AP Computer Science
Project 3 Grades or Admissions
Worth 25 Points
Due Thursday, October 23, 2014, prior to the beginning of school

Choose and complete Option 1 or Option 2. I encourage you to challenge yourself. It is in writing good, difficult programs that you learn to code well. Option 1 is a lengthier assignment than option 2. I would recommend that you tackle parts of the program at a time, rather than write the entire program at once. That is, write a small amount of code, compile and run it until it works properly, then write some more code. Have fun!

Option 1: Grades

Problem Description:

This assignment will give you practice with interactive programs, `if/else` statements, and methods that return values. Turn in a Java class named LastnameGrades in a file named LastnameGrades.java. You will be using a Scanner object for your console input, so you will need to 'import java.util.*;' into your program. Have only a single Scanner object that is passed to methods.

This program uses a student's grades on homework, a midterm exam, and a final exam to compute an overall course grade. The course grade is a weighted average. To compute a weighted average, the student's point scores in each category are divided by the total points for that category, then multiplied by that category's weight. (The sum of all categories' weights should be 100.)

$$Grade = WeightedHomeworkScore + WeightedMidtermScore + WeightedFinalExamScore$$

$$Grade = \left(\frac{HomeworkEarned}{HomeworkPossible} \times HomeworkWeight\right) + \left(\frac{MidtermEarned}{MidtermPossible} \times MidtermWeight\right) + \left(\frac{FinalEarned}{FinalPossible} \times FinalWeight\right)$$

Feel free to use a particular teacher's grading scheme as long as it is similarly complex. Check with me first! You will have to include what that scheme is in the introduction.

Example: A course has 50% weight for homework, 20% weight for its midterm, and 30% weight for its final. There are 3 homework assignments worth 15, 20, and 25 points respectively. The student received homework scores of 10, 16, and 19, a midterm score of 81, and a final exam score of 73 (which was curved by +5 points to make a curved score of 78).

$$Grade = WeightedHomeworkScore + WeightedMidtermScore + WeightedFinalExamScore$$

$$Grade = \left(\frac{HomeworkEarned}{HomeworkPossible} \times HomeworkWeight\right) + \left(\frac{MidtermEarned}{MidtermPossible} \times MidtermWeight\right) + \left(\frac{FinalEarned}{FinalPossible} \times FinalWeight\right)$$

$$Grade = \left(\frac{10+16+19}{15+20+25} \times 50\right) + \left(\frac{81}{100} \times 20\right) + \left(\frac{73+5}{100} \times 30\right)$$

$$Grade = 37.5 + 16.2 + 23.4$$

$$Grade = 77.1$$

Note that the above math is written as real math and not Java math; in Java, an integer expression such as 81/100 would evaluate to 0, but the value intended is 0.81.

This program asks the user to enter his/her grades on homework, a midterm exam, and (optionally) a final exam. Using this information, the program can either compute the student's overall course grade (if the student has taken the final exam) or tell the student what score he/she needs to earn on the final exam to achieve a certain grade in the class.

Program Behavior:

The following section describes the program's behavior in detail. You can also get a good idea of the program's behavior by looking at the logs of execution that will follow, but you should read this section completely to make sure you see all of the cases and behaviors your program should have.

- First, the program prints a header message describing itself.

- Second, the program asks the user for homework information. The user enters the weight of the homework assignments, which can be any number between 0 and 100. The user enters how many homework assignments were completed. For each of these assignments, the user enters his/her score along with the maximum possible score, separated by spaces. Your code should work for any number of assignments greater than or equal to 1.

- Third, the program asks the user for midterm exam information. The user enters its weight from 0 to 100 and his/her score. (The user does not enter the maximum score for the midterm, because this is assumed to be 100.) The user is asked whether the midterm was curved; a response of 1 means yes, and 2 means no.

    - If there was a curve, the user is asked how many points were added. These points are added to the user's midterm score.

    - No exam's score can be above 100, so if the curve would have made the user's score exceed 100, a score of 100 is used.

- Fourth, the program asks the user for final exam information. The program can be run before or after the final exam, so first the user enters whether the final exam has been completed; a response of 1 means yes, and 2 means no.

    - If the final exam has been completed, the program will help the user show his/her overall course grade. The user enters the exam weight and his/her score on the final exam. The user is asked whether the final exam was curved; a response of 1 means yes, and 2 means no. If there was a curve, the user is asked how many points were added. These points are added to the user's final exam score. No exam's score can be above 100, so if the curve would have made the user's score exceed 100, a score of 100 is used. Lastly, the user's overall course grade is printed.

    - If the final exam has not yet been completed, the program will help the user see what final exam score he/she needs to earn a particular overall grade in the course. The final exam weight and the user's desired course grade are entered. The program

computes and shows what score the student needs on the final to achieve the desired course grade.

- If the student can achieve the desired course grade without taking the final exam (in other words, if a final exam score of 0 or less is required), the program shows 0.0 as the needed final exam score.

- If the required score for the student to get the desired course grade is greater than 100, it should still be printed as normal, and then the user should receive a message indicating that the desired score cannot be achieved, and a message showing the highest course grade that the student can get (which is the grade that would result if the student earns 100 on the final exam).

Expected Output:

The following logs of execution indicate the exact format of the output that you should reproduce. Your program's output should match these samples exactly when the same input is typed. Please note that there are some blank lines between sections of output and that some lines of output are indented by four spaces. Also note that input values typed by the user appear on the same line as the corresponding prompt message.

First log of execution (user input underlined)

```
This program accepts your homework and exam
scores as input, and computes your grade in
the course or indicates what grade you need
to earn on the final exam.

Homework:
    What is its weight (0-100)? 50
    How many homework assignments were there? 3
    Homework 1 score and max score: 14 15
    Homework 2 score and max score: 18 20
    Homework 3 score and max score: 19 25
    Weighted homework score: 42.5

Midterm exam:
    What is its weight (0-100)? 20
    Exam score: 81
    Was there a curve? (1 for yes, 2 for no) 2
    Weighted exam score: 16.2

Final exam:
    Have you taken the final exam yet? (1 for yes, 2 for no) 2
    What is its weight (0-100)? 30
    What percentage would you like to earn in the course? 80

    You need a score of 71.0 on the final exam.
```

Second log of execution (user input underlined)

```
This program accepts your homework and exam
scores as input, and computes your grade in
the course or indicates what grade you need
to earn on the final exam.

Homework:
    What is its weight (0-100)? 40
    How many homework assignments were there? 4
```

```
    Homework 1 score and max score: 21 30
    Homework 2 score and max score: 11 20
    Homework 3 score and max score: 28 50
    Homework 4 score and max score: 5 10
    Weighted homework score: 23.64

Midterm exam:
    What is its weight (0-100)? 30
    Exam score: 95
    Was there a curve? (1 for yes, 2 for no) 1
    How much was the curve? 10
    Weighted exam score: 30.0

Final exam:
    Have you taken the final exam yet? (1 for yes, 2 for no) 1
    What is its weight (0-100)? 30
    Exam score: 63
    Was there a curve? (1 for yes, 2 for no) 1
    How much was the curve? 5
    Weighted exam score: 20.4

    Your course grade is 74.04
```

Third log of execution (user input underlined)

```
This program accepts your homework and exam
scores as input, and computes your grade in
the course or indicates what grade you need
to earn on the final exam.

Homework:
    What is its weight (0-100)? 50
    How many homework assignments were there? 2
    Homework 1 score and max score: 10 50
    Homework 2 score and max score: 12 25
    Weighted homework score: 14.67

Midterm exam:
    What is its weight (0-100)? 25
    Exam score: 56
    Was there a curve? (1 for yes, 2 for no) 2
    Weighted exam score: 14.0

Final exam:
    Have you taken the final exam yet? (1 for yes, 2 for no) 2
    What is its weight (0-100)? 25
    What percentage would you like to earn in the course? 90

    You need a score of 245.33 on the final exam.
    Sorry, it is impossible to achieve this percentage.
    The highest percentage you can get is 53.67.
```

Input and Output Details:

You do not have to perform any error checking on user input. You may assume that the user types legal values of the proper type and in the appropriate range. For example, when prompted for a number, assume that the user does enter a number and not a String. When prompted for a number within an expected range, such as a weight or exam score between 0 and 100, the user will enter a valid number in that range and not a number outside the range. When prompted for the number of homework assignments, the user will enter a number no less than 1. The sum of the weights the user will enter will always be 100.

Notice that all real numbers output by the program are printed with no more than 2 digits after the decimal point. To achieve this, you may use code such as the following method to round a `double` value to the nearest hundredth:

```
// Returns the given double value rounded to the nearest hundredth.
   public static double round2(double number)
  {
       return Math.round(number * 100.0) / 100.0;
  }
```

Only round numbers as you are about to print them, such as:

```
System.out.println("Weighted exam score: " + round2(weightedExamScore));
```

Another way to print a number rounded to 2 places is to use the `System.out.printf` method as follows.

```
// print weighted exam score, rounded to 2 decimal places
System.out.printf("%.2f", weightedExamScore);
```

The `System.out.printf` command will always print 2 digits after the decimal point, such as 93.50 instead of 93.5. This will be considered acceptable and correct output for this assignment. printf is not part of the AP Computer Science command set.

Stylistic Guidelines:

For this assignment use the language features covered to date.

Structure your solution and eliminate redundancy in the output by using static methods that accept parameters and return values where appropriate. **For full credit, you must have at least 4 non-trivial methods other than `main` in your program.** In grading, I will examine your main method to make sure that it is short, and I will look at whether you have broken down the problem into several appropriate static methods that capture the structure of the task. To fully achieve this goal, some or all of your methods should return values to pass information back to their caller. Each method should perform a coherent task and should not do too large a share of the overall work. See me if your have questions about your structure of methods.

Use class constants as appropriate to represent important fixed data values in your program.

You are required to properly indent your code and will lose points if you make significant indentation mistakes. You should also use whitespace properly to make your program more readable, such as between operators and their operands, between parameters, and blank lines between groups of statements or methods.

Give meaningful names to methods and variables in your code. Follow Java's naming standards about the format of ClassNames, methodAndVariableNames, and CONSTANT_NAMES. Localize variables whenever possible -- that is, declare them in the smallest scope in which they are needed.

Include a comment at the beginning of your program with basic information and a description of the program and include a comment at the start of each method.

Submission and Grading:

Part of your program's score will come from its "external correctness." External correctness measures whether your log of execution matches exactly what is expected, including identical prompting for user input, identical spacing and indentation, and identical weighted scores and course grades printed. In particular, your program will be tested to make sure that it handles edge cases such as a student needing more than 100% or less than or equal to 0% score on the final exam.

The rest of your program's score will come from its "internal correctness." Internal correctness measures whether your source code follows the stylistic guidelines specified in this document. This includes features such as using `if` and `if/else` statements to represent conditional execution, using `for` loops to capture repetition, representing the structure and redundancy of the program using appropriate parameterized static methods with return values as needed, commenting, naming identifiers, and indentation of your source code.

Option 2: Admissions

Problem Description:

This assignment will give you practice with interactive programs, if/else statements and methods that return values. Turn in a Java class named LastnameAdmissions in a file named LastnameAdmissions.java.

You are to write a program that prompts the user for information about two applicants and that computes an overall score for each applicant. This is a simplified version of a program that might be used for admissions purposes.

Look at the sample logs of execution to see how your program is to behave. Your program must exactly reproduce the behavior demonstrated in these logs. For each applicant, we prompt for exam scores (either SAT or ACT) and overall GPA. The exam information is turned into a number between 0 and 100, the GPA information is turned into a number between 0 and 100 and these two scores are added together to get an overall score between 0 and 200. After obtaining scores for each applicant, the program reports which one looks better or whether they look equal.

Notice that the program asks each applicant whether to enter SAT scores or ACT scores (SAT scores are integers that vary between 200 and 800, ACT scores are integers that vary between 1 and 36). In the case of SAT scores, the user is prompted for SAT verbal and SAT math subscores. In the case of ACT scores, the user is prompted for English, math, reading and science subscores. These scores are turned into a number between 0 and 100 using the following formulas:

$$\text{For SAT Scores: } \frac{2 \cdot verbal + math}{24}$$

$$\text{For ACT Scores: } \frac{2 \cdot reading + English + math + science}{1.8}$$

These formulas produce numbers in the range of 0 to 100. After computing this exam score, we compute a number between 0 and 100 based on the GPA. You will notice that the program prompts for the GPA and the maximum GPA. Both the GPA and maximum GPA are real values (i.e., they can have a decimal part). You should turn this into a score between 0 and 100 using the following formula:

$$\frac{actualGPA}{\max GPA} \cdot 100$$

At this point your program has two scores that vary from 0 to 100, one from their test score and one from their GPA. The overall score for the applicant is computed as the sum of these two numbers (exam result + gpa result). Because each of these numbers is between 0 and 100, the overall score for an applicant ranges from 0 to 200.

Expected Output:

The following logs of execution indicate the exact format of the output that you should reproduce. Your program's output should match these samples exactly when the same input is typed. Please note that there are some blank lines between sections of output and that some lines of output are indented by four spaces. Also note that input values typed by the user appear on the same line as the corresponding prompt message.

First log of execution (user input underlined)

```
This program compares two applicants to
determine which one seems like the stronger
applicant.  For each candidate I will need
either SAT or ACT scores plus a weighted GPA.

Information for the first applicant:
    do you have 1) SAT scores or 2) ACT scores? 1
    SAT math? 450
    SAT verbal? 530
    overall GPA? 3.4
    max GPA? 4.0

Information for the second applicant:
    do you have 1) SAT scores or 2) ACT scores? 2
    ACT English? 25
    ACT math? 20
    ACT reading? 18
    ACT science? 15
    overall GPA? 3.3
    max GPA? 4.0

First applicant overall score  = 147.91666666666666
Second applicant overall score = 135.83333333333331
The first applicant seems to be better
```

Second log of execution (user input underlined)

```
This program compares two applicants to
determine which one seems like the stronger
applicant.  For each candidate I will need
either SAT or ACT scores plus a weighted GPA.

Information for the first applicant:
    do you have 1) SAT scores or 2) ACT scores? 2
    ACT English? 20
    ACT math? 19
    ACT reading? 21
    ACT science? 30
    overall GPA? 3.5
    max GPA? 4.0

Information for the second applicant:
    do you have 1) SAT scores or 2) ACT scores? 1
    SAT math? 610
    SAT verbal? 640
    overall GPA? 4.3
    max GPA? 5.0

First applicant overall score  = 149.16666666666666
Second applicant overall score = 164.75
The second applicant seems to be better
```

Third log of execution (user input underlined)

```
This program compares two applicants to
determine which one seems like the stronger
applicant.  For each candidate I will need
either SAT or ACT scores plus a weighted GPA.

Information for the first applicant:
    do you have 1) SAT scores or 2) ACT scores? 1
    SAT math? 510
    SAT verbal? 530
    overall GPA? 3.4
    max GPA? 4.0

Information for the second applicant:
    do you have 1) SAT scores or 2) ACT scores? 1
    SAT math? 570
    SAT verbal? 500
    overall GPA? 3.4
    max GPA? 4.0

First applicant overall score  = 150.41666666666669
Second applicant overall score = 150.41666666666669
The two applicants seem to be equal
```

Input and Output Details:

You do not have to perform any error checking. We will assume that the user enters numbers and that they are in the appropriate range.

Because your program will be using a Scanner object, you will need to include the following declaration at the beginning of your program:

```
import java.util.*;
```

Have only a single Scanner object that is passed to methods.

Your program should be stored in a file called LastnameAdmissions.java.

Stylistic Guidelines:

For this assignment use the language features covered to date.

Structure your solution and eliminate redundancy in the output by using static methods that accept parameters and return values where appropriate. **You are to introduce at least 4 non-trivial static methods other than main to break this problem into smaller subtasks and you should make sure that no single method is doing too much work.** In grading, I will examine your main method to make sure that it is short, and I will look at whether you have broken down the problem into several appropriate static methods that capture the structure of the task. To fully achieve this goal, some or all of your methods should return values to pass information back to their caller. Each method should perform a coherent task and should not do too large a share of the overall work. In this program, none of your methods should have more than 15 lines of code in the body of the method (not counting blank lines or lines with just curly braces on them). See me if your have questions about your structure of methods.

Also remember that because this program involves both integer data and real data, you need to use appropriate type declarations (type int and calls on nextInt for integer data, type double and calls on nextDouble for real-valued data).

You are required to properly indent your code and will lose points if you make significant indentation mistakes. You should also use whitespace properly to make your program more readable, such as between operators and their operands, between parameters, and blank lines between groups of statements or methods.

Give meaningful names to methods and variables in your code. Follow Java's naming standards about the format of ClassNames, methodAndVariableNames, and CONSTANT_NAMES. Localize variables whenever possible -- that is, declare them in the smallest scope in which they are needed.

Include a comment at the beginning of your program with basic information and a description of the program and include a comment at the start of each method.

Submission and Grading:

Part of your program's score will come from its "external correctness." External correctness measures whether your log of execution matches exactly what is expected, including identical prompting for user input, identical spacing and indentation, and identical overall scores and applicant comparison statements. Your program will be tested to make sure that it handles edge cases such as applicants being equal.

The rest of your program's score will come from its "internal correctness." Internal correctness measures whether your source code follows the stylistic guidelines specified in this document. This includes features such as using `if` and `if/else` statements to represent conditional execution, representing the structure and redundancy of the program using appropriate parameterized static methods with return values as needed, commenting, naming identifiers, and indentation of your source code.