


R U SMARTER THAN A 5TH GRADER?

PART 3

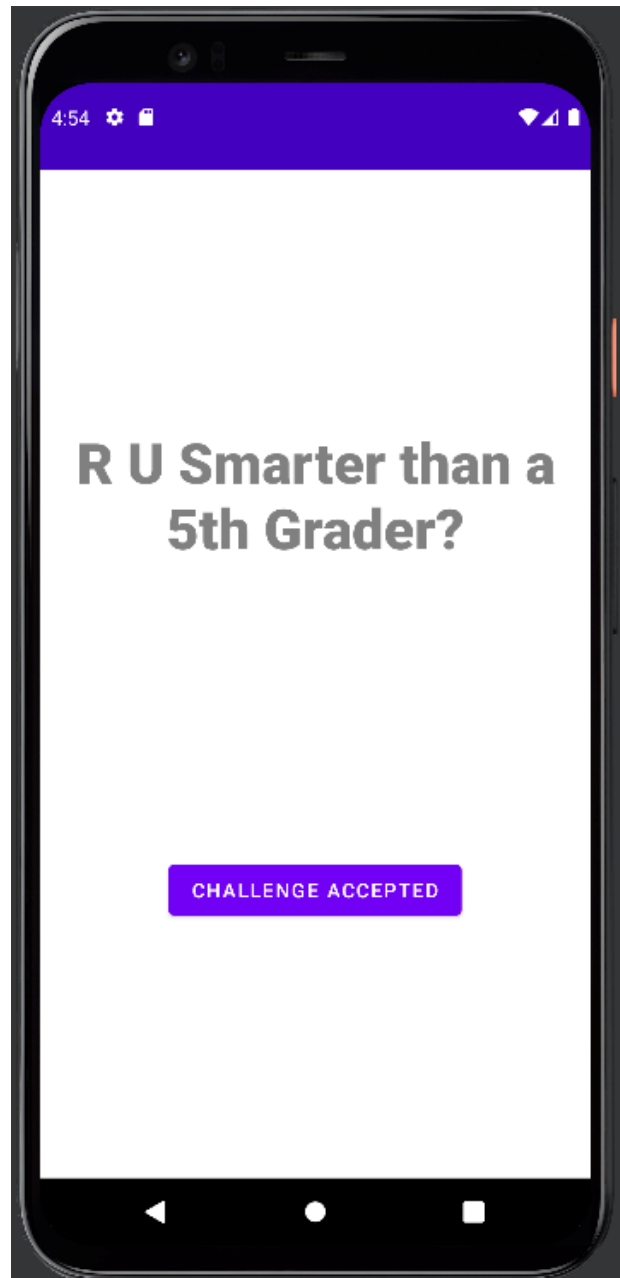
In the previous part of the project, you defined how to restart the game from the result screen and how to display the game's questions for the player to answer.

For this final part of the project, you'll be defining how to capture the player's answer for each question in the game, as well as what happens when that answer is submitted. Finally, you'll define what the result screen should tell the player in the case they won, lost, or cheated in the game.

PART 1: OPENING THE PROJECT

Open the provided project template and run the app (clicking on the "Run app" icon  or "Debug app" icon  located in the menu near the top right-hand side of the Android Studio window).

You will see that the app launches on the emulator to the title screen of the game.



Navigating through the app, you'll come to the first question screen. You will not be able to progress beyond this screen whether or not you answer the question correctly.

5:55

Question 1: Grade 1 Math

$1 + 1 = ?$

Your answer

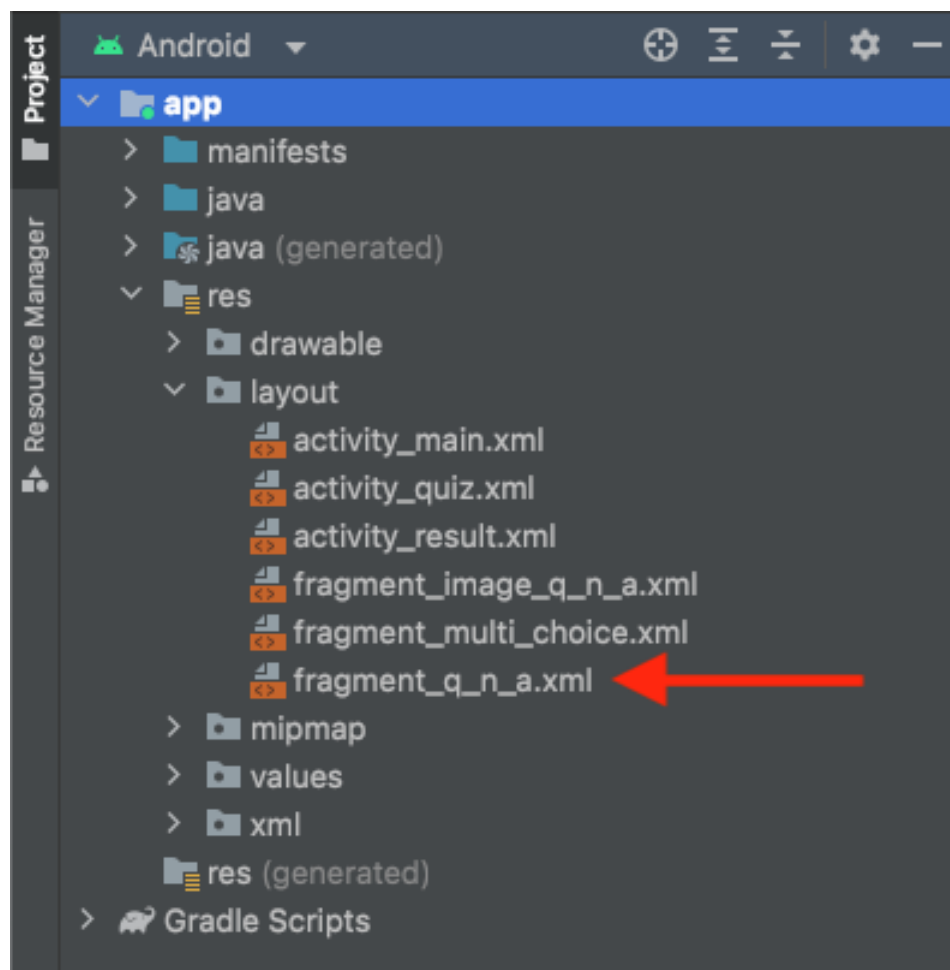
SUBMIT

PART 2: CAPTURING THE PLAYER'S ANSWER

To be able to progress in the game, the game needs to know what the player's answer is to be able to determine whether they can proceed further. Capturing the player's answer depends on the type of question and how they are able to input their answer. As we've seen in Part 2 of this project, there are 3 types of questions in this game:

Type #1: Question and Answer

For this type of question, the player simply types in an answer to the given question. Open the layout file "fragment_q_n_a.xml" under app/res/layout folder, which is where you had defined the layout for this type of question in Part 2 of this project.



An EditText (ID: "qna_answer") is included in this layout for the player to enter their answer.

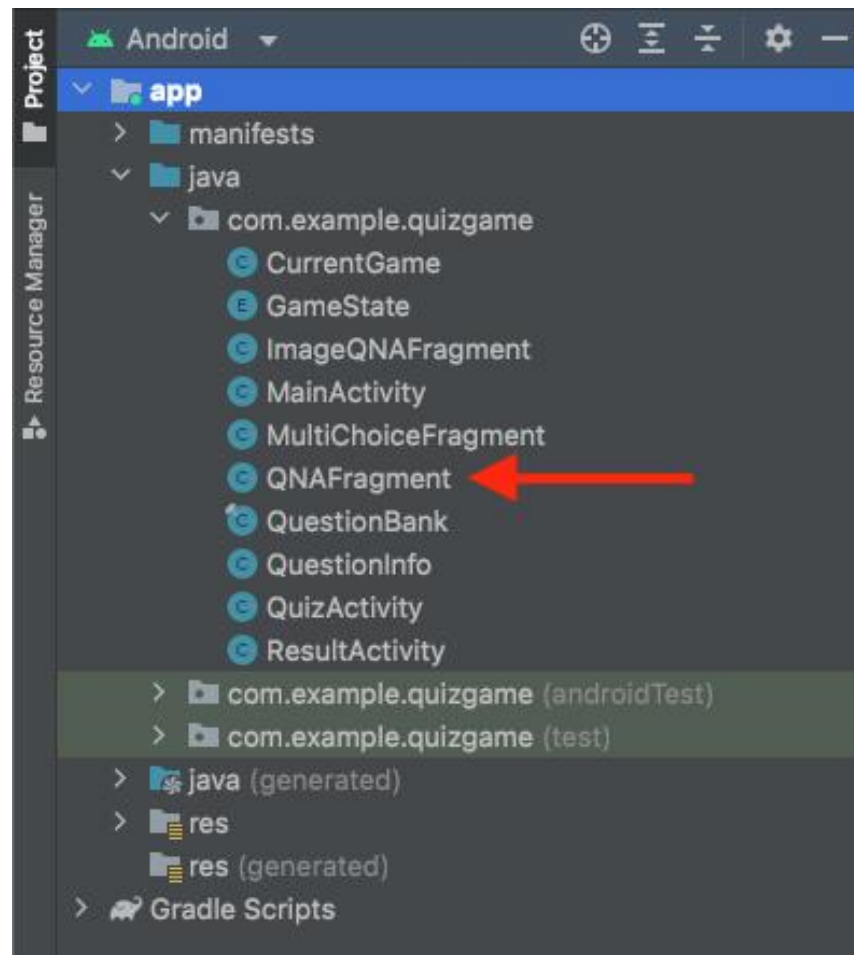
```
21 <LinearLayout
22     android:layout_width="match_parent"
23     android:layout_height="wrap_content"
24     android:orientation="vertical"
25     android:gravity="center">
26
27     <TextView...>
28
35
36     <EditText
37         android:id="@+id/qna_answer"
38         android:layout_width="match_parent"
39         android:layout_height="wrap_content"
40         android:hint="Your answer"
41         android:layout_marginTop="16dp"
42         android:layout_marginBottom="16dp" />
43
44 </LinearLayout>
```

For all questions of this type, the game needs to know what the player is typing into the EditText "qna_answer", since this could be their final answer before pressing the "submit" button.

Near the top of this layout file, you'll see that this layout is associated with QNAFragment.

```
fragment_q_n_a.xml
1 <?xml version="1.0" encoding="utf-8"?>
2 <androidx.appcompat.widget.LinearLayoutCompat xmlns:android="http://schemas.android.com/apk/res/android"
3     xmlns:app="http://schemas.android.com/apk/res-auto"
4     xmlns:tools="http://schemas.android.com/tools"
5     android:layout_width="match_parent"
6     android:layout_height="match_parent"
7     android:gravity="center"
8     android:orientation="vertical"
9     tools:context=".QNAFragment">
10
11     <androidx.cardview.widget.CardView...>
12
50
51 </androidx.appcompat.widget.LinearLayoutCompat>
```

QNAFragment is defined in “QNAFragment.java”. Open the Java file “QNAFragment.java” under app/java/com/example/quizgame folder.



All the information about the current game being played is provided in the variable “currentGame”. Here, we’ll be modifying “currentGame.currentAnswer” to let the game know what the player has typed into the EditText “qna_answer”.

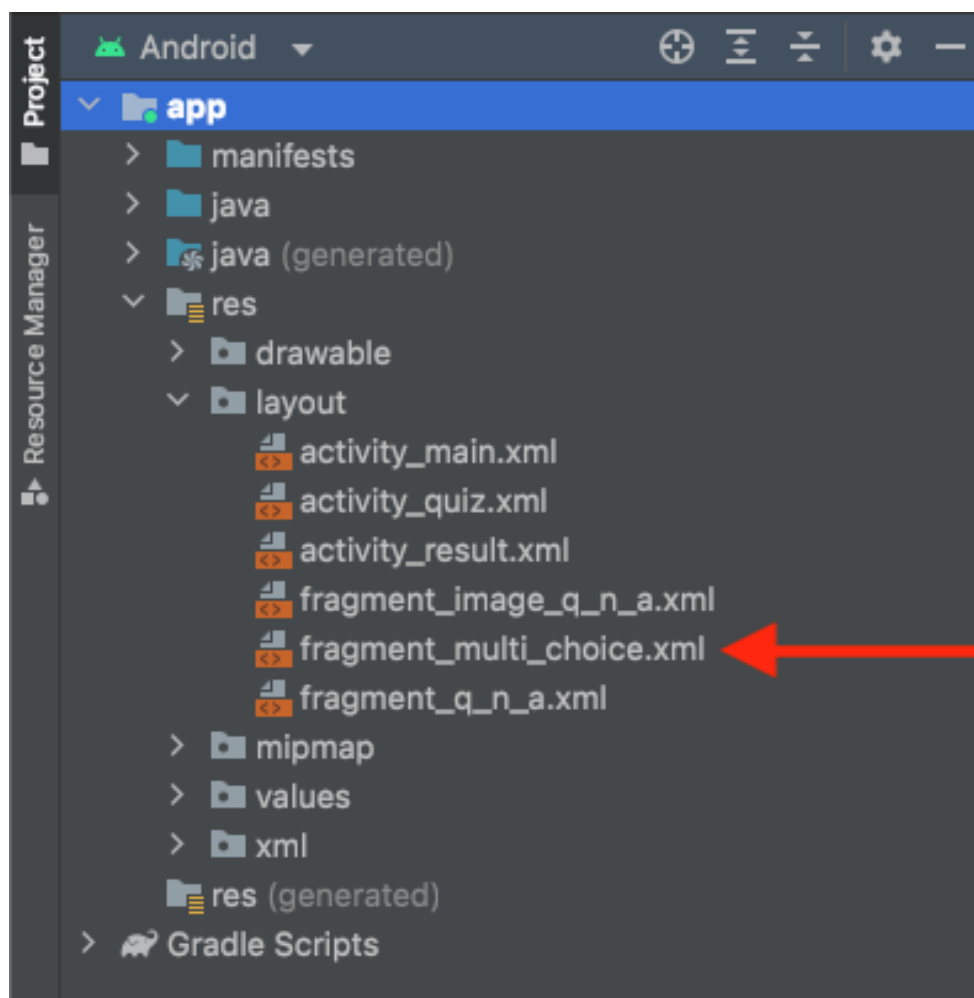
Inside the `linkViews()` function, capture the player’s answer for this type of question. Breaking the process down:

1. Have the EditText “qna_answer” be able to detect that the player is typing in or removing any text from it
2. When text input or removal is detected:
 - a. Extract the currently inputted text from the EditText, typically done with `[editTextObject].getText().toString()`
 - b. Assign “currentGame.currentAnswer” to this extracted text to let the game know that this is the current answer entered by the player

NOTE: Although the code is organized similarly to that in an Activity, common functions you may be used to calling are written slightly differently in a Fragment, ie. `view.findViewById()` instead of `findViewById()`.

Type #2: Multiple Choice

For this type of question, the player selects one of 4 choices provided to answer the given question. Open the layout file “`fragment_multi_choice.xml`” under `app/res/layout` folder, which is where you had defined the layout for this type of question in Part 2 of this project.

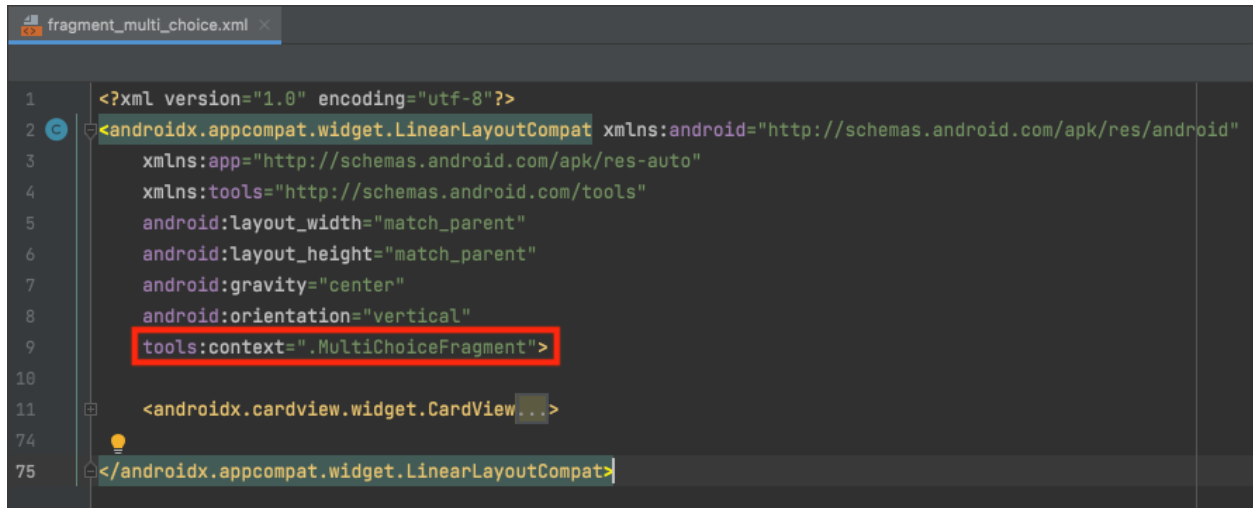


4 Buttons (IDs: “choice_1_button”, “choice_2_button”, “choice_3_button”, “choice_4_button”) are included in this layout for the player to choose their answer.

```
21 <LinearLayout
22     android:layout_width="match_parent"
23     android:layout_height="wrap_content"
24     android:orientation="vertical"
25     android:gravity="center">
26
27     <TextView...>
28
29
30
31
32
33
34
35
36     <Button
37         android:id="@+id/choice_1_button"
38         android:layout_width="wrap_content"
39         android:layout_height="wrap_content"
40         android:layout_marginTop="16dp"
41         android:layout_marginBottom="16dp"
42         android:text="Choice 1"/>
43
44     <Button
45         android:id="@+id/choice_2_button"
46         android:layout_width="wrap_content"
47         android:layout_height="wrap_content"
48         android:layout_marginTop="16dp"
49         android:layout_marginBottom="16dp"
50         android:text="Choice 2"/>
51
52     <Button
53         android:id="@+id/choice_3_button"
54         android:layout_width="wrap_content"
55         android:layout_height="wrap_content"
56         android:layout_marginTop="16dp"
57         android:layout_marginBottom="16dp"
58         android:text="Choice 3"/>
59
60     <Button
61         android:id="@+id/choice_4_button"
62         android:layout_width="wrap_content"
63         android:layout_height="wrap_content"
64         android:layout_marginTop="16dp"
65         android:layout_marginBottom="16dp"
66         android:text="Choice 4"/>
67
68 </LinearLayout>
```

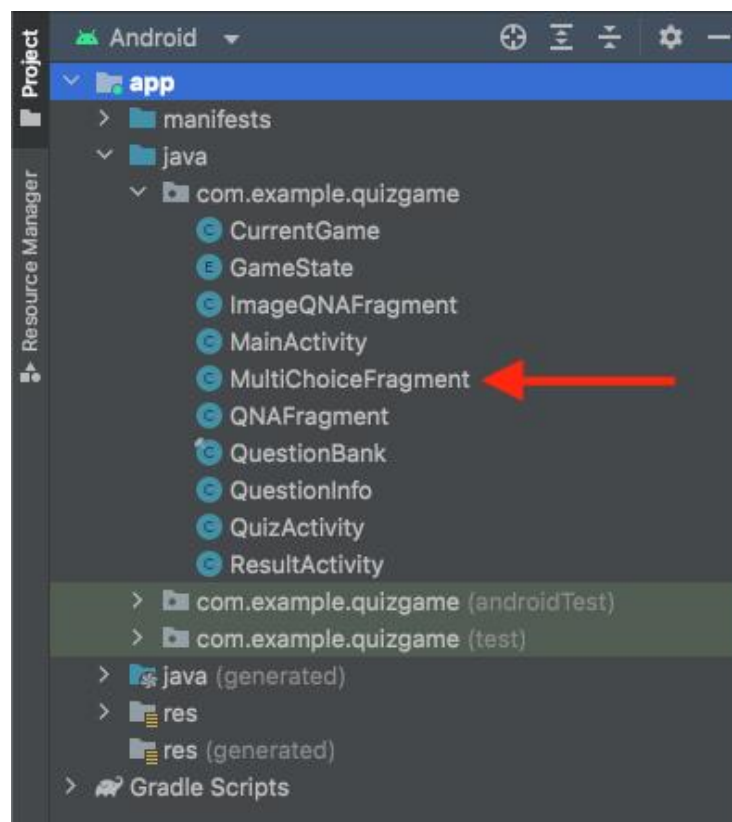
For all questions of this type, the game needs to know which button the player has most recently clicked on, since this could be their final answer before pressing the “submit” button.

Near the top of this layout file, you'll see that this layout is associated with MultiChoiceFragment.



```
1 <?xml version="1.0" encoding="utf-8"?>
2 <androidx.appcompat.widget.LinearLayoutCompat xmlns:android="http://schemas.android.com/apk/res/android"
3     xmlns:app="http://schemas.android.com/apk/res-auto"
4     xmlns:tools="http://schemas.android.com/tools"
5     android:layout_width="match_parent"
6     android:layout_height="match_parent"
7     android:gravity="center"
8     android:orientation="vertical"
9     tools:context=".MultiChoiceFragment">
10
11     <androidx.cardview.widget.CardView ...>
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75 </androidx.appcompat.widget.LinearLayoutCompat>
```

MultiChoiceFragment is defined in “MultiChoiceFragment.java”. Open the Java file “MultiChoiceFragment.java” under app/java/com/example/quizgame folder.

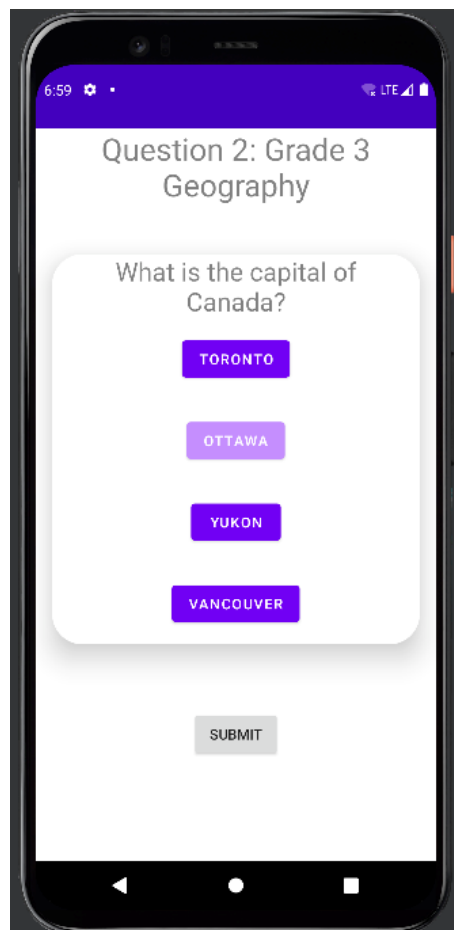


All the information about the current game being played is provided in the variable “currentGame”. Here, we’ll be modifying “currentGame.currentAnswer” to let the game know which button the player has most recently clicked on.

Inside the `linkViews()` function, capture the player’s answer for this type of question. For each Button:

1. Have the Button be able to detect that the player has clicked on it
2. When a Button click is detected:
 - a. Extract the label of the Button, typically done with `[buttonObject].getText().toString()`
 - b. Assign “currentGame.currentAnswer” to this extracted text to let the game know that this is the current answer selected by the player

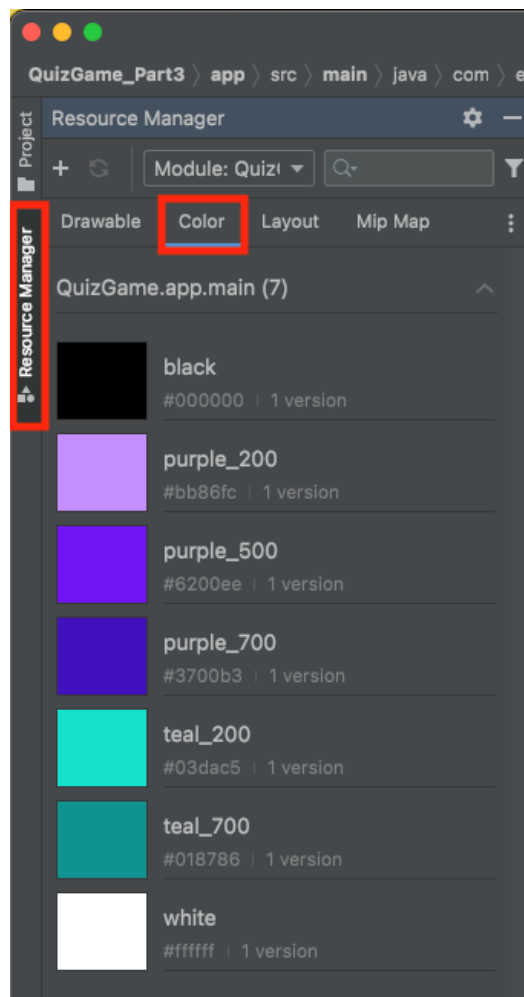
Another thing to note is that when clicking on any of the Buttons, there is no visual feedback letting the player know what their latest selection is. A simple way to let the player know is by changing the color of the chosen Button to indicate that it is now the “selected” choice. In the example below, the selected Button labeled “Ottawa” has changed to a lighter purple color, while the other Buttons are still in the darker purple color:



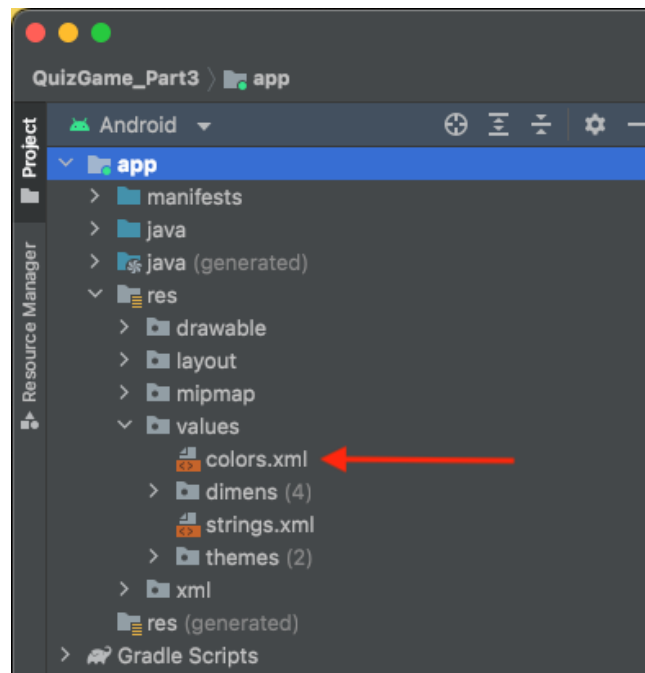
Inside the *linkViews()* function, indicate to the player the choice they have most recently selected. For each Button:

1. When a Button click is detected:
 - a. Change the color of the Button to denote that it is now the “selected” choice, typically done with
`[buttonObject].setBackgroundColor(getResources().getColor(R.color.[colorResourceName]))`
 - b. Change the color of the other Buttons to denote they are now “unselected”. This is important so that the Buttons do not all look like they are “selected” if the player happens to click on all the Buttons before submitting an answer.

You can view the available color resources you have at your disposal by clicking on the “Resource Manager” tab on the left side of the Android Studio window (below the “Project” tab). All color resources can be viewed under the “Color” section. In the Java code, use “R.color.[color_resource_name]” to identify the color resource you want to use.

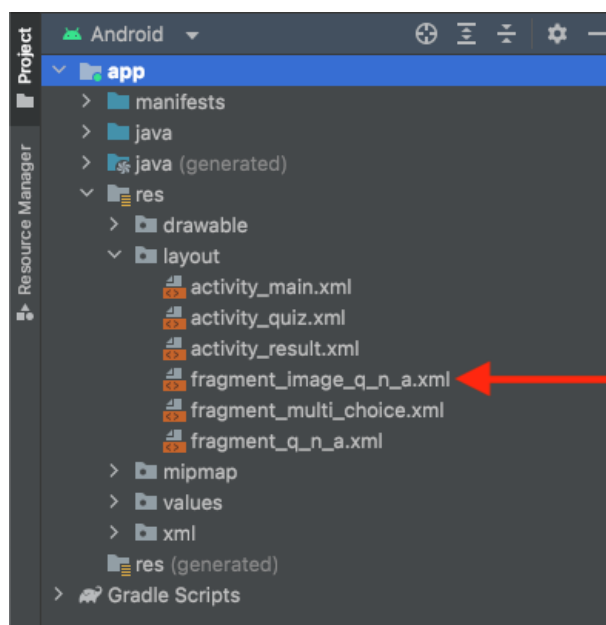


Going back to the project tab, you'll also be able to view available color resources and add your own in the resources file "colors.xml" under app/res/values folder.



Type #3: Question and Answer Regarding an Image

For this type of question, the player types in an answer to the given question about an image. Open the layout file "fragment_image_q_n_a.xml" under app/res/layout folder, which is where you had defined the layout for this type of question in Part 2 of this project.



An EditText (ID: "image_qna_answer") is included in this layout for the player to provide their answer.

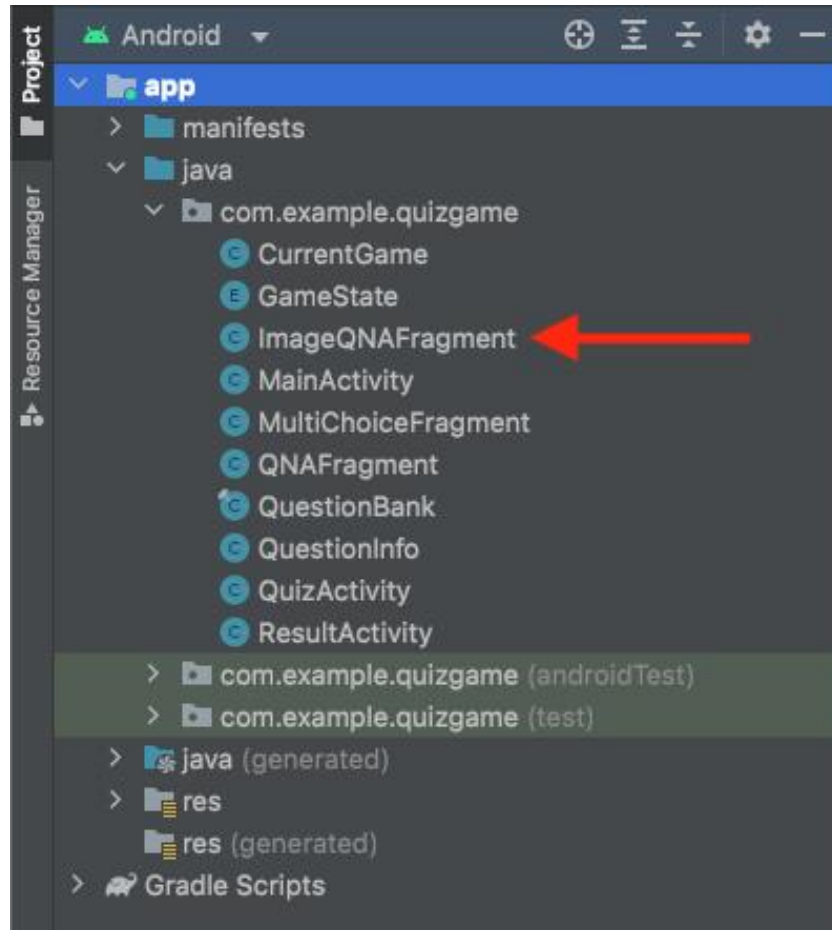
```
21 <LinearLayout
22     android:layout_width="match_parent"
23     android:layout_height="wrap_content"
24     android:orientation="vertical"
25     android:gravity="center">
26
27     <TextView...>
35
36     <ImageView...>
44
45     <EditText
46         android:id="@+id/image_qna_answer"
47         android:layout_width="match_parent"
48         android:layout_height="wrap_content"
49         android:hint="Your answer"
50         android:layout_marginTop="16dp"
51         android:layout_marginBottom="16dp"/>
52
53 </LinearLayout>
```

Similar to question type #1, for all questions of this type, the game needs to know what the player is typing into the EditText "image_qna_answer", since this could be their final answer before pressing the "submit" button.

Near the top of this layout file, you'll see that this layout is associated with ImageQNAFragment.

```
fragment_image_q_n_a.xml
1 <?xml version="1.0" encoding="utf-8"?>
2 <androidx.appcompat.widget.LinearLayoutCompat xmlns:android="http://schemas.android.com/apk/res/android"
3     xmlns:app="http://schemas.android.com/apk/res-auto"
4     xmlns:tools="http://schemas.android.com/tools"
5     android:layout_width="match_parent"
6     android:layout_height="match_parent"
7     android:gravity="center"
8     android:orientation="vertical"
9     tools:context=".ImageQNAFragment">
10
11     <androidx.cardview.widget.CardView...>
60
61 </androidx.appcompat.widget.LinearLayoutCompat>
```

ImageQNAFragment is defined in “ImageQNAFragment.java”. Open the Java file “ImageQNAFragment.java” under app/java/com/example/quizgame folder.



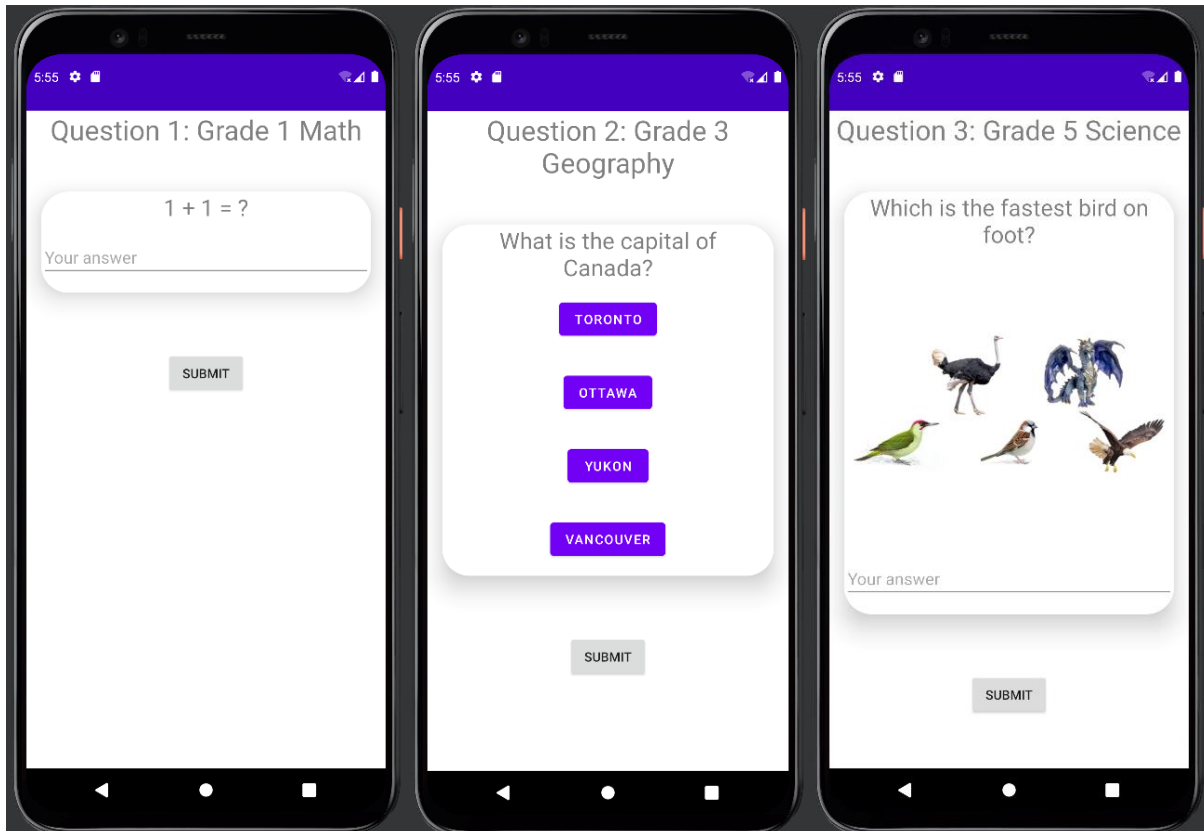
All the information about the current game being played is provided in the variable “currentGame”. Here, we’ll be modifying “currentGame.currentAnswer” to let the game know what the player has typed into the EditText “image_qna_answer”.

Inside the `linkViews()` function, capture the player’s answer for this type of question. Breaking the process down:

1. Have the EditText “image_qna_answer” be able to detect that the player is typing in or removing any text from it
2. When text input or removal is detected:
 - a. Extract the currently inputted text from the EditText, typically done with `[editTextObject].getText().toString()`
 - b. Assign “currentGame.currentAnswer” to this extracted text to let the game know that this is the current answer entered by the player

NOTE: Although the code is organized similarly to that in an Activity, common functions you may be used to calling are written slightly differently in a Fragment, ie. `view.findViewById()` instead of `findViewById()`.

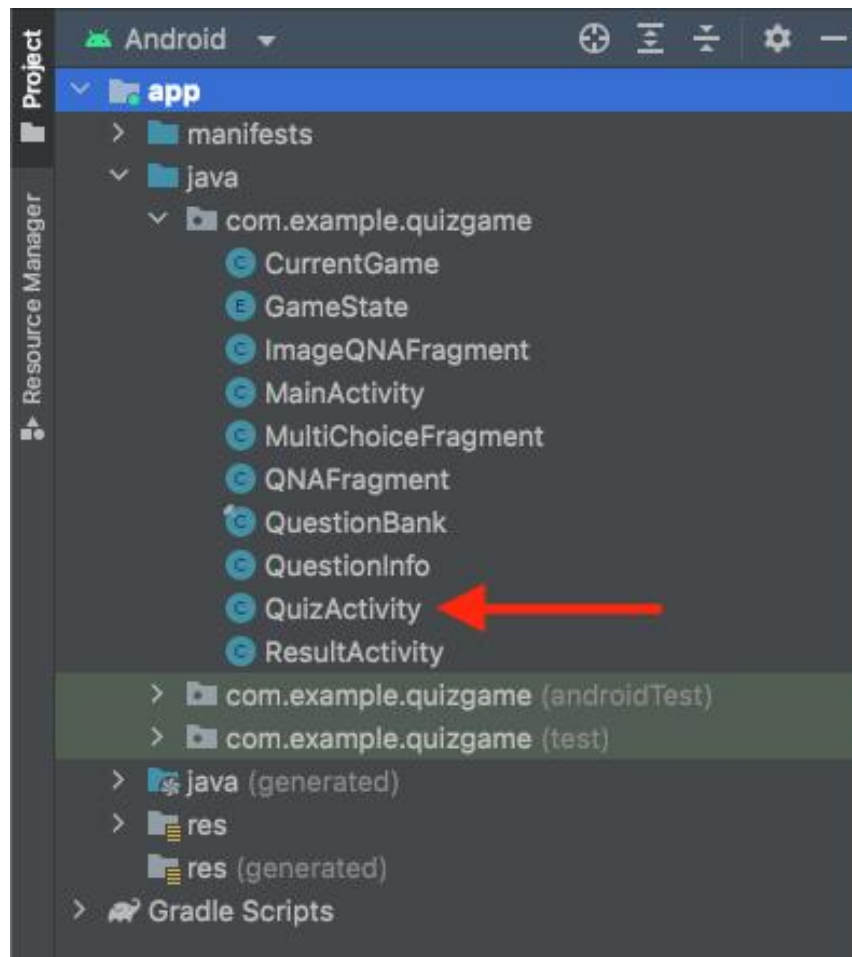
After capturing the answer for all 3 types of questions, you should be able to progress to the other question screens.



PART 3: DEFINING WHAT HAPPENS WHEN AN ANSWER IS SUBMITTED

The question screens in this game are represented by the QuizActivity, which contains a TextView indicating the question category, a Fragment displaying the question itself, and a “submit” button.

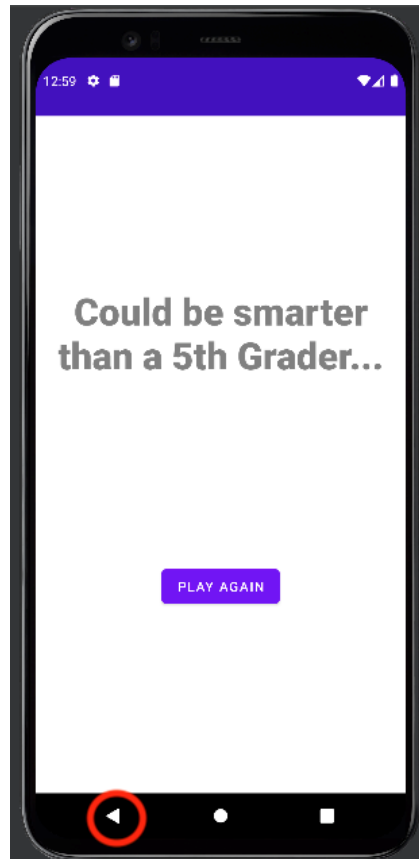
QuizActivity is defined in “QuizActivity.java”. Open the Java file “QuizActivity.java” under app/java/com/example/quizgame folder.



Here, you'll see that in the `linkViews()` function, clicking on the "submit" button checks for 3 things:

Check #1: Check for cheating

In this game, cheating is defined as going back to the quiz screen from the result screen to answer the given question again using the emulator's back button (see below). If the player cheats this way, the game will end immediately and they will be directed back to the result screen.



Check #2: Check whether or not an answer is given

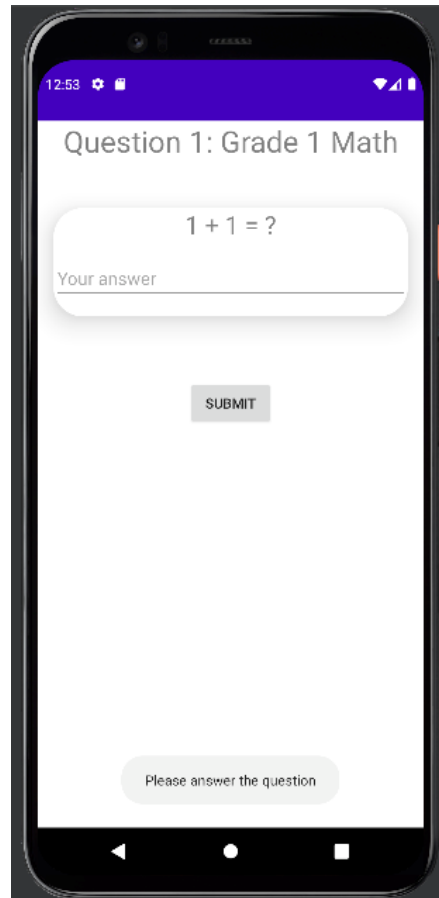
Currently, if the player did not enter or select any answer, there is nothing to let them know that they need to do so.

A simple way to warn the player that they have not given an answer yet is to display a Toast when this happens.

In the `handleBlankAnswer()` function, display a Toast to warn the player that they have not provided any answer to the given question. This is typically done with:

```
Toast.makeText(getApplicationContext(), "Warning message here", Toast.LENGTH_SHORT).show()
```

When running the app again, you'll see that a Toast appears when no answer is given for the current question:



Check #3: Check whether or not the given answer is correct

For any question, there may be only one or there may be multiple correct answers.

In the `checkAnswer()` function, check the player's answer (given as the argument “currentAnswer”) **against a list of accepted answers** (given as the argument “acceptedAnswers”).

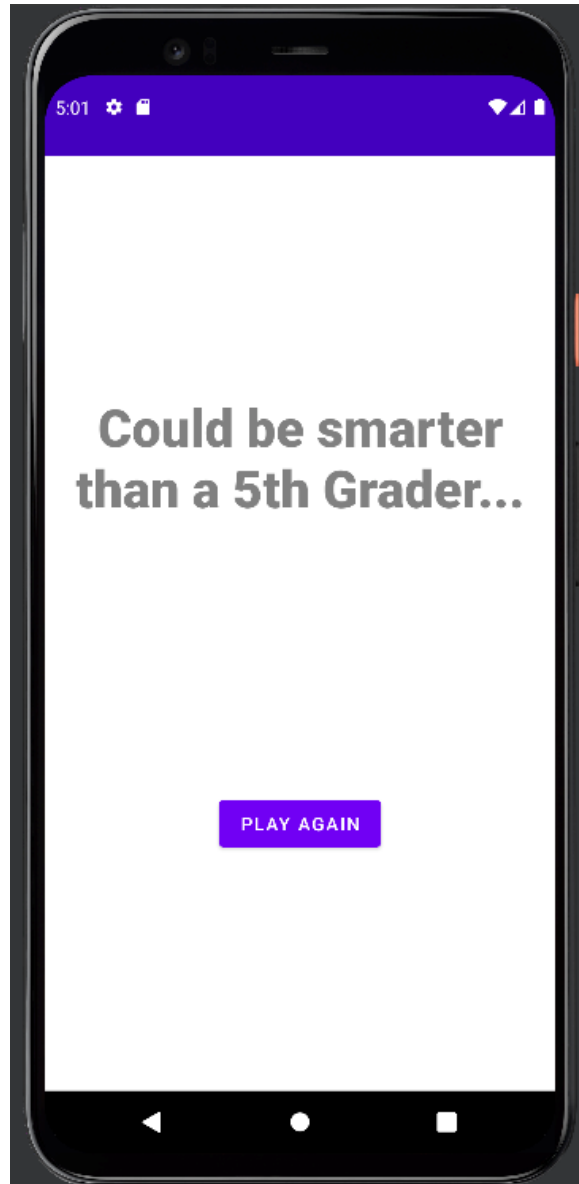
Return true if the player's answer is determined to be correct, and false otherwise.

Things to consider:

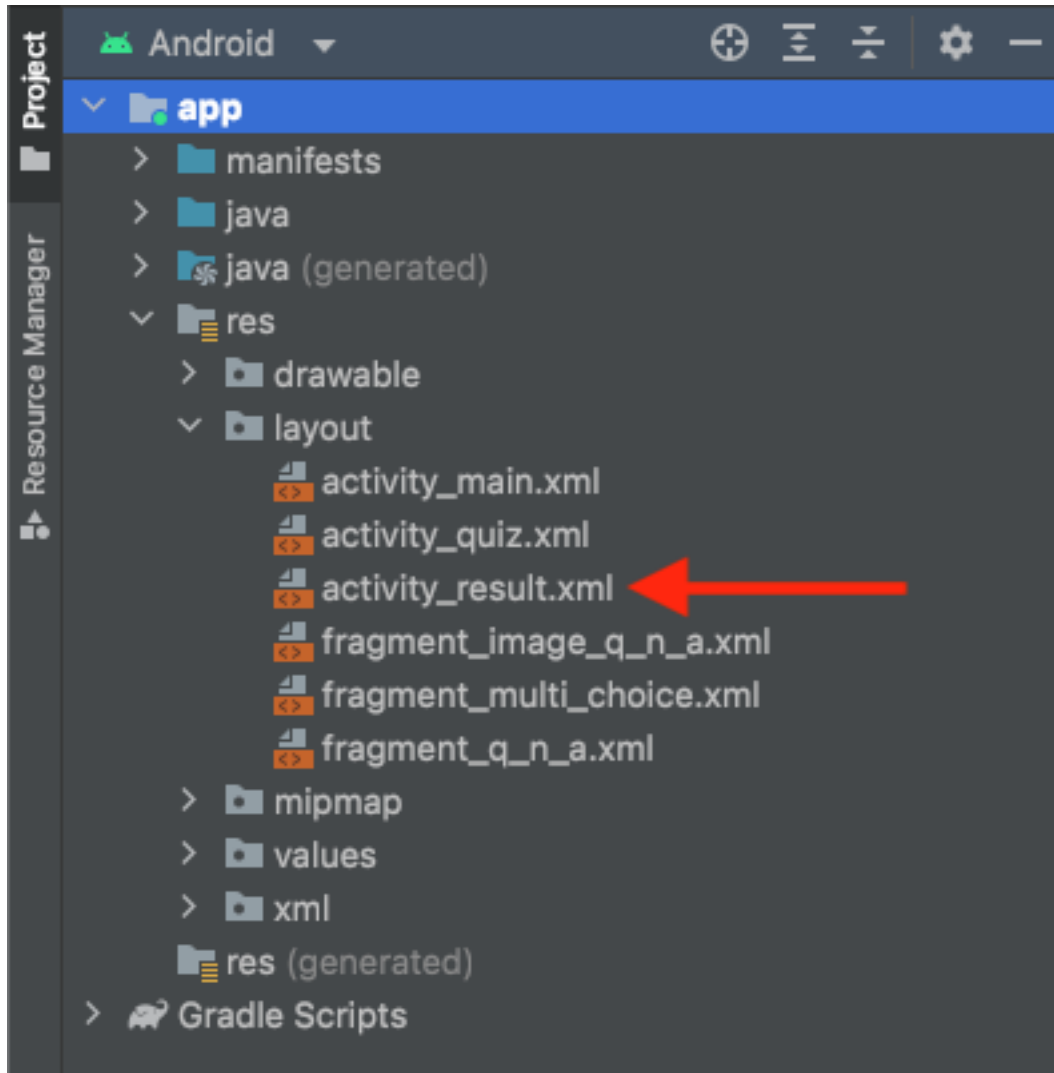
- What if the player entered an answer like “jupiter” when the accepted answer is listed as “Jupiter”? What if the player entered an answer in mixed case, ie. “aNsWeR”?
- What if the player added some extra whitespace before or after the answer?

PART 4: DEFINING WHAT THE RESULT SCREEN TELLS THE PLAYER

Currently, regardless if the player answers all the questions in the game correctly, the result screen will always say that they “could be smarter than a 5th grader...”, which does not tell them much about how they did in the game.



Open the layout file “activity_result.xml” under app/res/layout folder, which is where you had defined the layout for this screen in Part 1 of this project.



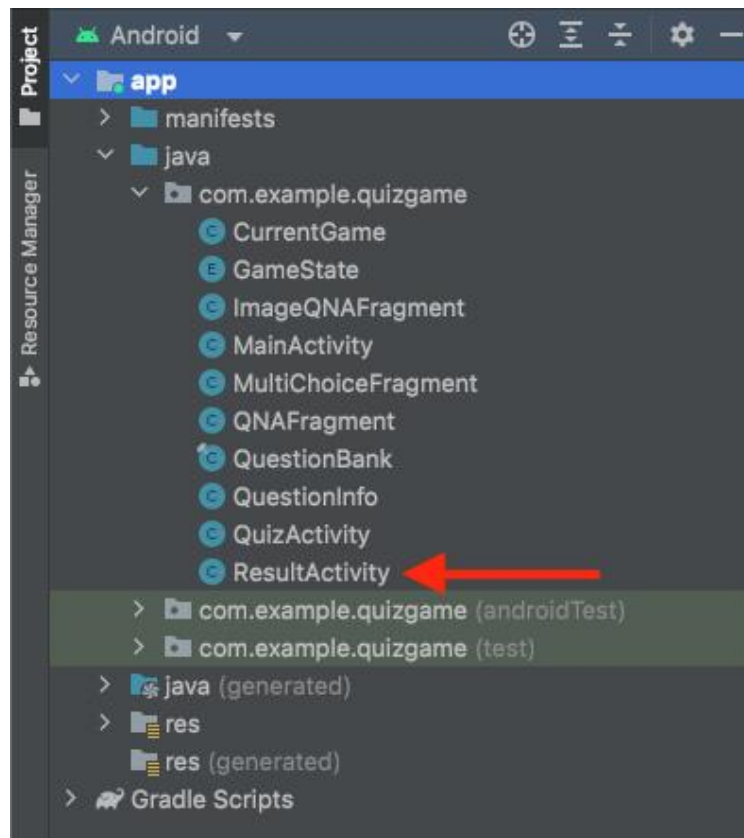
A TextView(ID: "result_text") is included in this layout to let the player know whether or not they are smarter than a 5th grader.

```
9      <TextView
10          android:id="@+id/result_text"
11          android:gravity="center"
12          android:layout_width="wrap_content"
13          android:layout_height="wrap_content"
14          android:textSize="40sp"
15          android:fontFamily="sans-serif-black"
16          android:text="Could be smarter than a 5th Grader..."
17          app:layout_constraintTop_toTopOf="parent"
18          app:layout_constraintLeft_toLeftOf="parent"
19          app:layout_constraintRight_toRightOf="parent"
20          app:layout_constraintBottom_toTopOf="@+id/play_again_button"
21          android:layout_marginTop="144dp"
22          android:layout_marginBottom="16dp"/>
23
24      <Button...>
```

Near the top of this layout file, you'll see that this layout is associated with ResultActivity.

```
activity_result.xml
1  <?xml version="1.0" encoding="utf-8"?>
2  <androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:app="http://schemas.android.com/apk/res-auto"
4      xmlns:tools="http://schemas.android.com/tools"
5      android:layout_width="match_parent"
6      android:layout_height="match_parent"
7      tools:context=".ResultActivity">
```

ResultActivity is defined in “ResultActivity.java”. Open the Java file “ResultActivity.java” under app/java/com/example/quizgame folder.



Inside the *linkViews()* function, define what the result screen should tell the player when they have won, lost, or cheated in the game.

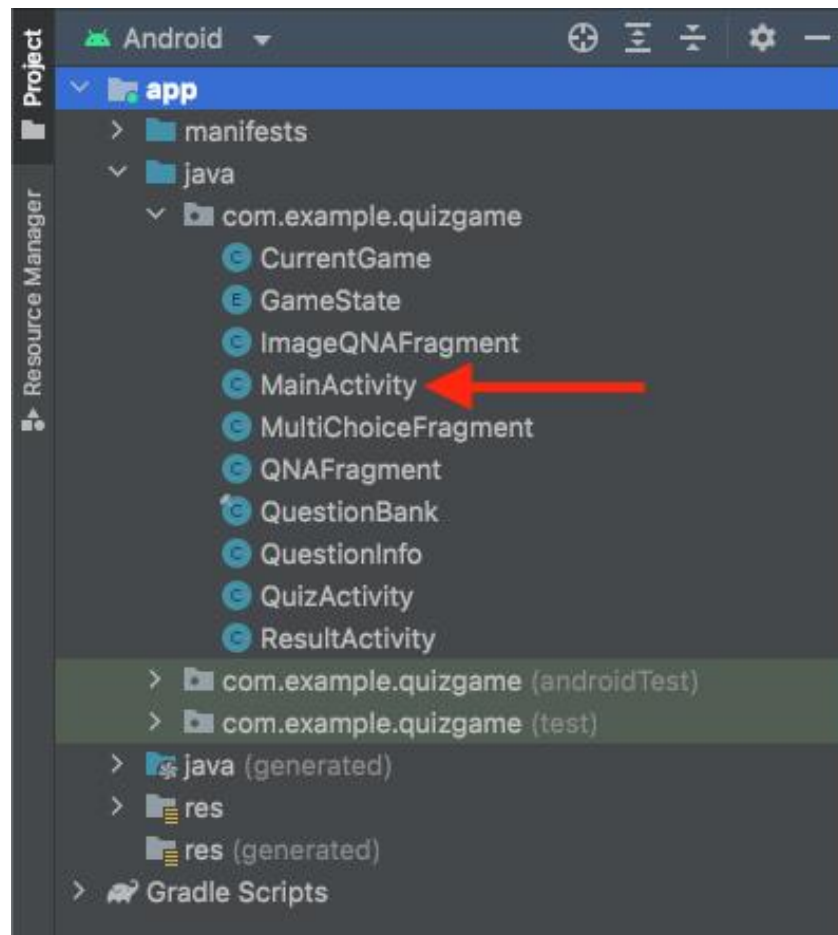
Notice there is a switch statement written in the function. Switch statements operate very similarly to *if/else if/else* code blocks, and are typically used when there are several possible values that an expression can evaluate to. In this case, the expression is “currentGame.currentState” (representing the player’s status in the game), and the possibilities are WIN, LOSE, and CHEAT. The “default” keyword takes care of the *else* case, when the player’s status is none of the other possibilities. Unlike in *if/else if/else* code blocks, the “break” keyword is needed in a switch statement to exit the entire code block after executing the code specified under the possibility that matches the value of the expression.

PART 5: TESTING THE APP

When you are now answering any question in the game, the game itself should be able to capture your answer and compare it to the list of accepted answers for that question to determine whether you can continue. If any question is left unanswered, you’ll be warned about this and won’t be able to move on until you provide an answer. For multiple choice

questions, you should always be able to easily identify the current answer you have selected. Whenever you reach the result screen, whether that is through answering all questions correctly (winning), answering any of the questions incorrectly (losing), or going back to answer a question after already winning or losing (cheating), the message on the screen should be able to tell you which of these 3 situations you are in.

After implementing and debugging through all the features in this game, you can now load and play the full game. Open the Java file “MainActivity.java” under app/java/com/example/quizgame folder.



MainActivity represents the game’s title screen, and clicking on the “CHALLENGE ACCEPTED” Button sets up the game before starting it. Inside the function *setupGame()*, a call is made to the *CurrentGame.loadDemoGameQuestions()* function to load the demo game, which consists of the 3 questions we were working with when implementing and debugging the features of this game. Replace this call with a call to *CurrentGame.generateFullGameQuestions()* to load the full game, which consists of 10 questions + 1 final bonus question. Win this version of the game to prove you’re smarter than a 5th grader!