CMPSCI 111L – Intro to Algorithms & Programming: Java Lab

Programming Project #5 – Arrays and Sorting (30 points)

Due: 5/11/20

AssignGrades (10 points)

Create a Java main class named **AssignGrades**. In the `main()` method of the **AssignGrades** class, write a Java program that reads student scores, gets the best score, and then assigns grades based on the following scheme:

Grade is A if score is >= best – 10;
Grade is B if score is >= best – 20;
Grade is C if score is >= best – 30;
Grade is D if score is >= best – 40;
Grade is F otherwise.

The program prompts the user to enter the total number of students, then prompts the user to enter all of the scores, and concludes by displaying the grades. Here is a sample run:

```
Enter the number of students: 4
Enter 4 scores: 40 55 70 58
Student 0 score is 40 and grade is C
Student 1 score is 55 and grade is B
Student 2 score is 70 and grade is A
Student 3 score is 58 and grade is B
```

AnalyzeScores (10 points)

Create a Java main class named **AnalyzeScores**. In the `main()` method of the **AnalyzeScores** class, write a Java program that reads an unspecified number of **double** scores from the keyboard, stores those scores in an array named **scores**, determines the average of the scores in the array, and then determines how many scores are above or equal to the average, and how many scores are below the average. The program should then provide the following output in this order:

- All of the numbers stored in the scores array (one score per line).
- The average score.
- The number of scores which were above or equal to the average.
- The number of scores which were below the average.

The program must not prompt the user to enter the number of scores which will be input but rather, the user should enter a negative number to signify the end of the input. Assume the maximum number of scores is 100.

Here is an example run of my program. The output of your **AnalyzeScores** program must follow the same format:

```
Enter score 1 (a negative score exits): 72.25
Enter score 2 (a negative score exits): 79.0
Enter score 3 (a negative score exits): 80.3
Enter score 4 (a negative score exits): 89.95
Enter score 5 (a negative score exits): 99.9
Enter score 6 (a negative score exits): 59
Enter score 7 (a negative score exits): 60
Enter score 8 (a negative score exits): 68.25
Enter score 9 (a negative score exits): 0
Enter score 10 (a negative score exits): -1

The scores in the array:
91.5
72.25
79.0
80.3
89.95
99.9
59.0
60.0
68.25
0.0

The average score is: 70.015
The number of scores above or equal to the average: 6
The number of scores below the average: 4
```

Here are a few hints:

- The program should use a loop to store each of these numbers in the array named **scores** as they are typed in.
- The program should use a loop to compute the sum by adding each of the scores stored in the **scores** array to a variable named sum. You may want the program to do this in the same loop which stores the scores in the **scores** array.
- Remember that the maximum array length should be 100 elements, but that the user **may not want to type in 100 values**, so the program must "count" the number of scores that have been stored in the array. This means that if the user only types in 10 scores, the program should only process the first 10 elements of the array. Likewise, if the user enters 50 scores, the program should only process the first 50 elements of the array.
- The program should use a loop to display each of the scores in the **scores** array. It should use `System.out.println()` to display the scores one per line.
- **After** the program has computed the average, it should use a loop to iterate over the **scores** array and "count" the numbers in the array which are above or equal to the average as well as the numbers in the array which are below the average.

__**AnalyzeAndSort100Scores (10 points)**__

Create a Java main class called **AnalyzeAndSort100Scores**. Copy all of the statements in your **AnalyzeScores** `main()` method and paste them into the `main()` method of this **AnalyzeAndSort100Scores** class. Re-write the program so that it now:

1. Reads 100 scores from a data file named scores.dat.
2. Sorts the **scores** array into ascending numeric order before displaying them in the output.

Here is an example run of my program. The output of your **AnalyzeAndSort100Scores** program __must__ follow the same format:

```
The scores in the array:
10.09
12.19
16.18
20.37
20.58
21.03
21.26
21.69
21.74
22.63
22.89
24.67
25.81
26.2
26.21
26.39
26.85
26.98
27.0
27.68
28.65
29.45
31.13
33.47
34.39
34.59
36.29
36.9
37.61
38.1
38.28
38.47
39.41
39.59
```

```
40.96
41.17
45.5
45.72
46.91
48.61
51.1
51.38
51.68
52.49
52.53
52.82
53.23
53.9
54.93
55.03
55.23
56.61
57.07
61.09
61.56
62.05
62.73
65.98
66.14
66.75
68.74
68.88
69.06
69.62
70.07
70.07
70.5
72.05
72.34
72.91
74.1
74.1
74.99
77.13
78.3
78.49
80.15
80.45
80.6
80.94
81.63
```

```
82.86
83.14
83.66
84.34
86.58
86.86
87.27
87.94
89.53
90.09
92.07
95.88
96.0
96.46
97.64
98.11
100.0
100.0
100.0

The average score is: 56.61490000000001
The number of scores above or equal to the average: 48
The number of scores below the average: 52
```

Here are some hints:

- **USE THE `selectionSort()` METHOD GIVEN IN THE TEXTBOOK. Do not try to reinvent the wheel when a working sort method is given in the textbook.**
- If your original **AnalyzeScores** program worked properly for an arbitrary number of scores, those same statements should work in this program with very few modifications. Re-writing my program in this way required me to only: save scores.dat to the root folder of my Project5 folder, add one File object declaration statement, remove the user prompt, add one statement to invoke the `selectionSort()` method, change three loop continuation conditions, then I copied and pasted the authors' `selectionSort()` method into my program.

Remember to write a Universal comment header at the top of each source code file in this project.

When you have completed the exercise, **ZIP** the entire project folder and upload this **ZIP** file to Canvas. In **NetBeans**, click **File**, select **Export Project**, click **To ZIP...**, select the location where you want the **ZIP** file to be saved, type the name of the **ZIP** file, **Project5.zip** in the **File Name** textbox, click the **Save** button, then click the **Export** button. If the IDE you are using does not have an export-project-to-zip option, you may have to manually **ZIP** the file by navigating to the project folder in a file browser and selecting your OS's "compress folder" option. Under Mac OS, the file browser is called **Finder** which offers a **Compress Folder** option. Under Windows, the file browser is called **File Explorer** and it offers a **Send to=>Compressed(zipped) folder** option.