Mobile Application Lab 2

Instructors: Legand Burge, Asad Ullah Naweed
Date: 08/23/2018
Topics: Model View Controller

Be sure to click on the links in this doc to view more information about certain topics. These may or may not help you in your lab, but can always provide some extra context and information.

Initial Setup

1. Start Android Studio and create a new project.
    - Application name: Lab 2
    - Company domain: `mobileapps.techexchange.com`.
    - Be sure to create a project for a phone/tablet running at least API 26.
2. Delete the `androidTest` and `test` directories from your project.
3. Open the AVD Manager, and launch a new Android Virtual Device running Android Oreo.
4. Build and run your newly created project on the Virtual Device.

For this lab, you will be creating a Geography Quiz app. This application will present questions to the user, with two possible answer choices, only one of which is correct.

The View Layer

In this lab, you're going to learn more about how to use string resources instead of hard-coding strings in your Java code. Putting displayable strings in resource files has many advantages, including localization.

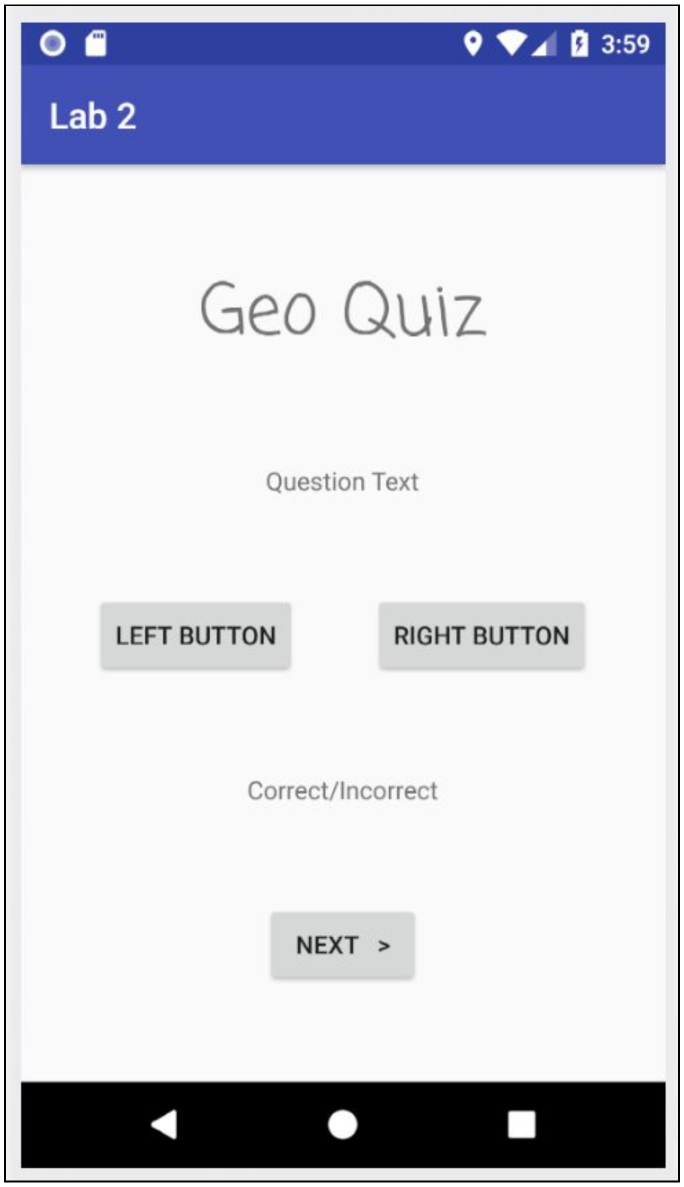
Find and open the `res/values/strings.xml` file in your project. For now, this contains only one string, which specifies the app's name.

Now, open the `res/layout/activity_main.xml` file. Replace the contents of that file with the following:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.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=".MainActivity">

    <TextView
        android:id="@+id/title_text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Geo Quiz"
        android:fontFamily="casual"
        android:textSize="40dp"
        app:layout_constraintBottom_toTopOf="@id/question_text"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/question_text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Question Text"
        app:layout_constraintBottom_toTopOf="@id/left_button"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@id/title_text" />

    <Button
        android:id="@+id/left_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Left Button"
        app:layout_constraintBottom_toTopOf="@id/correct_incorrect_text"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toLeftOf="@id/right_button"
        app:layout_constraintTop_toBottomOf="@id/question_text" />

    <Button
        android:id="@+id/right_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Right Button"
        app:layout_constraintBottom_toTopOf="@id/correct_incorrect_text"
        app:layout_constraintLeft_toRightOf="@id/left_button"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@id/question_text" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Correct/Incorrect"
        android:id="@+id/correct_incorrect_text"
        app:layout_constraintBottom_toTopOf="@id/next_button"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@id/right_button" />

    <Button
        android:id="@+id/next_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Next   >"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@id/correct_incorrect_text" />

</android.support.constraint.ConstraintLayout>
```

Now, build and run your project. You should see a UI like this:



The layout and all its components form the View layer of this application.

The Model Layer

We will now define the model for this application. The model, in this case, would be a bank of questions. First of all, you need to create a class that represents a single Question.

1. Right-click on the `com.techexchange.mobileapps.lab2` directory in your project
2. Select New > Class.
3. Create a package-private class called `Question`.

4. Add three private String fields to your class called `question`, `correctAnswer` and `wrongAnswer`.
5. Right click inside the `Question` class, and click on "Generate..." Generate a constructor that initializes all three fields.
6. Now, right-click inside the Question class again, and generate Getters for all three fields.
7. Because you are initializing all fields on construction and never changing them, you can make all three fields in this class final.

Note how the IDE generates a lot of boilerplate code for you, so you don't have to type it out yourself 😃.

Pay close attention to your Question class: Note how it has no information about the views used to display this information. This is an example of how the model layer should be completely independent of the view layer.

Now that we've defined a class that can contain all the information about a single question, we now need to create a question bank. For this lab, you will define a static question bank in the `res/values/strings.xml` file you opened earlier.

This is a strings resource file, and can contain strings and string arrays. Modify the file and add the highlighted portion shown below:

```
<resources>
    <string name="app_name">Lab 2</string>

    <string-array name="questions">
        <item>What is the capital of Australia?</item>
        <item>What is the capital of Pakistan?</item>
        <item>Paris is a city in which country?</item>
        <item>What is the capital of Kenya?</item>
        <item>Rio de Janeiro is a city in which country?</item>
    </string-array>

    <string-array name="correct_answers">
        <item>Canberra</item>
        <item>Islamabad</item>
        <item>France</item>
        <item>Nairobi</item>
        <item>Brazil</item>
    </string-array>

    <string-array name="incorrect_answers">
        <item>Sydney</item>
        <item>Karachi</item>
        <item>Switzerland</item>
        <item>Mombasa</item>
        <item>Argentina</item>
    </string-array>

</resources>
```

Take a look at what you've defined: These are not Java string arrays yet, but they are specific names, which will allow us to access them in Java code later.

Now, open your MainActivity class, and add the following method.

```
private List<Question> initQuestionList() {
    Resources res = getResources();
    String[] questions = res.getStringArray(R.array.questions);
    String[] correctAnswers = res.getStringArray(R.array.correct_answers);
    String[] wrongAnswers = res.getStringArray(R.array.incorrect_answers);

    // Make sure that all arrays have the same length.
    Preconditions.checkState(questions.length == correctAnswers.length);
    Preconditions.checkState(questions.length == wrongAnswers.length);

    List<Question> qList = new ArrayList<>();

    for (int i = 0; i < questions.length; ++i) {
        qList.add(new Question(questions[i], correctAnswers[i], wrongAnswers[i]));
    }
    return qList;
}
```

The IDE will offer a helpful suggestion to you to import `android.support.v4.util.Preconditions`. Take advantage of that. Preconditions are used here to enforce the equality of the lengths of the three arrays that are defined. If any of these preconditions fail, the app will die.

Take a look at what this is doing. It's using standard Android APIs to get access to the resources for this app, and then get the string arrays using their names. A list of questions is created and returned.

This is very similar to how you can get the Java objects for various views using the `findViewById` method.

Inside the MainActivity class, add the following fields:

```java
private TextView questionText;
private TextView correctText;
private Button leftButton;
private Button rightButton;
private Button nextButton;

private List<Question> questionList;
private int currentQuestionIndex;
```

Now, in the onCreate method, add the following lines:

```java
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    questionList = initQuestionList();
    questionText = findViewById(R.id.question_text);
    leftButton = findViewById(R.id.left_button);
    rightButton = findViewById(R.id.right_button);

    nextButton = findViewById(R.id.next_button);
    nextButton.setEnabled(false);

    correctText = findViewById(R.id.correct_incorrect_text);
    currentQuestionIndex = 0;
}
```

Before proceeding any further, make sure that you can build and compile your app. Don't expect any behavior changes at this point, but the app should still build and run.

The Controller layer

You have now set up your views, and the model as well. The main logic that will drive the quiz is going to reside in the Controller layer. In this case, it is simply your MainActivity class.

Because we're dealing with user input, the vast majority of logic that needs to be put in will be in the form of callbacks: functions that are invoked when a certain user action takes place. In this case, we will define callbacks for the buttons that are present in our view.

Before proceeding further, add the following to your app's build.gradle file:

```
android {
    ...
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
}
```

We will now use the information in the model to update the displayed text in the views. Add the following to your MainActivity file:

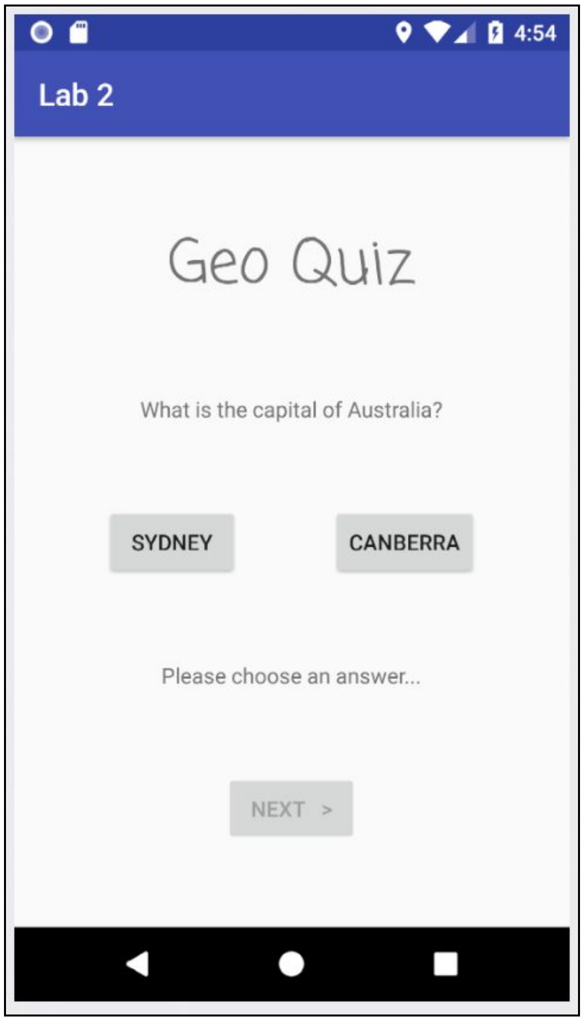
```
public class MainActivity extends AppCompatActivity {
    ...
    @Override
    protected void onResume() {
        super.onResume();
        updateView();
    }

    private void updateView() {
        Question currentQuestion = questionList.get(currentQuestionIndex);
        questionText.setText(currentQuestion.getQuestion());
        if (Math.random() < 0.5) {
            leftButton.setText(currentQuestion.getCorrectAnswer());
            rightButton.setText(currentQuestion.getWrongAnswer());
        } else {
            rightButton.setText(currentQuestion.getCorrectAnswer());
            leftButton.setText(currentQuestion.getWrongAnswer());
        }
        nextButton.setEnabled(false);
        correctText.setText(R.string.initial_correct_incorrect);
    }
}
```

Look up the documentation for `Math.random()` to see what it does.

Also add the following string to your `strings.xml` file:

```
<resources>
    ...
    <string name="initial_correct_incorrect">Please choose an answer...</string>
</resources>
```

Now, build and run your app. You should see a UI like this:



Of course, tapping on any of the buttons does nothing at the moment. Mainly because we haven't defined what should happen.

We will now define callbacks for all the buttons in this view. Callbacks are simply functions that get executed whenever a view is interacted with. However, before proceeding, make sure that you're familiar with how [Java lambdas and method references]() work, because we'll be using those.

Now, begin by adding the following lines to your MainActivity class:

```java
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        questionList = initQuestionList();
        questionText = findViewById(R.id.question_text);
        leftButton = findViewById(R.id.left_button);
        leftButton.setOnClickListener(this::onAnswerButtonPressed);

        rightButton = findViewById(R.id.right_button);
        rightButton.setOnClickListener(this::onAnswerButtonPressed);

        nextButton = findViewById(R.id.next_button);
        nextButton.setEnabled(false);

        correctText = findViewById(R.id.correct_incorrect_text);
        currentQuestionIndex = 0;
    }

    private void onAnswerButtonPressed(View v) {
        Button selectedButton = (Button) v;
        Question ques = questionList.get(currentQuestionIndex);
        if (ques.getCorrectAnswer().contentEquals(selectedButton.getText())) {
            correctText.setText("Correct!");
        } else {
            correctText.setText("Wrong!");
        }
        nextButton.setEnabled(true);
    }
}
```

Here's what just happened: You defined a new method called `onAnswerButtonPressed` and then set it as the callback to be invoked when the left or right answer buttons are clicked. The argument of the `onAnswerButtonPressed` method is the view that has been clicked, which in this case will always be a `Button`, so it's safe to cast.

Using method references allows us to do this kind of stuff in a very straightforward and readable manner, as opposed to the old Java way of using anonymous classes.

Before proceeding, build and run your app and see what happens when you tap any of the answer buttons.

Now, as an exercise, think about what kind of a callback you need to add for the "next" button. *Hint: At best, it's only two lines of code* 😁