Project Assignment Report
Design of Software System

Jasper Marien
Romain Carlier
Robin Vanden Ecker
Daniel Andrés Pérez Pérez

I. Iterations

A. Iteration 2

We can use, like the assignment mentioned, a configuration file which will be read by the class ProxyThread.java. The configuration file could contains a simple txt file which each row has a pair (image_capability, is_actived). Then, instead of having the hard code we did for iteration 1, we read the config.txt file and transform the image accordingly.

Two possible implementation came up to my mind with out using a switch:

1) we can add a new class called ImageTransformation.java, that actually has all the different capabilities for the images (flip images, image enhancing, water mark), for each capabilities we will have one method. ProxyThread.java will have an instance of ImageTransformation.java and will use its methods in the loop of the reading file. Here it becomes handy to use the attribute of image_capability in the config.txt has actually the name of the method. For instance, if ImageTransformation.java has the method flip_image(), then the config.txt would have the pair (flip_image, true).
   - **Pros**: Since we assign the capabilities to a single class there is no coupling between ImageTransformation.java and ProxyThread.java. ImageTransformation.java can easily be used in other classes, the cohesion is actually kept by the nature of the single class.
   - **Cons**: As a whole, the ImageTransformation.java contains a well defined responsibility, transform/add capability to a image, however every method differ in the algorithm itself, all of them has the same input and output (an image). Additionally, if we continuously add more and more capabilities, ImageTransformation.java will have many ”similar” methods, in terms of input and output, and if can become very extensive, highlighting that also many of the methods are potentially not used.
2) We can take advantage of polymorphism, we can add an abstract ImageTransformation.java with one method transform_image(), then we create a concrete transformation class for every capability (in this case flip_transformation.java, image_enhancing_trasformation.java, water_mark_transformation.java) and then we implement the abstract method with the proper algorithm. Here becomes handy to store the name of the concrete transformation class in the config.txt file, for instance (flip_transformation, true) and, finally, we use the unique transform_image() method.
   - **Pros**: The responsibilities are granular, each concrete class has its own algorithm, the concrete classes do not have useless methods. Whenever we want to add a new capability, we only need to create a new concrete class of the abstract class and we will keep the cohesion and the granular responsibilities with out adding complexity and potential useless methods.
   - **Cons**: Although ImageTransformation classes can work together and can be used by other classes, the internal estructure increase the coupling due to the inheritance mechanisms used.

B. Iteration 3

C. Iteration 4