# Table of Contents

[Table of Contents 1](#_Toc536111145)

[Introduction 1](#_Toc536111146)

[Scenario 2](#_Toc536111147)

[Scope 3](#_Toc536111148)

[Non-Functional Requirements 3](#_Toc536111149)

[Class Diagram 3](#_Toc536111150)

[Classes 3](#_Toc536111151)

[CipherMachine.java 3](#_Toc536111152)

[Methods 4](#_Toc536111153)

[ElonEncoder 4](#_Toc536111154)

[ElonDecoder 5](#_Toc536111155)

[Testing 6](#_Toc536111156)

[Review 7](#_Toc536111157)

[Development 7](#_Toc536111158)

[QA Testing 7](#_Toc536111159)

[Review 8](#_Toc536111160)

[Requirements 8](#_Toc536111161)

[Part 1 8](#_Toc536111162)

[Part 2 9](#_Toc536111163)

[Part 3 10](#_Toc536111164)

[Part 4 11](#_Toc536111165)

# Introduction

This is a report of my assignment from Infor; “CipherMachine”.

CipherMachine is a Console application written in Java using Eclipse Oxygen. I was tasked to produce a software application that meets the requirements below while following Infor’s professional coding standards. The finished solution contains the application classes and methods.

# Scenario

*Please provide an implementation of CipherMachine.java according to the description below. Where requirements may be vague - please provide a note about the issue and decisions taken.*

*Solutions should make use of standard platform features - no 3rd party libraries.*

*Build a simple substitution 'cipher machine'.*

*This algorithm will take a String message and 'encode' it such that it is no longer human readable. The approach will be to move the standard alphabet by a 'shift' value, such that a shift of 3 would mean A becomes D, D becomes G. The alphabetic shift should rotate the overhanging values to the beginning, such that Z becomes C. Casing should be preserved, but the character shift is the same, e.g. Z becomes C, z becomes c.*

*Upgrade the 'cipher machine' to include the following fixed substitutions:*

* *‘:’ should output ‘!’*
* *‘/’ should output ‘-‘*
* *‘?’ should output ‘+’*
* *‘#’ should output ‘,’*
* *‘.’ should output ‘]’*

*Whitespace characters should be preserved in the output*

*Any unrecognized characters in the input message should output ‘\_’*

*Write the counterpart method 'decode' which takes an encoded String message, and a 'shift' value, and then returns the original human readable message.*

e.g. KhoorZruog should output an encoded message of: HelloWorld

*Any encoded \_ characters should not be not modified.*

*Any unrecognized characters should be output as* \*

*Decode the following message*

Ugfyjslmdslagfk ]]] Lwdd fgtgvq lzw hskkogjv ak! ,Uzwvvsj,

# Scope

## In Scope

### Part 1

* Take a String message and 'encode' it such that it is no longer human readable. The approach will be to move the standard alphabet by a **'shift'** value.
* Move the standard alphabet by a **'shift'** value, such that a shift of 3 would mean **A** becomes **D**, **D** becomes **G**.
* The alphabetic shift should rotate the **overhanging** values to the beginning, such that **Z** becomes **C**.
* Casing should be **preserved**, but the character shift is the same, e.g. **Z** becomes **C**, **z** becomes **c**.

### Part 2

* Upgrade the 'cipher machine' to **include** the following fixed substitutions:
  + **:** should output **!**
  + **/** should output **-**
  + **?** should output **+**
  + **#** should output **,**
  + **.** should output **]**
* Whitespace characters should be preserved in the output
* Any unrecognized characters in the input message should output **\_**

### Part 3

* Write the counterpart method **'decode'** which takes an **encoded** String message, and a **'shift'** value, and then returns the **original human readable message**.
* Any **unrecognized** characters in the input message should **output \_**
* Whitespace characters should be **preserved** in the output

### Part 4

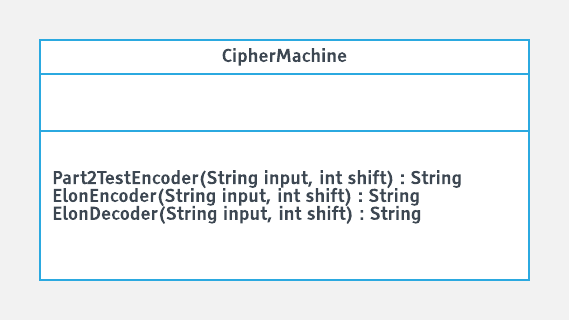
* **Decode** the following message:

Ugfyjslmdslagfk ]]] Lwdd fgtgvq lzw hskkogjv ak! ,Uzwvvsj,

## Non-Functional Requirements

Must be a Console application written in Java

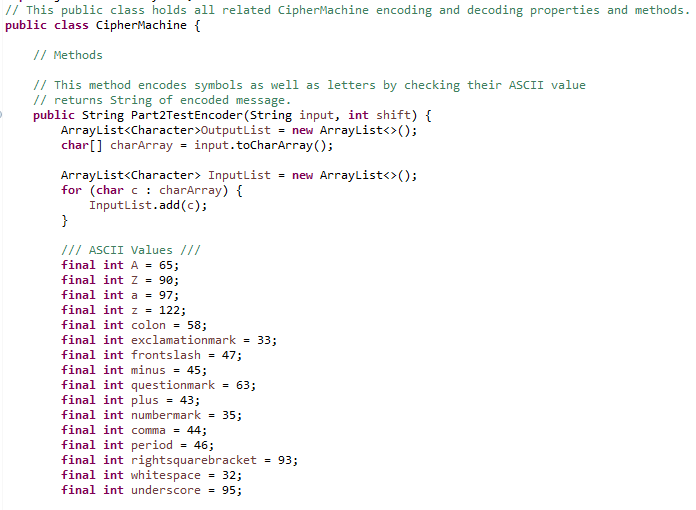
# Class Diagram



# Classes

## CipherMachine.java

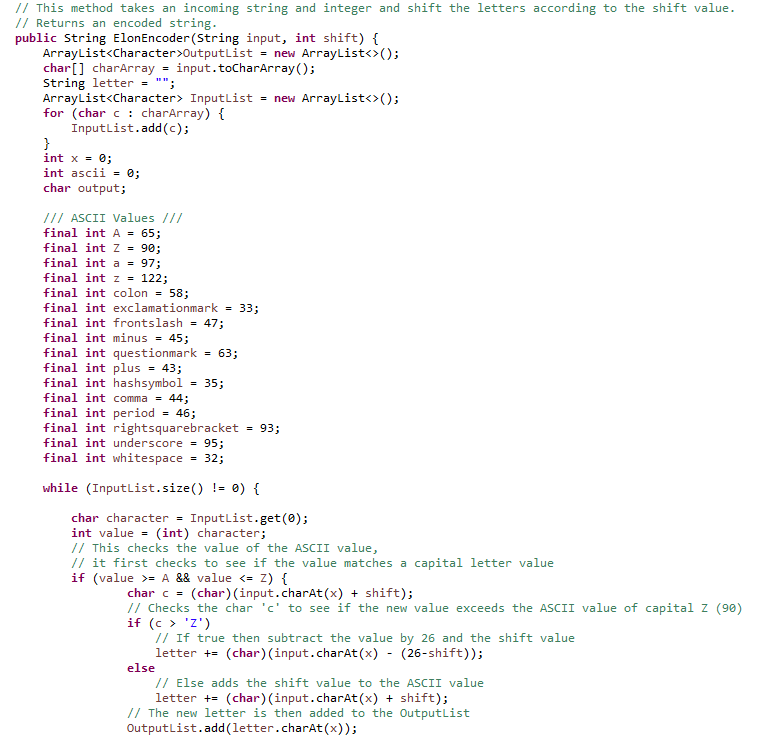
This class holds 3 different methods, however two of them are very similar. ElonEncoder and ElonDecoder are similar in that they either decode or encode a string. The third method is used to encode and decode symbol inputs as they act different.



# Methods

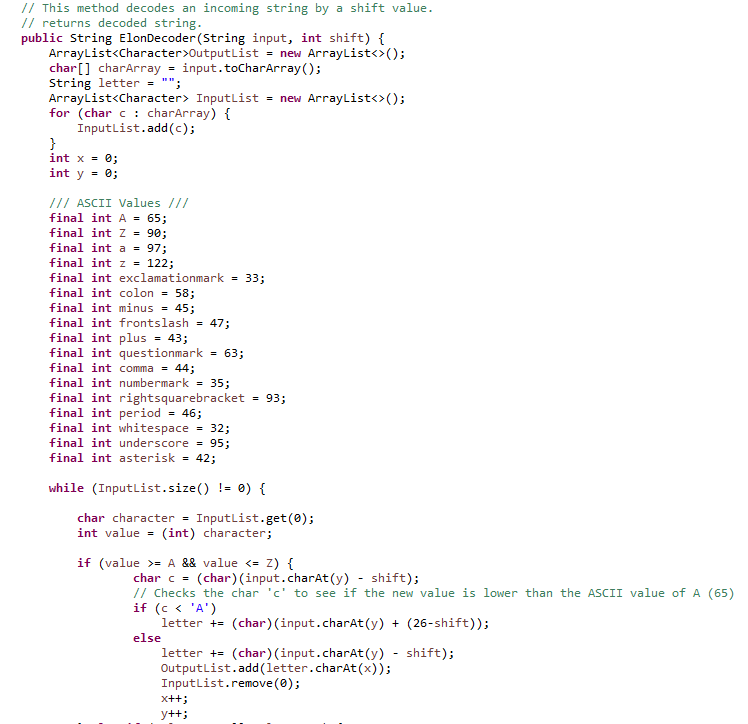
## ElonEncoder

The 'ElonEncoder' method takes a string and encodes it. It does this by moving the ASCII value of the letters by a shift value. It also changes certain symbols to other symbols as asked from the assignment, it does this by checking for certain ASCII values.



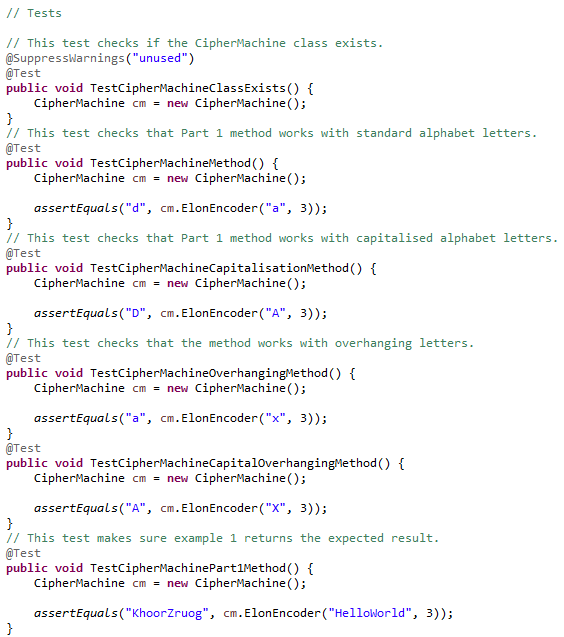
## ElonDecoder

This method is basically the inverse of ElonEncoder with some differences by decoding a string value. It does this by subtracting the ASCII values by the shift value. The symbols are also reversed too.



# Testing

This class is used to hold all the unit testing for the application. Some tests are simple in that they check if class exists or properties are functioning correctly, while others check if the correct encoded/decoded message is being returned or overhanging values are being returned as English characters. All tests passed successfully.



# Review

## Development

During development, I learnt how to manipulate ASCII values. I also extended my knowledge of conditional flow of control coding constructs such as nested if’s and another alternative switch constructs. As the finished program only uses two methods, I learned the importance of keeping classes short and readable, making sure to find elegant and succinct solutions to complicated problems.

## Documentation

I made sure to keep everything documented as I developed the application, I felt this was the best approach since I had just coded the method and had it fresh on my mind. I didn’t want to go back to the methods to then document them all.

## Debugging

## QA Testing

For testing, I used JUnit for unit testing during development. This allowed me to test my methods against the requirements.

## Review

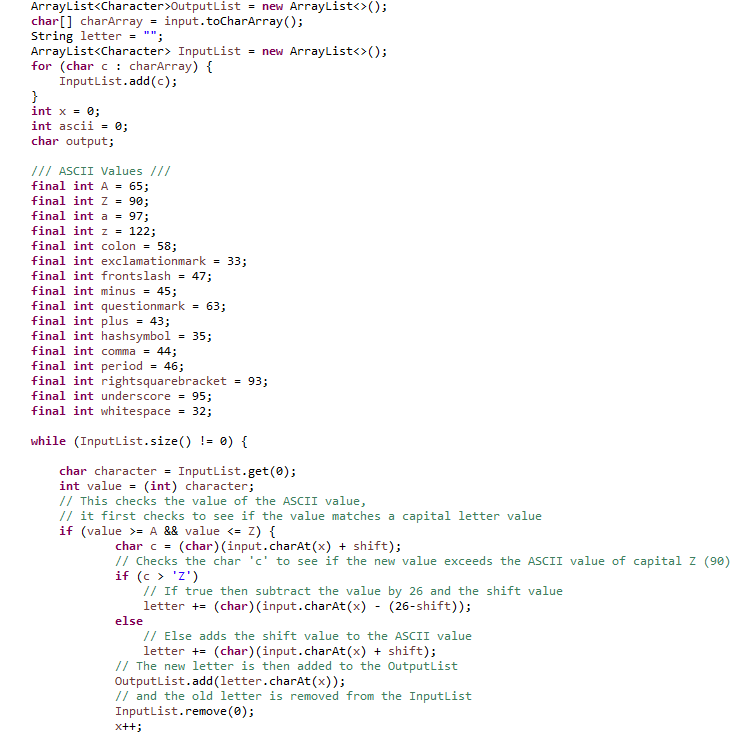
This was the first time I worked on a Java application at a more complex level, I had a Java mentor review my finished solution as well as teach me about unit testing during development.

## Requirements

### Part 1

The first part of the assignment asked me to take a string message and encode it so that it was no longer human readable. The values must be shifted by a value. For example, with a shift value of 3, 'A' would become 'D'.

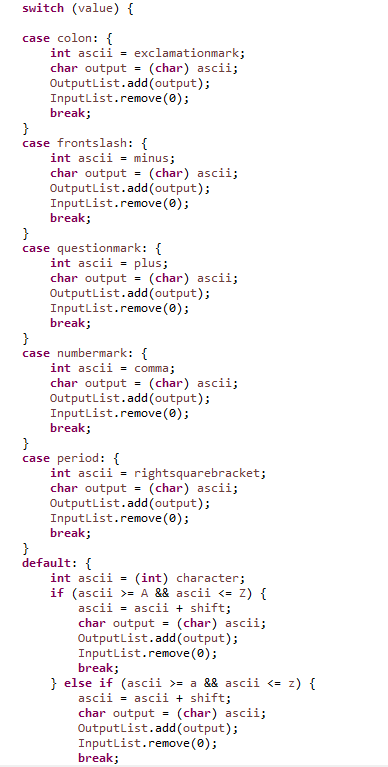
This first part saw many changes and revisions, but I've been able to make a method that does this correctly.



The 'ElonEncoder' method works by breaking apart the incoming string to a character list ("Leon" would become "L, E, O, N") this list is called the 'InputList'. While this list has contents inside, it grabs the first character from the list and declares a new integer of the ASCII value of that first letter. It then adds the shift value to the ASCII value, then it gets converted to a letter which is added to an 'OutputList', the item is also removed from the 'InputList'.

### Part 2

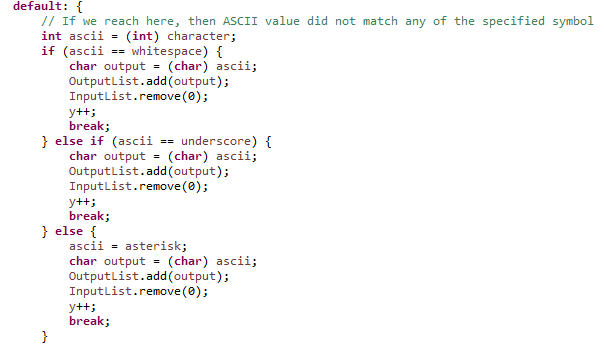
Part 2 asked for a conversion of symbols to different ones, whitespaces should not be changed, and any unrecognized symbols should return '\_'



The way I did this was by adding a case statement, I declared integers that matched the ASCII values of the characters and assigned them to this case statement so that if a symbol matched the same value, it would return another value and add it to the 'OutputList'.

### Part 3

Part 3 was like the first part, but instead of encoding a string message, now it's being decoded, instead of adding the values of the ASCII and the shift together, now it was being subtracted. I duplicated my method and reversed it, this would result in the 'ElonDecoder' method. While similar, part 3 also asked that '\_' characters should be left alone, so the case statement is a little different in that if it matches an '\_' ASCII values, it doesn't change anything.



### Part 4

Part 4 asked that a certain be decoded using the method I made in part 3. The string contained letters and symbols so if the program could decode it all successfully, then it was working as intended.

Since the shift value could have been anything, I created a loop in 'Main' class that would run the method 26 times. This proved unsuccessfully many times which told me my method wasn't working. After many revisions, I was able to find out what the encoded message said.

