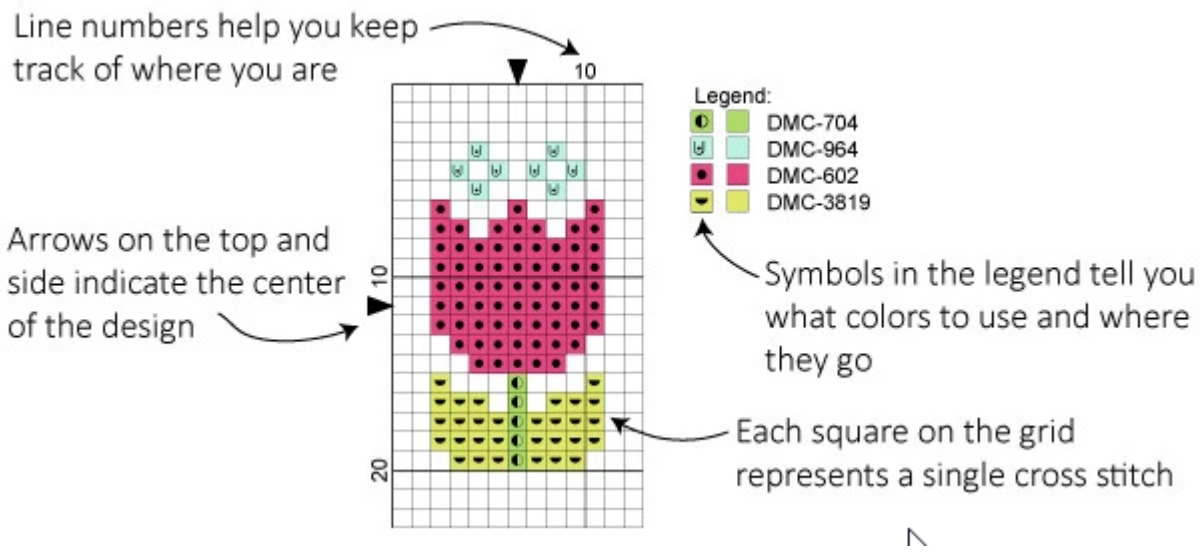
Project 4 -- Cross-Stitch Controller

Due Friday by 11:59pm **Points** 100 **Available** after Mar 8 at 12am

Cross-Stitch Controller

Cross-stitch (https://en.wikipedia.org/wiki/Cross-stitch) is a very old form of counted thread embroidery which has been found all over the world since the middle ages. It is one of the easiest forms of hand embroidery to learn. It gets it's name from the X-shaped stitches that are done on an even and open weave fabric. A pattern consists of a chart that tells you everything you need to know about where to stitch and what color to use:



(Image from **https://stitchedmodern.com/blogs/news/a-beginners-guide-to-cross-stitch#** (https://stitchedmodern.com/blogs/news/a-beginners-guide-to-cross-stitch#))

In this example, the pattern uses both a color and a symbol to tell you which color thread (called *floss*) to use. In this project, we will extend the image model that we developed in the last project by adding to two algorithms for breaking down an image into "chunks", and then use one of these to develop a controller for creating cross-stitch patterns.

1 -- Image Mosaic

One such image "chunking" approach that you may have seen is an effect that gives a mosaic effect. **Mosaics** (https://en.wikipedia.org/wiki/Mosaic) are a popular art form where small pieces irregularly shaped pieces of color stone, glass, or ceramic are used to construct an image (check out **Mosaic Park in Vancouver, Canada** (https://www.insidevancouver.ca/2010/08/27/hidden-art-in-the-ground-mosaic-park/)).

To simulate mosaics on a computer, an image can be "broken down" into such shapes, by choosing a set of points in the image (called *seeds*). Each pixel in the image is then paired to the seed that is closest to it (by distance). This creates a cluster of pixels for each seed. The color of each pixel in the image is replaced with the color of its seed pixel.

The seeds can be chosen in a number of ways. The simplest method (that you will implement) is to choose the seeds randomly (i.e., choose random pixel locations from the image). The pictures below illustrate the effect of mosaicing:

Original Image:

Mosaic with 6600 seeds:

Mosaic with 1650 seeds:

Mosaic with 570 seeds:

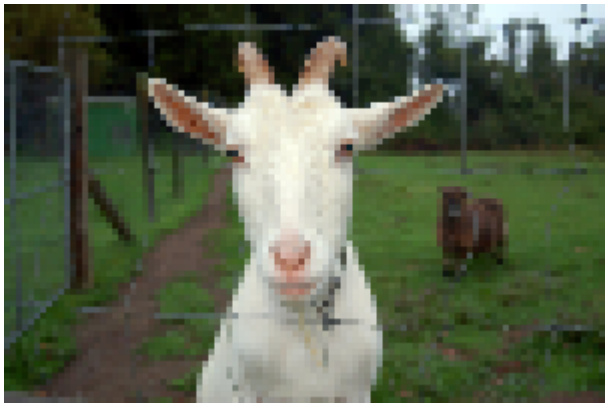
2 -- Image Pixelation

Another way to chunk an image is by chunking the image into regular squares across the rows and columns. This chunking method produces what many think of as an equivalent to *pixelating* the image. The big difference between this effect and resizing the image is that the resulting image is the same size as the original image, and the color of each pixel in the original image is replaced with the color of the pixel at the center of its square creating *super-pixels*

To do this we decide how many super-pixels we want to have across the width of the image and then create evenly sized squares across. Each super-pixel should be no more than one pixel different in width and/or height of another super-pixel in the image. The pictures below illustrate the effect of pixelation:

Original Image:

Pixelation with 100 squares across:



Pixelation with 50 squares across:

Pixelation with 30 squares across:

3 -- Pattern Generation

Generated chunked images makes it possible to convert an image that has many pixels into one that has fewer pixels without actually changing the number of colors that the image uses. This comes in handy when we want to create a cross-stitch pattern from an image.

To create a cross-stitch patterns for an image, start by pixelating the image and then map each super-pixel in the image to a floss color by calculating the "closest" color of available floss. Use this **DMC floss to RBG value conversion chart (http://my.crazyartzone.com/dmc.asp)** to do this.

3.1 -- Calculating Closest Color

The **distance between two colors (https://en.wikipedia.org/wiki/Color_difference#cite_note-euc-1)** can be measured in a variety of ways, but not all of them are created equal. For example, using Euclidean distance is one way but it often suffers from color distortion similar to the way grayscale did because of the way that humans perceive color. To this end, we will a better approximation called the *redmean*:

$$\Delta C = \sqrt{\left(2 + \frac{\bar{r}}{256}\right) \times \Delta R^2 + 4 \times \Delta G^2 + \left(2 + \frac{255 - \bar{r}}{256}\right) \times \Delta B^2} \text{ where } \bar{r} = \frac{R_1 + R_2}{2}$$

and R, G, and B represent the values between 0-255 for each channel respectively.

3.2 -- The Pattern Specification

The result of creating a image pattern is text. Each pattern should have 3 parts:

1. The *size* of the pattern expressed a the width and height
2. An *image map* that consists of an n x m grid of symbols where each symbol represents the single floss color that should be used to stitch that super-pixel.
3. A *legend* that lists the DMC floss used in the pattern in numeric order along with the symbol used to represent that floss color in the image map.

We provide **this example pattern** **(example-pattern.txt)** as an example of the pattern for the image at the very top of this project description. Your patterns do not need to look exactly like this but it does need to have all three parts.

4 -- Batch Controller

Until now the *driver* of your program was a meaningless `main` method that did its best to demonstrate how to use your model. In this assignment we want to create a *controller* that will respond to user commands. One way to do this is to prompt the user for input from the keyboard and process each command one at a time. While this works, a better way to deal with user input would be to allow the user to load an image, apply various effects to it, and save the result by placing the input in a *batch file*. For example:

```
load goat.png
dither 2
save goat-dither.png
load goat.png
blur
save goat-blur.png
load goat.png
pixelate 50
pattern
save goat-pattern.txt
...
```

A user could provide these commands into file and then provide it to the program. The program would then read the batch file one line at a time, process it, and execute the commands that it contains.

5 -- What to do

In this assignment you must *enhance the model* that you implemented in the previous project and *add a synchronous controller* to your program that supports a script like the one shown above. Take the time to design the interfaces/classes that are required for your controller. Be sure to specify appropriate constructors, what methods and fields those classes have, the visibility of the methods and fields, and the relationships between. Capture this information along with any changes you have decided to make in

the model in a UML diagram. add a testing plan for the testing the controller, convert it to a pdf, and submit it to Canvas using the **Project 4 -- Preliminary Design** assignment.

Your program must be able to:

1. Include all the features from part 1 of this assignment including blur, sharpen, grayscale, sepia, and dithering.
2. Create mosaic and pixelations of images.
3. Create a cross-stitch pattern for an image.
4. Support the user entering commands from a file by executing them (although in the future it may not just be a file!). The file must provided to the program using command-line arguments (the parameter of the `main` method). To do this, when running your `main` method from your IDE, you will need to change the **Run Configuration** by adding the file into the **program arguments**. if you type "input.txt" here, save the changes and run you program, then "input.txt" is available as `args[0]` in `public static void main(String[] args)` method. For this to work, this file should be inside your project folder (where src/ and test/ are).
   - In Eclipse, you can follow these **directions for using command line arguments (https://www.cs.colostate.edu/helpdocs/cmd.pdf)**
   - In IntelliJ, you can follow these **directions for using command line arguments) (https://stackoverflow.com/questions/2066307/how-do-you-input-commandline-argument-in-intellij-idea)** (look at the 2nd and 3rd ranked responses ranked 86 and 59 respectively)
5. You are not expected to support the exact commands shown in the script above. This was just an illustrative example of the kinds of things a user should be able to do. You may choose different words, or syntax. Whatever words and syntax you choose should be reasonable for a user to provide. **Do not try to support expressive or intuitive syntax: start with basic, easy-to-parse commands and enhance if you have time.**

6 -- Documentation

We expect your code to be well-commented using well-formed English sentences. The expectations are:

- Each interface and class contains a comment above it explaining specifically what it represents. This should be in plain language, understandable by anybody wishing to use it. Comment above a class should be specific: it should not merely state that it is an implementation of a particular interface.
- Each public method should have information about what this method accomplishes (purpose), the nature and explanation of any arguments, return values and exceptions thrown by it and whether it changes the calling object in any way (contract).
- If a class implements a method declared in an interface that it implements, **and** the comments in the interface describe this implementation completely and accurately, there is no need to replicate that documentation in the class.
- All comments should be in `Javadoc`-style.

7 -- Create a JAR file of your program

In order to make your application easier to run, you are required to create and submit a `JAR` file:

- Directions for doing this in Eclipse can be found at **this link (https://www.codejava.net/ides/eclipse/how-to-create-jar-file-in-eclipse)** .
- Directions for doing this in IntelliJ can be found at **this link (https://www.jetbrains.com/help/idea/packaging-a-module-into-a-jar-file.html)** .

8 -- What to submit

Log on to the **Handins submission server (https://handins.ccs.neu.edu/)** and upload a ZIP file of your assignment. Your ZIP file should contain three folders: src/, test/, and res/

- All your code should be in src/.
- All your tests should be test/.
- Your design documents should still be in res/. You should include your design documents from the part 1 of this project as well as a preliminary revised design document for this part of the project.
- Submit a correct JAR file in the res/ folder. We should be able to run your program from this jar file.
- In the res/ folder, also submit at least two images and their corresponding mosaic and pixelation results (**please limit the size of any generated image so that the width + height is less than 40Kb to limit the size of your submission to 5Mb).**
- In the res/ folder, also submit cross-stitch patterns for your two example images.
- Submit a ***README.md* file that documents your program**.

It is your responsibility to ensure that you are legally allowed to use any images, and your citation should be detailed enough for us to access the image and read its terms of usage. If an image is your own photograph or other original work, specify that you own it and are authorizing its use in the project.

All images must be "work-appropriate" and must not violate the Code of Student Conduct.) (http://www.northeastern.edu/osccr/code-of-student-conduct) .

10 -- Criteria for grading

Your projects will be assess in three different ways:

1. Completeness, correctness, design, and testing will be assessed through the completion of a *self-evaluation* similar to the ones you have been doing for lab assignments. In the self-evaluation, you will be asked to answer a series of questions about your code, provide pointers (tags) into your code, and provide explanations of your answers and/or implementation. The pointers into your code will allow us to confirm your answer since it will show us where to look in your implementation. Self-evaluations are worth ~50% of the project grade.

2. Style and whether you are following good programming principles will be assessed on the Handins server via your submission. Style is assessed automatically by Handins and good programming principles will be assessed manually. To see details of what will be manually assessed, refer to the **Manual Grading Rubric**. The submission and the manual grading of said submission is worth ~20% of the project grade.
3. Finally, you are asked to defend your design and/or implementation choices in a meeting with your instructors for the last ~30% of the project grade including submitting a reflection of your meeting experience.