CSE 2221 - Project 9

Task
Get familiar with **Set** objects and their respective methods. Get familiar with writing more **JUnit** test cases. Light exposure to computational biology from the Human Genome Project.

Original Project Instructions
Project 9 Instructions from CSE2221 Project Site

Program Requirements

- Implement the combination method according to the method contract
- Implement the addToSetAvoidingSubstrings method according to the method contract
- Implement the linesFromInput method according to the method contract
- Implement the printWithLineSeparators method according to the method contract
- Create test cases for each of the methods you have implemented

Tips, Rules, & Things to Note

- On all methods you have to implement: just follow the method contract (JavaDoc) and it should not be too difficult to understand what is being asked of you
- If you are thinking recursion on any of these methods, you are thinking wrong
- (For Logan's class only) Some of your test case method names might be super long, in some cases it might be better to have "testCombination_largeOverlap", etc.
- Yes, it is tedious but you are expected to have several test cases for the linesFromInput and printWithLineSeparators methods
- combination: you are allowed ONE line of code, no more
- combination: use the substring method as mentioned
- addToSetAvoidingSubstrings: when it says "precondition not checked", ignore it, you don't have to do anything(just implement the method, you can assume the precondition is true without it being explicitly checked)
- addToSetAvoidingSubstrings: reread the method description in the JavaDoc
- addToSetAvoidingSubstrings: use the indexOf method as mentioned
- addToSetAvoidingSubstrings: (for Logan's class only) you are allowed one return statement in this method if you want (use wisely, this can either do almost nothing but have extremely efficient code or can severely hurt you), however if you want to be on the safe side, just don't have any return in the method and proceed like you normally would according to the CSE2221 rules
- addToSetAvoidingSubstrings: if you do add a return statement, you must provide an explanation why it is a good idea to place the return where you did
- linesFromInput: keep the SimpleReader atEOS method in mind
- linesFromInput: this should be 5-6 lines of code max
- printWithLineSeparators: you are not allowed to use the replaceAll String method
- printWithLineSeparator: you are not allowed to have "\n" in the method

- Recall that the greedy solution does not always result in an optimal result, declaration-50-8.txt and gettysburg-30-4.txt will not be fully reassembled by our implementation (for added challenge, try to determine why it occurs for these files but not the others).

Data

You can find some example txt files in this directory here. Download them and put them in the *data* directory. To then input them, the path you need to provide to console is “data/<file_name>.txt”. You should additionally try to create your own txt files for testing.

For files with over 1000 lines, you can expect to wait a minute or more to completely reassemble the strings.

Steps

1. Go read all of the feedback your graders gave you for projects 2-8
2. Copy and paste the *ProjectTemplate* project to create a new project folder for this project
3. Name the project *StringReassemblyFromFragments*
4. Open the *src* folder, then open *(default package)*
5. Rename any ONE file to *StringReassembly.java*
6. Delete the other files
7. Right-click the *test* folder and create a new JUnit test fixture, name it *StringReassemblyTest.java*
8. Go reread all of the feedback your graders gave you for projects 2-8
9. Open *StringReassembly.java*
10. Go to this page and copy and paste the source code there into *StringReassembly.java*
11. Read the math definitions at the top and read the JavaDocs and method contracts for EVERY method in the program
12. Look at and understand the JavaDocs and the implementation for each of the methods already implemented (overlap, bestOverlap, assemble, and main)
13. After reading the JavaDocs and method contracts, create test cases for each of the methods you will implement (explain why each test is necessary)
14. Trace the following code:

    ```
    if(readFeedback && readJavaDocsAndContracts) {
         continueToNextStep();
    } else {
         rereadGradersFeedback();
         rereadJavaDocsAndMethodContracts();
    }
    ```

15. Implement the methods mentioned in the Program Requirements section
16. Once finished implementing all of the methods, go reread the method contracts and make sure you implemented them correctly
17. Once finished implementing all of the methods correctly, go reread your feedback from previous projects

18. Ask yourself: have you improved your code style & quality from previous projects? Is your current solution the most elegant or the most efficient? Is there any way you can make your code better? I guarantee the answer is yes, there is a way to make your code better

19. Make sure all of the test cases you have made have the correct end result and all of your test cases pass

20. Once completed and convinced your code looks good: Zip it up with the naming scheme I recommend and submit to Carmen