PDP Assignment 6
GRIME: Graphical Image Manipulation and Enhancement

This was done as part of our assignment during our Master's CS course, Programming Design Paradigm by Vaibhav Sankaran and Ishwarya Rajkumar.

Overall Design:
The design follows Model, View, Controller architecture. When we run the main class, a GUI opens up where the image is loaded and manipulation is done to the respective image and it is saved. The workflow of the above model is, the user interacts with the VIew, which sends the commands to the Controller. The Controller processes the commands by taking in the corresponding Model methods and ensures that the user input is properly mapped to the image operations, which modifies the image. The View is updated with the modified image or results and is finally reflected on the GUI.

Changes:

1. From Assignment 4 to Assignment 5:
   - The design of the model is as mentioned above and there are no changes.
   - In ImageModelImpl the following operations are being added: compress, histogram, color-correct, levels-adjust, split view of the image. To use these operations, the rules are mentioned below in the "How to run the code" section.
   - Accordingly the ImageControllerImpl handles the user input and calls the required operation's implementation in the model.
2. From Assignment 5 to Assignment 6:
   - The design of the model remains the same as mentioned above.
   - In ImageModelImpl image downscaling operation is being added. To use the operation the rules are mentioned below in the "How to run the code" section.
   - Accordingly the ImageControllerImpl handles the user input and calls the required operation's implementation in the model.
   - Furthermore, the GUI implementation of the model is added. It creates a window for user interaction, with buttons for operations, input fields specific to that particular operation and display area for images and its corresponding histogram.

/src

1. Main.java - The Main class serves as the entry point for the image processing application. It handles the initialization of the model, view, and controller components. The application follows the MVC (Model-View-Controller) design pattern. The model handles the logic and data manipulation. The view is responsible for displaying the data. The controller mediates between the model and view to process user commands.

2. Packages:
    I. model:
        - ImageModel.java - This interface is for the image model, defining methods for manipulating images. This interface provides various operations for loading, saving, and modifying images. It uses different image processing techniques such as color extraction, flipping, and filtering. Images are represented as a 3D array where each pixel is defined by its RGB values. The first dimension represents rows, the second represents columns, and the third is the color channels. The code handles different image formats to load pixel data. For PPM pixel values are read in text format. Whereas, for PNG/JPG, RGB values are extracted from packed integers.
        - ImageModelImpl.java - An implementation of the ImageModel interface. It manages image loading, saving, and manipulation operations. Images are represented as a 3D array where each pixel is defined by its RGB values. The first dimension represents rows, the second represents columns, and the third is the color channels. The code handles different image formats to load pixel data. For PPM pixel values are read in text format. Whereas, for PNG/JPG, RGB values are extracted from packed integers.
        - MockImageModelImpl.java - This class is used to mock the actual model to test the controller alone.
    II. view:
        - ImageView.java - Interface for the image view, defining methods for user interaction. This interface represents the communication between the system and the user. This can be an image manipulation application, providing methods to display messages.
        - ImageViewImpl.java - This is the GUI Implementation of the mvc.ImageView interface. It creates a window for user interaction, with input fields and display areas. It includes a display area for images and its corresponding histogram, dynamic updates based on model changes, and event handling for user actions. A JButton is created for the operation and then it goes through the listener which gets the respective inputs for the operation and then it is finally passed through the controller.
    III. controller:
        - ImageController.java - Interface for controlling image operations within an image processing application. This interface defines the contract for any class that implements image controller functionality. It allows the execution of commands that manipulate or process images.

- ImageControllerImpl.java - Implementation of the ImageController interface. This class handles user input and processes image commands with ImageModel and ImageView. It allows for various image operations to be executed based on user commands.

IV. operations:

- ImageOperation.java - Interface for controlling image operations within an image processing application. This interface defines the contract for any class that implements image controller functionality. It allows the execution of commands that manipulate or process images.
- AbstractImageOperation.java - An abstract class representing a generic image operation. Subclasses must implement the specific operation logic by overriding the performOperation method. This class defines the structure for image operations that modify an image.
- LoadOperation - Represents Class for loading an image from a specified file into an image model. This operation reads the image data from a file and stores it in the image model.
- SaveOperation - Represents an operation to save an image to a specified file. This class extends AbstractImageOperation and implements the logic. It saves the image identified by the source image name to the destination file name.
- BlurOperation - Represents an image operation that applies a blur effect to an image. This operation processes an image stored in the model, and applies a blur filter. It stores the blurred image with a new name in the model.
- BrightenOperation - Represents an image operation that increases the brightness of an image. This operation modifies the pixel values of the specified image by a given increment, resulting in a brighter version of the image, which is then stored with a new name. To darken the image give a negative value.
- GreenComponentOperation - Represents an image operation that extracts the green component from an image. This operation processes the specified image to isolate the green color channel. It also stores the resulting image with a new name.
- BlueComponentOperation - Represents an image operation that extracts the blue component from an image. This operation processes an image stored in the model and creates a new image. Only the blue component is preserved and stores the result with a new name.
- RedComponentOperation - Represents an image operation that extracts the red component from an image. This operation processes the specified image to isolate the red color channel. It also stores the resulting image with a new name.
- ValueComponentOperation - Represents an operation to extract the value component from an image. This class extends AbstractImageOperation and is responsible for processing the image. It isolates and saves its value component to a destination file.

- LumaComponentOperation - Class for extracting the luma component from an image. This operation takes an image and generates a new image. It contains only the luma component.
- IntensityComponentOperation - Class for the intensity component operation on an image. This operation extracts the intensity component from a specified image. and stores it in a destination image.
- HorizontalFlipOperation - Represents an image operation that performs a horizontal flip on an image. This operation processes the specified image by flipping it horizontally. It also stores the resulting image with a new name.
- VerticalFlipOperation - This represents an operation to flip an image vertically. This class extends AbstractImageOperation and is responsible for processing the image. It creates a vertically flipped version and saves it to a destination file.
- SharpenOperation - Represents an operation to sharpen an image. This class extends AbstractImageOperation and implements the logic. It enhances the edges and details of the specified image.
- SepiaOperation - Represents an operation to apply a sepia tone effect to an image. This class extends AbstractImageOperation and implements the logic. It transforms the specified image into a sepia-toned version.
- RGBSplitOperation - Represents an operation to split an RGB image into its individual red, green, and blue components. This class extends AbstractImageOperation and implements the logic.
- RGBCombineOperation - Represents an operation to combine red, green, and blue components into a single RGB image. This class extends AbstractImageOperation. It implements the logic to combine the specified color component images into a destination image.
- ColorCorrectOperation - This represents an operation to correct the color meaningfully in an image. This class extends AbstractImageOperation and implements the logic. It transforms the specified image into a color-corrected version.
- CompressOperation - Represents an image operation that compresses an image. This operation modifies the pixel values of the specified image by a given percentage, resulting in a compressed version of the image (in terms of size), which is then stored with a new name.
- HistogramOperation - Represents an operation to produce an image that gives a histogram for the given image. This class extends AbstractImageOperation and implements the logic. It produces the histogram of the given image.
- LevelsAdjustOperation - This represents an operation to adjust the color meaningfully in an image where b, m, and w are the three relevant black, mid, and white values respectively. The values should be in ascending order. This class extends AbstractImageOperation and implements the logic.

- ResizeOperation - This represents an operation where the image is resized. Here, the image is being downscaled where the number of pixels are being decreased. This class extends AbstractImageOperation and implements the logic.

Image Downscaling:

This model supports Image Downscaling as well where the image is being downscaled where the number of pixels are being decreased and resized. This class extends AbstractImageOperation and implements the logic.

The following changes are being made:

- A new class called ResizeOperation is created which extends the AbstractImageOperation, executing this will call the implementation in the Model.
- In ImageModelImpl image downscaling implementation has been added. To use the operation the rules are mentioned in the "How to run the code" section.
- Accordingly, the ImageControllerImpl handles the user input and calls the required operation's implementation in the model.
- The image downsizing is exposed suitably in the graphical user interface too. In ImageViewImpl we have created a new resize JButton and added a listener which gets the image name, destination image name, width of the image and the height of the image, this is then passed to the controller.

This operation can be used only using GUI. You are expected to enter the expected width and height of the image as input.

Features:

Exposed in UI and command-line

- Load
- Upload Script
- Save
- Blur
- Red-component
- Green-component
- Blue-component
- Luma-component
- Sepia
- Horizontal-flip
- Vertical-flip
- Color-Correct

- Compress
- Levels-Adjust
- Sharpen
- Resize Image

Exposed only in UI
- Resize Image (Image Downscaling)

Exposed only in Command Line
- Intensity-component
- Value-component
- RGB Split
- RGB Combine
- Brighten
- Histogram

Support Split
- Blur
- Sharpen
- Sepia
- Red-component
- Green-component
- Blue-component
- Luma-component
- Intensity-component
- Value-component
- Color-correct
- Levels-Adjust

How to run the code:
1. First, install IntelliJ or a similar IDE.
2. Download the zip file, unzip it, and keep it in the desired folder.
3. Make sure you have the src, res, test, out, and docs directories.
4. Expand the src directory and open Main.java.
5. Follow the below steps to either run using GUI or run using JAR.

   To run using IDE -
   - Run the main.java class, this will automatically open the Graphical User Interface.

- First, load the desired image by choosing the file from the system by selecting the load button and type the image name. Initially, only the load button and the Upload Script button will be visible. Only after the image is successfully loaded the other manipulation buttons will be visible.
- Make desired manipulation to the image by clicking the button for other operations which will now be available.
- Save the final image.

To run using JAR (3 ways) -

- Open command prompt/terminal and navigate to the respective folder:
  1. Type `java -jar <Program.jar>` and press Enter.
     - This will now automatically open the GUI. Now, load the desired image by choosing the file from the system by selecting the load button and type the image name. Initially, only the “Load” and the “Upload Script” buttons will be visible. You can also click on the “Upload Script” and upload the script file which contains multiple commands which are to be executed. Only after the image is successfully loaded the other buttons to manipulate the image can be used.
     - Make desired manipulation to the image by clicking the button for other operations which will now be available.
     - Save the final image.

  2. Type `java -jar Program.jar -file path-of-script-file` and press Enter.
     - This will now automatically run all the commands in the script file.
     - If any error occurs while executing any command, a message starting with “Error: “ will be displayed in the terminal.
  3. Type `java -jar Program.jar -text` and press Enter.
     - This will now ask for input from the user.
     - Follow the rules below to use the available operations.

Rules for commands:

- load image-path image-name: Load an image from the specified path and refer it to henceforth in the program by the given image name.
- save image-path image-name: Save the image with the given name to the specified path which should include the name of the file.
- red-component image-name dest-image-name: Create an image with the red-component of the image with the given name, and refer to it henceforth in the program by the given destination name. Similar commands for green, blue, value,

luma, and intensity components should be supported. Note that the images for value, luma, and intensity will be greyscale images.

- horizontal-flip image-name dest-image-name: Flip an image horizontally to create a new image, referred to henceforth by the given destination name.
- vertical-flip image-name dest-image-name: Flip an image vertically to create a new image, referred to henceforth by the given destination name.
- brighten increment image-name dest-image-name: brighten the image by the given increment to create a new image, referred to henceforth by the given destination name. The increment may be positive (brightening) or negative (darkening).
- rgb-split image-name dest-image-name-red dest-image-name-green dest-image-name-blue: split the given image into three images containing its red, green, and blue components respectively. These would be the same images that would be individually produced with the red-component, green-component, and blue-component commands.
- rgb-combine image-name red-image green-image blue-image: Combine the three images that are individually red, green, and blue into a single image that gets its red, green, and blue components from the three images respectively.
- blur image-name dest-image-name: blur the given image and store the result in another image with the given name.
- sharpen image-name dest-image-name: sharpen the given image and store the result in another image with the given name.
- sepia image-name dest-image-name: produce a sepia-toned version of the given image and store the result in another image with the given name.
- compress percentage image-name dest-image-name: modifies the pixel values of the specified image by a given percentage, resulting in a compressed version of the image (in terms of size), which is then stored with a new name.
- color-correct image-name dest-image-name: color corrects an image by aligning the meaningful peaks of its histogram.
- histogram image-name dest-image-name: produces an image that gives a histogram for the given image.
- levels-adjust b m w image-name dest-image-name: adjusts the color meaningfully in an image where b, m, and w are the three relevant black, mid, and white values respectively. The values should be in ascending order.
- Operations Preview: For certain operations, a split view of the image can be seen. The operated image is on the left and the original image is on the right. blur, sharpen, sepia, greyscale (red component, green component, blue component, luma component, value component, intensity component), color correction, and levels adjustment. To use this feature, give the command as mentioned earlier followed by “split percentage_value”.

Citation:
- Images used in the /res folder are our own and I authorize their use in this project.