Part 1: Finding a Gene - Using the Simplified Algorithm

This assignment is to write the code from the lesson from scratch by following the steps below. This will help you see if you really understood how to put the code together, and might identify a part that you did not fully understand. If you get stuck, then you can go back and watch the coding videos that go with this lesson again. We recommend you try this with many of the future Java coding examples before starting programming exercises.

Specifically, you should do the following:

1. Create a new Java project named StringsFirstAssignments. You can put all the classes for this programming exercise in this project.

2. Create a new Java Class named Part1. The following methods go in this class.

3. . Write the method findSimpleGene that has one String parameter dna, representing a string of DNA. This method does the following:

- Finds the index position of the start codon “ATG”. If there is no “ATG”, return the empty string.
- Finds the index position of the first stop codon “TAA” appearing after the “ATG” that was found. If there is no such “TAA”, return the empty string.
- If the length of the substring between the “ATG” and “TAA” is a multiple of 3, then return the substring that starts with that “ATG” and ends with that “TAA”.

4. Write the void method testSimpleGene that has no parameters. You should create five DNA strings. The strings should have specific test cases, such as: DNA with no “ATG”, DNA with no “TAA”, DNA with no “ATG” or “TAA”, DNA with ATG, TAA and the substring between them is a multiple of 3 (a gene), and DNA with ATG, TAA and the substring between them is not a multiple of 3. For each DNA string you should:

- Print the DNA string.
- See if there is a gene by calling findSimpleGene with this string as the parameter. If a gene exists following our algorithm above, then print the gene, otherwise print the empty string.

Part 2: Finding a Gene - Using the Simplified Algorithm Reorganized

This assignment will determine if a DNA strand has a gene in it by using the simplified algorithm from the lesson, but organizing the code in a slightly different way. You will modify the method

findSimpleGene to have three parameters, one for the DNA string, one for the start codon and one for the stop codon.

Specifically, you should do the following:

1. Create a new Java Class named Part2 in the StringsFirstAssignments project.

2. Copy and paste the two methods findSimpleGene and testSimpleGene from the Part1 class into the Part2 class.

3. The method findSimpleGene has one parameter for the DNA string named dna. Modify findSimpleGene to add two additional parameters, one named startCodon for the start codon and one named stopCodon for the stop codon. What additional changes do you need to make for the program to compile? After making all changes, run your program to check that you get the same output as before.

4. Modify the findSimpleGene method to work with DNA strings that are either all uppercase letters such as “ATGGGTTAAGTC” or all lowercase letters such as “gatgctataat”. Calling findSimpleGene with “ATGGGTTAAGTC” should return the answer with uppercase letters, the gene “ATGGGTTAA”, and calling findSimpleGene with “gatgctataat” should return the answer with lowercase letters, the gene “atgctataa”. HINT: there are two string methods toUpperCase() and toLowerCase(). If dna is the string “ATGTAA” then dna.toLowerCase() results in the string “atgtaa”.

Part 3: Problem Solving with Strings

This assignment will give you additional practice using String methods. You will write two methods to solve some problems using strings and a third method to test these two methods.

Specifically, you should do the following:

1. Create a new Java Class named Part3 in the StringsFirstAssignments project. Put the following methods in this class.

2. Write the method named twoOccurrences that has two String parameters named stringa and stringb. This method returns true if stringa appears at least twice in stringb, otherwise it returns false. For example, the call twoOccurrences(“by”, “A story by Abby Long”) returns true as there are two occurrences of “by”, the call twoOccurrences(“a”, “banana”) returns true as there are three occurrences of “a” so “a” occurs at least twice, and the call twoOccurrences(“atg”, “ctgtatgta”) returns false as there is only one occurence of “atg”.

3. Write the void method named testing that has no parameters. This method should call twoOccurrences on several pairs of strings and print the strings and the result of calling

twoOccurrences (true or false) for each pair. Be sure to test examples that should result in true and examples that should result in false.

4. Write the method named lastPart that has two String parameters named stringa and stringb. This method finds the first occurrence of stringa in stringb, and returns the part of stringb that follows stringa. If stringa does not occur in stringb, then return stringb. For example, the call lastPart(“an”, “banana”) returns the string “ana”, the part of the string after the first “an”. The call lastPart(“zoo”, “forest”) returns the string “forest” since “zoo” does not appear in that word.

5. Add code to the method testing to call the method lastPart with several pairs of strings. For each call print the strings passed in and the result. For example, the output for the two calls above might be:

- The part of the string after an in banana is ana.
- The part of the string after zoo in forest is forest.

Part 4: Finding Web Links

Write a program that reads the lines from the file at this URL location, http://www.dukelearntoprogram.com/course2/data/manylinks.html, and prints each URL on the page that is a link to youtube.com. Assume that a link to youtube.com has no spaces in it and would be in the format (where [stuff] represents characters that are not verbatim): “http:[stuff]youtube.com[stuff]”

Here are suggestions to get started.

1. Create a new Java Class named Part4 in the StringsFirstAssignments project and put your code in that class.

2. Use URLResource to read the file at http://www.dukelearntoprogram.com/course2/data/manylinks.html word by word.

3. For each word, check to see if “youtube.com” is in it. If it is, find the double quote to the left and right of the occurrence of “youtube.com” to identify the beginning and end of the URL. Note, the double quotation mark is a special character in Java. To look for a double quote, look for (\”), since the backslash (\) character indicates we want the literal quotation marks (“) and not the Java character. The string you search for would be written “\”” for one double quotation mark.

4. In addition to the String method indexOf(x, num), you might want to consider using the String method lastIndexOf(s, num) that can be used with two parameters s and num. The parameter s is the string or character to look for, and num is the last position in the string to look for it. This method returns the last match from the start of the string up to the num position, so it is a good option for

finding the opening quotation mark of a string searching backward from “youtube.com.” More information on String methods can be found in the Java documentation for Strings: http://docs.oracle.com/javase/7/docs/api/java/lang/String.html.

Caution: The word Youtube could appear in different cases such as YouTube, youtube, or YOUTUBE. You can find the URLs more easily by converting the string to lowercase. However, you will need the original string (with uppercase and lowercase letters) to view the YouTube URL to answer a quiz question because YouTube links are case sensitive. The link https://www.youtube.com/watch?v=ji5_MqicxSo is different than the link https://www.youtube.com/watch?v=ji5_mqicxso, where all the letters are lowercase.