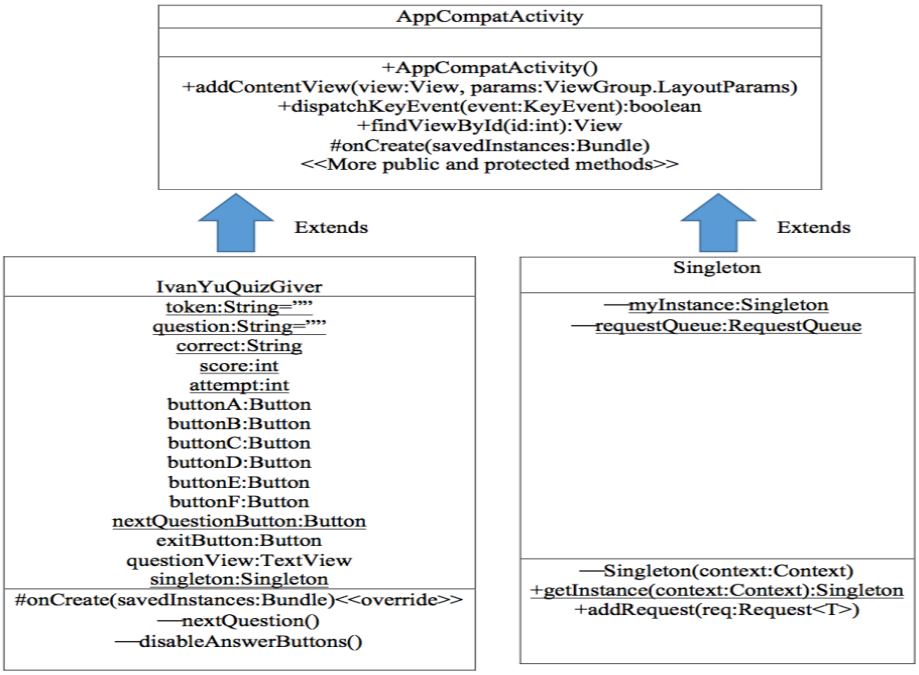
Ivan Yu
CSC 413, Section 1
Assignment 2 Written Report
6/23/16

Assignment 2 Written Report

i. IvanYuQuizGiverAssignment2 Brief Overview

This program begins by displaying the initial score, a button for generating the next question, and a button for exiting the program on the screen. The user can start the program by clicking on the question generating button. Once the question and answers are displayed on the screen, the button that generated the question will be disabled until one of the answer choices are clicked by the user; therefore, the user cannot generate another question without having answered the current question on the screen. Also, upon generating the question, the token retrieved from the server will be displayed as well. Once the user clicks on any of the answer choices, the user will be immediately notified whether their answer is correct or wrong, and the score is incremented appropriately. The score will be represented with the text "Score: 0/0" initially, and the number to the left of "/" is incremented if the user selected the correct answer, while the number to the right is incremented every time a question is answered. The percentage of correct answers the user has given will also be displayed. Also, the same question will not be asked immediately after the same question is answered. Once an answer is clicked by the user, the user will not be able to click on any of the answer choices for that particular question, and to continue the program, the user must click on the question generating button. The questions, tokens and answers are generated from a remote server located in the website sfsuswe.com/413, and the process from retrieving these values will be explained in the "Learning and Coding Resources Consulted" section of this write-up.

ii. UML Class Diagram



iii. Learning and Coding Resources Consulted

API Communication

The creation of this program required the ability to communicate with a remote Application Programming Interface (API) from the sfsuswe.com server, using the Volley library. Firstly, it is important to note that in order to communicate with a remote server, the correct permission must be specified within the manifest file, and the Volley library must be imported as a module in the project.[i] Also, within the application's gradle build, the following statement must be included in the dependencies to allow access to Volley: "compile 'com.mcxiaoke.volley:library:1.0.19'".The Volley library provides classes and methods that can be utilized to communicate with an API properly, such as the JsonObjectRequest and the RequestQueue. Volley is utilized in the first task of the program: retrieving a token string from the website http://sfsuswe.com/413/get_token/, which returns a json object upon a successful request. In order to request a json object from the server, a JsonObjectRequest must first be initialized, and then it must be added into a RequestQueue. It is important to note that only one instance of a RequestQueue object is instantiated throughout this program, and this instance is made through a method in the Singleton class (which follows the Singleton software design pattern). In order to properly initialize a JsonObjectRequest object, its constructor must have the the proper arguments: the request get method, the correct string representation of the url, a Responser.Listener object with an overridden onResponse method taking in a json object argument and a Response.ErrorListener object with an overridden onErrorResponse method which is called upon failure to make the request to the server. Since the constructor uses the request get method as an argument, the username and password can be properly sent to the server by using the following url as an argument: the website name "http://sfsuswe.com/413/get_token"?username=USERNAME&password=PASSWORD", where "USERNAME" and "PASSWORD" are respectively the student's assigned username and password.[ii] The parameter of the Response.Listener object argument's overridden onResponse method is the json object returned by the server; the token can be retrieved from this argument by using the getString method of the JSONObject class, and passing the name of the proper key (which is the string "token") as its argument.[iii] By preference, I decided to use a static string data field called "token", setting this variable equal to the json object's token string; the "token" variable will then be used for the next JsonObjectRequest object.

To finally retrieve a question, a similar JsonObjectRequest must be made, and since this JsonObjectRequest's constructor will also use the request get method as an argument, the url argument can be written as "http://sfsuswe.com/413/get_question?token=" followed by the actual token. The remaining arguments are similar to that of the JsonObjectRequest's constructor for retrieving the token. For the "Response.Listener" object argument's overridden onResponse method, the parameter is the json object returned by the server, and from this json object, the question, answer choices and the correct answer can be extracted. Again, this JsonObjectRequest must be added into a RequestQueue in order to complete the attempt to request the json object. If the request is successful, the onResponse method within the "Response.Listener" argument of the JsonObjectRequest's constructor will be called. Within this method, I created views such as text views for the question and the token, and buttons for the answer choices, all of which are programmatically placed on the screen, on top of the previous questions that were answered. To

successfully implement this onResponse method as such, the ability to utilize views and layouts is required.

Views and Layouts

In order to properly place the questions and answers acquired from the server onto the screen, the knowledge of views and layouts is necessary. For this program, I created a full screen scrollable view, by utilizing my xml file to place views and layouts onto the screen, and Java codes to programmatically add more views onto a layout already placed in the screen. In my activity xml file, I placed a linear layout within a scroll view; this linear layout contains a table layout on top of another inner linear layout. The table layout has a single row in which the leftmost position is occupied by the score of the user, followed by the question generating button to the right, and the exit button placed in the rightmost position. The inner linear layout will contain the tokens, questions and answers generated as the user interacts with the program; these contents will be placed into this layout programmatically. Initially, the screen contains only the default display of the score, the question generating button and the exit button; the user will have to press the question button to display the first and preceding sets of tokens, questions and answers.

The state of the contents of the table layout, with the exception of the exit button, changes as the user interacts with the program, as the score changes properly upon a user's right or wrong answer and the button for generating another question is disabled when clicked and until an answer choice is clicked by the user. Such changes are allowed by using the findViewById method to create references to both the question button and the score text, and then manipulating its attributes by using the references.[iv] The inner linear layout's contents will be organized as such: upon generating a new set of token, question and answers, the token is placed above the question, and the question is placed above the answers. In numerical order, the answer buttons for "A", "B", "C", "D", "E" and "F" are placed from top to bottom. This new set of token, question, and answers will be placed above the previous set which contains the last answered question. The user will be able to see the previously answered questions by scrolling down the screen.

The codes that allow the screen to change as such is executed within the onResponse method of the Response.Listener object argument of the JsonObjectRequest constructor for retrieving a new question. I designed this onResponse method such that it creates a TextView and several buttons, which respectively has the text for the question and the answer choices. The functionality of the buttons for the answer choices – such as incrementing the score and disabling all answer buttons when clicked – is also implemented within this method.

IvanYuQuizGiver Class

The IvanYuQuizGiver class contains the methods that allow communication with the remote server and the programmatic changes applied to the screen. As described by the UML class diagram, this class has three methods: onCreate, disableAnswerButtons and nextQuestion. The onCreate method first initializes the activity screen for the user, providing only the score texts (which are initially "0/0" and an empty field for the percentage of correct answers) and the buttons for generating new questions and exiting the program. In this method, the button for generating questions is referenced by a variable using the findViewById method, and its onClick method simply calls the nextQuestion method of this class and then disables the button's own functionality. It is important to note that upon clicking this button, the user will be able to click this button again

only after the user has clicked on one of the answer buttons; this design prevents the user to generate more than 1 set of questions and answers without answering the initial question. The exit button was implemented in the onCreate method to simply terminate the program. The nextQuestion method contains all the requests creation, question generation and the programmatic changes to the screen that were previously described in this write-up in the "API communication" section. The disableAnswerButtons method simply disables all of the buttons for the answer choices for a question. This method is called when one of the buttons for the answer choices is clicked, preventing the user to answer a previously answered question.

The data fields of the IvanYuQuizGiver class are used to reference contents placed on the screen, such as answer buttons, the text view for the question, and the score and attempts of the user. The IvanYuQuizGiver class also has a static Singleton variable data field that will also be used to complete JsonObjectRequests. The button variables representing answer choices for the user will be given the texts corresponding to the answer choice (a to f) retrieved from the server; so for example, the text of the button data field "buttonA" will have the string corresponding to the string value assigned to the JSONObject's "answer_a" key, "buttonB" will have that of "answer_b", "buttonC" will have that of "answer_c" and so forth.

Singleton Class

The Singleton class follows the Singleton software design pattern; only 1 object of this class can be instantiated. The purpose of this class is to create a single RequestQueue, which will be used throughout the entire program in order to successfully make requests to retrieve json objects from the server. An instance of a Singleton object is also made in the onCreate method of the IvanYuQuizGiver class which is used to instantiate the RequestQueue. As an advantage, this Singleton class prevents repeated instantiation of a RequestQueue in order to make json object requests in the program.

---

[i] "Sending a Simple Request", *Android Developers*, accessed June, 22, 2016, https://developer.android.com/training/volley/simple.html

[ii] "Passing parameters to volley request", *Mobikul,* accessed June, 23, 2016 http://mobikul.com/parameters-to-volley/

[iii] "JSONObject", *Android Developers* , accessed June, 23, 2016, https://developer.android.com/reference/org/json/JSONObject.html

[iv] "Android Text Control", *Android Developers*, accessed June 23, 2016, http://www.tutorialspoint.com/android/android_textview_control.htm