Assignment objectives

Design and implement a recursive algorithm in the pattern of "Decrease by a constant", a type of Decrease and Conquer algorithm.

Introduction/backstory

No matter how it looks or feels outside today, springtime is definitely coming, and Santa's elves are returning from their post-holiday vacations to begin preparations for the next year of efforts on behalf of children around the world. The first step is to decorate Santa's workshop. And the first step of this is hanging modern, programmable, LED holiday lights!

"Programming" the light-string controllers is a bit of an overstatement. They require only a list of character strings. Each character string designates the colours of all the bulbs on the string at one instant.

Each light bulb can display 3 different colours (red, white, green). So every character string will use (up to) 3 different characters. For simplicity, we will use 'A' and 'B' and 'C'. For example, the following character string for 8 lights means that the 1st, 3rd, 5th, and 7th lights are red and the rest are green:

```
ACACACAC
```

If given a *list* of such strings, the controller will display them one after the other, in the order that they appear, so you can have animated blinking lights! Here is a list of three sequences that (if played repeatedly) will make the 3 colours look like they are travelling along the string. This also is for a string of 8 lights.

```
ABCABCAB
BCABCABC
CABCABCA
```

Santa's request (demand)

Santa Claus loves *symmetry*, and he also loves *variety*. So he wants to see the coloured light strings displaying palindromes, and he wants to see *all* of the possible palindromes.

You must write a function that will take an integer N and produce *all the palindromes of length N that contain (up to) 3 different letters.*

For example, if this function is called with N=3, it will return the following 9 sequences:

```
[AAA, ABA, ACA, BAB, BBB, BCB, CAC, CBC, CCC]
```

For N=4 there are also 9 palindromic sequences:

```
[AAAA, ABBA, ACCA, BAAB, BBBB, BCCB, CAAC, CBBC, CCCC]
```

For N=5 there are 27 sequences:

```
[AAAAA, AABAA, AACAA, ABABA, ABBBA, ABCBA, ACACA, ACBCA, ACCCA, BAAAB,
      BABAB, BACAB, BBABB, BBBBB, BBCBB, BCACB, BCBCB, BCCCB, CAAAC,
      CABAC, CACAC, CBABC, CBBBC, CBCBC, CCACC, CCBCC, CCCCC]
```

It is okay if the sequences appear in your own output in a different order, as long as they include the correct set of sequences.

Decrease and Conquer

Your algorithm for generating light sequences *must* follow the recursive decrease-and-conquer pattern that we have used extensively in class.

1. (Base case) If the problem is trivial in size, manually construct the answer and return it.
2. (General case) Suppose the size of the input is N.
    a. Recursively call the algorithm to get results for a problem of size N-a. IMPORTANT: These results must be fully generated first and saved in a temporary list before continuing with the calculations.
    b. Perform additional computation that uses these results to produce a solution to the problem of size N.
    c. Return the extended results.

Program specifications

Note: these specifications are strictly required for us to grade your program!

Class name: `Lab2`
Note the exact spelling and capitalization.

Required method: `public ArrayList<String> generatePalindromeSequences (int)`
Note the exact spelling, capitalization, and signature.

Argument: An integer, the length of one sequence of coloured lights. You may assume the given value is >= 1.

Return value: An ArrayList containing all palindromic sequences of the given length.

Other Notes and Strict Requirements

1. This function should not do any console output. The list of strings is simply returned as the return value.
2. You do not need, and must not use, any math (java.math.*, etc.) while solving this problem. Yes: there is a simple arithmetic formula that tells how many sequences there will be as a function of N. But you do not need this. Do not use any such tricks to count the number of strings you are building, or to guide your algorithm to produce the correct number of strings. If your recursive algorithm is written correctly, the correct number of strings on your list will "just happen".
3. Also, you should not need to have any code that "tests" a String to see if it is a palindrome. This is an extremely *inefficient* method to generate palindromes, essentially adding a brute-force algorithm *embedded* inside your own algorithm. Once again, if your recursive algorithm is written correctly, there will only ever be palindromes added to your list to begin with.
4. You may not use any Java collection types besides ArrayList.
5. **IMPORTANT: You must not use any "helper function" to implement the recursion for this algorithm. The public method must *call itself* to solve a subproblem.**

6. Your lists of strings (both the "temporary" list and the result list that is to be returned) must NOT be "global" or member variables of your class. They must be local variables *inside* the method.

Except for the rules above, you may use any other private (or public) helper functions that you wish to have.

You may have a main function in your code for testing purposes, or you may use a separate driver class. But this is not required, and we will not call or use your main function for marking.

Here is some output intended to show you how many sequences should be on the results list for different values of input N. This output would be what you might see from a driver/testing program. You do not need to produce output like this.

```
Generating ALL sequences of a given length:
Length 1 produces 3 sequences.
Length 2 produces 3 sequences.
Length 3 produces 9 sequences.
Length 4 produces 9 sequences.
Length 5 produces 27 sequences.
Length 6 produces 27 sequences.
Length 7 produces 81 sequences.
Length 8 produces 81 sequences.
Length 9 produces 243 sequences.
Length 10 produces 243 sequences.
Length 11 produces 729 sequences.
Length 12 produces 729 sequences.
Length 13 produces 2187 sequences.
Length 14 produces 2187 sequences.
Length 15 produces 6561 sequences.
Length 16 produces 6561 sequences.
Length 17 produces 19683 sequences.
Length 18 produces 19683 sequences.
Length 19 produces 59049 sequences.
Length 20 produces 59049 sequences.
```

Submission information

Due date: As shown on Learning Hub.

Submit the following to the drop box on Learning Hub:

- Just your Java source code (*.java file).
- File name is not important.
- Please *do not zip* or otherwise archive your code. Plain Java files only.
- Please *do not zip* or include your entire project directory.

Marking information

This lab is worth 20 points. 5/20 of these points are reserved for compliance with the COMP 3760 Coding Requirements.

Virtual donut problem!!

🍩 *This part is not required; it is just for fun.* 🍩

Write another method that takes two parameters:

- int N, the same as before
- int numColours, which specifies how many different colours each individual light bulb can be, *i.e.*, instead of 'A', 'B', 'C', you might have more (or fewer) different letters.

Important: if you do this part of the assignment, be sure you write it as a separate method! Our testing code still needs to be able to call your regular `generatePalindromeSequences()` method for marking purposes. You can choose any name for the bonus problem.

If you answer this part of the problem, write a note when you submit your code, to be sure that we will notice and look at it.