Introduction to Java Section

CS9053

Thursday 6:00 PM – 8:30 PM

Prof. Dean Christakos

May 31st, 2024

Due: June 7th, 2024 11:59 PM

**Assignment 2**

**Part I – Loops**

**Catalan's constant** *G* is given by:

*G* = 0.915965594177219015054603514932384110774…

This is 1/4th of the volume of an ideal hyperbolic octahedron.

This can be estimated by the equation:

Write a method that executes this summation using a loop and figure out how many iterations of n it takes for the sum to estimate *G* within .00001.

You should print out something like this:

G is estimated as <x> after <y> iterations

Where <x> is the estimation of *G* to within .00001

*G* is provided in the code.

Because the method estimateCatalan() cannot return both the estimation and the number of iterations, this output should be printed from within the estimateCatalan() method.

**Part II: Arrays**

1. Permutations.java, you are going to start with an array (you can use the sample startingArray, but your code should work for any array). You will then come up with every permutation of those array values and store them in a two dimensional “array of arrays”. Each array should contain a permutation of the input array.

I’ll give you two hints:

First, this is a recursive algorithm.

Second, because arrays are fixed in size, you’re going to need some means of keeping track of which permutation you’re currently on in order to put the new permutation in the right location. There is a static field variable called “currentIndex” which you can increment and use to keep track of where the next permutation array goes.

Example:

startingArray = [1, 3, 4]

Output:

[1, 3, 4]

[1, 4, 3]

[3, 1, 4]

[3, 4, 1]

[4, 1, 3]

[4, 3, 1]

1. Pascal’s Triangle

You’re going to create a Pascal’s triangle which will be stored in an array of arrays.

We have a method createPascalTriangle(int rows) which takes an argument of rows and creates a Pascal’s triangle of that number of rows. You can see that the return type is int[][]. If the input is 5, the output should be:

1

1 1

1 2 1

1 3 3 1

1 4 6 4 1

I’ll make this convenient for you so you don’t have to center-justify the output like you did in math class.

For those who need a review, the *nth* row of a Pascal’s triangle has *n* elements. Row 1 is “1”, row 2 is “1 1” and for each index i of the subsequent row, the first index and last index are 1, and the other indices are given by (i-1) + (i) of the previous row.

A two dimensional array is an array of arrays. Creating a two dimensional array starts with

int[][] pascalTriangle = new int[rows][];

And then for each row of the triangle, you will have to create another array of integers for the number of columns of each row.

After returning the two dimensional array, you should have a nested loop that prints out the pascal’s triangle as in the above example.

**Part III: Strings**

1. Write a Java program that finds the length of the longest substring without repeating characters in a given string.

Here, you should take a string as an input, and return the longest substring of that string that doesn’t have repeating characters.

1. In the class DumbPasswords, we will use loops to generate Strings.

The method printDumbPasswords takes two arguments, m and n.

The format of a dumb password is as follows: number-number-letter-letter-number

Character 1: a digit from **1** to **m (non inclusive)**.

Character 2: a digit from **1** to **m (non inclusive)**.

Character 3: a small letter from the first **n** (inclusive) letters of the alphabet.

Character 4: a small letter from the first **n (**inclusive)letters of the alphabet.

Character 5: a digit from **1** to **m+1 (non inclusive), greater than the first 2 digits**.

printDumbPasswords should print out all the dumb passwords in alphabetical order, separated by a space.

So printDumbPasswords(3, 1) should output

11aa2 11aa3 12aa3 21aa3 22aa3