Assignment 1: Camera Depth of Field – Java

Pre-Assignment Notes

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

This is an **individual assignment**, meaning you're not supposed to share or copy your solution/code from other students or the internet. 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!

1. Java Work

This assignment is about making a Java application, which we'll then turn into an Android app in the next assignment. Similarly, we'll be using the **IntelliJ IDE** to actually write the code for this assignment, then use the related Android Studio for the next one.

Install the latest Java SE JDK:
http://www.oracle.com/technetwork/java/javase/downloads/index.html

Install IntelliJ Community Edition (don't worry, it's free):
https://www.jetbrains.com/idea/download/

Important note about Java build systems: The Java tutorial videos linked from the course website recommend choosing the Gradle build system when creating a new project. Due to some updates leading to compatibility issues, **do not select Gradle when setting up a project.** Choose to make a default Java project, as standard compilation should be enough for our purposes.

1.1 Depth of Field (DoF) Background

When using a camera to take a photograph, the camera will focus on one point, the focal point. Anything at that distance from the camera will be in focus.

Some things closer or further than this distance will also be in focus.

The goal of this assignment is to write a program to help photographers determine how much will be in focus.

The course website lists further resources about depth of field, and Appendix B of this assignment includes details for what your application will need to calculate.

1.2 GitLab

This assignment also requires using SFU's GitLab to manage the project's code repository. If those terms are unfamiliar, don't worry, check **Appendix A** of this assignment for directions or check a video tutorial on the course website.

You must commit your work to your git repository at least three times during this assignment to help familiarize yourself with backing up code in this way. Once every hour is a good starting pace.

1.3 The Program

The code for your program should be organized into two packages (consult the videos on the course website for more on packages).

The **Model Package** is the part of the application that manages the data and most of the application's logic. The User Interface (UI) classes in the UI Package will call on classes from this package, but Model Package classes will not call on UI classes.

You should have at least three classes in the Model package:

1. A *Lens*, which stores info about a single lens, such as:
    *The *make*, a string, such as Canon, or Tamron.
    *The *maximum aperture*, which is the F-number of the lens. For an F5.6 lens, store 5.6. This represents how much light the lens can let in. Note that a larger aperture (more light, wider open) has a smaller F numbers. So, for a lens that can do F2.8 through F22, the “maximum aperture” (wide open) is 2.8
    *The *focal length*, which is related to the magnification of the lens. Smaller focal lengths are wide-angle, larger focal lengths are more zoomed in. So an 18mm lens sees quite a wide area; a 200mm lens is quite zoomed in; and the Hubble telescope is 57,600mm.
2. A *Lens Manager*, which stores a collection of lenses. This class must support adding new lenses, retrieving a specific lens by its index, and be iterable over all lenses. Be sure to watch the video tutorials on the course website to learn how to make a class like this one!
3. A *Depth of Field Calculator*, which can compute the depth of field values, when given a lens and some information about the camera's settings. Check Appendix B for more on that info.

The **Text UI Package** should include a *Text UI* class which interacts with the user by printing to the screen and reading from the keyboard. It should follow this loop:

1. Display a numbered list of known lenses.
2. Allow the user to select a lens by inputting a number. If the user inputs -1, exit. If they input an invalid index number, the command should fail (i.e. tell them it failed and ask again).
3. Ask the user for an aperture. The command should fail if the given value is larger than the lens's maximum aperture (i.e. the aperture number entered is *smaller* than the lens maximum aperture number)
4. Ask the user for the distance to the subject.

There is a sample output on the course website. There is also some sample text UI code that shows how to use the screen and keyboard. You may change this provided code, but note that it is trying to make use of the lens manager class.

When reading from the keyboard using the 'in' Scanner, assume the user always enters the correct type of value, meaning if you are expecting an integer for an index value, you should be

prepared to handle a number that is too high, but do not have to worry about a letter.

When printing a distance, use the provided formatM() function for formatting a distance (in metres) to two decimal places. Also note that while distances must be shown to the users in metres [m], the calculations in Appendix B are in millimetres [mm]

1.4 Testing

Write a JUnit 5 test class for your depth of field calculator class which tests all of its methods. Test at least three different lens and camera setting combinations.

2. Looking Ahead to Assignment 2: Starting with Android

This section will **not be graded**, it is an optional section to introduce you to Android before we start working with it in the next assignment. Just a little warm-up for you.

We'll need to start by setting up another development environment for Android:

A) Make sure you downloaded the latest Java SE JDK (which you should already have if you did these in order): http://www.oracle.com/technetwork/java/javase/downloads/index.html

B) Download and install Android Studios: https://developer.android.com/studio/index.html

2.1 Resources

If you found the recommended text, *Android Programming: The Big Nerd Ranch*, try reading chapters 1 through 5 for a good primer.

Otherwise, there are videos linked on the course website to get you started with a few basic concepts, in particular the ones on creating a button, UI layouts, Java objects in Android activities, and creating a second activity.

3. Deliverables

All assignment submissions will be done through CourSys at https://courses.cs.sfu.ca/. For this assignment, that should include:

1. A screenshot of the commits page of the GitLab repository. You should be able to access the commits page of your repository using your web browser by visiting SFU's GitLab page. This is to show you have made at least three commits.

2. The URL and Tag for your Git repository:

   a. Add the course's instructor and TAs as “Developer” members of your repo by going to csil-git1.cs.surrey.sfu.ca and select your project. On the left hand side, click the cog-wheel drop down menu (“Settings”). Select Members and add the Tas to your repo as Developers. The instructor's SFU ID is **jackt**, while the TAs' IDs will be added to the assignment page of the course website when available.

3. A .zip file of your project, as per directions on the course website.

Appendix A: Git Lab

Initial GitLab Checkin

1. Install Git on your computer.
2. Create a new project on the SFU Computing Science GitLab server:
    a. Via your web browser, log into https://csil-git1.cs.surrey.sfu.ca/ and click New Project in the top right.
    b. Name it something like cmpt276A1 and click Create Project.
    d. Copy the Git URL for the project. It should look something like: "https://csil-git1.cs.surrey.sfu.ca/jackt/cmpt276a1.git"
3. In IntelliJ or Android Studio, enable Git and commit the project:
    a. From the menu, select VCS -> Enable Version Control Integration and choose Git.
    b. You can now right-click your project folder in the Project viewer on the left side of the screen, go to Git, and select +Add to add all the files currently in your project to the repo (don't forget to +Add any new files you make!)
    c. From the menu, select VCS -> Commit, enter a description like "initial commit", then select Commit And Push.
        *If asked about Code Analysis, for the time being you can just Commit instead of reviewing issues.
        *In the Push Commits window, click Define Remote in the top left. Enter the URL you copied from above (something like https://csil-git1.cs.surrey.sfu.ca/jackt/cmpt276a1.git) and click Ok.
        *Finally, click Push!
4. Ensure the files were pushed by viewing the project in GitLab via the web.

Checking in Changes

After making changes to your code, commit the changes to your Git repository and push them to the GitLab server.

1. Select VCS -> Commit. It will present the files to be added.
2. Enter a commit message. Try and make it something useful to remind you what changed.
4. Select Commit And Push. (If asked about any code analysis warnings, you should correct them and then repeat this process; however, you could just click Commit and push the code as-is.)

Appendix B: Depth of Field

Further resources explaining depth of field calculations can be found on the course webpage.

Your depth of field calculator class will be provided:
-A **lens** (specifically, we need the lens's focal length and maximum aperture).
-The **distance** to subject (i.e., how far away is the thing we are focusing on)
-The **aperture** to use for the photo (A lens has a maximum aperture, like 2.8 ("F2.8"), which is "wide open". Its aperture can close down to be smaller, such as 22 ("F22"). The formulas below use the current aperture selected, not the lens's maximum aperture)
-The "**Circle of confusion**" of the camera (We don't really need know much of this. It can be derived for a specific cameras and it relates to how much blur the camera will record for a single point of light. We will just always be using 0.029mm for this value, as listed in the sample text UI).

From these values, we can compute the following:

1. **Hyperfocal distance**
With a given lens and camera settings, the hyperfocal distance is the distance from the camera beyond which all objects will seem in focus.
Hyper focal distance [mm] = (lens focal length [mm])^2
/
(selected aperture * camera's circle of confusion [mm])

2. **Near focal point**
Near focal point is distance from the camera to the nearest point which will seem in focus.
Near Focal Point [mm] = (hyper focal point [mm] * distance to subject [mm])
/
(hyper focal point [mm] + (distance to subject [mm] – lens focal length [mm]))

3. **Far focal point**
Far focal point is distance from the camera to the farthest point which will seem in focus.
If the subject is beyond the hyperfocal distance, then the far focal point is +Infinity
(in Java use Double.POSITIVE_INFINITY).
Far Focal Point [mm] = (hyper focal point [mm] * distance to subject [mm])
/
(hyper focal point [mm] – (distance to subject [mm] – lens focal length [mm]))

4. **Depth of field**
Depth of field [mm] = (far focal point [mm]) - (near focal point [mm])