Assignment 5: Image Processing (Part 2)

Due Thursday by 9pm **Points** 0

MIME: More Image Manipulation and Enhancement (Part 2)

In part 2 of this series, you will some new image processing operations in your application and add support for more conventional file formats. Since your program already has an MVC design, each feature will be implemented "end-to-end" (i.e. implement the feature in model and expose it to the user through the view and controller).

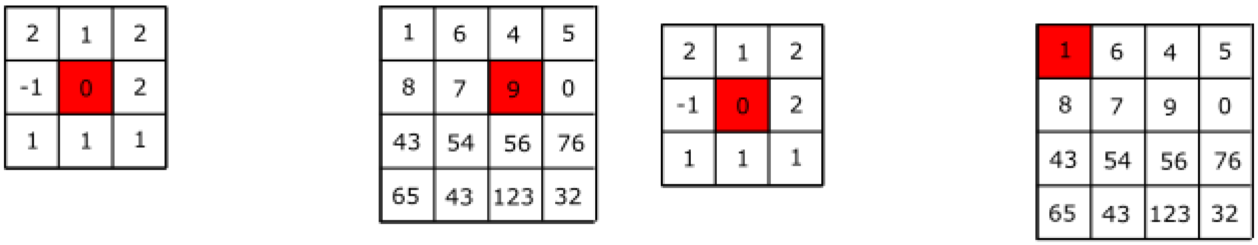
1 Support for conventional file formats

Although PPM is a simple and standard image format, it is not very popular. In order to make your application more usable, you will add support to import and export images in popular file formats, such as JPEG and PNG. Look at the ImageIO class (https://docs.oracle.com/javase/7/docs/api/javax/imageio/ImageIO.html) in Java to see how one can read and write images of many popular file formats. Find a way to incorporate this into your application.

2 Filtering

A basic operation in many image processing algorithms is filtering. A filter has a "kernel," which is a 2D array of numbers, having odd dimensions (3x3, 5x5, etc.). Given a pixel in the image and a channel, the result of the filter can be computed for that pixel and channel.

As an example, consider a 3x3 kernel being applied for the pixel at (5,4) of the red channel of an image(row 5, column 4). We place the center of the kernel at the particular pixel (e.g. kernel[1][1] overlaps pixel (5,4) in channel 0. This aligns each number in the kernel with a corresponding number in that channel (kernel[0][0] aligns with pixel (4,3), kernel[1][2] aligns with pixel(5,5), etc.). The result of the filter is calculated by multiplying together corresponding numbers in the kernel and the pixels and adding them. If some portions of the kernel do not overlap any pixels, those pixels are not included in the computation. The picture below illustrates the filter operation (not using the same numbers as above).



2*6 + 1*4 + 2*5 + (-1)*7 + 0*9 + 2*0 + 1*54 + 1*56 + 1*76

0*1 + 2*6+ 1*8 + 1*7

As with earlier image operations, one may have to clamp the resulting values in the range $[0, 255]$ after filtering.

2.1 Applications of filtering

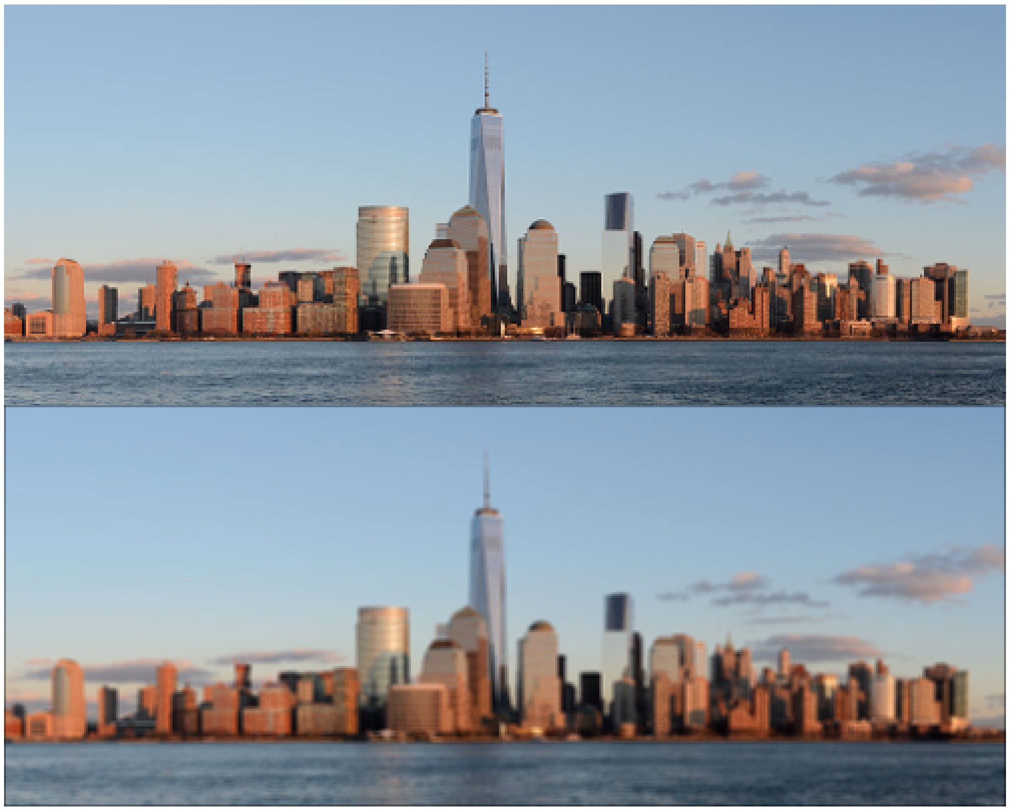
Many image processing operations can be framed in terms of filtering (i.e. they are a matter of designing an appropriate kernel and filtering the image with it).

2.1.1 Image blur

Blurring an image is probably the simplest example of filtering. We can use a 3x3 filter as follows:

$$\begin{bmatrix} \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \\ \frac{1}{8} & \frac{1}{4} & \frac{1}{8} \\ \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \end{bmatrix}$$

This is called a "Gaussian" blur (imagine the values to be heights of a 3D bar graph to see how they form a coarse bell-like Gaussian surface). Blurring can be done by applying this filter to every channel of every pixel to produce the output image. Here is an example of blurring an image this way (in order: original image, blurred, blurred again):



)

The filter can be repeatedly applied to an image to blur it further. The last image above shows the result of applying the same filter to the second image.

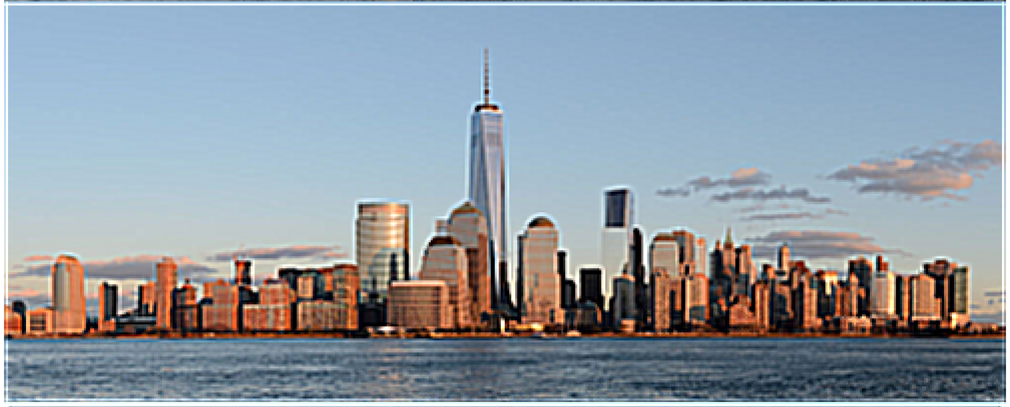
2.1.2 Image sharpening

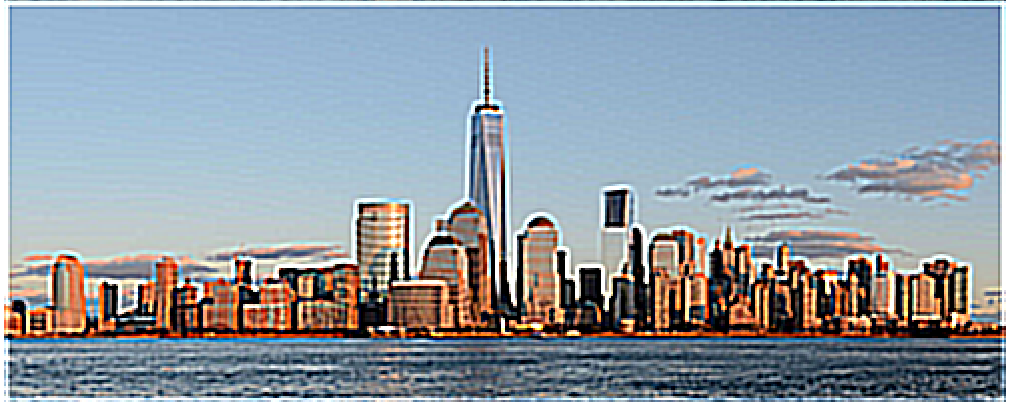
Image sharpening is in some ways, the opposite of blurring. Sharpening accentuates edges (the boundaries between regions of high contrast), thereby giving the image a "sharper" look.

Sharpening is done by using the following filter:

$$\begin{bmatrix} \frac{-1}{8} & \frac{-1}{8} & \frac{-1}{8} & \frac{-1}{8} & \frac{-1}{8} \\ \frac{-1}{8} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{-1}{8} \\ \frac{-1}{8} & \frac{1}{4} & 1 & \frac{1}{4} & \frac{-1}{8} \\ \frac{-1}{8} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{-1}{8} \\ \frac{-1}{8} & \frac{-1}{8} & \frac{-1}{8} & \frac{-1}{8} & \frac{-1}{8} \end{bmatrix}$$

As with blurring, it is possible to repeatedly sharpen the image. Here is an example of sharpening an image (in order: original image, sharpened, sharpened more):





3 Color transformations

Filtering modifies the value of a pixel depending on the values of its neighbors. Filtering is applied separately to each channel. In contrast, a color transformation modifies the color of a pixel based on its own color. Consider a pixel with color (r,g,b). A color transformation results in the new color of this pixel to be (r',g',b') such that each of them are dependent only on the values (r,g,b).

A simple example of a color transformation is a linear color transformation. A linear color transformation is simply a color transformation in which the final red, green and blue values of a pixel are linear combinations of its initial red, green and blue values. For example, if the initial color is (r,g,b), then the final red value being $0.3r + 0.4g + 0.6b$ is a linear color transformation.

Linear color transformations can be succinctly represented in matrix form as follows:

$$\begin{bmatrix} r' \\ g' \\ b' \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} * \begin{bmatrix} r \\ g \\ b \end{bmatrix}$$

where a_{ij} are numbers. Using the above notation, the final values will be:

$$r' = a_{11}r + a_{12}g + a_{13}b \, g' = a_{21}r + a_{22}g + a_{23}b \, b' = a_{31}r + a_{32}g + a_{33}b$$

Similar to filtering, clamping may be necessary after a color transformation to save and display an image properly.

3.1 Applications of color transformations

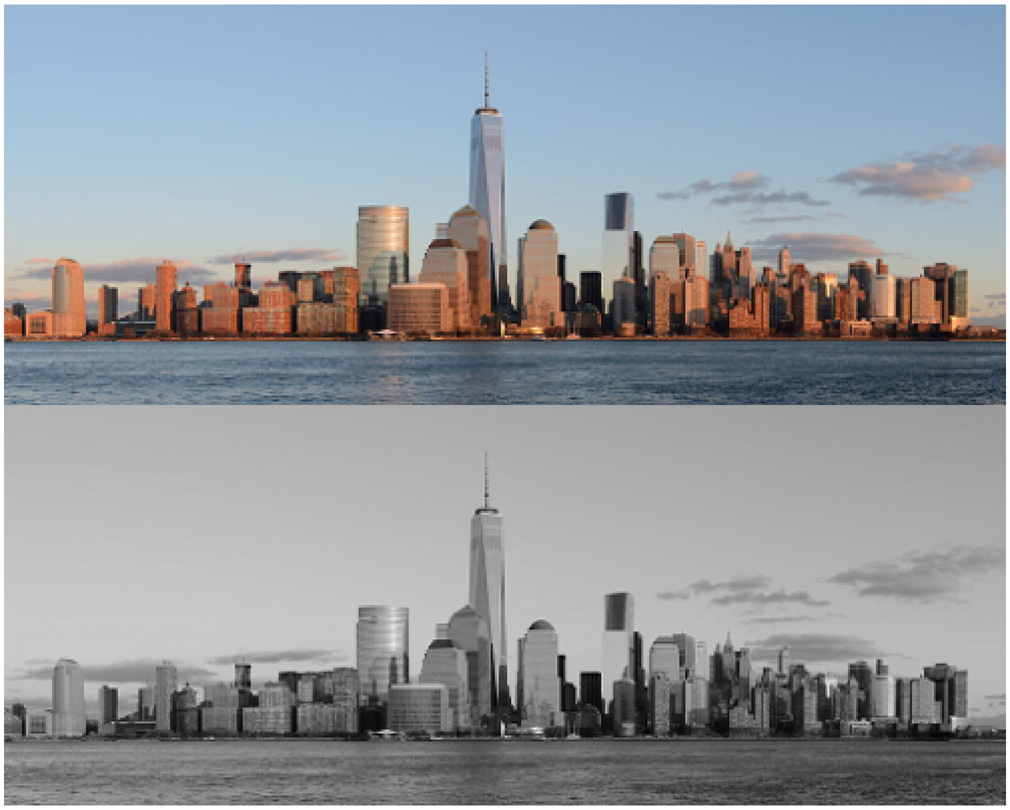
There are many image processing operations that can be expressed as color transformations on individual pixels (i.e. implementing them is a matter of using the correct numbers a_{ij} on all pixels of the image).

3.1.1 Greyscale

A simple operation is to convert a color image into a greyscale image. A greyscale image is composed only of shades of grey (if the red, green and blue values are the same, it is a shade of grey). There are many ways to convert a color image into greyscale. A simple way is to use the following color transformation: $r' = g' = b' = 0.2126r + 0.7152g + 0.0722b$. Recall that this is the "luma" component from the previous assignment. Luma is useful for (high-definition television (https://en.wikipedia.org/wiki/Rec._709)). This can be expressed as a color transformation in matrix form as:

$$\begin{bmatrix} r' \\ g' \\ b' \end{bmatrix} = \begin{bmatrix} 0.2126 & 0.7152 & 0.0722 \\ 0.2126 & 0.7152 & 0.0722 \\ 0.2126 & 0.7152 & 0.0722 \end{bmatrix} * \begin{bmatrix} r \\ g \\ b \end{bmatrix}$$

The effect of this conversion is illustrated below:



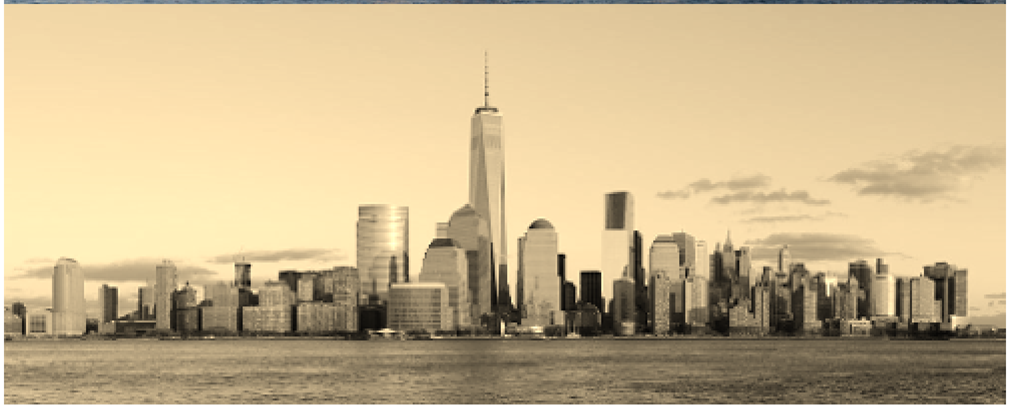
Can other existing operations be implemented as color transformations similarly?

3.1.2 Sepia tone

Photographs taken in the 19th and early 20th century had a characteristic reddish brown tone. This is referred to as a sepia tone (see the origin of this term (https://en.wikipedia.org/wiki/Sepia_(color))). Most image processing programs allow an operation to convert a normal color image into a sepia-toned image. This conversion can be done using the following color transformation:

$$\begin{bmatrix} r' \\ g' \\ b' \end{bmatrix} = \begin{bmatrix} 0.393 & 0.769 & 0.189 \\ 0.349 & 0.686 & 0.168 \\ 0.272 & 0.534 & 0.131 \end{bmatrix} * \begin{bmatrix} r \\ g \\ b \end{bmatrix}$$

The effect of this conversion is illustrated below (original image, sepia-toned result):



4 What to do

In this assignment, you must enhance your application to:

- Support the ability to load and save conventional file formats (bmp, jpg and png) in addition to ASCII ppm files from before.
- Support the ability to load the save images in formats using the same script command as before. The extension of the file determines the format to be used.
- Support the ability to blur and sharpen an image, keeping in mind that other filtering operations may be possible in future.
- Support the ability to produce a greyscaled and sepia version of an image using color transformations, keeping in mind that other color transformations may be possible in future.
- Support the ability to specify the above operations using script commands: blur, sharpen, greyscale, sepia with the same set of parameters (source image name, destination image name).
- Support the ability to accept a script file as a command-line option. For example "-file name-of-script.txt". If a valid file is provided, the program should run the script and exit. If the program is run without any command line options, then it should allow interactive entry of script commands as before.
- Retain support for the PPM file format, and being able to switch between them (e.g. import a PPM file, work on it and save it as a JPEG file, etc.)

- Retain support for all previous operations and script commands.

5 Design Considerations

As you add these features, pay attention to your design.

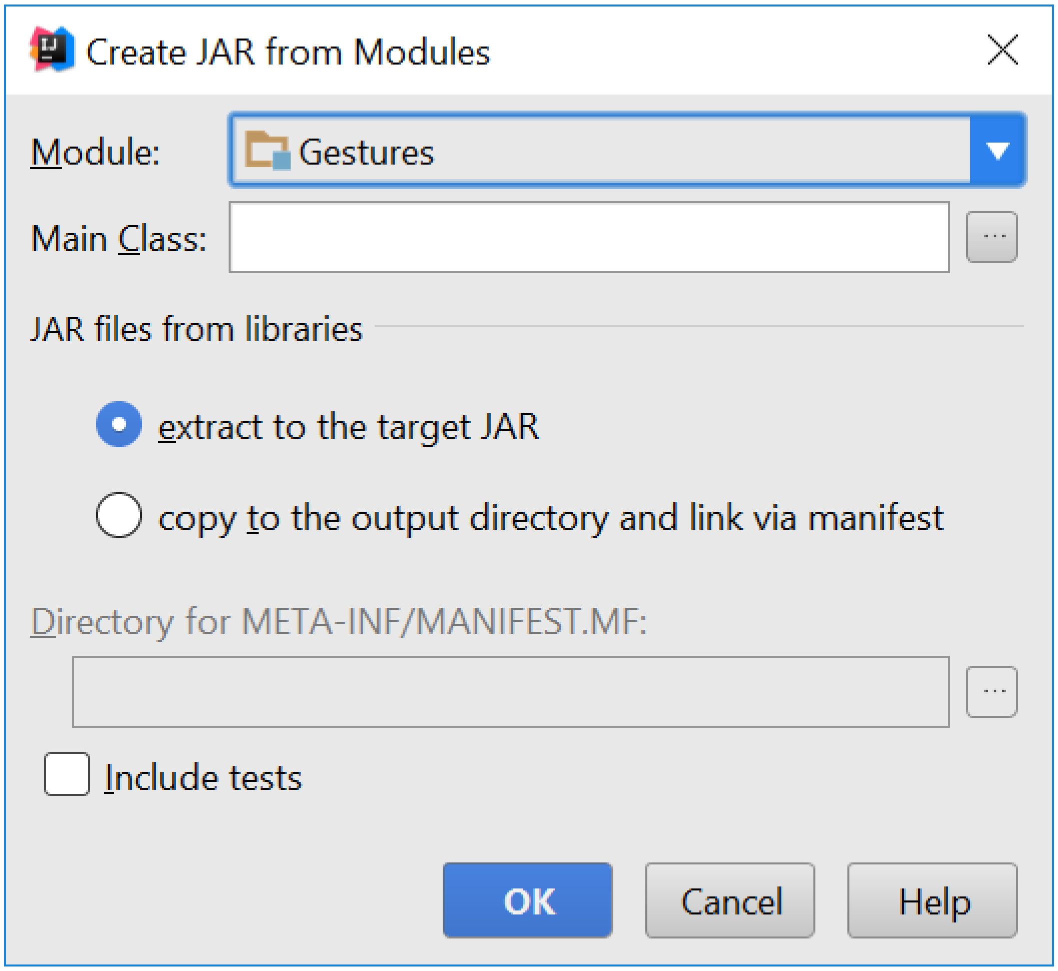
- What changes in your existing design do you make, and why?
- What is the best way to incorporate new features into an existing application? Do you support all features, or do you release incremental versions with different features (e.g. Starter Edition, Pro Edition, etc.)?
- As you make changes, are you still adhering to the MVC architecture? Are you putting each class and operation where it belongs?
- Have your design enhancements gone beyond the specific operations to be implemented? How easy would it be to add similar extensions in the future?
- Whatever design choices you make, what are its advantages and limitations?

While you are allowed to change your design, you should be able to justify it. The bigger the change, the better we expect your justification to be. Please document and justify each change in a README file (maintain this file as you complete this assignment, rather than summarize after you are done and risk missing something).

6 Create a JAR file of your program

To create a JAR file, do the following:

- Go to File -> Project Structure -> Project Settings -> Artifacts
- Click on the plus sign
- Choose JAR -> From Modules with dependencies. You should now see the module in your project that you are working on (may be different than what is shown in the image below)


Create JAR from Modules

Module: Gesture

Main Class:

JAR files from libraries

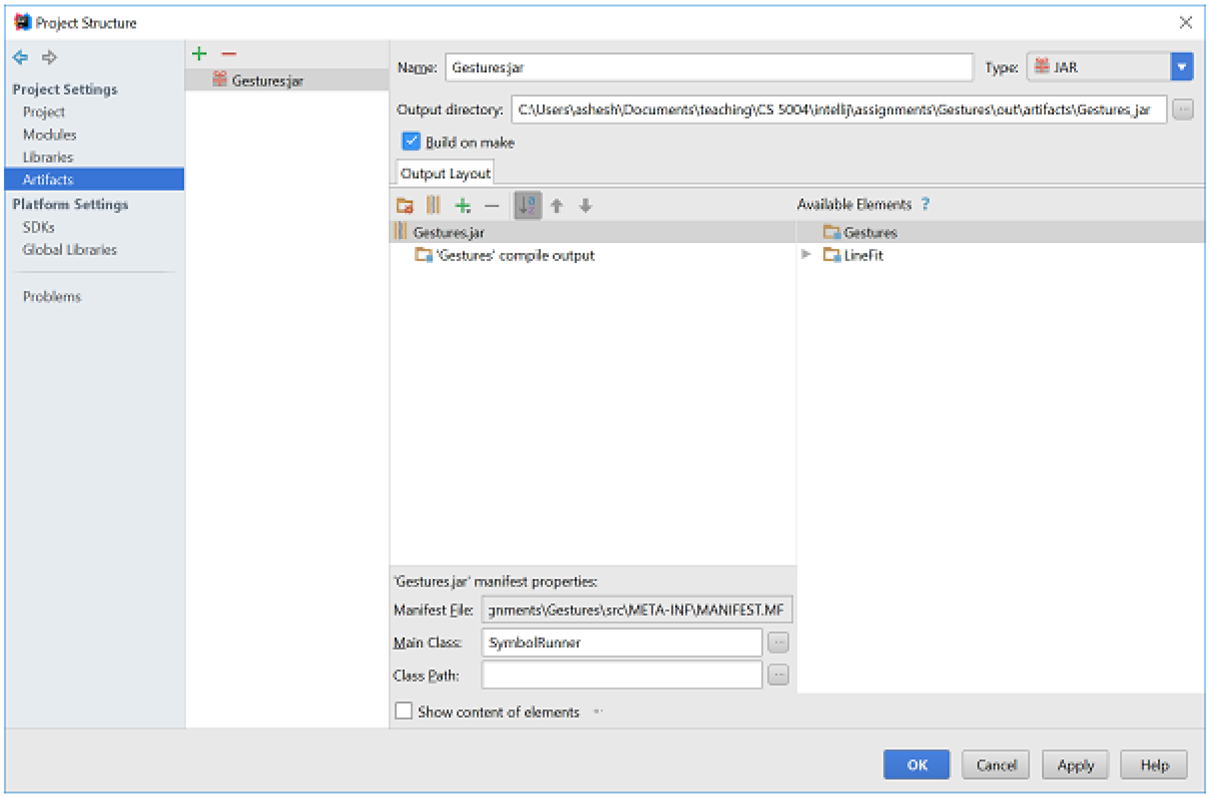
[✓ extract to the target JAR]

copy to the output directory and link via manifest

Directory for META-INF/MANIFEST.MF:

Include tests

- Select the main class of your program (where you defined the `main(String[] args)` method)
- If you see a checkbox labelled "Build on make", check it.
- Hit ok
- You should now see something like



Name: Gestures:jar
Type: JAR
Output directory: C:\Users\ashesh\Documents\teaching\CS 5004\intellij\assignments\Gestures\out\artifacts\Gestures_jar
☑ Build on make
'Gestures:jar' manifest properties:
Manifest File: gnments\Gestures\src\META-INF\MANIFEST.MF
Main Class: SymbolRunner
Class Path:
☐ Show content of elements

If now you see a checkbox labelled "Build on make", check it now.

- Make your project (the button to the left of the run configurations dropdown, with the ones and zeros and a down-arrow on it). Your .jar file should now be in /out/artifacts/.
- **Verify that your jar file works** . To do this, copy the jar file to another folder. Now open a command-prompt/terminal and navigate to that folder. Now type java -jar NameOfJARFile.jar and press ENTER. The program should behave accordingly. If instead you get errors review the above procedure to create the JAR file correctly. **You can also run the jar file by double-clicking on it (it will run without command-line arguments).**

7 What to submit

Please verify that you submit everything below, as each part is worth some points in this assignment.

A complete submission must include:

1. All your code in the src/ folder.
2. All your tests in the test/ folder.
3. At least one example source image in png/jpg/bmp/ppm format, its blurred, sharpened, greyscaled and sepia-ed versions in a res/ folder
4. At least one example of a script file that shows all the working features of your application in the res/ folder. If a command is missing from your example file, we will assume that it does not work and

grade accordingly. Your script file should run as-is when we run your JAR file with the command-line option, so please write and place it accordingly.

5. A correct JAR file in the res/ folder. We should be able to run your program from this jar file.
6. An updated class diagram in the res/ folder, which shows names of classes and interfaces, method signatures and inheritance relationships.
7. A README file in the root submission folder (that contains src, test and res). The README file should document which parts of the program are complete, and design changes and justifications.
8. A USEME file in the root submission folder (that contains src, test and res). The USEME file should summarize which script commands are supported by your application, examples of using them and conditions if any (e.g. X command should be typed before Y, etc.)