Assignment 4

Recursive productOfEvens()	20 pts.
Recursive Sum of Array Elements	20 pts.
Recursive Selection Sort	20 pts.
Recursive countDigitMatches()	20 pts.
Recursive generatePattern()	20 pts.

TOTAL: 100 pts.

General Requirements

- *All files you are turning in must be signed. Please add your name to the source code in comments. Unsigned work automatically loses **5 pts**.*
- *Add comments to the source code you are writing:*
    - *Describe the purpose of every variable*
    - *Explain the algorithm you are using for solution*
    - *Add proper comments for all methods. Include @param, @return, and @throws tags*
- *Uncommented work loses up to **25% of the grade.***
- *Archive the entire project using ZIP or RAR utility. Turn it in into the digital drop box.*

Create class called Recursive and write all methods below as public static methods of that class.
Requirement: Recursive class must not have any fields.

Recursive productOfEvens()

Write a recursive method `productOfEvens(int n)` that returns the product of the first n even integers. See sample method calls below. The method must throw an `IllegalArgumentException` if the value of argument passed to it is less than or equal to 0.

```
productOfEvens(1) = 2
productOfEvens(2) = 8     (2 * 4 = 8)
productOfEvens(3) = 48    (2 * 4 * 6 = 48)
productOfEvens(4) = 384   (2 * 4 * 6 * 8 = 384)
```

Requirement: Your method must not use any global variables or static fields. Only local variables can be used in the method implementation.

Test the method in main(). Hardcode all your test cases so that I can see how you tested the method.

Add this new method to the Recursive class.

Recursive Sum of Array Elements

Write a recursive static method `int recArraySum(int[] nums, int i)` that accepts an array of integers and an index i and returns the sum of all elements of that array up to and including element with the index i.
Test the method in main(). Hardcode all your test cases.

Requirement: Your method must not use any global variables or static fields. Only local variables can be used in the method implementation.

Add this new method to the Recursive class.

Recursive Selection Sort

Convert the regular iterative selection sort method into the recursive static one. In order to achieve that, replace the outer loop in the solution by recursive calls and leave the second (inner) loop to be more or less as it is.
Test your method in main(). Make sure that your code works for arrays of size 1, 2, and 3 as well as a regular unsorted array. Do not use user input when testing the method – hard-code all the test cases.

Requirement: Your method must not use any global variables or fields. Only local variables can be used in the method implementation.

Add this new method to the Recursive class.

Recursive countDigitMatches()

Write a recursive method countDigitMatches(int n1, int n2) that accepts two non-negative integers as parameters and returns the number of digits that match between them. Two digits match if they are equal and have the same position relative to the end of the number (i.e. starting with the ones digit). In other words, the method should compare the last digits of each number, the second-to-last digits of each number, the third-to-last digits of each number, and so forth, counting how many pairs match. For example, for the call of countDigitMatches (1072503891, 62530841), the method would compare as follows:

```
1 0 7 2 5 0 3 8 9 1
    | | | | | | | |
    6 2 5 3 0 8 4 1
```

The method should return 4 in this case because 4 of these pairs match (2-2, 5-5, 8-8, and 1-1). See more examples below.
The method should throw an IllegalArgumentException if either of the two parameters is negative.

Test the method in main(). Hardcode all your test cases so that I can see how you tested the method.

Requirements:

1. Your method must not use any global variables or fields. Only local variables can be used in the method implementation.
2. You are not allowed to construct any objects other than Strings (no array, List, Scanner, etc.) and you may not use any loops to solve this problem; you must use recursion.

Add this new method to the Recursive class.

Recursive generatePattern()

Write a method generatePattern(int n) that accepts an integer n as a parameter and returns a string containing a symmetric sequence of n numbers as in example below. The method should throw an IllegalArgumentException a value less than 1 is passed as an argument.

Call	**Returned String**
`generatePattern(1);`	`1`
`generatePattern(2);`	`1 1`
`generatePattern(3);`	`2 1 2`
`generatePattern(4);`	`2 1 1 2`
`generatePattern(5);`	`3 2 1 2 3`
`generatePattern(6);`	`3 2 1 1 2 3`
`generatePattern(7);`	`4 3 2 1 2 3 4`
`generatePattern(8);`	`4 3 2 1 1 2 3 4`
`generatePattern(9);`	`5 4 3 2 1 2 3 4 5`
`generatePattern(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 1s in the middle.

Test the method in main(). Hardcode all your test cases so that I can see how you tested the method.

Requirement: Your method must not use any global variables or fields. Only local variables can be used in the method implementation.

Add this new method to the Recursive class.