CS4518 **2020 A-Term**

Project 3: A More Feature-Rich Basketball Application

Lecturer: Prof. Tian Guo

Student Name: *YOUR NAME.*

1 Project Overview

In this project, you will be refactoring the simple basketball score application developed in Project 2 in three steps, as specified in Section 2 Incrementally adding more features is not only a common practice in software industry, but also presents the opportunity for you to practice each new concept in isolation. Moreover, you will be more likely to end up with a sizable project that can enhance your resume!

1.1 Learning Goals

By completing this project, the student will:

- Gain deeper understanding about how to pass control and data between activities with **Intent**;
- Have a practical knowledge about the use of the popular **Fragment** for enabling UI design flexibility;
- Obtain basic practice regarding the use of **RecyclerViewer** to efficiently display more data than that can be fit in the screen;
- Increase the comfort level working with more complex Kotlin-based Android applications.

1.2 Project Logistics

A few important things to know before starting this project.

- This **individual** project is due by Friday, September 18th 2020 and accounts for 6% of total course points. This project contains a number of moving pieces that will take **more** time, compared to the previous two projects, to complete[1]. Please do start early on this project.
- Please use the Slack Project Channel for general discussion. I encourage you to share ideas, advice and resources with each other. Please contact me if you have questions about what constitutes appropriate collaboration.
- Project instructions are provided based on Android Studio. You may, however, choose to use any IDEs of your choices. But keep in mind that teaching staffs will less likely be able to provide helps if needed.

[1]The workload increase is by design. If you have done a 5K training program, you will know what I mean.

1.3 Submission Guideline

For this project, you will submit the following items in one **zip file**:

- **The source code.** Please make sure to clean the project before zipping the source code. This helps reducing the total file size of the submission. To do so, you can go to "Build" and then "Clean Project" via the Android Studio's menu bar. *Note: I also introduced another way to achieve the same effect with the* `gradle clean` *task in Project 1.*

- **The PDF writeup.** This writeup should include the following aspects: *(i)* design rationale *(ii)* feature screenshots *(iii)* a reflection section that summarizes what you have learnt and any potential problems with the submitted implementation. *Note: It is important to spend the additional time to write down what you have learnt, as the process helps recalling and therefore internalizing the knowledge.*

  There will not be explicit requirements regarding report format, and you are welcome to write the report by extending the included latex project file. *Note: The use of latex is not required as it is not part of this course's key learning goals.*

  There is not page limits, but we ask you to write in concise prose. Having a shorter writeup helps emphasizes the important aspects you would like to convey to the teaching staffs.

1.4 Rubric

You will earn the specified points by successfully completing the corresponding category.

Category	Point(s)
Basic Requirements	5.5
PDF Document Quality	0.5

Evaluation of the basic requirements will be based on a combination of factors including correctness, code quality (e.g., readability), and compliance with submission guidelines.

The PDF document will be assessed based on the readability and completeness.

2 Project Overview

In this project, you will be adding bells and whistles to the simple Android application from Project 2[2].

Example screenshots from a reference implementation will be included for each part. Your implementation does not have to match the reference app exactly, but it is recommended that you follow the steps outlined in Section 2.1 to Section 2.3. Completing the project in the specified order facilitates obtaining deliverables along the way.

To start, you will be refactoring the code we wrote for the previous Project 2. *Note: Even though we will not be re-evaluating features from Project 2, having a working version will make completing Project 3 smoother. Do visit the office hours if you need help setting up the Project 2.*

[2] And at the same time, practice some useful Android development skills :)

2.1 Part 1: Working with Intent (2 Points)

In this part, you will be adding a new activity that will take data passed from the main activity and return data back as well. In short, you will work with Intent, Intent extra, and the two activity methods, i.e., `startActivityForResult(...)` and `onActivityResult(...)`. Screenshots of a reference implementation and the workflow are shown in Figure 1. Your design and implementation can be different, provided that you complete the steps describe below.

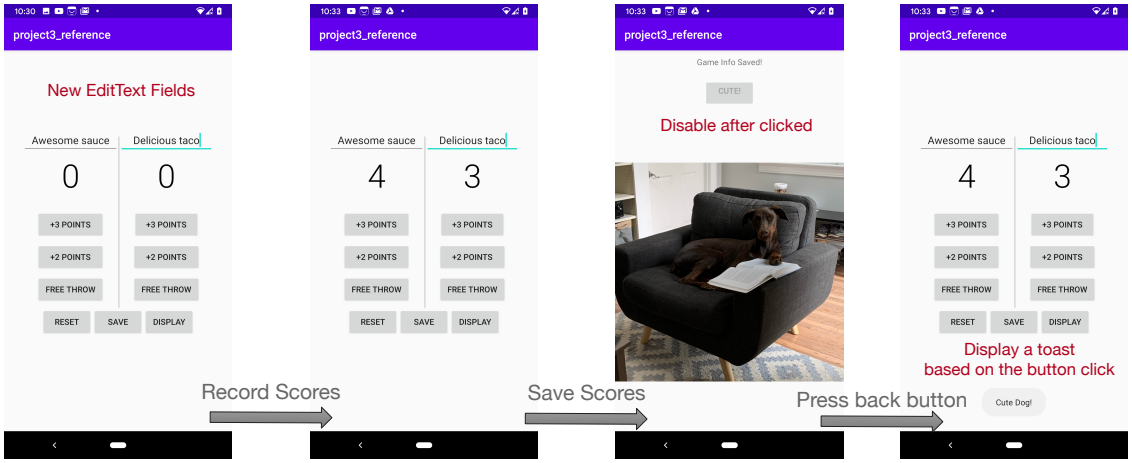

Figure 1: Example workflow with two activities.

First, you will design a second activity which will be launched by a button click event. *Note: Recall the steps you took in Project 1 for setting up any new activities.* Once done, you can test this new activity on the Android device (virtual or physical) by **temporarily** specifying it to be the launcher activity. *Hint: This can be achieved by modifying the AndroidManifest.xml file.* Take the corresponding screenshot and include in the PDF writeup.

(1 Point) Second, you will design the data and control sequences between the two activities in a sequence diagram. *Hint: If you need a refresher, you can reference The Big Nerd Range Guide Figure 6.10 Page 129.* Include the sequence diagram in the PDF writeup.

(1 Point) Third, you will refactor the code based on the sequence diagram. Once done, you need to test the execution sequence by *(i)* interacting with app and *(ii)* printing out the log messages using LogCat. Include the filtered LogCat output, similar to what you have done for Project 1, in the PDF writeup.

Lastly, you will save your progress. One option is to follow the source code submission guideline in Section 1.3 and then put the project code into a directory called `Part1/`.[3]

2.2 Part 2: Working with Fragment (1 Point)

In this part, you will be refactoring the main (i.e., launcher) activity to host a fragment. This exercise familiarizes you with the overall workflow to design and implement fragment-based applications. You should use the **dynamic** method to embed the fragment to the hosting activity, as it is a more commonly used approach. The refactored code should have the same visual effect as shown in Figure 2.

[3]If you are familiar with version control such as Git, you can save your progress in a feature branch called `feature/Part1`. Version control is widely used in the software industry and if you want to start using it today, try this Github: Hello World.

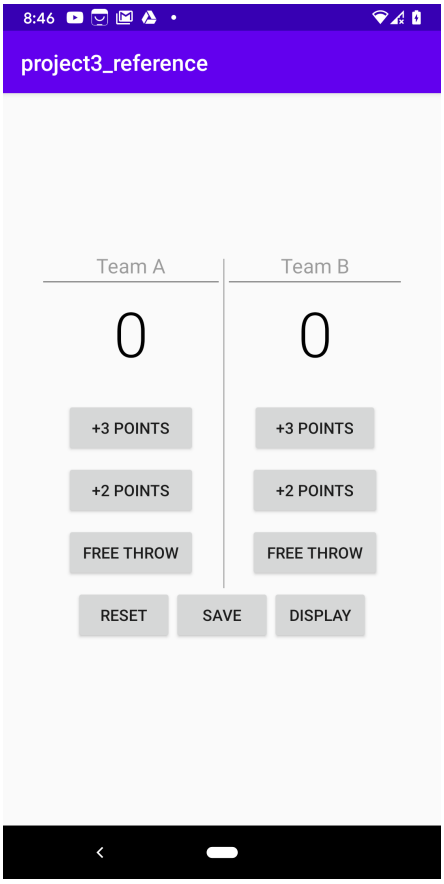

Figure 2: Example of a fragment-based activity.

Once you are done with the refactoring, you will save the progress using the same strategy as described in Section 2.1: either create a directory called `Part2/` or label it as a feature branch called `feature/Part2`.

2.3 Part 3: Working with RecyclerViewer (2.5 Points)

In this last part, you will be adding a summary fragment that displays the game history in a resource-efficient manner. You will be leveraging your newly acquired skill of Fragment and combining it with the RecyclerViewer, which allows only loading just a "screenful" information to ensure smooth scroll. An example screenshot is shown in Figure 3.

To get started, you will need to first plan out all the components needed to support the use of a RecyclerViewer, similar to Figure 9.9 of Page 187 of The Big Nerd Range Guide.

(0.5 Points) First, you will create a ViewModel that holds on to 100 randomly initialized game information. In the Figure 3, this includes a date, the names of the two teams, and the end game scores. In your implementation, you should also add **an index field** to each game to facilitate the testing (similar to "Crime #10" Figure 9.11 Page 189 of The Big Nerd Range Guide).

(0.5 Points) Next, you will implement a Fragment that is associated with the RecyclerViewer layout.

Figure 3: Example screenshot of RecyclerViewer-based Fragment.

(0.5 Points) Then, you will design how each individual child view of the RecyclerViewer should look at and implement its layout XML file. *Note: All the items will share the same XML file.*

(0.5 Points) Fourth, you will connect the items to the RecyclerViewer with the help of an Adapter. In essence, the Adapter will help inflate the layout for individual item and will be at the service of the RecyclerViewer. *Note: If you need a refresher, you can look at Figure 9.10 of Page 187 of The Big Nerd Range Guide.*

(0.5 Points) Finally, you can test the RecyclerViewer-powered Fragment by adding it to the Main activity. Save a resulting screenshot that displays starting from Game 21 at the top, and include it in the PDF writeup. *Note: For now, you will need to comment out the Fragment you created in Section 2.2.*

Once you are done with the refactoring, you will save the progress using the same strategy as described above: either create a directory called `Part3/` or label it as a feature branch called `feature/Part3`.