ASSIGNMENT 3

COMP-202, Winter 2017, All Sections

Due: Friday, February 24^{th}, 11:59pm

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

Question 1:	50 points
Question 2:	50 points
	100 points total

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, coding structure, or missing files. Marks will be removed as well if the class and method names are not respected. Make sure that you match the capitalisation of method and class names.

To get full marks, you must:

- Follow all directions below
- Make sure that your code compiles
  - Non-compiling code will receive a very low mark
- Write your name and student name is written 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 into it the empty `String`. 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 shift 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)

Write a method `firstNPrimes` that takes as input and integer `n` and returns an array of integers containing the first `n` prime numbers. You should use the method seen in class to check whether a number is prime of not.

Warm-up Question 4 (0 points)

This program prints the outline of a square made up of * signs. It should take as input the length of the sides in number of *'s. This program 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.

How do you generalize the program in order to print a rectangle where both the base and the height are given as input?

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: Draw a Circle (50 points)

Before starting this question, we strongly recommend that you complete the last two warm-up exercises if you have not already done so. These warm-up questions will be the best way to start the assignment.

For this question, you will write a program called `Circle` that displays a circle in the upper right quadrant of the Cartesian plane.

To do so, write (at a minimum) the following three methods:

A. A method called `onCircle` that takes as input five ints. The first three parameters represent (in order) the radius and x,y coordinates of the centre of the circle. The last two parameters represent an x,y position on the grid.

   This method determines whether or not the circle should be drawn at the grid coordinates given by the method's parameters. To get the appropriate thickness for the circle, return whether the following formula holds or not:

$$radius^2 \leq (x-a)^2 + (y-b)^2 \leq radius^2 + 1$$

   In the above formula, a and b are the coordinates of the centre of the circle, and x and y are the grid coordinates.

B. A method called `verifyInput` that takes as input three ints representing the radius and centre coordinates of the circle, and returns nothing. This method should throw an `IllegalArgumentException` and display a helpful error message if either the circle does not fit in the upper right quadrant, or if the radius is non-positive. Otherwise, the method will not print anything. As a hint, the circle does not fit in the upper-right quadrant if it is too close to the axes.

C. A method called `drawCircle` that takes as input an int representing the radius of the circle, another int representing the x coordinate of the centre of the circle, and a third int representing the y

coordinate of the circle's centre. It also takes as input a char representing the symbol with which the circle will be drawn. Both of the methods described above must be called and used appropriately in this method in order to get full points.

The rules for displaying the axes (the x-axis and the y-axis) are as follows: the minimum length and height of each axis is 9. If the circle fits in the 9 by 9 quadrant, there should be 9 vertical bars (|) representing the y-axis, and 9 horizontal dashes (-) representing the x-axis. You should use the plus symbol (+) to represent the origin. The y-axis should end in a hat symbol (∧), and the x-axis should end in a right triangle bracket (>). This means that there are a total of 11 characters on the x-axis, and 11 on the y-axis, with the origin counting as being on both.

If the circle does not fit, the appropriate axis (or axes) should be extended. They still must end in (∧) and (>), and the origin remains the same. Sample output for various input values is shown below. Note that, due to rounding errors and display dimensions of characters, the circle will not look like a perfect circle. This is expected behaviour.

If more than one character should be drawn at the same location (eg, if the circle intersects with an axis), the circle gets priority. This is slightly different from what happened in the second warm-up exercise. See the second example below for sample output in this scenario.

As suggested above, this program will be very similar to the last two warm-up questions. Please complete those if you are having trouble understanding this question.

Your main method *will not* be graded. However, we strongly advise that you write a main method in order to test your code. You may include it in your submission as long as it compiles.

```
drawCircle(1,3,3,'&')
^
|
|
|
|
|
| &&&
| & &
| &&&
|
+--------->

drawCircle(3,3,3,'&')
^
|
|
|
| &&&
|
&     &
&     &
&     &
|
+-&&&----->
```

```
drawCircle(4,10,5,'&')
^
|         &&&
|
|
|      &        &
|      &        &
|      &        &
|
|
|         &&&
+--------------->

drawCircle(5,10,12,'&')
^
|         &&&
|       &     &
|      &        &
|
|    &           &
|    &           &
|    &           &
|
|      &        &
|       &     &
|         &&&
|
|
|
|
|
|
+---------------->


drawCircle(5,2,2,'&')
java.lang.IllegalArgumentException: Circle must fit in upper right quadrant

drawCircle(-5,7,12,'&')
java.lang.IllegalArgumentException: Circle must have positive radius
```

Question 2: Cipher (50 points)

Caesar's cipher is a very well known and simple encryption scheme. The point of an encryption scheme is to transform a message so that only those authorized will be able to read it. Caesar's cipher conceals a message by replacing each letter in the original message (the *plaintext*), by a letter corresponding to a certain number of letters to the right on the alphabet. Of course, the message can be retrieved by replacing each letter in the encoded message (the *ciphertext*) with the letter corresponding to the same number of position to the left on the alphabet. To achieve this, the cipher has a key that needs to be kept private. Only those with the key can encode and decode a message. Such a key determines the shift that needs to be performed on each letter. For example, here is how a string containing the entire alphabet will be encrypted using a key equal to 3:

Original:	abcdefghijklmnopqrstuvwxyz
Encrypted:	defghijklmnopqrstuvwxyzabc

Vigenère's cipher is a slightly more complex encryption scheme, also used to transform a message. The

key of this cipher consists of a word and the cipher works by applying multiple Caesar ciphers based on the letters of the keyword. Each letter can be associated with a number corresponding to its position in the English alphabet (counting from 0). For instance, the letter 'a' is associated to 0, 'c' to 2, and 'z' to 25. Therefore, the keyword of the cipher will provide as many integers as letters in the word and these integers will be used to implement different Caesar ciphers.

Let's see how: suppose the message to encrypt is "elephants" and the keyword is "rats". The first thing to do is to repeat the keyword until its length matches the one of the message.

Message:	e	l	e	p	h	a	n	t	s
Keyword:	r	a	t	s	r	a	t	s	r

Now, each letter of "ratsratsr" is associated to both a letter in the message and an integer. We can encrypt each letter of the message using a Caesar cipher where the key corresponds to the integer associated to it through the keyword. In this case 'r' corresponds to 17, so the first letter of the message which is an 'e' will be encrypted using a 'v', the second letter 'l' as an 'l' since 'a' is associated to 0, and so on. The entire message will be encrypted as "vlxhyaglj".

The goal of this exercise is to write several methods in order to create a program that encodes and decodes messages using Caesar's and Vigenère's ciphers. **For the purpose of this exercise we will only consider messages written using lower case letters and blank spaces.** All the code for this question must be placed in a file named `Cipher.java`.

2a. Encoding a character

Let's start by writing a simple method called `charRightShift` which takes a character and an integer `n` as inputs, and returns a character. The method should verify that the integer is a number between 0 and 25 (both included). If that's not the case, the method should print out an error message and return the character with ASCII value 0. Note that ASCII value 0 is not '0', but is the char that maps to the value 0! Otherwise, if the character received as input is a lower case letter of the English alphabet, the method will return the letter of the alphabet which is `n` positions *to the right* on the alphabet. If the character received as input is not a lower case letter of the English alphabet, then the method returns the character itself with no modification.

For example:

- `charRightShift('g', 2 )` returns `'i'`,
- `charRightShift('#', 2 )` returns `'#'`, and
- `charRightShift('h', 32 )` returns the character with ASCII 0 and prints an error message.

2b. Decoding a character

Write a method `charLeftShift` which practically reverses what the previous method does. This method also takes a character and an integer `n` as inputs, and returns a character. The method should verify that the integer is a number between 0 and 25 (both included). If that's not the case it should print out an error message and return the character with ASCII value 0. Note that ASCII value 0 is not '0', but is the char that maps to the value 0! Otherwise, if the character received as input is a lower case letter of the English alphabet, the method will return the letter of the alphabet which is `n` positions *to the left* on the alphabet. If the character received as input is not a lower case letter of the English alphabet, then the method returns the character itself with no modification.

For example:

- `charLeftShift('i', 2 )` returns `'g'`,
- `charLeftShift('#', 2 )` returns `'#'`, and

- `charLeftShift('h', 32 )` returns the character with ASCII 0 and prints an error message.

Note: The two methods above are very similar. This suggests that you write one common method `charShift` which contains the shifting logic and can shift both left and right. Then `charRightShift` can simply call `charShift` with a positive `n`, and `charLeftShift` can call `charShift` with a negative version of `n`.

2c. Caesar's cipher - Encoding

Write a method `caesarEncode` that takes a `String message` and an `int key` as inputs and returns the string obtained by encrypting `message` using the Caesar's cipher with key equal to `key`. To create the encrypted string you need to replace each letter in `message`, by the letter corresponding to `key` letters to the *right* on the alphabet. You should call and use `charRightShift` appropriately in order to get full points.

The input `key` must be an integer from 0 to 25 (included). Your method should print out an error message and return the empty string if that's not the case.

For the purpose of this exercise you can assume that the strings to encrypt will only contain letters from the English alphabet in lower case and blank spaces. Blank spaces don't get modified by the encryption.

For example, `caesarEncode(''cats and dogs'', 5)` should return `''hfyx fsi itlx''`.

2d. Caesar's cipher - Decoding

Write a method `caesarDecode` that takes a `String message` and an `int key` as inputs and retunrs the string obtained by decrypting `message` using the Caesar's cipher with key equal to `key`. To decrypt the string you need to replace each letter in `message`, by the letter corresponding to `key` letters to the *left* on the alphabet. To get full points, you should call and use the method `charLeftShift` appropriately.

As for `caesarEncode`, the key must be a number between 0 and 25. Your method should print an error message and return an empty string if that's not the case. More over, you can expect strings to contain only lower case letters from the English alphabet and blank spaces which will not be modified by the decryption (as they were not modified by the encryption).

For example, `caesarDecode(''hfyx fsi itlx'', 5)` should return `''cats and dogs''`.

2e. From String to keys

Write a method called `obtainKeys` which takes a String as input and returns an array of integers. The size of the array will be equal to the length of the String. The elements of the array correspond to the position (counting from 0) of each character in the String as a letter of the English alphabet. For instance `obtainKeys(''hello'')` returns `[7, 4, 11, 11, 14]`.

For the purpose of this exercise you can assume that the input String to this method will only contain lower case letters of the English alphabet.

2f. Vigenère's cipher - Encoding

Write a method `vigenereEncode` that takes a `String message` and a `String keyword` as inputs and returns the string obtained by encrypting `message` using the Vigenère's cipher with key equal to `keyword`. Remember that this cipher first associates each letter of the keyword to a letter of the message. Then it shifts (*to the right*) each letter of the message by the number of positions determined by the corresponding letter in the keyword. Use the methods `obtainKeys` and `charRightShift` appropriately in order to implement the encryption.

The input `keyword` must contain only characters from the lower case English alphabet. Your method should print out an error message and return the empty string if that's not the case.

For the purpose of this exercise you can assume that the strings to encrypt will only contain letters from the English alphabet in lower case and blank spaces. Blank spaces don't get modified by the encryption.

For example, `vigenereEncode(``elephants and hippos'', ``rats'')` should return `` ``vlxhyaglj tfu aagphk'' ``.

2g. Vigenère's cipher - Decoding

Finally, write a method `vigenereDecode` that takes a `String message` and a `String keyword` as inputs and returns the string obtained by decrypting the `message` using the Vigenère's cipher with key equal to `keyword`. Remember that this cipher first associates each letter of the keyword to a letter of the message. Then it shifts (*to the left*) each letter of the message by the number of positions determined by the corresponding letter in the keyword. Use the methods `obtainKeys` and `charLeftShift` appropriately in order to implement the decryption.

Again, the input `keyword` must contain only characters from the lower case English alphabet. Your method should print out an error message and return the empty string if that's not the case.

For the purpose of this exercise you can assume that the strings to decrypt will only contain letters from the English alphabet in lower case and blank spaces.

For example, `vigenereDecode(``vlxhyaglj tfu aagphk'', ``rats'')` should return `` ``elephants and hippos'' ``.

What To Submit

You have to submit one zip file that contains all your files to myCourses - Assignment 3. If you do not know how to zip files, please enquire that information from any search engine or friends. Google might be your best friend with this, and for a lot of different little problems as well.

These files should all be inside your zip. Do not submit any other files, especially .class files.

`Circle.java`
`Cipher.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. On the other hand, it also may lead the TA to notice something that otherwise they would not.