# Assignment 1: Camera Depth of Field - Java

**Notes**

- This assignment is to be done **individually**. Do not share your code or solution, do not copy code found online; ask all questions on Piazza discussion forum (see webpage).
  - You may use code provide by the instructor, or in the guides/videos provided by the instructor (such as instructor's YouTube videos).
  - You may follow any guides you like online 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 or offering.
- Cite your sources in your code by putting the URL in a comment.

## 1. Java Work

In this assignment you will be writing a (plain old) Java application, not an Android app. Much of what you create will then be used in Assignment 2 when we implement a similar Android app.

You must submit an Android Studio project which contains your Java code. All work can be done inside Android Studio. See website for video showing how to write "plain old" Java code inside Android Studio.

Install and configure an Android development environment on a computer.

- Download the latest Java SE JDK (it includes the JRE) http://www.oracle.com/technetwork/java/javase/downloads/index.html
- Android studios: https://developer.android.com/studio/index.html
- You are encouraged to install this software on your own computer; however, if you wish, you can complete this assignment without doing an installation using the Linux CSIL labs.

### 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), and then anything at that distance from the camera will be in focus.
- In fact, some things closer to the camera, and some things farther from the camera, than the focal point are also in focus too.
- You are going to write a program to help photographers know how much will be in focus.
- Read the resources listed on the course website to understand more about depth of field.
- See Appendix B at the end of this document for details on what you will calculate.

### 1.2 GitLab

You must use SFU's GitLab for this assignment.

- See Appendix A at the end of this document for directions, or a video tutorial linked on the course website.
- You must commit your work to GitLab reasonably frequently. I suggest you do so about every hour or so of work.
- You must have 3 or more commits by the time you are done the assignment. (It might be significantly higher!)

## 1.3 The Program

Your program must have two packages, as shown in the posted Java videos:

### *Model Package*

- The model is the part of program which manages the data and much of the application logic.
  - The UI class(es) will call classes in the model.
  - Classes in the model must *not* call the UI classes[1].
- Have at least three classes in the model package:
  - **Lens**: Store info about a single lens such as its:
    - **make**: a string such as Canon, or Tamron, ...
    - **maximum aperture**: 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.
    - **focal length**: 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.
  - **Lens manager**: Store a collection of lenses.
    - Class must support adding new lenses, and retrieving a specific lens by its index.
    - *Hint: Watch the videos on the course website about learning Java: they show how to create a class very much like this!*
  - **Depth of field calculator**: Given a lens and some info about the camera settings, it can compute the depth of field values.
    - See Appendix B for what this class must compute.
- Classes in the model package must *not* print to the screen or read from the keyboard. Instead they should work with (be called by) the classes in the UI which handle the screen and keyboard.

### *Text UI Package*

- Have a text UI class which interacts with the user by printing to the screen and reading from the keyboard.
- Your interface must loop through:
  - Display the list of lenses.
  - Allow the user to select a lens.
    - Exit if user enter -1.
    - Fail if user entered an invalid valid index.
  - Ask user for the aperture
    - Fail if the aperture is larger than the lens's maximum aperture (i.e., the aperture number entered is smaller than the lens maximum aperture number).
    - Fails if the aperture is smaller than F22 (i.e., the aperture number entered is greater than 22).
  - Ask user for the distance to the subject
    - Fails if negative.
- See the sample output on the course website for the required flow when there is an error.

1 A model class can only 'call' a UI class via the observer pattern, which we will learn later in the course.

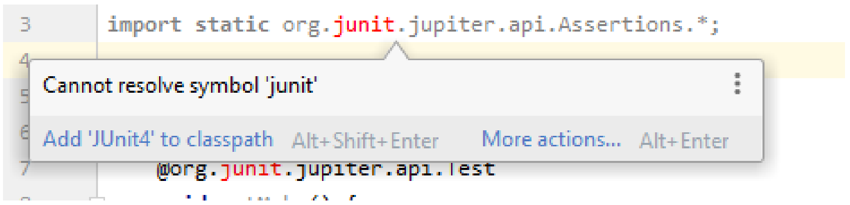
- Text UI Details
  - See the sample text UI code (on website) for how to use the screen and keyboard.
    - Change this provided code as needed.
    - Note that this code tries to make use of the lens manager class which you must write.
  - When reading from the keyboard (using the '`in`' Scanner), assume the user always enters the correct *type* of value. For example, if you are expecting an `int` (using `in.nextInt()`), then assume the user types an `int`.
  - When printing out a distance, use the provided `formatM()` function for formatting a distance (in meters) to 2 decimal places.
    - If a computed value is infinity or -infinity, you must print "INF" or "-INF". You must modify `formatM()` to do this.
    - Hint: Look at the class `Double` for constants related to infinity.
  - Be careful about units [m] vs [mm].
  - Your code will pre-populate the list of known lenses; there is no need for the user to be able to add/remove/edit the list of lenses.
    - Your text UI routines must not duplicate code for each of the required lenses. You may hard-code the creation of the required collection of lens objects, but your UI routines must access these objects to display values. For example, don't hard-code your menu to individually handle the specific provided lenses one-at-a-time.

## 1.4 Testing

- Write a JUnit 5 test class for your depth of field calculator class.
- Tests must achieve 100% code coverage of your depth-of-field calculator class (except for any asserts in your depth-of-field class).
- Must test each parameter to depth-of-field class out of range at least once.
- Test using at least two different lenses.
- Test at least the following conditions:
  - depth of field < infinity
  - depth of field = infinity

**Android Studio and JUnit 5 Build Problem**

After you create the JUnit 5 test class for your code, you may see an error: `Cannot resolve symbol 'junit'`



```
import static org.junit.jupiter.api.Assertions.*;
```

Cannot resolve symbol 'junit'

```
@org.junit.jupiter.api.Test
```

Add 'JUnit4' to classpath Alt+Shift+Enter
More actions... Alt+Enter

**Solution**

- File → Project Structure
- Select dependencies (left side)
- Select your module
- Select "junit-jupiter" in the top area
- In details below, select Requested Version to 5.6 (or newer)

## 2. Getting to Know Android Studio

Nothing from this section is to be submitted or for marks. However, the next assignment will require to to create an Android app. Therefore, it is **strongly** recommended that you start learning Android now.

### 2.1 Resources

- Read chapters 1 to 5 of Android Programming: The Big Nerd Ranch book.
- You may also want watch some of the provided videos on YouTube. Suggested videos:
  - Creating a button
  - UI Layouts
  - Java Objects in Android Activities
  - Creating a 2nd activity.

## 3. Deliverables

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

1. Screenshot of the commits page of GitLab repo (on csil-git1.cs.surrey.sfu.ca).
   - In web browser, open your repository and click “Commits”.
   - It should now show at least 3 commits.
   *(If there were problems with Git while completing the assignment, you are welcome to create three trivial commits, even if you have completed the rest of the assignment).*
   - Screenshot this and submit the image.

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 just the TA to your repo as a **Developer**. TA SFU ID = **srchauha**

   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`”

3. ZIP file of your project, as per directions on course website.

Please remember that all submissions will automatically be compared for unexplainable similar submissions. Please make sure you do your own original work.

## Appendix A: Git Lab

### Initial GitLab Checkin

1. Install Git on your computer.
2. Create a new project on SFU Computing Science GitLab server
   a. Via a web browser, log into `https://csil-git1.cs.surrey.sfu.ca/` and click "*New Project*" (top right).
   b. Name it something like `cmpt276As1` and click "*Create project*"
   c. Generate an SSH key on your computer and upload it to GitLab. See video on course website for directions in Windows.
   d. Copy the Git URL for the project. It's near the top middle of the page.
      In the drop-down to its left, select "SSH", then the URL should be something like:
      `git@csil-git1.cs.surrey.sfu.ca:bfraser/cmpt276As1.git`
3. In IntelliJ or Android Studio, enable Git and commit the project:
   a. From the menu select *VCS* --> *Enable Version Control Integration*
   b. Select *Git* and press *OK*.
   c. In the very bottom left corner of IntelliJ / Android Studio, hover over the button to show the applications menu and select "*Version Control*"
      - If it's not there, then enabling VCS likely failed because the IDE could not find Git. Ensure you have Git installed and then go to:
        File --> Settings --> Version Control --> Git, and set the path to Git
        (on Windows likely "`C:\Program Files\Git\bin\git.exe`")
   d. Expand the *Unversioned Files* and select them all.
   e. Right click the selected files and say "*Add to VCS*"
   f. From the menu select *VCS* --> *Commit Changes*.
      Enter a description like "Initial commit".
      Hover over the *Commit* button, from the drop-down 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 that `git@csil-git1.cs.surrey.sfu.ca:...` URL you copied from above and click *OK*.
      - 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 app, commit the changes to Git and push them to the GitLab server.

1. Hover over the bottom-left corner of IntelliJ or Android Studio to see the applications menu. Select *Version Control*.
2. Right-click in the panel and select "*Commit Changes*"
3. Enter a meaningful commit message.
4. Hover over the *Commit* button and select "*Commit and Push*"
   - If asked about any code analysis warnings, you should correct them and then repeat this process; however, you may just click *Commit* to push the code you currently have.

## Appendix B: Depth of Field

See resources linked on course website for more explanation of depth of field computations.

Your depth of field calculator class will be provided the following data:

- a lens (specifically, we need the lens's focal length and maximum aperture)
- distance to subject (i.e., how far away is the thing we are focusing on)
- 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 mention aperture: this is the *current* aperture selected, not the lens's *maximum* aperture.
- "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.

Your depth of field calculator class must throw an `IllegalArgumentException` when:

- distance to subject is negative
- aperture to use for the photo is more wide open than the lens supports (i.e., the selected aperture's F-number is smaller than the lens's maximum aperture F-number).
- Circle of confusion is less than or equal to zero. You may ignore this if your depth-of-field calculator does not accept the circle of confusion as an argument.
- lens object is null

Either your depth-of-field calculator class, or your lens class, must throw an `IllegalArgumentException` for:

- Lens make either null, or length 0.
- maximum aperture less than or equal to 0
- focal length less than or equal to 0

From these values, we can compute the following:

- **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.

  **Hyperfocal distance [mm]**
  = (lens focal length [mm])$^2$ / (selected aperture * camera's circle of confusion [mm])

- **Near focal point**
  - Near focal point is distance from the camera to the nearest point which will seem in focus.

  **Near Focal Point [mm]**
  = (hyperfocal distance [mm] * distance to subject [mm])
  /
  (hyperfocal distance [mm] + (distance to subject [mm] – lens focal length [mm]))

- **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]**
  = (hyperfocal distance [mm] * distance to subject [mm])
  /
  (hyperfocal distance [mm] – (distance to subject [mm] – lens focal length [mm]))

- **Depth of field**

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