Contributors:
> Akash Alaparthi- *Northeastern University, Khoury BS 2025*
>
> Druv Sarin - *Northeastern University, Khoury BS 2025*

Project Description

This project represents a basic image processing program. Files with the PPM, JPG and PNG format (images) can be loaded into the program. Once loaded, various image manipulating functions can be used on the images with the help of different buttons. These operations include flipping operations, dim/brighten operations, and color-conversion operations.

Design change from assignment 5
>

Model, View, Controller Overview:

Changes from Assignment 5
EXTRA CREDIT: Downsize function has been both designed and implemented.
To implement this feature: first I added a new method called downsize to both the photoModel class and the Photoeditor interface. Next I made the Downsize command class, which I then placed in the hashmap located in the controller to effectively handle user input. I then made a new button in the GUI view class and added another action listener for the downsize function. This called the downsize method from the model and implemented it on a given image.

> New view added called GuiView. This addition initializes all of the panels and buttons for the GUI and also sets the layout for it as well.
> PixelImplInterface added in the model. The Pixel class implements this interface and it includes all the same methods as before.
> ImageImplInterface added in the model as well. The Image class implements this interface and it also includes no new methods.
> Additionally a controller for the GUI was made in a class called SwingController
> Interface to represent a GUi view was also made.

In this section, I will go over the purpose of each class/interface in all the models, views, and controllers.

Note: This project makes use of the **command design pattern**.

Model:

Our model includes the following:

- PhotoEditor interface
- PhotoModel class
- Pixel class
- Image class
- GreyScaleEnum Enum
- PixelImplInterface
- ImageImplInterface

PhotoEditor Interface:

An interface for PhotoEditor that includes operations that can be performed on images with any model implementation. The implementation has a HashMap of known images that can be manipulated using given operations and this new Image will call a specific image stored in the model's imageList map.

PhotoModel Class:

This class includes the core implementation of the PhotoEditor interface. This model stores a hashmap of images - the *imageList* - which holds all the images that users can conduct operations on. Images can be added to the image library using the *load(String path, String name)* method. The model also stores various methods for operations that can be conducted on the images. Each of these methods will call a new PPM image to store the operated image; each operation will directly manipulate the Image or the 2D array of pixels.

****PixelImplInterface****

The Pixel class implements this interface and it includes all the same methods as before.

****ImageImplInterface****

The Image class implements this interface and it also includes no new methods.

Pixel Class:

This class represents a Pixel. PPM images are made up of these pixels, each of which has a Red, Green, and Blue component. RGB values are all integers and are stored in the format _0-255, 0-255, 0-255_, 255 being the maximum value for any particular color component.

Image Class:

Class to represent a PPM image created from pixels. This class stores all the pixel values in a 2D array of Pixels of a specified height and width. It also takes in a maximum RGB value.. This class stores image operations that can be conducted to get the attributes of an image.

GreyScale Enum:

This Enum sets the constants of the greyscale including “red”, “green”, “blue”, “value”, “intensity”, and “luma”.

View:

Our view includes the following:

- ImageView class

ImageTextViewImpl class:

The image view class offers a primitive method of displaying user commands and their results as strings on the command line using *renderMessage*. This class will accept orders from the controller in the form of strings to display to the user.

Controller:

Our controller includes the following:

- ControllerImageImpl class
- commands package

ControllerImageImpl class:

The primary implementation of the controller interface. This controller class is constructed with the PhotoModel (referenced when conducting operations), a Readable for user input, an ImageView to transmit information to the user, and a map of known commands. It also includes the option to quit the program at any given time using the ‘q’ key. This class also houses the image I/O functions.

commands package:

This package includes various command objects that represent different manipulative operations that can be conducted by the user. For example, the _Brighten_ class represents a brightening command that will increase the brightness of any image by the specified increment and save it to a new file.

The commands package has an ImageProcessingCommand interface that holds the execute(PhotoModel model) method. This is the method that each of the class implementations of this interface uses to interact with the model. Each implementation represents a different type of command that the user can use when interacting with the program.

Commands:

> **Brighten:** allows the user to brighten an image by a certain value and save it to a new name.
>
> **Darken:** allows user to dim an image by a certain value and save it to a new name.
>
> **Grayscale:** allows the user to convert an image to the specified grayscale type and save it to a new name.
>

> **FlipHorizontally:** allows the user to horizontally-flip an image and save it under a new name.
>
> **FlipVertically:** allows the user to vertically-flip an image and save it under a new name.
>
> **Load:** allows the user to load in a new image from the specified image-path and add it to the current _imageLibrary_.
>
> **Save:** allows the user to save an image that exists in the _imageLibrary_ to the specified image-path.
>
>**Blur** allows the user to blur an image and save it in the imageList hashmap with a new name.
>
>**Sharpen** allows the user to sharpen an image and save it in the imageList hashmap with a new name.
>
>**Sepia** allows the user to sepia transform an image and save it in the imageList hashmap with a new name.
>
>**GreyScaleLuma** allows the user to greyscale with transform an image and save it in the imageList hashmap with a new name.

***

Interacting With Program:

- First, the user should navigate to the Main class and run the main method. This will prompt the program to run.

- Now the user can load in images, manipulate them, and save them to their device.

Click Buttons to perform actions

> **Common references:**

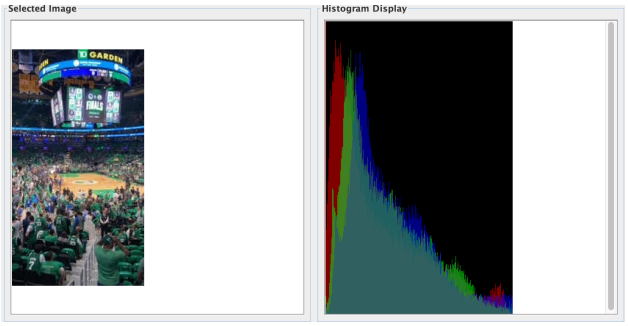
>
> - *image-path:* the path of the image on the user's device. For example, res/TDgardenPhoto.ppm

>
> - *image-name:* the name that will henceforth be how the user should refer to the image in the hashmap. For example, TDgardenPhoto.
>

** Example interaction **

*The below images shows the before and after of the brighten and downsize function performed using a factor of 25 on the TDgardenPhoto.PPM image along with the histogram created once these actions are performed.

Before:



After: