Assignment 2: Android Game Score App

- This assignment is to be done **individually or in pairs**. Do not share your code or solution with others who are not in your group, and do not copy code found online. Ask questions on Piazza discussion forum (see webpage) or in office hours.
  - You may use code provide by the instructor, or in videos provided by the instructor (such as my YouTube videos).
  - You may follow any online guides as long as it is providing you information on how to solve the problem vs a solution to the assignment.
    - If receiving help from someone, they must be teaching you not writing your code.
    - You may not resubmit code you have previously submitted for any course offering except your assignment 1 (which is expected to form the basis of your assignment 2's model).
  - If you copy a more than 4-5 lines of code from a guide/tutorial, it is a good idea to cite it in your code:
    `// Code taken from xyz.com/awesome/stuff/here.html`

1. Android App Overview

1. Maintains a list of games which have been played.
2. User can add and edit games in the list of known games played.

To learn Android programming you can use a book on Android (see course website for recommended book) or any online tutorials. The course website links a number of tutorials recorded for this course which cover many of the necessary topics.

2. Required Application Features

For marking, you must implement these "**Required Features**", but that only earns part of the marks. You must also complete some features in the "**Features you can Select to Implement**" section to earn full marks. See marking guide for details.

2.1 General Requirements

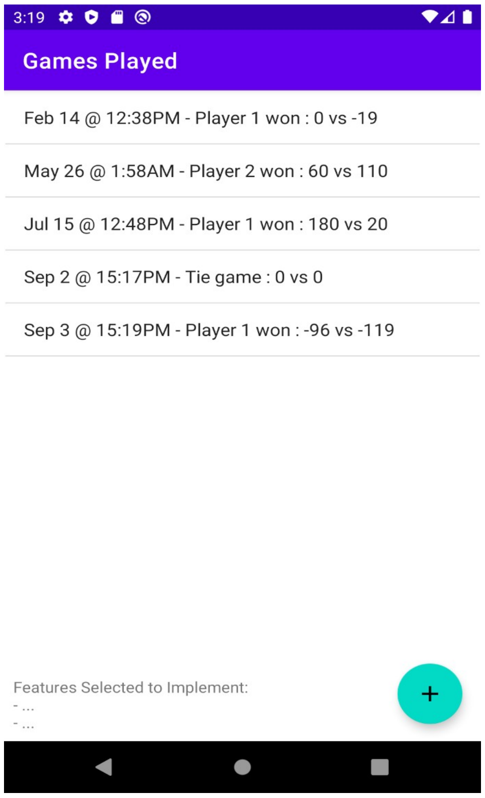
- Create an Android application targeting the minimum SDK version API 26 or lower.
  - Hint: Use API 26 so you can easily use `DateTimeFormatter`
- Use your assignment 1 solution as the basis for your model for this assignment.
  - In your Android project, under the source folder, create a new Java package (such as `ca.cmpt276.as2.model`); copy in assignment 1's model's `.java` files.
  - You will not need the text UI code.
  - You may edit your files any way you need to support your application's needs. Think of these files as a starting point: you are not in any way restricted to their interface, functionality, or design!
- Each activity should display a meaningful title. Do this in `strings.xml`.
- Screen-shots in this document are for inspiration. As long as your application correctly implements the required features, any nice and usable UI is fine.
- None of the things listed as "hints" are required; you may choose to do them or not.
- Activity files (`.java` and `.xml`) must be well named, but need not match this document.
- Create a GitLab repo on `csil-git1.cs.surrey.sfu.ca/`
  Commit your changes often (at least every 4 hours of work).
- You do not need to handle screen rotation: we will test in portrait (vertical) mode.

- Hint:
  - Each time you create a new activity for your project, choose “Empty Activity”. This will then always give you the same file structure without any extra code/fragments.

2.2 Screen 1: Games Played

- This is the initial activity displayed at startup.
- Display a list of the games played.
  - For each game, show the date, who won (or tied), and the score as shown in the figure.
- Use your solution to assignment 1 to store the games.
  - You may edit your code as needed.
  - Use the singleton design pattern[1] with your game manager class. See video on website.
- Use a Floating Action Button (FAB) to allow the user to add a new game to the collection by launching a new game activity.
  - Change the icon on the FAB to be a + See video on course website.
- User may tap on a game in the list to launch the activity to view and edit the game.
- In the bottom left, list the features you have implemented from “Features you can Select to Implement”
- You must make your game manager class a singleton; therefore, you’ll be able to access your model (collection of games) with code similar to the following anywhere in your app:

```
GameManager manager =
        GameManager.getInstance();
manager.add(new Game(....));
```


3:19

Games Played

Feb 14 @ 12:38PM - Player 1 won : 0 vs -19

May 26 @ 1:58AM - Player 2 won : 60 vs 110

Jul 15 @ 12:48PM - Player 1 won : 180 vs 20

Sep 2 @ 15:17PM - Tie game : 0 vs 0

Sep 3 @ 15:19PM - Player 1 won : -96 vs -119

Features Selected to Implement:
- ...
- ...

Figure 1: Possible look of Games Played

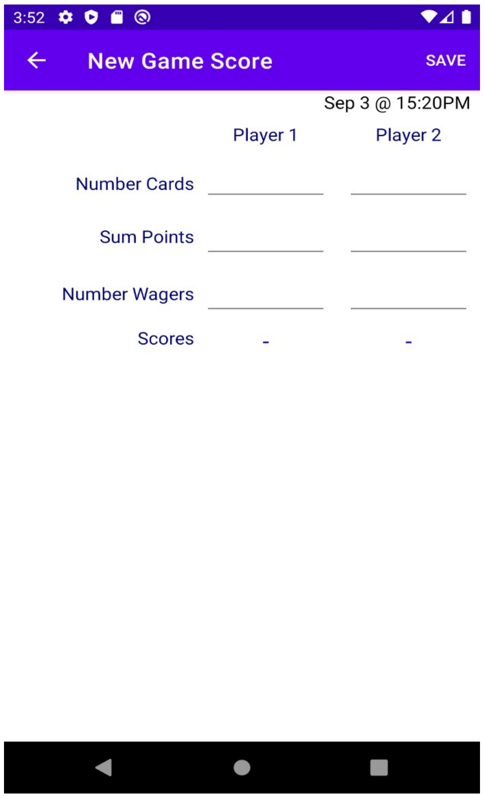
Hints

- Use a `ListView` or `RecyclerView` to show the list of games.
  - `ListView` is easier to use; `RecyclerView` is more modern and flexible but harder to use. I recommend `ListView` unless you are looking for a challenge.
  - See website for tutorial on populating the list.
- After adding a new game you’ll need to refresh your UI’s list. The simplest way to do this is to override your Games Played activity’s `onStart()` method and have it either reinitialize, or call `notifyDataSetChanged()` on, the `ListView` adapter.
- For how to pass data to another activity, see hints for the other activities.
- Avoid duplicate code: Extracting duplicate code to a function can help you better understand your code structure.

1 The Singleton pattern allows one instance of your model to exist for the lifetime of your application. Android activities come and go depending on events such as rotating the screen. If your Activity just held onto a `GameManager` object, then it would be destroyed every time the screen rotated and be inaccessible when you switched to a new activity. The singleton allows all activities to access the same instance of your model.

2.3 Screen 2: New Game

- Set the title to state it's a new game
- Only support 2 player games
- Allows user to enter the game's required values.
- All fields must allow only non-negative integers
- Navigation:
  - Tapping the Android back button (at the bottom) cancels and does not save
  - On the AppBar, tapping the left arrow (called Up) cancels and does not save
  - On the AppBar, tapping SAVE saves the new game and returns to the games played screen.
- When saving, app must check for the following errors. If either error is present, display a good error message and abort the save operation (don't save and stay on the screen). If both errors are present, show a message for at least one. You may use a Toast or something in the UI to show errors.
  - If either score is incomplete, such as no data for Number Cards, or Number Cards is > 0 but either of its other two fields don't a value.
  - Number Cards is 0, but there are non-zero values in for Sum Points or Number Wagers.
- When any field changes, recalculate the score
  - Shows score '-' for players which it cannot yet compute a score (no data)
  - Shows if game is a tie or who won when scores computed



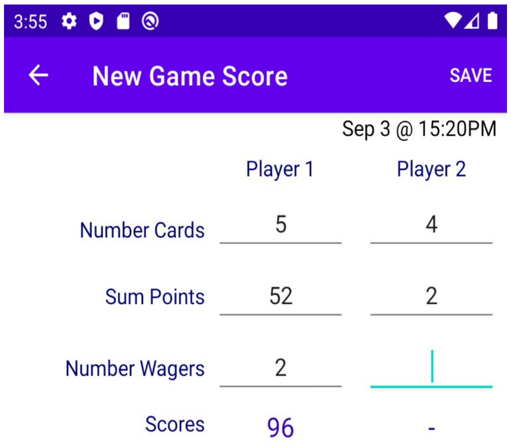
3:52

New Game Score SAVE

Sep 3 @ 15:20PM

	Player 1	Player 2
Number Cards		
Sum Points		
Number Wagers		
Scores	-	-

Figure 2: Possible look of add game activity.



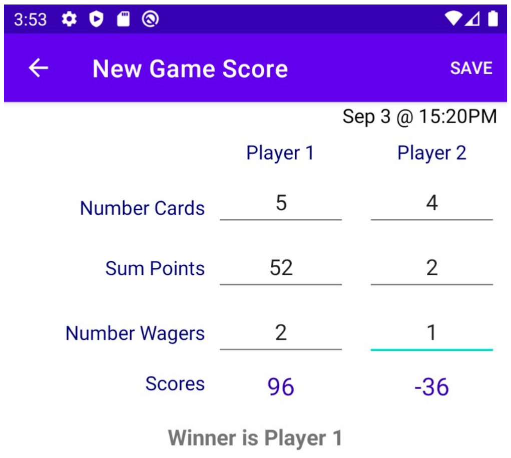
3:55

New Game Score SAVE

Sep 3 @ 15:20PM

	Player 1	Player 2
Number Cards	5	4
Sum Points	52	2
Number Wagers	2	
Scores	96	-

Figure 3: Possible look while filling in



3:53

New Game Score SAVE

Sep 3 @ 15:20PM

	Player 1	Player 2
Number Cards	5	4
Sum Points	52	2
Number Wagers	2	1
Scores	96	-36

Winner is Player 1

Figure 4: Possible look once all data is filled

Requirement on Launching and Passing Data to an activity

- Data is passed from one activity to another using an `Intent`. See video: https://www.youtube.com/watch?v=SaXYFHYGLj4
  - *Watch the video!* It shows the **required** way of creating an intent and respecting encapsulation; and it's testable.
  - For this app we can add the new games directly into the singleton model; therefore, we don't need to return a result from the activity.

Hints

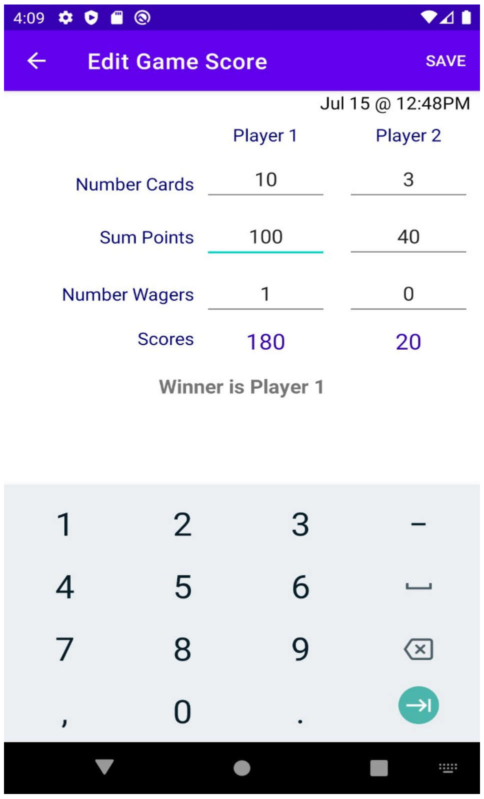
- To convert a `String` to an `int` or `double` with:
  `int x = Integer.parseInt("200");`
  - If the `String` does not contain a number it throws `NumberFormatException`.
- When adding a new game, you must refresh your Games Played activity's list. See tip in previous section related to `onStart()`

2.4 View / Edit Game

When a user taps a game on the first screen, it shows the user the same screen as for New Game, except reconfigured to work with an existing game.

You must use the same activity class and .XML file for viewing/editing the game as creating a new one.

- Display a title stating it is editing the game.
- Populate the UI with the details of the game.
- Show the date when the game was initially created.
- Allow user to edit the same fields as before, recalculating the score on each change.
- Navigation works as before
  - Ensure cancel (back or up) don't save changes
  - Ensure save edits the existing game, preserving the creation date/time.



Jul 15 @ 12:48PM

	Player 1	Player 2
Number Cards	10	3
Sum Points	100	40
Number Wagers	1	0
Scores	180	20

Winner is Player 1

Figure 5: Look when editing a game

3. Features you can Select to Implement

These features are ones you can pick and choose between. You may earn any number of these marks (see marking guide online). However, your total score for the assignment is capped at just over 100% (i.e., some bonus are possible).

By completing one or more of these features, you stand to move from “Good work” to “Great work”. **If you do not implement any of these features then you cannot score 100%.** See marking guide for specifics on what each portion of the assignment is worth.

You may only get marks for the features listed here if the “required” parts of the application work well (need not be perfect).

If you attempt any of these features, your first activity must state the features you added.

- List the feature’s number and name.
  - For example, have a `TextView` at the bottom of the screen listing the optional features you completed.
  - *Hint*: Enter text into the `TextView` in one line as follows (`\n` for linefeed):
    `Optional Features:\n1.\n3. Error Checking Inputs\n4.Auto-recalculate`
- You may also briefly mention how to access the feature (if not clear from your UI).

You may attempt any of these features, in any order.

3.1 Delete Game

- Support deleting a game.
- For example, add a button to the Edit Game Score screen which allows the user to delete.
- Must confirm with the user first that they in fact do want to delete it!

3.2 Complex List View

- Change your ListView (or RecyclerView) to instead of displaying a single string, display a complex view for each game:
  - Display an icon for each game showing if player 1 or player 2 won, or if the game was a tie.
  - In text, also state who won.
  - Display the game date and game score in different text boxes.
  - Style the display so it looks good.

3.3 Confirm on Cancel Edit

- If some data has been entered for a new game, or if any data has been edited for an existing game, then display a confirmation dialog if the user tries to cancel.
- Handle pressing android back button, or pressing the AppBar’s “up” button.
- Correctly save and discard data as required.

3.4 Save Data

- Save the games between executions of your application.
- Hints:
  - You may want to use `SharedPreferences`.
  - You may want to edit your game manager class to support working with a

  `SharedPreference` so it can load/save.
  - You *may* use external serialization libraries if you like (Gson/JSON/....).

3.5 Empty State

- Empty state is when the UI has nothing to show the user yet (such as no games for the main game list), but instead of showing nothing gives the user a hint about what they could do.
- When your application has no games to show on the main screen, help the user learn to use the program by:
  - display a nice looking message on the main screen stating there are no games yet
  - show the user how to create a new game
  - include an image or clip art
  - use a nice multi-element layout (not just a single `TextView`)
- The empty state must show any time the list is empty, and correctly disappear when the list is not empty.

4. Deliverables

To CourSys (https://coursys.sfu.ca/) you must submit:

1. **ZIP file of your project, as per directions on course website.**
   You can generate the ZIP file from Android Studio (File > Export > Export to Zip file), or from GitLab (project screen, select Download Source Code in a ZIP file).

   If you worked individually, you will need to create a group in CourSys which consists of just you before you can submit your ZIP file.

   If you worked in a pair, you will need to create a group in CourSys and invite your partner. Please ensure your partner accepts the invitation so that everyone gets credit for the work.

2. **URL and Tag for your Git repository:**
   1. Add the TA for the course as a "Developer" member of your repo:
      - Goto csil-git1.cs.surrey.sfu.ca and select your project
      - On the left hand side, click the cog-wheel drop-down ('Settings')
      - Select "Members"
      - Add both TA to your repo as a **Developer**. TA SFU IDs = **mrp8**

   2. Create a tag for your submission as follows:
      - In Android Studio, go to VCS --> Git --> Tag...
      - Enter a name for your tag, such as: `final_submission`
      - Leave Commit and Message blank.
      - Click Create Tag
      - Push changes to remote repo. On "Push Commits" dialog, select **"Push Tags: All"**.
      - You can check the tag was pushed correctly in GitLab online.
      - (If you resubmit, create a new tag as above and submit the new tag via CourSys).

   3. Submit the git@... URL and tag name to CourSys
      - Find Git URL on csil-git1.cs.surrey.sfu.ca/. Should be similar to:
      git@csil-git1.cs.surrey.sfu.ca:yourid/myProjName.git
      - The "tag" is the name you used above, such as "`final_submission`"

Please remember that all submissions will automatically be compared for unexplainable similar submissions. Everyone's submissions will be quite similar, given the nature of this assignment, but please make sure you do your own original work; we will still be checking.