CMPSC 111
Introduction to Computer Science I
Spring 2017

Lab 6
Assigned: February 23, 2017
Due: March 2, 2017 by 2:30pm

Objectives

Along with enhancing your teamwork skills, 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 explore the fundamental approaches that use Java classes and methods to organize a solution to an interdisciplinary problem.

Important Notes

This is another team-based assignment. As in the past assignments, you must work in a team of two during the laboratory session and throughout the coming week. You may select your own partner, as long as you are not working with the same person from a previous assignment. Since this is the most challenging assignment so far this semester, you and your partner should plan your time this week accordingly and work on it incrementally. Please make sure that you use Slack and your team's Git repository to collaborate effectively. Remember, divide and conquer!

General Guidelines for Labs

- **Work on the Alden Hall computers.** If you want to work on a different machine, be sure to transfer your programs to the Alden machines and re-run them before submitting.
- **Update your repository often!** You should add, commit, and push your updated files each time you work on them. I will not grade your programs until the due date has passed.
- **Review the Honor Code policy.** You may discuss programs with others, but programs that are nearly identical to others will be taken as evidence of violating the Honor Code.

Reading Assignment

To learn more about Java strings and random numbers, review Sections 3.1–3.5 in your textbook. To learn more about how to create and use Java classes, you may also want to read Sections 4.1 and 4.2, focusing on accepted conventions for organizing a class. Finally, you should carefully study the sample program described in the section of this assignment called "Study A Sample Program".

Create a New Directory and a Java Program

You and your partner are responsible for implementing a complete solution to a real-world problem involving the analysis of DNA strings in the field of bioinformatics. To start, you and your partner should create a new Git repository hosted by Bitbucket. This new repository must adhere to the naming convention `<first-partner-username>-<second-partner-username>-cs111S2017-lab6`.

After creating the repository with the correct name, please share it with each other and with the course instructor. Now, make sure that you can both access this repository through Git commands such as `clone`, `pull`, `add`, `commit`, and `push`. Additionally, you should make your own `lab6/` directory in your personal repository and make sure that, by the end of the laboratory assignment, all of the files that you collaboratively created are also available in this directory. Please see the course instructor or one of the teaching assistants if you cannot complete these first two steps.

Study a Sample Program

Go to the shared course repository and pull `StringDemo.java`. Copy this program into the `lab6/` directory inside your own `cs111S2017-<your user name>` repository. Open this program and examine it to see what it does. Then run it a few times with different input strings. Make sure that both you both can answer all of these questions before advancing to the next part of this assignment.

Questions to Discuss with Your Group Member (your responses are not to be handed in):

- Where is the random number generator *declared*?
- Suppose you name the random number generator `wilbur` instead of `r`. What other lines in the program need to be changed? Why do you have to change these lines?
- Where is the scanner *declared*?
- Suppose you name the scanner `orville` instead of `scan`. What other lines in the program need to be changed? Why do you have to change these lines?
- Suppose the line:

  ```
  s1 = s1.toUpperCase();
  ```

  is changed to:

  ```
  s1.toUpperCase();
  ```

  Will the program still correctly display the upper-case version of `s1`?
- Do the statements `s1.toUpperCase();` and `s1.toLowerCase();` change the contents of `s1`? (See previous question for more details about this matter!)
- Suppose the user types `abcde` when asked to enter a string.
  - Write this down with numbers to indicate the positions (indices) of each character.
  - What position number is immediately to the left of the letter `a`?
  - What position number is immediately to the right of the letter `e`?
  - How many different ways are there to insert an "`x`" into the string `abcde`? List all the position numbers where this `x` could be placed. (Don't forget the beginning and end.)
  - According to the book, the `nextInt(num)` method of the `Random` class returns a random number in the range 0 to `num - 1`. List all the values that could be returned by "`r.nextInt(5)`". Please ask the course instructor if you cannot make this list.

  - Answer the question in the comment next to the statement "`location = r.nextInt(len+1);`".

- On paper, write down the string `ABCDEFG`, with position numbers above the letters.
  - Underline the portion corresponding to the expression:

    `"ABCDEFG".substring(0,3)`

  - Underline the portion corresponding to the expression:

    `"ABCDEFG".substring(3)`

  - What string do we get if we evaluate the expression:

    `"ABCDEFG".substring(0,3) + "ABCDEFG".substring(3)`

  Explain, in English words, what the following statement does:

```
s2 = s1.substring(0,location) + 'x' + s1.substring(location);
```

- What is the value of the expression `"PQRST".charAt(0)` ?
- What is the value of the expression `"PQRST".charAt(2)` ?
- What is the value of the expression `"PQRST".charAt(4)` ?
- List all possible values that can be returned by the expression `r.nextInt(5)`
- Explain, in English words, what the following statement does:

```
c = "PQRST".charAt(r.nextInt(5));
```

Don't continue with the assignment until you understand the answers to all of these questions! You can discuss these questions with your partner, the teaching assistants, and your instructor.

Creating a DNA Manipulation Program

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.

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. The following table shows the examples of the replacement of letters and

the complement of the given sequence. Do you understand what took place during each of these replacements? Make sure that you explain the meaning of each row in the table to your partner. Of course, you may talk with the course instructor or a teaching assistant or post a question in our Slack team if you find these examples of DNA manipulation to be difficult to understand.

Strand	**Sequence**
S	`ACGTGCCTCTTGGTAC`
A $\rightarrow$ T	`TCGTGCCTCTTGGTTC`
T $\rightarrow$ A	`TCGAGCCACAAGGATC`
C $\rightarrow$ G	`TGGAGGGAGAAGGATG`
G $\rightarrow$ C	`TGCACGGAGAACCATG`
$S_{complementary}$	`TGCACGGAGAACCATG`

You and your partner 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. Prompt the user to type in a string of DNA, declare a variable and save the DNA string that the user inputs into a variable called "`dnaString`".

2. Print the complement of `dnaString`, appropriately labeled. (Hint: 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 `dnaString`, then I will say `dnaString.replace('A','T');`, but there's still a trick in this case of getting the complement!)

   Another hint regarding the trick: As you make your substitutions to get your complementary string, remember that you are replacing, for example, A's to T's from the S to make our complementary sequence ($S_{complementary}$). Do not simply perform such substitution directly because you will be unable to figure out which T's were original T's (and not the replaced ones) that you were supposed to change to A's. 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`.

NOTE: The next three parts of your program will also use Java's `java.util.Random` class. You may want to read about the `Random` class at the end of this document to understand the concepts behind the `Random` class better; you may also ask your instructor for help with this Java class.

3. Perform a random mutation consisting of inserting a randomly-chosen extra letter into `dnaString`; it must be one of the four "allowed" letters for DNA. Print this, 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 `dnaString`. Once again, display this in the terminal window, making sure that it is appropriately labeled and identifying the position of the insertion and the letter removed.

5. Perform a random mutation consisting of altering a single letter from a randomly-chosen position in `dnaString`; it must be changed to a randomly-chosen letter from the set of allowed letters for DNA. Print it, appropriately labeled and identifying the position of the replacement, the new letter, and the letter it replaces. This is the last output your program must perform.

The following gives two sample runs of a previously implemented version of the program. You will get different values, of course, since the changes are random, but the structure of the output will be the same. Please see the course instructor if you do not understand this program's output.

```
aldenv5:lab6 jjumadinova$ java ManipulateDNA
Janyl Jumadinova
Lab 6
Thu Sep 30 12:51:58 EDT 2015

Enter a string containing only C, G, T, and A: actg
Complement of ACTG is TGAC
Inserting T at position 0 gives TACTG
Deleting from position 1 gives ATG
Changing position 2 gives ACGG

aldenv5:lab6 jjumadinova$ java ManipulateDNA
Janyl Jumadinova
Lab 6
Thu Sep 30 12:52:58 EDT 2015

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
```

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.

Additional Program Requirements

- As in past assignments, make sure your program prints the names of both of your team members, the lab number, and the date as the first few output lines in the terminal.
- Make sure your program contains the comment header with the honor pledge, the names of all team members, lab number, date, and the purpose of the program.
- Make sure you document your program properly, by using comments throughout your program whenever appropriate; you and your partner should both understand the code's comments.
- Make sure your output in the terminal window is neat (e.g., no missing spaces) and that the source code of your Java program is precisely formatted (e.g, indenting and blank spaces).

Summary of the Required Deliverables

In addition to turning in printed and signed versions, for this assignment you are invited to submit electronic versions of the following deliverables through the Bitbucket repository that you created

for your team. As you complete this step, you should make sure that you created a `lab6/` directory within this Git repository. Then, you can save all of the required deliverables in the `lab6/` directory—please see the course instructor or a teaching assistant if you are not able to create your directory properly. Please make sure that, along with keeping the files in your team's Git repository, that you also save and commit the file to your individual course repository; this will help you as you are reviewing for upcoming quizzes and examinations in this course.

1. A completed, properly commented, and formatted `ManipulateDNA.java` program.
2. An output document containing at least **three different** outputs obtained after running `ManipulateDNA` in the terminal window at least three times.
3. A document describing the strategy your team developed for completing this assignment and the work each team member has completed. While the `ManipulateDNA` program and the output file have to be the same for both members of the same team, each of you should forthrightly include any challenges that you individually faced in this document.

Share your program, the output file, and the document describing your team work with me through your Git repository by correctly using "`git add`", "`git commit`", and "`git push`" commands. When you are done, please ensure that the Bitbucket Web site has a `lab6/` directory in your two repositories with the two files called `ManipulateDNA.java`, `output`, and `report`. You should see the course instructor if you have any questions about assignment submission.

A Quick Review of Random Number Generation

To generate random numbers, we need an object of the `Random` class. You can name this variable anything you want — "`rand`" or "`random`" are acceptable names, but others will work as well. To create a new random number generator named `rand`, be sure to import the `Random` class:

```
import java.util.Random;
```

and then create an instance of this class in the following fashion:

```
Random rand = new Random();
```

The three most useful methods in the `Random` class are `nextInt`, `nextFloat`, and `nextDouble`. If `rand` is the name of our random number generator (it can be called something else) and `n` is a positive integer, `rand.nextInt(n)` produces an `int` in the range 0, . . . , `n`-1 and `rand.nextDouble()` and `rand.nextFloat()` produce a `double` and a `float` respectively in the range from 0 to 1 (not including 1). By being clever, we can get different ranges — here are a few examples! In the Java code, assume `i` is an `int` variable, `d` is a `double` variable, and `rand` is an object of the `Random` class.

Desired Range	Java Statement
0, 1, 2, 3, 4, 5	`i = rand.nextInt(6);`
10, 11, . . . , 19, 20	`i = rand.nextInt(11) + 10;`
-5, -4, -3, . . . , 4, 5	`i = rand.nextInt(11) - 5;`
0, 3, 6, 9, 12	`i = 3 * rand.nextInt(5);`
-1, 1	`i = 2 * rand.nextInt(2) - 1;`
$0 \leq d < 1$	`d = rand.nextDouble();`
$0 \leq d < 10$	`d = 10 * rand.nextDouble();`
$-5 \leq d < 5$	`d = 10 * rand.nextDouble() - 5;`
0.0, 0.1, 0.2, . . . , 0.9, 1.0	`d = rand.nextInt(11)/10.0;`