Cryptography

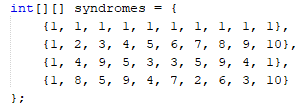
The first task introduced the use of Jframes, this was the first time I had used one and it made it very easy to debug code due to having a visual aid and easy action function implementations (mouse click).

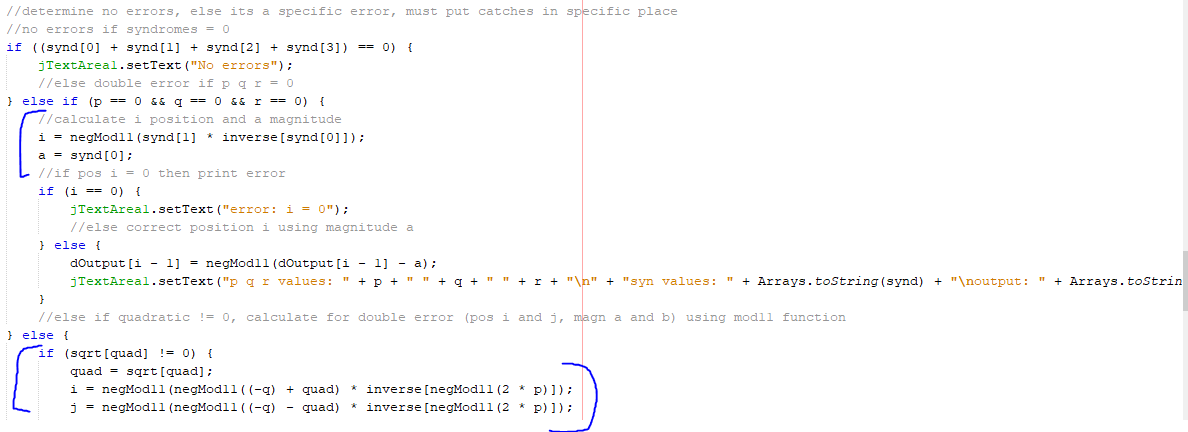
The first task required a program that took in a string representing a credit card number, and checking it with a checksum. This was an extremely small and fast program but was a good setup for future tasks, even for other modules like ESD as it introduced “IntegerParsing” and “String.valueof” "which has been needed a lot since.

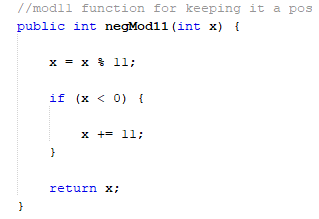
This task followed the conventional credit card checking rules and was implemented in the code.

The second task asked for a BCH generator and correction that identifies unusable numbers. This algorithm was a step up from the last, introducing the need to calculate parity bits for validation. This task mainly helped me figure out validation and implementing formulas for parity bit calculation. I learned how to fully create different functions and call them with set parameters, I was able to get the pseudocode formals from the powerpoint sheets and implement them. The task required BCH validation using syndromes and algorithms given to us in lectures. This meant extra functions needed to be added so that the formulas could work, such as sorting out the inverse of mod 11, or using the given table to calculate reversed numbers.

The hardest part about this task was getting the catch statements in the right place and making sure the formulas were implemented correctly as well as optimising the code by using a matrix for syndromes.







If I was to do this task again, I would make more use of Boolean functions for catching the errors and optimising the algorithms.

The brute force tasks introduced new built in functions within java that can be used like “MessageDigest” which output a set fixed length hash value for sha1. This introduced the getInstance function which was also handy for the start of ESD when testing servlets and sessions. It also taught me how to fully use ASCII values and set ranges for them, as well as reminding myself about hexadecimal values and that they are needed in sha1 as you can’t have 10 > as it goes into the alphabet. I split my code into three classes so I could just have one main class that called a brute force class contractor and ran over a certain amount of time (5 minutes).

The hardest part about this task was finding the fasted way to optimise the algorithm. I found that having a string that was the hash, and then setting up a class that has certain parameters set to follow a minimum and maximum set ASCII value from the chars. For example, for the non-bch task, it was the alphabet in lowercase as well as numbers 0-9, whereas for BCH, the alphabet could be removed leaving only the numbers, and the ASCII ranges were changed to match this, but the string length was increased by 4 due to BCH having 10 characters.









If I was to do this task in the future I would try and implement the use of more threads to spread the tasks, increasing the performance for the harder strings.

For task for I decided to use python due to recently improving my skills from the BioComputation module. Python has many built in functions to help code the stream cipher and Steganography using the required pseudo random number generator.

This task helped me visually understand how these ciphers work and how the correct variable lengths are needed to encode the message correctly, such as encoding every 7 characters with XOR.

Python made the task a lot simpler by being able to easily implement “random.()” for seed generation and by using Unicode string returns.

It was important to make sure the secret message couldn’t be found, luckily XOR is good at that. The original message was not very well encoded and was easy to read.

Overall, my java skills improved tremendously as well as my problem-solving skills. I have learnt the important of following simple math rules like “brackets first” so that formulas are correctly working. I use to struggle with parameter setting and calling before these tasks, so learning those really improved my codes performance. All of these tasks needed a strong implementation of catch statements that caught all the errors possible, especially BCH. If I was to do these tasks again I would either stick to using jFrame or move over to python as python is much simpler and the code is smaller, and I could improve the performance of the algorithms more using python.