COMPUTER SCIENCE 12B (FALL TERM, 2017)
PROGRAMMING IN JAVA

PROGRAMMING ASSIGNMENT 5
Due Date: Monday, 16 October, 2017, 11:55pm

Overview:

This assignment will give you practice with `ArrayList`. The assignment also touches on file input/output, command line arguments, string searching and the use of randomly-generated integers.

LATTE Downloads

Download files `hamlet.txt` and `moby.txt`. These will be your source files for testing your random writing application.

Random Writing

Imagine taking a book (say, *Tom Sawyer*) and determining the probability with which each character occurs. You'd probably find that spaces are the most common, that the character 'e' is fairly common, and that the character 'q' is rather uncommon. After completing this level 0 analysis, you'd be able to produce random *Tom Sawyer* text based on character probabilities. It wouldn't have much in common with the real thing, but at least the characters would tend to occur in the proper proportion. In fact, here's an example of what you might produce:

Level 0: *rla bsht eS ststofo hhfosdsdewno oe wee h .mr ae irii ela iad o r te u t mnyto onmalysnce, ifu en c fDwn oee iteo*

Now imagine doing a slightly more sophisticated level 1 analysis by determining the probability with which each character follows every other character. You would probably discover that 'h' follows 't' more frequently than 'x' does, and you would probably discover that a space follows '.' more frequently that ',' does. You could now produce some randomly generated *Tom Sawyer* by picking a character to begin with and then always choosing the next character based on the previous one and the probabilities revealed by the analysis. Here's an example:

Level 1 *Shand tucthiney m?" le ollds mind Theybooure He, he s whit Pereg lenigabo Jodind alllld ashanthe ainofevids tre lin--p asto oun theanthadomoere*

Now imagine doing a level *k* analysis by determining the probability with which each character follows every possible sequence of characters of length *k*. A level 5 analysis of *Tom Sawyer* for example, would reveal that 'r' follows "Sawye" more frequently than any other character. After a level *k* analysis, you'd be able to produce random *Tom Sawyer* by always choosing the next character based on the previous *k* characters (the *seed*) and the probabilities revealed by the analysis.

At only a moderate level of analysis (say, levels 5--7), the randomly generated text begins to take on many of the characteristics of the source text. It probably won't make complete sense, but you'll be able to tell that it was derived from *Tom Sawyer* as opposed to, say, *The Sound and the Fury*. Here are some more examples:

Level 2 *"Yess been." for gothin, Tome oso; ing, in to weliss of an'te cle -- armit. Papper a comeasione, and smomenty, fropeck hinticer, sid, a was Tom, be suck tied. He sis tred a youck to themen*

Level 4 *en themself, Mr. Welshman, but him awoke, the balmy shore. I'll give him that he couple overy because in the slated snufflindeed structure's kind was rath. She said that the wound the door a fever eyes that WITH him.*

Level 6 *people had eaten, leaving. Come -- didn't stand it better judgment; His hands and bury it again, tramped herself! She'd never would be. He found her spite of anything the one was a prime feature sunset, and hit upon that of the forever.*

Level 8 *look-a-here -- I told you before, Joe. I've heard a pin drop. The stillness was complete, how- ever, this is awful crime, beyond the village was sufficient. He would be a good enough to get that night, Tom and Becky.*

Level 10 *you understanding that they don't come around in the cave should get the word "beauteous" was over-fondled, and that together" and decided that he might as we used to do -- it's nobby fun. I'll learn you."*

1.2 Assignment Details

You are to implement a Java client program called `RandomWriter` that provides a random writing application. Your class should have a `public main` method that takes the following four command line arguments:

- A non-negative integer *k*, referred to as the seed length.
- A non-negative integer *length*, which is the length of the output string
- The name of an input file *source* that contains more than *k* characters.
- The name of an output file *result*, to store the output random string.

New

- An integer, *myRandomSeed*, to be used as a seed to your Random class generator. Note that this is an optional argument – your program should run even when this is not provided in the command line.

Your client program should validate the arguments (more on that later on) and then proceed into generating a random sequence of *length* characters as follows:

1. Create a `String` object that contains all of the characters from the source file.
2. Choose a random character from the `String` source based on a seed of length *k* (the process is described later). The character should be written to the output *result* file. Note that if *myRandomSeed* argument is provided, then this step should run deterministically, i.e., it should always return the same character. This is achieved by using myRandomSeed as the seed when you construct any Random object you use to implement this step.

New

3. Repeat Step 2 until there are *length* characters in the result. Each of these additional characters should be chosen based on the same seed

Choosing Which Character to Append

This problem will be tackled by a class called `CharGenerator`. This class chooses the next character to add to the output for a given source and a seed length. This class has the following methods:

1. Constructor `CharGenerator(String source, int k):` The constructor creates the *seed* by picking *k* consecutive characters at random from the `source` String. The constructor should be used only when myRandomSeed argument is NOT provided.

2. Constructor `CharGenerator(String source, int k, int myRandomSeed)`: The constructor creates the *seed* by picking *k* consecutive characters at random from the `source` String. This constructor uses myRandomSeed parameter to initialize any random generators and therefore it returns always the same consecutive characters. The constructor should be used only when myRandomSeed argument is provided.

New

3. `getNextChar()`: a method that returns the next character to add to the output file for a given *k*, seed and source. The return type can be anything you choose (int, char, String), whatever works for your program.

Next we describe how the next character should be chosen.
Suppose that *k* = 2 and the source file contains

the three pirates charted that course the other day

Here is how the first three characters might be chosen:

1. A 2-character seed is chosen at random to become the initial seed. Let's suppose that "th" is chosen.
2. The first character must be chosen based on the probability that it follows the seed (currently "th") in the source. The source contains five occurrences of "th". Three times it is followed by 'e', once it is followed by 'r', and once it is followed by 'a'. Thus, the next character must be chosen so that there is a 3/5 chance that an 'e' will be chosen, a 1/5 chance that an 'r' will be chosen, and a 1/5 chance that an 'a' will be chosen. Let's suppose that we choose an 'e' this time.
3. The next character must be chosen based on the probability that it follows the seed (currently "he") in the source. The source contains three occurrences of "he". Twice it is followed by a space and once it is followed by 'r'. Thus, the next character must be chosen so that there is a 2/3 chance that a space will be chosen and a 1/3 chance that an 'r' will be chosen. Let's suppose that we choose an 'r' this time.
4. The next character must be chosen based on the probability that it follows the seed (currently "er") in the source. The source contains only one occurrence of "er", and it is followed by a space. Thus, the next character must be a space.

Note If your program ever gets into a situation in which there are no characters to choose from (which can happen if the only occurrence of the current seed is at the exact end of the source), your program should pick a new random seed (create a new `CharGenerator` object) and continue.

Implementation Approach

There is a simple way to implement the above character selection process.

1. To select the initial seed, choose a random position P in the source string between L - k, where L is the length of the text source. Then, use the source characters between (P, P+k) as the random seed.
2. To choose the next character, find each occurrence of the seed in the source and store the character that follows it into an ArrayList.
3. When you have found all occurrences, choose a character at random from the ArrayList.

4. Each time a character *c* is written to *result*, remove the first character of the seed and append *c* to the end of the seed.

Java classes

Here are some useful things to know about Java. You'll need to read the Java documentation for more details.

- A `java.util.ArrayList` can contain only objects; it cannot contain a scalar values such as a char. To store a char into an `ArrayList`, you need to wrap it in a `java.lang.Character` object.
- You can use a `java.util.Random` object to generate random integers in a specified range. Also you can use a given seed (*myRandomSeed*) to construct your Random object and this will make your generator run deterministically, i.e., return the same sequence of numbers across every execution of your program.

New

- A `java.io.FileWriter` object is useful for writing strings and characters to a file.
- A `java.io.FileReader` object is useful for reading one character at a time from a file.
- A `java.lang.StringBuffer` object is useful for efficiently composing characters into a string.
- A `java.lang.String` object contains several member functions for searching for substrings.

Input Argument validation

Your program (either of the two classes) should validate the command line arguments by making sure that *k* and *length* are non-negative, that *source* contains more than *k* characters and can be opened for reading, and that result can be opened for writing. If any of the command line arguments are invalid, your program should write an informative error message to the console and terminate. You may choose to also through an `IllegalArgumentException`.

Determinism and Testing

New

Since your program will always create random text it can be challenging to debug it as well as test its correctness. The *myRandomSeed* argument and the `CharGenerator` constructor that accepts it as an argument can assist you: they guarantee (when used correctly) to generate always the same output file. In fact when you execute your program on hemlet.txt with a *k*=15, *length*=50 and *myRandomSeed*=3 you should get the output:

```
you honest?
  Oph. My lord, I do not know, my lo
```

Submission:

Your Java source code should be submitted via LATTE. For late policy check the syllabus.

Grading:

You will be graded on

- **External Correctness:** The output of your program should match exactly what is expected. Programs that do not compile will not receive points for external correctness.
- **Internal Correctness:** Your source code should follow the stylistic guidelines shown in class. Also, remember to include the comment header at the beginning of your program.
- **One-on-one interactive grading:** By the end of the day that the assignment is due, please make an appointment with your TA for an interactive 10-15 minute grading session. You will receive an email notifying you of the name of the TA who has been assigned to you for this assignment with further instructions on setting up the appointment. (You will be meeting with a different TA for each assignment). One-on-one interactive grading will help you improve your programming skills and avoid repeating mistakes from one assignment to the next.