Cryptography, the study of secret writing, has been around for a very long time, from simplistic techniques to sophisticated mathematical techniques. No matter what the form however, there are some underlying things that must be done – encrypt the message and decrypt the encoded message.

One of the earliest and simplest methods ever used to encrypt and decrypt messages is called the Caesar cipher method, used by Julius Caesar during the Gallic war. According to this method, letters of the alphabet are shifted by three, wrapping around at the end of the alphabet. For example,

PlainTest: **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**
Caesar shift: **d e f g h I j k l m n o p q r s t u v w x y z a b c**

When encrypting a message, you take each letter of the message and replace it with its corresponding letter from the shifted alphabet. To decrypt an encoded message, you simply reverse the operation. That is, you take the letter from the shifted alphabet and replace it with the corresponding letter from the **plaintext** alphabet. Thus the string **the quick brown fox** becomes **wkh txlfn eurzq ira**

Another type of cipher is known as Transposition cipher. In this type of cipher, letters in the original message are re-arranged in some methodical way – for instance, reverse the letters in each string. Thus the string **the quick brown fox** becomes **eht kciuq nworb xof**

Still yet another cipher method is the Reverser cipher. This method does not only reverses the letters in each word, but as does the Transposition cipher, but it also reverses the result generated from the Transposition cipher. Hence the original message **the quick brown fox** becomes **xof nworb kciuq eht**

Class design
Here are three Cryptography methods – **Caesar**, **Transposition** and **Reverser**. They all have something in common. They encrypt and decrypt messages. That is, they take a string of words and translate each word using the encoding algorithm appropriate for that cipher. Thus each class cipher will need polymorphic encode() and decode() methods, which take a word and encodes and decodes it according to the rule of the particular cipher.

From a design perspective, the encrypt() method and the decrypt() methods will be the same for every class. They simply break message into words and have each word encode or decode. However, the encode and decode methods will be different for each cipher. **Figure 1** shows a hierarchy of the classes.

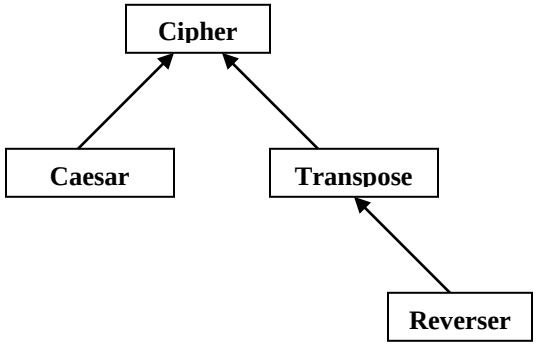

Figure 1. Inheritance hierarchy

From the above analysis a partial abstract class Cipher is depicted be by **Listing 1**.

```
import java.util.StringTokenizer;

public abstract class Cipher
{
    private String          message;
    StringBuffer            encrypted_message, decrypted_message;

    public Cipher(String text)
    {
        // Complete the constructor definition
    }

    public final void encrypt()
    {
        /* The message string is tokenized into individual words,
        *           and each word is encoded by calling the encode method
        */

        encrypted_message = new StringBuffer();
        StringTokenizer words = new StringTokenizer(message);

        while(words.hasMoreTokens())
        {
            String s = words.nextToken();
            s = encode(s) + " ";
            encrypted_message.append(s);
        }
    }

    public final void decrypt(String message)
    {
        /* The encoded message string is tokenized into individual words,
        * and each word is encoded  by calling the decode method
        */

         // Supply the code that will decrypt the encrypted string
    }

    public String getEncodedMessage()
    {
        return encrypted_message.toString();
    }

    public String getDecodedMessage()
    {
        return decrypted_message.toString();
    }

    public abstract String encode(String s);
    public abstract String decode(String s);
}
```

Listing 1. Abstract class Cipher

The class Caesar inherits the abstract class Cipher. This class defines the methods code and decode. The method encode takes a String parameter and returns a String result. It takes each character of the parameter and performs a Caesar shift on the character. That is, a shift with possible wrap around can be coded as follows:

char ch = word.charAt(i);
ch = (char)('a' + ch – 'a' + 3) % 26);

The method decode does the reverse. **Listing 2** shows a partial definition of this class - Caesar

```
public class Caeser extends Cipher
{
	public Caeser(String s)
	{
		super(s);
	}
	public String encode(String word)
	{
		 return code(word,Constants.ENCODE_SHIFT );
	}

	public String decode(String word)
	{
	    // Complete this method so that it decodes the encoded string
	}

	String code(String word, int SHIFT)
	{
		StringBuffer result = new StringBuffer();

		for (int i = 0; i < word.length(); i++)
		{
			char ch = word.charAt(i);
			ch = determineCharacter(ch, SHIFT);
			result.append(ch);
		}
		return result.toString();
	}

	public char determineCharacter(char ch, final int shift)
	{
		if(Character.isUpperCase(ch))
			ch = (char)('A' + (ch - 'A' + shift) % Constants.WRAP_AROUND);
		// Complete the if/else so that lower case letters are accounted for
		return ch;
	}
}
```

Listing 2.

In similar fashion, the class Transpose inherits Cipher and defines the methods code and encode. **Listing 3** shows an incomplete definition of the class Transpose.

```
public class Transpose extends Cipher
{
        Transpose(String s)
        {
                super(s);
        }

        public String encode(String word)
        {
                StringBuffer result = new StringBuffer(word);
                result.reverse();
                return result.toString();
        }
        public String decode(String word)
        {
                // Complete this method so that it reverses the encoded string;
        }
}
```

Listing 3

The class Reverser inherits the class Transpose. **Listing 4** shows an incomplete definition of this class.

```
public class Reverser extends Transpose
{
        public Reverser(String s)
        {
                // Complete the constructor
        }

        public String reverseText(String word)
        {
        // Complete this method so that it reverses the original string
        }

        public String decode(String word)
        {
             // Complete this method so that it reverses the reversed string
        }
}
```

Listing 4

Things to do
From the above discussion complete the classes Cipher, Caesar, and Transpose, and Reverser. In addition, define an interface called Constants that will store the value **26** in the identifier **WRAP_AROUND**; similarly, store **3** in the identifier, **ENCODE_SHIFT**, and **23** in the identifier **DECODE_SHIFT** (needed to decode the encoded message). The code in the method **determineCharacter(char ch, int shift)** accounts for lower case letters only, **extend the code to include both upper and lower case letters**.

Listing 5 shows a typical implementation of the algorithms.

```
import javax.swing.JOptionPane;

public class TestEncryption
{
        public static void main(String arg[])
        {
                String code, output = "";

                String text = JOptionPane.showInputDialog("Enter message");

                output += "The original message is \n" + text + "\n";

                Cipher c = new Caeser(text);
                c.encrypt();
                code = c.getEncodedMessage();
                output += "\nCeasar Cipher\nThe encrypted message is \n" + code + "\n";
                c.decrypt(code);
                code = c.getDecodedMessage();
                output +="The decrypted message is \n" + code + "\n";

                c = new Transpose(text);
                c.encrypt();
                code = c.getEncodedMessage();
                output += "\nTranspose\nThe encrypted Transpose message is \n" + code + "\n";
                c.decrypt(code);
                code = c.getDecodedMessage();
                output +="The decripted Transpose message is \n" + code + "\n";

                c = new Reverser(text);
                c.encrypt();
                code = c.getEncodedMessage();
                code =c.reverseText(code);
                output += "\nReverser\nThe encrypted Reverse message is \n" + code+ "\n";
                code = c.decode(code);
                output+="The decrypted Reverse message is \n" + code;
                display(output);
        }

        static void display(String s)
        {
              JOptionPane.showMessageDialog(null, s, "Encrypt/decrypt",
                                                        JOptionPane.INFORMATION_MESSAGE);
        }
}
```

Fix the problem

Listing 5.

Listing 5 has errors as shown, you must find these errors and fix them, in order for the program to compile. This is the only section that has problems, therefore do not alter any other parts of the code.

A typical output from the program can be seen in **Fig. 2.**

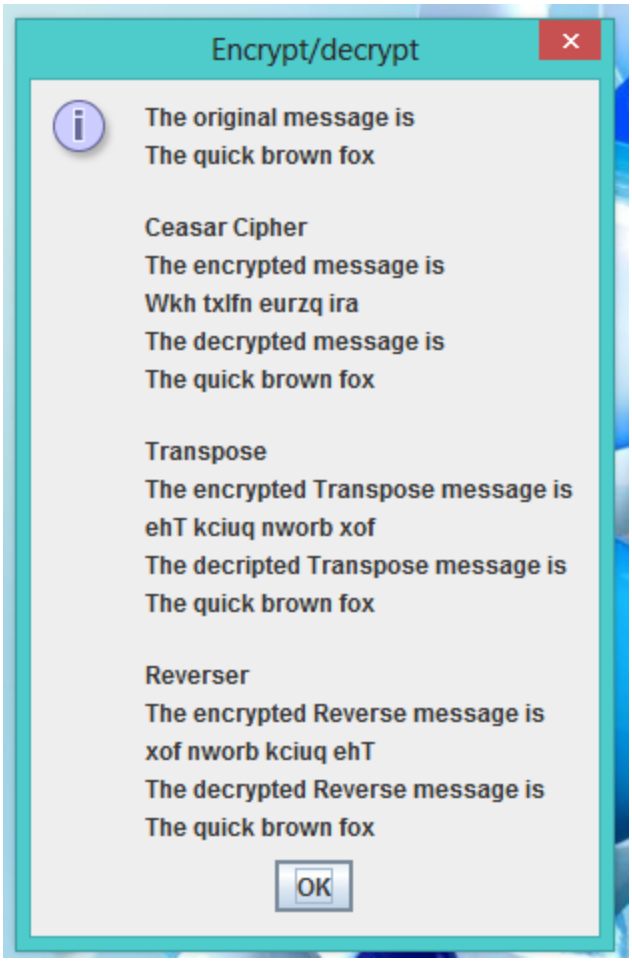
Encrypt/decrypt
The original message is
The quick brown fox
Ceasar Cipher
The encrypted message is
Wkh txlfn eurzq ira
The decrypted message is
The quick brown fox
Transpose
The encrypted Transpose message is
ehT kciuq nworb xof
The decripted Transpose message is
The quick brown fox
Reverser
The encrypted Reverse message is
xof nworb kciuq ehT
The decrypted Reverse message is
The quick brown fox
OK

Fig. 2.