ASSIGNMENT 3

COMP-202, Winter 2018, All Sections

Due: Thursday, March 1st, (23:59)

Please read the entire PDF before starting. You must do this assignment individually.

It is very important that you follow the directions as closely as possible. The directions, while perhaps tedious, are designed to make it as easy as possible for the TAs to mark the assignments by letting them run your assignment, in some cases through automated tests. While these tests will never be used to determine your entire grade, they speed up the process significantly, which allows the TAs to provide better feedback and not waste time on administrative details. Plus, if the TA is in a good mood while he or she is grading, then that increases the chance of them giving out partial marks. :)

Up to 30% can be removed for bad indentation of your code as well as omitting comments, or poor coding structure.

To get full marks, you must:

- Follow all directions below
  - In particular, make sure that all classes and method names are **spelled and capitalized exactly** as described in this document. Otherwise, marks will be removed
- Make sure that your code compiles
  - Non-compiling code will receive a very low mark
- Write your name and student ID as a comment in all .java files you hand in
- Indent your code properly
- Name your variables appropriately
  - The purpose of each variable should be obvious from the name
- Comment your work
  - A comment every line is not needed, but there should be enough comments to fully understand your program

Part 1 (0 points): Warm-up

Do **NOT** *submit this part, as it will not be graded. However, doing these exercises might help you to do the second part of the assignment, which will be graded. If you have difficulties with the questions of Part 1, then we suggest that you consult the TAs during their office hours; they can help you and work with you through the warm-up questions. You are responsible for knowing all of the material in these questions.*

Warm-up Question 1 (0 points)

Write a method `reverseString` which takes as input a `String` and returns the string in reverse order. For example if the input `String` is "Comp 202 is awesome." the result should be ".emosewa si 202 pmoC"

Hint: Use a new `String` called `reverse` and initially store the empty `String` in it. Then read the input `String` in reverse by using the method .charAt(int i) to get a specific element.

Warm-up Question 2 (0 points)

Write a method `shiftString` which takes as input a `String s` an `int n`, and returns a new string obtained by shifting the characters in `s` by `n` positions to the right. For example: `shiftString(''banana'', 2)` returns "nabana", `shiftString(''banana'', 9)` returns "anaban", and `shiftString(''banana'', -1)` returns "ananab" (a negative number will produce a shift to left!).

Hint: Start by writing a method that shifts the characters of a string by a fixed number of positions (say 2). Then generalize the method by letting the number of positions be determined by an input `n` which is less than the length of the string. And finally, write a method that works for any integer `n`.

Warm-up Question 3 (0 points)

Create a method to print the outline of a square made up of * signs. This method must have one parameter, which is the length of the sides in number of *'s. This method should use only two *for loops*, and use *if statements* within the *for loops* to decide whether to draw a space or a star.

Draw the outline of a square as follows:

```
**********
*        *
*        *
*        *
*        *
*        *
*        *
*        *
*        *
**********
```

N.B. It is normal that the square does not appear to be a perfect square on screen as the width and the length of the characters are not equal.

Warm-up Question 4 (0 points)

Change the method you just created to have two parameters, so that you can create rectangles. The first parameter will be the height of the rectangle, and the second parameter will be the width of the rectangle.

Warm-up Question 5 (0 points)

Write a program to display the (x,y) coordinates up to (9,9) of the upper right quadrant of a Cartesian plane. As in the previous warm-up question, your solution should use two nested *for loops*. Your program should also display the axes, by checking to see if the x-coordinate is zero or if the y-coordinate is zero. Note that when both the x and y coordinates are zero, you should print a + character.

For example, the output of your code should look like:

```
^
|(1,9)(2,9)(3,9)(4,9)(5,9)(6,9)(7,9)(8,9)(9,9)
|(1,8)(2,8)(3,8)(4,8)(5,8)(6,8)(7,8)(8,8)(9,8)
|(1,7)(2,7)(3,7)(4,7)(5,7)(6,7)(7,7)(8,7)(9,7)
|(1,6)(2,6)(3,6)(4,6)(5,6)(6,6)(7,6)(8,6)(9,6)
|(1,5)(2,5)(3,5)(4,5)(5,5)(6,5)(7,5)(8,5)(9,5)
|(1,4)(2,4)(3,4)(4,4)(5,4)(6,4)(7,4)(8,4)(9,4)
|(1,3)(2,3)(3,3)(4,3)(5,3)(6,3)(7,3)(8,3)(9,3)
|(1,2)(2,2)(3,2)(4,2)(5,2)(6,2)(7,2)(8,2)(9,2)
|(1,1)(2,1)(3,1)(4,1)(5,1)(6,1)(7,1)(8,1)(9,1)
+----------------------------------------->
```

Note that in the above image, all of the coordinates containing 0's are not displayed, since we are printing axes instead.

Part 2

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

Question 1: Mountains Program (38 points)

For this question, you will have to design and write a Java program which displays a randomized mountain range using text art. **The end of this question shows two examples.**

The overall idea for creating the mountain range is to generate a one-dimensional array of `doubles` which represent the height of different points along the mountain range. Each value in the array depends on the previous one, using a parameter called *steepness* which controls how much the next point can shift up or down.

Once the array of points is created, then we can start the drawing process. This process is based on the last warm-up question where we use nested *for-loops*, so we strongly encourage you to complete that warm-up question before attempting this one. In the drawing process, we will loop through the mountain range horizontally and vertically and decide which symbols to print.

To complete these tasks, you must write the methods below. Note that you are free to write more methods if they help the design or readability of your code.

1a) Generating the next point in the range

Write a method `getNextPoint` that takes as input three `doubles` parameters. The first parameter is a `double` which represents the height of the previous point in the mountain range. The second parameter is a `double` which is the possible `steepness` between adjacent points. Finally, the last parameter is a `double`, which is the maximum height of any point in the mountain range.

This method takes the previous point in the mountain range, and creates a height for the new point. The difference between the previous height and the new point is based on a random change controlled by the `steepness` parameter.

For example, let the previous point be *10.5*, and the `steepness` parameter be *3.0*. The new point must be *10.5* plus a random number in the range of *-3.0* to *3.0*. Therefore, the new point will be in the range of *7.5* to *13.5*.

This means that the larger the `steepness` parameter is, the larger the difference in height between one point in the mountain range and the next. Note that in the two example mountains provided, the steepness of the first example is much higher than that of the second example, and the mountains are much steeper.

We also need to make sure that the height of the new point is not too large or too small. If the new point is above the maximum height (which is the third parameter to the method), then the new point's height must be set to be equal to the maximum height. Similarly, if the new point's height is below zero, then the new point's height must be set to zero.

To summarize, the new point is based on the old point, plus a change based on the steepness. The new point's height cannot be above the maximum height, or below zero.

1b) Method for generating the heights of all points

Write a method `generateMountains` to generate the heights for all points in the mountain range. This method takes three parameters. The first parameter is an `integer` value, which is the number of points for the mountain range. The second parameter is a `double` representing the `steepness`. Finally, the third parameter is a `double` which is the maximum height of the points in this mountain range. This method returns an array of `doubles`.

The method must create the array of `doubles`, using the first parameter to this method as the size. Then, set the first point in this array to have a value of half of the maximum height of the mountain range.

Then, iterate over the array and set each new point using the `getNextPoint` method. Note that this method must take the value of the previous point as a parameter.

1c) Method for finding the maximum height

Write a method `findMaxHeight`. This method takes the array of heights, and returns the largest height in that array. If the array is empty, return zero.

1d) Method to draw the mountain range

Create the `drawMountains` method. This method takes as the first parameter an array of `doubles`, which is the height of the points in the mountain range. The second parameter is a `String`, which represents the symbol to draw the mountains with.

First, we need to find the tallest point in the mountain range. Call the `findMaxHeight` method with the mountain range. Take the returned value, add one, and then cast it to an `integer`. We want this *maximum height* to be an integer such that we can use nested *for-loops* to draw a grid of symbols representing the mountains.

Second, use nested *for-loops* (like in the last warm-up question) to draw the mountains. The outer *for-loop* should loop from the maximum height, down to zero. The inner *for-loop* should loop from the beginning of the points array to the end.

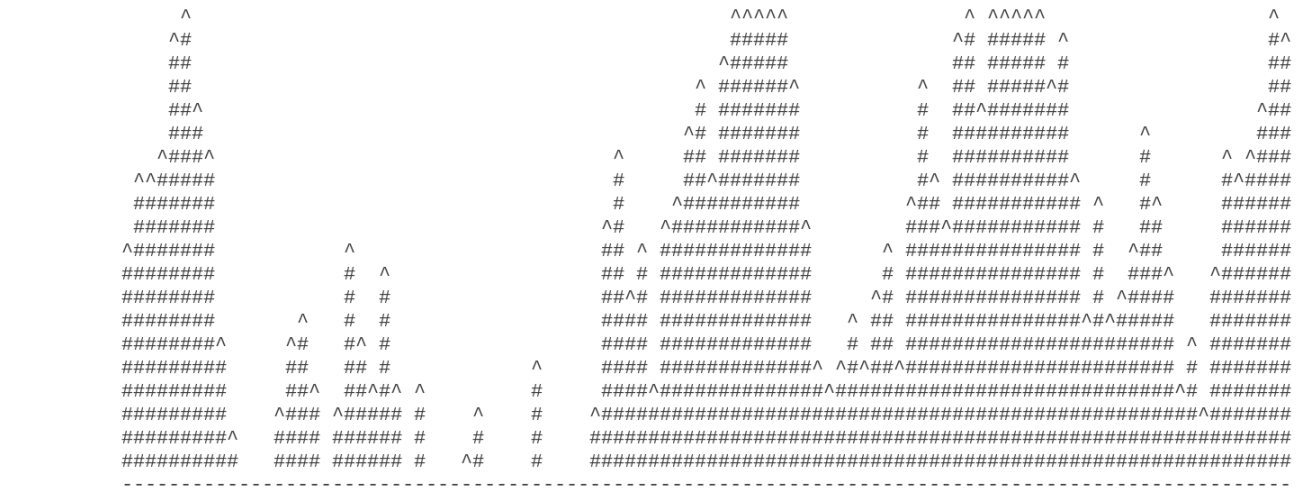
The iteration variables for the *for-loops* provide the coordinates to decide whether to draw sky, ground, or the mountain symbols. See the table for more details.

Let y be the vertical component of the grid you are drawing, and let *height* be the height of the current point.

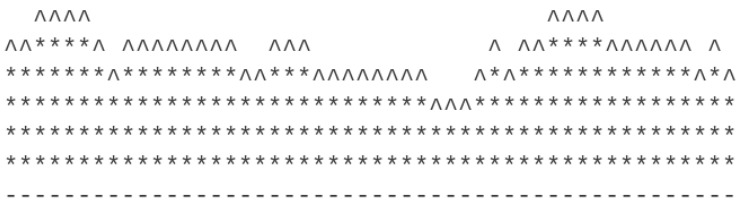
Condition	**Symbol to Draw**
y > height	The space character
y - height < 1 and y - height > -1	The mountaintop ^
y < height	The mountain symbol
y equal to zero	The ground -

Note that the mountain symbol is the second parameter to the `drawMountain` method, so that mountain ranges can be drawn with different symbols.

Examples This is an example of creating a rocky mountain range. This example shows a call to `generateMountains` with *size* 100, *maxHeight* 20, and *steepness* 8, and a call to the `drawMountains` method with that mountain range and the mountain symbol #.



This is an example of creating a gentle mountain range. This example shows a call to `generateMountains` with *size* 50, *maxHeight* 10, and *steepness* 0.8, and a call to the `drawMountains` method with that mountain range and the mountain symbol *.



Note that due to the use of random numbers, your mountain ranges will not look exactly like these.

Question 2: Phrases Program (62 points)

The second program for this assignment focuses on manipulating `Strings` and characters. The objective of this program is to combine `Strings` which have a common word.

Examples

```
String s1 = "Don't burn a bridge until it's behind you.";
String s2 = "Do not worry about crossing the bridge until you're ready to cross it.";
String combined1 = combineStrings(s1, s2);
System.out.println(combined1);
//Don't burn a bridge until you're ready to cross it.

String t1 = "Not all those who wander are lost.";
String t2 = "Make new friends, but keep the old; The first are silver, the rest are gold.";
String combined2 = combineStrings(t1, t2);
System.out.println(combined2);
//Not all those who wander are silver, the rest are gold.
```

Note that in the first example the common word is “bridge”, while in the second example the common word is “are”.

2a) Method to turn a String array into a String

Write a method `join` which takes as a parameter one `String` array and returns a `String`. This `String` must contain the words in the `String` array, separated by spaces.

For example, if the `String` array parameter is { ``I'', ``like'', ``green'', ``apples!''} then the returned `String` must be "I like green apples!".

This method will help you print out the `String` arrays within this question for debugging.

2b) Find in String array method

Write a method `findInStringArray` which takes one `String` parameter and one `String` array parameter. This method will search the `String` array for the `String`, and return the position of the first `String` found. Ignore the capitalization when searching for a result

If the `String` cannot be found in the array, return -1.

For example, searching the `String` array { ``Hello'', ``hello?'', ``THIS'', ``this'', ``pancakes''} for the `String` "this" must return 2.

2c) Tokenize method

The methods in this question operate on words within a `String`. Therefore, we require a method to divide a `String` into its component words.

Create a method `tokenize` that takes one `String` parameter and returns a `String` array composed of the words in the `String`. We will assume that words in our `String` are separated by one space only, and that the sentence cannot end or begin with a space.

For example, if the `tokenize` method is called with the `String` "That's one small step for a man, one giant leap for mankind."[1] then the returned `String[]` array must contain 12 `Strings`:
{``That's'', ``one'', ``small'', ``step'', ``for'', ``a'', ``man,'', ``one'', ``giant'', ``leap'', ``for'', ``mankind.''}

The first step is to create a `String[]` array of the correct size. One suggestion is to write a helper method which counts the number of words in the `String`.

Then, add the characters in the `String` into the individual `Strings` in the `String` array. For example, the characters 'T' 'h' 'a' 't' ' ' 's' should be placed in the first `String` in the array, the characters 'o' 'n' 'e' should be placed in the second `String`, and so on. You know to begin placing characters in the next `String` in the array when you reach the space character.

Note that the `split` method cannot be used for this question. However, it may be helpful to test your results against the results of the `split` method.

2d) Method to combine String arrays

Write a method `combineArray` which has four parameters. The first two parameters are `String` arrays. The last two parameters are `integers`, which represent the positions where we will combine the `String` arrays. This method will return a `String` array.

This method will create a new `String` array to hold `Strings` from the first and second `String` array. The new array must contain the `Strings` from the first `String` array, from the first position up to (and including) the index specified by the first `integer` parameter. As well, the new array must contain the

[1] Note that an "a" has been added to make the quote complete.

`Strings` from the second `String` array from the second `integer` parameter (but not including) until the end of the second `String` array.

For example, let the first `String` array be { ``Hello,'', ``how'', ``are'', ``you'', ``doing?''} and the first `integer` parameter be 2. As well, let the second `String` array be
{ ``I'', ``see'', ``two'', ``giant'', ``rhinos!!''} and the second `integer` parameter be 1.

Then, the returned `String` array must contain
{ ``Hello,'', ``how'', ``are'', ``two'', ``giant'', ``rhinos!!''}.

2e) Method to combine Strings

Finally, create a method `combineStrings`. This method will take two `Strings`, which are the phrases to combine. This method must return a `String`, which is the phrase created from the combination of the two input `Strings`.

The first step is to use the `tokenize` method to split the `Strings` into arrays.

Then, for each word in the first `String` array, search for that word in the second `String` array.

If no word in the first `String` array can be found in the second `String` array, then the method must `throw` an `IllegalArgumentException` with an informative error message. Note that you must not catch this Exception in your program.

Otherwise, you will know there exists a common word between the two `Strings`. As well, you will know the position of the word in the first and second `String` arrays. Then, use the `combineArray` method with the two `String` arrays and the positions to obtain a combined `String` array containing the combined phrase.

Finally, return the combined phrase as a `String` using the `join` method.

What To Submit

Please put all your files in a folder called Assignment3. Zip the folder (DO NOT RAR it) and submit it in MyCourses. If you do not know how to zip files, please ask any search engine or friends. Google will be your best friend with this, and a lot of different little problems as well.

Instead your zipped folder, there must be the following files. **Do not submit any other files, especially .class files.** Any deviation from these requirements may lead to lost marks.

`Mountains.java`
`Phrases.java`
`Confession.txt` (optional) In this file, you can tell the TA about any issues you ran into doing this assignment. If you point out an error that you know occurs in your problem, it may lead the TA to give you more partial credit.

Question 1

Question 1a:	8	points
Question 1b:	8	points
Question 1c:	4	points
Question 1d:	18	points
Question 1 Total:	38	points

Question 2

Question 2a:	8	points
Question 2b:	8	points
Question 2c:	15	points
Question 2d:	15	points
Question 2e:	16	points
Question 2 Total:	62	points
	100	**points**