Code Review

Design critique

The design is good overall. Multiple levels of abstraction were made to reduce code duplication, command design pattern was widely used for future extension. Component-specific critique will be put in corresponding sections below.

I. Model

The model has three levels - the state manager (/the temp cache/image library), the image itself, and the pixels that construct images.

Library

An image library is essential for this program as it has to be capable of loading and processing multiple images in a single run. The provided code has such a library, implemented by a `LinkedHashMap` and supports store and retrieve, nicely done. However, some improvement can be made to take the whole advantage of this library structure. A major issue is the program fails to present library in the GUI view, and as it's more a view-aspect issue, see §LibraryPanel. Another minor issue is the program fail to draw a clear line between the file system and the library (temporary cache). On the one hand, it provides enough convenience for users to manipulate an image without loading and saving, on the other hand however, this design may make the interaction (especially in GUI) ambiguous and distracting:

1. Both CLI & GUI, when user want to save, they may only want to save that image from the library to the machine file system, but the current design allows them saving the current image into the library again, as a new alias or overwrite an existing one, yet this can be a bit tricky since users don't actually need two exact same images in the library. The same logic applies to the load operation.
    - I dig into this problem so deep because when I was trying to save an image the first time from GUI, and name it without an extension, I can't find it anywhere in my machine. Well, obviously I CANNOT because it never writes into my machine. The thing is there isn't an image list in GUI tells me I actually save it into the library, nor the program complains that I forget to add an extension to write it into the file system. We believe adding any of these prompt, or disable the program from unilateral loading/saving [1], would make the save process more user-friendly.
2. Say if the future implementation supports an image selection list in the GUI, we may want our user to rename each new image, so every of them is identifiable and the program can returns the correct one when user tries to retrieve. But the current design allows them to rename an image with an extension and that would save the image directly to the file system. This makes less sense for GUI, as users may not want to switch between the program window and the file explorer. [1]

Image & Pixels

Both image and pixel objects are property-retrievable and mutable. They also guarantee the mutation on a copy would not affect anything from the original object. Furthermore, using self-implemented Pixel class, instead of using `java.awt.Color`, definitely allows implementors have more control on the provided image. Well done!

One heads-up would be both Image interface and Pixel interface may not be extensible as expected. The problem is the copy method always returns an instance of the obsolete interface, which can no longer use the newer methods declared in the extending interface. Unfortunately, we were also plagued by this problem and failed to come up with an ideal solution. One potential direction could be making the interface generic on the color profile (RGB/RGBA/CMYK), yet may at an expense of making the whole project complicated and hard to read.

Operation

Operation package was clearly organized and classified so that future implementor (like us) would know where the new operations should go, which keeps the coherence of the whole project. Common code and helper methods were abstracted out for different operation classes, which reduces the code duplication and lessen the required code for new operations. I found both grayscale operation and convolution operation were designed in a highly extensible way - they only require implementer to overwrite one method/provide a convolution kernel, definitely a good practice.

II. Controller

Handler

Handlers process the program level input and ask an appropriate operation to update the model. This part is nicely done!

CLI & Script

As we will mention below (I recommend reading the linked review below before proceeding to this section), it has to be the controller's responsibility to run the program, by the MVC principle, so the controller needs to have a run method, yet currently the `run()` is put in the view.

As for the existing methods, since `fetchImage` and `cacheImage` are not directly recive arguments from a readable stream, nor any I/O is involved, but only are manipulating the model, I would suggest put them into the model module; the `loadFile` and `storeFileIfPossible` , on the other hand, are I/O involved, but essentially they are things have to be **handled** by a controller (but doesn't have to be this particluar CLI one), so it would be better to put them into handler package.

GUI

The ImageProcessorGUI wasn't used or implemented for this project. But as we mention below, the GUI should use a GUI controller to communicate the model and the view, as well as accept (click/select) events.

Here we will assume the project is going to be fixed so the controller runs the program.

The reason a different controller is needed is the graphical view would emit signals(list selection event) back to the controller, and accept users' inputs(brighten value/mosaic seed). In this case, the controller has to be capable of processing signals and receiving inputs from the view. The arguments parsing checking for the GUI controller can be looser than the CLI controller, as this controller is less likely to recive wrong inputs (particularly invalid operation command), becuase only the supported operations are visible and clickable in the GUI view.

III. View

The view for both CLI and GUI are able to render a graceful user interface: CLI interface has a help menu, input prompt, and error message feedback; GUI has a program window and clickable items as event triggers. Both of them are visually decent, yet from the implementation perspective, they may violate some design principle, see Implementation Issue of View.

CLI

No major issue. CLI works very well in terms of a visualizer.

- One opportunity would be providing a more specific/accurate error message:
    - Example 1
        - Input: brighten p1 p3 pp
        - Current: Failed to handle [brighten, p1, p3, pp]: For input string: "pp"
        - Suggested: Failed to brighten p1 with magnitude pp: magnitude should be an integer.
    - Example 2
        - Input: load res/parrot.png parrot
        - Current: Failed to handle [load, res/parrot.png, p1]: Issue loading file res/parrot.png using COMMON transferrer
        - Suggested: Failed to load res/parrot.png: cannot find such a file in the machine
    - Example 3
        - Input: load res/parrot.mp4 parrot
        - Current: Failed to handle [load, res/parrot.mp4, p1]: Issue loading file res/parrot.mp4 using COMMON transferrer
        - Suggested: Failed to load res/parrot.mp4: mp4 is an unsupported file format.

GUI

The graphic user interface supports basic I/O, image processing, and image preview. The layout is rational and the triggers is recognizable enough. I like the way they put an image placeholder in the preview window to inform users the program hasn't had an image loaded to be processed.

Improvement Suggestion

Window

- Set a minimum size of the program window so that users cannot scale down the window to a size that makes all components squeezed together
- Wrap the preview image into a scroll panel so that it's possible to view a relative large image in a small window size
- When resize the program window, also resize the preview panel together.
- Fix the "load and save" area, so it correctly shows the file path of an image

LibraryPanel

- Put the library on the GUI view. Missing a library makes the interaction more like a linear process, meaning after users performing an operation on an image, they can never retrieve it but can only continue on the most recent one, and if they want to view/operate the previous one, they have to load again from the machine's file system.

Histogram

- Histogram is missing from the GUI. We do believe, additionally, the histogram package should be put in the view (view.GUI will make more sense) module instead of the model module, as
    1. model doesn't need the information from the histogram to run. Even if the future implementation wants to make the histogram interactive, like adjust image by dragging a curve, that interaction is fundamentally a trigger and has no difference with buttons and selection lists.
    2. A histogram is only demanded by GUI, putting a histogram in a text will make no sense.

Interaction

- Fix the extra argument text area
    - The text area will shrink when it's empty, and becomes unrecognizable as well as hard to click.
    - Everytime the program starts, the text area would be filled with a placeholder string. User have to delete them one character by one, instead of clicking in and type immediately, to continue.
- Rather than letting users put extra arguments into the text area themselves, program should use pop-up windows whenever an operation needs extra arguments. This is more intuitive and user-friendly.
- When an operation fails, an alert window should pop up and let user know how to correct inputs or continue.
    - The program will be more user-friendly if an operation-completed-window pops up when an operation is finished and informs/confirms what the user just did. (for instance: "The blurred image of *Picture1* has been successfully created and is named *Picture2* stored in the library!"). Same thing

applies to CLI.

- I would recommend use buttons instead of selection lists to trigger operations. One problem is JList triggers `valueChanged()` twice after one item is selected, and to avoid this side effect, the actual action should be surrounded by

```
if (!e.getValueIsAdjusting()) {
    ...
}
```

  However, this will introduce another issue: `valueChanged()` will not be triggered if the user select the same item again, and this in effect prevents them to repeat the same operation.

Action Listener

- See Implementation Issue of View.

IV. Extensibility

The program is overall highly extensible benefit by the clear structure.

- To add a new feature, only two classes have to be added -one operation and one handler-and only one existing class has to be modified for supporting the new feature - simply put an instance of the new handler into the main class's support handler list.
- Adding a new file type support to this program was also documented well in the README, and seems plausible and easy to implement.
- One advantage of this program is it's also open for adding new command line arguments (--file/--text/...), this can be useful if the program is going to support another UI.

Implementation critique

I. Strength

It's been observed that the code has a very clear package structure and a relatively understandable hierarchy of inheritance. Every class is implementing a well-designed interface and doesn't leak any implementation details as all public methods return abstraction. Common code was abstracted out to reduce duplication. Most methods have a relative small size and only do one thing.

II. Limitation (Implementation Issue of View)

- During the inspection of the main class, we found main ask the view to run the program. By the definition of MVC, however, it should be the **controller** updates the models and views by accepting input and performing the corresponding actions. View is merely the means of displaying objects within an application, without directly manipulating the model. Therefore, all arguments parsing components (for CLI)/ event listening components (for GUI) should be put in the controller package.

- For CLI, view should only be capable of rendering the message and error. Note here, again, view is just a displaying unit, so it doesn't know what exact message to render until it receives that message from the controller. That is saying, when print out the command menu, it's also the **controller's** job to **supply** the name of the supported command of its command set [2] and ask the view to print out.
- For GUI, view should only construct the program window that renders the image and presents clickable items (as well as the render message/error capability, ideally through a pop up window), while not to provide any functionality to respond to the event, even these events are triggered within the view.
  - We would suggest when adding the `ListSelectionListener` (for JList) or `EventListener` (for buttons), instead of setting the view itself as the Listener, set a Listener object (an instance of the class who implements the `ListSelectionListener` / `EventListener`) who is considered as a controller component. That will also imply view doesn't have to implement the `Listener` interfaces.
  - That "Listener class" can be put in the GUI controller as a private inner class, which can be instantiated in GUI controller's constructor, at the point of initializing the associative GUIView (so the view will need an additional argument for construction, i.e. the `Listener` that is going to be set as the listener of view's triggers) of the controller, and is dispatched to hear events from the view panel. We recommend this, instead of letting the new GUI controller implements multiple interfaces, is because the alternative way may lead to casting at the point the program needs view's controller object as an instance of `ListSelectionListener`. Take the advantage of inner class, these listeners can still pass the arguments they receive from the view to the controller and let the controller perform the operation.

III. Improvement Suggestion

- Multiple abstract classes exists just for scoping purpose, yet not providing any abstractions (`AbstractGeometricCommandHandler` / `AbstractComponentCommandHandler`), nor does any method only accepts the sub-scope object as an argument. Thus, these abstract classes can be trivial in terms of code implementation, and may be safely dropped.
- Using code inspecting tool, we found the project has several declaration redundancy and believe they can be removed without any side effects.
- While we do value the convenience brought by the static list `SUPPORTED_HANDLERS` in the main class, we still believe it has to be the controller's own responsibility to tell what command it can handle. To elaborate it, the client may expect controller will upgrade iteratively based on existing operations and thus, rather than supplying a list of handlers in Main for controller, each version of the controller should be encapsulated such that it knows what command it supports. The same applies for the `SUPPORTED_TYPES`.

  The `chooseViewFromArguments` and `parseArgs` are using `Optional` object and is a bit hard to read due to the multiple methods calling. Try to implement these two methods alternatively for better readability.
  - The above suggestions are all based on the baseline that the main class should be short.

Documentation critique

Documentation is well done for this project. All interfaces have documentation what are they designed for; concrete classes were documented with more implementation details; documentation for public methods strictly follow the javadoc coding standard, with purpose statement, required arguments, return type, and throw condition. All of them were described in a straightforward way and can be easily understood. There are also multiple places the brief explanations were provided inside a method for better understanding. One opportunity is to include the documentation for private and protected methods.

1. That is to say, we suggest the program set a restriction (at least for GUI) such that image operations will only consume an image from the library and save it into the library; and load/save has to communicate both the machine & library rather than only one of them. ↩ ↩

2. Refer §Implementation critique - III. Improvement Suggestion-3 to see why controller would have the complete command set. ↩