CSC110 Assignment 6: Two Dimensional Arrays and File I/O

Objectives

Upon completion of this assignment, you need to be able to:

- Pass `parameters` and `return` values using `static` methods.
- Pass arguments from the command-line to the `String[] args` array.
- Begin handling input and output files and using `try-catch`
- Implement Java code that handles *two-dimensional* (`2D`) arrays.
- Extend previously written code.
- Continue building good programming skills.

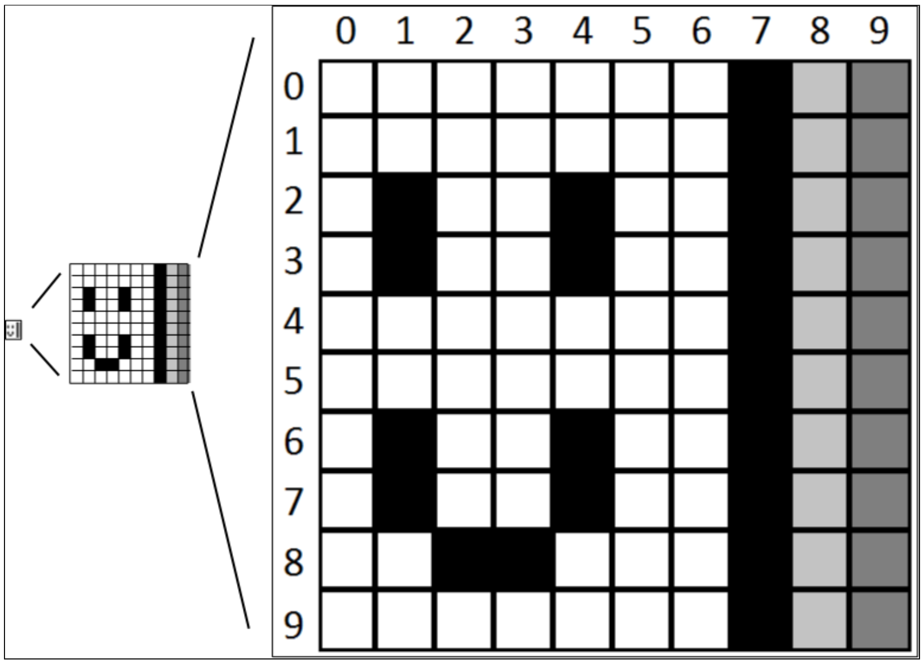
Introduction

One well-known set of tasks for computers today is the manipulation of images. Your work for this assignment is to add additional methods to the class `ImageConversions`. Each execution of the program will perform the following:

- An `ASCII`-art transformation,
- A *scaling* of an image,
- Two *reflections*, one about the `x`axis, the other about the `y`-axis,
- An *inversion*, which creates the negative of an image, and
- A *rotation*, which rotates an image by $90°$, in a clockwise direction.

For this assignment, you will be using the command-line interface to pass in the names of the input and output files, as well as the image manipulation operation to use.

To illustrate how the data in an image file is represented in a 2D array of integers, consider the very small image below that is 10 pixels wide by 10 pixels high to the far left of the following image:



If we really zoom in on this image, we can see that the image is a matrix of pixels, each representing some shade of gray. A 2D array of integers with values ranging from 0 to 255 to represent the above image can be constructed in Java. Suppose this image was in a file called `happyFace.jpg`. If the following Java statement was included in the `main` method of `ImageConversions`:

```
int[][] image = readGrayscaleImage("happyFace.jpg");
```

then `image` will contain integers where a gridcell that was white in `happyFace.jpg` has the value 255 in same place in the array. Likewise, a black cell will produce a value of 0 in the array.

Quick Start

(1) Download this pdf file and store it on your computer.
(2) Create a directory / folder on your computer specific to this assignment. for example, `CSC110/assn6` is a good name.
(3) Save the file called `ImageConversions.java` in your directory, as well as the `computer.jpg` image file. Fill the source code with the class name and the methods outlined in the specification document.

(4) Start with the easiest methods first. See the following detailed instructions below for some tips.
(5) Complete each method and test it thoroughly before moving onto the next one. Compile and run the program frequently; it is much easier to debug an error shortly after a successful and error-free run.
(6) The `args` array in the `main` method is used to relay input from the intelligent and well-behaved *user*, who runs the program. The user provides the name of the input image file to read and the output image to create; the user also specifies the conversion to apply to the image.
(7) Inside the `main` method, begin by calling the completed `readGrayscaleImage` method, to retrieve the 2D array of integers that represents the input image file. It is this array that you will be copying and/or manipulating in the various methods. Except for the case when the array is stored as an `ASCII` image file, the modfied 2D array is sent, as input, to the completed `writeGrayscaleImage` method. This method produces the new jpg image file.
(8) The easiest method to start (in our opinion) is the `invert` method. We have provided the algorithm in the detailed section to get you started. Once you have this working, when you run the program, a new `jpg` file will appear in the directory. To see it, drag it into an open browser and see the inverted image.

Detailed Instructions

The specifications document details the *what* for each of the methods. In each of the following sections, we also provide information on the *how* of each method.

In the following subsections, we provide a detailed description of how to approach the implementation of each of the methods named in the `ImageManipulation` class:

main

All of the information needed by the program will be passed in through the `args` array when the program is executed. (**No Scanners allowed.**) Supposed you have run the following from the console:

```
javac ImageConversions.java
java ImageConversions computer.jpg invertTest.jpg invert
```

The `main` method extracts all the words after `ImageConversions` (the Java filename) and stores it in the `args` array. In the above example, `args` is an array of length 3 and contains {"computer.jpg", "invertTest.jpg", "invert"}. The `main` method needs to perform the following tasks:

- Check to make sure that the `args` array has at least 3 elements in it,
- set up the necessary variables: `arg[0],arg[1],arg[2]` as the inputfilename, the outputfilename, and which method to call (in this case `invert`),
- call the `readGrayscaleImage` with the name of the input file, which
- returns the image as an array of numbers,
- call the `invert` method, which
- returns an array with different numbers, and
- call the `writeGrayscaleImage` with the name of the output file.

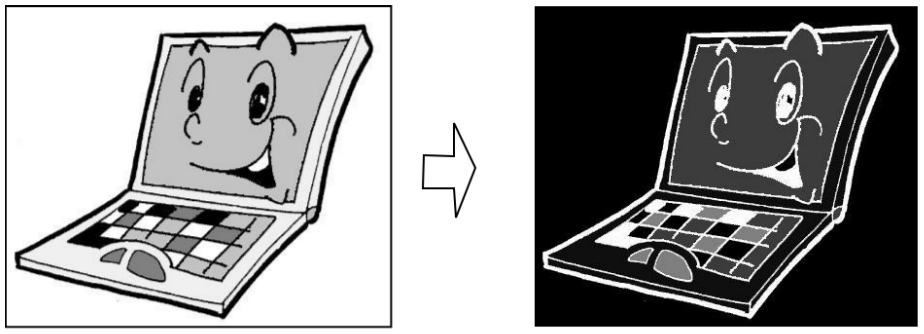
Open the new file to see the results!

Every order to run the program will be executed in the same fashion:

```
java ImageConversions <inputFile> <outputFile> <operation> <optional extras>
```

invert

The figure below illustrates what happens when `main` interprets the input and calls `invert`:



computer.jpg invertTest.jpg

Each number that represents a grayscale is *inverted*, so 0 becomes 255, 1 becomes 254, and so on, until the last number 255 becomes 0.

To get you started, consider the following algorithm, written in easy-to-transfer *pseudo-code*:

Input:
- $image$: A 2D array of integers of the finite set $\{0 \ldots 255\}$
  representing gray-scale colours The number of rows is denoted by $rows$, while the number of columns is denoted by $cols$.

Output:
- A 2D array of integers of the finite set $\{0 \ldots 255\}$,
  where each cell contains the inverse. $0 \to 255, 1 \to 254, \ldots, 255 \to 0$.

```
$N \leftarrow 255$;
$result \leftarrow$ new int[$rows$][$cols$];
for ( $i \leftarrow 0$; $i < rows$; $i \leftarrow i + 1$ ) {
    for ( $j \leftarrow 0$; $j < cols$; $j \leftarrow j + 1$ ) {
        $result[i][j] \leftarrow (N - image[i][j])$;
    }
}
return $result$;
```

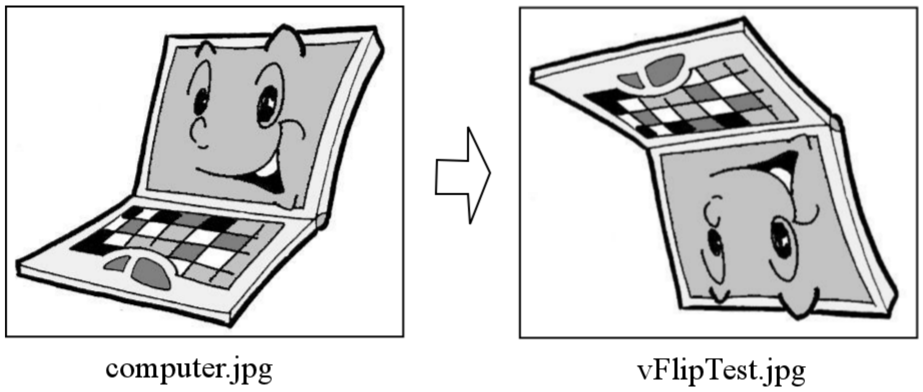


FIGURE 1. verticalFlip example

verticalFlip

The `verticalFlip` method *flips* the image vertically by rotating it vertically. The running order:

```
java ImageConversions computer.jpg vflipTest.jpg verticalFlip
```

performs the action illustrated in FIGURE 1:

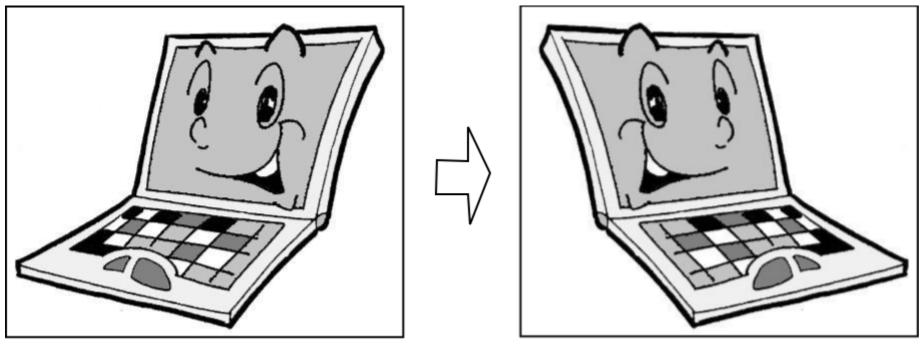


FIGURE 2. horizontalFlip example

Programming notes:

(1) This method reverses all the values down each column of a 2D array.
(2) Each value does not move left or right to a different column, only its row number is changed.

horizontalFlip

The `horizontallFlip` method *flips* the image vertically by rotating it horizontally. The running order:

```
java ImageConversions computer.jpg hflipTest.jpg horizontalFlip
```

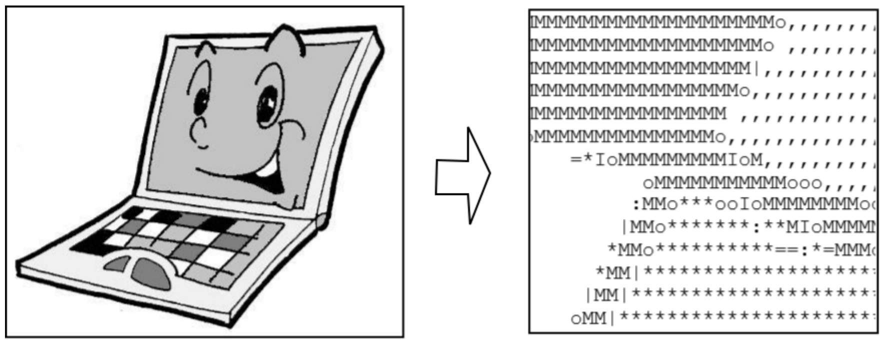
performs the action illustrated in FIGURE 2:

Programming notes:

(1) This method reverses all the values across each row in a 2D array. Think about how to reverse the values in an array.
(2) Each value does not move up or down a row, on its column number is changed.

makeAscii

The `makeAscii` method converts an image to an ascii image, an image similar to what could be created using a common typewriter. This image is sent to a *text* file. Instead of sending the converted array to `writeGrayscaleImage`, it creates a new array of `chars`, using the following mapping that converts an integer into a character:



```
MMMMMMMMMMMMMMMMMMMMMo,,,,,,,,
MMMMMMMMMMMMMMMMMMMMo ,,,,,,,,
MMMMMMMMMMMMMMMMMMM|,,,,,,,,,,
MMMMMMMMMMMMMMMMMMo,,,,,,,,,,,
MMMMMMMMMMMMMMMMM ,,,,,,,,,,,,
oMMMMMMMMMMMMMMMo,,,,,,,,,,,,,
   =*IoMMMMMMMMMIoM,,,,,,,,,,
          oMMMMMMMMMMMooo,,,,
         :MMo***ooIoMMMMMMMMoo
        |MMo*******:**MIoMMMMM
       *MMo**********==:*=MMMo
      *MM|********************:
     |MM|*********************:
    oMM|**********************:
```

FIGURE 3. makeAscii example

Value	Character
0 to 20	`M`
21 to 40	`L`
41 to 60	`I`
61 to 80	`o`
81 to 100	`|`
101 to 120	`=`
121 to 140	`*`
141 to 160	`:`
161 to 180	`-`
181 to 200	`,`
201 to 220	`.`
221 to 255	`[space]`

The running order:

```
java ImageConversions computer.jpg ascii.txt makeAscii
```

performs the action illustrated in FIGURE 3:

scale

The `scale` method *scale* reduces or enlarges the image by a particular *scale factor* The running order:

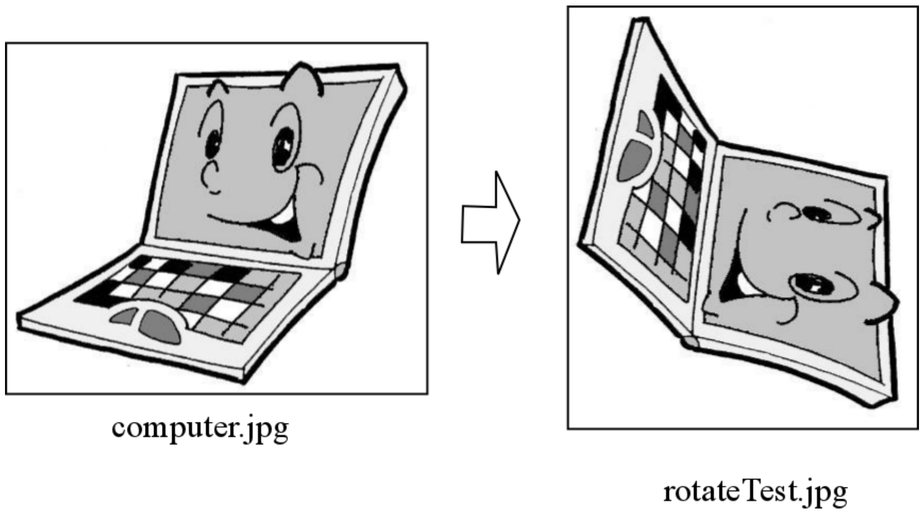


FIGURE 4. rotate example

```
java ImageConversions computer.jpg smaller.jpg scale 0.5
```

produces a shrunken image that is one half the width and height of the original.

Likewise, the running order

```
java ImageConversions computer.jpg smaller.jpg scale 2.0
```

produces a larger image that is twice the width and height of the original

Programming notes:

(1) The output image should be as close to a replica of the original image as possible. Obviously, some values will not produce nice divisions of the grids, and we can only approximate a replica. Usually, the eye cannot see the margin of error.
(2) Use the `Double.parseDouble(String s)` method to convert `args[3]` from a `String` to a `double`. Remember that `"3.2"` is a `String` in Java, and it cannot be used in an arithmetic expression.

rotate

The `rotate` method *rotates* the image in a clockwise direction 90°. The running order:

```
java ImageConversions computer.jpg rotateTest.jpg rotate
```

performs the action illustrated in FIGURE 4:

Programming notes:

(1) If the image is not square, the output image's height is equal to the original image's width, and the width equal to the original image's height.
(2) Use some graph-lined paper and draw the values, mapping where they will go in a clockwise rotation. Try to find the pattern and discover the bit of arithmetic needed for the array indices in the conversion.

Submission

Submit the following completed file to the Assignment folder on conneX.

- `ImageConversions.java`

Please make sure you have submitted the required file(s) and conneX has sent you a confirmation email. Do not send `[.class]` (the byte code) files. Also, make sure you *submit* your assignment, not just save a draft. Draft copies are not made available to the instructors, so they are not collected with the other submissions. We *can* find your draft submission, but only if we know that it's there.

A note about academic integrity

It is OK to talk about your assignment with your classmates, and you are encouraged to design solutions together, but each student must implement their own solution.

Grading

Marks are allocated for . . .

- No errors during compilation of the source code.
- The method headers must be exactly as specified and the methods must perform as specified. Be sure to read the **Method Details** in the specification document.
- The `main` method handles the extraction of the information in the `args` array and calls the required methods.
- Some user mistakes are handled with helpful information. For example, if the user does not execute the program with the correct number of arguments, or the input file does not exist, then the program stops after a helpful message is printed to the console.
- Style of the source code meets the requirements outlined in the Coding conventions document available in the Lab Resources folder of conneX.