CMPSCI 111L Intro to Algorithms & Programming with Java / Lab

Programming Project #3 – Grade Book (30 points)

Due 4/1/20

Write a command-line Java program which allows me to input exam **scores** and track grades. Since I do not want to count the number of exam scores to be entered, the program must not prompt to enter the number of exam scores in my list. This means that the program should work properly for a list of exam scores of any length. The key to developing this program will be in writing a **sentinel loop** which will allow the user to input exam scores, one at a time and process those scores, until a special **sentinel value** of -1 (or any negative number) is entered. This **sentinel value** will indicate that the program should exit this sentinel loop and display a grade report. The prompt, "Enter a score (a negative score xits): " must be repeatedly displayed to the user.

The **scores** that the user will enter will be a series of **double**-type numbers between 0.0 and 100.0, inclusive of 0.0 and 100.0, where an **A** grade is a score that is greater than or equal to 90.0 **and** less than or equal to 100.0, a **B** grade is a score that is greater than or equal to 80.0 **and** less than 90.0, a **C** grade is a score that is greater than or equal to 70.0 **and** less than 80.0, a **D** grade is a score that is greater than or equal to 60.0 **and** less than 70.0 and an **F** is a score that is greater than or equal to 0.0 **and** less than 60.0.

As each score is entered, the program must display a message that "echoes" the score that was just entered by the user as well as the letter grade that the score represents, so that the program's input and output look like:

```
Enter a score (a negative score exits): 72.5
```


```
The score of 72.5 is a C
```

The program must output a message like this for every score that the user enters. This also means that your program must keep a **count of the number of A's, B's, C's, D's and F's the user has entered** so that you can list individual **letter grade totals** in the program's grade report after the program exits its sentinel loop. You will also need to keep **count** of the **total number of scores** that the user entered as well as the **sum of all the scores** that the user entered so that your program can compute the **average score** after the program exits its sentinel loop.

AFTER the user enters a **negative** score and the program exits its sentinel loop, the program must output a grade report containing all of the following statistics:

```
----------------------------
Grade Report
----------------------------
Total number of A's: V
Total number of B's: W
Total number of C's: X
Total number of D's: Y
Total number of F's: Z
Total number of scores input: XX
Average class score: YY.YY
Average class grade: A or B or C . . .
```

It should be fairly obvious that the individual grade totals (Total number of A's, Total number of B's, etc.) should be computed by your program and that the total number of scores input (XX) should be equal to the total number of A's + B's + C's + D's + F's. The **average class score** (YY.YY) should be computed as a **double**-type number by dividing the **sum** of all scores by the total number of scores input (XX). Be sure to compute the average class score (YY.YY) as a **double** data type value. If the average class score is 82.45, I DO NOT want it to be rounded to 82 or 83.0. In other words, the average class score should have AT LEAST two decimal place precision (more than two decimal places is fine). Once the program has computed the average class score, it should use a selection structure to "decide" what the **average class grade** is.

Break large problems into smaller, more manageable pieces!

Sometimes it is useful for beginners to complete a smaller, simpler program before tackling the full project. I recommend starting with an easy program that only completes part of the project. Name your NetBeans project **Project3** and your main class **GradeBook**. Remember, always use class, variable and project names that clearly describe what you are creating.

Start Here:

Write a program which prompts a user to input **one** score. The program should use some sort of if-else or switch selection structure to display the letter grade, depending on the value of the user-entered score. The program should only compute **one** letter grade per run.

Add a Loop:

Add a loop to the beginning of your program, before the user prompt and input statement. The loop should allow a user to input an arbitrary-length list of scores. Within the loop, as each score Is entered, the program's selection structure should determine whether the entered score is an A, B, C, D or an F. If the score is an A, B, C, D, or F, the program should update the variables which are used to track the program's totals and print the grade message to the console. The user should enter a sentinel value of -1 to exit the loop. **Remember to check the input so that the sentinel value (-1) is not "counted" as a score (i.e. if the score is less than zero, do not count the score as an F)**.

Dividing a larger problem into smaller-sub problems is an important technique in computer programming known as the "Divide and Conquer" approach to problem solving.

Please Design First

It is good practice to design your software before you write the code. Here are some design tasks you should complete before you start:

- Write out a list of variables you think you will need in the program (i.e. int totalA; / to track the number of scores that are A's, etc.)
- Flow chart (or pseudocode) the selection structures needed to determine whether a score is an A, B, C… The flow chart for such a selection would look similar to Figure 3.4 on page 82 of the textbook.
- Flow chart (or pseudocode the loop, include the check for the sentinel value (-1) input and update of variables.

In other words, have some idea about **how** you are going to solve the problem **before** you write the code which attempts to solve the problem.

When you have completed this exercise, ZIP the entire project folder and upload this ZIP file to Canvas. In NetBeans, click **File** then 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 (remember to include the .zip extension) then click the **Save** and **Export** buttons.

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 **Windows Explorer** and it offers a **Send to => Compressed (zipped folder)** option.