Introduction to Java

CS9053

Tuesday 6 PM – 8:30 PM

Prof. Dean Christakos

Sept. 15, 2020

Due: Sept. 22, 2020

**Assignment 2**

**Part I – Procedures/Functions**

1. Looking back on Assignment 1, Part III, we had the GravityCalculator. The final position of an object falling in a vacuum at time *t* is x(t) = 0.5at2 + vit + xi

Instead of implementing it in the main method, we have created a function called calculatePosition. It looks like this:

**public** **static** **double** calculatePosition() {

double a = -9.81;

**return** -1;

}

The function now has no arguments, and the return value, -1, is just a placeholder.

|  |  |  |
| --- | --- | --- |
| **Variable** | **Meaning** | **Value** |
| a | Acceleration (m/s2) | -9.81 |
| t | Time(s) | 10 |
| vi | Initial Velocity (m/s) | 0 |
| xi | Initial position | 0 |

Using the above explanation for the variables, modify calculatePosition so that it takes arguments for t, vi, and xi and returns x(t). a will be set already.

**Part II**

1. **Grades.** See this table mapping numbers to grades:

|  |  |
| --- | --- |
| **Score** | **Grade** |
| 100-90 | A |
| 80-89 | B |
| 70-79 | C |
| 60-69 | D |
| 0-59 | F |

In Grade.java, use if … else statements to take a score and return a String with the corresponding grade.

If the score is less than 0, return null!

1. **Wind chill.** Given the temperature t (in Fahrenheit) and the wind speed v (in miles per hour), the National Weather Service defines the [wind chill](https://www.weather.gov/safety/cold-wind-chill-chart) to be:

w = 35.74 + 0.6215 t + (0.4275 t - 35.75) v0.16

Write a program WindChill.java that takes two double command-line arguments t and v and prints the wind chill. Use Math.pow(a, b) to compute ab. Note: the formula is not valid if t is larger than 50 F or less than -50 F or if v is larger than 110 mph or less than 3 mph

1. **Muay Thai kickboxing.** In MuayThai.java, write a program that reads in the weight of a Muay Thai kickboxer (in pounds) as an input and prints their weight class. **Use a switch statement**.

|  |  |  |
| --- | --- | --- |
| **CLASS** | **FROM** | **TO** |
| Flyweight | 0 | 112 |
| Super flyweight | 112 | 115 |
| Bantamweight | 115 | 118 |
| Super bantamweight | 118 | 122 |
| Featherweight | 122 | 126 |
| Super featherweight | 126 | 130 |
| Lightweight | 130 | 135 |
| Super lightweight | 135 | 140 |
| Welterweight | 140 | 147 |
| Super welterweight | 147 | 154 |
| Middleweight | 154 | 160 |
| Super middleweight | 160 | 167 |
| Light heavyweight | 167 | 175 |
| Super light heavyweight | 175 | 183 |
| Cruiserweight | 183 | 190 |
| Heavyweight | 190 | 220 |
| Super heavyweight | 220 | - |

**Part III**

1. In the class LoopSum, we will look at using loops to sum up integers.
   1. The function sum100 should use a loop (of your choice) to return the sum of the integers 1 through 100.
   2. The function sumN should take an integer argument to return the sum of integers 1 through N.

Print out the result of each function using System.out.println

1. The factorial operator ! gives the product of all integers less than or equal to n but greater than or equal to 1. That is, n! = n \* (n-1) \* (n – 2) \* … \* 2 \* 1. For example, 5! gives the result 5\*4\*3\*2\*1 = 120. In the class Factorial, write a loop that calculates the factorial of a value and prints the result.
2. Number-to-English: This problem is similar to ComputeChange from the lecture. Write a program to read in a command line integer between -999,999,999 and 999,999,999 and print the English equivalent. Here is an exhaustive list of words that your program should use: negative, zero, one, two, three, four, five, six, seven, eight, nine, ten, eleven, twelve, thirteen, fourteen, fifteen, sixteen, seventeen, eighteen, nineteen, twenty, thirty, forty, fifty, sixty, seventy, eighty, ninety, hundred, thousand, million. Don't use hundred, when you can use thousand, e.g., use one thousand five hundred instead of fifteen hundred. Get started in the NumberToEnglish class
3. You’re going to sum up 1 + 2 + 3 + 4 + … and stop when the sum is over 100. You will do this in one of two ways. The first you will use check to see if the sum is over 100, and if so, the loop will break. Here is the loop in SumToOneHundred.java:

**while**(**true**) {

sum += counter;

counter++;

}

Obviously as you can see, it will never terminate, unless you place a condition that the look will break, and you will have to use the break keyword for this.

Next, you will implement a while loop that does the same thing but WITHOUT the break statement. You will do this by having a condition in the while() statement that will cause it to terminate when the sum is over 100.

1. Now you’re going to sum up all the integers from 0 to 100 (inclusive) using any kind of loop you want (for, while, do … while, etc.). But the caveat is that you have to skip all numbers divisible by 10, so 10, 20, 30, 40, 50, 60, 70, 80, 90, and 100 are not included in the sum. You will write **two loops**. One will use an if statement that determines whether to add the numbers to the sum, and the other will use an if statement that sees if the number is divisible by 10 and will “continue” the loop if it is. Your answer goes in SumZeroToOneHundredSkipTens.java

**Part IV: Arrays**

1. The file DisplayInRows.java has an array of 100 random integers from 0 to 99. Print out the contents of the array intArray on 10 lines of 10 integers each.
2. In CopyReverse.java, there is the array sourceArray with 50 random integers from 0 to 99. Copy the contents of sourceArray into destArray, but in reverse order. For example, sourceArray[0] should be copied to destArray[49]. sourceArray[1] should be copied to destArray[48], and so on.
3. In ArrayStats.java, the variable intArray has 50 random integers between 0 and 99. Find the smallest value in the array, the largest value in the array, and the mean average of the values in the array.
4. In LargestPairProduct.java, there is a variable intArray which contains 10 integers. Find the largest product of all pairs of numbers in the array and display the result. Nested loops will probably be the right thing to do:

(get each index that will be the first member of the pair)

(get each index that will be the second member of the pair)

For efficiency’s sake you will want to avoid multiplying pairs repetitively. I.e., if you multiple intArray[5]\*intArray[6], there is no need to multiply intArray[6]\*intArray[5]

1. In TwoDimensionalArray.java, we have a two dimensional array called twoDimArray and two one-dimensional arrays, arrayOne and arrayTwo. You will copy the CONTENTS of arrayOne into twoDimArray[0] and the CONTENTS of arrayTwo into twoDimArray[1]. Use two nested loops and index 0 of the first dimension of twoDimArray means choosing arrayOne as the array to copy and index 1 of the first dimension means choosing arrayTwo.