Hands-on Experiment # 9 : Worksheet

Section_______1_______ Date________27/10/2563__________________________

Student ID ___________6338110221________________ Name_____Nonthapat Kaewamporn______________

Part A: Practicing multi-dimensional array

Complete the `ArrayUtility.java`

- `public static int[] merge(int[][]d)` combines a given 2D array into 1D array.
  For example, `{ {1, 2}, {3, 4, 5} }` → `{1, 2, 3, 4, 5}`
- `public static int[][] split(int[]d, int n)` splits a given 1D array into 2D array with `n` equal parts.
  For example, a = {1,2,3,4}; split(a, 2) → { {1, 2}, {3, 4} }.
  If the given array cannot be split into `n` equal parts ($0 < n \leq$ `d.length`), this method should return 2D array of size 0,0 (`new int[0][0]`).

Method show()'s are provided to help in display 1D and 2D arrays of int and String.

Run class ArrayUtilityTest to test merge() and split().

Add code to myTestMerge() and myTestSplit() with your own data and expected results.

Part B: Read data from file in to 1D and 2D array

CSVUtility.java provides methods to read a text file and return data as array.

- **public static** String[] readFile(String filename) reads all line in a given filename into an array of String. Throws FileNotFoundException if filename does not exist.
- public static String[][] readCSV(String filename, boolean header) read given csv file into 2D array of String. Throws FileNotFoundException if filename does not exist. If header is true, the first line is the header. The data start from second line. If header is false, the file contains only data.

Run class CSVUtilityTest to test readFile() and readCSV(). You can use CSVUtilityGiven to verify your program.

Note: Read http://docs.oracle.com/javase/8/docs/api/java/util/Scanner.html to learn how to read a text file using an instance of the Scanner class.

Part C: Multi-dimension Array with Text File Processing (Golf score and Handicap)

Golf is a club-and-ball sport in which players use various clubs to hit balls into a series of holes on a course in as few strokes as possible. For more information, read https://en.wikipedia.org/wiki/Golf.

- **Par**: A hole is classified by its par, meaning the number of strokes a skilled golfer should require to complete play of the hole. Normally, there are par 3's, par 4's and par 5's.
- **Score**: The goal is to play as few strokes per round as possible. A golfer's number of strokes in a hole, course, or tournament is compared to its respective par score, and is then reported either as the number that the golfer was "under-" or "over-par", or if it was "equal to par". A hole in one (or an "ace") occurs when a golfer sinks their ball into the cup with their first stroke from the tee. Common scores for a hole also have specific terms.
- A **handicap (HC)** is a numerical measure of an amateur golfer's ability to play golf over the course of 18 holes. A player's handicap generally represents the number of strokes above par that the player will make over the course of an above-average round of golf. The better the player the lower their handicap is.
- To calculate handicap using System-36 formula, use the following table:

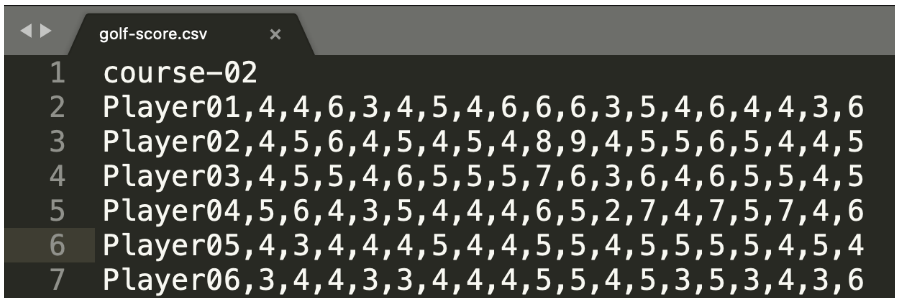
Score - Par	HC get for that hole
≤ 0	0
1	1
≥ 2	2

- Total HC is total HC of each hole.

Hole	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	TOTAL
Par	4	4	5	3	4	4	4	3	5	4	3	5	4	5	4	4	3	4	(72)
Score	4	4	6	2	4	6	4	6	3	5	6	4	4	5	4	5	5	5	82
Score-Par	0	0	1	-1	0	2	0	3	-2	1	3	-1	0	0	0	1	2	1	10
HC	0	0	1	0	0	1	0	2	0	1	2	0	0	0	0	1	2	1	11

- Gross score is total strokes = 82
- Handicap (HC) = sum of 18 holes hc = 11
- Net score = Gross score – Handicap = 82 -11 = 71

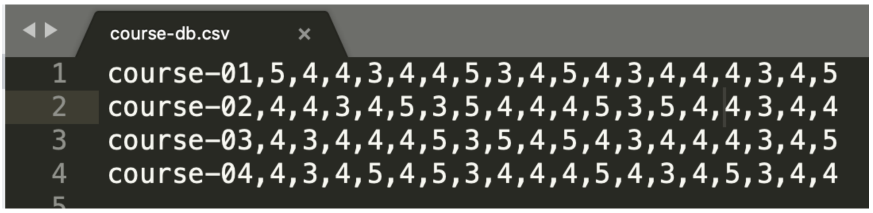
File *golf-score.csv* contains scores of golfers (total stroke in each hole that each golfer taken to finish each hole) for 18 holes. The file is in the "Comma-separated Value" format (http://en.wikipedia.org/wiki/Comma-separated_values) with the first line show golf course id as show in Figure 1.



```
course-02
Player01,4,4,6,3,4,5,4,6,6,6,3,5,4,6,4,4,3,6
Player02,4,5,6,4,5,4,5,4,8,9,4,5,5,6,5,4,4,5
Player03,4,5,5,4,6,5,5,5,7,6,3,6,4,6,5,5,4,5
Player04,5,6,4,3,5,4,4,4,6,5,2,7,4,7,5,7,4,6
Player05,4,3,4,4,4,5,4,4,5,5,4,5,5,5,5,4,5,4
Player06,3,4,4,3,3,4,4,4,5,5,4,5,3,5,3,4,3,6
```

Figure 1: golf-score.csv

File course-db.csv contains course data: course id, course name, follow by 18-hole's par. For example, course-02 is "Siam Country Club Plantation" Par of hole 1-9 is 4,4,3,4,5,3,5,4,4. Par of hole 10-18 is 4,5,3,5,4,4,3,4,4 (Normal par for a course is 36 + 36 = 72).



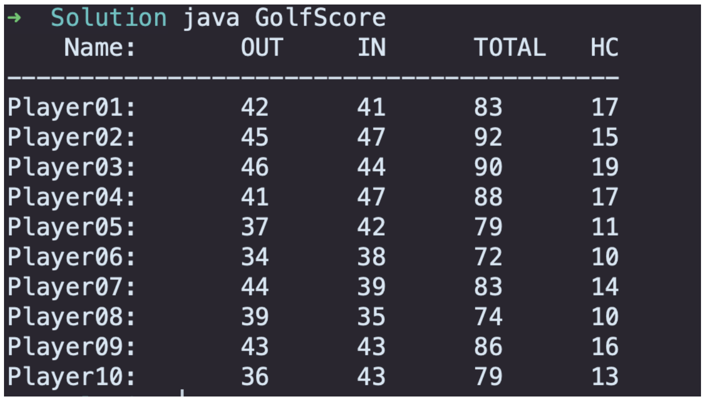
```
course-01,5,4,4,3,4,4,5,3,4,5,4,3,4,4,4,3,4,5
course-02,4,4,3,4,5,3,5,4,4,4,5,3,5,4,4,3,4,4
course-03,4,3,4,4,4,5,3,5,4,5,4,3,4,4,4,3,4,5
course-04,4,3,4,5,4,5,3,4,4,4,5,4,3,4,5,3,4,4
```

Complete GolfScore.java to generate golf competition result as shown in the following screen

- `public static String[] getGolfers(String csvFilename)` gets golfers name from given CSV file using provided `CSVUtility` class. Golfer's names are the first element's in each sub array. Each line of CSV file is in format:
    - golfer's name,h1,h2,h3,h4,h5,h6,h7,h8,h9,h10,h11,h12,h13,h14,h15,h16,h17,h18
- `public static int[][] getScores(String csvFilename)` gets all scores from given CSV file.
    - If the result is store in variable named, `score`
        - `score[i]` is the score for player `i`
        - `score[i][j]` is the score for player I, hole `j+1`
- `public static String getCourseName(String csvFilename)` return course name form given CSV file.
- `public static int[] getCoursePar(String csvFilename, String coursename)` gets course 18-hole par for given course name extracted from csv file
- `public static int[] calculateHandicap(int[][] scores, int[] par)` calculates handicap for all golfers.
- `public static int calculateHoleHandicap(int par, int score)` calculates handicap for a hole from par and player's score Hole's par number is 3, 4, or 5. score is the player score in that hole. Result is 0 if score-par is <= 0, 1 if score - par is equal to 1, and 2 otherwise.
- `public static int[][] calculateScore(int[][] golfScoreList)` calculates:

  sum(hole1 to hole9) -> out
  sum(hole10 to hole18) -> in
  out + in -> total

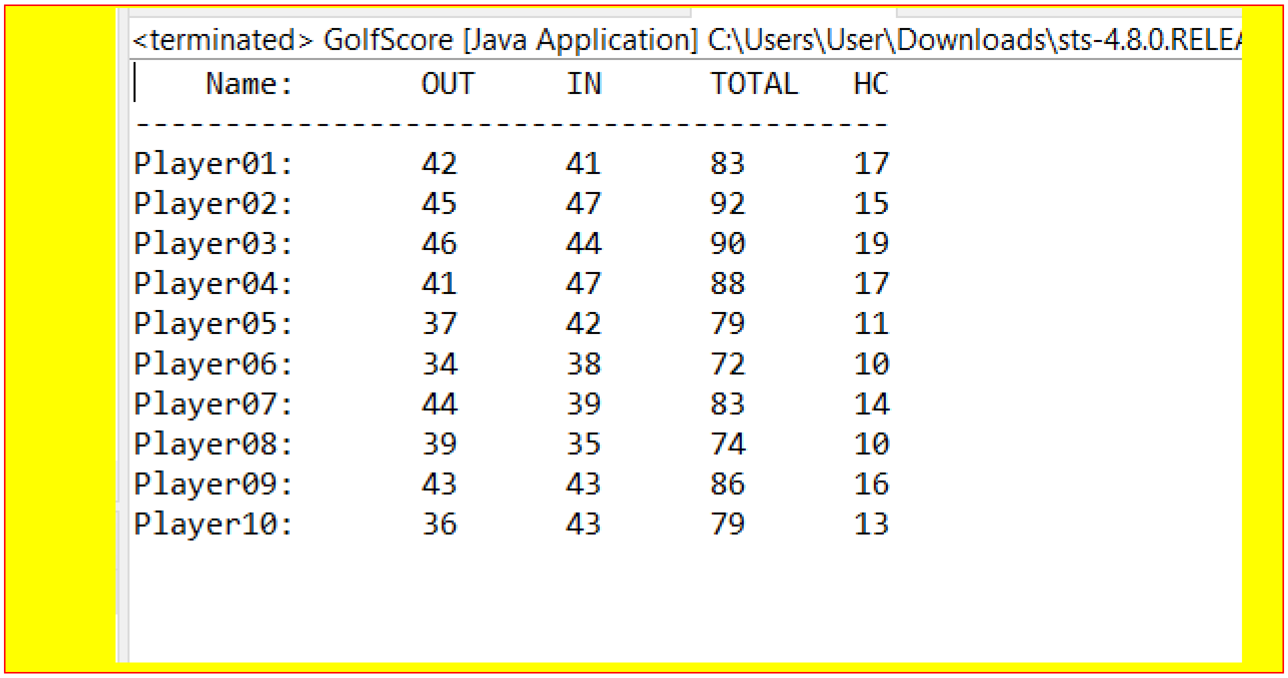
  return out, in, total for all players



```
➜  Solution java GolfScore
    Name:           OUT       IN        TOTAL     HC
------------------------------------------------------
Player01:           42        41        83        17
Player02:           45        47        92        15
Player03:           46        44        90        19
Player04:           41        47        88        17
Player05:           37        42        79        11
Player06:           34        38        72        10
Player07:           44        39        83        14
Player08:           39        35        74        10
Player09:           43        43        86        16
Player10:           36        43        79        13
```

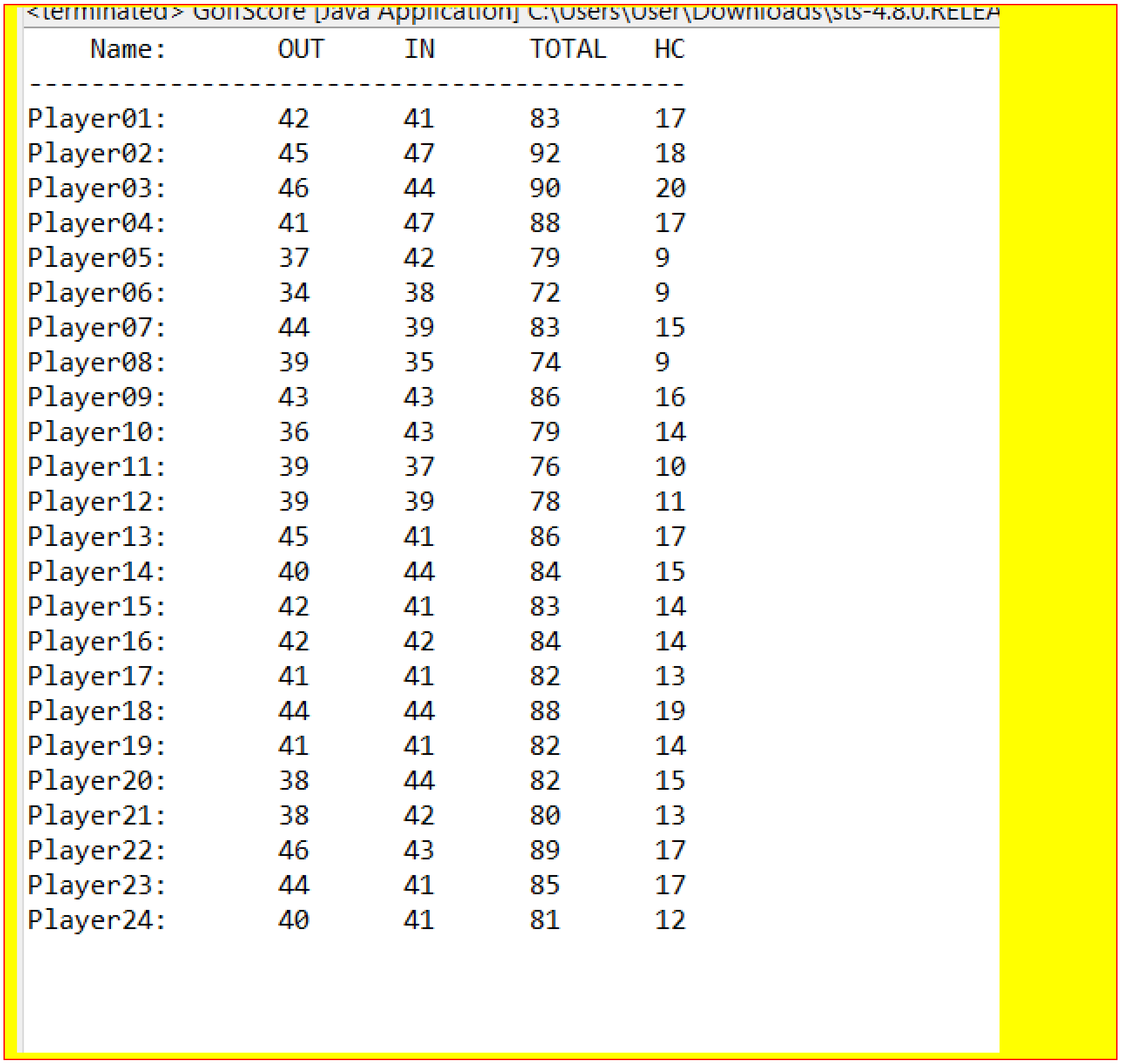
Run GolfScore with “golf-score-1.csv” and “golf-score-2.csv”

Screen capture when run “java GolfScore golf-score-1.csv”



```
<terminated> GolfScore [Java Application] C:\Users\User\Downloads\sts-4.8.0.RELE
    Name:           OUT       IN        TOTAL     HC
------------------------------------------------------
Player01:           42        41        83        17
Player02:           45        47        92        15
Player03:           46        44        90        19
Player04:           41        47        88        17
Player05:           37        42        79        11
Player06:           34        38        72        10
Player07:           44        39        83        14
Player08:           39        35        74        10
Player09:           43        43        86        16
Player10:           36        43        79        13
```

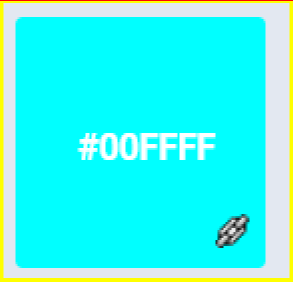
Screen capture when run “java GolfScore golf-score-2.csv”



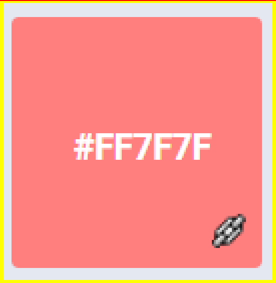
```
<terminated> GolfScore [Java Application] C:\Users\User\Downloads\sts-4.8.0.RELEA
    Name:           OUT       IN        TOTAL     HC
------------------------------------------
Player01:           42        41        83        17
Player02:           45        47        92        18
Player03:           46        44        90        20
Player04:           41        47        88        17
Player05:           37        42        79        9
Player06:           34        38        72        9
Player07:           44        39        83        15
Player08:           39        35        74        9
Player09:           43        43        86        16
Player10:           36        43        79        14
Player11:           39        37        76        10
Player12:           39        39        78        11
Player13:           45        41        86        17
Player14:           40        44        84        15
Player15:           42        41        83        14
Player16:           42        42        84        14
Player17:           41        41        82        13
Player18:           44        44        88        19
Player19:           41        41        82        14
Player20:           38        44        82        15
Player21:           38        42        80        13
Player22:           46        43        89        17
Player23:           44        41        85        17
Player24:           40        41        81        12
```

Part D: Understanding Given Resources/Backgrounds

1) Get yourself familiar with the RGB color model. Play around with the color picker on http://www.colorpicker.com/ and answer the following questions.
    a. What color is it that has the maximal value in B, G and 0 in R? Capture the picture of the color and post it here.



    b. What color is it that has the maximal value in R, and middle value in G and B? Capture the picture of the color and post it here.



(I rounded down the value of G andB)

    c. What are the requirements on the RGB values for all shades of gray?

Same value for R, G, and B

2) Read the API specification of the class *Java101ImageUtil* in *Java101ImageUtilDoc.pdf*.
    a. How many static methods are there in the class?

6

    b. How many overloaded methods are there in the class?

3

    c. Write the "method signatures" of all the overloaded methods. (** Write only the signature)

showViewer

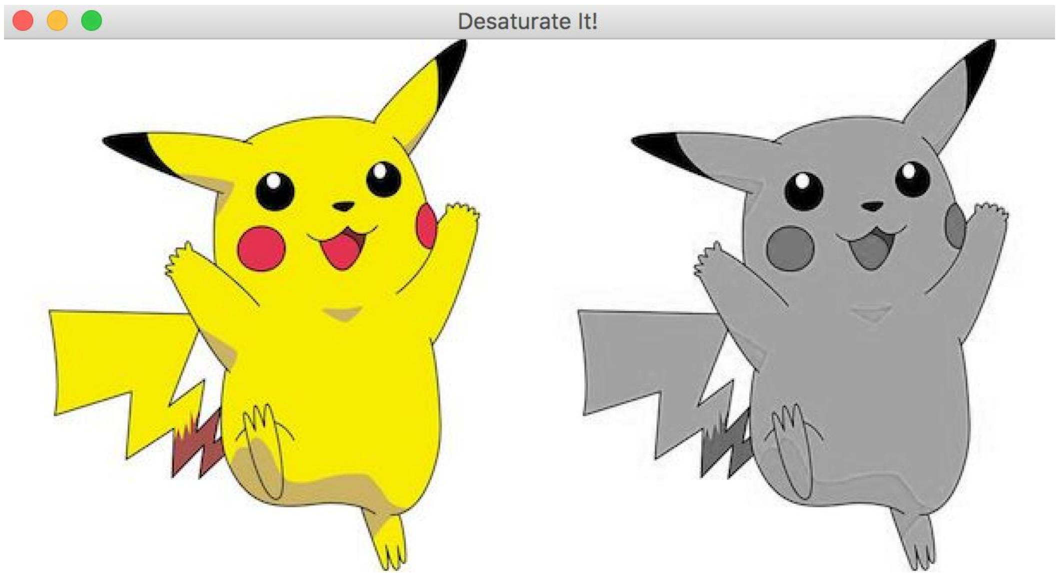
3) Read the source code of *Java101ImageUtilExample.java* and try executing the program. Briefly explain what the program does. (** It is recommended NOT TO open big images. The program was not optimized in any ways. Try the program on some images with a few hundreds of pixels in their width/height)

When the program is run, the program prompt for user input in either JPG or GIF format. Then, the program will give the user 3 choices to choose. One being that the program will straight out display the picture right away. Two is that program will display the original picture alongside the flipped picture. The third one is where the program displays original picture, flipped picture, and the 64x128 picture filled with just red.

Part E: Image Manipulation

Modify *DesaturateIt_Template.java* to obtain a Java program performing the following steps.

1) Ask the user to select a gif or a jpg file.
2) Show the original image and its "desaturated" (grayscale) version using `showViewer()`.



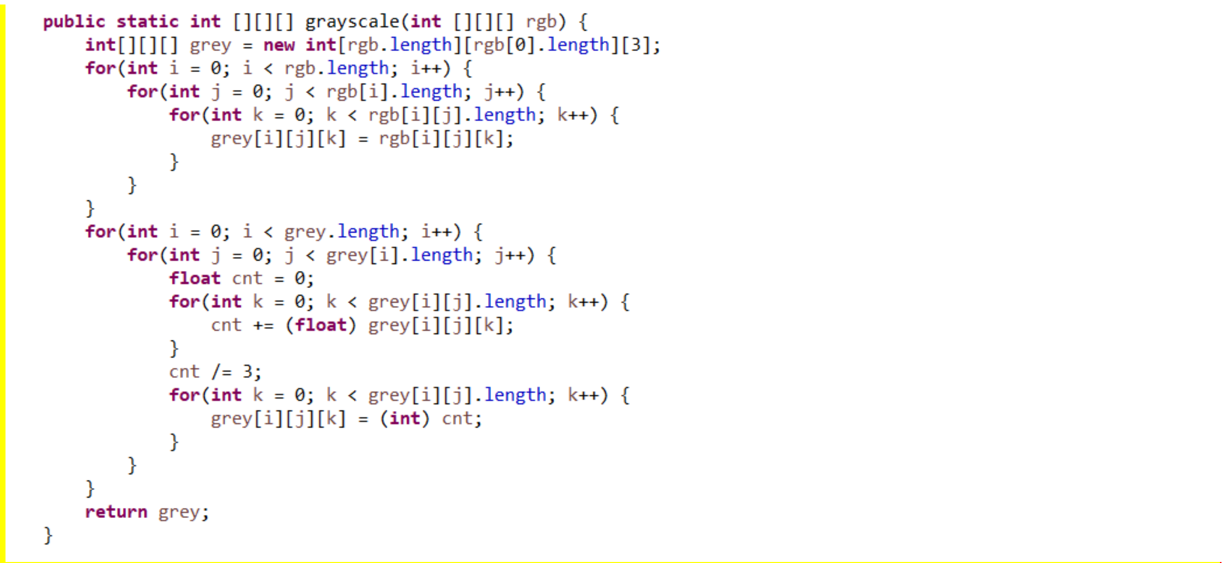
Explain how the grayscale values are computed.

Grayscale works by averaging out the red,green, and blue value of the particular pixel; then, set all of the red green blue value to the average value.

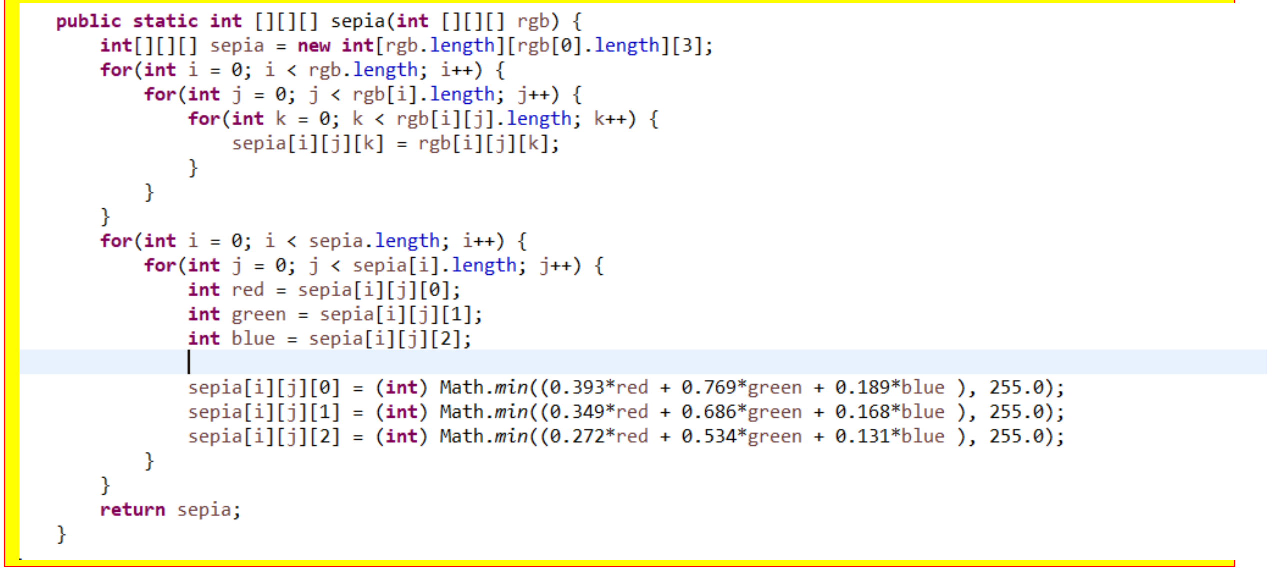
List your source code here.

```
import java.io.File;

public class DesaturateIt_Template{
    public static void main(String [] args){
        int [][][] rgb = Java101ImageUtil.getRGBArrayFromFile();
        if(rgb==null){return;}
        int[][][] gray = grayscale(rgb);
        int[][][] sepia = sepia(rgb);
        // fill code to show all three images
        // - original, grayscale, sepia
        int[][][][] rgbs = {rgb,gray,sepia};
        Java101ImageUtil.showViewer(rgbs, "show all");
    }
```



```
public static int [][][] grayscale(int [][][] rgb) {
    int[][][] grey = new int[rgb.length][rgb[0].length][3];
    for(int i = 0; i < rgb.length; i++) {
        for(int j = 0; j < rgb[i].length; j++) {
            for(int k = 0; k < rgb[i][j].length; k++) {
                grey[i][j][k] = rgb[i][j][k];
            }
        }
    }
    for(int i = 0; i < grey.length; i++) {
        for(int j = 0; j < grey[i].length; j++) {
            float cnt = 0;
            for(int k = 0; k < grey[i][j].length; k++) {
                cnt += (float) grey[i][j][k];
            }
            cnt /= 3;
            for(int k = 0; k < grey[i][j].length; k++) {
                grey[i][j][k] = (int) cnt;
            }
        }
    }
    return grey;
}
```



```
public static int [][][] sepia(int [][][] rgb) {
    int[][][] sepia = new int[rgb.length][rgb[0].length][3];
    for(int i = 0; i < rgb.length; i++) {
        for(int j = 0; j < rgb[i].length; j++) {
            for(int k = 0; k < rgb[i][j].length; k++) {
                sepia[i][j][k] = rgb[i][j][k];
            }
        }
    }
    for(int i = 0; i < sepia.length; i++) {
        for(int j = 0; j < sepia[i].length; j++) {
            int red = sepia[i][j][0];
            int green = sepia[i][j][1];
            int blue = sepia[i][j][2];

            sepia[i][j][0] = (int) Math.min((0.393*red + 0.769*green + 0.189*blue ), 255.0);
            sepia[i][j][1] = (int) Math.min((0.349*red + 0.686*green + 0.168*blue ), 255.0);
            sepia[i][j][2] = (int) Math.min((0.272*red + 0.534*green + 0.131*blue ), 255.0);
        }
    }
    return sepia;
}
```

Submission Instruction

Combine the following file in to a zip file named, **YourStudentID-hw9.zip**

1) L09-2020.pdf
2) ArrayUtility.java
3) CSVUtility.java
4) GolfScore.java
5) DesaturateIt.java

Submit the zip file via http://www.myCourseVille.com (Assignments > Assignment #9) **within three days after your lecture.**