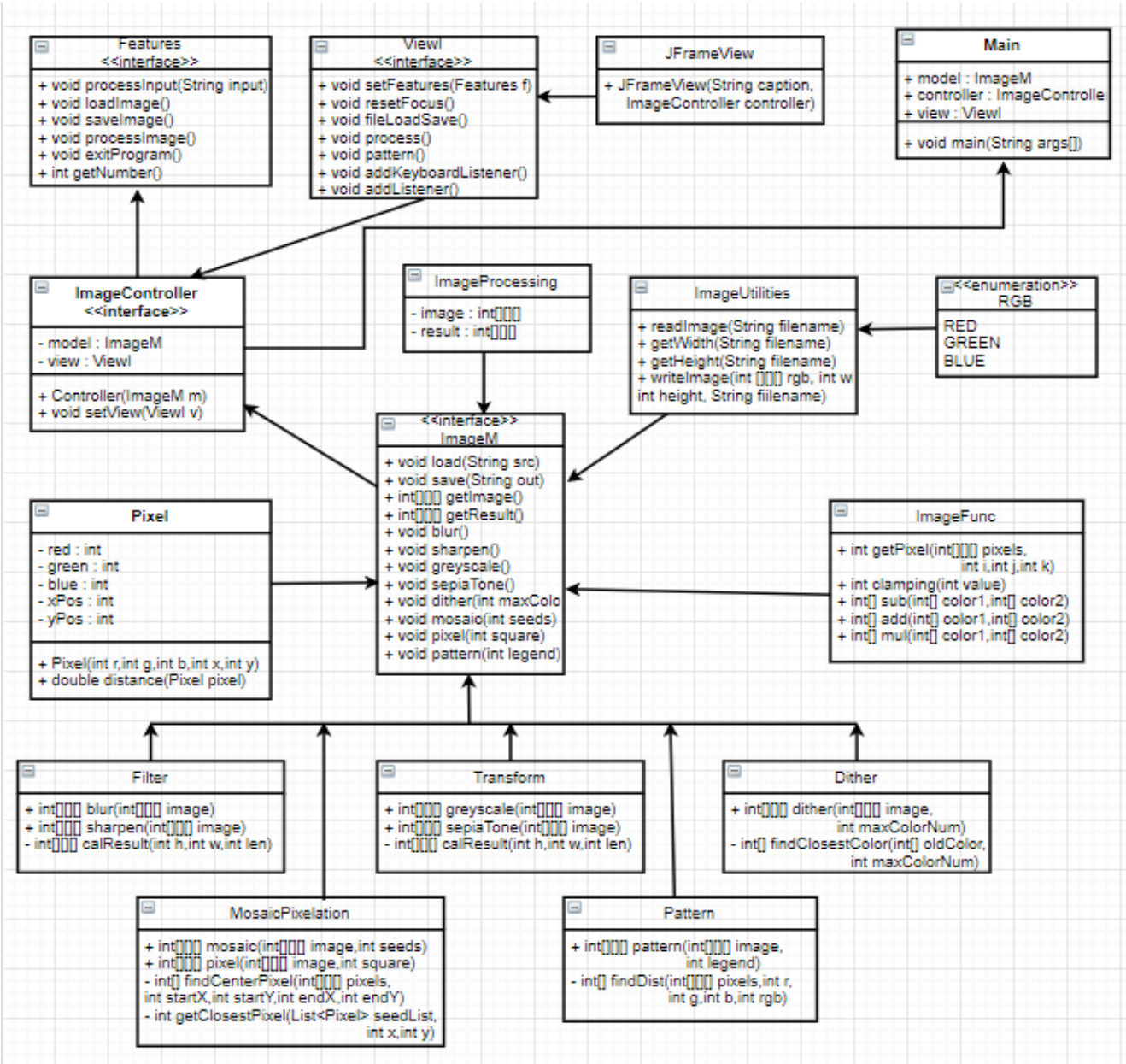
Project 5 -- Cross-Stitch Application Preliminary Design

This project follows the MVC design. Here, ImageM represents the **Model**, ViewI represents the **View** and ImageController represents the **Controller** of the design. The model implements the actual functionalities offered by the program. The view is the part of the program that shows results to the user. The controller takes inputs from the user and bosses the model and view around by telling the model what to do and the view what to show.



Testing plan

The Controller will be tested by creating a mock model and a mock view. Testing with mock view will check the input to the view from the controller and the output to the controller from the view. Testing with mock model will check the input to the model from the controller and the output to the controller from the model.

Testing Controller with MockModel

1. Test image being loaded into the model on input.
   assertEquals("image loaded",log.toString()) // input reaches model correctly

2. Test image processing – blur
   assertEquals("image blurred",out.toString()) // output from model received correctly

3. Test image processing – sharpen
   assertEquals("image sharpened",out.toString())

4. Test image processing – greyscale
   assertEquals("image greyscaled",out.toString())

5. Test image processing – sepiaTone
   assertEquals("image sepiatoned",out.toString())

6. Test image processing – dither
   assertEquals("image dithered",out.toString())

7. Test image processing – mosaic
   assertEquals("image mosaic",out.toString())

8. Test image processing – pixelation
   assertEquals("image pixeled",out.toString())

9. Test image processing – pattern
   assertEquals("image pattern generated",out.toString())

10. Test image being saved after processing.
    assertEquals("image saved",out.toString())

Testing Controller with MockView

1. Test image being opened
   assertEquals("image opened",log.toString)
2. Test on menu option "blur" click, blur image to be displayed
   assertEquals("blur image displayed",out.toString())

3. Test on menu option “sharpen” click, sharpened image to be displayed
   assertEquals(“sharpened image displayed”,out.toString())
4. Test on menu option “greyscale” click, greyscaled image to be displayed
   assertEquals(“greyscaled image displayed”,out.toString())
5. Test on menu option “sepiatone” click, sepiatoned image to be displayed
   assertEquals(“sepiatoned image displayed”,out.toString())
6. Test on menu option “dither” click, dithered image to be displayed
   assertEquals(“dithered image displayed”,out.toString())
7. Test on menu option “mosaic” click, mosaic image to be displayed
   assertEquals(“mosaic image displayed”,out.toString())
8. Test on menu option “pixelation” click, pixeled image to be displayed
   assertEquals(“pixeled image displayed”,out.toString())
9. Test on menu option “pattern” click, pattern to be displayed
   assertEquals(“pattern generated”,out.toString())
10. Test on menu option “save” click, image should be saved to destination folder
    assertEquals(“image saved to given destination”,out.toString())
11. Test on menu option “exit” click, application closes
    assertEquals(“application closed”,out.toString())