |
| **Project Template** | Empty Views Activity |
| **Application Name** | Demo Riddle |
| **Package Name** | sg.edu.rp.c346.id<your student ID>.demoriddle |
| **Project Location** | D:\C346\Workspace\DemoRiddle |
| **Language** | Java |
| **Minimum API Level** | API 24 |

## Design the UI Layout

Create a layout file with the following content as per the component tree and device screen in the Android Studio.

Note that Question 1 text and "Reveal" button takes up half of the screen width each. Also, Question 1 & 2 take up half of the screen height each.

|  |  |
| --- | --- |
|  | C:\Users\denise_quek\Desktop\Screenshot_1589908211.png |

The UI elements are sticking to the sides of the screen. Leave a bit of space by add the folllowing attribute "**android:padding**" with a value of "**16dp**" to the root layout.

Note: For Android Studio Flamingo, if you would like to view the Action Bar (the bar at the top of the screen showing the app name “Demo Riddle” in this case), change the theme to “Base.Theme.MaterialComponents.Light.DarkActionBar.Bridge” in the design mode of the layout file.

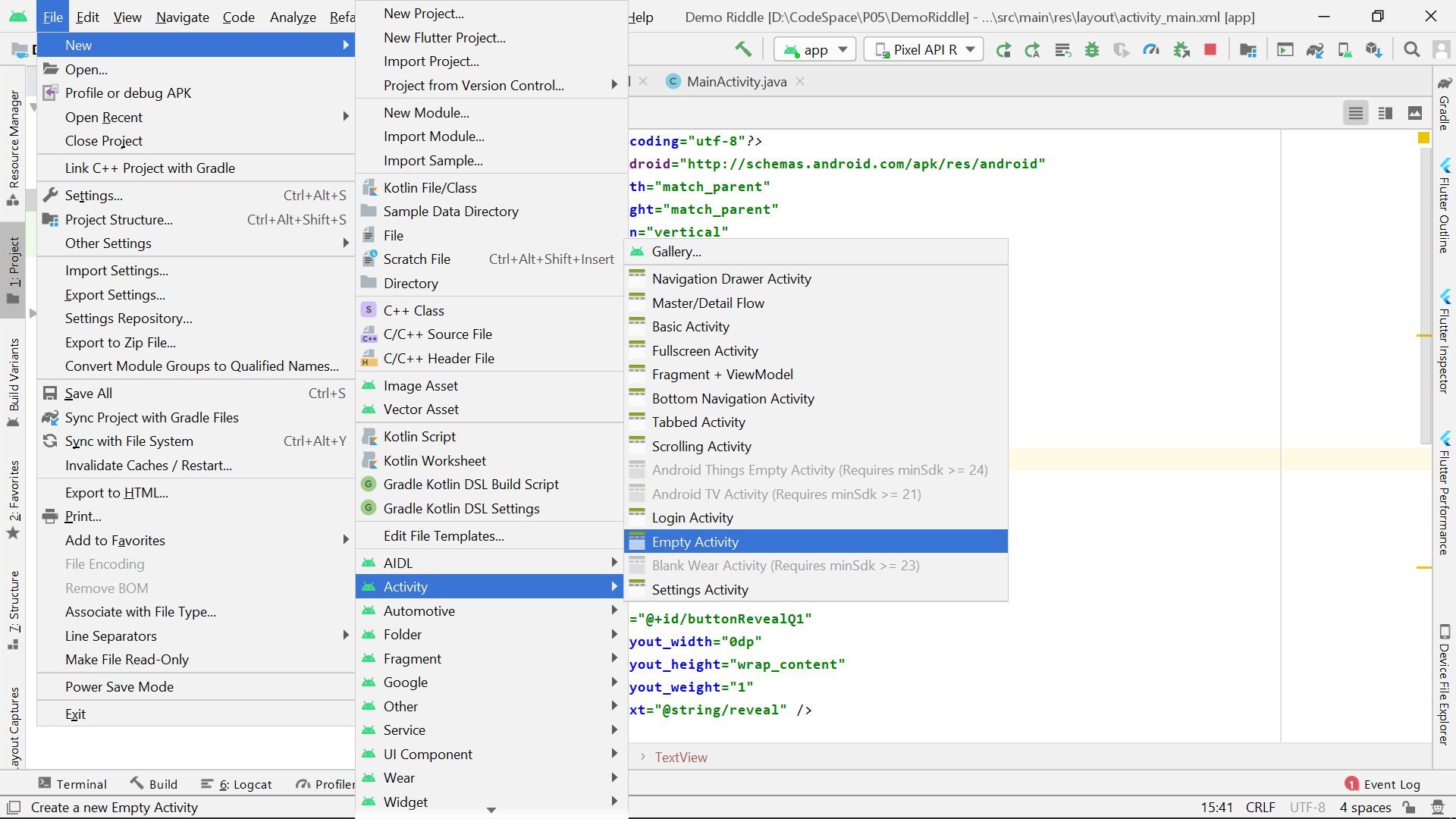


main\_activity.xml

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?> <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"  android:orientation="vertical"    android:padding="16dp">   <LinearLayout  android:layout\_width="match\_parent"  android:layout\_height="0dp"  android:layout\_weight="1"  android:orientation="horizontal">   <TextView  android:id="@+id/textViewQ1"  android:layout\_width="0dp"  android:layout\_height="match\_parent"  android:layout\_weight="1"  android:text="@string/q1" />   <Button  android:id="@+id/buttonRevealQ1"  android:layout\_width="0dp"  android:layout\_height="wrap\_content"  android:layout\_weight="1"  android:text="@string/reveal" />   </LinearLayout>   <LinearLayout  android:layout\_width="match\_parent"  android:layout\_height="0dp"  android:layout\_weight="1"  android:orientation="horizontal">   <TextView  android:id="@+id/textViewQ2"  android:layout\_width="0dp"  android:layout\_height="match\_parent"  android:layout\_weight="1"  android:text="@string/q2" />   <Button  android:id="@+id/buttonRevealQ2"  android:layout\_width="0dp"  android:layout\_height="wrap\_content"  android:layout\_weight="1"  android:text="@string/reveal" />  </LinearLayout> </LinearLayout> |

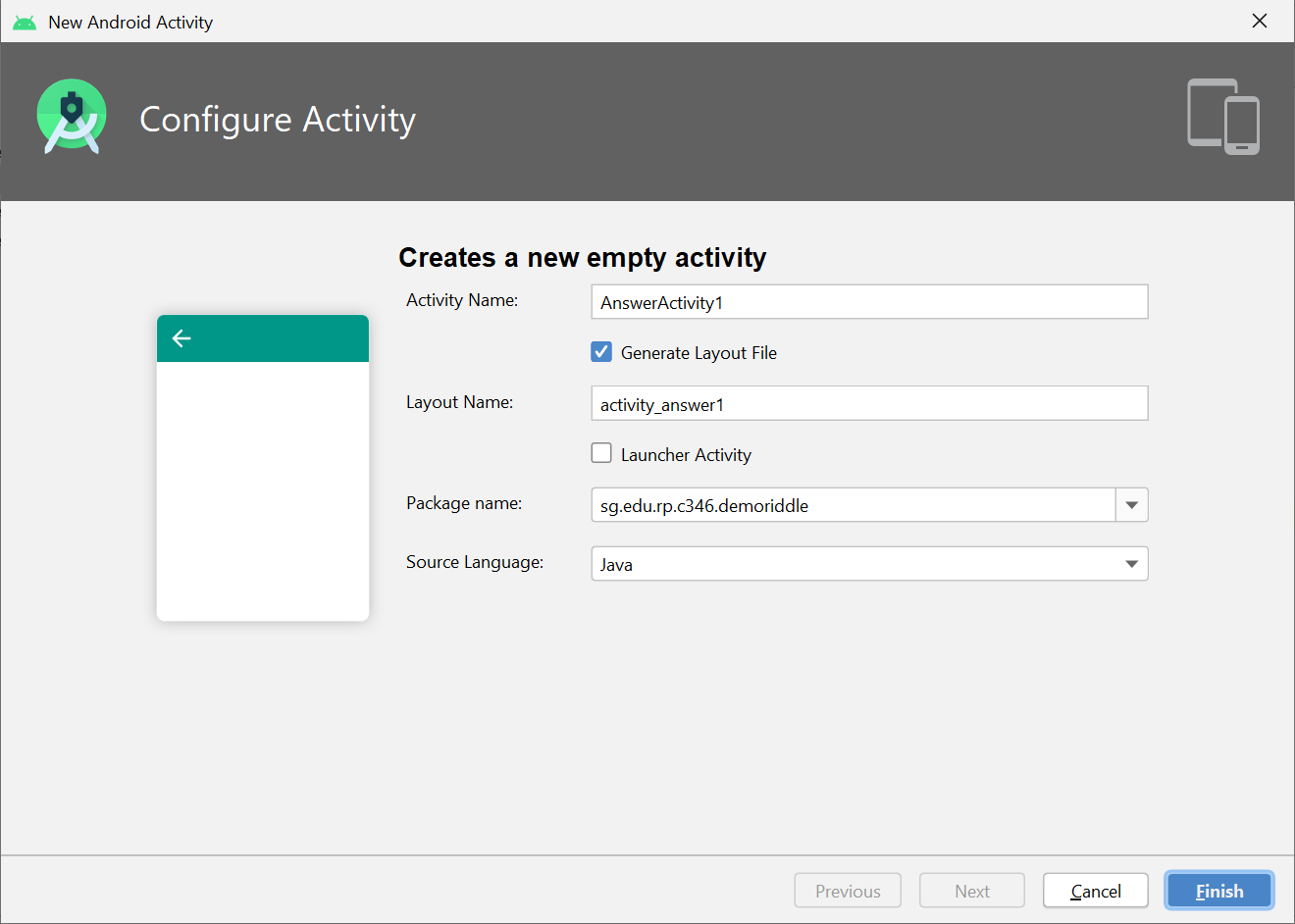
## Create the Second Activity

1. Click File 🡪 New 🡪 Activity 🡪 Empty Activity



1. Label the second Activity name as ***AnswerActivity1*** and click "**Finish**".

Take note of the corresponding layout name for this Activity.



1. Open the "**AndroidManifest.xml**" in the "**manifests**" folder. It contains some important information about this application. One key information is the registration of the Activities as follow:

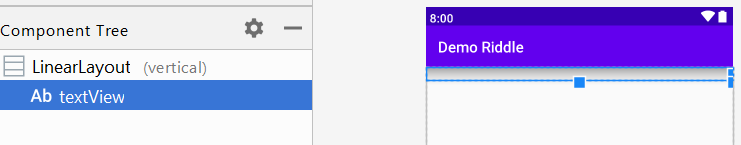


You should note that an XML element, “activity” is generated by the Android Studio, corresponding to the creation of the Activity class.

|  |  |
| --- | --- |
| ? | What is the purpose of the following XML content?  <action android:name="android.intent.action.MAIN" />  <category android:name="android.intent.category.LAUNCHER" />  You can read up more in the URL: <https://developer.android.com/reference/android/content/Intent.html> |
| <action android:name="android.intent.action.MAIN" />: This line specifies that the component (usually an activity) is the main entry point of the application. It indicates that this component should be launched when the application starts.  <category android:name="android.intent.category.LAUNCHER" />: This line declares the component as a launcher activity, which means it should appear in the user's list of available applications to launch. It adds the launcher category to the intent filter, indicating that this activity should be shown as an option for the user to launch the application from the home screen or app launcher. | |

## Design the Second Activity

1. Open the layout xml file of the second Activity and modify the layout file as per the component tree and device screen below.



To launch the second Activity by clicking the "**Reveal**" button in the main Activity, you will need to perform the following coding in the MainActivity class:

1. Implement listener code to handle the "**Reveal**" button click event from the user.
2. Create an Intent object to start the second Activity in revealing the answer of the riddle.

## Event Handling in MainActivity Class

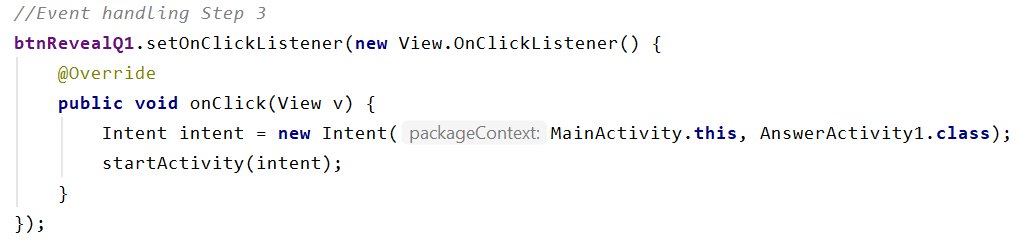
1. Write the code (in the red boxes below) for Button and TextView supporting Riddle Question 1 in **MainActivity** class:

|  |
| --- |
|  |

|  |  |
| --- | --- |
|  | **Learning Checkpoint 1** |
| By now, you should be able to   * Create a new Activity. * Apply the coding steps to support event handling in Android Activity.   To recap on what we have learnt so far,  In the above exercise, we have revisited the steps in writing codes to support handling as follow:  //Event handling Step 1 Button **btnRevealQ1**;  //Event handling Step 2 **btnRevealQ1** = findViewById(R.id.**buttonRevealQ1**);  *//Event handling Step 3* **btnRevealQ1**.setOnClickListener(**new** View.OnClickListener() {  @Override  **public void** onClick(View view) {  // Add your event handling code for button click action  } });  By now, you should be familiar with the above steps and able to apply in other common UI elements.  Next, there are 2 files that are generated when you create a new Activity, namely:   1. Activity class, i.e., AnswerActivity1.java 2. Layout file, i,e., activity\_answer1.xml   Importantly, the AndroidManifest file content will be updated to register the new class as well.  In summary, 2 files (Activity class and Layout file) will be created and 1 file (AndroidManifest.xml) will be updated. | |

## Launching 2nd Activity

1. The “Event handling step 3” is incomplete in **MainActivity** class, let’s write the following code (in red box) to launch the 2nd Activity (**AnswerActivity1**) when the button is tapped.



* The first line of the code above prepares the Activity that the app will be launching by creating an **Intent** object, with a reference to the 2nd Activity (**AnswerActivity1**) as the method parameter.
* The second line of the code starts the 2nd Activity with the method startActivity().

1. Modify the 2nd Activity (**AnswerActivity1**) class with the following code in the red boxes below.

|  |
| --- |
|  |

1. Run the application to confirm that the 2nd Activity is launched after tapping the first “Reveal” button.

|  |  |
| --- | --- |
| C:\Users\denise_quek\Desktop\Screenshot_1589908211.png | C:\Users\denise_quek\Desktop\Screenshot_1589971481.png |

## Passing Data from Main Activity to the Second Activity

Currently no answer is displayed in the 2nd Activity simply because we have not

1) informed the 2nd Activity which answer to reveal

2) written the code to display the answer

1. Add the highlighted code below in the **MainActivity class** to pass a “signal” to the 2nd Activity. This will “inform” the 2nd Activity to display the answer for Riddle Question 1.

|  |
| --- |
| *//Event handling Step 3 //Event handling Step 3* **btnRevealQ1**.setOnClickListener(**new** View.OnClickListener() {  @Override  **public void** onClick(View v) {  Intent intent = **new** Intent(MainActivity.**this**, AnswerActivity1.**class**);  intent.putExtra(**"Question"**, **"Q1"**);  startActivity(intent);  } }); |

The purpose of the 2 parameters in the **putExtra()** method is as follow:

|  |  |  |
| --- | --- | --- |
|  | Data Type | Meaning |
| 1st Parameter | String | The name of the extra data |
| 2nd Parameter | Multiple data type supported | The data value |

## Receiving Data in the Second Activity

1. Modify the 2nd Activity (**AnswerActivity1**) class, by commenting out the previous "setText()"statement and adding in the highlighted code segment as below.

This code segment means that, the 2nd Activity receives the data passed from the **MainActivity** and displays the answer accordingly.

|  |
| --- |
| **public class** AnswerActivity1 **extends** AppCompatActivity {  TextView **tvAnswer**;  @Override  **protected void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);  setContentView(R.layout.***activity\_answer1***);  **tvAnswer** = findViewById(R.id.***textView***); *// tvAnswer.setText("In Second Activity");* Intent intentReceived = getIntent();  String questionsSelected = intentReceived.getStringExtra(**"Question"**);  **tvAnswer**.setText(questionsSelected + **" answer is: Queue"**);  } } |

1. Run the app again and you should get the following output.

|  |  |
| --- | --- |
| C:\Users\denise_quek\Desktop\Screenshot_1589908211.png | C:\Users\denise_quek\Desktop\Screenshot_1589971824.png |

## Writing Code to Launch the Third Activity

The code to display the answer for the 2nd Riddle Question is not done yet. We can create another Activity to handle it.

1. Create a 3rd Activity named ***AnswerActivity2***.
2. Ensure that the Manifest file has registered the newly created Activity as follow. Include the code below if it is not in the content.

|  |
| --- |
| <activity android:name=".AnswerActivity2"></activity> |

1. Complete the code by following the steps we did just now, so that the completed app will yield the following output:

|  |  |
| --- | --- |
| C:\Users\denise_quek\Desktop\Screenshot_1589908211.png | C:\Users\denise_quek\Desktop\Screenshot_1589971824.png  C:\Users\denise_quek\Desktop\Screenshot_1589972334.png |

# Section B: Intent Data Passing

Besides String, an Activity can send other types of data to another Activity as well. In this section, we will learn how to pass and receive other types of data between activities.

## Create Another Project on Intent Data Passing

Let’s learn how to pass data from one Activity to another with a different data type.

Create a project based on the following requirements:

|  |  |
| --- | --- |
| **Project Template** | Empty Views Activity |
| **Application Name** | Demo Data Passing Test |
| **Package Name** | sg.edu.rp.c346.id<your student ID>.demodatapassingtest |
| **Project Location** | D:\C346\Workspace\DemoDataPassingTest |
| **Language** | Java |
| **Minimum API Level** | API 24 |

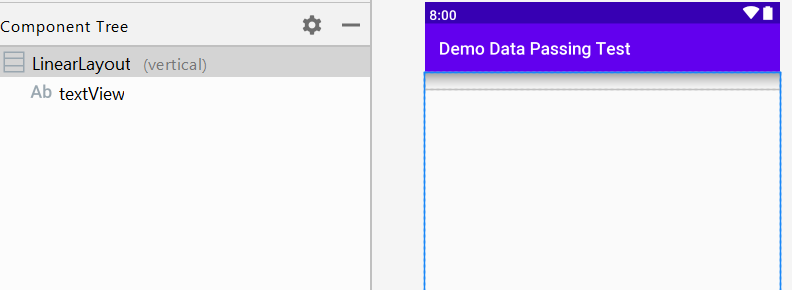
## Design the Main Activity UI Layout

Update the Main Activity layout file with the content as per the component tree and the device screen below.

|  |  |
| --- | --- |
|  | C:\Users\denise_quek\Desktop\Screenshot_1589972741.png |

## Create the Second Activity and Update the UI Layout

1. Create another Activity, known as *SecondActivity*.
2. Update the *SecondActivity* layout file with the content as per the component tree and device screen below.



1. Let’s write code in **MainActivity** to pass an integer value to the **SecondActivity**.

In the **MainActivity** class, add the following highlighted code segments to pass the integer value 1 to the **SecondActivity** via **Intent**, when tapping the button.

|  |
| --- |
| **public class** MainActivity **extends** AppCompatActivity {   *//Event handling Step 1* Button **btnPassInteger**;   @Override  **protected void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);  setContentView(R.layout.***activity\_main***);   *//Event handling Step 2* **btnPassInteger** = findViewById(R.id.***buttonPassInteger***);   *//Event handling Step 3* **btnPassInteger**.setOnClickListener(**new** View.OnClickListener() {  @Override  **public void** onClick(View view) {  Intent intent = **new** Intent(MainActivity.**this**, SecondActivity.**class**);  intent.putExtra(**"value"**, 1);  startActivity(intent);  }  });   } } |

1. Write code in the **SecondActivity** to receive the integer value 1 from the **MainActivity**.

Here are some of the methods to receive various types of data:

|  |  |  |  |
| --- | --- | --- | --- |
| Data Type | Method to use | Number of parameters | Meaning of each parameter |
| int | getIntExtra(String name, int defaultValue) | 2 | 1st: The name of the desired item.  2nd: the value to be returned if no value of the desired type is stored with the given name. |
| double | getDoubleExtra (String name, double defaultValue) | 2 | same as above |
| char | getCharExtra (String name,  char defaultValue) | 2 | same as above |

In the **SecondActivity** class, add the following highlighted code segments.

|  |
| --- |
| **public class** SecondActivity **extends** AppCompatActivity {   *//Step 1* TextView **tvAnswer**;   @Override  **protected void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);  setContentView(R.layout.***activity\_second***);   *//Step 2* **tvAnswer** = findViewById(R.id.***textView***);  Intent intentReceived = getIntent();  **int** value = intentReceived.getIntExtra(**"value"**, 0);  **tvAnswer**.setText(**"Integer value received is: "** + value);  } } |

1. Test out the program and it should yield the following output:

|  |  |
| --- | --- |
| C:\Users\denise_quek\Desktop\Screenshot_1589972741.png | C:\Users\denise_quek\Desktop\Screenshot_1589973229.png |

1. The code for passing character value is not implemented yet. Write the necessary code to yield the output as follow.

Hint: Referring to the **SecondActivity**, create a **ThirdActivity** to receive the character value passed from the **MainActivity**.

|  |  |
| --- | --- |
| C:\Users\denise_quek\Desktop\Screenshot_1589972741.png | C:\Users\denise_quek\Desktop\Screenshot_1589976916.png |

Paste your code snippets for the passing and the receiving in the box below.

|  |
| --- |
| PASSING (in MainActivity):  package sg.edu.rp.c346.id22001027.demodatapassingtest;  import android.content.Intent; import android.os.Bundle; import android.view.View; import android.widget.Button;  import androidx.appcompat.app.AppCompatActivity;  public class MainActivity extends AppCompatActivity {   // Event handling Step 1  Button btnPassInteger;  Button btnPassCharacter;   @Override  protected void onCreate(Bundle savedInstanceState) {  super.onCreate(savedInstanceState);  setContentView(R.layout.*activity\_main*);   // Event handling Step 2  btnPassInteger = findViewById(R.id.*buttonPassInteger*);  btnPassCharacter = findViewById(R.id.*buttonPassChar*);   // Event handling Step 3 - Pass Integer  btnPassInteger.setOnClickListener(new View.OnClickListener() {  @Override  public void onClick(View view) {  Intent intent = new Intent(MainActivity.this, SecondActivity.class);  intent.putExtra("value", 1);  startActivity(intent);  }  });   // Event handling Step 3 - Pass Character  btnPassCharacter.setOnClickListener(new View.OnClickListener() {  @Override  public void onClick(View view) {  Intent intent = new Intent(MainActivity.this, ThirdActivity.class);  intent.putExtra("character", 'a');  startActivity(intent);  }  });  } }  RECEIVING (in ThirdActivity):  package sg.edu.rp.c346.id22001027.demodatapassingtest;  import androidx.appcompat.app.AppCompatActivity;  import android.os.Bundle; import android.widget.TextView;  public class ThirdActivity extends AppCompatActivity {   // Step 1  TextView tvAnswer;   @Override  protected void onCreate(Bundle savedInstanceState) {  super.onCreate(savedInstanceState);  setContentView(R.layout.*activity\_third*);   // Step 2  tvAnswer = findViewById(R.id.*textView*);  char character = getIntent().getCharExtra("character", ' ');  tvAnswer.setText("Character value received is: " + character);  } } |

1. Add a TextView in the layout of the MainActivity as follow. When the TextView is clicked, a new Activity will be launched and a double value 99.99 will be passed to it.

|  |  |
| --- | --- |
| C:\Users\denise_quek\Desktop\Screenshot_1589973766.png | C:\Users\denise_quek\Desktop\Screenshot_1589973799.png |

Paste your code snippets for the passing and the receiving in the box below.

Hint: you can also set an OnClick listener for a TextView.

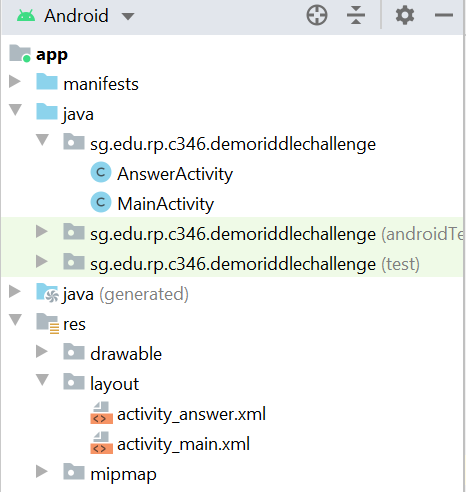
|  |
| --- |
| PASSING (in MainActivity):  package sg.edu.rp.c346.id22001027.demodatapassingtest;  import android.content.Intent; import android.os.Bundle; import android.view.View; import android.widget.Button; import android.widget.TextView;  import androidx.appcompat.app.AppCompatActivity;  public class MainActivity extends AppCompatActivity {   // Event handling Step 1  Button btnPassInteger;  Button btnPassCharacter;  TextView tvPassDouble;   @Override  protected void onCreate(Bundle savedInstanceState) {  super.onCreate(savedInstanceState);  setContentView(R.layout.*activity\_main*);   // Event handling Step 2  btnPassInteger = findViewById(R.id.*buttonPassInteger*);  btnPassCharacter = findViewById(R.id.*buttonPassChar*);  tvPassDouble = findViewById(R.id.*textViewPassDouble*);   // Event handling Step 3 - Pass Integer  btnPassInteger.setOnClickListener(new View.OnClickListener() {  @Override  public void onClick(View view) {  Intent intent = new Intent(MainActivity.this, SecondActivity.class);  intent.putExtra("value", 1);  startActivity(intent);  }  });   // Event handling Step 3 - Pass Character  btnPassCharacter.setOnClickListener(new View.OnClickListener() {  @Override  public void onClick(View view) {  Intent intent = new Intent(MainActivity.this, ThirdActivity.class);  intent.putExtra("character", 'a');  startActivity(intent);  }  });   // Event handling Step 3 - Pass Double  tvPassDouble.setOnClickListener(new View.OnClickListener() {  @Override  public void onClick(View view) {  Intent intent = new Intent(MainActivity.this, FourthActivity.class);  intent.putExtra("doubleValue", 99.99);  startActivity(intent);  }  });  } }  RECEIVING (in FourthActivity):  package sg.edu.rp.c346.id22001027.demodatapassingtest;  import androidx.appcompat.app.AppCompatActivity;  import android.content.Intent; import android.os.Bundle; import android.widget.TextView;  public class FourthActivity extends AppCompatActivity {   TextView tvAnswer;   @Override  protected void onCreate(Bundle savedInstanceState) {  super.onCreate(savedInstanceState);  setContentView(R.layout.*activity\_fourth*);   tvAnswer = findViewById(R.id.*textViewPassDouble*);  Intent intentReceived = getIntent();  double doubleValue = intentReceived.getDoubleExtra("doubleValue", 0.0);  tvAnswer.setText("Double value received is: " + doubleValue);  } } |

## Additional Challenge

In the Riddle project, we create two Answer Activities to display the two answers. Obviously, this design is not code efficient as we expand to more Riddle Questions.

A better approach is to employ only one Activity to reveal the answer depending on the Riddle Question number that is passed over.

Enhance the Riddle project using this better approach. The Android View of the modified project is as shown below.



|  |  |
| --- | --- |
|  | **Learning Checkpoint 2** |
| By now, you should be able to   * Write an Android program with 2 or more Activities. * Write code to launch an Activity using Intent. * Write code to pass and receive data between 2 activities using Intent.   To recap on what we have learnt so far,  In the above exercise, we reckoned that app would need more than 1 Activity for reasons such as supporting the aesthetic integrity of the app, or infusing richer functionalities into the app. To implement Intent, it will be as easy as writing the following 3 lines of code.  Intent intent = new Intent(MainActivity.this, AnswerActivity.class); //Create an Intent object  intent.putExtra("Question",2); //Pass data: name is “Question”; value is 2  startActivity(intent); //Launch the AnswerActivity | |

# Section C: Activity Lifecycle and Intent

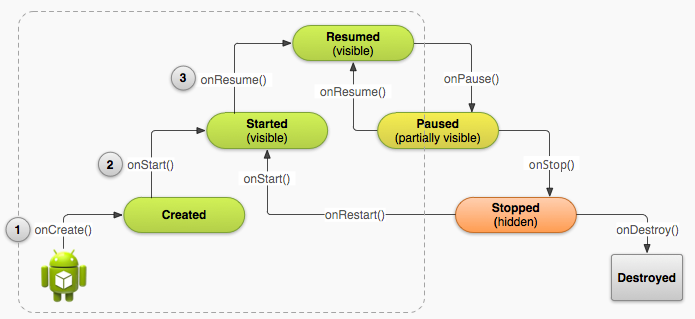


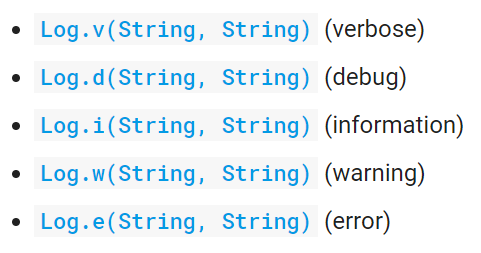
Diagram above shows the Android Activity Lifecycle. In the whole lifecycle, it goes through states such as “**Created**”, “**Started**”, “**Resumed**”, etc. Moreover, it executes callback methods like “**onCreate()**”, “**onStart()**” and so on.

In P03, we have learnt the method “**onCreate()**”, which is the first callback method to be run when the app is launched. It performs all initializations and the required UI setup.

Now, let’s trace the executing path of an Android Activity using the logcat API.

## Learning Activity Lifecycle Using Logcat

The followings are the different methods in the logcat API we learnt in the last problem.



Let’s apply the “Log.d(String, String)” method (used for debugging purposes) to trace the executing path of the Riddle app.

1. Launch the **Riddle app** project again.
2. Copy the following highlighted code and paste it **into** the **MainActivity** class.

Note:

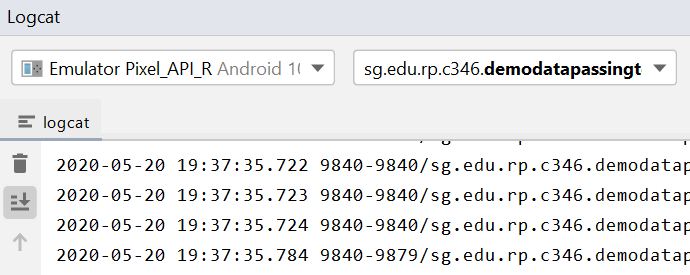
* Place the highlighted methods **after** the **onCreate()** method.
* Press “Alt+Enter” to import the necessary class.

|  |
| --- |
| @Override **protected void** onCreate(Bundle savedInstanceState) {**. . .**}  @Override **protected void** onStart() {  Log.*d*(**"MainActivity"**, **"onStart() called."**);  **super**.onStart(); } @Override **protected void** onResume() {  Log.*d*(**"MainActivity"**, **"onResume() called."**);  **super**.onResume(); } @Override **protected void** onPause() {  Log.*d*(**"MainActivity"**, **"onPause() called."**);  **super**.onPause(); } @Override **protected void** onStop() {  Log.*d*(**"MainActivity"**, **"onStop() called."**);  **super**.onStop(); } @Override **protected void** onDestroy() {  Log.*d*(**"MainActivity"**, **"onDestroy() called."**);  **super**.onDestroy(); } @Override **protected void** onRestart() {  Log.*d*(**"MainActivity"**, **"onRestart() called."**);  **super**.onRestart(); }  The end of “onCreate()” |

1. Add the following highlighted line of code to the **onCreate()** method.

|  |
| --- |
| @Override **protected void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);  setContentView(R.layout.***activity\_main***);  Log.*d*(**"MainActivity"**, **"onCreate() called."**); |

1. Clear the old message in the logcat so that you can easily examine the log content output from your app.



1. Run your app and check the message generated in the logcat monitor.

Hint: You can filter by “MainActivity” to show logs that contain “MainActivity” only.

|  |  |
| --- | --- |
| ? | What is the sequence of the executing among the methods that you added in step 2? i.e. When the app is first opened |
| onCreate() - This method is called when the activity is being created. It initializes the activity and sets the layout. The log statement Log.d("MainActivity", "onCreate() called."); is executed within this method.  onStart() - This method is called after onCreate() and indicates that the activity is about to become visible to the user. The log statement Log.d("MainActivity", "onStart() called."); is executed within this method.  onResume() - This method is called after onStart() and indicates that the activity is now in the foreground and has focus. The log statement Log.d("MainActivity", "onResume() called."); is executed within this method. | |

|  |  |
| --- | --- |
| ? | Referring to the Activity Lifecycle diagram at the start of this section, what is the current state of the **MainActivity**? What state(s) did the Activity go through? |
| Created: The log statement onCreate() called. indicates that the activity was created. This is the initial state of the activity.  Started: The log statement onStart() called. indicates that the activity transitioned to the started state. This state means that the activity is now visible to the user but may not have focus.  Resumed: The log statement onResume() called. indicates that the activity transitioned to the resumed state. This state represents the activity being in the foreground and actively interacting with the user.  Paused: The log statement onPause() called. indicates that the activity transitioned to the paused state. This state occurs when the activity loses focus but is still partially visible to the user.  Stopped: The log statement onStop() called. indicates that the activity transitioned to the stopped state. This state occurs when the activity is no longer visible to the user. | |

1. Click on any of the button in the app. Fill in the methods called in order based on the logcat output.

|  |
| --- |
|  |

|  |  |
| --- | --- |
| ? | Referring to the Activity Lifecycle diagram, what is current state of the **MainActivity**? Can you see the **MainActivity** now? What state(s) did the Activity go through? |
| The current state of the MainActivity is "onStop()" as indicated by the log message "onStop() called." The MainActivity was visible before it entered the "onStop()" state.  State that the activity go through: onPause(), onStop() | |

1. Click on the “Back” button.



Fill in the methods called in order based on the logcat output.

|  |
| --- |
| onRestart(), onStart(), onResume() |

|  |  |
| --- | --- |
| ? | Referring to the Activity Lifecycle diagram, what is current state of the **MainActivity**? Can you see the **MainActivity** now? What state(s) did the Activity go through? |
| The MainActivity is currently visible and active. The state that the activity go through is onRestart(), onStart(), onResume() | |

1. Let’s monitor the program flow in the other Activity class, **AnswerActivity1**. Similar to step 2, copy the following highlighted code and paste it **into** the **AnswerActivity1** class.

|  |
| --- |
| @Override  **protected void** onCreate(Bundle savedInstanceState) {. . .}  @Override **protected void** onStart() {  Log.*d*(**"AnswerActivity1"**, **"onStart() called."**);  **super**.onStart(); } @Override **protected void** onResume() {  Log.*d*(**"AnswerActivity1"**, **"onResume() called."**);  **super**.onResume(); } @Override **protected void** onPause() {  Log.*d*(**"AnswerActivity1"**, **"onPause() called."**);  **super**.onPause(); } @Override **protected void** onStop() {  Log.*d*(**"AnswerActivity1"**, **"onStop() called."**);  **super**.onStop(); } @Override **protected void** onDestroy() {  Log.*d*(**"AnswerActivity1"**, **"onDestroy() called."**);  **super**.onDestroy(); } @Override **protected void** onRestart() {  Log.*d*(**"AnswerActivity1"**, **"onRestart() called."**);  **super**.onRestart(); } |

1. Similar to step 3, add the following line of code to the **onCreate()** method.

|  |
| --- |
| Log.*d*(**"AnswerActivity1"**, **"onCreate() called."**); |

1. Clear the logcat, and run the application.
2. Examine the program flow using logcat messages. It should yield the following sequences. Fill in the missing ones highlighted in yellow.

|  |  |  |
| --- | --- | --- |
| Action | Method Called by  MainActivity | Method Called by  AnswerActivity1 |
| Start Application | onCreate() | ----- |
|  | onStart() | ----- |
|  | onResume() | ----- |
| Click on “Reveal” button | onPause() | ----- |
|  | ----- | onCreate() |
|  | ----- | onStart() |
|  | ----- | onResume() |
|  | onStop() | ----- |
| Click on “Back” button | ----- | onPause() |
|  | onRestart() | ----- |
|  | onStart() | ----- |
|  | onResume() | ----- |
|  | ----- | onStop() |
|  | ----- | onDestroy() |

|  |  |
| --- | --- |
|  | **Learning Checkpoint 3** |
| By now, you should be able to   * Understand the Android Activity Lifecycle * Identify the lifecycle callback methods executed in order * Identify the app’s current state and states gone through   To recap on what we have learnt so far,  To navigate transitions between stages of the Activity Lifecycle, the Activity class provides a core set of six callbacks: [onCreate()](https://developer.android.com/reference/android/app/Activity.html#onCreate(android.os.Bundle)), [onStart()](https://developer.android.com/reference/android/app/Activity.html" \l "onStart()), [onResume()](https://developer.android.com/reference/android/app/Activity.html" \l "onResume()), [onPause()](https://developer.android.com/reference/android/app/Activity.html" \l "onPause()), [onStop()](https://developer.android.com/reference/android/app/Activity.html" \l "onStop()), and [onDestroy()](https://developer.android.com/reference/android/app/Activity.html#onDestroy()). The system invokes each of these callbacks as an Activity enters a new state. | |

# Back to the Problem Statement

## Create a New Project for the App

Create a new Android project with the following details:

|  |  |
| --- | --- |
| **Project Template** | Empty Views Activity |
| **Application Name** | My Modules |
| **Package Name** | sg.edu.rp.c346.id<your student ID>.mymodules |
| **Project Location** | D:\CodeSpace\P05\MyModules |
| **Language** | Java |
| **Minimum API Level** | API 24 |

**GitHub repository URL**

|  |
| --- |
|  |

## Design the Layout

Based on the Problem Statement, design the layout according to the screenshot given.

* Change the layout to LinearLayout (vertical)
* Assign proper IDs to the UI elements
* Use string resources to display all the texts

## Handle the User Events

Continue to finish up the rest of the code.

* Create another Activity called **ModuleDetailActivity**, and design the layout according to the screenshot given in the Problem Statement.
* In **MainActivity**, add code to pass relevant information to **ModuleDetailActivity** using **Intent** when any TextView is clicked.
* In **ModuleDetailActivity**, add code to retrieve information passed from **MainActivity** and display it on the UI.

## Additional Challenge

Provide a button on the UI of the **ModuleDetailActivity** for the user to navigate back to the **MainActivity** when the button is clicked.