Assignment 2: Camera Depth of Field – Android

Pre-Assignment Notes

The **late penalty** is -10% per day late. After two days, late submissions won't be accepted.

This assignment can be completed **individually OR in pairs**, so do not share your code or solution with others who are not in your group, and don't copy code you found online. There's still the class's Piazza if you want to ask questions to the group, or you can ask during lecture, send an email to the class's help account, or attend an office hour on Discord.

Note you can use any code shared by the instructor or anything from class or the tutorial videos linked on the course website. You *can* get outside help from guides and other people, just try to make sure they're teaching you how to solve the assignment, not writing the code for you. You won't learn anything that way, and they can't write your exam or midterm for you. If in doubt about whether something is plagiarism or not, don't be afraid to reach out and ask!

One tip: if you copy more than 4-5 lines of code from a guide or tutorial, try citing it in your code with a comment, like //Code found at [url goes here].

Whether or not you're working in pairs, you will have to use the group submission feature in Coursys to submit! There will be more details about this in the deliverables section of this document, so don't ignore this.

1. Android App Overview

The goal of this assignment is to make an Android app where:

1. The user can add (and possibly edit or remove) lenses in the list of known lenses.
2. The user selects a lens and enters information about the photo they are taking. The app captures and displays the depth of field for the photo.

If you haven't yet, be sure to complete the optional section 2 of assignment 1 to help you get set up with Android. To remind you, the key links were the Java SDK (which you should have if you completed assignment 1) and Android Studio (https://developer.android.com/studio/index.html)

IMPORTANT NOTES ABOUT ANDROID STUDIO: Unlike with IntelliJ, **it's safe to use the Gradle build system** with the latest version of Android we're using in Android Studio. In fact, I'll be using that build system for the sample solutions to assignments 2 and 3, so feel free to use it yourself and follow steps that include it in any tutorial videos.

There can still be compatibility issues, however, due to the work-from-home situation meaning that we're all working on different hardware. For this course, we will try to target Android 10.0, API level 29 (also called Q). When setting up a virtual Android device through Android Studio to test your app, I also recommend creating a standard-sized Pixel 2 phone with those same Android settings. If you are unable to run an Android virtual device with these settings, you may need to look into the resources for working remotely on a lab computer described on the course website's assignment page.

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

2. Required Application Features

Implementing the required features can earn you up to 75% on this assignment. To get the remaining 25%, you must also complete some optional features from the later list.

2.1 General Requirements

Create an Android application targeting the Android 10.0 SDK version 29 (Q). Use your assignment 1 solution as the basis for your model for this assignment, though once the solution to assignment 1 is posted you may also look at it to help fix any errors you may have had. You may edit your files any way you need to support your application's needs.

Each activity should display a meaningful title. Do this in strings.xml. Activity files (.java and .xml) must be well named, but need not match this document. Screen shots in this document are for inspiration, as long as your application correctly implements the required features, any nice and usable UI appearance/layout is fine. You do not need to handle screen rotation.

None of the things listed as hints are required, you may choose to do them or not. Create a GitLab repo on csil-git1.cs.surrey.sfu.ca/ and commit your changes often (at least every 4 hours).

Hint: each time you create a new activity for your project, choose “Basic Activity”. This will then always give you the same file structure. Delete the floating action bar if not needed.

2.2 Screen 1: Lens List

This is the initial activity displayed at startup. Display the lenses from your model. For each lens, show the make, focal length, and maximum aperture.

Use your solution to assignment 1 to store the lenses. You may edit your code as needed. Use the singleton design pattern[1] with your lens manager (see the video on the website for more).

Use a Floating Action Button (FAB) to allow the user to add a new lens to the collection by launching a new activity to enter lens details. Change the icon on the FAB to be a plus symbol (+), see another video on the course website for more. The user may tap on a lens in the list to launch the Calculate Depth of Field activity.

When your app starts up, pre-populate the lens manager with the lenses show in the screenshot below:

Hints: You must make your lens manager class a singleton; therefore, you'll be able to access your model (collection of lenses) with code similar to:

```
LensManager lenses = LensManager.getInstance();
Lens lens = lenses.get(0);
```

Use a ListView or RecyclerView to show the list of lenses. ListView is easier to use, RecyclerView is more modern and flexible, but harder to use. Just use ListView unless you want a challenge. The website has another tutorial on populating the list.

After adding a new lens, you'll need to refresh your UI lens list. The simplest way to do this is to fully reinitialize the ListView (create a new adapter for the ListView). Put this code in its own method to avoid needless duplication. For how to pass data to another activity, see the hints on the other activities!

---

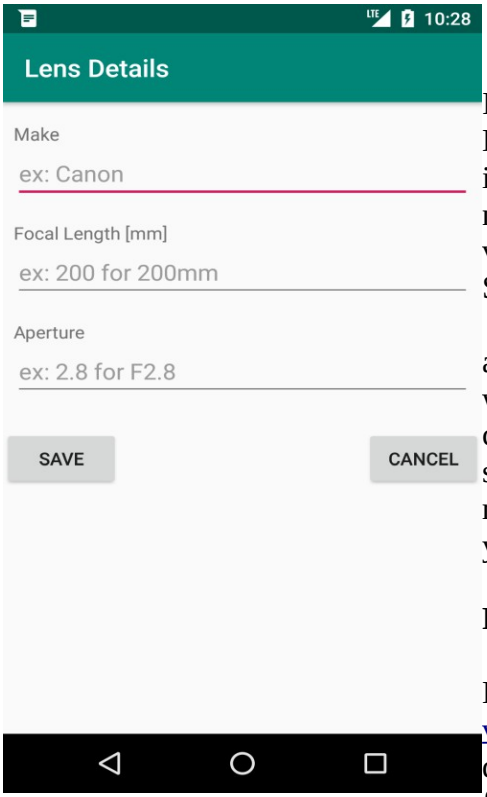
1 The Singleton design pattern allows your model to exist the entire time your application is running. Android activities come and go depending on a number of things, such as rotating the screen. If your activity just held onto the LensManager object, then it would be destroyed every time the screen rotated and be inaccessible when you switched to another activity. Singletons allow all activities to access the same instance of your model (LensManager)

2.3 Screen 2: Add Lens

Have entry boxes for the lens's required values. If using EditText widgets for data input, each must have a hint for what goes in it (as shown in the screenshot below). You may use other data entry widgets if you wish.

For focal length entry, only allow non-negative integer values. For aperture entry, if using a text entry box, only allow non-negative floating point values. On the other hand, you can also use a drop-down box, but check online for some of the most common aperture values.

Have a way of either accepting or cancelling adding the lens, such as Ok / Cancel buttons. When we get to optional features, consider replacing with Back/Save arrows in the action bar at the top of the screen.



Hints: Convert a String to an int or double with int x = Integer.parseInt(“200”); or double y = Double.parseDouble(“2.0”);. Note that if the string does not include a number, it throws a NumberFormatException. You need not handle validating the user's input – if the user doesn't add any value and just clicks Ok, your program may crash, and that's okay. See optional features for fixing this issue.

When adding a new lens, you must refresh your Lens List activity's list. Have your Lens List activity launch this activity with startActivityForResult(). When this activity finishes, it will call your Lens List activity's onActivityResult() method, which should refresh the list view by either reinitializing the list view by recreating a new adapter, or calling notifyDataSetChanged() on your adapters.

Requirements on Launching and Passing Data to an activity

Data is passed from one activity to another using an Intent. Here's a video: https://www.youtube.com/watch?v=SaXYFHYGLj4 This video has been linked directly in this document instead of simply telling you it's on the course website for extra emphasis: watch this video for how to create an intent in a way that respects encapsulation.

For this app we can add the new lens directly into the model (via the Singleton), therefore it does not matter what result we return from this activity.

The required steps to launching Add Lens activity from Lens List activity:

1. Add Lens provides a method for Lens List to call and get the launch intent:
2. Lens List calls startActivityForResult(...) to launch Add Lens.
3. Lens List overrides onActivityResult(...) to handle the “returned” data.

2.4 Screen 3: Calculate Depth of Field

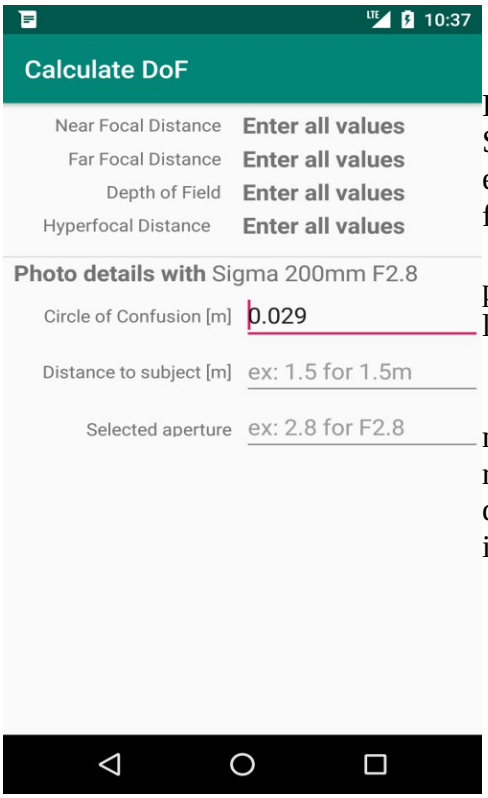
Display the selected lens's description. Allow users to enter:

1. Camera's circle of confusion.
2. Distance to subject (in meters)
3. Selected aperture (the F number)

The UI must allow only non-negative decimal values and pre-populate the circle of confusion with 0.029. When the three fields are filled, use a button to calculate and display the four depth of field values.

Like Add Lens, you must use a public static method to encapsulate creating the Intent to launch this activity, see the hints below for more.

If the entered aperture is less than the lens's maximum aperture, display an error message such as “Invalid aperture”. It's okay to display “NaN” (not a number) if the user enters a circle of confusion of 0. (This will likely happen automatically when you calculate the values).

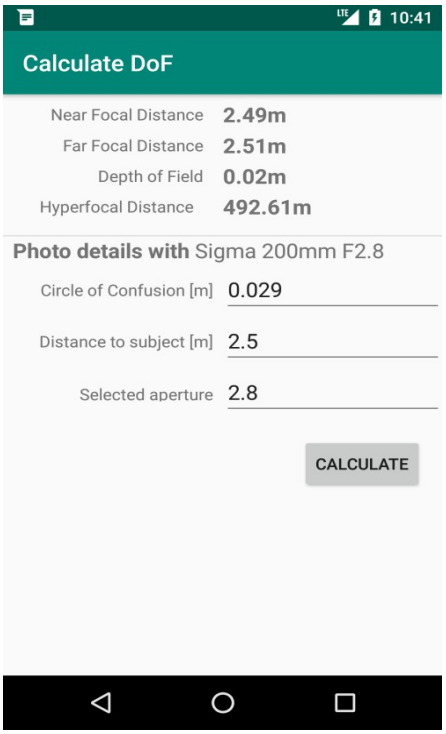


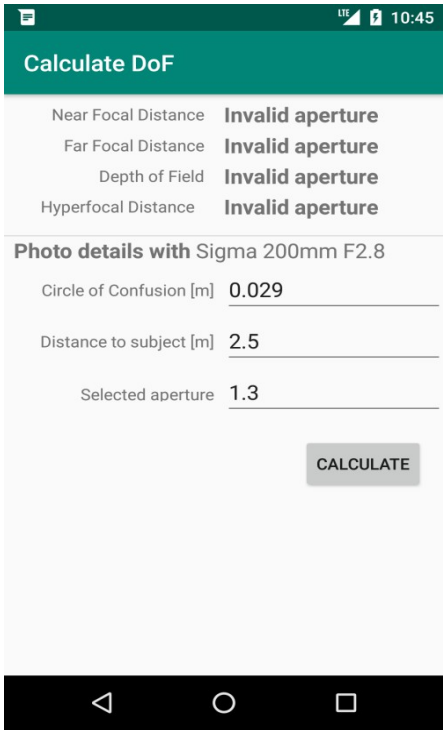
Hints: Pass data to the Calculate activity using an Intent. Pass in the lens's index for accesing it via the LensManager (the Singleton!). For good encapsulation, have the Calculate activity expose a method which creates the intent to start it. Pass this function the data to be encoded into the Intent.

public static Intent makeLaunchIntent(Context context, int lensIdx);

If trying to display a number in a TextVIew, note that myTextView.setText(42); will attempt to load the strings.xml resource which has number 42 into the TextView, which likely does not exist and will crash your program. Instead, convert this into a String: myTextView.setText(“” + 42);

Below are two more screenshots showing what the UI might look like when the user has calculated all values, or entered an invalid aperture.





3. Optional Features

By completing one or more of these features, you can gain points toward the remaining 25% of the grade for this assignment. Each feature has the score you can gain by completing it listed in the title, and you can earn up to 30% (so potentially a final score of 105%) through completing more optional features. You may only get marks for the optional features if the required parts of the application work (or at least, work well enough).

If you attempt any of these features, your Lens List activity must state the features you added. List the feature number and title. For example, have a TextView at the bottom of the screen listing the optional features you completed. (**Hint**: Enter text into the TextView like Features\n5. Save data\n7. Empty State.) See the screenshot below for an example.

You may also briefly mention how to access the feature if it isn't clear from the UI. You may attempt any of these features in any order.

3.1 App Bar Buttons Via Toolbar (8%)

Use the Toolbar widget to give all activities an app bar (also called the action bar) at the top of the activities to give at least the following buttons:

1. Add Lens activity (and Edit, if you have it) should have Back and Save.
2. Calculate activity should have Back.

When adding these buttons to the tool bar, you must remove any duplicate buttons from the rest of the user interface.

3.2 Edit and Delete Lens (8%)

Support editing and removing a lens stored in the list of lenses.

Hints:

From the Calculate activity, add a button to edit the selected lens.

Re-use the Add Lens activity and just pass it extra data (via an Intent) of which lens is to be edited.

From the Calculate activity, add a button to delete the selected lens.

Make sure that you update the Lens List activity's list view when the model changes.

3.3 Error Checking Input (5%)

Enforce at least the following constraints on user input:

1. Add Lens Activity:
    a. Make length is > 0
    b. Focal length is > 0
    c. Aperture >= 1.4
2. Calculate Activity:
    a. Circle of Confusion must be > 0

b. Distance to subject > 0
c. Selected aperture >= 1.4

When you detect an error, display a good error message (if appropriate, you may use a toast). In the case of Add Lens (or Edit), prevent saving the lens until all values are valid. In the case of Calculate, only calculate once all values are valid.

3.4 Auto-recalculate (5%)

On the Calculate activity, automatically recalculate all depth of field values when the user changes any one of the input fields.

Remove any redundant buttons from the UI.

Hint: To recompute while the user is entering data, you'll need to pass a TextWatcher object to the addTextChangedListener(...) method of each of the input EditText. In this TextWatcher.afterTextChanged(...) call your code to recompute the depth of field values; other methods in TextWatcher can be left untouched. You have three input EditTexts, so you can create one TextWatcher object and then pass it to each of the EditTexts to reduce code duplication.

3.5 Save Data (10%)

Save all the lenses between executions of your application. When your app starts up, if there are no stored lenses, add the sample lenses from Assignment 1. However, if there are saved lenses then don't add these sample lenses.

Hints: You may want to use SharedPreferences, and to edit your lens manager class to support working with SharedPreferences. You may also use external serialization libraries if you like (Gson/JSON/etc).

3.6 Lens Icons (10%)

Significantly enhance the user interface by allowing the user to set an icon or image for each lens. You may, for example, have 5 built-in icons the user can choose between. Change the lens list activity to use a complex layout with the lens's icon/image and text. See the video on the course website related to making complex list views.

3.7 Empty State (5%)

When your application has no lens to show, display a nice looking message on the main screen instead of the list of lenses. The message must give the user some directions on how to start creating a new lens.

4. Deliverables

Submit the following to Coursys (https://coursys.sfu.ca/):

1. A .zip file of your project, as per directions on the course website. **If you worked individually, you will still 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 instructor and TAs as "Developer" members of your repo:
        1.Go to csil-git1.cs.surrey.sfu.ca and select your project.
        2. On the left hand side, click the cog-wheel drop down ("Settings")
        3. Select "Members"
        4. Add the instructor and TAs to your repo as developers. Use our email addresses (jackt@sfu.ca, tirthp@sfu.ca, and kpathani@sfu.ca).

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

    3. Submit the Git URL and Tag name to CourSys. The Git url ends in .git and is used to clone the repository, while the tag name is just the name you used in step 2.

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