BSM 261: Object Oriented Programming
Programming Project 2

Due date: Sunday, November 27 at 11:59PM

1 Overview

In the second programming assignment, you practice writing loops and manipulating strings. *Arrays are not needed, and you should not use arrays in your assignment.*

Please submit your assignment to the ubs as a single zipfile containing: HW2.java and your testing report. Creating a JUnit file is optional. So, if you create a JUnit then you will submit HW2.java, HW2Junit.java, and testing document files.

2 Code Readability (20% of your project grade)

Back in time, the only goal of the programming was to get a program working. There was also less number of programmers and the complexity of the projects was not that complicated. Now, with highly complex software running much of our lives, the industry has learned that computer code is a written document that must be able to communicate to other humans what the code is doing. If the program is too hard for a human to quickly understand, the industry does not want it.

The companies in the market enforces strict rules about how the program should look. This rules are not fixed for all companies. In our class will do the same, but will not be quite as strict so you can have some freedom for developing your own style.

To receive the full readability scores, your code must follow the following guideline:
• All variables (fields, parameters, local variables) must be given appropriate and descriptive names.
• All variable and method names must start with a lowercase letter. All class names must start with an uppercase letter.
• The class body should be organized so that all the fields are at the top of the file, the constructors are next, and then the rest of the methods.
• Every statement of the program should be on it's own line and not sharing a line with another statement.
• All code must be properly indented (see Appendix F of the Lewis book for an example of good style). The amount of indentation is up to you, but it should be at least 2 spaces, and it must be used consistently throughout the code.
• You must be consistent in your use of {, }. The closing } must be on its own line and indented the same amount as the line containing the opening {.
• There must be an empty line between each method.
• There must be a space separating each operator from its operands as well as a space after each comma.
• There must be a comment at the top of the file that includes both your name and a description of what the class represents.
• There must be a comment directly above each method that, in one or two lines, states what task the method is doing, not how it is doing it. Do not directly copy the homework instructions.
• There must be a comment directly above each field that, in one line, states what the field is storing.

• There must be a comment either above or to the right of each non-field variable indicating what the variable is storing. Any comments placed to the right should be aligned so they start on the same column.

Here is the readability rubric:

Points	Metric
20	Excellent, readable code. Correct comments at top of class file, above each method and above each field. Each comment at most one or two sentences and is a good description. Class has fields first, then constructors, then the remaining methods. All code follows the proper indentation and naming style.
18	Very readable code. There are only a few places (at most 5) where there is inconsistent or missing indentation or poorly named fields/methods, or missing comments, or comments that are not good descriptions, or fields or constructors buried inside the code or missing whitespaces.
15	Reasonably readable code. Overall good commenting, naming, and indentation. However, there are more than 5 places with missing or inconsistent indentation, poor variable names, or missing, unhelpful, or very long comments OR many comments that are a direct copy of the homework description.
12	Poorly readable code. A significant number of missing comments or inconsistent and/or missing indentation or a large number of poorly named variables. However, at least half of the places that need comments have them.
10	Some commenting done (close to half), but missing majority of the comments.
5	There is at least five useful comment in the code OR there is some reasonable indentation of the code.
0	Code very unreadable. No useful comments in the code AND/OR no indentation in the code.

3 Program Testing Document (20% of your project grade)

In this case, it means documenting tests for each of the methods listed below. If you are unable to complete a method above, you should still describe tests that would test the method had it been completed.

Hints for testing loops. Your tests need to, at the minimum cover the following cases:
1: Test 0, test 1, test many: This means you have to test cases where the parameters, if they are integers, are 0, 1 or some value other than 1. If the parameters are strings, you have to test strings of length 0, 1, and more than 1. If the strings must contain certain data, you need to test cases where they contain 0, 1, and more than 1 of the desired data.

What must go in the report: For each method below, your report should describe, in English, what "**test 0, 1, many**" and **1**"test first, middle, last" mean for each of the methods. Then, you should list the specific tests, that you will do, what the expected output is, and then (if you completed the method) a cut-and-paste from DrJava showing the actual test.

JUnit: You do not have to create JUnit for this a assignment. You are welcome to use it with this homework. If you choose to write JUnit tests for your code, you do not need to include the actual tests in your report. Your JUnit file should include comments with each test that link to the report and indicate what you are testing. For example, if your report indicates that the method requires a test with a string of length 0, your JUnit class should have such a test and a comment on the test noting that it is the test of a length 0 string that your report described. Try to organize the JUnit class and report to make it easy for a reader to jump back and forth between the report and the tests.

Here is the testing rubric:

Points	Metric
20	Excellent demonstration of how to test the program. A description of how to thoroughly test all the methods of the assignment, and the tests clearly check the required 0, 1, many, and first, middle last, cases. Completed tests for each method written in the student's submission. Each test gives the true output of the student's submission.
18	Very good: Good descriptions, tests covers most of the program but a few tests were missed.
15	Reasonable demonstration of how to test the program. The tests performed and the testing description covers a majority of the program; however, multiple obvious cases were missed.
12	Poor demonstration of how to test the program. The testing report shows how to test some of the methods and conditional statements, but at least half of the needed tests are missing.
10	Some testing done (at least half), but nothing that demonstrates the proper way to test a conditional statement.
5	There is at least five proper testing of the methods/constructors in the code.
0	The testing report does not match the behavior of the student's code.
-10	Includes output for methods that were not in the student's submission OR gives results misrepresenting how the student's code works.

4 Programming (60% of your grade)

Guidelines for the programming part:

• All methods listed below must be public and static.
• If your code is using a loop to modify or create a string, you need to use the StringBuilder class from the API.
• Keep the loops simple but also efficient. Remember that you want each loop to do only one "task" while also avoiding unnecessary traversals of the data.
• No additional methods are needed. However, you may write additional private helper methods, but you still need to have efficient and simple algorithms. Do not write helper methods that do a significant number of unnecessary traversals of the data.
• **Important:** you must not use either break or continue in your code. These two commands almost always are used to compensate for a poorly designed loop. Likewise, you must not write code that mimics what break does. Instead, re-write your loop so that the loop logic does not need break-type behavior.
• While it may be tempting to hunt through the API to find classes and methods that can shorten your code, you may not do that. The first reason is that this homework is an exercise in writing loops, not in using the API. The second reason is that in a lot of cases, the API methods may shorten the writing of your code but increase its running time. The only classes and methods you can use are listed below.

You are allowed to use the following from the Java API:
• class String : length and charAt methods
• class StringBuilder : length, charAt, append, and toString methods
• class Character: any method

In the following, we define a word to be any sequence of consecutive non-whitespace characters. A whitespace character is a character such as a space, tab, or return. For the purpose of this homework, we will use only spaces for the whitespace character, but you are welcome to get your code to work for all possible whitespace character types.

Create a class called HW2 that contains the following methods:

4.1 samePrefix:

takes two String and one int parameter and returns a boolean. Let x be the input int value (though you should use a better name). The method should return true if the first x characters of each input String are exactly the same.

Example:
HW2.samePrefix("this is a test", "this is a trial", 11) $\Longrightarrow$ true
HW2.samePrefix("this is a test", "this is a trial", 12) $\Longrightarrow$ false
HW2.samePrefix("this is a test", "This is a trial", 4) $\Longrightarrow$ false
HW2.samePrefix("this is a test", "this is a test", 100) $\Longrightarrow$ false

4.2 trimSpacesFromFront:

takes a single String parameter and returns a String. The method will create a new String that is the same as the parameter String, but any whitespaces that occur before the first non-whitespace character are removed.

Example:
HW2.trimSpacesFromFront("   Good day!") $\Longrightarrow$ "Good day!"
HW2.trimSpacesFromFront("   What   a wonderful day!!!!   ") $\Longrightarrow$"What   a wonderful day!!!!   "
HW2.trimSpacesFromFront("   How   are   you   ???   ") $\Longrightarrow$ "How   are   you   ???   "

4.3 repeatChars:

takes two parameters, a String and an int and returns a String. The method will create a new String that is the same as the parameter String, but each character is repeated a number of times equal to the int parameter. The int parameter is assumed to not be negative.

Example:
HW2.repeatChars("Nice try !!", 2) $\Longrightarrow$ "NNiiccee ttrryy !!!!"
HW2.repeatChars("Good day ! ", 3) $\Longrightarrow$ "GGGoooooooddd   dddaaayyy   !!!   "
HW2.repeatChars("Hi there ?", 4) $\Longrightarrow$ "HHHHiiii   tttthhhheeeerrrreeee   ????"

4.4 countWords:

takes a single String parameter and returns an int. The method will return the number of words in the string. Basically anything between spaces (even there are multiple spaces) is a word (except from front or to back). In addition, the spaced in the front of back is not counted as word.

Example:
HW2.countWords("One fish, two fish, red fish , blue fish !") $\Longrightarrow$ 10
HW2.countWords(" Hi hello   ? ?   ,!? ") $\Longrightarrow$ 5
HW2.countWords("Actions speak louder than words !") $\Longrightarrow$ 6
HW2.countWords("X   y   z   !   ?") $\Longrightarrow$ 5

4.5 subSequence:

takes two Strings as input and returns a boolean. The method returns true if the first input string is a subsequence of the second. A subsequence of a string is a sequence of characters that occur in order, but not necessarily subsequently, in the string.

Example:
HW2.subSequence("abc", "about chocolate") $\Longrightarrow$ true

HW2.subSequence("abc", "acorn bud") $\Longrightarrow$ false

Each of the five methods will be graded as follows. While you want to create methods that work, you will also be judged on your ability to use loop design principles. There are:

• The method completes the task with a reasonable number of traversals of the data.
• The method correctly uses StringBuilder and does not create unnecessary String instances.
• The method does not use break (or break-like loop constructs), continue or anything other than the allowed methods of String, StringBuilder, and Character.

Each method: 12 points

Points	Metric
12	The method gives the correct output and follows the good design principles.
10	The method works correctly on all but a couple boundary cases (for example: the test 0, test first, or test last cases), and the method follows good design principles.
9	The method works correctly on most non-boundary cases and follows good design principles OR the method works correctly on all cases but fails to follow good design principles.
8	The method fails most cases, but the loop(s) provided is a good design for the problem and follows good loop design principles OR the method works correctly on most cases but does not follow good loop design principles.
5	The method fails most cases, but the loop provided is a reasonable loop for the problem.
3	Major problems but finds a result
0	Completely wrong

5 Course Honor Policy

See the university policy on academic integrity for general rules that you should follow on tests and quizzes. Rules specific to programming in this class are listed here.

Programming is a collaborative enterprise. However, you cannot be an equal collaborator until you first build up your own programming skill set. Because of that, it is essential that you do programming project coding on your own. That does not mean that you can not seek help or give help to other students, but the assistance must be at a high level - for example English descriptions of how to solve a problem and not coding descriptions. The most important skill you will develop in this class is to translate a solution description into the Java commands needed to implement the solution. If you copy code from other sources, you are not developing this needed skill for yourself.

Here is a short list of things you may and may not do:
• **DO NOT** send an electronic copy of your code to another student. (Exception: You may share with your lab partner a copy of the code you wrote together during the recitation section.)
• **DO NOT** look at another person's code for the purpose of writing your own.
• **DO NOT** tell another student the Java code needed for an assignment.
• **DO** sit down with a student and go over the student's code together in order to help the student find a bug.
• **DO** describe in English (not code!) the steps needed to solve an assignment problem.
• **DO** show students how to do certain Java coding tasks as long as the task does not directly relate to a homework question.
• **DO** use coding examples from the lecture and textbook as a guide in your programming.
• **DO NOT** search the internet to find the exact solutions to assignment problems.
• **DO NOT** post code you write for this class onto an internet site such as StackExchange.
• **DO NOT** post the homework specific problem descriptions onto internet sites such as StackExchange.