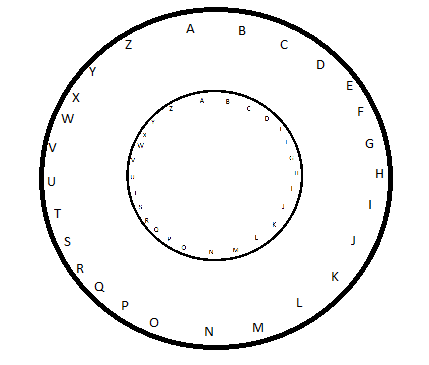
**The Caesar Cipher**

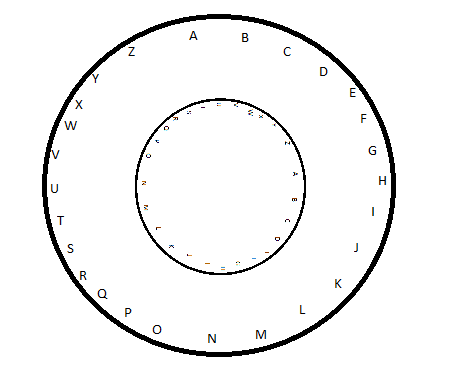
**I. Encrypting a message using the Caesar cipher.**

In order to encrypt a message using the Caesar cipher, one makes use of 2 disks,



in which the inner one is rotated clockwise by some angle to get some configuration.

The angle may be measured by the *number of characters* it rotates by, and is called the *shift*. For instance, it may be rotated by 7 *characters*, and this corresponds to *shift*=7. The disks then have a configuration as in the figure (below). Then, each character on the outer disk is matched to the character on the inner disk by drawing a line to the centre.



Mathematically, this corresponds to performing modular arithmetic, *modulo* 26.

**c= (c + shift)%26;**

It is said that we are *shifting* the characters by a factor of **shift**.

This transformation is used to encrypt the message *M* by treating it as an array of numbers (**a**-0, **b**-1, **c**-2, **d**-3, etc.) and matching each number to its *shifted* value.

**String Message= br.readLine();**

**int[]M= {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};//Memory initially blank**

**//Convert message into integer array**

**//(assuming all lowercase characters.)**

**for( i=0; i< Message.length(); i++ )**

**{**

**M[i]= (int) Message.charAt(i)-97;**

**}**

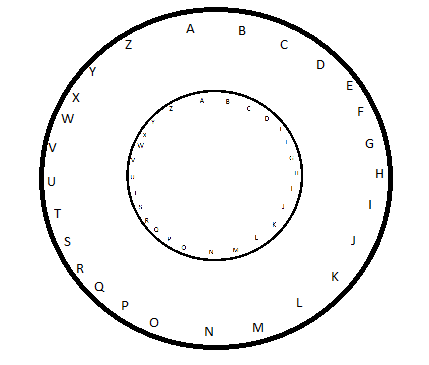
This array *M* itself may be transformed by shifting each number.

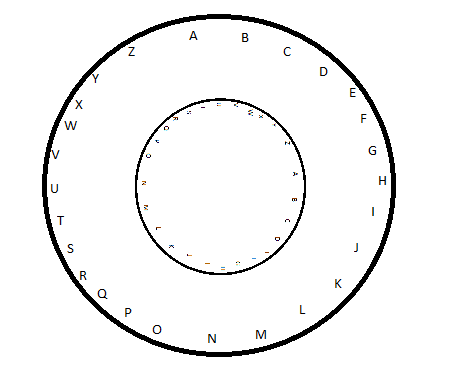
**//Encryption**

**for( i=0; i< Message.length(); i++ )**

**{ M[i]= ( M[i]+shift )%26 +97; }**

For decryption, the simplest method is to shift every number *back* by the shift amount. This will have the effect of rotating the smaller disk *anti-clockwise* to produce the original values.





It is directly achieved as:

**//Decryption**

**for( i=0; i< Message.length(); i++ )**

**{ M[i]= ( M[i]-97-shift )%26 + 97; }**

**APPENDIX.**

1. **Input/Output- characters vs integers**

The input and output must be done via characters- **a**, **b**, **c**, **d** …etc. But these are stored in memory as integers via the ASCII encoding. By this encoding, the characters are encoded as: **a**-97, **b**-98, **c**-99, **d**-100, etc.

In order to convert this to the more convenient numbering of letters, i.e. **a**-0, **b**-1, **c**-2, **d**-3, etc., we must *subtract* 97 from each of the characters. It is for this reason that we *subtract* 97 for calculation purposes (eg. for **c= (c + shift)%26**) and we *add* 97 to get back the ASCII encoding in order to display the output in characters.

Since Java has the facility of *explicit casting*, we may directly convert numbers to characters for the sake of printing the output using:

**//Convert back:**

**char[]\_M=**

**{'\_','\_','\_','\_','\_','\_','\_','\_','\_','\_','\_','\_','\_','\_','\_'};**

**for( i=0; i<Message.length(); i++ )**

**{ \_M[i]= (char)M[i]; }**

1. **Full Program.**

The above code can be used in a Java program as below.

**import java.io.\*;**

**public class CC**

**{**

**public static void main( String a[] ) throws Exception**

**{**

**BufferedReader br=**

**new BufferedReader(**

**new InputStreamReader( System.in )**

**);**

**int i,shift;**

**System.out.println("Enter shift:");**

**shift= Integer.parseInt(br.readLine());**

**System.out.println("Enter message:");**

**//1.Convert to int-array (see above)**

**//2.Encryption (see above)**

**//2.1.Convert back**

**//2.2.Print result:**

**System.out.println("Encryption:\n");**

**for( i=0; i<Message.length(); i++ )**

**{ System.out.println( \_M[i] ); }**

**//3.Decryption (see above)**

**//3.1. Convert back**

**//3.2.Print result:**

**System.out.println("Decryption:\n");**

**for( i=0; i<Message.length(); i++ )**

**{ System.out.println( \_M[i] ); }**

**}**

**}**

**C. Sample Output**

A test run:

