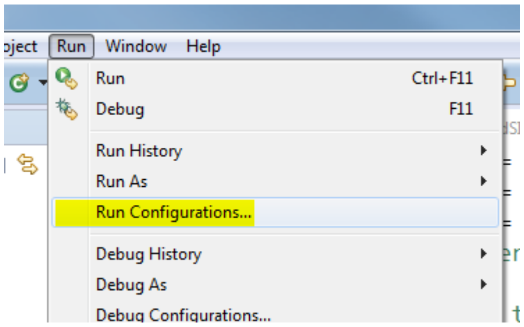
EECS 2030: Lab 2

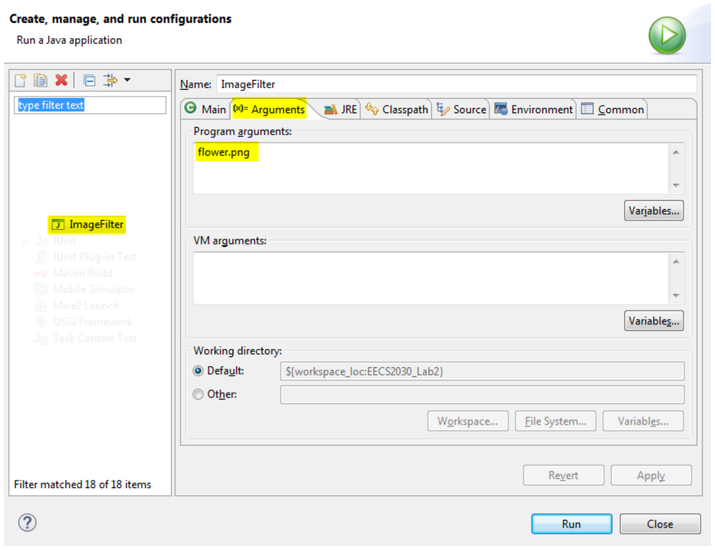
(1.5 % of the final grade, may be done in groups of up to three students)

Introduction

For this assignment you will *finish implementing* a command-line application that takes one argument – an image file name (JPG, PNG, or GIF files can be used), reads the image from the specified file and processes it as soon as the user clicks a button in the application window, performing one of the two operations: blurring or sharpening. All operations are based on convolving images with proper kernels[1].

To run your code from the command-line during the development, do the following changes to your Run Configurations:





Name: ImageFilter

Program arguments: flower.png

Working directory:
Default: ${workspace_loc:EECS2030_Lab2}
Other:

Currently, the `main` methods always loads the **flower.png** image.

Please leave your class in the default package.

Blurring

Blurring is based on Gaussian Blur[2]. Gaussian kernel, based on a mathematical Gaussian bell-shaped function, has a useful property of *separability*: it is possible to replace a more lengthy 2D convolution with a faster sequence of two 1D convolutions – one per dimension. A sample way of calculating Gaussian kernel coefficients is available here[3].

As an example,

- for a kernel [0.06136 0.24477 0.38774 0.24477 0.06136]
- step 1: convolution in the X dimension. The value of a pixel in column J, *pixel[J]* is

```
pixel[J] =      pixel [J−2] * 0.06136 +
                pixel [J−1] * 0.24477 +
                pixel [J−0] * 0.38774 +
                pixel [J+1] * 0.24477 +
                pixel [J+2] * 0.06136
```

1 https://en.wikipedia.org/wiki/Kernel_(image_processing)
2 https://en.wikipedia.org/wiki/Gaussian_blur
3 http://dev.theomader.com/gaussian-kernel-calculator/

In case a pixel coordinate is outside of the image boundaries, replace the index with either 0 or the image size−1 (whichever appropriate). Note that both input and output pixels are in the same row.
- step 2: starting with the result from step 1, do the same in the Y dimension.

For your application, you will have to write a method
`private void blur (int[] imageData, int width, double sigma)`
Based on the sigma value, first you will have to generate a proper 1D kernel of a small size (you can assume sigma value will not exceed 5), and then apply the convolution to the image data supplied. For testing purposes, the code contains three 1D kernels, for sigma values of 1.0, 3.0, and 5.0.
Note that you will probably need to use temporary arrays for intermediate steps.

Sharpening
For sharpening, your will have to implement the following method
`private void sharpen(int[] imageData, int width)`
Feel free to use any kernel you can find; you can hard-code it in your Java code. Link [1] contains one possible example of a sharpening kernel. Note that kernels for sharpening are (mostly) 2D kernels, and you will have to perform a 2D convolution (the process is not that different from the 1D case, but it will probably run slower).

Image Data Internal Format
The images are read for you into a `BufferedImage` object (called `image)`. Among many things, `BufferedImage` allows one to query image parameters, such as width and height, as well as reading it into a 1D array of `int`'s (in the starter code – `rgbData`).

The data for every pixel has the following format (you can assume that all images will use 24-bit colour, with 8 bits per channel): **0x__RRGGBB**[4], where RR, GG, BB are red, green, and blue bytes respectively. The 8 most significant bits are not used. The individual colour components can be extracted using bitwise operations, as you will notice in the code fragments (look inside `sharpen`).

Remember that you can always calculate the image pixels addresses in the 1D array using the following procedure:
```
image1DIndex = imageRowIndex * image.getWidth() +  imageColumnIndex
```

Starter Code
You are expected to work with the starter code supplied (find the TODO sections; there are three of them at the time of writing). Feel free to create extra private methods if needed. You may modify the code as you see fit – as long as the source and the result images are shown in the same window at the end.

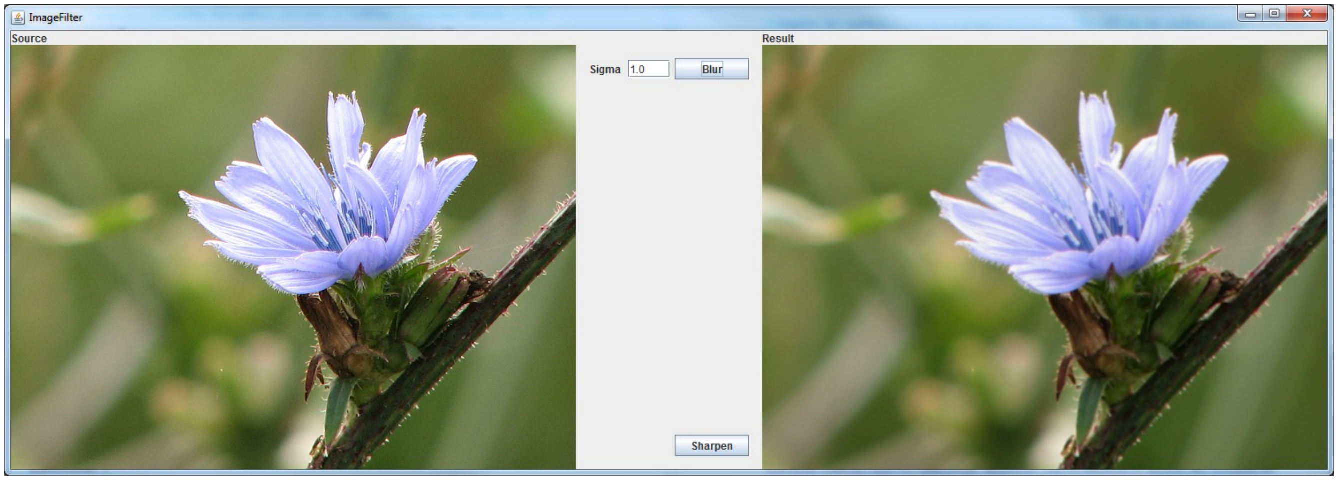
The application displays its output to the right of the original image window. For simplicity, the application does not display larger images properly (no scaling is possible). Your methods, however, should not assume any limits for image sizes. E.g., if you need to create a temporary array in your code, do not assume the images are no larger than N×M, where N

4 0x indicates hexadecimal representation, where each symbol represents 4 bits, and every pair of symbols represents one byte

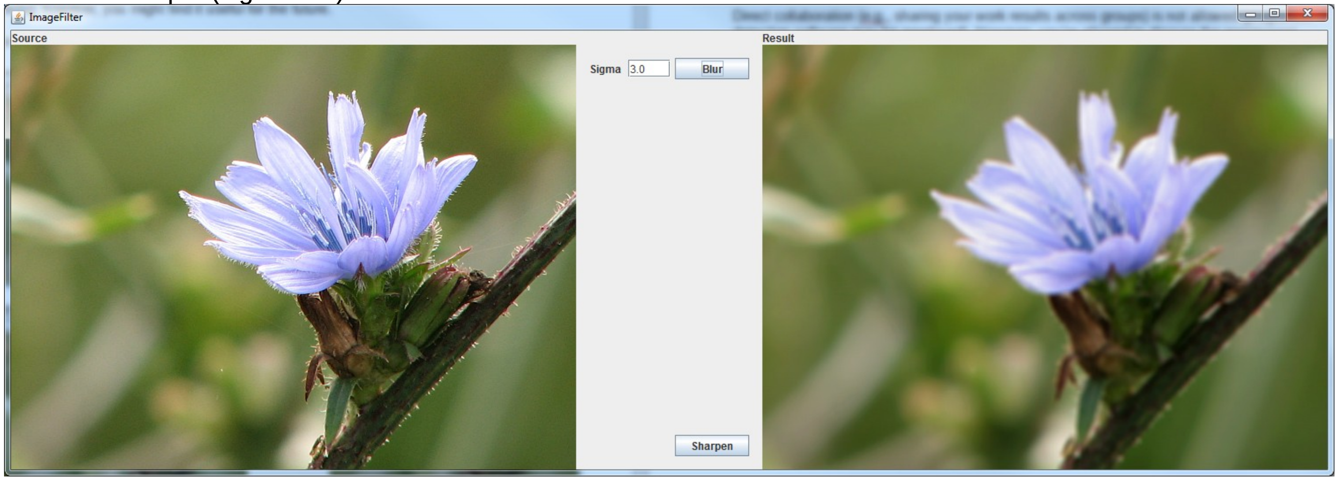
and M are “huge” numbers. Instead, read the image dimensions from the image object (or equivalent) and create that temporary array of the *exact* size required.

You *don’t have to* understand the Graphical User Interface (GUI) code written using Java Swing, however, you might find it useful for the future.

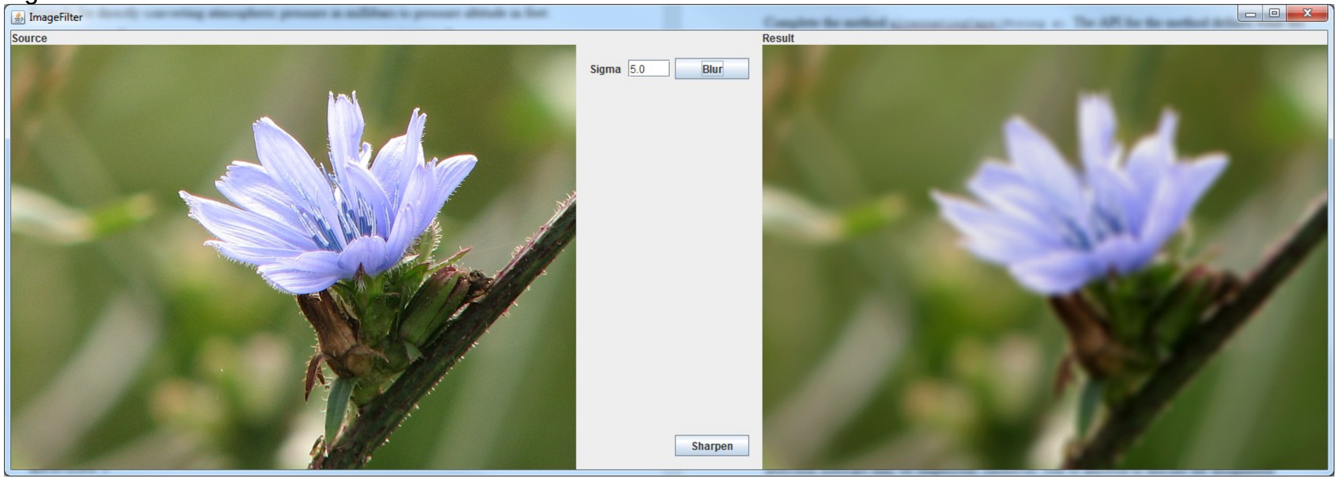
An example input and the output the application produces for bluring (sigma = 1):



another example (sigma = 3):



sigma = 5:



Grading

The assignment will be graded using *the Common Grading Scheme for Undergraduate Faculties*[5]. We look at whether the code passes the unit tests, and whether it conforms to the code style rules.

Submission:

You have to submit to Moodle the following files (and no more):

1. `ImageFilter.java` (your code)

2. `group.txt` (optional, contains student numbers and names of the students in the group)

Academic Honesty

Direct collaboration (e.g., sharing your work results across groups) is not allowed (plagiarism detection software may be employed). However, you're allowed to discuss the assignment requirements, approaches you take, etc. Also, make sure to state any sources you use (online sources – including old solutions, books, etc.). Although using outside sources is allowed – with proper citing, if the amount of non-original work is excessive, your grade may be reduced.

5 https://secretariat-policies.info.yorku.ca/policies/common-grading-scheme-for-undergraduate-faculties/