

CSE 142, Spring 2018

Programming Assignment #4: Gradanator (20 points)

Due Tuesday, April 24, 2018, 9:00 PM

This interactive program focuses on if/else statements, Scanner, and returning values. Turn in a file named `Gradanator.java`. To use a Scanner for console input, you must import `java.util.*` in your code.

This program prompts a student for grades on homework and two exams and uses them to compute the student's course grade. Below is one example log of execution from the program. This program's behavior is dependent on the user input (user input is bold and underlined below to make it stand out). Your output should match our examples exactly when given the same input. (Be mindful of spacing, such as after input prompts and between output sections.) **Look at the other example logs on the course web site** to get more examples of the program's behavior.

```
This program reads exam/homework scores
and reports your overall course grade.
```

```
Midterm:
Weight (0-100)? 20
Score earned? 78
Were scores shifted (1=yes, 2=no)? 2
Total points = 78 / 100
Weighted score = 15.6 / 20
```

```
Final:
Weight (0-100)? 35
Score earned? 95
Were scores shifted (1=yes, 2=no)? 1
Shift amount? 10
Total points = 100 / 100
Weighted score = 35.0 / 35
```

```
Homework:
Weight (0-100)? 45
Number of assignments? 3
Assignment 1 score and max? 18 20
Assignment 2 score and max? 29 32
Assignment 3 score and max? 31 40
How many sections did you attend? 4
Section points = 20 / 30
Total points = 98 / 122
Weighted score = 36.1 / 45
```

```
Overall percentage = 86.7
Your grade will be at least: 3.0
<< your custom grade message here >>
```

The program begins with an introductory message that briefly explains the program. The program then reads scores in three categories: midterm, final, and homework. Each category is weighted; that is, its points are scaled up to a fraction of the 100 percent grade for the course. As the program begins reading each category, it first prompts for the category's weight.

The user begins by entering scores earned on the midterm. The program asks whether exam scores were shifted, interpreting an answer of 1 to mean "yes" and 2 to mean "no." If there is a shift, the program prompts for the shift amount, and the shift is added to the user's midterm score. Exam scores are capped at 100. For example, if the user scored a 95 and there was a shift of 10, the shifted score would be 100, not 105. The midterm's "weighted score" is printed, which is equal to the user's score multiplied by the exam's weight.

Next, the program prompts for data about the final. This behavior is the same as for the midterm.

Next, the user enters information about their homework, including the weight and how many assignments were given. For each assignment, the user enters a score and points possible.

Section attendance is included in the homework category. You should assume that each section attended is worth 5 points, up to a maximum of 30 points. This means that section points are capped to 30 before they are added to the homework category.

Once the program has read the user information for both exams and homework, it prints the student's overall percentage earned in the course, which is the sum of the weighted scores from the three categories, as shown below:

$$\text{Grade} = \text{WeightedMidtermScore} + \text{WeightedFinalExamScore} + \text{WeightedHomeworkScore}$$

$$\text{Grade} = \left(\frac{78}{100} \times 20 \right) + \left(\frac{100}{100} \times 35 \right) + \left(\frac{18 + 29 + 31 + (4 \times 5)}{20 + 32 + 40 + 30} \times 45 \right)$$

$$\text{Grade} = 15.6 + 35.0 + 36.1475411$$

$$\text{Grade} = \text{round}(86.7475411) = 86.7$$

The program prints a loose guarantee about a minimum grade on the 4.0-scale the student will get in the course, based on the following scale. See the logs of execution on the course web site to see the expected output for each grade range, but note that these numbers here are simplified compared to the guarantees on the syllabus.

85% and above: 3.0; **84.99% - 75%:** 2.0; **74.99% - 60%:** 0.7; **under 60%:** 0.0

After printing the guaranteed minimum grade, print a custom message of your choice about the grade. This message should be different for each grade range shown above. It should be at least 1 line of any non-offensive text you like.

This program processes user input using a `Scanner`. You should handle the following two special cases of input:

- A student can receive extra credit on an *individual assignment*, but **the total points for homework are capped at the maximum possible**. For example, a student who earns 41/40, 39/40, and 47/40 on three assignments, and 25/30 on section attendance will receive 150 homework points (the max) even though they earned 152. Sections points are capped at 30.
- **Cap exam scores at 100**. If the raw or shifted exam score exceeds 100, a score of 100 is used.

Otherwise, you may assume the user enters **valid input**. When prompted for a value, the user will enter an integer in the proper range. The user will enter a number of homework assignments ≥ 1 , and the sum of the three weights will be exactly 100. The weight of each category will be a non-negative number. Exam shifts will be ≥ 0 .

This document describes several numbers that are important to the overall program. For full credit, **you should make at least one of such numbers into a class constant** so that the constant could be changed and your program would adapt.

Development Strategy and Hints:

- Tackle parts of the program (midterm, homework, final exam) one at a time, rather than writing the entire program at once. Write a bit of code, get it to compile, and test what you have so far. If you try to write large amounts of code without attempting to compile it, you may encounter a large list of compiler errors and/or bugs.
- To compute homework scores, use a **cumulative sum** as described in lecture and in textbook section 4.2. You will need to cumulatively sum not only the total points the student has earned, but also the total points possible on all homework assignments.
- The homework part reads two values on one line from the `Scanner`. See the lecture slides for an example of this.
- Many students get "cannot find symbol" compiler errors. Common mistakes include forgetting to pass / return a needed value, forgetting to store a returned value into a variable, and referring to a variable by the wrong name.
- All weighted scores and grades are printed with no more than 1 digit after the decimal point. Achieve this with a custom rounding method or `System.out.printf`. The following code prints variable `x` rounded to the nearest tenth:

```
double x = 1.2345;  
System.out.printf("x is around %.1f in size.\n", x); // 1.2
```
- If you are getting scores of 0 regardless of what data the user types, you may have a problem with integer division. See Chapter 2 about types `int` and `double`, type-casting, and how to avoid integer division problems. If you have a value of type `double` but need to convert it into an `int`, use a type-cast such as the following:

```
double d = 5.678;  
int i = (int) d; // 5
```
- Use `Math.max` and `Math.min` to constrain numbers to within a particular bound.

Style Guidelines:

For this assignment, you are limited to Java features from Chapters 1-4 of the textbook. A major part of this assignment is demonstrating that you understand parameters and return values. Use static methods, parameters, and returns for structure and to eliminate redundancy. For full credit, your program must have and use **at least 4 non-trivial methods** other than `main`. (For reference, our solution is roughly 110 lines long (66 “substantive”), with 6 methods other than `main`.)

Like on previous assignments, you should not have `println` statements in your `main` method. Also, `main` should be a concise summary of the overall program, and should make calls to several other methods that implement the majority of the program's behavior. Your methods will need to make appropriate use of parameters and return values. Each method should perform a coherent task and should not do too large a share of the overall work. Avoid lengthy “chaining” of method calls, where each method calls the next, no values are returned, and control does not come back to `main`. See textbook Chapter 4's case study for a discussion of well-designed versus poorly designed methods.

When handling numeric data, you are expected to choose appropriately between types `int` and `double`. You will lose points if you use type `double` for variables in cases where type `int` would be more appropriate.

Some of your code will use conditional execution with `if` and `if/else` statements. You are expected to use these statements properly. Review book sections 4.1-4.3 about nested `if/else` statements and factoring them.

Give meaningful names to methods, variables, and parameters. Use proper indentation and whitespace. Follow Java's naming standards as specified in Chapter 1. Limit line lengths to 100 chars. Localize variables when possible; (i.e. declare them in the smallest scope needed). Include meaningful header comments at the top of your program and at the start of each method.