Objectives

In addition to enhancing the skills that you have learned in the past laboratory assignments, the purpose of this assignment is to explore the ideas of a “class” and an “object” in the Java programming language. You will continue learning how to use the methods provided by the `java.lang.String` class to inspect and manipulate a `String` object. Specifically, you will write a Java program that manipulates `Strings` of Deoxyribonucleic acid (DNA) by appropriately using methods from the `java.util.String` and `java.util.Random` classes. Additionally, you will investigate ways to use Java classes and methods to organize a complete solution to an interdisciplinary problem.

Reading Assignment

To learn more about Java strings and random numbers, review Sections 3.1–3.5 in your textbook. You should also study all of the slides that we discussed during class. Don’t forget to examine the “GitHub Guides” if you have questions about how to create and use a Git repository. Please see the instructor if you have any questions about these readings. Finally, to explore the ethical considerations of gene editing read “What are the Ethical Concerns of Genome Editing?” from the National Human Genome Research Institute.

Accessing the Laboratory Assignment on GitHub

To access the laboratory assignment, you should go into the `#labs` channel in our Slack team and find the announcement that provides a link for it. Copy this link and paste it into your web browser. Now, you should accept the laboratory assignment and see that GitHub Classroom created a new GitHub repository for you to access the assignment’s starting materials and to store the completed version of your assignment. Specifically, to access your new GitHub repository for this assignment, please click the green “Accept” button and then click the link that is prefaced with the label “Your assignment has been created here”. If you accepted the assignment and correctly followed these steps, you should have created a GitHub repository with a name like “allegheny-computer-science-100-01-f2019/lab05-jjumadinova”. Unless you provide the instructor with documentation of the extenuating circumstances that you are facing, not accepting the assignment means that you automatically receive a failing grade for it.

Before you move to the next step of this assignment, please make sure that you read all of the content on the web site for your new GitHub repository, paying close attention to the technical details about the commands that you will type and the output that your program must produce. Now you are ready to download the starting materials to your laboratory computer. Click the “Clone or download” button and, after ensuring that you have selected “Clone with SSH”, please copy this command to your clipboard. By typing “git clone” in your terminal’s “labs” directory and then pasting in the string that you copied from the GitHub site you will download all of the code for this assignment. For instance, if the course instructor ran the “git clone” command in the terminal, it would look like:

```
git clone git@allegheny-computer-science-100-01-f2019/lab05-jjumadinova.git
```
Creating a DNA Manipulation Program

Overview

Bioinformatics is the study of biological phenomena by the use of biology, mathematics, and computer science. One of the most important study areas in bioinformatics concerns DNA. Deoxyribonucleic acid is a molecule that encodes the genetic instructions (genes) which are used by all known living organisms and many viruses to build the proteins required to sustain existence. The genes of DNA are written in the nucleotides guanine (G), adenine (A), thymine (T), and cytosine (C), (chemical compounds) which serve as the alphabet of the genetic language. Essentially, a DNA string is a string consisting of only the letters A, C, G, and T, for instance, “CAATGTCAC”. These strings encode various genetic traits such as hair color, eye color, and many others. Genes are made of DNA, with a gene consists of enough DNA to code for one protein, whereas a genome is an organisms complete set of DNA, including all of its genes.

Each DNA string has a complement formed by replacing each code letter by its complementary code. A and T are complements; so are G and C. Thus, the complement to the string “CAATGTCAC” is “GTTACAGTG”. DNA sometimes undergoes a mutation. There are three types of mutation: insertion of a new letter somewhere in the string; removal of a letter from the string; and replacement of one letter by another. Genome editing is the insertion, deletion or replacement of DNA at a specific site in the genome. The following table shows the examples of the replacement of letters and the complement of the given sequence.

<table>
<thead>
<tr>
<th>Strand</th>
<th>Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S$</td>
<td>ACGTGCTCTTGGTAC</td>
</tr>
<tr>
<td>A → T</td>
<td>TCGTGCTCTTGGTTC</td>
</tr>
<tr>
<td>T → A</td>
<td>TCGAGCCAAGGATC</td>
</tr>
<tr>
<td>C → G</td>
<td>TGGAGGGAGAGGATG</td>
</tr>
<tr>
<td>G → C</td>
<td>TGCACGGAGAACCAGT</td>
</tr>
<tr>
<td>$S$ <em>complementary</em></td>
<td>TGCACGGAGAACCATG</td>
</tr>
</tbody>
</table>

Program Requirements

You should design and implement a Java program that does the following. Note that all changes are made to the original input string — that is, they are not cumulative.

1. Read in a string of DNA from a text file named “dna_inputs.txt”. Please note this code has been provided in the template program, you should study it and see how it creates a variable and saves the DNA string from the text file into that variable. Also, you should see it output a message that includes the user’s DNA string in the terminal.

2. Find the complement of the given DNA string. You will need to declare a variable that stores the complement and then display the result in the terminal.

To find the complement you can use several applications of the String class’s “replace” method. For example, if I want to replace all characters ‘A’ with characters ‘T’ in the String variable called dna, then I will say dna.replace(‘A’, ‘T’). However, how do you keep track of which ‘T’s were in the original DNA sequence and which ones are the replaced one? You will need to devise a strategy to overcome this problem!
You can check your complementary sequence from the website (using the *Complement* option after you enter your sequence): [http://arep.med.harvard.edu/labgc/adnan/projects/Utilities/revcomp.html](http://arep.med.harvard.edu/labgc/adnan/projects/Utilities/revcomp.html).

**NOTE:** The next three parts of your program will also use Java’s `java.util.Random` class. You may want to read the textbook’s description of the inputs to the methods of the `Random` class.

3. Perform a random mutation consisting of inserting a randomly-chosen extra letter into a random position in the user’s DNA string; it must be one of the four “allowed” letters for DNA. Store the insertion mutation into a new variable and print it, ensuring that it is appropriately labeled and identifying the position of the insertion and the letter inserted.

4. Perform a random mutation consisting of removing a letter from a randomly-chosen position in the `dna`. Once again, store the result on the deletion mutation in the new variable and display it in the terminal window, making sure that it is appropriately labeled and identifying the position of the insertion and the letter removed. You should also declare and use other variables as needed to complete this task. Also, note that you can not re-use the same random position from the previous step here, you should generate a new random number.

5. Perform a random mutation consisting of altering a single letter from a randomly-chosen position in `dna`; it must be changed to a randomly-chosen letter from the set of allowed letters for DNA. Declare a variable to store your result, print it, appropriately labeled and identifying the position of the replacement, the new letter, and the letter it replaces. Also, note that you can not re-use the same random position or the same random letter from the previous step here, you should generate a new random number and a new random letter.

6. Print an exit message.

The following gives two sample runs of a previously implemented version of the program. Your output should be different! Please see the course instructor if you do not understand this program’s output.

I am going to manipulate the DNA string "actg".

Complement of ACTG is TGAC
Inserting T at position 0 gives TACTG
Deleting from position 1 gives ATG
Changing position 2 gives ACGG

Thanks for using the ManipulateDna program.

Janyl Jumadinova Mon Oct 3 11:11:34 EDT 2019
I am going to manipulate the DNA string "actg".

Enter a string containing only C, G, T, and A: actg
Complement of ACTG is TGAC
Inserting G at position 0 gives GACTG
Deleting from position 0 gives CTG
Changing position 0 gives ACTG
Thanks for using the ManipulateDna program.

In the second example, nothing was changed in the last line—the ManipulateDNA program randomly replaced the letter “A” with the letter “A”! This behavior is acceptable.

Testing your Program

If you are using Docker Desktop, you can use the following “docker run” command to start “gradle grade” as a containerized application, using the “DockaGator” Docker image available on DockerHub. You can run the following command to run the “gradle grade” on your project:

docker run --rm --name dockagator \
  -v "$(pwd)":/project \
  -v "$HOME/.dockagator":/root/.local/share \
  gatoreducator/dockagator

The aforementioned command will use "$(pwd)" (i.e., the current directory) as the project directory and "$HOME/.dockagator" as the cached GatorGrader directory. Please note that both of these directories must exist, although only the project directory must contain some content. Generally, the project directory should contain the source code and technical writing for this assignment, as provided to you through GitHub during the completion of a previous step. Additionally, the cache directory should not contain anything other than directories and programs created by DockaGator, thus ensuring that they are not otherwise overwritten during the completion of the assignment. To ensure that the previous command will work correctly, you should create the cache directory by running the command “mkdir $HOME/.dockagator”; you will only need to do this once. If the above “docker run” command does not work correctly on the Windows operating system, then you may need to instead run the following command to work around limitations in the terminal window:

docker run --rm --name dockagator \
  -v "$(pwd)":/project \
  -v "$HOME/.dockagator:/root/.local/share" \
  gatoreducator/dockagator

To enter into an “interactive terminal” in the Docker container, you can instead use the following command

docker run -it --rm --name dockagator \
  -v "$(pwd)":/project \
  -v "$HOME/.dockagator:/root/.local/share" \
  gatoreducator/dockagator /bin/bash

Now, if you want to “build” your program you can type the command “gradle build” in your terminal, thereby causing the Java compiler to check your program for errors and get it ready to run. If you get any error messages, go back into your atom text editor and try to figure out what you mis-typed and fix it. Once you have solved the problem, make a note of the error and the solution for resolving it. Re-save your program and then build it again by re-running the “gradle build”. If you cannot build ManipulateDna correctly, then please talk with a technical leader or the instructor. When all of the errors are eliminated, you can run your program by typing “gradle run” in the terminal window—this is the “execute” step that will run your program and produce the designated output.
Checking the Correctness of Your Program and Writing

As verified by Checkstyle, the code for the `src/main/java/labfive/ManipulateDna.java` file must adhere to all of the requirements in the Google Java Style Guide available at [https://google.github.io/styleguide/javaguide.html](https://google.github.io/styleguide/javaguide.html). The Markdown file that contains your reflection must adhere to the standards described in the Markdown Syntax Guide [https://guides.github.com/features/mastering-markdown/](https://guides.github.com/features/mastering-markdown/). Instead of requiring you to manually check that your deliverables adhere to these industry-accepted standards, the GatorGrader tool that you will use in this laboratory assignment makes it easy for you to automatically check if your submission meets these well-established standards for correctness. Please see the instructor if you have questions about GatorGrader.

To get started with the use of GatorGrader, type the command “gradle grade” in your terminal window. If you do have mistakes in your assignment, then you will need to review GatorGrader’s output, find the mistake, and try to fix it. Once your program is building correctly, fulfilling at least some of the assignment’s requirements, you should transfer your files to GitHub using the “git commit” and “git push” commands. For example, if you want to signal that the `src/main/java/labfive/ManipulateDna.java` file has been changed and is ready for transfer to GitHub you would first type “git commit src/main/java/labfive/ManipulateDna.java -m "Your descriptive commit message"” in your terminal, followed by typing “git push”, and then checking to see that the transfer to GitHub is successful. Remember, to correctly complete this assignment you need to commit all code and writing files to GitHub. If you notice that the network communication with GitHub did not work, then please try to determine why, asking a technical leader or the course instructor for additional assistance.

When you use the “git push” command to transfer your source code to your GitHub repository, Travis CI will initialize a “build” of your assignment, checking to see if it meets all of the requirements. If both your source code and writing meet all of the established requirements, then you will see a green ✓ in the listing of commits in GitHub after awhile. If your submission does not meet the requirements, a red ✗ will appear instead. The instructor will reduce a student’s grade for this assignment if the red ✗ appears on the last commit in GitHub immediately before the assignment’s due date. Yet, if the green ✓ appears on the last commit in your GitHub repository, then you satisfied all of the main checks, thereby allowing the course instructor to evaluate other aspects of your source code and writing, as further described in the “Evaluation” section of this assignment sheet.

Technical Writing

After you have completed the programming tasks for the assignment, please reflect on the process. You should write your reflections in a file, called `writing/reflection.md`, that uses the Markdown writing language. To complete this aspect of the assignment, you should answer the questions posed in the template `writing/reflection.md` document. In addition to regular questions, you are also asked to read about ethical concerns with genome editing and answer five questions related to it based on your experiences in the lab and understanding of ethical considerations of genome editing.

Summary of the Required Deliverables

This assignment invites you to submit, using GitHub, the following deliverables. **Both your reflection document and your `src/main/java/labfive/ManipulateDna.java` needs to contain all of your team members’ names.**

2. A complete and correct version of `src/main/java/labfive/ManipulateDna.java` that both meets all of the established requirements and produces the desired textual output in the terminal. See Section named “Program Requirements” in this document for details of this requirement.

**Evaluation of Your Laboratory Assignment**

The grade that a student receives through Sakai on this assignment will have the following components:

- **Percentage of Correct GatorGrader Checks [up to 75%]:** Students are encouraged to repeatedly try to implement a Java program that passes all of GatorGrader’s checks by, for instance, creating a program that produces the correct output. Students should also repeatedly revise their technical writing to ensure that it also passes all of GatorGrader’s checks about, for instance, the length of its content and its appropriate use of Markdown.

- **Travis CI Build Status [5%]:** Since additional checks on the source code and/or technical writing may be encoded in Travis CI’s actions and, moreover, all of the GatorGrader checks are also run in Travis CI, a portion of the students’ lab grade depends on whether their last before-the-deadline build passes and a green checkmark; appears in their GitHub commit log instead of a red mark. As with the previous grading component, students are encouraged to repeatedly revise their source code and technical writing in an attempt to get their Travis CI build to pass.

- **Mastery of Technical Writing [up to 10%]:** Students will also a portion of the lab grade when the responses to the technical writing questions presented in the ‘writing/reflection.md’ reveal a mastery of both writing skills and technical knowledge. To receive this portion of the grade, the submitted writing should have correct spelling, grammar, and punctuation in addition to following the rules of Markdown and providing technically accurate answers. Students are encouraged to ask the course instructor or a student technical leader to use the GitHub issue tracker to provide feedback on their mastery of technical writing skills.

- **Mastery of Technical Knowledge and Skills [up to 10%]:** Students will receive a portion of their assignment grade when their GitHub repository reveals that they have mastered all of the technical knowledge and skills developed during the completion of this project. As a part of this grade, the instructor will assess aspects of the project including, but not limited to, the use of effective source code comments and Git commit messages. Students are encouraged to ask the course instructor or a student technical leader to use the GitHub issue tracker to provide feedback on how well their work demonstrates mastery of the assignment’s technical knowledge and skills.