KIT101 Programming Fundamentals

PP Task 9.1 Divide and Conquer

Overview

Purpose: Apply your functional decomposition skills to a new task with familiar elements.

Task: Implement a DNA sequence (i.e., String) manipulation program consisting of a number of relatively small methods.

Learning Outcomes: 2 3 4 5

Time: Aim to complete this task before the start of Week 10.

Resources:

- Introductory Programming Notes:
  - 05 Using Objects
  - 07 Methods in Self-contained Programs
  - 08 Making Decisions
  - 09 Repeating Actions with Loops
  - 13 Functional Decomposition
- Online Mini-lectures:
  - Making your own methods: ...in self-contained programs

Submission Details

Upload the following to the MyLO submission folder for this task:

- The **source code** for your program
- A **screenshot** of the program during execution
- An image (photo or scan) of the **structure chart** describing your program

Assessment Criteria

A Completed submission will:

- Implement all required functionality
- Import only those additional classes that it actually uses

- Have little code in main() (likely less than 10 lines of code, but also more than just one)
- Exhibit good use of functional decomposition, with a number of small methods to solve different parts of the task
- Show the connections between methods in your program in the structure chart
- Make use of String methods wherever suitable
- Follow the unit's coding style for layout, variable names (use of case), and commenting (including your name at the top)
- Compile and run, with the screenshot showing that it works

Instructions

In this task you will implement a single-source file program called **DNA Sequencer**, which will perform text operations on a String representing a DNA sequence. You do not need to remember high school biology for this task. All relevant information is included here.

In this task it is left to you to decide how the program should be broken down into methods (as a rough guide, a Completed solution will have between 5 and 8 relatively small methods, including main). We will accept a broad range of solutions, not merely the sample solution we created when designing the task.

A description of the program and its functionality is given next, followed by a number of suggestions for how to implement some of the trickier functionality.

Background information about DNA

- A DNA sequence consists of a restricted alphabet of 'A', 'T', 'G' and 'C', which are known as *bases*. Your program will read a single piece of text representing such a sequence
- Because DNA sequencing technology is imperfect the sequence given to your program may include invalid characters (which your program will replace with a special character meaning 'unknown')
- DNA strands are actually made up of two complementary sequences, with strict rules about which bases appear with which other bases (*A* only ever pairs with *T*, *G* only ever pairs with *C*). Your program will be able to switch between the original sequence and its complement (more details are given later)
- A common operation on a DNA sequence is *transcription*, where a part of the sequence is converted into *messenger RNA* (mRNA), which is an intermediate step to translating the code into proteins (this second step is beyond your program). RNA, which is an older form of genetic material than DNA, uses the same alphabet as DNA except that *T* is replaced by *U*

DNA Sequencer

The program should implement the following high-level algorithm:

```
Program: DNA Sequencer

Steps:
1    Display the program's name
2    Prompt the user to enter a DNA sequence (any non-whitespace characters are allowed)
3    Convert that String to upper case and replace all non-DNA characters with the lower case letter n
4    Display the error rate (details below)
5    Do
5-1  |   Display a menu of operations for working with that DNA sequence
6    While the use has not selected quit
```

After step 3, only the 'cleaned' sequence of characters will be used. For example, if the user typed 'gaxTT/aca' then it will be displayed later as 'GAnTTnACA'. For each operation that displays some information the program should present that information with some suitable prefix, such as 'Sequence: '.

The available operations are:

- **Display the sequence**, which should display the current value of the sequence.
- **Display the error rate**, which is the proportion of invalid characters in the sequence, expressed as a percentage. In the example above this is approximately 22%. (It's OK if the program displays this with too much precision, that is, too many digits after the decimal place. We don't cover number formatting in the unit.)
- **Transcribe** the entire sequence, which should display the mRNA equivalent of the entire sequence (i.e., the sequence with all *T*s converted to *U*s)
- **Transcribe a section** of the sequence, between two points. The user should be asked to give the start and end of the section to transcribe, which will be values between 1 and the length of the sequence, inclusive. Both the start and end should be included in the transcribed section so, given the example above, if the user asked to transcribe between positions 2 and 5 the program will produce 'AnUU'.
- **Switch to the sequence's complement**, which should change the current value of the sequence to its complementary sequence and then display the new value. The rules to apply are: *A* becomes *T*, *T* becomes *A*, *G* becomes *C*, *C* becomes *G*, *n* remains unchanged. For example, the sequence above would change to CTnAAnTGT.

Note that the last operation changes the value of the sequence that the program is working with. To get back to the original sequence the user needs to select the last option again.

Implementation advice

Cleaning the input

Strings have a `replaceAll(String regex, String replacement)` method that will replace all substrings that match the given *regular expression* (a pattern that can match a variety of String values) with the given replacement. As we don't cover regular expressions in the unit, here are some patterns that you will find useful:

- `"[XYZ]"` matches any single character from the set 'X', 'Y' or 'Z'
- `"[^XYZ]"` matches any single character that is *not* in the the set 'X', 'Y' or 'Z'

You *should not* write a loop to clean the input text.

Calculating the error rate

The error rate can be calculated either with a loop (which is conceptually easier) or using a combination of String methods (which is a little shorter).

Transcribing the sequence

This is most easily done using a String method (mentioned above) instead of a loop. Remember you only need to change one of the characters.

There's another String method, discussed in the notes, that will help with transcribing a section, but remember that the user will describe positions in the sequence with values 1 and above.

Constructing the sequence's complement

This is possible with the use of several String methods, but is actually easier if you write a loop to construct a new String from the current value. Remember that, given a String variable `s`, initialised as `""`, `s += "some text";` will append *some text* to the end of `s`.

Ask us for help

We can give you additional advice if you tell us which part you're stuck on. Just ask.
