CSE 11 SS1-2022 PA2 - COVID Genomic Sequence

Due date: Tuesday, July 12 @ 11:59PM PST

No slip days or late submission

Provided Files

None.

Files to Submit

- CovidGenomeAnalysis.java
- CovidMutation.java

Goal:

Programming Assignment 2 is an introduction to loops and Strings in Java. You will use loops, String methods, and other programming techniques to complete the assignment.

Please read the **entire write-up** before getting started.

Some General Notes

- Most of these notes are necessary for autograding purposes. If any of these does not make sense, you probably aren't doing anything that it applies to. However, you should make sure that when you do learn about it later on in this quarter that you follow these notes. We cannot be lenient regarding these things because everything is autograded to ensure fairness.
- Make sure to read the autograder output after you submit to Gradescope (wait until the autograder is finished running).
- Match the specifications that we provide *exactly*, otherwise we cannot ensure that the autograder will function correctly. This includes file names, class names, method signatures, extending, throwing, etc.
- "Constants" should be defined as `private static final` variables.
- Do not use *any* static variables (other than for constants) or instance variables that are not specified in this writeup. We cannot ensure that these do not get clobbered during grading. Any extra variables used should be local only.
- Unless otherwise specified, do not add any extra `import`s or use any packages not from `java.lang` other than those implied or explicitly named by this writeup.
- Do not specify a package for your files. This will cause them to fail to compile with the autograder.
- Do not add any extra classes to your files and do not write code in files that are not specified.
- Do not call helper methods except from the class where they are implemented, as we will be using our own version of classes during grading (which will only have the instance variables and methods specified in this writeup). You shouldn't be able to call them anyway if you have declared them as `private`.
- If a behavior (say, on some specific input) is not specified, you may handle that case however you want. However, if you implement a specific behavior for some special case of a required method, do not rely on that specific behavior when using that method as a helper method (only assume that the method works as specified in the writeup).
- For the AI form, make sure you are filling them out and specifying your email address as the one linked with your Gradescope account. If you fill it out after submitting, you can either resubmit to update your score immediately or wait for us to rerun the autograder for everyone after the deadline.

Part 1: CovidGenomeAnalysis.java

```
public class CovidGenomeAnalysis {...}
```

You have just been recruited to join a lab working on the genetic sequence of COVID-19 because of the skills you demonstrated in the first PA of CSE 11. You will now use your knowledge of computer science to help scientists develop a more effective and cheaper vaccine for COVID-19.

COVID-19 is a RNA virus, meaning that its genetic sequence is made of adenine (A), cytosine (C), guanine (G), and uracil (U). This is in contrast with DNA, which has thymine (T) instead of uracil. However, because of instability of RNA, it must be converted into DNA to be sequenced (you can read more about this here). For this reason, we will use the complementary DNA for this assignment. In other words, the genome we are using will contain A, T, C, and G.

> A DNA molecule consists of two strands wound around each other, with each strand held together by bonds between the nitrogenous nucleobases. Adenine (A) pairs with thymine (T) and cytosine (C) pairs with guanine (G). Reference: https://www.genome.gov/genetics-glossary/acgt

Task

```
public class CovidGenomeAnalysis {
    ...
    public static void main(String[] args);
    ...
}
```

We want to first run some simple analyses. Given what we know about DNA, if we know the bases on one strand of the DNA, we can know which bases are on the other strand of the DNA. We also want to count the number of times a specific base appears on one strand. We will implement this in the `main()` method of the `CovidGenomeAnalysis` class in the `CovidGenomeAnalysis.java` file.

Input

Just like in PA1, we will read user input from `System.in` using a `Scanner` (remember to import it). After a user runs `java CovidGenomeAnalysis`, they will then type the input to our program. For this assignment, you can safely assume that this input will come as a single line of characters and that each character before the newline character is a capital letter A, T, C, or G.

Output

After reading in the string given as input, we will print some output to `System.out`. This printout will always be two items, separated by a single space, followed by a single newline character. The `System.out.println()` method will print whatever String is passed in followed by a newline, so if you use it, you do not need to explicitly add a newline character. If you use `System.out.print()`, you will need to add the newline character yourself. There is no invalid input case (as we are assuming the input will always follow the format above), so the first item should be the number of thymine (T) nucleotides that appear in the strand opposite the input sequence and the second item should be the strand opposite the input sequence. The opposite strand's bases should be in the order determined by the input strand and should only contain the capital characters A, T, C, and G.

Implementation

Valid Inputs

Your program should work for any input that does not have characters other than A, T, C, and G.

Testing

It would be easier to use input from a file rather than typing the input each time we want to test our code. We haven't learned file IO yet, but we can still do it with the `Scanner` from `System.in` and pipe input to `System.in` from a file.

There are many ways to do this, but the following way is pretty system agnostic. First, we will need to make some file whose raw data has the genome sequence we want to use as input (as the first line). For this purpose, we should use the .txt extension for the file name to avoid confusion. To have the .txt file be read from `System.in`, we will use a combination of the `cat` command and `|` (pipe) symbol.

Cat is short for concatenate. If we were to just do `cat file.txt`, it would display the content of the file.txt to the terminal. We instead then use `|` to redirect the output of a command to the input of another. In our case, we will be redirecting the output of the cat command into the input of our Java program.

(For Windows users, `cat` is available in PowerShell. If you are using command prompt, the equivalent command is `type`.)

Sample Test Case

- Case 1
    - **Input**: `ACGTAAGCA`
    - **Output**: `4 TGCATTCGT` (there are 4 letter `T`s in the output sequence)
    - You can run this case by having a file `sample1.txt` with the contents: `ACGTAAGCA` and running the command `cat sample1.txt | java CovidGenomeAnalysis`.

Part 2: CovidMutation.java

```
public class CovidMutation {...}
```

You are recruited to another research lab at UCSD because of your exceptional work in helping create a new vaccine. This lab is on the forefront of a COVID-19 cure using an antibody treatment. This antibody treatment works by mutilating the genome sequence of the COVID-19 virus such that the nucleotides formed by this mutilated sequence turn the virus into a clump of molecules that is easily broken down by white blood cells. The research lab has a genome sequence simulator that takes in a string of nucleobases (Adenine, Guanine, Thymine, Cytosine, Uracil) and simulates the DNA/RNA chain of that potential organism and its characteristics. The lab has found that they can control the antibodies to specifically reverse every k (some integer) nucleobases of the virus. In order to produce this treatment with the greatest efficacy possible, the research lab must find some k that has the highest rate of weakening the COVID-19 virus.

Task

```
public class CovidMutation {
    ...
    public static void main(String[] args);
    ...
}
```

Given some genome sequence, we want to find the output if we did this k-reversing. Since this technique might be used in the future, we will allow it to work for any string and hope the researchers pass in a valid genome sequence for now. Our application should be able to take any string and any integer k and reverse every k-sized chunk of the string. We will implement this in the `main()` method of the `CovidMutation` class in the `CovidMutation.java` file.

Input

We will read user input from `System.in` using a `Scanner` (remember to import it). After a user runs `java CovidMutation`, they will then type the input to our program. For this assignment, you can safely assume that this input will come as a single line of characters followed by an integer on the next line. The first line will be the string to k-reverse (all you need to know is that it is a string of characters) and the integer k in the next line will represent the "chunk size."

Output

After checking the inputs, we will print some output to `System.out`. This printout will always be a single string followed by a single newline character. In the case of invalid inputs (see below), print out the original input string (the string from the first line of input). In the case of valid inputs, print out the k-reversed version of the input string. If the length of the string is not divisible by k then you should reverse all characters in the remainder of the string after the last full chunk. If the given k is greater than the length of string, you should completely reverse the entire string (this is a special case of the case described in the previous sentence). Note that this means that the output should always have the same length as the input string.

Implementation

Valid Inputs

The first line will be any string (i.e., possibly composed of characters other than A, C, T, and G). The integer in the second line must be at least 1 to be valid.

Testing

You can use the same procedure as for Part 1. The only difference will be that the second line of your input file will contain the integer representing the chunk size.

Sample Test Case

- Case 1
    - **Input**: sequence = `ACGTAAGCA` ; k = `3`
    - **Output**: `GCAAATACG` (the k chunk size given is 3 which means we will reverse every 3 nucleotides. This gives ACG|TAA|GCA, if we are to reverse each chunk we will have GCA|AAT|ACG.)
- Case 2
    - **Input**: sequence = `ACGTAAGCA` ; k = `7`
    - **Output**: `GAATGCAAC` (the k chunk size given is 7 which means we will reverse every 7 nucleotides. This gives ACGTAAG|CA, if we are to reverse the first full chunk we will have GAATGCA. If there is a remainder left that is smaller than the k given, it should be fully reversed. In this case we have 2 nucleobases (CA) left and those should be reversed to result in GAATGCA|AC.)

Style

Coding style is an important part of ensuring readability and maintainability of your code. We will grade your code style in all submitted code files according to the style guidelines. Namely, there are a few things you must have in each file/class/method: 1. File headers 2. Class headers 3. Method headers 4. Inline comments 5. Proper indentation (do not intermingle spaces and tabs for indentation) 6. Descriptive variable names 7. No magic numbers 8. Reasonably short methods (if you have implemented each method according to specification in this write-up, you’re fine) 9. Lines shorter than 80 characters (note, tabs will be counted as 4 characters toward this limit. It is a good idea to set your tab width/size to be 4. A good way to check is using the command `grep -n '.\{81,\}' *.java` in the directory with your files, but note that this won't necessarily take tabulation into account appropriately.) 10. Javadoc conventions (@param, @return tags, /** header comments */, etc.)

A full style guide can be found here. If you need any clarifications, feel free to ask on Edstem.

Submission:

Turning in your code

Submit all of the following files to Gradescope by **Tuesday, July 12 @ 11:59PM PST** - CovidGenomeAnalysis.java - CovidMutation.java

When submitting, please wait until the autograder finishes running and read the output. **Your code must compile in the autograder in order to receive proper partial credit. Make sure that you have followed the General Notes to ensure that the autograder can run.**