**Comp-122 Assignment 2**

**Andrew Lawler**

Introduction

For this assignment I was tasked with creating a cipher text encryption and decryption program. For the first part of the task I was required to create an interface and then implement the interface In part B. For part b, I had to inplement the methods in a class called Caesar. Within this class I had the methods (rotate, frequencies, chiSquared and decipher). I implemented these methods using my knowledge of cipher text and the suitable examples shown to us.

Next Parts

**c.** Testing input for Rotate.java

I implemented a system that will print wrong input length when there is not two inputs. It then tests the inputs to make sure they are in the correct format.

|  |  |  |  |
| --- | --- | --- | --- |
| Input | Expected Output | Actual Output | Comments |
|  | Too few parameters!  Usage: java Rotate n ‘cipher text’ | Too few parameters!  Usage: java Rotate n ‘cipher text’ | This shows that the program ends with an error message when there is nothing entered. |
| . | Too few parameters!  Usage: java Rotate n ‘cipher text’ | Too few parameters!  Usage: java Rotate n ‘cipher text’ | This result tells us that the program will revoke all types of input when there is not the correct amount of inputs. |
| 3 | Too few parameters!  Usage: java Rotate n ‘cipher text’ | Too few parameters!  Usage: java Rotate n ‘cipher text’ | This example shows the error message once again. |
| Andy | Too few parameters!  Usage: java Rotate n ‘cipher text’ | Too few parameters!  Usage: java Rotate n ‘cipher text’ | This shows that the program handles the one string input and provides the same error message. |
| Andy 3 | No integer first, no string second, wrong input | You did not enter an integer first!  You did not enter a string second!  Wrong inputs! | The output here shows that order matters. It tested the first input and found it was not an integer and also found the second input was not a string. These are the wrong inputs. |
| 3 Andy | dqgb | dqgb | Here you can see if we enter in the correct format it will work. |

**d.** Testing input for BreakCaesar.java

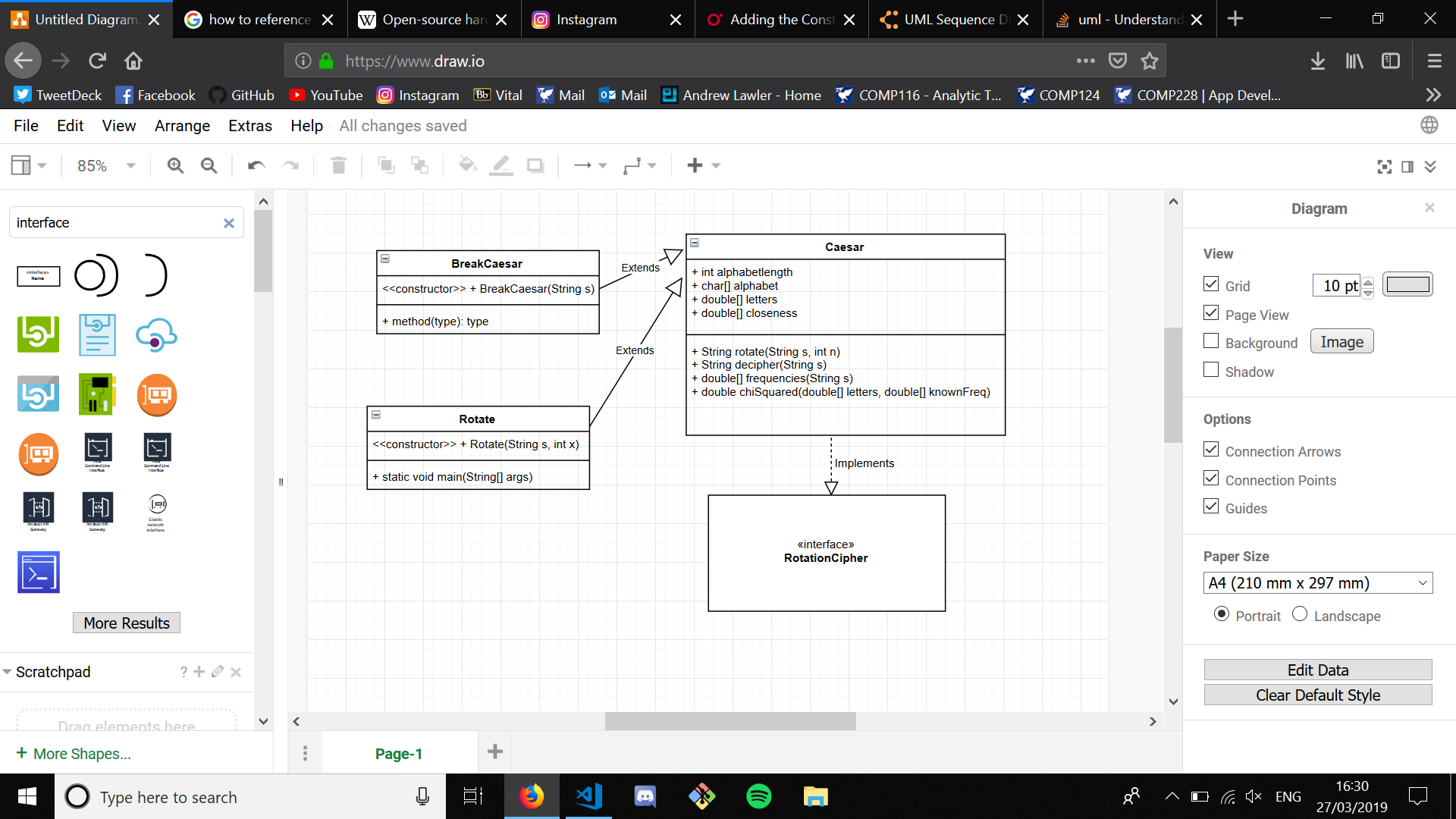
I implemented the same system as for part c but with one input this time. It will check if the input size is equal to one and will then check if that input is a string. If either case is false we will get a specific error message.

|  |  |  |  |
| --- | --- | --- | --- |
| Input | Expected Output | Actual Output | Comments |
|  | Too few parameters!  Usage: java BreakCaesar ‘cipher text’ | Too few parameters!  Usage: java BreakCaesar ‘cipher text’ | This shows that the program requires inputs and will not let you enter nothing. |
| 3 | You didn’t enter a string | You did not enter a string! | This shows that you need to enter a string because the input is tested. |
| . | . | . | This was to test that the function did in fact let this in. A dot is allowed as we may form sentences to decrypt. |
| xf bsf ifsf | we are here | we are here | This was to test that the program would work. |

**e.** UML Diagram

I first initialized the interface for the assignment. This interface is called RotationCipher and its job is to simply state the methods for the subclass to implement. Interfaces state methods as public but it is known that they are in fact abstract also. Once I had implemented my interface, I moved onto my Caesar subclass. This subclass implements the interface itself. I wrote the code for the methods and made sure they followed the same input parameters and same name as the ones in the interface, it in turn overrides them.

I then was tasked with implementing the BreakCaesar and Rotate applications. For this task I extended the Caesar class further, once in the subclass I created a constructor which takes the string input and sends it to the decipher method from the parent class. This is the way my application works. I followed the same layout for Rotate. I extended the subclass and created a constructor which took the inputted argument and send it to the rotate method which in turn gets printed out.

I think the way I designed my classes is good because they all link together well and it keeps the code neat and efficient.

**f.** JavaDocs

**g.**

If the language we are examining is not English we would need to modify some things. I made this process easy by adding global variables at the top of the Caesar program. You would need to edit the alphabet string with the new languages alphabet and then edit the alphabet length for the new alphabet length. The only other change you then need is for the frequencies of the letters. You would need to edit the array knownFreq to add your new languages frequencies. I decided to implement this approach as it makes the entire process really easy.

**h.**

The way I have written my program would allow me to calculate a shift for lower case and upper case. All I would do is I would have to use my rotation method and add a counter to show the positions of the lower-case letters and the positions of the upper case letters. I would then need to test all possible rotations of these amounts using my decipher method to then find the compatible rotation to decipher the text. This process would obviously take longer than just a loop of 26 as we would have to test every possible rotation against every possible rotation. I’d be looking at two embedded for loops to do this. On-top of this edit, I would have to edit my rotation method to take a second input as the one input n will not be able to apply two different rotations.