CSE 142, Autumn 2008
Programming Assignment #4: Grades (20 points)
Due Tuesday, October 21, 2008, 11:30 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 uses a student's grades on homework, a midterm exam, and a final exam to compute an overall course grade as a weighted average. The student's points earned in each category are divided by the total possible points for that category and multiplied by the category's weight. The program either computes the student's course grade (if the student has taken the final exam) or tells the student what score he/she needs to earn on the final to achieve a certain course grade.

Below is an example log of execution from the program. User input is bold and underlined. Your program's output should match these examples exactly given the same input. (Be mindful of spacing, such as after input prompts and between output sections.) It is important to note that this program behaves differently depending on the user input. 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 your scores on homework
and exams and reports your course grade or
what score you need on the final exam.

Exam weights? 20 30

Homework (weight 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
Sections attended? 5
Section points = 15 / 20
Total points = 65 / 80
Weighted score = 40.6

Exam (weight 20):
Score? 78
Curve? 3
Total points = 81 / 100
Weighted score = 16.2

Have you taken the final? (1=yes, 2=no) 1
Exam (weight 30):
Score? 95
Curve? 10
Total points = 100 / 100
Weighted score = 30.0

Course grade = 86.8
```

The program begins with an introduction message that briefly explains the program. Then the program asks the user for weights of the two exams. You should subtract the sum of the exam weights from 100 to get the homework weight.

Next the user enters how many homework assignments were given. For each assignment, the user enters his/her score and the max score. Use a cumulative sum to compute homework scores, as taught in textbook section 4.1.

Part of the homework score comes from sections the student attended. Each is worth 3 points, up to a maximum of 20.

The user next enters his/her midterm exam score. The maximum exam score is 100. If the exam was curved, these points are added to the user's score, up to a max of 100.

The program can be run before or after the final exam, so first the user enters whether it has been taken; 1 means yes, 2 means no. If it has, the user enters his/her score and the curve, and the student's grade is shown. If not, it computes the final exam score needed to earn a particular course grade.

If the student can achieve the desired course grade without taking the final (in other words, if a final exam score ≤ 0 is needed), the program shows 0.0 as the needed score. If the needed score is above 100, the program indicates this and shows the highest course grade the student can get (the grade that would result if the student earns 100 on the final exam).

In the log of execution shown, the student earned 14/15, 17/20, and 19/25 on homework assignments. The student attended 5 sections, earning 15/20 section points for doing so. The student received a midterm exam score of 78, curved by 3 to 81. The student has taken the final exam and earned a score of 95; it was curved by 10, but exam scores are capped at 100, so the student gets a 100. The following calculations produce the student's grade from the above log:

$$Grade = WeightedHomeworkScore + WeightedMidtermScore + WeightedFinalExamScore$$

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

$$Grade = 40.6 + 16.2 + 30.0$$

$$Grade = 86.8$$

Note that in Java, 81/100 (the midterm score) would evaluate to 0, but the value desired is 0.81.

Implementation Details:

You should handle the following two special cases of user 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**. If a student receives a total of 63 / 60, you should cap this to 60 / 60.
- The maximum score for an exam is 100. **If the 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 two exam weights will be no more than 100. The weight of a particular category will be non-negative but could be 0. Curves will be ≥ 0.

Additional Output:

The following additional logs demonstrate the program's behavior when the user has not taken the final exam.

Log #2

```
This program reads your scores on homework
and exams and reports your course grade or
what score you need on the final exam.

Exam weights? 15 25

Homework (weight 60):
Number of assignments? 5
Assignment 1 score and max? 9 10
Assignment 2 score and max? 15 15
Assignment 3 score and max? 15 20
Assignment 4 score and max? 30 20
Assignment 5 score and max? 20 25
Sections attended? 2
Section points = 6 / 20
Total points = 95 / 110
Weighted score = 51.8

Exam (weight 15):
Score? 61
Curve? 12
Total points = 73 / 100
Weighted score = 11.0

Have you taken the final? (1=yes, 2=no) 2
Desired percentage? 80
Score needed = 68.9
```

Log #3

```
This program reads your scores on homework
and exams and reports your course grade or
what score you need on the final exam.

Exam weights? 33 34

Homework (weight 33):
Number of assignments? 2
Assignment 1 score and max? 75 50
Assignment 2 score and max? 5 25
Sections attended? 8
Section points = 20 / 20
Total points = 95 / 95
Weighted score = 33.0

Exam (weight 33):
Score? 56
Curve? 0
Total points = 56 / 100
Weighted score = 18.5

Have you taken the final? (1=yes, 2=no) 2
Desired percentage? 90
Score needed = 113.3
Sorry, it is impossible to earn this score.
The highest percentage you can get is 85.5.
```

The needed final exam score is computed using a formula such as the following, using the data from Log #2:

$$NeededFinalScore = 100 * \frac{DesiredCourseGrade - WeightedHomeworkScore - WeightedMidtermScore}{FinalExamWeight}$$

$$NeededFinalScore = 100 * \frac{80 - 51.8 - 11.0}{25}$$

$$NeededFinalScore = 68.9$$

Log #3's output demonstrates the case where the needed final exam score is impossible to earn (> 100).

It is also possible that the user's desired percentage in the course is so low that he/she does not need to take the final exam at all. In such a case, your program should print 0.0 as the needed final exam score. The following partial log is a variation of Log #2 where the user enters a desired percentage of 50 instead of 80:

Log #2, second variation (partial log)

```
Have you taken the final? (1=yes, 2=no) 2
Desired percentage? 50
Score needed = 0.0
```

Even the logs in this document do not cover all possible cases; **see the course web site for additional logs** and make sure your program matches all of their output before turning in.

Development Strategy and Hints:

- Tackle parts of the program (homework, midterm, 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. Though the homework output appears early in the program, this is a hard part to write, so you may want to do the exam code first.
- It is very important to write small amounts of code and then compile and test this code to make sure it works properly. If you try to write large amounts of code without attempting to compile it, you will encounter a large list of compiler errors and/or bugs and will have great difficulty fixing the program.
- Many students encounter "cannot find symbol" compiler errors related to scope, parameters, and return values. Avoid common mistakes such as forgetting to pass / return a needed value, forgetting to store a returned value into an appropriate variable, and referring to a variable by the wrong name.
- 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 4.1 about cumulative sums.
- The final exam code can be tricky, so initially make your program just assume that the student has taken the final. Once this works properly, then add the code for the case where the user hasn't yet taken the final exam.
- Worry about edge cases and cleanup last (such as extra credit, exam scores over 100, and rounding).
- Parts of this program (such as the initial prompt for exam weights) read two values from the user on the same line. This is done by printing a single prompt and then two `nextInt` calls on a `Scanner` back-to-back, like this:
```
System.out.print("Type two values: ")
int firstValue = console.nextInt();
int secondValue = console.nextInt();                  // Type two values: 10 20
```
- 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 that accept parameters and return values as appropriate for structure and to eliminate redundancy. **For full credit, use at least 3 non-trivial methods other than `main`.**

Unlike on previous assignments, **you *can* have `println` statements in your `main` method**. However, `main` should still be short and represent a summary of the overall program; the majority of behavior should come from other methods. To achieve this goal, some of your methods will need to return values. Each method should perform a coherent task and should not do too large a share of the overall work. You will not receive full credit if your solution has lengthy "chains" of method calls, where each method calls the next, no values are returned, and control does not come back to `main`. For reference, our solution is roughly 110 lines long, with 5 methods other than `main`, though you do not need to match this.

When handling numeric data, you are expected to choose appropriately between types `int` and `double`. For example, you will lose points if you always use type `double` for your variables even 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 4.2-4.3 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.