Introduction to Java
CS9053
Thursday 6:00 PM – 8:30 PM
Prof. Dean Christakos
June 1, 2023
Due: June 9th, 2023 11:59 PM

Assignment 2

Part I – Procedures/Functions

Looking back on Assignment 1, Part III, we had the `PressureChange`. The capacitance of a pair of parallel wires is given by

$$\Delta p = \frac{128}{\pi} \cdot \frac{\mu Q}{D_c^4} L$$

and we want to file the difference in pressure over the length `L`

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

```
public static double calculatePressureChange () {

	return -1;
}
```

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

Variable	Meaning	Value
D_c	Diameter of a typical home water supply pipe – 1 in. (in meters)	$.0254m$
L	Length of pipe	5m
μ	Dynamic viscosity of water	$8.9 \cdot 10^{-4}$ Pa·s
Q	Volumetric flow rate of typical household (8 gallons/minute converted to m^3/s)	$5 \cdot 10^{-4}$ m^3/s

Using the above explanation for the variables, modify `calculatePressureChange` so that it takes arguments for `Dc`, `L`, μ, and `Q` and returns the change in pressure, Δp

Part II

1. **case statements and loops.** There's a menu at a fast food restaurant:

Item	Cost
Hamburger	$5.25
French Fries	$2.50
Cheeseburger	$7.00
Milkshake	$3.75
Buffalo Wings	$6.25
Soda	$1.25

You are to write a loop that repeatedly prompts for an order. The user responds with a letter corresponding to the first letter of the order (H, F, C, M, B, or S), and the cost is added to the total bill. When the order is done, the user responds with "X", at which point the loop ends and the total bill is presented to the user. If the response does not correspond to anything on the menu, an error message appears, and nothing is added.

In `FoodMenu.java`, **using a loop (of your choice) and case statements**, add up the total of the order.

It should work like this (letters in bold are user responses):

```
Enter menu item: F

Total = $5.25

Enter menu item: M

Total = $9.00

Enter menu item: Z

Invalid menu item

Enter menu item: S

Total = $10.25

Enter menu item: X

Order complete. Total is $10.25
```

The program ends at this point.

Part III

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
   ```

2. The Taylor series for the natural logarithm for |x| < 1 is given by:

$$\sum_{n=1}^{\infty} (-1)^{n+1} \frac{x^n}{n} = \ln(1+x)$$

   Figure out, for x = 0 to .9, in steps of .01, how many iterations of n it takes for the sum to estimate ln(1+x) to within .0001.

   You should print out something like this:

   ```
   ln(1.0) is 0.0
   it requires <x> iterations to estimate ln(1.0) within .0001

   ln(1.1) is 0.09531017980432493
   it requires <x> iterations to estimate ln(1.1) within .0001

   ln(1.2) is 0.1823215567939546
   it requires <x> iterations to estimate ln(1.2) within .0001
   ```

where <x> is the number of iterations.

Part IV: Arrays

1. In `CopyShift.java`, there is the array `sourceArray` with 50 random integers from 0 to 99. Copy the contents of `sourceArray` into `destArray`, but shifted n spaces to the right. For example, if `sourceArray` has the contents `[1,5,6,7,9]` and we want to shift it 3 spaces over, `destArray` should have the contents `[6,7,9,1,5]`.

   It should work for arbitrarily large shift values.

2. `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]
```