# MultiQuiz App (Part 1 of 3)

In this series of project parts, we'll develop an app similar to the GeoQuiz app covered by BNRG Chapters 2 through 7. Each project part will iteratively build upon the earlier parts, introducing new features and making a few changes along the way.

This project part closely corresponds to Chapters 2 and 3, without the challenges, so you're highly encouraged to follow along and complete those textbook chapters before starting.

## Getting Started

A project template with starter code is provided via Git, a popular version control system. Android Studio has an integrated client to make this process straightforward. Just start Project 1A by selecting "Get from VCS" from the Android Studio welcome screen, or use File | New | Project from Version Control from the menu of any existing project. In either case, specify the following remote URL, and specify any convenient directory on your development system:

- `https://www.prof-oliva.com/cs5254/2024.Spring/git/P1-MultiQuiz.git`

This will create a new project by cloning a local copy of the remote repository. The project will contain the all starter code needed for this assignment in a single `assignment` branch. If you're familiar with Git, please feel free to use the built-in Git features to make commits to this branch, or create another branch if you prefer. If you're not familiar with Git, or prefer not to use it during the development process, you can safely ignore it while you complete this project.

Beyond the basic MainActivity/activity_main and other files normally created in a new Android Studio project, the repository includes the following changes specific to this project:

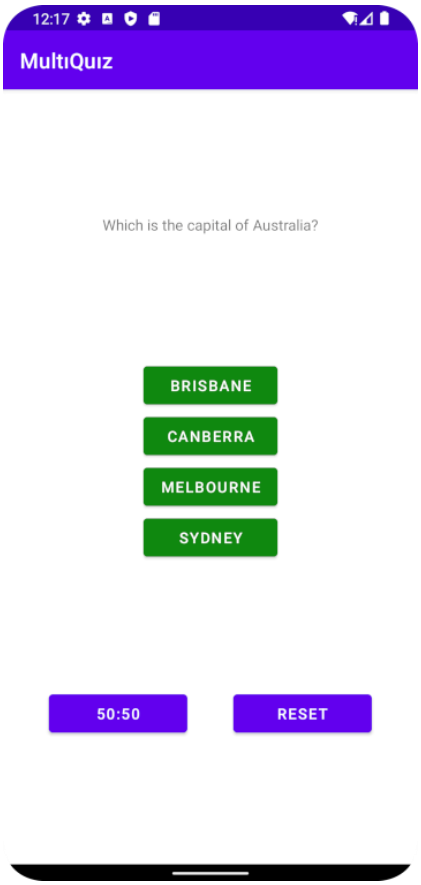
- **build.gradle.kts** (Project: MultiQuiz)
    - The configuration file has been updated to include the most recent plugin versions
- **build.gradle.kts** (Module :app)
    - The configuration file has been updated according to BNRG Listing 2.6 to support view binding (with dependencies updated to the most recent compatible versions)
- **Answer.kt**
    - A new data class, as part of the Model, where each instance holds a single answer option, whether it is a correct answer option, and its current selected/enabled state
- **ButtonColorUtil.kt**

- A utility class providing an updateColor() "extension function" to the Button class
    - This function sets the color scheme of any button, based on its current selected/enabled state. You'll call this on each answer button within the interface to set its color

Please use the above files as-is, without making any changes to them.

## Layout

Use nested LinearLayouts, along with TextView, Button, and Space views to create the layout. The interface should be arranged like this when the app is launched:




[Enlarge Image]

Space views, each with an `android:layout_weight` of 1, can be placed between and around the other views (including layouts) to spread those other views across the screen with equal gaps/margins. Specify `0dp` as the `layout_width` or `layout_height` to let the weight dictate the size. Other view components shouldn't have a weight specified.

For this project, View Binding (from BNRG Chapter 2) must be used rather than findViewById (from BNRG Chapter 1) to access the layout components from the activity. As noted above, the appropriate build.gradle file has already been updated according to Listing 2.6.

# Behavior

In this part of the app, there will be a single multiple-choice question with four possible answers. The answers are mutually exclusive, so at most one may be selected at any time. Each possible answer is displayed in a separate answer button. The answer buttons behave as follows:

- All answer buttons begin as deselected and enabled
- When any answer button is clicked:
    - The clicked button's selected state is reversed (from true to false, or false to true)
    - All other buttons become deselected

Below the answer buttons, there will two additional buttons. These should just have the default theme, style, and colors. On the left is a "lifelineLinks to an external site." 50:50 button, and on the right is a Reset button. These buttons are both initially enabled, with the following behaviors:

- When the 50:50 button is clicked:
    - The first two incorrect answer buttons become disabled and deselected
    - The 50:50 button itself becomes disabled
- When the Reset button is clicked:
    - All of the answer buttons become deselected and enabled
    - The 50:50 button becomes enabled

A short video demonstrating the expected behavior is available in Piazza.

# Strings

The following strings must be defined only within the strings.xml file:

- `Which is the capital of Australia?`
- `Brisbane`
- `Canberra`
- `Melbourne`
- `Sydney`
- `50:50`
- `Reset`

## Functional Programming

While developing the app, practice using Kotlin's functional programming features, rather than an imperative programming style. For example, use the `forEach` function -- rather than any for loops -- along with Kotlin's `zip`, `filter`, and `take` functions to complete the assignment, along with either `any` or `all`.

## Grading

Please see the rubric at the bottom of this page.

## Deliverable

Upload the following two files:

- MainActivity.kt
    - Include your name and PID in comment lines, just below the class declaration
- Screenshot of the emulator
    - Similar to the image above (note that the remainder of the IDE doesn't need to be visible)
    - The emulator must show two disabled (incorrect) answers, the disabled 50:50 button, and one selected answer