CSC 214 FINAL EXAM

May 13, 2016

This exam will be a practical. It should not take you more than 2-3 hours to complete the required tasks, but you will have a 12 hour window in which to do it. **Because of this generous allotment of extra time, late submissions will absolutely not be accepted at all under any circumstances period. If you miss the closing window by even one second you will receive a 0 grade for the exam.**

Your final submission should be a compressed (ZIP) Android Studio project (exactly the same as you submit for your homework assignments).

Follow all style guidelines as much as possible.

This exam is “open internet” meaning that you are allowed to use any resources that you feel that you need to complete the requirements including Google searches, the Android developer site, Stack Overflow, etc. However, **the work that you submit must be your own**. You may not collaborate with other students during the exam, or copy code directly from the internet.

Your submission must include a README that contains the U of R honor pledge (below) followed by your name to indicate that you understand these instructions.

HONOR PLEDGE: “I affirm that I will not give or receive any unauthorized help on this exam, and that all work will be my own.”

Getting Started

To get started, download the *Final Exam* project from the Final Exam assignment on Black Board and extract it to a location on your hard drive. Use the ***Open an Existing Android Studio project*** option in the welcome screen to open the project in the location to which you extracted it, or use the ***File*** → ***Open...*** option from the Android Studio menu.

The project is already started for you including a main activity with buttons that can be used to start the activity for each problem below. Write the code necessary to implement the requirements for each problem.

Problem 1: Multiple Activities (25%)

For the first problem you will demonstrate that you can pass input parameters from one activity to another, and receive a result back. You must use best practices to pass parameters and results back and forth.

1. Create a README file and paste the honor pledge into it. **DO IT NOW**.
2. Modify MainActivity so that it starts ProblemOneFirstActivity when the "Problem One" button is pressed.
3. Add a unique tag to ProblemOneFirstActivity (hereafter **first activity)** and override each of the activity lifecycle methods so that the lifecycle is logged using the tag.
4. Add a unique tag to ProblemOneSecondActivity (hereafter **second activity**) and override each of the activity lifecycle methods so that the lifecycle is logged using the tag.
5. In the **first activity** add the code necessary to start the **second activity**. The operands typed into the edit texts should be passed in as parameters. Remember, you are expecting a result back.
6. In the **second activity** display the result of raising the first operand to the power of the second operand in the appropriate text views.
7. In the **second activity** add the code necessary to subtract the number that the user types in from the result calculated above and return that value to the **first activity**.
8. Finally, modify the **first activity** to display the result returned by the second in the text view at the bottom of the screen.

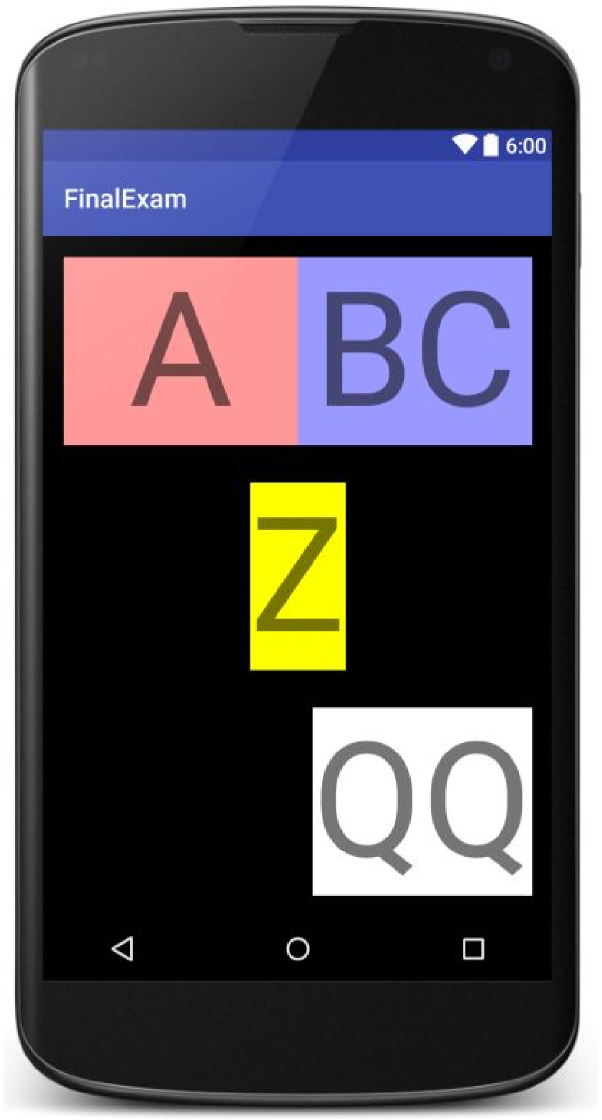
Problem 2: Communicating Between Fragments (25%)

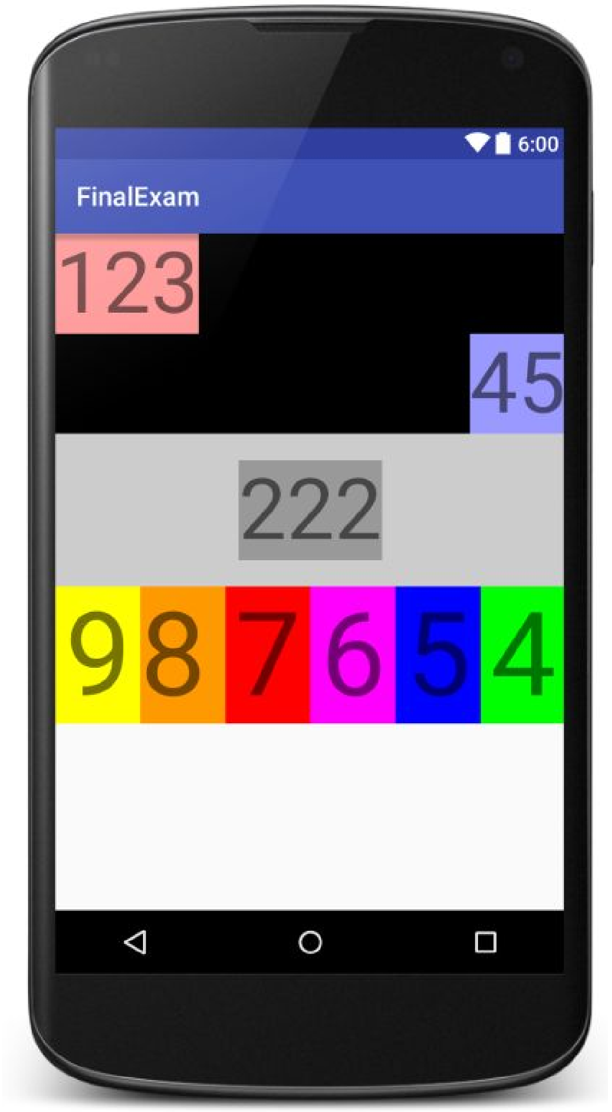
For the second problem you will demonstrate *best practices* to pass data between fragments hosted in the same activity. This means that the two fragments should not communicate directly, but should instead use the host activity to communicate. You should also make use of factory methods where appropriate.

1. Modify the Main Activity so that pressing the "problem 2" button starts ProblemTwoActivity (hereafter **problem two**).
2. Modify the layout for **problem two** such that ProblemTwoTopFragment (hereafter **top fragment**) is *hard wired* into the top half of the activity's user interface.
3. Add a unique tag to **top fragment** and override each of the fragment lifecycle methods so that the lifecycle is logged using the tag.
4. Write the code necessary so that, when the "send below" button is pressed in **top fragment** a new instance of ProblemTwoBottomFragment (hereafter **bottom fragment**) is dynamically added to the bottom half of **problem two**'s user interface. The newly added fragment's text view should display the text that was typed into **top fragment** at the time that the button was pressed.
5. Write the code necessary so that, when the "upper" button is pressed in **bottom fragment** that the string message is converted to all upper case and displayed in the **top fragment**'s text view.
6. Write the code necessary so that, when the "lower" button is pressed in **bottom fragment** that the string message is converted to all lower case and displayed in the **top fragment**'s text view.

Problem 3: Layouts (20%; 10% each)

1. Modify Main Activity so that pressing the "Problem 3A" button launches AActivity, and pressing the "Problem 3B" button launches BActivity.
2. Modify the layouts for AActivity and BActivity to match the screenshots below as closely as possible. **Do not worry about exact color matching** (just as long as I can tell where one widget's borders are relative to another).





AActivity ***BActivity***

Problem 4: Kitchen Sink (30%)

This final problem will require you to combine the file system, networking, and asynchronous tasks to download and display an image. The bitmap scaling code has been provided for you in the ImageDownloaderActivity (hereafter **downloader**). You will find the lecture 16 and lecture 20 slides useful as well as the lecture 16 and lecture 20 code examples in the course Github repository.

1. Do a Google search and find an image that you'd like to download and display in your app. Copy the URL for the image.
2. Modify the Main Activity so that pressing the "Problem 4" button starts the **downloader**.
3. Modify the **downloader** so that when the "download" button is pressed, an asynchronous task is used to download the image and save it to a file in app's private storage area. Once the image is completely downloaded and saved, display the image in the **downloader**'s image view.