This README is for an image processing program created by Henry deGrasse and Wanru Shao for Northeastern University's Object Oriented Design class. The image processing program works by loading in any image and altering the image and saving it as a new altered image.

This is the third version of our image processing project, which contains a model, controller and a view. As of now the user can use the program by running the main method in the ImageUtil class. The user can then input different commands depending on what they want to do and they can end the program by typing quit. They inputs must be the following or they won't be considered as valid.

———

The differences and Updates:
Updates on hw 6:

Changes for last design:
N/A since we got 100% for hw 5's assignment. As a result, we implemented the features for hw 6 directly.

HW 6:

For this third version of our Image processor, we added on and built a view for our image processing application, which featured a GUI. This addition will allow a more streamlined user experience by making it easier to interact with our program. The other addition to this version was the addition to create and display histograms for the image in the newly created GUI.

To create the Histogram we created an interface ImageHistogramDataGenerator and the subsequent ImageHistogramDataGeneratorImpl which goes through the image and gathers every value of the specific channel or intensity and creates a list with length until the maximum color value (256) with each value being the amount of times the specified color occurs at that value.

In order to get the image that we are working on without possibly changing the original we created a new interface named ImageModelState, which gets the current state of the image and saves it.

Then, in order to create the visual of the GUI an interface GUIView and a class GuiViewWIndow implementing it, this class will take care of doing actions on a given image, displaying the image, loading an image, saving the image, getting the brightness of the image and lastly refreshing the image.

To allow for the commands and user input to be processed, we also created a new controller class which implements ImageEditorCommands named GuiViewControllerImpl, this allows the users inputs on the GUI to be translated into

commands so the program understands what image processing method to use.

Updates on hw 5:

in this version is that our code should be able to accept multiple types of image formats, JPEG, PNG, BMP, and PPM. We can now alter images with a new filter, that filter being Sepia. There are two new image manipulations, Blur and Sharpen, which do what the name infers.

Changes for last design:
We only have two small issues from hw 4:

The changes in the design are as follows, we placed both the brighten and darken classes into one classes to avoid hardcoding it.

We also moved the I/O from model interface to the controller interface (Save and Load commands into the controller from the model.) because the controller should be one that deal with the files directly.

For hw 5:
We also added a color transform interface and implemented the Sepia color transformation there, and moved the Luma color transformation from the last version to there as well.

For the sharpening and blurring. We created a new interface named filtering, which has both Sharpen and Blur implementing it so that we allow for room of adding more filters in the future all while avoiding code duplication.

———

[ ]
‘Load image-path image-name’
'RedComponent image-name dest-image-name'
‘GreenComponent image-name dest-image-name'
'BlueComponent image-name dest-image-name'
'ValueGreyScaleImage image-name dest-image-name'
'IntensityGreyScaleImage image-name dest-image-name'
'LumaGreyScaleImage image-name dest-image-name'
'Flip-horizontally image-name dest-image-name'
'Flip-vertically image-name dest-image-name'
'Brighten (increment) image-name dest-image-name’
'Darken (increment) image-name dest-image-name’

‘Blur image-name dest-image-name’
‘Sharpen image-name dest-image-name’
‘Sepia image-name dest-image-name’
[ ]

Design Thought Process and Classes/ Interfaces Explained

The idea behind our design was to have an image class that had an 2d array with the dimensions width and height. And each element in the array was another class, pixel, which represents each individual pixels RGB colors respectively in fields. Both of these class have their own respective interfaces as well to allow for possible code additions later on.

We also have the main class that was given to us named ImageUtil and if you run the main method in that class it allows you to run the entire program.

We also have an ImageModel class and interface which is the main class for editing. It is through this class that you can edit any of the loaded pictures using the list of valid commands above.
(So when you run the imageUtil main method the model that we use is under the imageModel class/interface setup.

We also created a view Interface named TextScriptView and one method that implements it ‘ImageTextScriptView’ and the purpose of this method is to allow the program to render a message while the user is attempting to run and use the program.

To create the Histogram we created an interface ImageHistogramDataGenerator and the subsequent ImageHistogramDataGeneratorImpl which goes through the image and gathers every value of the specific channel or intensity and creates a list with length until the maximum color value (256) with each value being the amount of times the specified color occurs at that value.

In order to get the image that we are working on without possibly changing the original we created a new interface named ImageModelState, which gets the current state of the image and saves it.

Then, in order to create the visual of the GUI an interface GUIView and a class GuiViewWIndow implementing it, this class will take care of doing actions on a given image, displaying the image, loading an image, saving the image, getting the brightness of the image and lastly refreshing the image.

Now that I’ve gotten the main less messy setup out of the way, here’s where the code gets a little more crazy.

Manipulating the Image Design
So in order to manipulate the image , we first created and ImageManipulationModel interface which allows us edit the image in relative simple and directly based on what

type of command is used.
Off of that we have an abstract ManipulationModel class extending the interface because it turns out there are a bunch of classes that could save a lot of code if we abstract it.
Then, extending the abstract Manipulation model we have a ChangeBrightness class, and the FlipImage class, the change brightness class will either make the picture brighter or darker and the flip image class with flip the picture either vertically or horizontally.

Also extending the abstract ManipulationModel class, we have another abstract class called GrayScaleImage, we made this abstract because yet again there is a lot of code duplication that can be solved by just making it inside of an abstract class and there is a lot of classes that will need to reuse the code that makes a picture “greyScalified”.

Furthermore, extending the abstract GrayScaleImage we have four classes which are the three greyScales that require calculating, those being the IntensityGreyScaleImage, ValueGreyScaleImage, and the all of which most do some calculating to figure out what the greyScale value of each pixel should be. Finally, the last of the four classes is yet another abstract ComponentImage class, which takes care of the greyscale cases that takes one of the established pixel colors and makes the rest of the colors the same. Extending that componentImage class we have 3 classes that correspond to the three colors. Those being, the classes, BlueComponent, RedComponent, and GreenComponent.

In addition to this design, as the assignment called for, we added two new interfaces, Filter and ColorTransformation interfaces. The point of the Filter interface and implementation of Filter, is to allow for more types of manipulation to be done on the image. In this particular example, we have both Blur and Sharpen extending the FilterImpl. These classes allow for the image to be blurred and sharpened respectively. The purpose of the ColorTransformation interface and implementation is to allow for certain color changes and manipulations to occur on the photo. We have Luma, which we previously implemented, extending the ColorTransformationImpl, as well as a new color transformation, Sepia, which now also manipulates the image to be a older yellow looking photo. We made these design changes and additions as interfaces and extending a generic implementation to help get rid of code duplication and allow for future code add ons.

Command Design

For design the command aspect of this assignment we first started by creating a Commands Interface, this interface will let the given model run a command depending on the input.

Implementing this interface we have three more classes, two of them are for saving and loading commands, the names being SaveCommands and LoadCommands respectively. What those classes do is self explanatory. Lastly, the third class that implements the the command Interface directly is an abstract class named

CommandsImpl, the purpose of this class is to abstract the methods to reduce code duplication and to allow code reuse for the rest of the commands the user may use.

This abstract class has nine classes directly extending it the classes are listed and explained below.

- LumaGreyScaleCommands - Runs the luma grey scale method.
- VerticalFlipCommands - calls the method that flips the image vertically
- ValueGreyScaleCommands - calls the value greyscale method.
- ComponentCommands - runs the blue/green/red component grey scale method
- IntensityGreyScaleCommands - calls the intensity greyscale method.
- HorizontalFlipCommands - calls the method that flips the image horizontally
- BlurCommands - calls the method that blur a image
- SharpenCommands - calls the method that sharpen a image
- SepiaCommands - calls the method that sepia a image
- UpdateBrightnessCommands - update the brightness, either dimming the picture or brightening it.

Controller Design

Lastly we have the controller design, we first have the TextScriptController interface which represents a simple Image editor controller which attempts to alter an image and save it as a different new image. Implementing this class we have the class TextScriptControllerImpl which is the main point of heavy lifting in making the code run and allows the user to control what they want to happen. The class will run looking for inputs and doing the commands based off of the inputs and whether they are valid or not. It will run the correct code accordingly based on what the user inputs.

To allow for the commands and user input to be processed, we also created a new controller class which implements ImageEditorCommands named GuiViewControllerImpl, this allows the users inputs on the GUI to be translated into commands so the program understands what image processing method to use.

How to use Jar file(how to run the program):

Our program now supports three modes: Text Mode, Script Mode and GUI mode

1. Text Mode: The user can run the image editor in the console and then type the image editing command manually.

How to user to Text Mode:

-Open a command-prompt/terminal and navigate to the folder where the jar file is located.

-type'java -jar 6.jar -text' and press ENTER

2. Script Mode: when invoked in this manner the program should open the script file, execute it and then shut down.

How to user to Script Mode:

-Open a command-prompt/terminal and navigate to the folder where the jar file is located.

-type'java -jar 6.jar -fileTHE FILEPATHOFCOMMANDSFILE' and press ENTER
3. GUI mode: 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.
Or
-Open a command-prompt/terminal and navigate to the folder where the jar file is located.
-type'java -jar 6.jar and press ENTER

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

Citation for Images
'quincy' - Wanru Shao
'check' - Wanru Shao
(and all of the rest of the quincy variations are also from Wanru Shao)