COMP 1600 Fall 2025

Project 5: A Succession of Images

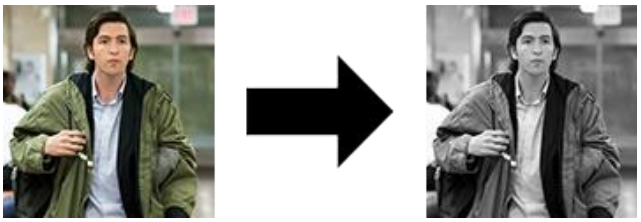
Due by: Wednesday, December 3, 2025 by 11:59 p.m.

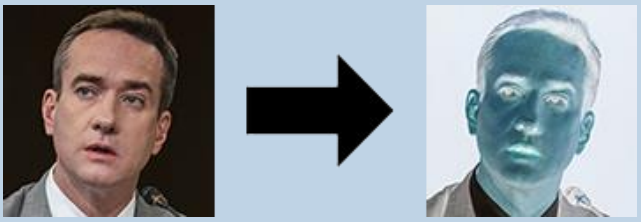
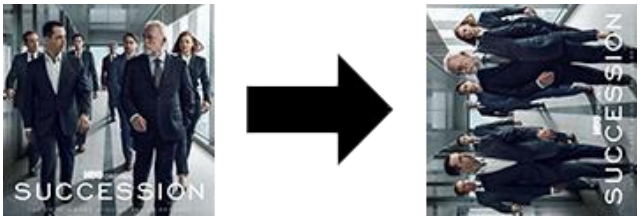
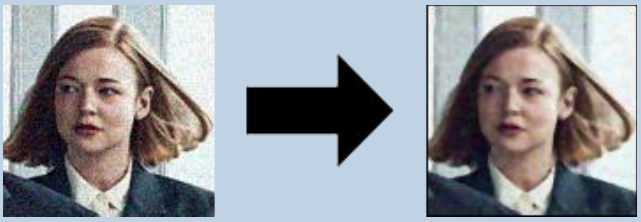
"Uh oh..."

Cousin Greg just tripped, knocking a pile of evidence into a bucket of benzene! Was Greg trying to sabotage the investigation, or was he just clumsy? It doesn't matter now since the chemicals have splashed all over countless documents and images. The effects of the benzene are unpredictable, and the print and pictures on the documents are doing all kinds of strange things: stretching, rotating, turning bizarre colors, and getting covered with random noise. As a lowly intern at the Department of Justice, you must use Java to fix the problem.

The Mission

In order to reconstruct the images, we need you to write a program that can read in an image file using the `Picture` class and perform a sequence of modifications consisting of six operations, depending on which of the following seven commands (including quit) is given by the user.

User Command	Operation	Example
1	Grow the image to be double in size using the **`grow()`** method.	
2	Make the image grayscale using the **`grayscale()`** method.	

3	Invert the image using the **`invert()`** method.	
4	Rotate the picture to the left 90 degrees using the **`rotateLeft()`** method.	
5	Apply a median filter to the image using the **`medianFilter()`** method.	
6	Display the image using the `Picture` object's `show()` method.	
7	Quit the program.	

Specification

Your class should be called `Editor`. In addition to the **`main()`** method, it should contain six static methods with the following signatures:

1. `public static Picture grow( Picture p )`
2. `public static Picture grayscale( Picture p )`
3. `public static Picture invert( Picture p )`
4. `public static Picture rotateLeft( Picture p )`
5. `public static Picture medianFilter( Picture p )`
6. `public static void sort( int[] values )`

When you create your project, you will need to add the `Picture` class to it. Once you have created such a class, delete everything inside it and paste in the code from **Picture.java**.

The `main()` Method

When your program starts, it should prompt the user for a file to open. Then, it should run a loop, asking the user to input the command to alter the image. Legal commands are 1 through 6, with 7 being quit.

Sample output from `main()` might look like the following:

```
What picture would you like to edit? table.jpg

Operations
1. Grow
2. Grayscale
3. Invert
4. Rotate Left
5. Median Filter
6. Display
7. Quit
Enter command: 1

Operations
1. Grow
2. Grayscale
3. Invert
4. Rotate Left
5. Median Filter
6. Display
7. Quit
Enter command: 3

Operations
1. Grow
2. Grayscale
3. Invert
4. Rotate Left
5. Median Filter
6. Display
7. Quit
Enter command: 4

Operations
1. Grow
2. Grayscale
3. Invert
4. Rotate Left
5. Median Filter
6. Display
7. Quit
Enter command: 4
```

```
Operations
1. Grow
2. Grayscale
3. Invert
4. Rotate Left
5. Median Filter
6. Display
7. Quit
Enter command: 5

Operations
1. Grow
2. Grayscale
3. Invert
4. Rotate Left
5. Median Filter
6. Display
7. Quit
Enter command: 6

Operations
1. Grow
2. Grayscale
3. Invert
4. Rotate Left
5. Median Filter
6. Display
7. Quit
Enter command: 7
```

This sequence of operations grew the image, inverted it, rotated it left twice, ran a median filter, displayed the image, and then quit. You can find the original image **here** and the final image **here**.

The `grow()` Method

Make a new `Picture` object whose size is double the input in both height and width directions. Then, copy the `Color` values from the original `Picture` to the new one. You will have to make 4 pixels have the same `Color` value as 1 pixel did in the original.

Then, return the new `Picture` object.

The `grayscale()` Method

Make a new `Picture` object the same size as the input object. Then, make the `Color` value for each of the pixels in the new `Picture` the grayscale version of the corresponding one in the original. To do so, set the R, G, and B values of the `Color` object for the new pixel all to the same

value. That value should be: .3R + .59G + .11B, rounded to the nearest integer, where R, G, and B are the color values from the original pixel.

Then, return the new `Picture` object.

Note that when the red, green and blue components are all the same, the color is a shade of gray. For example (0,0,0) is black, (255, 255, 255) is white, and (128, 128, 128) is a 50% gray.

The `invert()` Method

Make a new `Picture` object the same size as the input object. Then, make the `Color` value for each of the pixels in the new `Picture` the inverted version of the corresponding one in the original. To do so, if a pixel's color values are (R, G, B), the new color value should be (255 - R, 255 - G, 255 - B).

Then, return the new `Picture` object.

The `rotateLeft()` Method

The goal is to rotate the picture to the left 90 degrees. Make a new `Picture` object whose width and height are the same as the height and width, respectively, of the input object. Then copy the pixels from the original `Picture` to the new one in the correct order. As with all the other transformations, you will need two nested `for`-loops, but the meaning of row and column will be switched from the original to the new `Picture`.

Watch out! The order that you fill in columns or rows might be different after a rotation. Draw a picture with a very small sample image in it, perhaps a 2 x 3 or a 3 x 5.

Then, return the new `Picture` object.

The `medianFilter()` Method

To goal is to apply a median filter to the picture. You may want to read up on the idea of a **median filter** at Wikipedia. The purpose of a median filter is to reduce noise. One way to reduce noise is to blur an image by making it a weighted average of itself and its neighbors. Doing so destroys a lot of edge information and makes the image much less sharp. An alternative is to replace the pixel value with the median (rather than the average) of the surrounding pixels. Of course, we can't find the median of full color values. Instead, we take the median of the red, green, and blue components separately.

To implement a median filter, make a new `Picture` object the same size as the input object. Then, for each pixel that is not in the top row, bottom row, left column, or right column, record the values

of the R, G, and B values of all its neighbors (and itself) in three separate arrays. Think of the pixel and its neighbors as a 3 x 3 grid, except of course, there are three 3 x 3 grids, one each for R, G, and B. We want to take those three grids and put the elements into three arrays.

The easiest way to copy these values into arrays is with another pair of nested `for`-loops **inside** the loops you are already using to iterate over the width and height of the `Picture`. The first of these two loops should iterate from the column before the current column to the column after the current column, and the second loop should iterate from the row before the current row to the row after the current row. That's the reason that you don't touch the pixels in the top row, bottom row, left column, or right column: There would be a risk of accessing an element outside the bounds of the `Picture`. Because you don't set the values of these edge pixels, your filtered image will have a black border: Don't worry about that.

After copying the values from the neighbors of the pixel, you should have 3 arrays with 9 elements in each one, containing all 9 values for red in the first one, all 9 values for green in the second one, and all 9 values for blue in the third one. Next, sort each of these three arrays using the **`sort()`** method. Finally, set the value of the pixel in the new `Picture` to be a `Color` object whose red, green, and blue values are the median value (element 4) from these three arrays.

Finally, return the new `Picture` object.

Note: A median filter can be a subtle effect in some situations. The example image above does not give a clear view of the change that happens. For a better example, look first at the full-size noisy image **noisy.jpg** and the filtered version **filtered.jpg**.

The `sort()` Method

This method simply sorts an input array of `int` values. It is used as a subroutine by the **`medianFilter()`** method. Because you only care about the median value, it doesn't even matter if you sort in ascending or descending order. Feel free to use Bubble Sort or Selection Sort.

Image Files

The following image files are provided for your use. Of course, you should feel free to download other images from the Internet for testing. Any image file with a `.jpg`, `.jpeg`, or `.png` extension should work.

- **connor.jpg**
- **filtered.jpg**
- **greg.jpg**
- **kendall.jpg**
- **logan.jpg**
- **noisy.jpg**
- **roman.jpg**
- **siobhan.jpg**
- **table.jpg**
- **tom.jpg**

Turn In

Your IntelliJ project should be called `Project5`, but the class you create inside should be called `Editor`. Upload the `Editor.java` file from the `Project5\src` folder wherever you created your project to **Brightspace**.

All work must be submitted before Wednesday, December 3, 2025 by 11:59 p.m. unless you are going to use a grace day.

All work must be done individually. You may discuss general concepts with your classmates, but it is never acceptable for you to look at someone else's code. Please refer to the course policies if you have any questions about academic integrity. If you have trouble with the assignment, I am always available for assistance.

Grading

Your grade will be determined by the following categories:

Category	Weight
Growing images	15%
Making images grayscale	10%
Inverting images	10%
Rotating images	15%
Applying a median filter	25%
Displaying images	5%
Menu and output formatting	10%
Style and comments	10%

Under no circumstances should any student look at the code written by another student. Tools will be used to detect code similarity automatically.

Code that does not compile will automatically score zero points.