Задачи для курса Java-basic - рекурсия

1. Given n of 1 or more, return the factorial of n, which is n * (n-1) * (n-2) ... 1. Compute the result recursively (without loops).
   Дано N большее либо равное 1. Вернуть факториал N = n * (n-1) * (n-2) .... 1. Вычислить результат рекурсивно (без циклов).

   ```
   factorial(1) → 1
   factorial(2) → 2
   factorial(3) → 6
   ```

2. We have bunnies standing in a line, numbered 1, 2, ... The odd bunnies (1, 3, ..) have the normal 2 ears. The even bunnies (2, 4, ..) we'll say have 3 ears, because they each have a raised foot. Recursively return the number of "ears" in the bunny line 1, 2, ... n (without loops or multiplication).
   У нас есть очередь из кроликов и мы получаем количество стоящих в этой очереди кроликов. У нечётных кроликов по два уха. А у чётных кроликов вроде как по три уха. Это из-за того, что они все подняли ногу. Посчитайте количество видимых нами "ушей". Не использовать циклы или умножение.

   ```
   bunnyEars2(0) → 0
   bunnyEars2(1) → 2
   bunnyEars2(2) → 5
   ```

3. Given a non-negative int n, return the count of the occurrences of 7 as a digit, so for example 717 yields 2. (no loops). Note that mod (%) by 10 yields the rightmost digit (126 % 10 is 6), while divide (/) by 10 removes the rightmost digit (126 / 10 is 12). Дано положительное целочисленное N. Верните количество семёрок в числе. 717 - вернёт 2. Обратите внимание, что модуло 10 даёт вам правую цифру(126 % 10 равно 6), а деление на 10 удалит правую цифру(126 / 10 равно 12). Авторы задачи были к нам слишком добры и практически решили её прямов условии.

   ```
   count7(717) → 2
   count7(7) → 1
   count7(123) → 0
   ```

4. Given a string, compute recursively (no loops) the number of lowercase 'x' chars in the string.
   Получив строку, вычислите рекурсивно (без циклов) количество строчных символов 'x' в строке.

   ```
   countX("xxhixx") → 4
   countX("xhixhix") → 3
   countX("hi") → 0
   ```

5. Given a string, compute recursively (no loops) a new string where all appearances of "pi" have been replaced by "3.14".
   Получив строку, вычислите рекурсивно (без циклов) новую строку, в которой все появления "pi" были заменены на "3.14".

   ```
   changePi("xpix") → "x3.14x"
   changePi("pipi") → "3.143.14"
   changePi("pip") → "3.14p"
   ```

6. Given a string, compute recursively a new string where identical chars that are adjacent in the original string are separated from each other by a "*".

```
pairStar("hello") → "hel*lo"
pairStar("xxyy") → "x*xy*y"
pairStar("aaaa") → "a*a*a*a"
```

7. Count recursively the total number of "abc" and "aba" substrings that appear in the given string.

```
countAbc("abc") → 1
countAbc("abcxxabc") → 2
countAbc("abaxxaba") → 2
```

8. Given a string, compute recursively the number of times lowercase "hi" appears in the string, however do not count "hi" that have an 'x' immedately before them.

```
countHi2("ahixhi") → 1
countHi2("ahibhi") → 2
countHi2("xhixhi") → 0
```

9. Given a string and a non-empty substring sub, compute recursively the number of times that sub appears in the string, without the sub strings overlapping.

```
strCount("catcowcat", "cat") → 2
strCount("catcowcat", "cow") → 1
strCount("catcowcat", "dog") → 0
```

10. We have a number of bunnies and each bunny has two big floppy ears. We want to compute the total number of ears across all the bunnies recursively (without loops or multiplication).

```
bunnyEars(0) → 0
bunnyEars(1) → 2
bunnyEars(2) → 4
```

11. We have triangle made of blocks. The topmost row has 1 block, the next row down has 2 blocks, the next row has 3 blocks, and so on. Compute recursively (no loops or multiplication) the total number of blocks in such a triangle with the given number of rows.

```
triangle(0) → 0
triangle(1) → 1
triangle(2) → 3
```

12. Given a non-negative int n, compute recursively (no loops) the count of the occurrences of 8 as a digit, except that an 8 with another 8 immediately to its left counts double, so 8818 yields 4. Note that mod (%) by 10 yields the rightmost digit (126 % 10 is 6), while divide (/) by 10 removes the rightmost digit (126 / 10 is 12).

```
count8(8) → 1
count8(818) → 2
count8(8818) → 4
```

13. Given a string, compute recursively (no loops) the number of times lowercase "hi" appears in the string.

```
countHi("xxhixx") → 1
countHi("xhixhix") → 2
countHi("hi") → 1
```

14. Given a string, compute recursively a new string where all the 'x' chars have been removed.

```
noX("xaxb") → "ab"
noX("abc") → "abc"
noX("xx") → ""
```

15. Given a string, compute recursively a new string where all the lowercase 'x' chars have been moved to the end of the string.

```
endX("xxre") → "rexx"
endX("xxhixx") → "hixxxx"
endX("xhixhix") → "hihixxx"
```

16. Given a string, compute recursively (no loops) the number of "11" substrings in the string. The "11" substrings should not overlap.

```
count11("11abc11") → 2
count11("abc11x11x11") → 3
count11("111") → 1
```

17. Given a string that contains a single pair of parenthesis, compute recursively a new string made of only of the parenthesis and their contents, so "xyz(abc)123" yields "(abc)".

```
parenBit("xyz(abc)123") → "(abc)"
parenBit("x(hello)") → "(hello)"
parenBit("(xy)1") → "(xy)"
```

18. Given a string and a non-empty substring sub, compute recursively if at least n copies of sub appear in the string somewhere, possibly with overlapping. N will be non-negative.

```
strCopies("catcowcat", "cat", 2) → true
strCopies("catcowcat", "cow", 2) → false
strCopies("catcowcat", "cow", 1) → true
```

19. The fibonacci sequence is a famous bit of mathematics, and it happens to have a recursive definition. The first two values in the sequence are 0 and 1 (essentially 2 base cases). Each subsequent value is the sum of the previous two values, so the whole sequence is: 0, 1, 1, 2, 3, 5, 8, 13, 21 and so on. Define a recursive fibonacci(n) method that returns the nth fibonacci number, with n=0 representing the start of the sequence.

```
fibonacci(0) → 0
fibonacci(1) → 1
fibonacci(2) → 1
```

20. Given a non-negative int n, return the sum of its digits recursively (no loops). Note that mod (%) by 10 yields the rightmost digit (126 % 10 is 6), while divide (/) by 10 removes the rightmost digit (126 / 10 is 12).

```
sumDigits(126) → 9
sumDigits(49) → 13
sumDigits(12) → 3
```

21. Given base and n that are both 1 or more, compute recursively (no loops) the value of base to the n power, so powerN(3, 2) is 9 (3 squared).

```
powerN(3, 1) → 3
powerN(3, 2) → 9
powerN(3, 3) → 27
```

22. Given a string, compute recursively (no loops) a new string where all the lowercase 'x' chars have been changed to 'y' chars.

```
changeXY("codex") → "codey"
changeXY("xxhixx") → "yyhiyy"
changeXY("xhixhix") → "yhiyhiy"
```

23. Given a string, compute recursively a new string where all the adjacent chars are now separated by a "*".

```
allStar("hello") → "h*e*l*l*o"
allStar("abc") → "a*b*c"
allStar("ab") → "a*b"
```

24. We'll say that a "pair" in a string is two instances of a char separated by a char. So "AxA" the A's make a pair. Pair's can overlap, so "AxAxA" contains 3 pairs -- 2 for A and 1 for x. Recursively compute the number of pairs in the given string.

```
countPairs("axa") → 1
countPairs("axax") → 2
countPairs("axbx") → 1
```

25. Given a string, return recursively a "cleaned" string where adjacent chars that are the same have been reduced to a single char. So "yyzzza" yields "yza".

```
stringClean("yyzzza") → "yza"
stringClean("abbbcdd") → "abcd"
stringClean("Hello") → "Helo"
```

26. Given a string, return true if it is a nesting of zero or more pairs of parenthesis, like "(())" or "((()))". Suggestion: check the first and last chars, and then recur on what's inside them.

```
nestParen("(())") → true
nestParen("((()))") → true
nestParen("(((x))") → false
```

27. Given a string and a non-empty substring sub, compute recursively the largest substring which starts and ends with sub and return its length.

```
strDist("catcowcat", "cat") → 9
strDist("catcowcat", "cow") → 3
strDist("cccatcowcatxx", "cat") → 9
```

Рекурсия с массивами

1. Given an array of ints, compute recursively if the array contains somewhere a value followed in the array by that value times 10. We'll use the convention of considering only the part of the array that begins at the given index. In this way, a recursive call can pass index+1 to move down the array. The initial call will pass in index as 0.

```
array220([1, 2, 20], 0) → true
array220([3, 30], 0) → true
array220([3], 0) → false
```

2. Given an array of ints, compute recursively if the array contains a 6. We'll use the convention of considering only the part of the array that begins at the given index. In this way, a recursive call can pass index+1 to move down the array. The initial call will pass in index as 0.

```
array6([1, 6, 4], 0) → true
array6([1, 4], 0) → false
array6([6], 0) → true
```

3. Given an array of ints, compute recursively the number of times that the value 11 appears in the array. We'll use the convention of considering only the part of the array that begins at the given index. In this way, a recursive call can pass index+1 to move down the array. The initial call will pass in index as 0. `array11([1, 2, 11], 0) → 1 array11([11, 11], 0) → 2 array11([1, 2, 3, 4], 0) → 0`