Engima Assignment

February 20, 2017

1 The Assignment

The purpose of this assignment is to apply the concepts of a linked list in an interesting and engaging way.

Through the assignment, you are expected to make a version of the Enigma machine (more on this later). The structure of this will be as follows:

1. A Rotor class
2. An enigma class
3. A driver (you are not expected to implement this - this will run your code)

In addition, you can expect the following files which will be German U-boat reports from the North Atlantic (75 years ago):

1. **message1.txt:** A German U-boat was attacked and forced to submerge. Gives the enemy position, movement, and weather.
2. **message2.txt:** The U-boat's captain (Oberleutnant) reports nothing on a convoy's course. Cloudy and windy weather.
3. **message3.txt:** The U-boat is heading for fuel and needs binoculars (has only 4). The position given is wrong but weather is nice.

You will be expected to encrypt and decrypt the messages from English (so that German characters and short-hands do not further obfuscate the assignment).

2 The Code Expected

Here are the expectations for the submission. Relevant parts of the enigma are explained. **You are not expected to develop a full enigma emulator - read below for details.**

This is divided by class for convenience. A point-breakdown is provided at the end of this section.

2.1 The Rotor Class - Constructor

A rotor represents a rotor of the Enigma machine. The rotor mapped the alpha-numeric characters to other ones. An example is below:

Original	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
Encrypted	S	6	R	3	M	T	O	5	8	0	G	Q	W	J	P	H	4	Z	E	C

This rotor will be contructed by providing the `char` array:

```
char [] rotor = {'S', '6', 'R', '3', 'M', 'T', 'O', '5', '8', '0',
'G', 'Q', 'W', 'J', 'P', 'H', '4', 'Z', 'E', 'C', '1', '2', '7',
'D', 'N', 'V', 'Y', 'L', 'B', 'I', '9', 'F', 'A', 'X', 'K', 'U'};
```

The code should be above to make multiple rotors. In general, the first value will map to A and so until until the 27^{th} which will map to 1 and the final one which will map to 0.

For encryption convenience, the `char` array should be converted to a circular linked list with the head as the first value (that maps to A).

2.2 The Rotor Class - encodeOne

Encodes a single char. Find the corresponding value in your circular linked list and return it.

Note that this value will change as the rotor rotates - see below. **Do not include rotation here!**

2.3 The Rotor Class - decodeOne

Decodes a single char. Find the corresponding index in your circular linked list and return it.

Note that this value will change as the rotor rotates - see below. **Do not include rotation here!**

2.4 The Rotor Class - rotate

Every time a character is encoded, the rotor is rotated once. So if A mapped to C and B mapped to D, A would map to D once a character is encoded.

This method will encapsulate this behavior. A boolean will be returned to flag that the rotor has completed a full cycle. This will be useful for the Enigma class.

2.5 The Rotor Class - setState

The enigma was set to a different starting state at the start of a new message. This is needed to be able to encipher and decipher a message.

This would essentially rotate the rotor to a specific state before anything were encrypted.

For this method, only the character corresponding to A is given.

2.6 The Enigma Class - Constructor

The enigma class models a 3-rotor enigma. As it stands, the machine has characters for the start state so that the rotors can be reset.

All the data needed for execution is given.

2.7 The Enigma Class - encrypt

The encrypt method encrypts a string. The string is provided as a `java.util.LinkedList<Character>` and the output is expected to be a **copy**. Read the standard library for more.

The algorithm is as follows: encode the character through `r1`, then `r2`, then `r3`. After passing the result through the reflector, pass it through the rotors in the reverse order (3, 2, 1).

After this, each rotor is rotated once. If a rotor rotates a full circle, the next rotor is advanced. If `r1` completes a circle, `r2` is rotated. If that finishes a full circle, `r3` is advanced. Advancing `r3` would advance `r1`. **The `reflector` is never rotated.**

Note that if you encrypt an encrypted message after resetting a machine, you will get the original message back.

2.8 The Enigma Class - reset

This should reset the machine's internal state. This way many messages can be consistently enciphered.

2.9 Point Breakdown

The points will be allocated as follows (out of 75 total):

What	Points
Rotor - Constructor	12
Rotor - encodeOne	12
Rotor - decodeOne	12
Rotor - rotate	5
Rotor - setState	12
Enigma - Constructor	5
Enigma - encrypt	12
Enigma - reset	5

3 The History

This project is based on a famous series of events in the second world war.

At the outset, Germany and Britain had secure encryption systems. The German machine was called the enigma and acted almost entirely as above. There were more than 3 rotors (5 usually) and these could be placed into the machine in 1,054,560 possible ways after their starting positions are set. There was an additional step of the "Steckerbox" that swapped 10 random pairs of letters. This allows for around $1.5896256 * 10^{19}$ settings of the machine.

At Bletchley Park, British intelligence broke the enigma. By the end of the war, they were able to consistently intercept and interpret German war messages. They knew all.

This was the work of a large group of geniuses such as Alan Turing. Turing, using Polish enigma-breaking logic, found a way to implement a machine that tested all enigma states. Given a message and a guess of some of the decryption, Turing's Bombe machine would find all the possible enigma machine states. Each would be checked. Then, the most reliable setting would be considered the setting for the day.

Ironically, Turing's machine was not Turing complete. Towards the end of the war, the Colossus Mark 1 was built to decrypt the more complex Lorenz cipher. The issue was that while versions of the enigma machine were captured and analysed, the Lorenz cipher was not for a long time. The Colossus Mark 1 was the world's first Turing complete machine, that is, a modern computer.