Thomas and Alex's Image Processing Tool

Description

Welcome :) This project is an image processing tool that allows you, the user, to load, save, and apply various filters on images. Currently, one can only interact with the program through the CLI, but in the future support for GUI will be added.

How to run

For a detailed guide on using our program, see our USEME.md

How It Works

Command handling

Almost all commands include `image-name` and `image-name` arguments. The program uses a smart caching system to remember your friendly names, and if any name provided matches one of the supported file transferers' extensions, it will pick that up and automagically load or save the file to and from the file system. When using aliases, the program will remember the name of the alias, and use that if supplied in further commands.

Command processing flow:

1. A command is received by `ImageProcessorCLI`.
2. A command is delegated for processing to the `ImageProcessorController`, which then attempts to match up this command against all the available commands.
3. If a command is found and accepted, the processing of a command is further delegated to an individual command handler. All the command handlers can be found in `src/controller/handler`. The general idea behind command handlers is to introduce an easy way to add new commands to the program, as adding a new handler would only require adding a dedicated handler and including that in a lilst of available handlers in the main class.
4. A handler will attempt to process the command and its arguments, possible calling on passed `loadImage` and `saveImage` methods for loading/saving images in the file system and/or local memory.

Operations

All operations are contained and classified under `src/mode/image/operation` and must implement `ImageOperation`. All operations must have one public method: `apply` without any arguments, which would apply the current operation on a supplied image and return that image. If there is a need for some additional arguments (for example, the level of brightness to apply), they must be a part of the constructor for that operation.

- `operation.color_transform` applies to all operations that transform the colors but do not change the positions of the pixels.
    - `color_transform.linear` pertains only to linear transformations -- i.e. transformations where the color of the transformed pixel is a linear combination of the initial channel values for that pixel. An example would be the sepia operation.
        - `linear.grayscale` applies to all grayscale operations. While a grayscale operation is almost always a linear operation, we are able to abstract this further, as grayscale operation set the same value to all of a pixel's channels.
    - `color_transform.ranged` applies to all the color transformations that apply a specific kernel filter to the image. An example would be `blur` or `sharpen` operations.
- `operation.geometric` apply to all the transformations that do not modify the color of the pixels, but instead just rearrange their position in the image. An example would be a horizontal flip operation.

The model

There are two main parts to the model: the application state manager and the implementation of images.

- `model.ApplicationStateManager` is an interface to be extended by the specific implementations of application state managers. The current implementation of the application state manager contains and operates on multiple images at the same time. In the future, perhaps, this may contain other data (for instance, user settings).
- `model.image.Image` is an interface that describes all public methods to apply on an image. The purpose here is not to write specific operations; instead, we just have generic add/retrieval/update methods that would later be used by specific operations.
- `model.image.pixel` is a package to define different types of pixels and basic operations on them. Currently, the image processor only supports standard RGB-color pixels, but in the future we plan to add support for RGBA (with alpha values) pixels as well.

How to add new features

Adding New File Type Suppport

To support the transfer of new file types to and from a file system:

1. Create a new `controller.transfer.ImageTransferType` which will recognize the file's type from its path.
2. Write a new `controller.transfer.ImageTransferer` that defines the process for going to and from the file name to the system.
3. Add this pair as an `Entry` to the `Map<ImageTransferType, ImageTransferer> SUPPORTED_TYPES` in `ImageProcessorMain`.

Adding New Commands and Processes

1. Implement the `controller.handler.ImageProcessCommandHandler` with your new command handler.
2. Add an instance of this handler to the `Set<ImageProcessCommandHandler> SUPPORTED_HANDLERS` in `ImageProcessorMain`.

Voila!

Testing

Testing is comprehensive and maintains the same folder structure as the `src` directory. All the tests in `test` directory are unit-tests, dedicated to verifying only specific, single actions per method/class. Coverage reports can be generated through IntelliJ.

Image citation

Photo
by
sanjoy saha
on
Unsplash

Changelog since HW#4

Refactoring

- In hw#4, our controller also maintained the current state of the application, and it was pointed out to us that such design is not efficient. Thus, we now have a dedicated application state manager (which is a part of the model) in `model.ApplicationStateManager` that maintains the state of the application at any given time, and to which the controller delegates storage of images in the local cache. Note that we also slightly modified the structure of the project, adding a top-level folder `model` above `image`, which was previously dedicated to the model.

- We introduced better structure to the operations, adding an `AbstractOperation` parent class that abstracted a lot of the functionality that was previously distributed and repeated between various operations. Furthermore, there are now `AbstractColorTransformOperation` and `AbstractLinearColorTransformationOperation` parent classes. Note that we didn't change any of the interfaces or any of the previously publicly available functionality here.

- We abstracted argument handling by adding everything in `src/arguments`. This structure will allow us to add any new command arguments with ease.

New features

- New file type handling: we added handling for all common file types (.jpg, .png, .jpeg, .bmp, .tif, .gif, .tiff, and .wbmp)
- Sepia color transformation filter
- Blur and sharpen color transformation operations