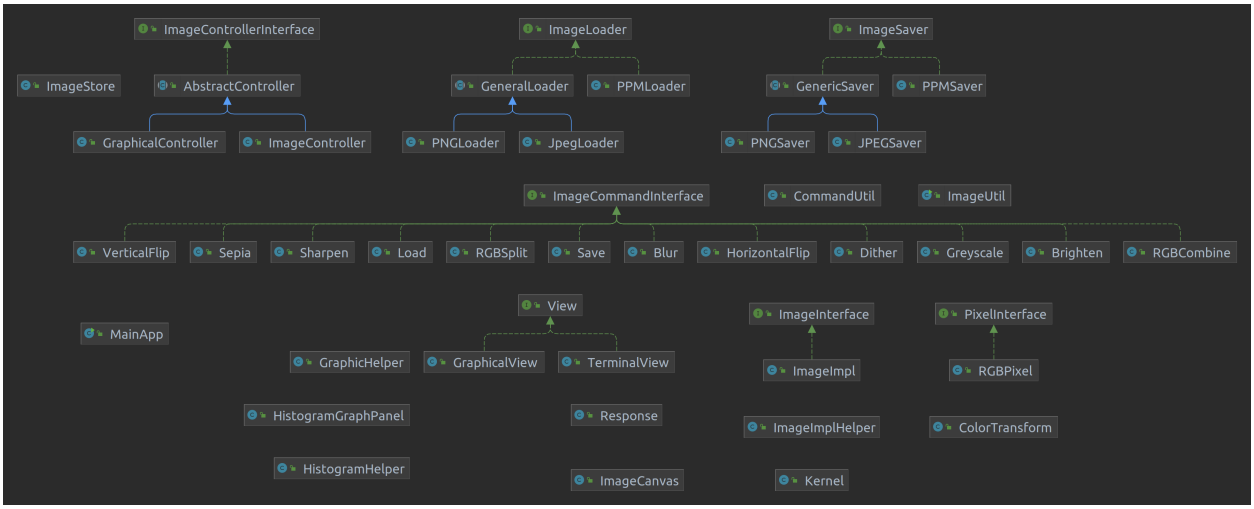
README

This is a text-based application that can perform a set of operations on images. The application is designed with the following key ideas:

1. Extensibility
2. Code-reuse
3. Well-defined separation of Responsibilities

The class diagram of the application:



Purpose of classes by Package - Interface - Implementation

1. imageModel
    a. ImageInterface - The ImageInterface represents the image and the Model. It encompasses all the methods that are operated on the model itself. It does not include methods such as rgbCombine and loadImage which don't require an object to do the operation on.
        i. ImageImpl
        The ImageImpl class is the concrete implementation of the ImageInterface.

In addition to implementing the methods described in the interface we have a few constructors.

The Model is designed such that all method will return new copies and explicitly forbid the modification of the image data contained within once created.

   b. PixelInterface - The Pixel interface represents the individual pixel in the image.
      i. RGBPixel - Concrete implementation, represents a single pixel of type RGB.

2. ImageSaver - Ther interface represents the common methods for all the loaders for any of the supported image formats
   a. GenericSaver - Abstract Class - Implements the loading functionality for all the methods supported by ImageIO class
      i. PNGSaver
      ii. JPEGSaver
   b. PPMSaver - Implements the functionality for saving ASCII PPM images.
3. imageController
   a. ImageControllerInterface - Represents the controller of the application, which manages all the operations of the application based off the input received from the user via the view. The interface only exposes one method which allows the application to be started
   b. ImageController - Concrete implementation of the interface for the application. It operates in the way:
      i. Retrieves input from the view - The view blocks until the user hits enter.
      ii. The input from the view is parsed and delegated to the operationHandler method.
      iii. The operationHandler method will delegate it to the respective method that performs the operation
4. ImageCommandInterface - Provides a common interface for all the commands that are supported as part of the application.

5. imageView
    a. View Interface - The view interface holds the methods for the controller to receive and pass information to and from the user. This may be implemented to support any further type of view at a later stage.
    b. ImageView
        i. TerminalView - The text based implementation of the View Interface. This operates with the idea of the user input being structured to a format that is extensible enough for future commands should be accommodatable in the existing structure.
6. Helpers
    a. Response - Response class is designed to hold information beyond just the intimation of the success or exception from the controller to the view. Possibly allowing for addition of more complex data that may need to be passed on to the View when we move to a GUI
    b. ImageLoader - The interface represents the common methods for all the loaders for any of the supported image formats
        i. GeneralLoader - Abstract Class - Implements the loading functionality for all the methods supported by the ImageIO Class
            1. PNGLoader
            2. JpegLoader
        ii. PPMLoader - Implements the functionality for loading ASCII PPM images.
    c. ImageImplHelper - Provides a singular interface for the controller to load multiple image formats and perform operations that involve multiple model objects, such as RGB-combine.

To Run the program:

Run the app jar to start the application.

- On Windows, both double clicking on the jar and starting from the terminal will work
- On Linux, the jar might be treated as an archive and opened up as such. Please launch from the terminal if this is the case.

Terminal :

```
# Non-interactive Script Mode
java -jar res/CS5010-Assignment6.jar -file path-of-script-file

# EXAMPLE SCRIPT_NAME - res/scripts/script1

# Interactive Text Mode
java -jar res/CS5010-Assignment6.jar -text

# Interactive Graphical Mode
java -jar res/CS5010-Assignment6.jar

#Double clicking also works for running in graphical mode
```

Further information on the commands themselves can be found in the USEME.

Design Updates - Assignment 5

- Design has been updated to allow applying filters(blur, sharpen, other future kernels), applying color transforms (sepia, greyscale, other future color transformations), dithering.
- The program is also capable of handling loading/saving multiple image formats(jpeg, png, ppm etc).
- We also added loader and saver interfaces to improve code reuse and flexibility of the code.
- Instead of directly using a HashMap in our controller to store images being worked on, we created a wrapper model around the HashMap and pass it to our controller. We did this so the same HashMap could be used across multiple controllers.

- Implemented command design pattern for all operations to ensure code-reuse and extensibility

Design Updates - Assignment 6

- Added GraphicalController and pulled out common helper functions into an AbstractController that all controllers can extend from.
- Added GraphicalView that implements the View Interface.
- GraphicalController waits for events from the GraphicalView and runs them accordingly
- Added option to run the program in text mode if need be using the —text argument(in MainApp), default is Graphical mode.
- All functionality from text mode is preserved in the graphical mode apart from running an entire script.
- Used the wrapper ImageStore to contain a stack of the images being worked on, this way we can undo an operation easily.
- Added functionality to get histogram of the current image being worked on
- Added code to get a BufferedImage from ImageInterface model, to display it on the view.

Image Citation:

File:SMPTE Color Bars.svg. In *Wikipedia*
. https://commons.wikimedia.org/wiki/File:SMPTE_Color_Bars.svg