CSE 142, Summer 2012
Programming Assignment #4: Grades (20 points)
Due Tuesday, July 17, 2012, 11:00 PM

This interactive program focuses on `if/else` statements, `Scanner`, and returning values. Turn in a file named `Grades.java`. To use a `Scanner` for console input, you must `import java.util.*;` in your code. The program reads as input a student's 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 behaves differently depending on the user input; user input is bold and underlined below. Your output should match our examples exactly 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** and on the next page to get more examples of the program's behavior.

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

Exam 1:
What is its weight (0-100)? 20
Score earned? 78
Was there a curve (1=yes, 2=no)? 2
Total points = 78 / 100
Weighted score = 15.6 / 20

Exam 2:
What is its weight (0-100)? 30
Score earned? 95
Was there a curve (1=yes, 2=no)? 1
How much was the curve? 10
Total points = 100 / 100
Weighted score = 30.0 / 30

Homework:
What is its weight (0-100)? 50
Number of assignments? 3
Assignment 1 score and max? 14 15
Assignment 2 score and max? 17 20
Assignment 3 score and max? 19 25
How many sections did you attend? 5
Section points = 15 / 20
Total points = 65 / 80
Weighted score = 40.6 / 50

Overall percentage = 86.2
Your grade will be at least: 3.0
<< your custom grade message here >>
```

The program begins with an introduction message that briefly explains the program. The program then reads scores in three categories: homework and two exams. Each category is weighted; 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 an exam. The program asks whether the exam was curved, interpreting an answer of 1 to mean "yes" and 2 to mean "no." If there is a curve, the program prompts for the curve amount, and the curve is added to the user's exam score. Exam scores are capped at a max of 100; for example, if the user got 95 and there was a curve of 10, the score to use would be 100. The exam'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 a second exam. This behavior is the same as the behavior for the first exam.

Next the user enters information about his/her homework, including the weight and how many assignments were given. For each assignment, the user enters a score and points possible. Use a **cumulative sum** as in textbook section 4.2.

Part of the homework score comes from sections attended. We will simplify the formula to assume that each section attended is worth 3 points, up to a maximum of 20 points.

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:

$$Grade = WeightedMidtermScore + WeightedFinalExamScore + WeightedHomeworkScore$$

$$Grade = \left(\frac{78}{100} \times 20\right) + \left(\frac{100}{100} \times 30\right) + \left(\frac{14+17+19+(5 \times 3)}{15+20+25+20} \times 50\right)$$

$$Grade = 15.6 + 30.0 + 40.6$$

$$Grade = 86.2$$

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.

85% and above: 3.0; **84.99% - 75%:** 2.0; **74.99% - 60%:** at least 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 assignment; for example, 22 / 20 is a legal score. But **the total homework points are capped at the max possible**. For example, if a student receives 63 / 60 total, cap this to 60 / 60.
- **Cap exam scores at 100.** If the raw or curved 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, and one in a 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. Curves will be ≥ 0.

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, 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. See textbook section 4.2 about cumulative sums.
- 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 by the program with no more than 1 digit after the decimal point. Achieve this with `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 Ch. 1-4. 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, 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`, though you do not need to match this.

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; main should make calls to several of your 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.)

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.

When handling numeric data, you are expected to choose appropriately between types `int` and `double`. For example, 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. Part of your grade will come from using these statements properly. Review book sections 4.1-4.3 about nested `if/else` statements and factoring them.

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