COMP-202A, Fall 2007, All Sections
Assignment 3

Due: Tuesday October 23, 2007 (23:55)

You **MUST** do this assignment individually and, unless otherwise specified, you **MUST** follow all the general instructions and regulations for assignments.

Note that for this and all further assignments, graders **will generously** deduct marks for assignments that do not follow instructions and regulations.

Failure to respect instructions and regulations:	0 to -10 points
Part 1:	0 points
Part 2, Question 1:	2 points
Part 2, Question 2:	8 points
	10 points total

Part 1 (0 points)

Do not hand in this part. It will not be graded. But doing this exercise might help you to do the second part of the assignment (that will be graded). If you have difficulties with the questions of Part 2, then we suggest you go to one of the office hours. The TA can help you and work with you through the warm-up questions.

Warm-up Question 1 (0 points)

The following program reads lines of text from the user until they input an empty line. It then searches the entire text for each capital letter, concatenates them together and prints out the resulting String. Unfortunately, since it used 'A' and 'Z' for comparison it won't work well with all the other alphabets unicode supports. Can you find a better way? (Hint: look at the methods in the Character class).

Every time we go through the second loop we also re-calculate the length of the *text* String. This is redundant since it does not change. Can you either change the loop structure or otherwise modify the code to improve its performance?

```
import java.util.Scanner;

public class FirstLetters {
    public static void main(String[] args) {
        String text="";
        String line,output="";
```

```
        Scanner scan = new Scanner(System.in);
        // get input text until a blank line is entered
        System.out.println("Enter a blank line to terminate:");
        do {
            line = scan.nextLine();
            text += line;
        } while(line.length()!=0);
        // accumulate all capital letters in the input text
        for (int i=0;i<text.length();i++) {
            char c = text.charAt(i);
            if (c>='A' && c<='Z') {
                output = output+c;
            }
        }
        System.out.println("Capitals: "+output);
    }
}
```

Part 2 (2 + 8 = 10 points)

The questions in this part of the assignment will be graded.

Question 1 (2 points)

Manipulating Strings in Java is easy, and String(-like) objects are common in many domains. Here you must develop a program "**Mutate.java**" that performs mutations on a simple DNA-sequence. The program should ask the user for a number, and then ask for a String. The String consists of letters ACGT and no spaces. The number indicates how many random mutations to apply to the sequence, converting a letter at a random location in the String into a random ACGT letter. You do not need to check correctness of the input, and can assume the number of mutations is ≥ 0 and the String is at least 1 character long. Output should be the final String with all randomized changes applied.

Note that as well as being familiar with the String class you will need to use a *Random* object for this question. Initialize this random number with the random seed value "3735928559" (eg create "`... = new Random(3735928559L);''`

Question 2 (8 points)

In order verify whether a text is written by a particular person people have made use of several kinds of textual analyses to show similarities and differences between writing styles. Simple approaches are based on easily calculated values such as the statistics based on word sizes. For this question you should be familiar with the functionality provided by the String, Math, and Character classes. You may *not* use String methods that use arrays or non-String Objects (eg `String.split()`), *nor* may you use `String.trim()` or other classes such as `StringTokenizer`.

Write a program "**Word.java**" that analyzes *words* in Strings. You should calculate the average, and standard deviation of the length of the *words* in the String. Words are at least one non-*whitespace* character long and are separated (only) by one or more

whitespace characters. Recall the definition of standard deviation σ:

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(w_i - \overline{w})^2}$$

where n is the total number of words, $\overline{w}$ is the average word length, and each w_i is the length of word i.

Note:

- Your solution should make good use of existing class methods, and try not to reproduce functionality that is already provided for you.
- All floating point values should be printed out with 8 digits to the right of the decimal place and at least one digit to the left.
- Make no assumptions about the String itself; it may be long, short, contain many or even no words.

Some Strings are provided for you to apply to your code. Download the class "SomeStrings" from the course website (or webct), put it in the same directory as your other DrJava files. You can then refer to 5 Strings upon which you should perform the above calculations, `SomeStrings.s1`, `SomeStrings.s2`, `SomeStrings.s3`, `SomeStrings.s4`, and `SomeStrings.s5`. You program should provide formatted output for each of these Strings showing each statistic on a separate line, eg

```
String:  s1
Average:  ...
Standard Deviation:  ...
```

Notice that as well as multiple strings to test each string will require more than one "pass" over the String to compute both statistics. Try to minimize repetition of code (while still being clear). You are allowed to use new method definitions if you wish, but you do not need to do so to solve this problem.

Develop your solution following a top down method: you need to figure out:

- how to go through a String and find and count lengths of words. Remember that words are separated by 1 or more *whitespace* characters.
- how to compute the average word length.
- how to compute the standard deviation (without duplicating code).
- how to print out the formatted output once calculated both statistics.
- how to do all the above for all 5 Strings without duplicating all your code for each separate String.

Based on examining your program output, which of the Strings do you think represent texts most likely written by the same author? As well as handing in your program provide your (explained/justified) answer in a file "**a3q2.txt**."