Assignment 8 C212 FA22

Last modified 2022-11-14 11:40 PM

Learning Objectives

After completing this assignment, students will be proficient in developing sophisticated programs with GUI elements in Java. Starting with a problem statement and an existing codebase, they will be able to adapt existing solutions into a broader project. They will also be able to use tests to demonstrate the quality and behavior of their work.

Summary

The gist of this assignment document's contents are as follows:

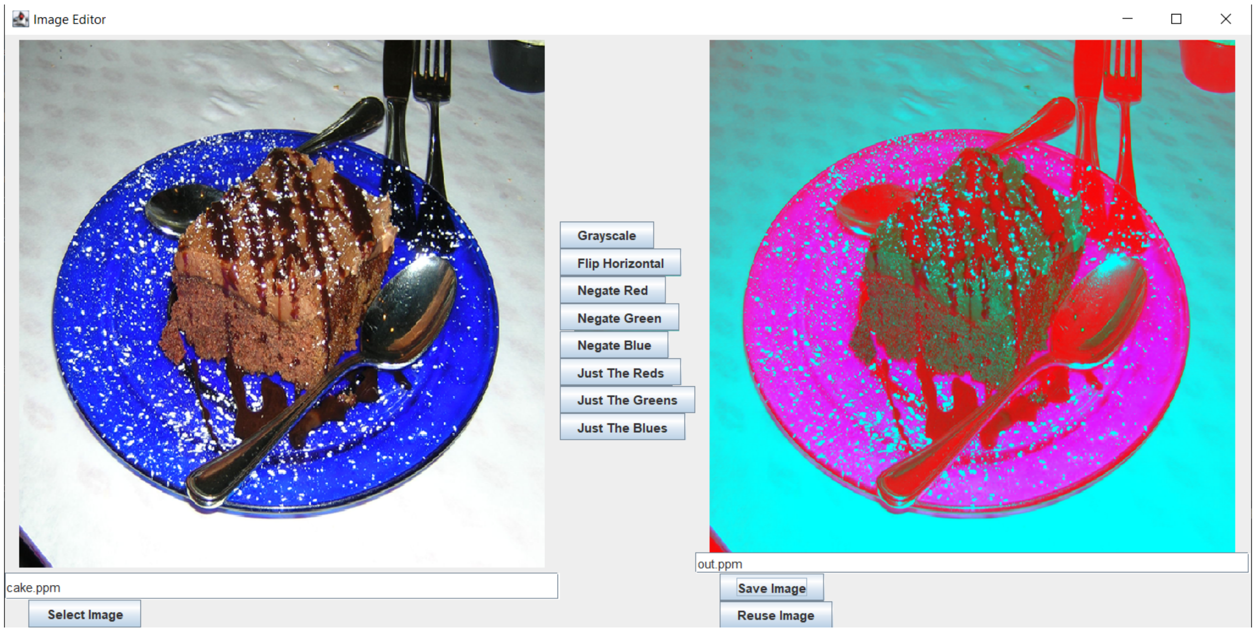
- Program Description - a GUI-based image editor similar to A5
- Your Tasks
- Technologies We Use
  - TwelveMonkeys ImageIO plugin for PPM file support
  - Maven build automation and dependency manager
  - A little bit of MVC
- Rubric
- Changelog
  - No changes yet…

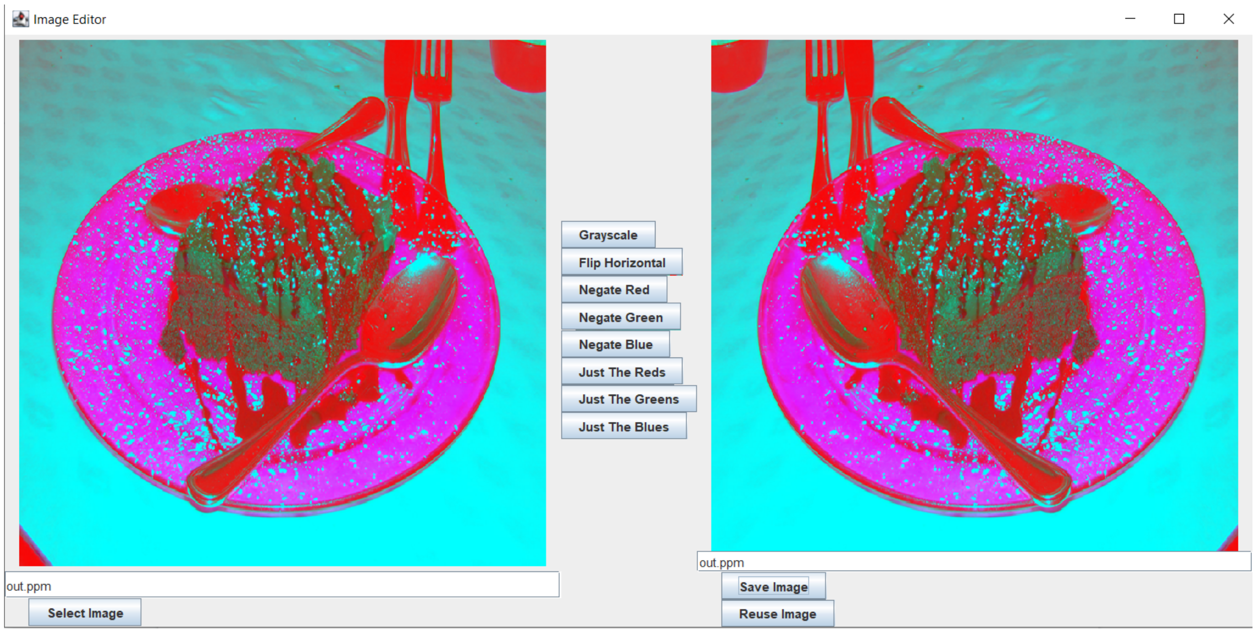
Program Description

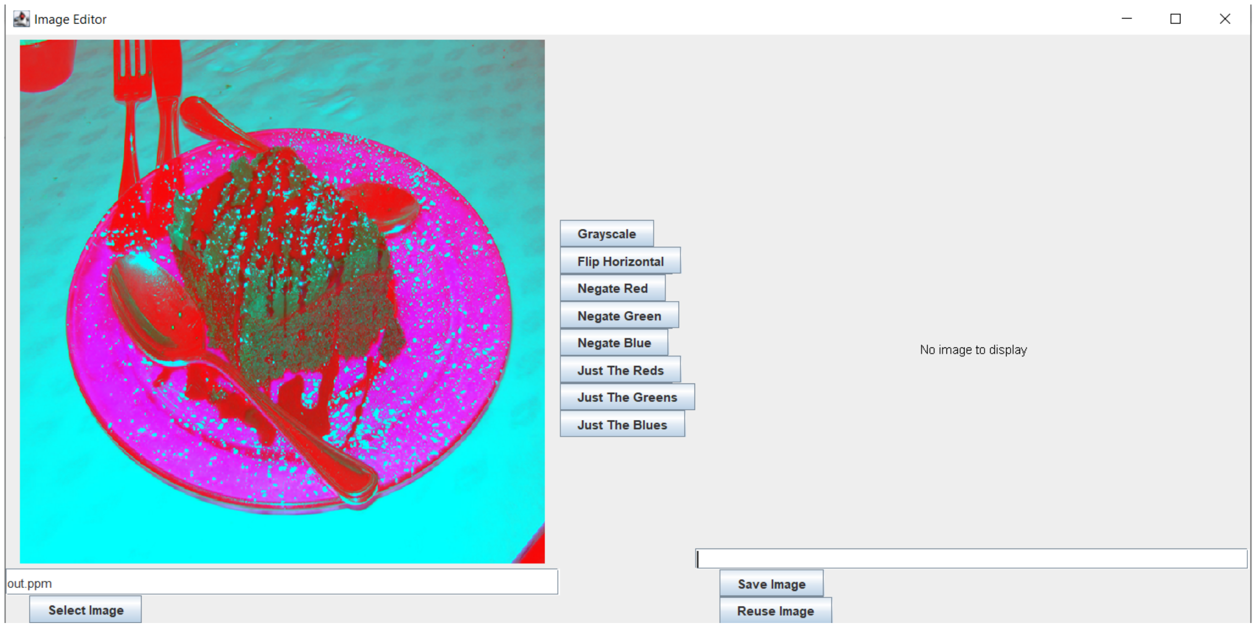
This assignment asks you to create a GUI-based application for image editing. It will be reminiscent of the project produced in Assignment 5, but with a better user experience through the use of a Graphical User Interface – particularly, the Swing framework in Java.

You've already written the logic to handle representation and transformation of images in the PPM format. You'd be welcome to reuse this, however we will provide a working version of the methods that read, write, and modify PPM images via representing them as a 3D array of color values at each pixel.

Your job is to build the front-end GUI for this application, based on some starter code, and connect the functionality of its interface elements to the behind-the-scenes logic. In the finished product, the user will be able to choose an image, apply a transformation, save that image, and repeat the process in an easy, visualizable way.







Your Tasks

You're given the project starter file which contains lots of starter code. In particular, you'll notice a skeletonized frontend class that (will eventually) make the JFrame and all its components, no event handlers defined yet, a custom JComponent subclass that appears to draw images but needs some work, a helper class with methods for managing the GUI elements and files which needs some work too, and a helper class with methods that deal with the specifics of the PPM file format. **You have to use the starter project** – read Technologies We Use if you want to know why.

For the most part, the stuff you need to do is listed in TODOs in the code. You should read through all the files before you work to get a rough idea of what's going on and what you need to implement. We would recommend working on this one file-by-file, going in this order:

1. DrawnImageComponent (very little work to do)
2. EditorHelperMethods (little work to do)
3. ImageEditorDisplay (most of the work is here)

Believe it or not, there's not too much work to do overall, though. Excluding a big if-else if-else and the inner static classes for each event handler, there are not that many lines of code left to write, and the lines left to write are not that complicated. The challenge with this assignment is working with the starter code that was given to you – understanding how it works and how to build off of it. There are a LOT of moving pieces that get this app working, and as is very common in the professional world, you're not writing it all by yourself from scratch, but rather using something that is already made.

Another challenge is that you can't really see the progress you're making until near the end when you have the program able to be run. It's also tricky to write meaningful test cases for Swing stuff based on what you know about JUnit, so it might take some trial and error to get to a point where you could fix bugs that could've arisen in DrawnImageComponent or EditorHelperMethods. You can always comment out incomplete/buggy blocks of code if you just want to run and experiment with a certain part of the program.

Something to watch out for on this assignment is that rather than defining the components and event listener inner classes as static belonging to our runner class, we instead define certain elements at the scope they are needed and pass in references to them to those components which need it. For example, the input text field needs to be read in both the left, right, and even middle panels, so we scope it as low as possible (the getFrame() method), instantiate it there, then **pass it in as a reference** to those panels' maker methods so they can use .getText() on it as needed. For another example, the output text field is only used by the right panel, so it can be safely defined only in the right panel method. Compared to declaring everything statically, this would be considered better practice, as we want to control the scope of our members on a need-to-know basis, so to say. You should not deviate from this approach in your solution – in fact, most of the considerations have been made for you as each method header present is fully complete and includes the needed parameters.

You don't need to add any new methods, except the new constructor in DrawnImageComponent and the event handler inner classes' actionPerformed methods. Every method, every variable, etc. that's there is meant to be used. Remember to not hardcode values. If a class has constants, you should be using them, even in other classes (which you can do because they are static members)!

For this assignment, you do not need to write any test cases, and frankly it's not recommended you try. Normally when making an application like this you'd be writing unit tests for the backend, which in this case is the code that deals with the intricacies of PPM files and reading/writing/copying/editing files, but that stuff is done for you here.

Technologies We Use

This section of the document is more about feeding your curiosity than the assignment itself, but we would encourage you to read it if you're at all interested in software development.

As you know, Java Swing is the framework we've been learning to create GUIs. So far, we've made simple interfaces using buttons, labels, frames, panels, etc. We've also made our own custom subclasses of Swing classes, like to add custom painting commands to components to create simple visuals with shapes and colors using the Graphics class.

The Graphics class we used to draw also can draw images based on the abstract Image class in Java. The common image formats like JPEG and PNG are supported, but more esoteric formats like PPM are not natively supported. Thankfully, however, other Java developers have put out open-source solutions to extend the functionality of Java's Image class to formats like PPM, and these modifications to the workings of Java are known as plugins. For this project, we use the TwelveMonkeys plugin for Java's ImageIO class that allows us to read in PPM-format images and display them in the GUI using Graphics's drawImage method.

Luckily, something called Maven lets us set up this dependency for use in our project really easily. In fact, if we had just given you the starter project file and not told you we were using TwelveMonkeys, you probably wouldn't even have known. That's one of the things that makes Maven great: it is a build automation tool, so it grabs the necessary dependencies for you from a central repository and adds them to your project, that way any code you run will be able to use it super easily.

Chances are, at this point in your careers as developers, you've never really had to work with these kinds of things, but as you get more into the profession you'll become more familiar and comfortable with using tools like dependency managers (e.g. you might already know pip from Python or npm from JavaScript) and existing open-source solutions and frameworks, and the value of incorporating them into the projects you make will become more clear.

It's also hard to talk about what makes a good GUI-based application without talking about Model-View-Controller, which is a design pattern we use to standardize the separation of concerns needed to realistically manage the complexity of applications with distinct frontends and backends. But… we won't today. You might begin to notice some patterns emerge in how this project is structured as you work on it, since it was designed with MVC at least partially in mind.

Rubric

A rubric is also available on the assignment page on Canvas, but here is an outline of the point breakdown for different categories.

- (10pts) Code cleanliness, conventions, and comments
  - (10/10pts) for >95% correctness and all required comments
  - (7.5/10pts) for repeated minor mistakes or one egregious mistake, or for missing either overall or inline comments
  - (5/10pts) for code that is sloppy and hard to understand or is missing all comments
  - (<=2.5/10pts) for code that has no comments, is hard to understand, and consistently does not follow Java conventions
- (5pts) for DrawnImageComponent
  - (5pts) for constructor which assigns argument to instance variable
- (20pts) for EditorHelperMethods
  - (5pts) for calling correct PPMHelperMethod
  - (5pts) for passing correct path argument to that method
  - (10pts) for big conditional structure that matches transformCode to correct PPMHelperMethods call
- (65pts) for ImageEditorDisplay
  - (12.5pts) for left panel
  - (20pts) for middle panel
  - (17.5pts) for right panel
  - (15pts) for frame, including scoping components correctly

Changelog

If any corrections, clarifications, etc. are made to the assignment after its release, they will be listed here. Any changes more substantial than simply fixing typos will get corresponding Canvas announcements. That way, you don't need to keep checking whether updates have occurred.

- 11/14 11:40pm
    - Rubric updates