CSE143X Section #11 Problems

For each of these problems, you may NOT use a while loop, for loop or do/while loop to solve the problem; you MUST use recursion. Unless otherwise noted, you are not allowed to construct any structured objects (no array, ArrayList, Stack, Queue, String, StringBuilder, etc).

1. Write a method factorial that takes an integer n as a parameter and that uses recursion to compute the value of n factorial (also known as n!).

   n! = 1 * 2 * 3 ... * n

   By definition, 0! is 1. The method should throw an IllegalArgumentException if passed a negative value.

2. Write a method called parenthesize that takes a String and an integer n as parameters and that prints the string inside n sets of parentheses. For example, this code:

```
parenthesize("Joe", 2);
System.out.println();  // to complete line of output
parenthesize("The University of Washington", 6);
System.out.println();  // to complete line of output
parenthesize("midterm", 1);
System.out.println();  // to complete line of output
```

   should produce these 3 lines of output:

```
((Joe))
((((((The University of Washington))))))
(midterm)
```

   Your method should throw an IllegalArgumentException if passed a negative number. It could be passed 0, as in:

```
parenthesize("CS143X, Autumn 2023, 0);
System.out.println();  // to complete line of output
```

   In this case the output would have no (i.e., 0) parentheses:

```
CS143, Autumn 2023
```

3. Write a method starString that takes an integer n as a parameter and that returns a string of stars (asterisks) 2^n long (i.e., 2 to the nth power). For example:

```
starString(0) should return "*" (because 2^0 == 1)
starString(1) should return "**" (because 2^1 == 2)
starString(2) should return "****" (because 2^2 == 4)
starString(3) should return "********" (because 2^3 == 8)
starString(4) should return "****************" (because 2^4 == 16)
```

   Your method should take a single integer parameter that specifies the power of 2. The method should throw an IllegalArgumentException if passed a value less than 0.

4. Write a method writeNums that takes an integer n as a parameter and that writes the first n integers starting with 1 to System.out in sequential order separated by commas. For example, the following calls:

```
writeNums(5);
System.out.println(); // to complete the line of output
writeNums(12);
System.out.println(); // to complete the line of output
```

should produce the output:

```
1, 2, 3, 4, 5
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12
```

You must exactly reproduce the format of the examples above. Your method should throw an IllegalArgumentException if passed a value less than 1.

5. Write a method writeChars that takes an integer n as a parameter and that writes out n characters as follows. The middle character of the output should always be an asterisk ("*"). If you are asked to write out an even number of characters, then there will be two asterisks in the middle. Before the asterisk(s) you should write out less-than characters ("<"). After the asterisk(s) you should write out greater-than characters (">"). You should write all of the characters on a single line of output.

For example, the following calls:

```
writeChars(5);
System.out.println(); // to complete the line of output
writeChars(8);
System.out.println(); // to complete the line of output
```

Would produce the following output:

```
<<*>>
<<<**>>>
```

If your method is passed 1 or 2 as an argument, it should write out just asterisks. Your method should throw an IllegalArgumentException if passed a value less than 1.

6. Write a method printTwos that takes an integer n as a parameter and that prints an expression composed of a single odd number multiplied by twos that is equal to n. The twos should surround the odd number with an equal number of twos on either side if possible. For example, the call:

```
printTwos(80);
```

should produce the following output:

```
2 * 2 * 5 * 2 * 2
```

If the expression has an odd number of twos, then the "extra" two should appear at the front of the expression. For example, the call:

```
printTwos(96);
```

should produce the following output:

```
2 * 2 * 2 * 3 * 2 * 2
```

If the number is odd to begin with, it should simply be printed. It is possible that the odd number to print will be 1. For example, the following calls:

```
printTwos(1);
System.out.println(); // to complete the line of output
printTwos(2);
System.out.println(); // to complete the line of output
printTwos(32);
System.out.println(); // to complete the line of output
```

should produce the following output:

```
1
2 * 1
2 * 2 * 2 * 1 * 2 * 2
```

You must exactly reproduce the format of the examples above. Your method should throw an IllegalArgumentException if passed a value less than 1.

7. Write a method stutter that takes a stack containing a list of integers and that replaces every value in the stack with 2 of that value. For example, suppose a stack stores these values:

```
bottom [3, 7, 1, 14, 9] top
```

Then the stack should store these values after the method terminates:

```
bottom [3, 3, 7, 7, 1, 1, 14, 14, 9, 9] top
```

Notice that you must preserve the original order. In the original list the 9 was at the top and would have been popped first. In the new stack the two 9's would be the first values popped from the stack. Your method should take a single parameter of type Stack<Integer>.

8. Write a method writeSquares that takes an integer n as a parameter and that writes the first n squares to System.out separated by commas with the odd squares in descending order followed by the even squares in ascending order. For example, the call:

```
writeSquares(5);
```

should produce the following output:

```
25, 9, 1, 4, 16
```

The odd squares (25, 9, and 1) appear first in descending order followed by the even squares (4 and 16) in ascending order. Notice that commas are used to separate consecutive values in the list. Your method should send its output to System.out and should not call println. For example, the following calls:

```
writeSquares(5);
System.out.println();  // to complete the line of output
writeSquares(1);
System.out.println();  // to complete the line of output
writeSquares(8);
System.out.println();  // to complete the line of output
```

should produce exactly three lines of output:

```
25, 9, 1, 4, 16
1
49, 25, 9, 1, 4, 16, 36, 64
```

You must exactly reproduce the format of these examples. Your method should throw an IllegalArgumentException if passed a value less than 1.

9. Write a method called substring that takes as parameters a string, a start index, and an ending index, and that returns a specified substring of the string. You are implementing a recursive alternative to the standard substring method. As with the standard substring method, your method should return the substring that begins at the start index and that extends to the character just before the ending index. For example:

```
substring("hello", 0, 2) should return "he"
substring("hamburger", 4, 8) should return "urge"
substring("smiles", 1, 5) should return "mile"
substring("howdy", 3, 3) should return ""
```

The method should throw an IllegalArgumentException if the start index is negative or if the ending index is greater than the length of the string or if the start index is greater than the ending index. The method should return an empty string if the two indexes are equal. In implementing this method, you are restricted to the following string methods:

```
charAt(index)     returns the character at the given index
equals(other)     returns whether this String is equal to the other
length()          returns the length of the String
```

You are not allowed to construct any structured objects other than Strings (no array, StringBuilder, Scanner, etc).

10. Write a method writeSequence that takes an integer n as a parameter and that writes to System.out a symmetric sequence of n numbers with descending integers ending in 1 followed by ascending integers beginning with 1, as in the table below:

```
Method Call               Output Produced
-------------------------------------------
writeSequence(1);       1
writeSequence(2);       1 1
writeSequence(3);       2 1 2
writeSequence(4);       2 1 1 2
writeSequence(5);       3 2 1 2 3
writeSequence(6);       3 2 1 1 2 3
writeSequence(7);       4 3 2 1 2 3 4
writeSequence(8);       4 3 2 1 1 2 3 4
writeSequence(9);       5 4 3 2 1 2 3 4 5
writeSequence(10);      5 4 3 2 1 1 2 3 4 5
```

Notice that for odd numbers the sequence has a single 1 in the middle while for even values it has two 1's in the middle.

Your method should throw an IllegalArgumentException if passed a value less than 1. Someone using this method would have to call println to complete the line of output.