**CSC 143 Assignment #1  
A Simple Enigma Model[[1]](#footnote-1)**

**BACKGROUND**

Substitution ciphers that encode a message by substituting one character for another go back at least as far as Julius Caesar, who used a rotating character scheme to encode military orders. This simple type of encryption is vulnerable to statistical attacks, however, as anyone who has solved CRYPTOGRAM puzzles can attest. In World War II, the Nazi military employed an encryption scheme that addressed this weakness of simple substitution ciphers. This scheme, implemented by typewriter-sized devices known as Enigma machines, gave the Nazis a tactical advantage that greatly contributed to their early success in the war. In fact, the eventual breaking of this coding scheme by researchers at Bletchley Park, England (including Alan Turing) is hailed as one of the turning points of the war.

Enigma machines used interchangeable rotors that could be placed in different orientations to obtain different substitution patterns. More significantly, the rotors rotated after each character was encoded, changing the substitution pattern and making the code very difficult to break. The behavior of the rotating rotors can be modeled, in a simplified form, by a device consisting of labeled, concentric rings. For example, the model below has three rings labeled with the letters of the alphabet and '#' (representing a space).

|  |  |
| --- | --- |
| http://nifty.stanford.edu/2009/reed-enigma/HW2/enigma.gif | To encrypt a character using this model, find the character on the inner rotor (i.e., the inside ring) and note the character aligned with it on the outer rotor (i.e., the outside ring), then find that character on the middle rotor (i.e., the middle ring) and output the one aligned with it on the outer rotor. After a character is encrypted, turn the inner rotor clockwise one step. Whenever the inner rotor returns to its original orientation, the middle rotor turns once in lock-step, just like the odometer in a car.  For example, in this configuration the character 'A' would be encrypted as 'N', since 'A' on the inner rotor is aligned with 'H' on the outer rotor, and 'H' on the middle rotor is aligned with 'N' on the outer rotor. After performing this encryption, the inner rotor is rotated clockwise, so the letter 'A' would next be encrypted as 'D'. |

In other words, the rings are used in the following order: **Inner 🡪 Outer 🡪 Middle 🡪 Outer**. Note that decrypting a message requires following the same steps, only in reverse (i.e., find the character on the outer rotor, note the character aligned with it on the middle rotor, find that character on the outer rotor, then output the character aligned with it on the inner rotor). In short: **Outer 🡪 Middle 🡪 Outer 🡪 Inner**.

**ASSIGNMENT DESCRIPTION**

For this assignment, you are to design a Java class named Enigma that simulates this three-ring model. You may assume that all Enigma models have the same outer rotor, as shown in the above diagram. That is, the outer rotor consists of the 26 capital letters and the '#' symbol (representing a space) in the following clockwise order: #BDFHJLNPRTVXZACEGIKMOQSUWY. Since the other rotors are interchangeable, however, their contents and alignment relative to the outer rotor must be specified when constructing an Enigma model. For example, the initial settings of the inner and middle rotors in the above diagram are #GNUAHOVBIPWCJQXDKRYELSZFMT and #EJOTYCHMRWAFKPUZDINSXBGLQV, respectively. Using an Enigma object, it should be possible to encode and decode text messages, with the appropriate rotation of the rotors occurring after each character encoding/decoding.

You should also design and implement a client program. For users who do not want to specify their own rotor settings, have a “default” option that uses the setting shown in the diagram. In addition, create an option that makes it simple for the user to specify the rotor settings on an Enigma model, and encode or decode text. Once you have your model working, test it by decoding the message   OKNNWRDHGERPILRLAMFZF#FMUC   using the diagram settings.

**SPECIFIC TASKS**

Using the outlines provided as a starting point:

* Complete all method stubs in the Enigma class (*Enigma.java*)
* Implement a user-friendly client program (*EnigmaClient.java*)
* Provide user instructions fully explaining how to use your program ad making data entry easy for your user
* Fully demonstrate the proper functioning of your program with your EnigmaClient.

Design and implementation guidelines:

* Javadoc comment all class files and methods
* Handle exceptions as appropriate and include @throws documentation in Javadoc comments
* Structured code - use methods to eliminate redundancy and break large methods into smaller, logical subproblems

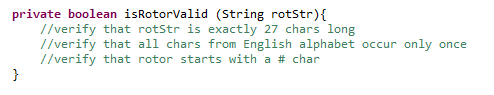
Additional (OPTIONAL) challenge:

* Create methods to validate user input appropriately – in your Enigma class, you need to check for rotor validity, but there will be other instances where you will be asking for user input in the main method in the EnigmaClient class.

For this assignment, you are being given an “outline” for each of the required classes. These outlines are described in the following section and can be found in the files provided along with this HW document: *Enigma.java* and *EnigmaClient.java*.

**ENIGMA CLASS OUTLINE**

This Enigma class outline contain method “stubs” along with commented descriptions of what each method should do. You need to write the actual methods, with the appropriate inputs and outputs. An example of a method stub is provided below:



This method stub tells you that the method takes a String as input and returns a Boolean value. The comments provided in the method stub list some hints on how the String should be validated. You need to write the method and the Javadoc comment. You should also comment your code using line comments as you see fit.

**WHAT TO SUBMIT**

You will need to submit the following files:

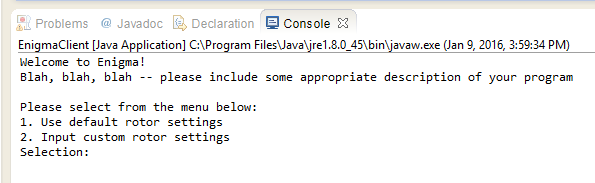
* Your *Enigma.java* file
* Your *EnigmaClient.java* file
* A text document showing how you tested your program (include screen shots or copy and paste sample output). Include a “Reflections” section in this text document that discusses any difficulties you had with the assignment, what you learned, what you found interesting, etc.

**PLEASE MAKE SURE THE JAVA FILES YOU SUBMIT COMPILE AND RUN – EVEN IF YOU ARE NOT ABLE TO FINISH THE PROGRAM COMPLETELY AND/OR CORRECTLY.**

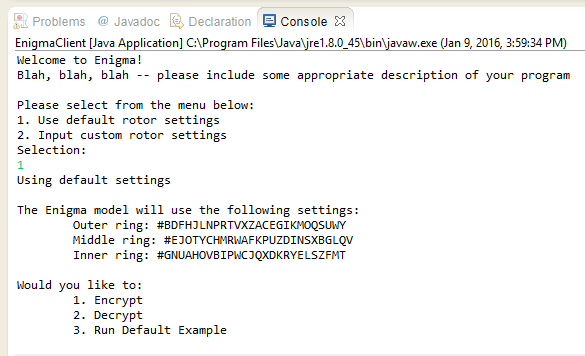
Sample output is provided below. You do not have to recreate this format exactly, but your program should have similar behavior.

**SAMPLE PROGRAM OUTPUT**

1. Upon running the program:

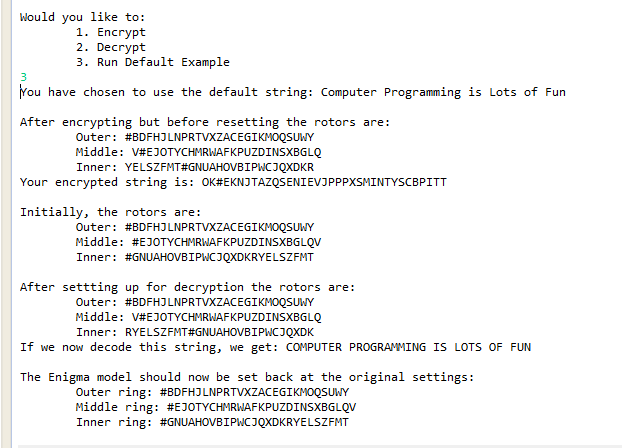


2. After first user selection:



3. After second user selection:

**NOTE:** You do not have to keep printing out rotor settings, but you may find this feature helpful when you are trying to write the encryption and decryption methods.



**OPTIONAL: MAKE YOUR OWN ENIGMA MODEL**

Note: if you would like to make your own 3-ring Enigma model out of paper, download this [online form](http://www.jambe.co.nz/makeenigma.html), cut out the rings, and follow the instructions at the site.

1. Adopted from Nifty Assignments - SIGCSE 2009 (http://nifty.stanford.edu/2009/reed-enigma/) [↑](#footnote-ref-1)