ITI 1121. Introduction to Computing II - Summer 2015

Assignment 1: Implementing an Enigma Machine (100 points, weight 6.25%)

(Last Modified in June 10, 2015)

Due date: Friday June 12 at 11:59PM.
The assignment must be uploaded on Virtual Campus by the due date.
Late assignments are accepted between 1 min late up to a maximum of 24 hours late and they will receive a 30% penalty.

Learning objectives

- Using arrays to store and retrieve information.
- Understanding reference variables and class associations.
- Applying basic object oriented programming principles to solve problems.
- Editing, compiling, debugging, and running Java programs.
- Raising awareness concerning the university policies for academic fraud/plagiarism.

This assignment introduces Cryptography, "the practice and study of techniques for secure communication in the presence of third parties (called adversaries)". You will learn about physical encoding devices called Enigma machines and will emulate one specific model of Enigma machine, M3 Enigma Machine, used by the Germans in World War II to transmit secret strategic messages. Besides the above learning objectives, this assignment will make you more aware of how cryptography works by experimenting with a specific type of encoding based on a complex model of character substitution.

Background Information

Enigma machines were electro-mechanical cypher machines used in the last century for encoding and decoding secret messages. Early models appeared in the 1920's and later models were used by the Nazi German government during World War II. A cypher machine encodes a "plain text" into a "cypher text" using certain machine settings only known to the parties that are communicating. Setting the Enigma machine to the initial settings used to encode a "plain text" into the "cypher text", results in retrieving the original text when encoding the "cypher text". However, there were too many initial settings of the machine, so that it should be nearly impossible to discover the settings by trial-and-error.
During the war, British cryptologists at Bletchley park were able to break the Enigma.
Alan Turing was one of these key cryptologists.

There are many resources that talk about the history of Enigma Machines and how they work.
We note some of these resources here.

- **Enigma machine emulator and information in its mechanics** http://enigma.louisedade.co.uk
  You will be implementing Navy M3 Enigma Machine. Please check the pages:
  - Introduction
  - How Enigma Machine Work to familiarize yourself with the various physical parts of the machine, some of which will correspond to objects of various classes implemented in this assignment.
  - Use the Enigma Machine Emulator to check that your implementation is making correct encodings of various messages.

- **Wikipedia information on Enigma Machines** http://en.wikipedia.org/wiki/Enigma_machine
- Videos by Dr. James Grime:
  Watch a real enigma machine and how does it work.
  Explanation of the flaw in Enigma that allowed it to be broken
- Hands on activity to understand how Enigma works using paper and scissors:
  download here a pdf and print a paper Enigma.
  Recommended for who has not yet understood how the stepping of the rotors work relative to the wiring.

Problem

For this assignment you will implement one type of Enigma Machine called Enigma Machine M3.
Here we summarize the main parts of this physical device. Please, consult illustration here.

- **Keyboard (input device)**
  One letter is input via the Enigma Machine keyboard.
- **Plugboard**
  The operator may "plug" pairs of letters on the plugboard. This has the effect of switching these letters on the way in and on the way out of the Enigma machine; the other letters go through the plugboard unchanged.
- **Static rotor**
  It is a physical part connecting the plugboard to the rotors.
  It does nothing to the letter/signal; just turns wires to contacts.
- **Rotors (also called scramblers, wheels or drums)**
  These form the heart of the Enigma machine. Five types of rotors (I..V) can be used in any order for the 3 rotor positions which are positioned in the order **right, middle, left** (which we will be calling rotors 1, 2 and 3, since this is the order traversed by the incoming letter in the way in). The rotor performs a simple type of encryption: a substitution cypher. Each input letter gets transformed into an output letter when it passes through the internal wiring of the rotor. Each rotor implements a permutation function in which each of the 26 letters of the alphabet gets converted to another letter, in a 1-to-1 correspondence.
  The rotor can also be rotated relative to the static rotor, so that the static rotor 'A' output is not connected to the 'A' input on the rotating rotor but to another letter - we call this latter letter the "rotor position".
  Moreover, as each letter enters the rotors, a rotation mechanism called stepping occurs; in summary, before the letter is encoded, rotor1 always advance one position and rotor2 and rotor3 may advance (in a similar way as an odometer) depending on the position of the previous rotor being "at notch" (a designated special position for the type of rotor chosen).
  For example for rotor type I we have:
  input: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
  output: E K M F L G D Q V Z N T O W Y H X U S P A I B R C J
  and notch turnover position Q.
  If the rotor position is 'A' the rotor follows the permutation above in the following way. If a signal enters as letter 'B' from the right it leaves into the left in the position of letter 'K'; conversely, when traveling from left to right on the rotor 'K' would become 'B'.
  However, if the rotor is rotated to position 'C' the effect is that an 'A' enters from the right in the position of a 'C' exiting the wiring in the position of an 'M', but it will effectively leave the rotor as a 'K' (since the wire at position 'M' is rotated two positions back with respect to its original position, because the rotor position is at 'C' instead of an 'A').
- **Reflector**

After the signal passes through rotors 1, 2 and 3 (positions right, middle and left, respectively), the signal goes through the reflector, and passes again through rotors 3, 2 and 1 in this order going through their wires in the opposite direction as before (now from right to left of the rotor, effectively doing the reverse permutation for this rotor).
The function of the reflector is to switch the letters in pairs. It effectively implements a permutation that does single swaps between pairs of letters. Our Enigma Machine will use one of two types of reflectors: Reflector B and Reflector C, each corresponding to a given fixed permutation.
Note: without the reflectors the journey back and forth through the rotors would leave the message unscrambled because the reverse permutations on the way back would undo the scrambling of the way in.

- **Lampboard (output device)**
  This is the output device of the enigma machine. When a letter lights up the Enigma machine operator would note down the letter manually.

We will use the wiring of the rotors (I,II, III, IV, V) and reflectors (B, C) of original German Enigma (from http://www.enigma-replica.com/wiring.html).

You will be representing the Enigma machine as well as some of its physical parts as variables, objects and classes in our program in order to emulate how it works. You will be able to encode and decode messages as done by Navy M3 Enigma Machine used by the Germans in World War II.

Implementation

When we think of designing classes to emulate the Enigma machine described above, it is clear that we need a classes for the Enigma machine and some of its parts. We see that input and output and simple wire connections can be accomplished by simple character variables and input and output to various methods; however the more complex scrambling parts that implement permutations (plugboard, rotors, reflector) deserve their own classes.
Thus this assignment has main classes: **EnigmaMachine**, **PlugBoard**, **Rotor** and **Reflector**.
Auxiliary classes **Test** and **Util** help with testing and provide auxiliary functionality.
Please also use class **StudentInfo** to enter your personal information.

IMPORTANT NOTE (added July 4th):
One of the objectives of the assignment is that you practice with arrays.
Arrays arise naturally as a solution to implement the various permutations ("scrambling") done by the various parts of the machine (rotors, reflector, plugboard).
You are required to use a private array in rotor class to store the rotor wiring for the rotor object that is created.
Example: for rotor type-I you can use a char array, where wiring[0]='E', wiring[1]='K', etc , wiring[25]='J'.
Similarly, plugboard and reflector can have such an array storing their own scrambling information.
Class Util contains auxiliary methods to convert between letters 'A'..'Z' and numbers 0..25 that you may find useful in this context.

Assignment Parts and what to hand in:

1. **EnigmaMachine** (35 marks)
   An **EnigmaMachine** works as described above and has 1 plugboard, 3 rotors which we refer as rotor1, rotor2, rotor3 from right to left, and 1 reflector. The startup code already specify several private instance variables and public methods to be implemented.

A detailed description of the class and start up code can be found here:
- JavaDoc Documentation
- EnigmaMachine.java

2. **Plugboard** (15 marks)
   An **EnigmaMachine** has one **Plugboard**.
   Plugboard main methods are **plug** and **outLetter**; its auxiliary methods are constructors and **toString**.
   A detailed description of the class and start up code can be found here:
   - JavaDoc Documentation
   - PlugBoard.java

3. **Rotor** (30 marks)
   An **EnigmaMachine** has 3 **Rotors**.
   For your convenience **Rotor.class** startup code contains some class variables/constants for specifying the wiring permutations and notch positions for rotor types I, II, III, IV, V.
   A detailed description of the class and start up code can be found here:
   - JavaDoc Documentation
   - Rotor.java

4. **Reflector** (15 marks)
   An **EnigmaMachine** has one **Reflector**.
   For your convenience **Reflector.class** startup code contains a class variable/constant for specifying the wiring permutations for Reflector types B and C.
   A detailed description of the class and start up code can be found here:
   - JavaDoc Documentation
   - Reflector.java

5. **Auxiliary classes** Util.java, Test.java and StudentInfo.java are provided here.
   NEW!!
   Some tests are provided here and others can be created later for marking purposes.
   For instance, this is a JUnit test for class rotor: RotorTest.java

6. **Rules and regulations** (5 marks)
   - Please follow the programming requirements:
     Use the classes (and their interface) that we have provided. Our test cases (Test.class plus any extra testing classes provided or created in the future) must work with your classes; for this reason the interface of these classes must not be changed.
     Use an array (26 positions) to store the wiring/permutation used in classes rotor, reflector, plugboard.
   - Please follow the general directives for assignments.
   - You must abide by the following submission instructions:
     Include all your java files into a director called **u1234567** where 1234567 is your student id.
     Zip this directory and submit via blackboard.
     The directory must contain the following files (with your implementation):
     - **README.txt** (relevant information can be added here explaining your solution; which features you know to work or know not to work; any information for the marker relevant to marking)
     - EnigmaMachine.java (add implementation to it)
     - PlugBoard.java (add implementation to it)
     - Rotor.java (add implementation to it)

        - Reflector.java (add implementation to it)
        - StudentInfo.java (add student information to it)
        - Util.java (no changes expected)
        - Test.java (no changes expected; grounds for changing include: if your code crashes with Test.class, you may restrict your test to show the parts that are working and add appropriate comments to README.txt
    - The assignment is individual (plagiarism or collaborations towards the solution of the assignment will not be tolerated).
      Read the information on academic fraud below.

Academic fraud

This part of the assignment is meant to raise awareness concerning plagiarism and academic fraud. Please read the following documents.

- http://www.uottawa.ca/about/academic-regulation-14-other-important-information
- http://web5.uottawa.ca/mcs-smc/academicintegrity/home.php

Cases of plagiarism will be dealt with according to the university regulations.

By submitting this assignment, you acknowledge:

1. **having read the academic regulations regarding academic fraud, and**
2. **understanding the consequences of plagiarism**