ARNAV SAWANT AND AHAAN CHAUDARY CODE REVIEW

Design Critique:

We would like to praise the design overall. We found the design to be very easy to change and implement the mosaic function in. After the Mosaic funcion was created it only took 10-20 minutes to change the controllers and view to account for the Mosaic function. The interfaces we were provided were very flexible, especially the transform interface which we extended for the Mosaic function. The Model, View and Controller are well separated, and there is only one model for both the GUI and the text based View/Controller which is amazing. One place where the model can be improved is that instead of having a method for each image function (flip, brighten, darken etc.), there can be just one method called “Transform” or something similar which takes in an Object of the “Transform” interface and then delegates to the object which is passed through to actually perform the function. This would have made it so we did not have to touch the model at all and even if we had to implement 100 new image functions, the model would still be unchanged. One more suggestion that we had was that the code could have been made more readable and cleaner by implementing an Image class and interface. This would have abstracted a lot of the code where you have to check whether the image is a PPM or of another type. In this case you would just delegate to the type of Image that implements the Image interface. Also the providers code would have benefited from using a similar approach with command design pattern in the controller. Overall the design was easy to work with to add the Mosaic function.

Implementation Critique

We would like to praise how intuitive the GUI that the providers created is. You could very well hand the GUI to a Elementary or Middle school student who has limited experience with computers and the student would be able to use the GUI. We also thought it was impressive that the providers went above and beyond and implemented a GUI in which you do not have to only be working on one image at a time but rather can work with multiple images. However, there were many issues that we found where the code does not work properly in implementation. The providers did not provide any tests for any of the code they gave. When asked about the tests the providers responded that they had not tested anything at all. A lot of the issues discussed later could have likely been caught if the code was well tested, so the #1 suggestion we would make is to test each class for multiple different cases/scenarios.

One issue we found with the implementation is that the GUI does not scale to the proper dimensions of the screen it is being opened on and is not scrollable. The GUI

needed to be opened on a monitor for us to be able to see all the elements of it (specifically the enter button).

Another issue we found was that the code was not able to take in “-text” or “-file” as a command line argument. The providers documented this in the Questionnaire they sent us (which we have submitted in the res folder). We would suggest adding cases for these command line arguments in the Go class in the controller to fulfill the requirements of the assignment.

The biggest issue we found with the provider's code was that the load does not work properly. For whatever reason, the GUI is only able to load the crab image that the providers provided in their res file. We tried for many hours to fix this bug, but because no tests were provided to us, and that the height and width are always passed through methods (this is where the bug seems to be) we were not able to fix this bug. Our recommendation on this is for the providers to, in the future, test their code at every step of the way so that any bugs can be fixed right when they occur. Because the code is not tested it became impossible for us to track down where the bug was because we do not know which parts work fully, so the bug could theoretically be anywhere (including the model). If the program was well tested, or even tested at all, the providers would have likely found this bug with the load and been able to fix it before it even came to us. Even if they had not, it would have been much easier for us to pinpoint exactly where the bug came from and how to fix it. So we would recommend that at this juncture, the provider go through and test all of the classes.

Documentation Critique

We would like to praise the documentation. We were able to understand essentially the purpose of every method as well as every class and interface. The great documentation certainly made it easier to work with the code and we thank the providers for the detailed documentation.

Design/Code strengths

Strengths of the design is definitely how the providers designed their interfaces and classes and the flexibility of their interfaces. It made it very easy to add onto and was very readable and easy to understand. With regards to the code, all variables and classes were suitably named which again made it easy to implement.

Design/Code limitations

The limitations are those that are discussed above. The main limitations are that Load does not work properly, and the program does not support -file or -text. Almost all of the limitations could have been fixed/caught if the providers tested their code. Without tests you are relying on the “eyeball test.” We suggest that in the future the providers make sure to always thoroughly test their code.