README

This is an image processing application developed in the Java programming language using the MVC architecture. The application currently only supports terminal as its view container to display the messages of the image operations while running the code. The application can also parse a file input containing the list of commands and perform the operation asynchronous.

Enum Package:

This stores all the enum data that are used in the model, view and controller implementation of the MVC architecture. It stores the name and value mapping along with some enums consisting of lambda methods which are used dynamically as when required by all the three component multiple classes.

Model Package:

Our model represents an image. The model package of our MVC architecture contains the one main model class named *RGBImage.java* which implements the *RGBImageInterface.java* interface. This class contains all the commands as public methods which can be called from the controller class as per the required user input.The model contains several utility classes for each individual operation on the image.

The model contains the following packages:

1. imageoperations: It contains multiple packages for handling different commands of the image operations like single input single output, single input multiple output, and multiple input single output. All the files in the packages deal with the image operations that are handled for the present image processing application.
2. RGBImage.java: It is the main model file which interacts with the controller, gets the command from controller and redirects the given command to above image operation utility classes.
3. RGBImageInterface.java: Interface which has the overview of all the methods that are expected to work for the given MVC architecture and user requirements.

Controller Package

The controller is made based on the command builder design pattern.
The controller contains the following packages:

1. imagecommads: This is similar to model class. It has all the helper utility commands divided into hierarchical packages and files. The main controller redirects the commands to these utility classes for passing the input to the model.

2. filehandling: The package deals with read and write operations on the files stored on the hard disk of the local machine containing the images. The package contains the cases for both imageIO and ppm file handling functionality.
3. RGBImageControllerInterface.java: The main controller interface class which contains the command parsing and builder design. All the commands are mapped to a lookup table which creates the objects of the controller command utility classes during runtime when received an input from the user.
4. features: This package contains details and implementation related to command callbacks methods. Using the object of the feature class view can put a request to the controller if any external event happens to the MVC environment. It has one interface which is implemented by a class. This class exposes only limited functionalities of the controller showcasing the benefit of composition over inheritance.
5. graphicalcontroller: This package contains implementation details related to the controller part which handles the GUI aspect of the image processing application. It contains an interface which represents the basic functionalities offered by the controller and its implementation details are present in the *GraphicalController.java*.
6. scriptcontroller: This package has all the implementation details of the controller which handles the interactive text based console and file scripting aspect of the Image Processing Application.

View Package

Our Image Processing Application currently now supports three types of user interactions-

1. In the first method, the input can be given through a file using the "-file" flag while running the main class or even the JAR file.
2. In the second method , the input can be given using an interactive text console using the "-text" flag while running the main class or even the JAR file . For our case it interacts with the user using the terminal itself.
3. In the last method, the user can directly interact with the Graphical User Interface which has buttons and labels for an interactive experience. The GUI opens up by passing no flags as input while running the main class or the JAR file.

This package contains code and implementation details related to the graphical user interface part of the image processing application. The view allows the user to interact with the application via the Graphical User Interface(GUI). It allows the user to load an image, perform operations on it and save the new image. The GUI has been built using Java Swing.

The view contains the following packages:

- dialogmenus: It contains the implementation details about the JDialog boxes that pop-up when additional information like split-preview %, level-adjustment values etc are needed to perform an operation. The package has two sub packages-
    - Multiindialog- This handles the dialog pop-up which needs to get multiple data from the dialog box.

    - singleindialog: This handles the dialog pop-up which needs to get only the split preview data from the dialog box.
- GraphicalView: This is the main view class which implements the *IView* interface. The controller sends all the commands to this class. It has the responsibility to display all the functionalities like buttons, headings, labels, etc. Whenever an event occurs this view sends the request about the happening to the controller.

The view contains the following features:

1. Load Image: It allows the user to upload a new image that needs to be operated upon.
2. Save Image: It allows the user to save the original or operated image.
3. Exit App: It allows the user to quit the application gracefully.
4. Buttons for operations: These buttons allow the user to perform various operations on the image like getting its red/green/blue components, flipping the image, blur, compress etc.
5. Image Preview: It allows the user to preview an image operation before it is actually applied upon the image.
6. Histogram: It allows the user to view the histogram of the image.
7. Save operation: It allows the user to save the current operation on the image after previewing it. Now this will be the image on which next operations will be applied.
8. Cancel operation: It allows the user to cancel performing an operation on the image.

The GUI displays the currently processed image, which can be larger than the allocated area. Users can scroll to view the entire image.

ImageProcessingApplication.java

- The main class now successfully parses the '-file' flag and takes the script file input as command line arguments to support the requirements mentioned in the assignment.
- The main class also takes the '-text' flag if the user wants to start the image processing application program using an interactive text based console.
- If no flag is passed to the main class then it opens up the GUI part of the MVC architecture where the user can interact with the application using components like buttons, labels, sliders etc. It can also see the live image and the histogram present in display.

Test Package

The test package has following structure-

Controller Package

It contains the test related to the controller and its integration test with the model.

1. ControllerModelIntegration.java: This file tests the controller and model combined as a whole. The input is passed from the user perspective and the output is checked against assertions.
2. filehandling: The package tests if the read and write operation works correctly for ppm, jpg, png, jpeg etc extensions. The ppm testing is done in the PPMUtilest.java and rest of the ImageIO extensions are checked in the ImaeIOTest.java.
3. Mocks: The package has information related to mack testing of controllers. The package has a test file for mock testing of script controllers with the model and graphical user interface controller with both model and view.
    a. ScriptControllerModelMock.java: Test the scripting controller mock with only model and logs the data.
    b. GraphicsControllerViewModel.java: Test the graphical user interface controller mock with both the model and the view. It checks all the tests using a log data variable.

Model Package

It contains the test related to the main model class and its image operation classes in the respective individual packages.

The model test has following packages-

a. multiin: This checks for test cases where the model's multiple input methods work correctly. For now we only have Combine rgb which is checked by this package.
b. multiout: This checks for test cases where the model outputs multiple images after an operation is performed. For now we only have the split rgb method which splits the single image to r,g and b images.
c. singlein: This package checks for the rest of the functionalities of the image processing application like brightness, flip operation, greyscale, single component(Monochrome.java), sepia and sharpness, level-adjustment, color-correction, compression, histogram operations.
d. RGBImageTest.java: This file checks for the testing cases for the main model class calling all the image operations one after the another.

Assumptions

1. After every successful image it needs to be saved using the “save operation” button or canceled using the “cancel operation”. When the save is applied the current image in use becomes the final image.
2. For pop-ups instead of clicking “Applying Preview” if any other button like closed is pressed then no action is performed on the image.
3. If a different color-greyscale operation is applied on a given grayscale then it should result in black image. For eg- Green component action on red component should result in black image.
4. If after any split preview operation(OP1), instead of saving the operation a different non-split operation(OP2) is applied then it performs the operation OP2 on either the last saved image or the loaded image. But none of the images are saved. So after performing any operation the application can only know that the user intends to perform an operation on a particular by only clicking the save operation after each one of them.
5. If after applying an operation the user doesn’t feel like keeping it, then the operation should be canceled first using the ‘cancel operation’ button before applying the other image operations. So, whatever operation is applied in the UI, it is applied on either the recently saved image or the freshly loaded image.

Design Changes

Controller Component-

- Created a new controller to handle the new view (Graphical User Interface) aspect of the image processing application.
  - As with the introduction of the new view the controller has to now perform completely different operations compared to its existing part which was scripting controller. Hence, making a new implementation makes the design more flexible.
  - The previous controller depended on the script parsing techniques where it has to get the particular syntax of word to perform any image operations whereas now the controller needs to interact with the UI to perform the image processing operations. Both of the scenarios are completely different so making a new controller extending the old interface and implementing the new interface made more sense..
  - As it is a scalable design, with the requirement one can create multiple controllers and can interact with different types of views as required by the business constraint.
  - The controller takes model and view as its constructor parameter during its creation so all of the three components are loosely coupled. This gives the

    flexibility option to provide any view and model class to the controller before starting the image processing application using the main method.

- Addition of features callbacks-
    - Added a new package for feature callbacks functionality. The feature class basically represents a high-level request communication sent by the view to the controller on occurrence of any events.
    - The feature class helps to make the implementation of controller and view more abstract with respect to each other as each of them now don't need to know their internal implementation details.
    - They now rather communicate through a common buffer class which is the feature class.
    -
    - Dynamically if required the features can be updated and a new set of features can be provided to the view which won't affect any of the existing implementation details of either the view or the controller. Even the program doesn't need to restart to get a new set of features for the updated callbacks.

Model Component

- Addition of methods to Model interface and class-
    - In order to make the model and the controller loosely coupled we introduced another method in the interface. This method allows you to change the value of the image matrix pixels dynamically. This even allows you to make an empty image object and change it as per the need when a new image is loaded or saved using the controllers.
    - Using this method the coupling between the controller and model reduces significantly and the controller doesn't need to exactly rely on the concrete class of the image in order to create a model for the application.

View Component

- This is not exactly a design change but new functionalities to support the GUI was added into the view package which manages the interaction of the user. There is no design change here but a new component(View) was added to the existing model and controller part of the MVC architecture.

Resources Package

The resources package contains the script, readme and useme files which can be used to run to generate the images as required for all the functionalities. The resources folder contains the source image along with the other images generated as a result upon the source image.

Citations

[1] Image used: Pixelation to represent endangered species counts | FlowingData