Learning Objectives:

Project 2

- Gain experience developing apps with multiple activities
- Gain experience with file I/O and JSON parsing
- Practice object oriented design and programming
- Use the Git/GitHub support integrated into Android studio to develop your project under version control

Project #2 demos are on Monday, 29-Oct-2018 and Tuesday, 30-Oct-2018. Deliverables are due on GitHub by 10:00 PM on Wednesday, 31-Oct-2018

You will work alone on this project

Project: ANDROID MULTIPLE CHOICE QUIZ APP

For the second programming project in ECE 558 you will create a multiple activity application using the Android SDK. This project will be to build on what you learned by completing and enhancing the GeoQuiz application in BNRG Ch 1 - 5 (see, I told you that it would be worth your effort to complete it). The app for this project is more complex than GeoQuiz but that's to be expected and significantly more difficult than Project #1. Android apps (like most GUI-based apps) trade programmer time for ease of use for the user.

The deliverables for this project will be a demo for the T/A or instructor, a design report for the project, your Android studio project and a signed executable (.apk file) of your final app.

We will be using GitHub classroom for this assignment. You will do your development in a local Git repository on your PC that will be linked to a private repository on GitHub that can also be accessed by the instructor and T/A for the course. Our plan/goal is to provide feedback directly to your repository but we are still learning about how to effectively use GitHub classroom so we'll see how it goes. Fortunately, Android Studio and IntelliJ have excellent integration with version control systems such as Git. You may find the following links useful if you are not comfortable with Git and GitHub:

Tutorials on using the Android Studio integration w/ Git and GitHub:

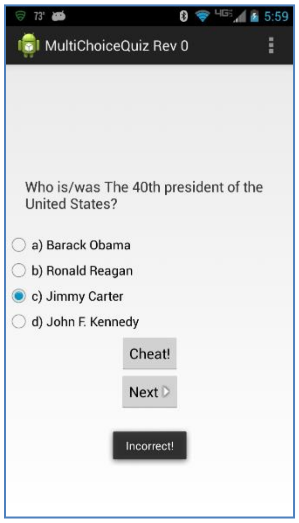
- (11 min) https://www.youtube.com/watch?v=3p_fgJEbsp8
- Will W. Part 1 - Setup (5 min): https://www.youtube.com/watch?v=AJune0EMP94
- Will W. Part 2 – Import and Push (15 min): https://www.youtube.com/watch?v=tjYQPsBba7s
- Will W. Part 2.1 – Pull and Clone (8 min): https://www.youtube.com/watch?v=u84uKjY5S0g
- Will W. Part 3 – Create, delete, and switch branches (7 min): https://www.youtube.com/watch?v=OJoW0eaPpgM

Will Willis also posted a video on merging using Android Studio and GitHub that you may find useful when you work on team projects.

Tutorials on using git and GitHub:

- Edureka (1hr, 45 min): https://www.youtube.com/watch?v=xuB1Id2Wxak
- TraversyMedia (32 min): https://www.youtube.com/watch?v=SWYqp7iY_Tc&t=96s

Problem Statement



Who is/was The 40th president of the United States?

a) Barack Obama
b) Ronald Reagan
[✓ c) Jimmy Carter]
d) John F. Kennedy

All of us have taken multiple choice quizzes, but not all of us have taken them on a phone or tablet. That's about to change for you. Your assignment for the project is to create a multiple choice quiz app that runs on your Android device. Your app must provide a "cheat" function that will open a new screen and give the user an opportunity to reveal the answer. The user can return to the question screen by pressing the ↩ (*Back*) button on the device. Of course answering a question that you already know the answer to is not very challenging so your app needs to keep track of whether the user has "cheated" (e.g. revealed the answer) and chastise him/her appropriately. The user should be able to rotate his/her device and continue from where he/she left off.

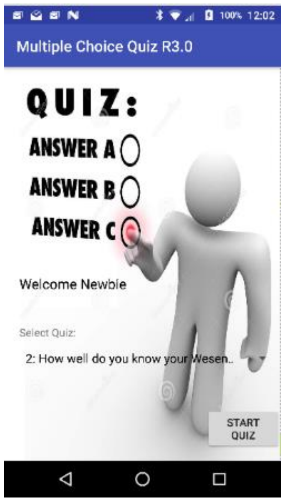
The user interface for this application does not have to be identical to the one shown here, but it should present a question, provide a way for the user to make a unique (i.e. only one) choice, tell the user whether his/her answer is correct and move to the next question (*Next* button in this example) while he/she is taking the quiz. The user should also have a way to request the answer (*Cheat* button in this example).

The app should implement these features at a minimum:

- The user should be able to select a quiz using an Android *Spinner* (a drop down list). The app must be able to support more than one quiz.

- A quiz should be defined in a `.json` file. We have specified the format of the `.json` quiz file and provided a sample quiz. You will create a second quiz in the same format, include it in your app, and write the code for the method(s) to parse the `.json` files. There is a pretty good JSON tutorial included with the project release and there are plenty of online examples and more than one way to parse JSON in Android.

- Your app should retrieve quiz files from *`res/assets`*. This is a change from previous versions of this assignment where the quizzes were downloaded and retrieved from external storage. Your app must be able to retrieve and use more than one quiz file in the directory.



- Your app should start with a splash screen (including a background image) that, at a minimum, includes the quiz select Spinner and a button to start the quiz. My splash screen looks like this, but express your creativity.

- The icon for your application should be something different than the default Android robot. Once you find an appropriate image you can use the *Image Asset* tool in Android Studio to create the icon in the `/res/mipmap` folder. Be sure to edit `AndroidManifest.xml` to change the icon for the app if you need to.

- Your app should save some piece of information using the *SharedPreferences* capability in Android. That piece of information should be retrieved the next time the app is started (*SharedPreferences* provides persistent storage between invocations of the app). For example, you may want to prompt the user for his/her name the first time they start your quiz app and then retrieve and display his/her name every time they start the app (`Welcome Roy` instead of `Welcome Newbie` as shown in my slash screen.

- You app should display a `Results` screen when the user has finished the test. The screen should show the final grade (number of questions, number answered correctly, number cheated on). You do not need to persist the score since the user may take a different quiz the next time. However, another use of *SharedPreferences* could be to save a high score and display it in the `Results` screen; This would also meet the project requirement to “save some piece of information to *SharedPreferences*…”

- You may make use of dialogs, `ListView`, `RecyclerView`, `ViewPagers`, `Fragments`, etc. if they don’t affect the usability of your application. We will be discussing several of these Android constructs before the end of the term, but not before the project is due. Using these more advanced constructs is optional; a multiple activity app is fine.

Suggested Procedure

1. Review your implementation of GeoQuiz and the relevant chapters in BNRG: 3e. After you feel that you have refreshed your understanding of `GeoQuiz` you are ready to design your Android app. Most of the concepts and techniques you learned while completing `GeoQuiz` are applicable to this new app. Some tradeoffs to consider while designing your app:
   - What control should be used to allow the user to make a unique choice from a list of choices? A Radio Button group is a logical choice because that's what radio buttons do, but there are alternatives. Use developer.android.com to learn about the various input widgets available for Android.
   - What classes and methods are you going to use to encapsulate a quiz question and a quiz? The key benefits (Roy's opinion) of Object-Oriented-Programming are encapsulation and abstraction. Well-designed and tested classes can be used and reused with little or no change.
   - What state do you need to preserve through configuration changes such as rotating the device? `GeoQuiz` has the capability to preserve state by overriding `onSaveInstanceState()` and restoring the state in `onCreate()`. The same method will work for your app, but you may need to save different state information.
   - What information do you need to pass between the activities and how do you do it? You are building a multiple screen, multiple activity app. The approach used in `GeoQuiz` may apply but you may need to pass different (and/or additional) information between the activities.

2. Implement a prototype of your design. This could be a single activity app that handles a single "hardwired" question. There are numerous code examples and tutorials on how to use input controls like radio buttons on the internet. Entering something like "Android radio button example" into a search bar yields plenty of good examples. Create a new project in Android Studio, enable version control, link the project to Git/GitHub, create a layout, create listeners for whatever buttons you have in your design and override `onCreate()`. Give thought to the structure of your program and create "helper" methods for specific functions. For example, having `checkAnswer()` and `updateQuestion()` methods will encapsulate specific functionality and make your code maintainable and easier to follow.

3. Add the functionality to retrieve a quiz file from the *`res/assets`* resource folder and parse the JSON to build the quiz. Make your application behave properly through configuration changes like rotating the device and pressing the BACK button.

4. Test and debug this functionality thoroughly to make sure unexpected things don't happen. For example, I had to find and fix several null reference exceptions and an annoying "feature" which resulted in the answer from the previous question being displayed along with the new question whenever the user pressed the *Next* button. The functionality that you create in the previous step is the core of the application. If you get this far with a solid application you are more than halfway to done.

5. Add the functionality to implement the "cheat" activity. If you have dutifully implemented, and you understand what you did in `GeoQuiz` you will find that doing this in your app is almost identical to the steps (and the code) outlined for `GeoQuiz` in BNRG. The layout and `CheatActivity` class can be used almost verbatim with the caveat that you will need to transfer information about what choice the user made (as opposed to true/false) and you need some way of displaying the choice (other than "True" or "False").

6. Generate at least one additional quiz file and figure out how to add it (them) to `res/assets`, include it (them) in the Spinner, select a quiz, and take the selected quiz in your app. A quiz should include at least 8 multiple choice questions. Each question should have at least 4 choices. There are online JSON display tools (ex: The Awesome JSON viewer plugin for Chrome) to verify that the quiz file is syntactically correct. A quiz template has been included in the release.

7. Test and debug your new application. Check that your app works properly whether the user cheats or not. Check that the message shown to the user when he/she answers the question is different after he/she has revealed the answer. Make sure configuration changes like rotation work.

8. Test the usability of your app. Have your friends take the quiz. Note their reactions and solicit feedback about how easy (or not) your app is to use. Encourage them to break the app by doing the unexpected. This step may not seem overly important in a programming class, but usability is a huge factor as to whether your app gets a few downloads or millions (OK, I'm probably exaggerating) and the customer ratings assigned to your app. It is important to note that we will grade the usability of your app and that it is worth 15% of the project grade.

9. Document your design. The `JavaDoc` support built into Android Studio provides a mechanism for documenting your class APIs without much additional work. Although Android Studio doesn't support UML class description generation, there are applications that do. You don't need to document the standard callbacks, but you should document any methods you write, classes you develop, etc.

10. Write a design report for your app. Describe and discuss your design, most notably the mechanisms you used to manipulate the question database. You may use UML, block diagrams, or whatever you feel best gets your design approach across. Be sure to discuss the tradeoffs you made in the design. Your design report should discuss the key methods that you implemented. Use code snippets and explanatory text to describe your key methods. Describe what information you preserved across configuration changes and the information you passed between the quiz activity and the cheat activity. Elaborate on any problems you found and how you debugged and fixed them.

    Try to keep your design report to under 8 pages in length. While neatness counts for your code, the page count for the design report does not. Submitting a report that is more than 8 pages does not guarantee a higher grade. Neither does submitting a report that is less than 3 pages leads to a lower grade. Think of the information you would want to have if you put aside your app for 2 years and then had to pick it up again to enhance it or fix a bug. The purpose of the design report is to help us understand your code and your implementation.

Extra Credit Opportunities (Optional, up to 5 points)

There are opportunities for creativity in this project. For example, you can choose other Android widgets to present the choices to the user instead of radio buttons and/or you can eliminate the *Next* button by having the user touch the question text to move to the next question. You could also use gestures to cycle through the questions. Explore http://developer.android.com/guide/topics/ui/index.html to get an idea of what is available. You can innovate in both the splash screen and the result screen. For example, you could give the user the choice of exiting the app or starting again (maybe selecting a different quiz) when they have finished the quiz. Although we aren't likely to discuss media readers this term you may also find it fun to experiment with playing a song or using sound clips. You could also get extra credit for creating your own JSON format and for storing your JSON files on External Storage rather than integrating the files directly into your app.

Keep in mind, though, that “extra credit” is just that…extra. You must submit a working app for the demo, document your project well, write well-structured and well-commented code and submit your project on time for the project to be considered for extra credit.

Demo

You will demonstrate your application on your own Android device but the instructor or T/A may “drive” the app to check that you have met the specification and have correctly handled the error cases. You may NOT do your demo on the Android emulator.

Deliverables

Push your deliverables to your private GitHub repository for the assignment. The repository should include:

- Your design report (.pdf preferred).
- JavaDoc output for your app, particularly for the key methods you created.
- A final version of your Android project. We will use this to grade your effort. “Neatness counts” for this project - we will grade the quality of your code. You code should be well structured, indented consistently (Android Studio helps with that) and you should include comments describing what long sections of your code do. Comments should be descriptive rather than explain the obvious (ex: //set a to b when the actual code says a = b; does not provide any value-added).
- A signed `.apk` (Android Application Package) file that can be downloaded and installed on an Android device. You can generate an `.apk` file from Android Studio by selecting `Build/Generate Signed APK`. It should be noted that using Android Studio to generate an `.apk` file will require that you “sign” your app with a keystore. The `Generate Signed APK` wizard will help you through the process.

Grading

You will be graded on the following:

• Demo	50 pts
• Usability of your app	15 pts
• Design report	20 pts
• Code quality	15 pts
	100 pts
• **Optional extra credit**	**up to 5 points**

References

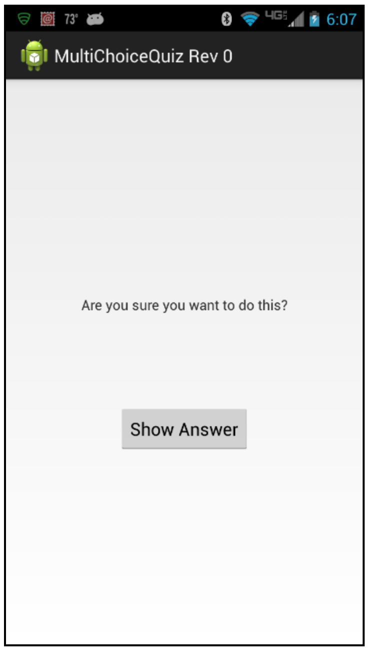
- Android Programming 3e: The Big Nerd Ranch Guide
- http://developer.android.com/index.html
- http://www.oracle.com/technetwork/java/javase/documentation/index-jsp-135444.html

Acknowledgments

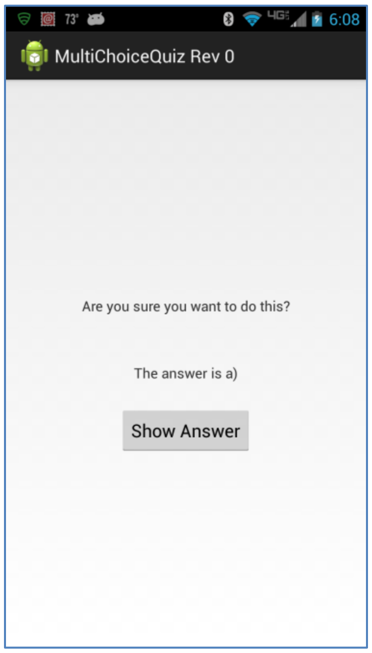
- Bill Phillips, Brian Hardy, Kristin Marsicano and the other authors, developers and instructors at Big Nerd Ranch, Inc. for creating excellent “how to” projects that show how to implement many of the features that have become standard on today’s mobile devices.

- The Android developer Community and Google for providing tutorials, explanations and Android program examples. There is a wealth of information "out there" and it's only a few searches away.
- JSON.org for their support and documentation on JSON and parsing JSON.

Sample User Interface Images

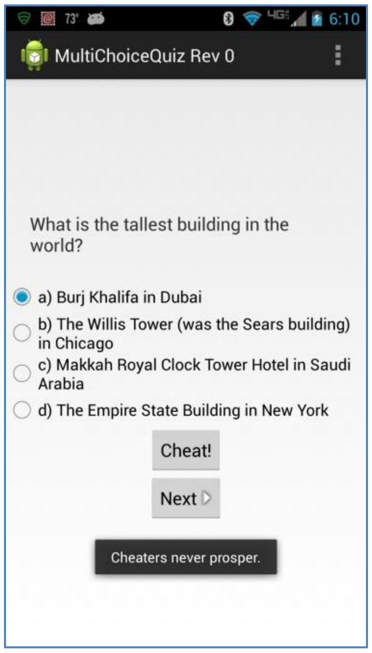


Are you sure you want to do this?



Are you sure you want to do this?

The answer is a)



What is the tallest building in the world?

- [✓ a) Burj Khalifa in Dubai]
- b) The Willis Tower (was the Sears building) in Chicago
- c) Makkah Royal Clock Tower Hotel in Saudi Arabia
- d) The Empire State Building in New York

Cheaters never prosper.

Images created in Android Studio