R U SMARTER THAN A 5TH GRADER?

PART 2

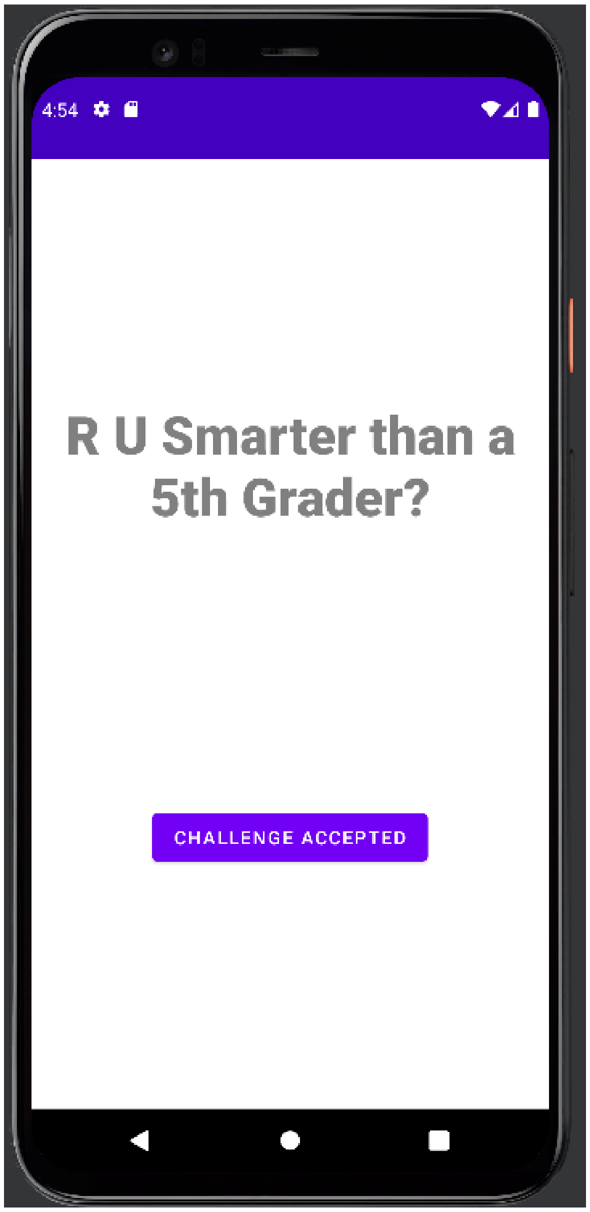
In Part 1, you created the layout of the result screen, which lets the player know if they won, lost, or cheated in the game, and gives them the option to play the game again.

For Part 2, you'll be defining how to restart the game from the result screen, as well as how to display the game's questions for the player to answer.

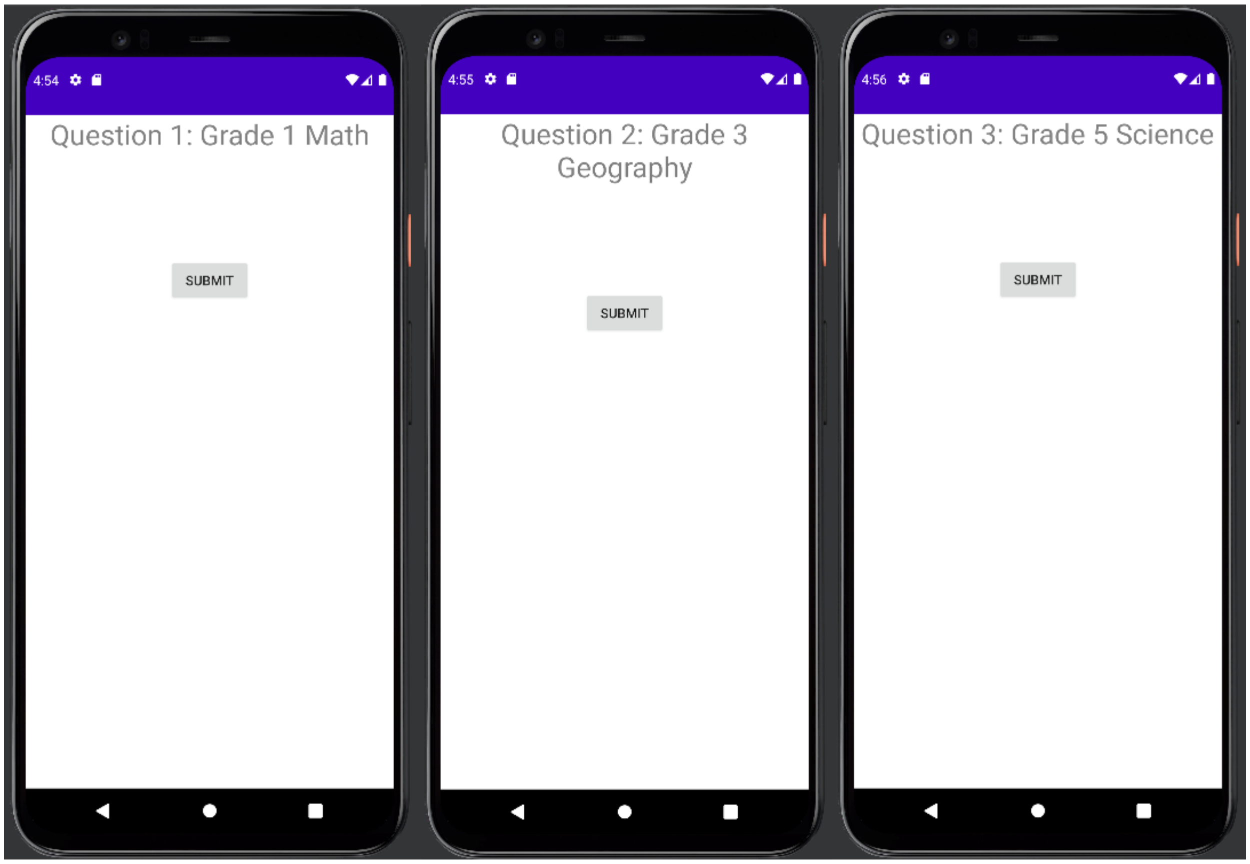
PART 1: OPENING THE PROJECT

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).

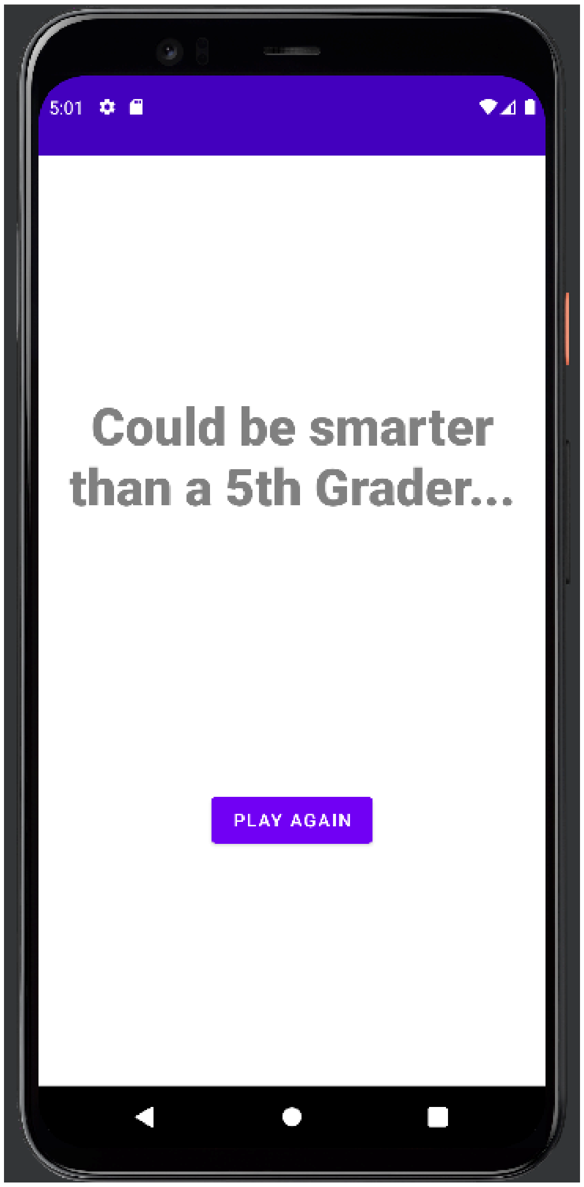
You will see that the app launches on the emulator to the title screen of the game.



Navigating through the app, you'll come across screens titled with a question category, but there is no actual question to be answered.

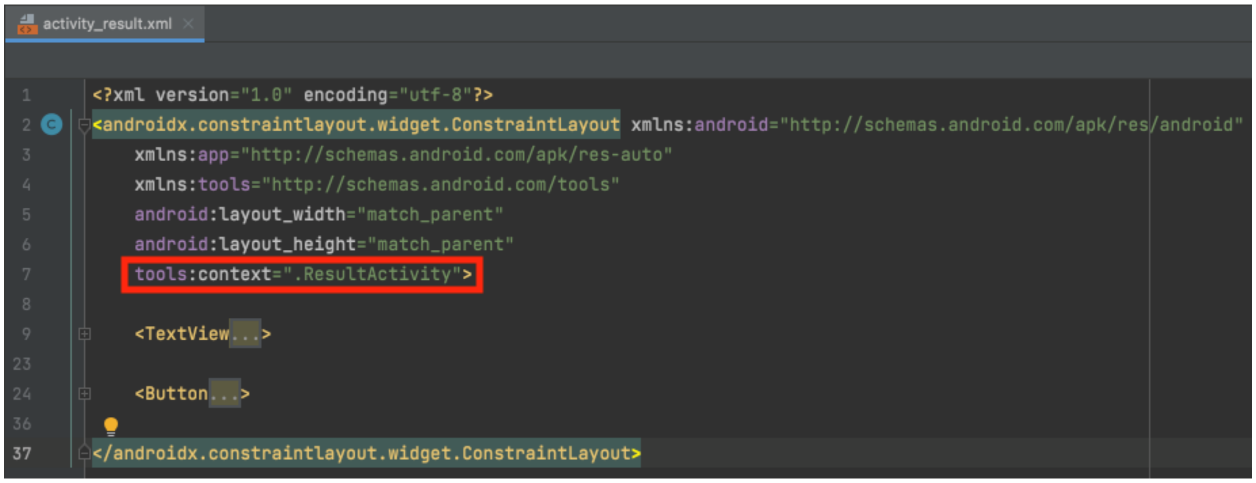


Finally, you'll come to the result screen for which you had created the layout in Part 1. Currently, clicking on the "Play Again" Button does not affect anything in the app.



PART 2: DEFINING HOW THE GAME RESTARTS

Looking at the layout file for the result screen (“activity_result.xml” under app/res/layout folder), you’ll see it is associated with ResultActivity, which represents the result screen in how it is displayed to the player and the events that occur when the player interacts with it.


activity_result.xml

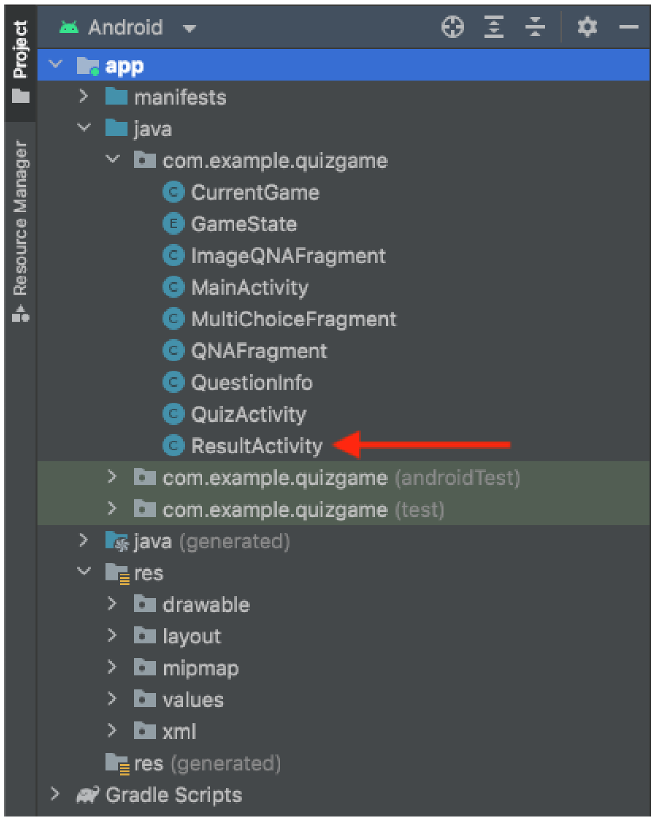
```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
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

    <TextView...>

    <Button...>

</androidx.constraintlayout.widget.ConstraintLayout>
```

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



The *onCreate()* function inside ResultActivity populates the result screen to look like how it was defined in “layout_result.xml”. A call is made to the *linkViews()* function to define how the UI components on the screen are shown and what happens when the player interacts with them.

Inside the *linkViews()* function, have the “Play Again” Button (ID: “play_again_button”) restart the game when it is clicked. The logic of restarting the game is conveniently provided to you as a function called *restartGame()*.

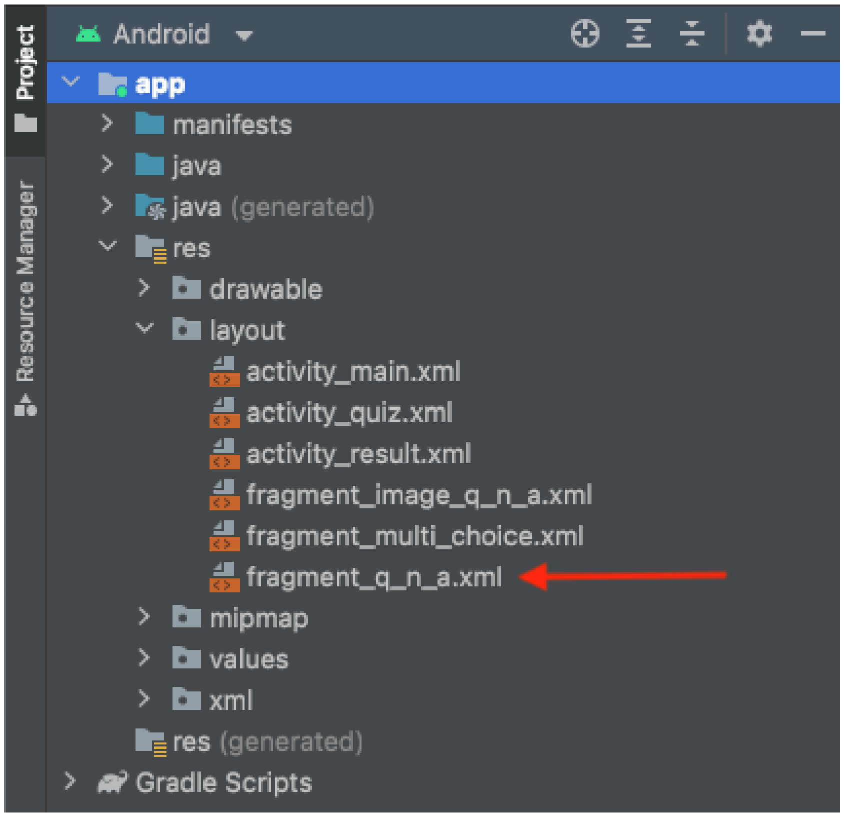
PART 3: DEFINING HOW TO DISPLAY THE GAME’S QUESTIONS

NOTE: In this section, you’ll be working with Fragments, which are like reusable mini-Activities. Fragments are used in this game to define how questions of a specific type are displayed and the events that happen when the player interacts with the UI components involved.

In this game, there are 3 question types:

Type #1: Question and Answer

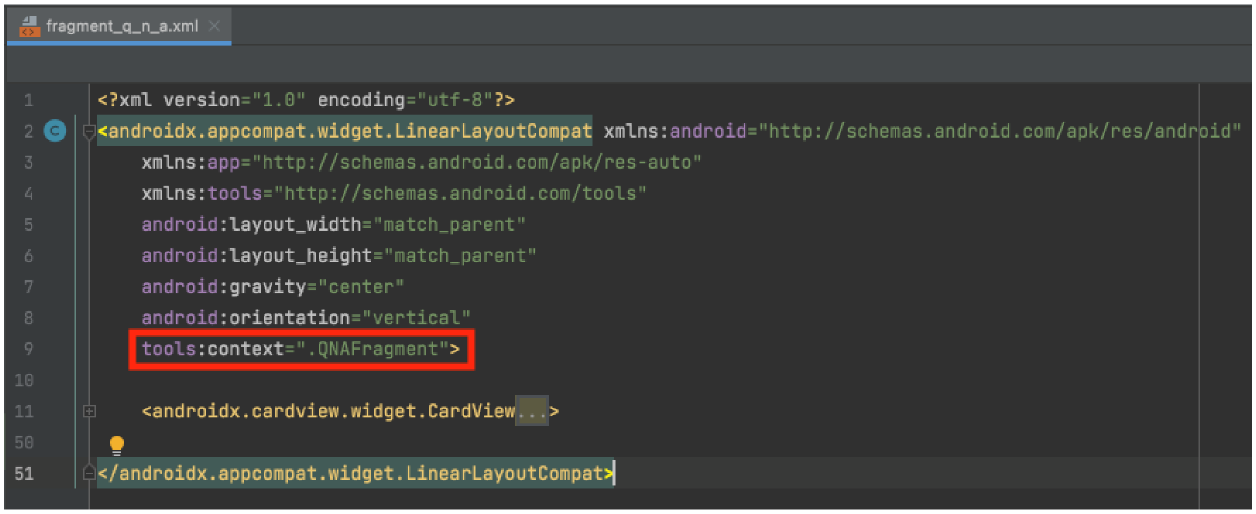
For this type of question, the player simply types in an answer to the given question. Open the layout file “fragment_q_n_a.xml” under app/res/layout folder.



Here in “fragment_q_n_a.xml”, define the layout for this type of question. The layout should include:

1. A TextView to display the question
2. An EditText for the player to provide an answer

For all questions of this type, the TextView needs to be programmatically set to display the current question in the game. Near the top of this layout file, you’ll see that this layout is associated with QNAFragment, which represents how questions of this type are displayed and the events that occur when the player enters an answer.

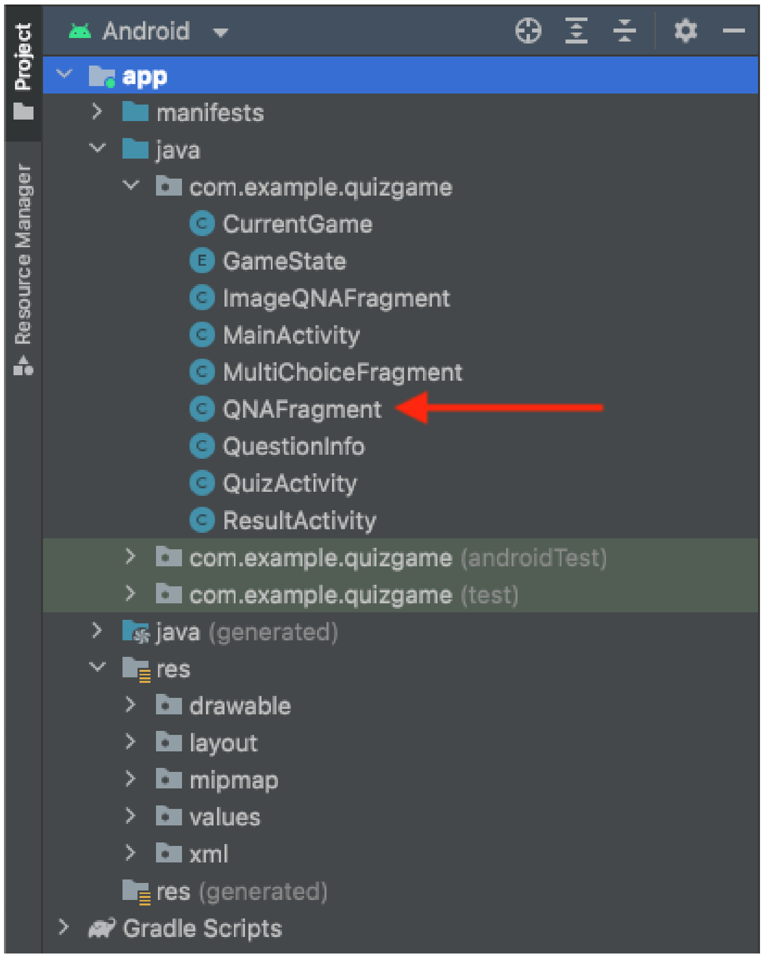

fragment_q_n_a.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.appcompat.widget.LinearLayoutCompat xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:orientation="vertical"
    tools:context=".QNAFragment">

    <androidx.cardview.widget.CardView...>

</androidx.appcompat.widget.LinearLayoutCompat>
```

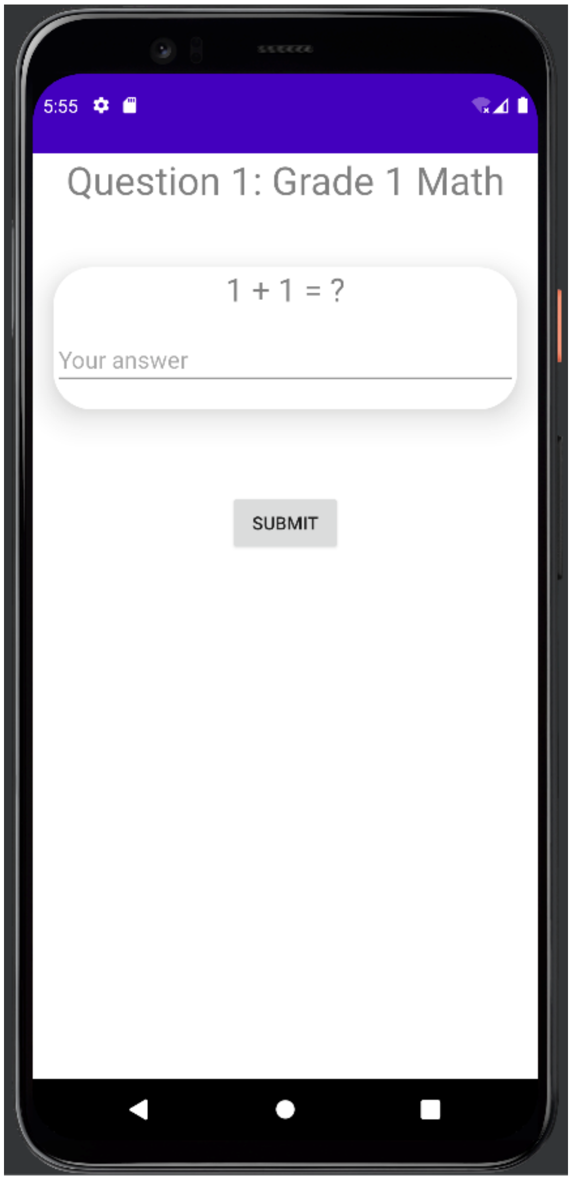
QNAFragment is defined in “QNAFragment.java”. Open the Java file “QNAFragment.java” under app/java/com/example/quizgame folder.



Inside the *linkViews()* function, have the TextView defined in the layout display the question the player is currently answering, given as the variable “currentQuestionString”.

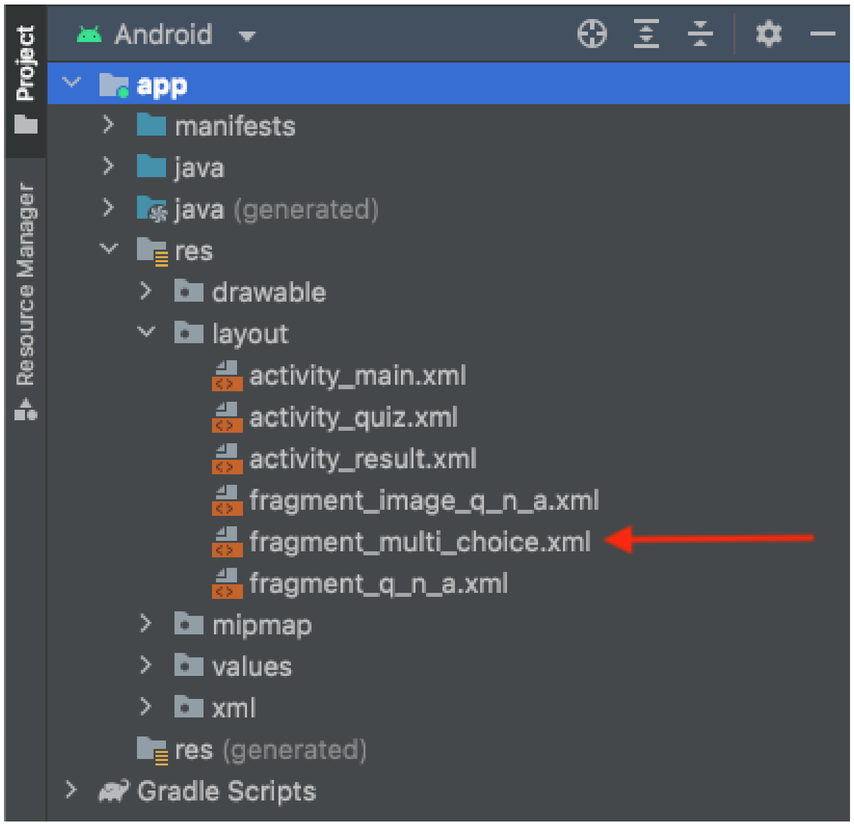
NOTE: Although the code is organized similarly to that in an Activity, common functions you may be used to calling are written slightly differently in a Fragment, ie. view.findViewById() instead of findViewById().

When running the app again, you'll see that the screen of Question 1 should be populated to look something like this:



Type #2: Multiple Choice

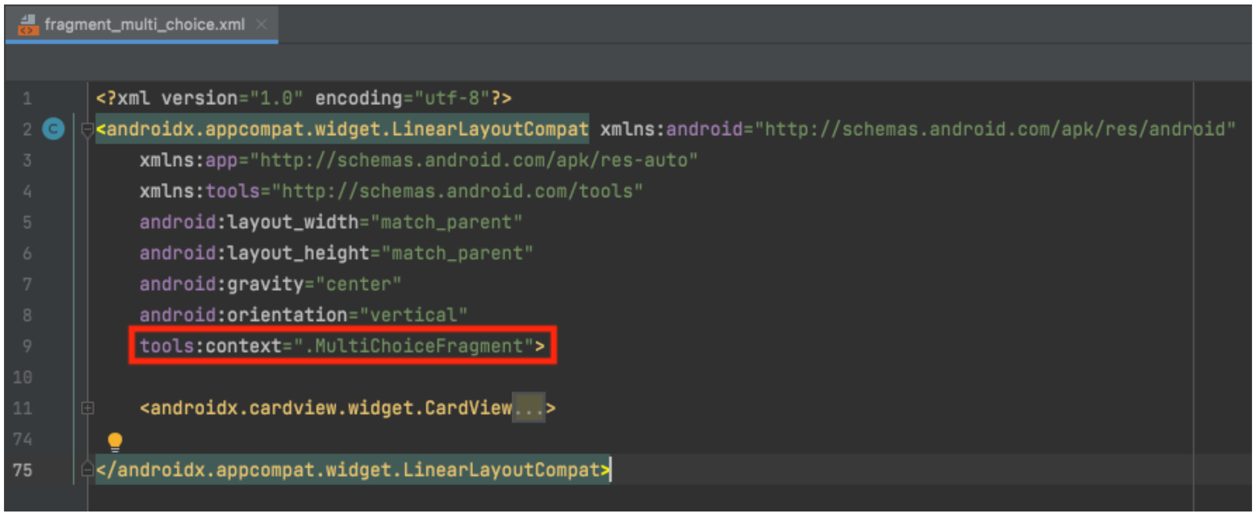
For this type of question, the player selects one of 4 choices provided to answer the given question. Open the layout file “fragment_multi_choice.xml” under app/res/layout folder.



Here in “fragment_multi_choice.xml”, define the layout for this type of question. The layout should include:

1. A TextView to display the question
2. 4 Buttons to display the 4 choices the player can choose from

For all questions of this type, the TextView needs to be programmatically set to display the current question in the game, and each Button needs to be labeled with one of the provided choices. Near the top of this layout file, you'll see that this layout is associated with MultiChoiceFragment, which represents how multiple choice questions are displayed and the events that occur when the player chooses an answer.

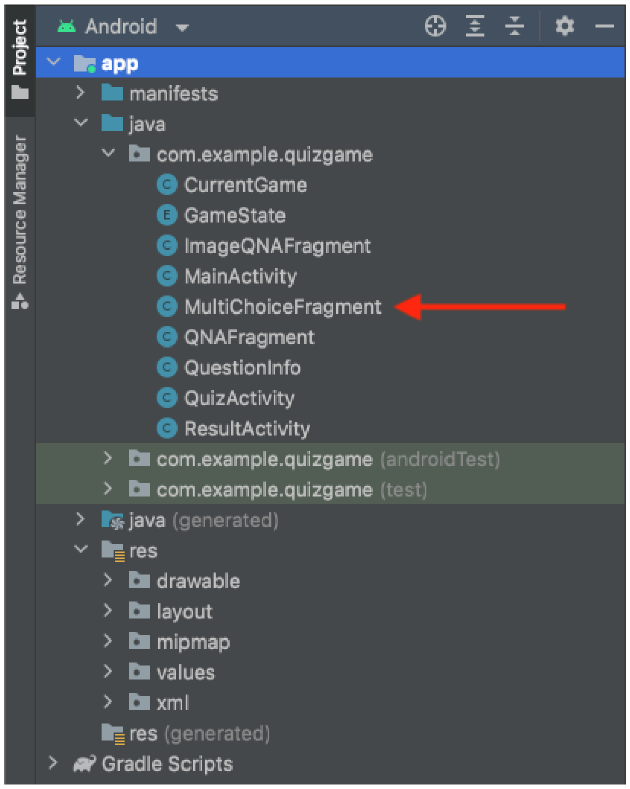


```
<?xml version="1.0" encoding="utf-8"?>
<androidx.appcompat.widget.LinearLayoutCompat xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:orientation="vertical"
    tools:context=".MultiChoiceFragment">

    <androidx.cardview.widget.CardView...>

</androidx.appcompat.widget.LinearLayoutCompat>
```

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

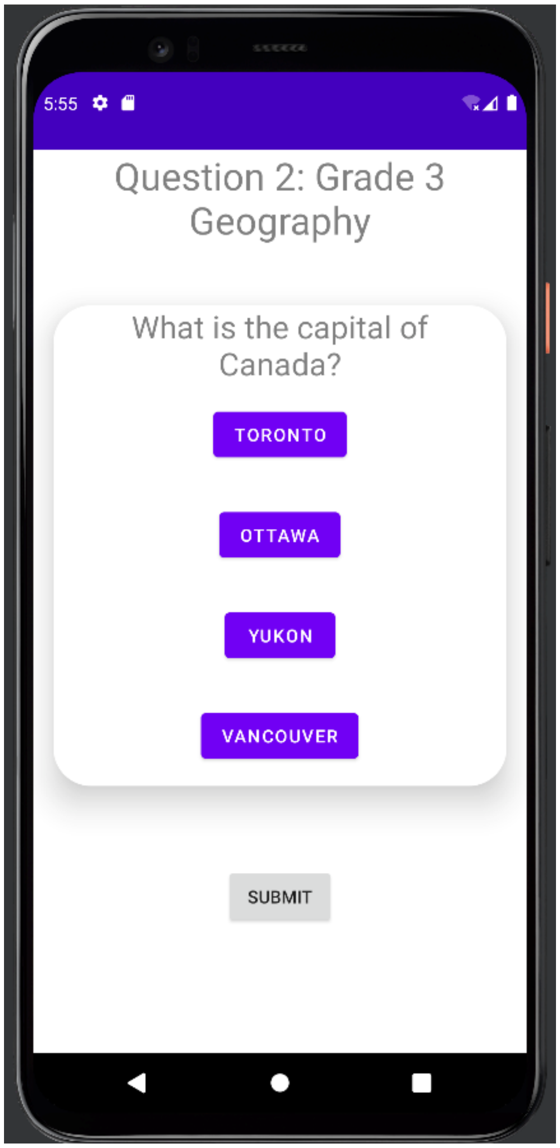


Inside the *linkViews()* function:

- **Have the TextView defined in the layout display the question the player is currently answering, given as the variable "currentQuestionString"**
- **Label the 4 Buttons defined in the layout with each of the 4 choices from which the player can select, given as the variables "choice1", "choice2", "choice3", and "choice4"**

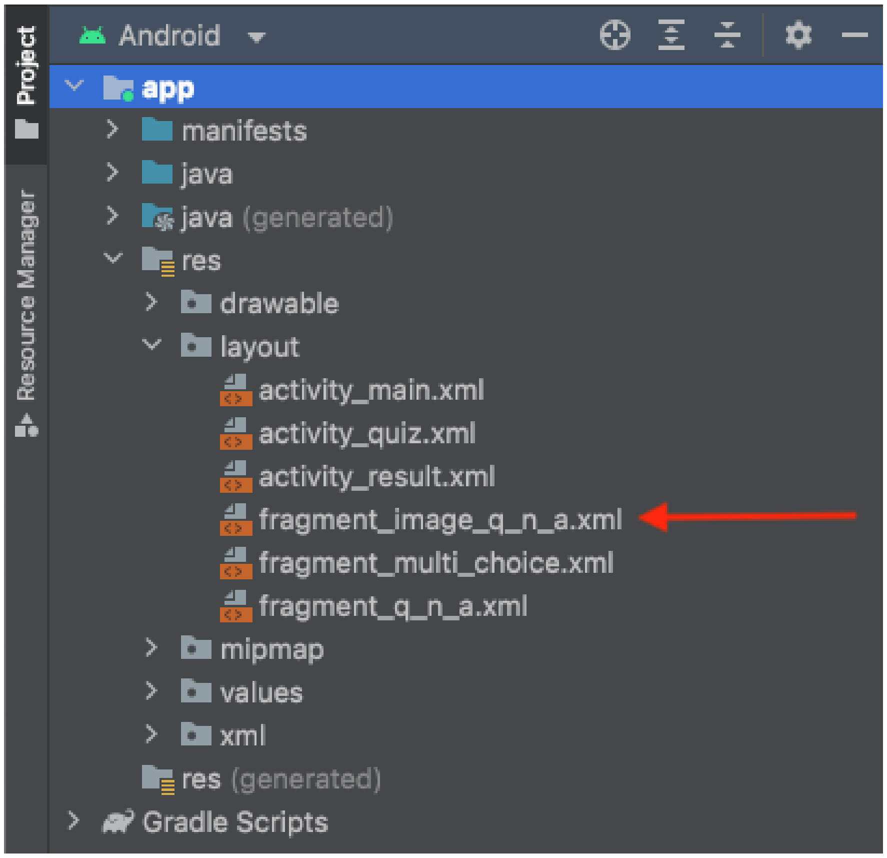
NOTE: Although the code is organized similarly to that in an Activity, common functions you may be used to calling are written slightly differently in a Fragment, ie. view.findViewById() instead of findViewById().

When running the app again, you'll see that the screen of Question 2 should be populated to look something like this:



Type #3: Question and Answer Regarding an Image

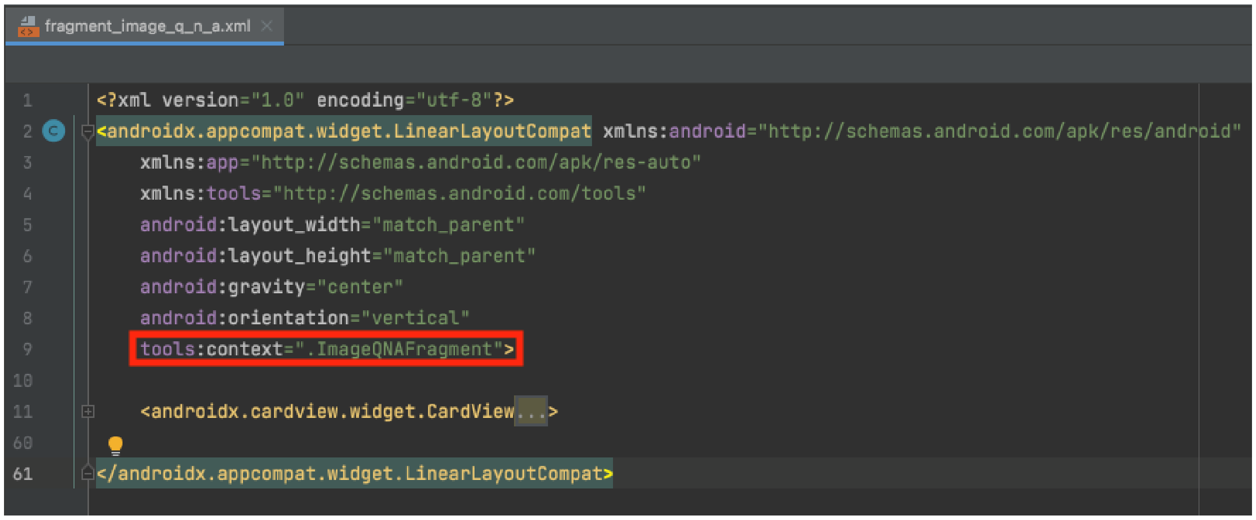
For this type of question, the player types in an answer to the given question about an image. Open the layout file “fragment_image_q_n_a.xml” under app/res/layout folder.



Here in “fragment_image_q_n_a.xml”, define the layout for this type of question. The layout should include:

1. A TextView to display the question
2. An ImageView to display the relevant image
3. An EditText for the player to provide an answer

For all questions of this type, the TextView needs to be programmatically set to display the current question in the game, and the ImageView needs to display the image the question is asking about. Near the top of this layout file, you'll see that this layout is associated with ImageQNAFragment, which represents how questions about an image are displayed and the events that occur when the player enters an answer.

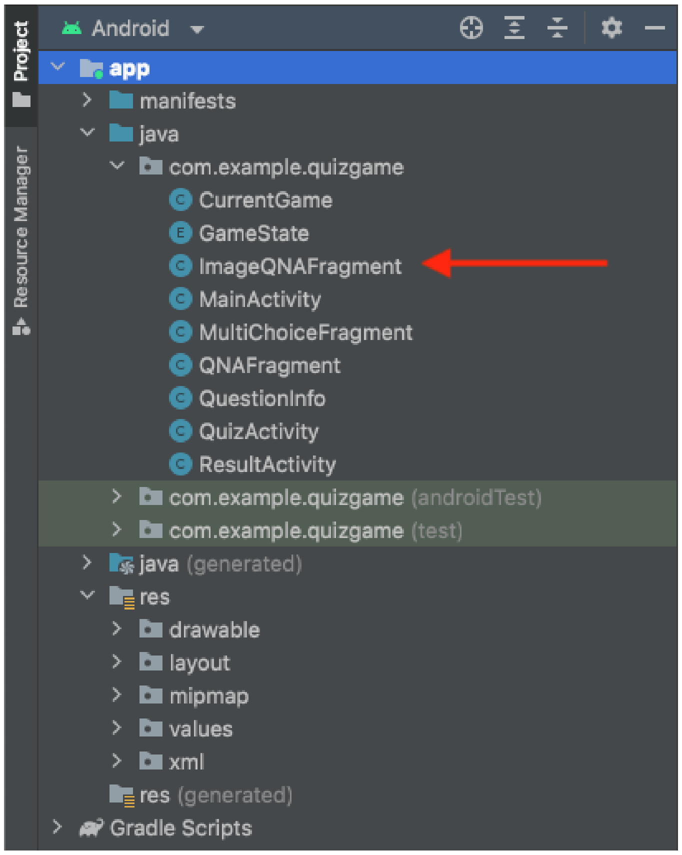


```
<?xml version="1.0" encoding="utf-8"?>
<androidx.appcompat.widget.LinearLayoutCompat xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:orientation="vertical"
    tools:context=".ImageQNAFragment">

    <androidx.cardview.widget.CardView...>

</androidx.appcompat.widget.LinearLayoutCompat>
```

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



Inside the *linkViews()* function:

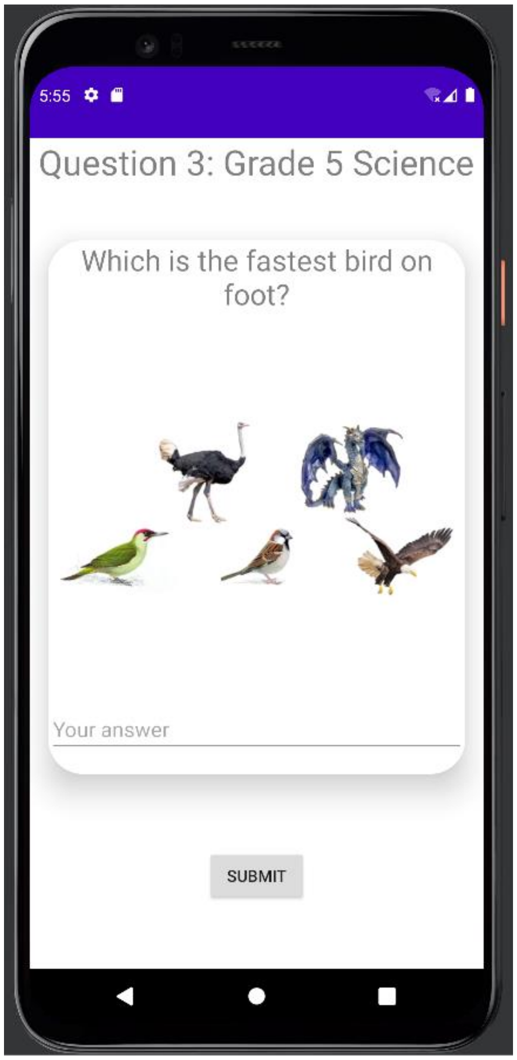
- **Have the TextView defined in the layout display the question the player is currently answering, given as the variable “currentQuestionString”**
- **Have the ImageView defined in the layout display the image the question is referring to, given as a reference to a drawable resource in the variable “currentQuestionImg”**

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()*
- *Setting an ImageView to display a specific drawable resource:*

[imageViewObject].setImageDrawable(ContextCompat.getDrawable(getActivity(), [drawableResource]))

When running the app again, you'll see that the screen of Question 3 should be populated to look something like this:


5:55
Question 3: Grade 5 Science
Which is the fastest bird on foot?
Your answer
SUBMIT

PART 4: TESTING THE APP

When navigating through the app, you should be able to see each question screen populated with a question relevant to the question category and a way for the player to enter an answer. The game will let you progress even if you enter an answer that is blank or incorrect.

Once you reach the result screen, you still won't be sure if you're smarter than a 5th grader, but at least you'll be able to navigate back to the game's title screen again when clicking on the "Play Again" Button.

In the next and final part of this project, you'll be defining how the mechanics of the game work. Once you have implemented and debugged through all those features, you can load the full game and try to prove that you're smarter than a 5th grader!