Assignment 6: Image Processing (Part 3)

Due Nov 22, 2022 by 9pm **Points** 0

GRIME: Graphical Image Manipulation and Enhancement (Part 3)

Things you can do with Swing: **code (https://northeastern.instructure.com/courses/119265/files/16309885/download)**

1 Additional Features

In this iteration of this project, you will build a view for your image processing application, featuring a graphical user interface. This will allow a user to interactively load, process and save images. The result of this iteration will be a program that a user can interact with, as well as use batch scripting from the previous iteration.

We will also add one feature to our program: the ability to create and visualize histograms for an image.

2 Image Histograms

A histogram is a table of (value,frequency) entries. Consider a greyscale image storing 256 distinct greyscale values (i.e. uses 8-bits per pixel). One can count the number of pixels with value 0, 1, ...255 and create a table of 256 entries. This would be a histogram of the image showing 256 distinct levels. If the image has color, we create histograms for each component as well as the intensity (the average of all components).

Histograms have several applications. One of the simplest is to visualize the distribution of color or intensities in an image. This can be done by plotting histograms using line or bar charts. Specifically for a histogram with 256 entries, one can draw 256 vertical bars with heights proportional to the frequencies. Alternatively one can plot (value,frequency) points and draw lines joining adjacent points, creating a line chart. For color images, we would have 4 sets of bars/lines in different colors.

Histograms may be used for general image manipulation as well. For example, if a histogram is concentrated near the left edge (i.e. most values are small) then the underlying image is dark. One can "stretch" the histogram by stretching the value range of this histogram to the [0-255] range. This amounts to scaling each value in the image proportionally while maintaining their relative distribution, thereby brightening the image. In general, a user can look at the histogram to create a function $f(x)$ that maps values to other values within the same [0-255] range. For example $f(x) = x$ preserves the values. $f(x) = min(255, 0.01x^2)$ is a parabolic function that is clamped to 255, which has the effect of suppressing values below 100 and amplifying others. All of these are examples of "levels adjustment", a fairly standard operation in programs like Photoshop.

In this assignment you are required to create a histogram and display it (see below). However as explained above, histograms may be useful beyond simply display.

3 Graphical View

In this iteration you will build an interactive view with a graphical user interface.

3.1 Graphical user interface

Build a graphical user interface for your program. While the choices about layout and behavior are up to you, your graphical user interface should have the following characteristics, and obey the following constraints:

1. You must use Java Swing to build your graphical user interface. Besides the code examples from lecture, the provided code example illustrates some other features of Swing that you may find useful.
2. The GUI should show the image that is currently being worked on. The image may be bigger than the area allocated to it in your graphical user interface. In this case, the user should be able to scroll the image. Any changes to the current image as a result of the image operations should be visible in the GUI.
3. The histogram of the visible image should be visible as a line chart on the screen at all times. If the image is manipulated, the histogram should automatically refresh. The histogram should show the red, green, blue and intensity components.
4. The user interface must expose all the required features of the program (flips, component visualization, greyscale, blurring, sharpening and sepia). You are not required to support interactive scripting through the GUI.
5. When saving an image as a PNG/PPM/JPG, it should save what the user is currently seeing.
6. The user should be able to specify suitably the image to be loaded and saved that the user is going to process. That is, the program cannot assume a hardcoded file or folder to load and save.
7. Any error conditions should be suitably displayed to the user, through pop-up messages or clearly visible text as appropriate.
8. Each user interaction or user input must be reasonably user-friendly (e.g. making the user type the path to a file is poor UI design). We do not expect snazzy, sophisticated user-friendly programs. Our standard is: can a user unfamiliar with your code and technical documentation operate the program correctly **without reading your code and technical documentation?**

3.2 View and controller

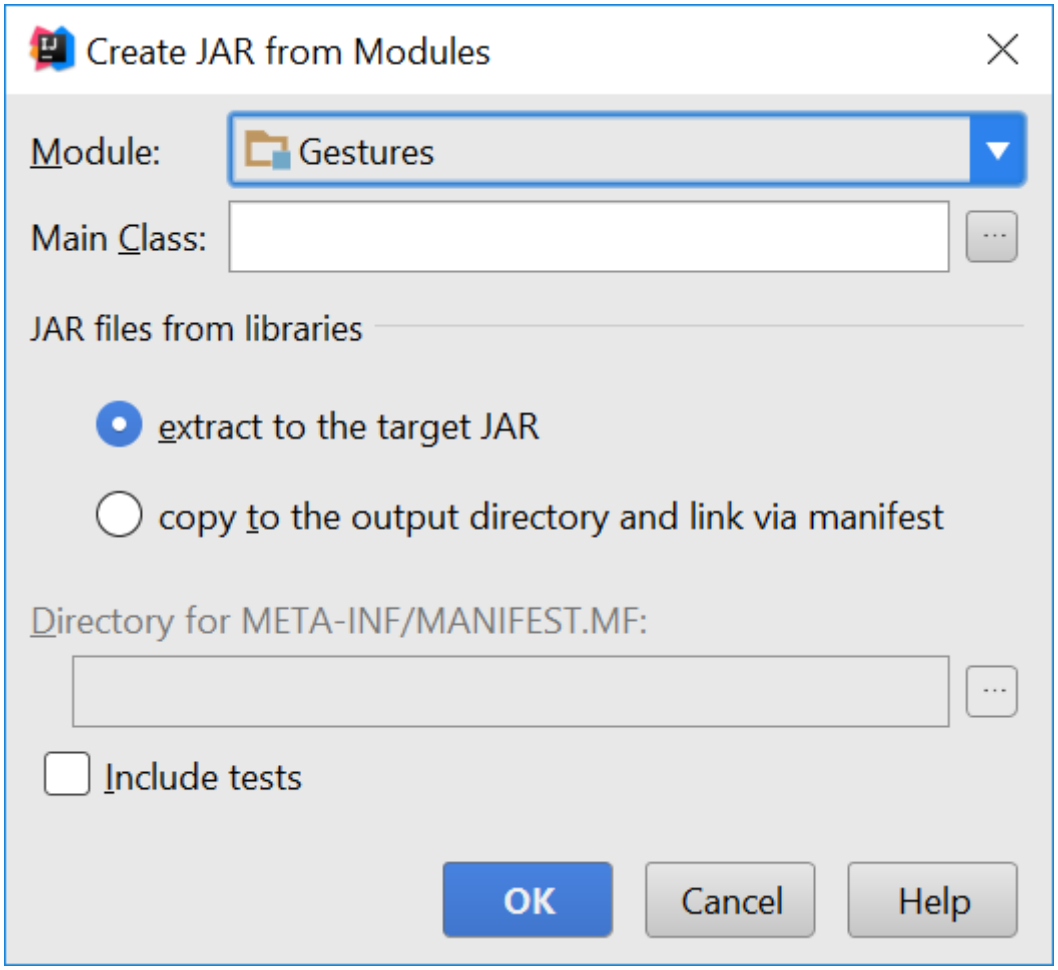
Carefully design the interaction between a view and a controller, and formalize the interactions with view and controller interfaces. You may design a single controller that manages the program in script mode and GUI mode. Different controllers for different views are also possible if the views are very different

from each other. However be mindful of the MVC principles and separation between the model, view and controller. When designing, always ask: "can I change one part with no/minimal changes to the others?"

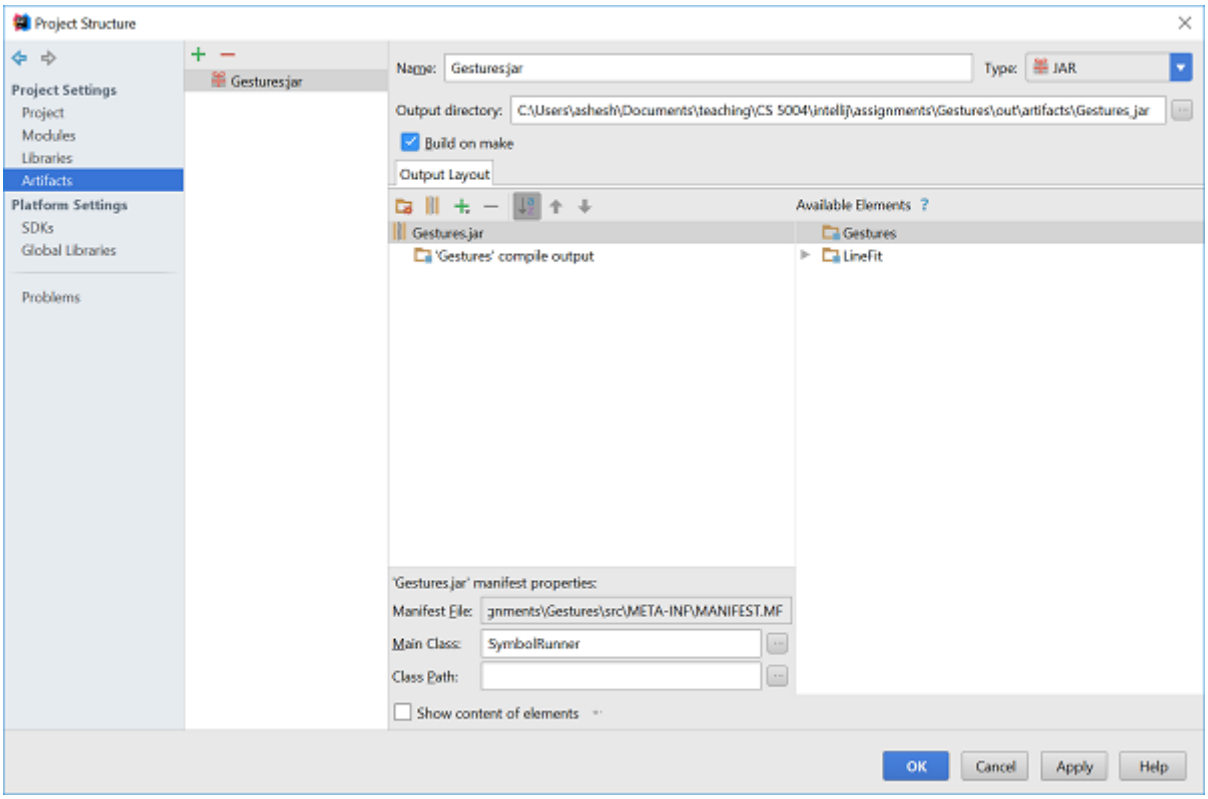
Create a JAR file of your program

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)



- 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



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.**

4 Command-line arguments

Your program (from IntelliJ or the JAR file) should accept command-line inputs. Three command-line inputs are valid:

- `java -jar Program.jar -file path-of-script-file` : when invoked in this manner the program should open the script file, execute it and then shut down.
- `java -jar Program.jar -text` : when invoked in this manner the program should open in an interactive text mode, allowing the user to type the script and execute it one line at a time. This is how the program worked in the last assignment.
- `java -jar Program.jar` : when invoked in this manner the program should open the graphical user interface. This is what will happen if you simply double-click on the jar file.

Any other command-line arguments are invalid: in these cases the program should display an error message suitably and quit.

Criteria for grading

You will be graded on:

1. The completeness, layout and behavior of your graphical user interface.
2. Your design (interfaces, classes, method signatures in them, etc.)
3. Whether your code looks well-structured and clean (i.e. not unnecessarily complicating things, or using unwieldy logic).
4. Correctness of your implementations, evidenced in part by the images you submit.
5. Whether you have written enough comments for your classes and methods, and whether they are in proper Javadoc style.
6. Whether you have used access modifiers correctly: `public` and `private`.
7. Whether your code is formatted correctly (according to the style grader).

What to submit

Your zip file should contain three folders: src, test and res (even if empty).

1. All your code should be in src/.
2. All your tests should be test/.
3. Submit a correct JAR file in the res/ folder. We should be able to run your program from this jar file.
4. **Submit a screen shot of your program with an image loaded.** You do not need to submit example images, but submitting a sample image to try out your program is required.
5. Submit a README file that documents how to use your program, which parts of the program are complete, design changes and justifications, and citations for all images.
6. A USEME file that contains a bullet-point list of how to use your GUI to use each operation supported by your program. Screenshots would be helpful, but not necessary.