AP Computer Science
Unit 3. Programs

For most of these programs I’m asking that you to limit what you print to the screen. This will help me in quickly running some tests on your code once you submit them on jGrader.

Also, even though file names may include the underscore character, _, jGrader does not like it. So do not use the underscore character when naming your classes.

1. Write a program where the user enters 3 integers and the program displays them in order from smallest to largest. For example, if the user enters 6, -2, 4 the program displays “-2 4 6” with exactly one space between each number. If the user enters 12, 5, 12 then the program displays “5 12 12” with exactly one space between each number. Do this using only if /else if/ else statements.

To the right is what your program should look like when you run it. The first three lines are the inputs and the fourth line is the output.

IMPORTANT. The Terminal window will NOT automatically open until you execute a print statement. Therefore you must open the Terminal window by clicking on the View menu and selecting “Show Terminal”.

2. Write a program that takes the lengths of three sides of a triangle as inputs. The program first checks if it possible for a triangle to have sides of those lengths and, if possible, displays whether the triangle is an acute, right, or obtuse triangle. Assume the user enters only positive integers in ascending order.

Your program should only print exactly one of the following words: RIGHT, ACUTE, OBTUSE, ERROR
The words must be upper-case. Of course, the word you print must be based on the three numbers that were input. To the right is a sample program inputs and result.

3. The first two minutes of a phone call cost a flat $1.50. Minutes 3 to 10 cost 50 cents each. Each minute after 10 cost 25 cents each. For example, a 13 minute call costs 6.25 (1.5 for the first 2 minutes plus 4.0 for minutes 3 through 10 and 0.75 minutes for the 11th, 12th, and 13th minutes).

The program should take an integer input (representing the number of minutes) and calculate and display the cost. No print statements other than the one that prints the cost as a double with no dollar signs. Assume the user enters a positive integer.
All of the rest of the programs involve using loops. It is not unusual to accidentally write a program that contains an endless loop. In BlueJ you can break out of an endless loop by right-clicking on the red-silver “barbershop” pole and clicking “Reset Java Virtual Machine.”

4. Copy the following code snippet into a main method and run the program.

```java
double x = 0;
while ( x != 1.0 ){
    System.out.println( x );
    x += 0.1;
}
```

Do you know why the program produced the results that it did?

5. Write a program that keeps prompting the user to guess a secret number (which is a random integer in your program) between 1 and 100. Give the user hints such as “too high” and “too low.” When you guess the number, the computer should respond “you guessed it in # tries.”

   You may want to display the secret number so that you can judge whether the program is working properly or not.

   I will not be collecting this with jGrader so don’t worry about how the inputs and outputs are formatted.

6. Write a program where the user enters a positive integer less than 2 billion and the program determines whether that number is prime or not.

   The program should only print PRIME or NOT. For instance, if the user enters 13, then PRIME should be printed. If the user enters 24, then NOT should be printed.
7. Write a program where the computer tries to guess the number (between 1 and 100) you are thinking of. When the computer guesses, you must respond with –1 (for too low), 1 (for too high), or 0 (for correct).

In the basic version the program will be guessing randomly, sometimes even guessing the same number more than once.

A better approach for the program is to track the minimum and maximum values that the secret number may fall between and to guess the number that is exactly in the middle of these values. If the guess was too low then the minimum value must be increased and it guesses again. If the guess was too high then the maximum value must be decreased and it guesses again. After each wrong guess the minimum and maximum are moving closer and closer together. If the minimum ever equals the maximum then that must be the number (or the user cheated and changed the number in the middle of the game).

I will not be collecting this with jGrader so don’t worry about how the inputs and outputs are formatted.

8. The German mathematician Leibniz developed the following method to approximate the value of $\pi$:
\[
\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \ldots
\]

Write a program that allows the user to specify the number of iterations used in this approximation and display the resulting value. The table below shows the results for the first 4 iterations. Notice how the approximation of $\pi$ becomes more accurate as the number of iterations increase. Assume the user enters a positive integer.

<table>
<thead>
<tr>
<th># of iterations</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>approx. $\pi$</td>
<td>4</td>
<td>4(\times\left(1 - \frac{1}{3}\right)) = 2.66...</td>
<td>4(\times\left(1 - \frac{1}{3} + \frac{1}{5}\right)) = 3.466...</td>
<td>4(\times\left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7}\right)) = 2.895...</td>
</tr>
</tbody>
</table>

The only thing your program should print is the estimate for pi. Here are some examples:
- If the user enter 1 then the program should print 4.0
- If the user enter 2 then the program should print 2.666666666666667
- If the user enter 5 then the program should print 3.3396825396825403

To the right is a sample result if the user enters 4.
9. The user enters a distance (in miles) and the program displays how long it will take to travel there if the driver maintains an average speed of 25, 30, 35, .. 70 miles per hour. For example, if the user enters 200 the following should display.

<table>
<thead>
<tr>
<th>Miles?</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
</tr>
<tr>
<td>This takes 8 hours and 0 minutes at a speed of 25 mph</td>
</tr>
<tr>
<td>This takes 6 hours and 40 minutes at a speed of 30 mph</td>
</tr>
<tr>
<td>This takes 5 hours and 42 minutes at a speed of 35 mph</td>
</tr>
<tr>
<td>This takes 5 hours and 0 minutes at a speed of 40 mph</td>
</tr>
<tr>
<td>This takes 4 hours and 26 minutes at a speed of 45 mph</td>
</tr>
<tr>
<td>This takes 4 hours and 0 minutes at a speed of 50 mph</td>
</tr>
<tr>
<td>This takes 3 hours and 38 minutes at a speed of 55 mph</td>
</tr>
<tr>
<td>This takes 3 hours and 20 minutes at a speed of 60 mph</td>
</tr>
<tr>
<td>This takes 3 hours and 4 minutes at a speed of 65 mph</td>
</tr>
<tr>
<td>This takes 2 hours and 51 minutes at a speed of 70 mph</td>
</tr>
</tbody>
</table>

Hint. First calculate the total travel time in minutes and then convert it to hours and minutes.

Don’t worry about how the data is input or displayed (except that it should look nice).

10. One way to estimate the value of $e$ (sometimes known as Euler’s number) is by evaluating Newton’s series expansion for a particular value of N.

$$
e = 1 + 1 + \frac{1}{2} + \frac{1}{6} + \frac{1}{24} + \cdots$$

or rewritten as:

$$e = \sum_{n=0}^{\infty} \left( \frac{1}{n!} \right)$$

Write a program where the user enters a value for N and the program calculates and displays the estimate. Below are some sample program runs.

- If N has a value of 0, then print 1.0
- If N has a value of 1, then print 2.0
- If N has a value of 2, then print 2.5
- If N has a value of 3, then print 2.6666666666666665
- If N has a value of 4, then print 2.708333333333333
- If N has a value of 5, then print 2.7166666666666663

To the right is a sample result if the user enters 3.
11. Write a program where the user enter a perfect square (e.g. 1, 4, 9, 16, etc.) less than or equal to 100 and the program prints all the numbers from 1 to that number in a square as shown.

Remember: System.out.print( something ); prints without inserting a line break.
System.out.println(); just prints a line break.

12. Optional. Suppose you want to find the area under a curve. It is bounded on the top by the curve, on the bottom by the x-axis, on the left by the y-axis and on the right by the line \( x = x_1 \).

One way to find the area under a curve is to divide the area into rectangles and find the sum of those areas. Your estimate will improve as you use more and more rectangles. Write a program to do this.

Assume that the curve follows the equation \( y = x^2 + b \) and the user will enter a positive number or zero for \( b \) and a positive number of \( x_1 \). The user also enters a positive integer for the number of rectangles to use.

In the above figure the area has been divided into 4 rectangles. The heights of each rectangle are equal to \( y_1, y_2, y_3, \) and \( y_4 \) (which are the midpoints of each rectangle). The widths of the rectangles are equal to \( x_1 / 4 \).

Use the following to test your program.
- If \( b = 2, x_1 = 2, \) and the number of rectangles is 2 then the area = 6.5
- If \( b = 2, x_1 = 2, \) and the number of rectangles is 5 then the area = 6.640000000000001
- If \( b = 2, x_1 = 2, \) and the number of rectangles is 100 then the area = 6.666600000000004

The exact answer is 6 and 2/3 (you need to use calculus to determine the exact solution).