**CSE 174 – Spring 2016**

**PROGRAM #3: 25 points – Due Sunday, September 18, by 11:59 p.m.**

**Outcomes:**

* Write programs that obtain user input
* Write programs that compute mathematical results
* Write programs that display numbers formatted according to a given specification
* Format and comment source code that adheres to a given set of formatting guidelines
* Use a zip compression tool to combine multiple files

**Scoring:**

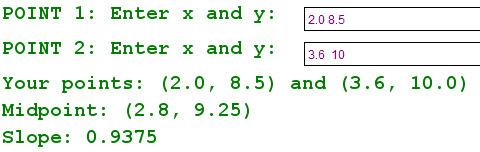
At a bare minimum, the program you submit must have the assigned source code, and your source code must compile and run without crashing.

* If you do not submit your source code (.java files), your score will be zero.
* If you submit source code that does not compile, your score will be zero.
* If you submit source code that roughly resembles the requirements and it compiles, but it crashes under normal operating conditions (nice input from the user), your score will be reduced by 75%.
* Deductions will be made for not meeting the usual requirements (properly formatted source code, class names that do not meet specifications, and so on).

|  |  |  |
| --- | --- | --- |
|  | **Full credit** | **No credit or Partial credit** |
| **Write solutions to the three given problems (15 points)** | The programs you submitted solved the specified problems. | The programs you submitted do not solve or only partially solve the specified problems. |
| **List test cases for the VanCalculator problem (5 points)** | You thoroughly tested your VanCalculator code, listing the test cases that do and do not work. | You did not list test cases, or missed important test cases. |
| **Format output as specified (5 points)** | Your output is formatted as specified, including proper spacing, spelling, rounding values to the specified number of places, and so on. | You did not follow some or all of the requirements for output. |

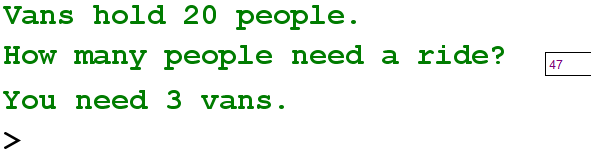
**Requirements: You will write three separate Java programs**

1. (5 points) In a Java class named **Letters**, solve problem P2.18 from the end of chapter 2 in your textbook. HOWEVER...do not use the word "HELLO". Instead, print some other word using the same guidelines, but some *different letters* (you could include H if you want). Your word should have at least 4 *different* letters in it.
2. (5 points) In a Java class named **LineSegment**, write a program that asks the user to enter the coordinates of two points, then display those points, the midpoint, and the slope of the segment that joins those points. The output result should be formatted to look exactly like this:



Note that Scanner objects are good at parsing numbers as shown above. The user could type all 4 values on separate lines, or type them separated with one or more spaces (but not commas), and the Scanner will handle them correctly. You do not need to do anything special to try to "break apart" the two numbers with spaces between them. The Scanner will handle them well. Just call the nextDouble() method two times in a row, and the Scanner will grab each number separately, ignoring the spaces.

1. (5 points) Vans hold 20 people. In a class named **VanCalculator** write a program that asks how many people need a ride, and then print the number of vans needed, formatting your output to match the following as closely as possible.



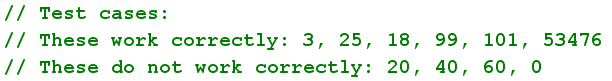
There are a variety of ways to solve this problem. Your goal is to find a solution that only involves simple arithmetic (the symbols + - \* / %). In order to earn full credit, you should have a solution that does NOT use:

* if statements
* loops
* Methods in the Math class (such as Math.sqrt())

If you write a correct solution but it uses one or more of the ideas from this list, you will only earn partial credit for your solution.

If your program *mostly* gives correct answers, and you note that in your comments when you do your testing, you can earn 4 out of the 5 points for this problem.

1. (5 points) Test your **VanCalculator** class thoroughly. You may find that your program works well with some numbers, but not others. (You only need to test your program with positive whole numbers.) If it does not work correctly with certain numbers, then you should try a different approach. In the **VanCalculator** class, at the top of the comments, include a list of the numbers of people you tested with your program, indicating which test cases worked, and which cases did not. It is important to test thoroughly. If you find that your program does not work with certain numbers, you will earn credit for including the numbers that don't work. If you say that your program works with all numbers, but it does not, you will lose credit for this part of the assignment. You should indicate at least 10 numbers that you tried. Be selective. The goal is to test and find mistakes. Here is a sample of what your source code might say (these comments should appear before the rest of your source code):



Note that this part of the assignment is worth 5 points. You can get full credit for this, even if your VanCalculator class does not always produce the correct answer.

1. Follow these steps to submit your work:
   1. Create an empty folder named program3
   2. Put copies of your three source code files (.java) in the program3 folder. *Do not put any other files in that folder.* There should be exactly 3 .java files
   3. Compress the folder to create a zip file. Name the zip folder whatever you want, but be sure it ends with .zip.
   4. Submit only the zip file to the Canvas website.

Note: If you submit your work and decide to modify one of your programs, you need to resubmit a new zip file containing all three source code files (even if you only modified one of them). Do not rename your source code files. Do not rename the zip file. Canvas may add a number to the name of your zip file. That is fine. But you should keep all filenames the same.