Criterion C: Development

Techniques performed to develop the *CryptoVault*:

- **The use of two Integrated Development Environments (IDEs)**
- **Use of Arrays and Arraylists derived from the Java Collections Framework in order to:**
    1. **manipulate setlists and arrays**
    2. **creating new lists and arrays that are dynamic with each Collection object (Arrays and Arraylists) being dependent on each other**
    3. **Includes the four cryptographic methods explained.**
- **Implementing a Graphical User Interface**

Use of two IDEs:

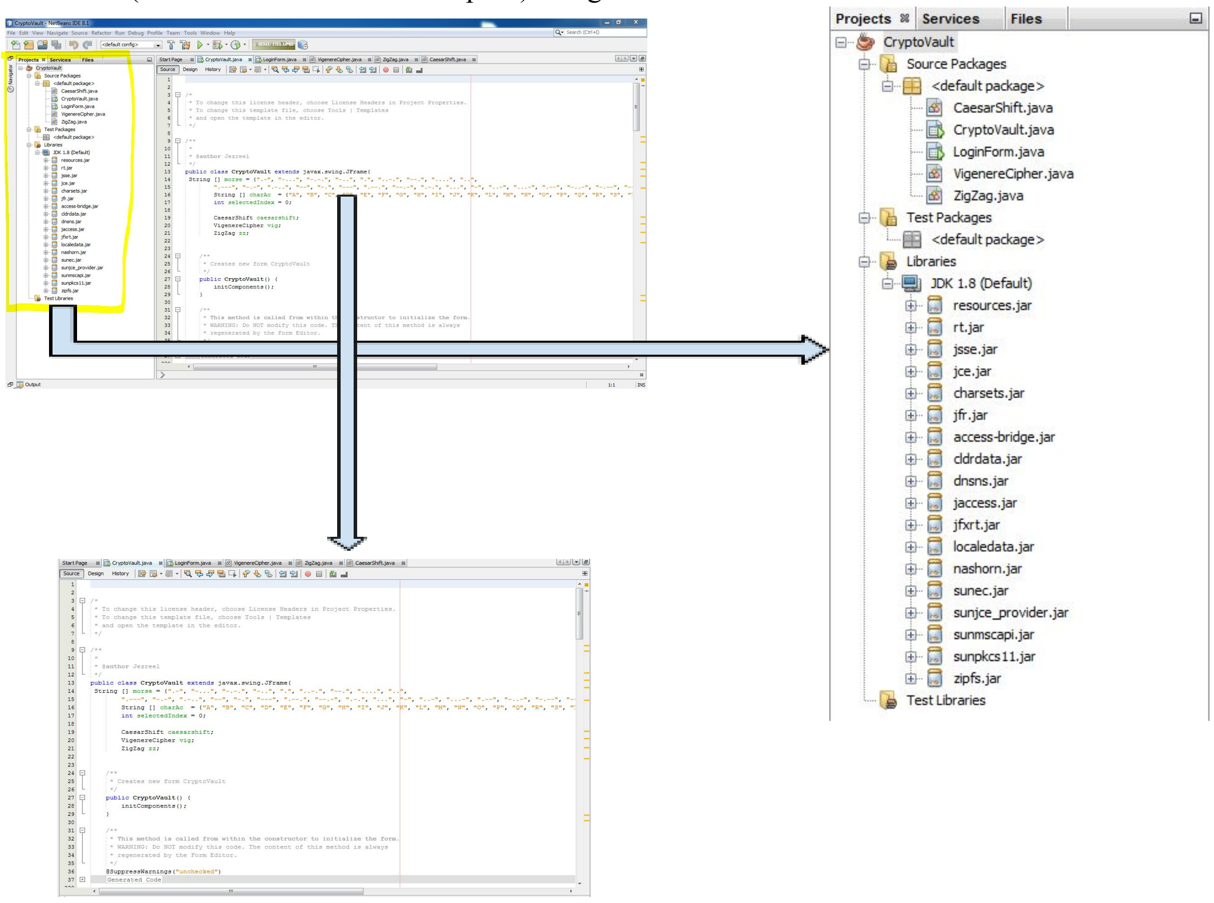
Throughout the development process, the use of NetBeans IDE and BlueJ IDE has allowed me to develop my algorithmic thinking by separating two tasks: the development of a GUI, with the use of NetBeans, and the development of my methods, with the use of BlueJ.

Differentiation of each IDE's unique features which I took into account:

NetBeans	**BlueJ**
Contains a form creator for developing a GUI	Simple and less cluttered than IDEs like NetBeans and Eclipse.
Automatically compiles the program (CoS)	Objects are interactable allowing the developer to arrange their classes in a structured manner.
Contains a toolbar that allows the developer to view the classes contained within the project while still being able to edit code in the same workspace (Customizable Workspace).	Color coded statements depending on the statement type (e.g. a *for* statement is highlighted in purple while a method is highlighted in yellow)
Automatically inserts necessary parentheses and brackets, and also provides parameter suggestions while editing source code. NetBeans also displays error with code during the editing process and offers solutions on the side with a lightbulb indication suggesting what course of action the developer can take in order to resolve errors.	

NetBeans:

Toolbar (on the left side of the workspace) alongside source code editor in NetBeans



Projects | Services | Files

- CryptoVault
  - Source Packages
    - <default package>
      - CaesarShift.java
      - CryptoVault.java
      - LoginForm.java
      - VigenereCipher.java
      - ZigZag.java
  - Test Packages
    - <default package>
  - Libraries
    - JDK 1.8 (Default)
      - resources.jar
      - rt.jar
      - jsse.jar
      - jce.jar
      - charsets.jar
      - jfr.jar
      - access-bridge.jar
      - cldrdata.jar
      - dnsns.jar
      - jaccess.jar
      - jfxrt.jar
      - localedata.jar
      - nashorn.jar
      - sunec.jar
      - sunjce_provider.jar
      - sunmscapi.jar
      - sunpkcs11.jar
      - zipfs.jar
  - Test Libraries

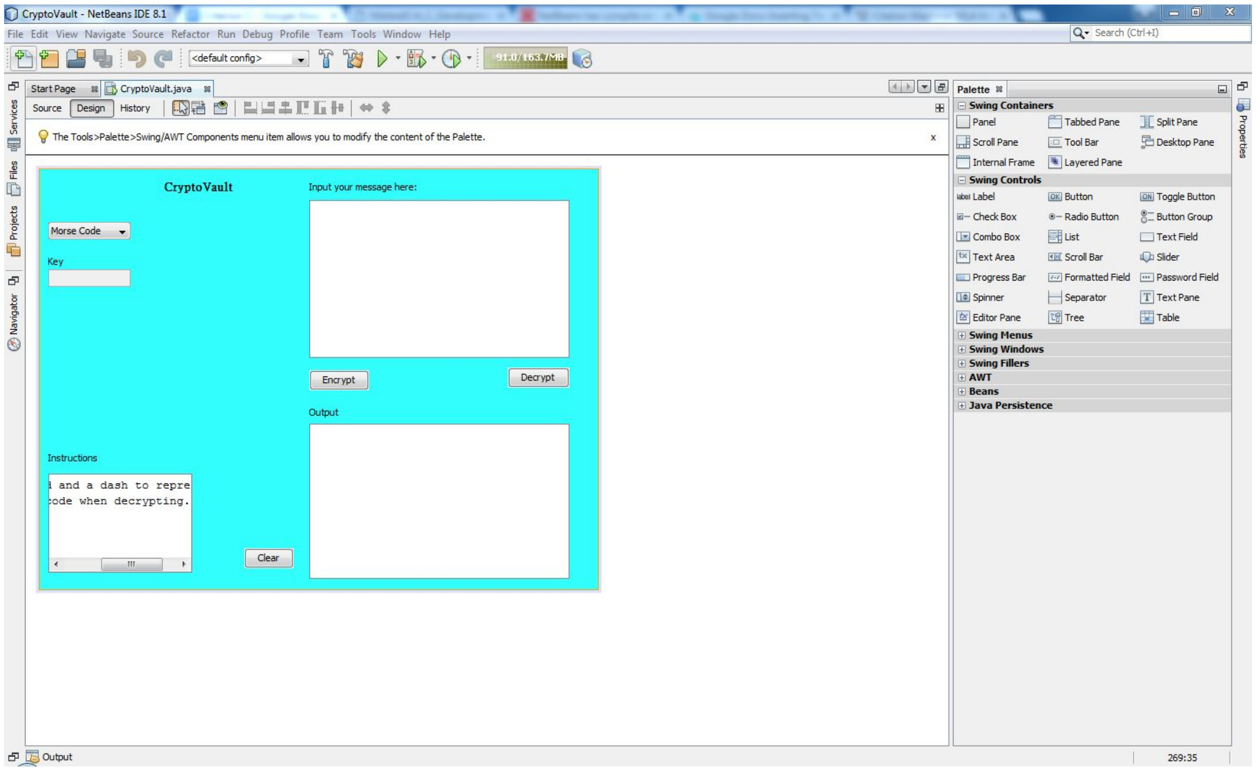
```

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

/**
 *
 * @author Jezreel
 */
public class CryptoVault extends javax.swing.JFrame{
 String [] morse = {".-", "-...", "-.-.", "-..", ".", "..-.", "--.", "....", "..",
        ".---", "-.-", ".-..", "--", "-.", "---", ".--.", "--.-", ".-.", "...", "-", "..-", "...-", ".--", "-..-", "-.--", "-
        String [] charAc  = {"A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M", "N", "O", "P", "Q", "R", "S", "
        int selectedIndex = 0;

        CaesarShift caesarshift;
        VigenereCipher vig;
        ZigZag zz;


    /**
     * Creates new form CryptoVault
     */
    public CryptoVault() {
        initComponents();
    }

    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    Generated Code
```

Being able to have an organized and structured workspace allowed me to switch between tasks back and forth in order to save time exiting the workspace in order to view classes contrary to BlueJ. The toolbar on the left shows the classes and forms being used which minimizes occurrences of duplicate forms and classes that may hinder the development process which I ran into while using BlueJ.

NetBeans also contains a Compile on Save (CoS) feature that catalyzes the process of editing and compiling source code.[1]

[1] Strobl, Roman. "NetBeans Has Compile on Save! (Roumen's Weblog)." *NetBeans Has Compile on Save! (Roumen's Weblog)*. Blog for Roumen, 26 July 2008. Web. 17 Feb. 2016.

NetBeans Form Creator

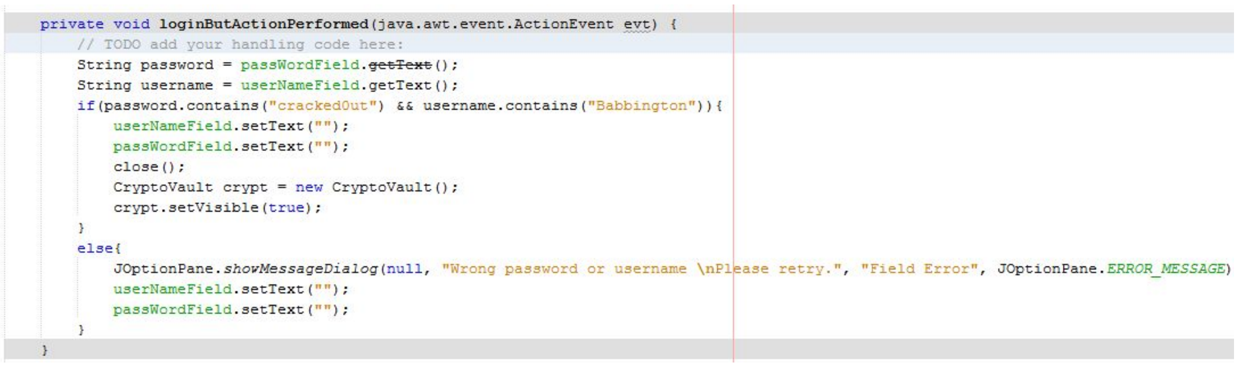

CryptoVault - NetBeans IDE 8.1
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
<default config>
Start Page
CryptoVault.java
Source Design History
CryptoVault
Input your message here:
Morse Code
Key
Encrypt
Decrypt
Output
Instructions
Clear
Palette
Swing Containers
Panel Tabbed Pane Split Pane
Scroll Pane Tool Bar Desktop Pane
Internal Frame Layered Pane
Swing Controls
Label Button Toggle Button
Check Box Radio Button Button Group
Combo Box List Text Field
Text Area Scroll Bar Slider
Progress Bar Formatted Field Password Field
Spinner Separator Text Pane
Editor Pane Tree Table
Swing Menus
Swing Windows
Swing Fillers
AWT
Beans
Java Persistence
Services Files Projects Navigator Properties
Output
269:35

The Tools>Palette>Swing/AWT Components menu item allows you to modify the content of the Palette.

```
i and a dash to repre
ode when decrypting.
```

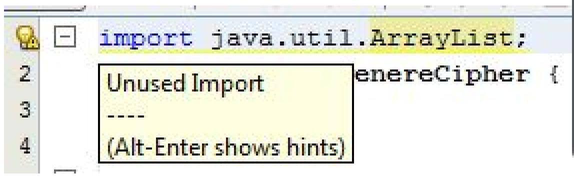
The form creator future of NetBeans was the major factor of why I chose to use this IDE for developing the GUI. The simple drag-and-drop methods saved much time in hard-coding the source code for the GUI. After dragging creating the different GUI objects, all I needed to do was double-click on an object to edit the source code for event-listeners and method processing.

Example Button source code editor



```
private void loginButActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    String password = passWordField.~~getText~~();
    String username = userNameField.getText();
    if(password.contains("crackedOut") && username.contains("Babbington")){
        userNameField.setText("");
        passWordField.setText("");
        close();
        CryptoVault crypt = new CryptoVault();
        crypt.setVisible(true);
    }
    else{
        JOptionPane.showMessageDialog(null, "Wrong password or username \nPlease retry.", "Field Error", JOptionPane.ERROR_MESSAGE)
        userNameField.setText("");
        passWordField.setText("");
    }
}
```

Code Assistance in NetBeans



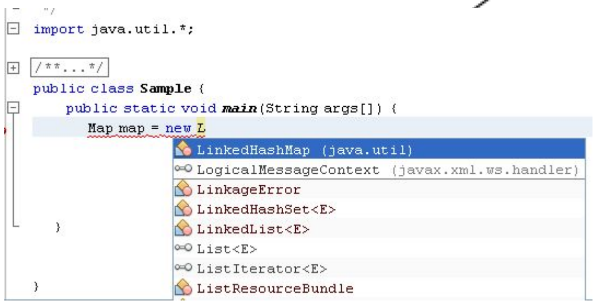
```
import java.util.ArrayList;
                       enereCipher {
```

Unused Import

(Alt-Enter shows hints)

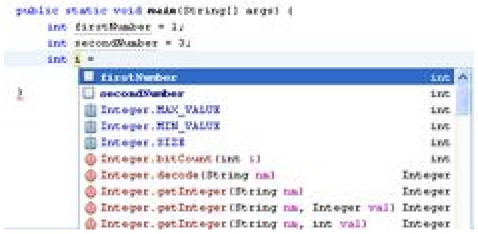
Notifies the user of unused class imports, which again reiterates the idea of organization and clarity during development

Contains a smart-suggestion feature and parameter completer to enhance efficiency and minimize syntax errors.



```
import java.util.*;

/**...*/
public class Sample {
    public static void main(String args[]) {
        Map map = new L
    }
}
```

LinkedHashMap (java.util)
LogicalMessageContext (javax.xml.ws.handler)
LinkageError
LinkedHashSet<E>
LinkedList<E>
List<E>
ListIterator<E>
ListResourceBundle

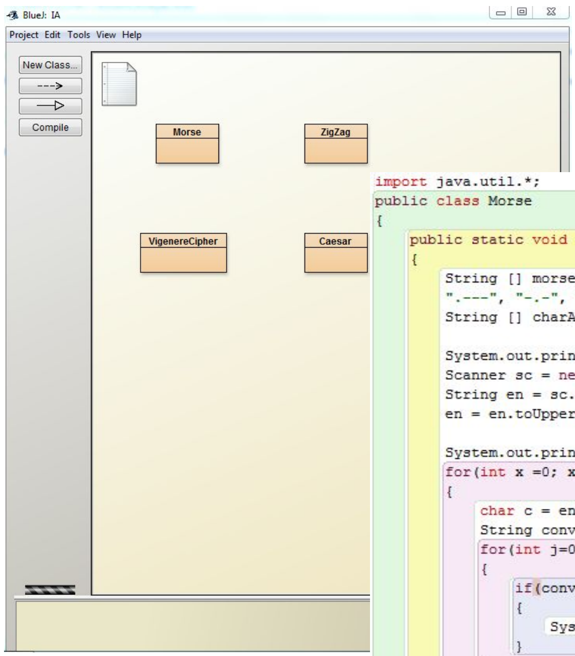


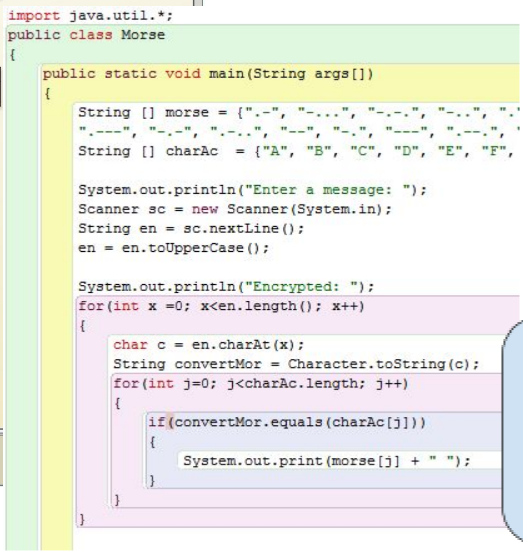
```
public static void main(String[] args) {
    int firstNumber = 1;
    int secondNumber = 3;
    int i =
```

firstNumber	int
secondNumber	int
Integer.MAX_VALUE	int
Integer.MIN_VALUE	int
Integer.SIZE	int
Integer.bitCount(int i)	int
Integer.decode(String nm)	Integer
Integer.getInteger(String nm)	Integer
Integer.getInteger(String nm, Integer val)	Integer
Integer.getInteger(String nm, int val)	Integer

"Code Assistance in the NetBeans IDE Java Editor: A Reference Guide." *To NetBeans IDE*. Web. 19 Feb. 2016.

BlueJ:





```
import java.util.*;
public class Morse
{
    public static void main(String args[])
    {
        String [] morse = {".-", "-...", "-.-.", "-..", ".
        ".---", "-.-", ".-..", "--", "-.", "---", ".--.", 
        String [] charAc  = {"A", "B", "C", "D", "E", "F",

        System.out.println("Enter a message: ");
        Scanner sc = new Scanner(System.in);
        String en = sc.nextLine();
        en = en.toUpperCase();

        System.out.println("Encrypted: ");
        for(int x =0; x<en.length(); x++)
        {
            char c = en.charAt(x);
            String convertMor = Character.toString(c);
            for(int j=0; j<charAc.length; j++)
            {
                if(convertMor.equals(charAc[j]))
                {
                    System.out.print(morse[j] + " ");
                }
            }
        }
```

BlueJ, contrary to NetBeans, has a less cluttered environment which I found perfect for developing methods and classes separately which often contain meticulous actions. The classes can also be moved around and organized within the project menu shown on the far left which displayed more visibility compared to NetBeans' toolbar.

Color coded statements enabled me to differentiate control structures, methods, and class imports. By doing so I was able to develop my program in a fluid manner.

Development of Cryptographic Methods using the Collections Framework:

I first developed the methods in BlueJ by creating a console version of each cipher in order to test if the methods worked.

Source Code for Morse Class:

```
import java.util.Arrays;
import java.io.*;
import java.util.*;
public class Morse
{
    public static void main(String args[])
    {
        String [] morse = {".-", "-...", "-.-.", "-..", ".", "..-.", "--.", "....", "..",
        ".---", "-.-", ".-..", "--", "-.", "---", ".--.", "--.-", ".-.", "...", "-", "..-",
"...-", ".--", "-..-", "-.--", "--..", ".----", "..---", "...--", "....-", ".....", "-....",
"--...", "---..", "----.","-----"};
        String [] charAc  = {"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", "1", "2", "3", "4", "5", "6",
"7", "8", "9", "0"};

        System.out.println("Enter a message: ");
        Scanner sc = new Scanner(System.in);
        String en = sc.nextLine();
        en = en.toUpperCase();
        String output = "";
        System.out.println("Encrypted: ");
        for(int x =0; x<en.length(); x++)
        {
            char c = en.charAt(x);
            String convertMor = Character.toString(c);
            for(int j=0; j<charAc.length; j++)
            {
                if(convertMor.equals(charAc[j]))
                {
                    output += morse[j] + " ";
                }
            }
        }
        System.out.println(output);
        System.out.println(" ");
        System.out.println("Enter a morse code message: ");
        Scanner sc1 = new Scanner(System.in);
        String dec = sc1.nextLine();
        String [] dec_array = dec.split(" ");

        System.out.println("Decrypted: ");
        for(int l = 0; l<dec_array.length; l++)
        {
            for(int p = 0; p<morse.length; p++)
            {
                if(dec_array[l].equals(morse[p]))
                {
                    System.out.print(charAc[p] + " " );
                }
            }
        }
       }
    }
```

First I created two String arrays: the first array `morse` contains the morse code alphabet and numerical system that the user's input will be compared to; the second array contains the standard capitalized alphabetic system and the counting numbers which are the available characters the user must input in order to use this cipher. The program then runs the user's input message and converts it to morse code using the following for-loop:

```
for(int x =0; x<en.length(); x++)
        {
            char c = en.charAt(x);
            String convertMor = Character.toString(c);
            for(int j=0; j<charAc.length; j++)
            {
                if(convertMor.equals(charAc[j]))
                {
                    output += morse[j] + " ";
                }
            }
        }
```

The outer for-loop loops through the entire message that the user inputted and converts each character into a String in order for the character to be compared to the `charAc` String array. The second for-loop then loops through the `charAc` array. The if-statement then determines if the character the user entered is within the array. If so, then the program refers to the `morse` array to find the corresponding index of the index presented in the `charAc` array and prints out the corresponding `morse` element. This repeats till each character the user inputs has been converted.

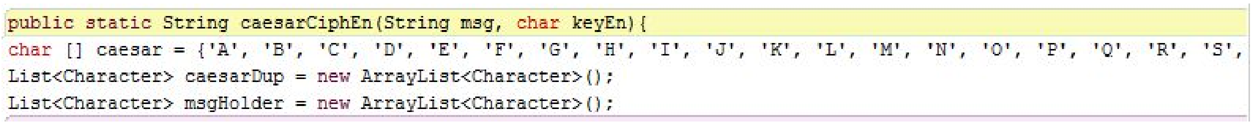
The deciphering method is the reverse of the encipher method above, however, it first has to split each morse code "character" (since they are strings after all) into separate strings into a String array, `dec_array`, and uses this array for the `morse` array to refer to:

```
String dec = sc1.nextLine();
        String [] dec_array = dec.split(" ");

for(int l = 0; l<dec_array.length; l++)
        {
            for(int p = 0; p<morse.length; p++)
            {
                if(dec_array[l].equals(morse[p]))
                {
                    System.out.print(charAc[p] + " " );
                }
            }
        }
       }
    }
```

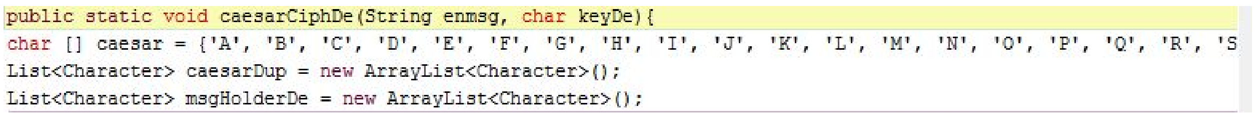
Source Code CaesarShift Class:

```
import java.util.Arrays;
import java.io.*;
import java.util.*;
import java.lang.Object;
import java.util.Collections;
import java.util.ArrayList;
//

public class Caesar{
    public static void main(String args[]){
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter a key: ");
        char encryptKey = sc.next().charAt(0);

        Scanner sc1 = new Scanner(System.in);
        System.out.println("Enter your message: ");
        String inputEnMes = sc1.nextLine();
        inputEnMes = inputEnMes.toUpperCase();
        System.out.println(caesarCiphEn(inputEnMes, encryptKey.toUpperCase()));

        Scanner sc2 = new Scanner(System.in);
        System.out.println("Enter your key used to encrypt: ");
        char decryptKey = sc2.next().charAt(0);

        Scanner sc3 = new Scanner(System.in);
        System.out.println("Enter your encrypted message: ");
        String inputDeMes = sc3.nextLine();
        inputDeMes = inputDeMes.toUpperCase();
        caesarCiphDe(inputDeMes, decryptKey.toUpperCase());

    }

    public static String caesarCiphEn(String msg, char keyEn){
    char [] caesar = {'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'};
    List<Character> caesarDup = new ArrayList<Character>();
    List<Character> msgHolder = new ArrayList<Character>();
    for(int i = 0; i<caesar.length; i++)
    {
        caesarDup.add(caesar[i]);

    }//CREATES AN ARRAYLIST FOR SHIFTING THE CIPHER

    int shift = 0;
    for( int j = 0; j < caesar.length; j++)
        {

            if(keyEn==(caesar[j]))
            {
                shift = j;
            }
        }//SHIFTER METHOD USING KEY

    for(int m = 0; m<msg.length(); m++)
    {
        msgHolder.add(msg.charAt(m));
    }//CREATES THE STRING ARRAYLIST

    Collections.rotate(caesarDup, -shift); //ROTATES CIPHER ACCORDING TO KEY
    for(int l = 0; l<msgHolder.size(); l++)
    {

        for(int k = 0; k<caesar.length; k++)
        {
            if(msgHolder.get(l) == caesar[k])
            {

            msgHolder.set( l,caesarDup.get(k));
            break;


        }
        }
    }//ENCRYPTS THE MESSAGE

    StringBuffer enToString = new StringBuffer();
    for(char msgH : msgHolder){
        enToString.append(msgH);
    } //ARRAY to STRING

    Collections.rotate(caesarDup, shift);
    return("" + enToString);

    }

    public static void caesarCiphDe(String enmsg, char keyDe){
    char [] caesar = {'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'};
    List<Character> caesarDup = new ArrayList<Character>();
    List<Character> msgHolderDe = new ArrayList<Character>();
    for(int i = 0; i<caesar.length; i++)
    {
        caesarDup.add(caesar[i]);

    }//CREATES AN ARRAYLIST FOR SHIFTING THE CIPHER

    int shiftDe = 0;
    for( int j = 0; j < caesar.length; j++)
        {

            if(keyDe==(caesar[j]))
            {
                shiftDe = j;
            }
        }//SHIFTER METHOD FOR KEY IN REVERSE

        for(int m = 0; m<enmsg.length(); m++)
    {
        msgHolderDe.add(enmsg.charAt(m));
    }//CREATES THE STRING ARRAYLIST

     Collections.rotate(caesarDup, -shiftDe);

    for(int l = 0; l<msgHolderDe.size(); l++)
    {

        for(int k = 0; k<caesarDup.size(); k++)
        {
            if(msgHolderDe.get(l) == caesarDup.get(k))
            {

            msgHolderDe.set( l,caesar[k]);
            break;


        }
        }
    }//DECRYPTS THE MESSAGE

    StringBuffer deToString = new StringBuffer();
    for(char msgH : msgHolderDe){
        deToString.append(msgH);
    } //ARRAY to STRING

     System.out.println("Decrypted: " + deToString);
     Collections.rotate(caesarDup, shiftDe);


    }
}
```

Two action methods are used in this class `caesarCiphEn` and `caesarCiphDe`. Each method first creates an initial array, `caesar`, which ArrayLists `caesarDup` and `msgHolder` (`msgHolderDe` for the `caesarCiphDe` method) will refer to in order to create a dynamic ArrayList. Each method accepts a String input and a Character which is a key (NOTE: The key used in each class is set to uppercase[2]):

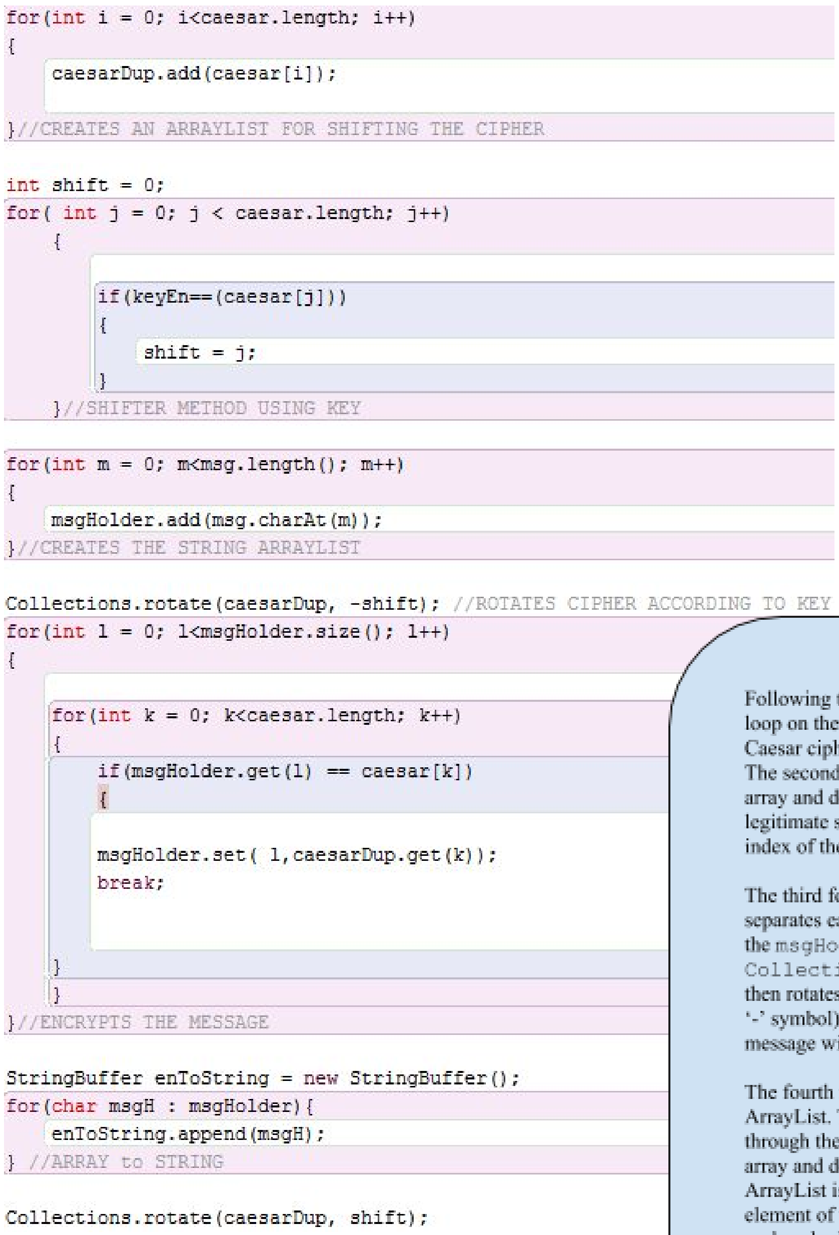


```
public static String caesarCiphEn(String msg, char keyEn){
char [] caesar = {'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S',
List<Character> caesarDup = new ArrayList<Character>();
List<Character> msgHolder = new ArrayList<Character>();
```



```
public static void caesarCiphDe(String enmsg, char keyDe){
char [] caesar = {'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S
List<Character> caesarDup = new ArrayList<Character>();
List<Character> msgHolderDe = new ArrayList<Character>();
```

[2] using `.toUpperCase()` method.



```
for(int i = 0; i<caesar.length; i++)
{
    caesarDup.add(caesar[i]);

}//CREATES AN ARRAYLIST FOR SHIFTING THE CIPHER

int shift = 0;
for( int j = 0; j < caesar.length; j++)
    {

        if(keyEn==(caesar[j]))
        {
            shift = j;
        }
    }//SHIFTER METHOD USING KEY

for(int m = 0; m<msg.length(); m++)
{
    msgHolder.add(msg.charAt(m));
}//CREATES THE STRING ARRAYLIST

Collections.rotate(caesarDup, -shift); //ROTATES CIPHER ACCORDING TO KEY
for(int l = 0; l<msgHolder.size(); l++)
{

    for(int k = 0; k<caesar.length; k++)
    {
        if(msgHolder.get(l) == caesar[k])
        {

        msgHolder.set( l,caesarDup.get(k));
        break;

    }
    }
}//ENCRYPTS THE MESSAGE

StringBuffer enToString = new StringBuffer();
for(char msgH : msgHolder){
    enToString.append(msgH);
} //ARRAY to STRING

Collections.rotate(caesarDup, shift);
```

Following the creation of the ArrayLists, using the first for-loop on the left, the second for-loop determines the shift of the Caesar cipher which is stored in the ArrayList `caesarDup`. The second for-loop loops through the original `caesar` array and determines whether or not the key entered is legitimate since the key must be a letter. If the key is valid, the index of the key in `caesar` then becomes the shift factor.

The third for-loop then takes the user's input message and separates each character into a separate element which forms the `msgHolder` ArrayList The statement : `Collections.rotate(caesarDup, -shift);` then rotates the `caesarDup` ArrayList to the left (hence the '-' symbol) forming the now shifted Caesar Cipher which the message will refer to in order to be encrypted.

The fourth for-loop then loops through the `msgHolder` ArrayList. The nested for-loop within this for-loop loops through the original alphabet represented by the `caesar` array and determines if an element within the `msgHolder` ArrayList is within the `caesar` array. If it is, then the element of that index within the `msgHolder` ArrayList replaced with the element located within the `caesarDup` ArrayList of the same index, thus "encrypting" the character. This is repeated until every character in the `msgHolder` array is encrypted.

The final for-loop is responsible for appending the `msgHolder` array, which now contains the encrypted message, into a string. The `caesarDup` ArrayList is then shifted back to its original position as the standard alphabet.

The `caesarCiphDe` is the reverse method and performs the exact same actions as the method above:

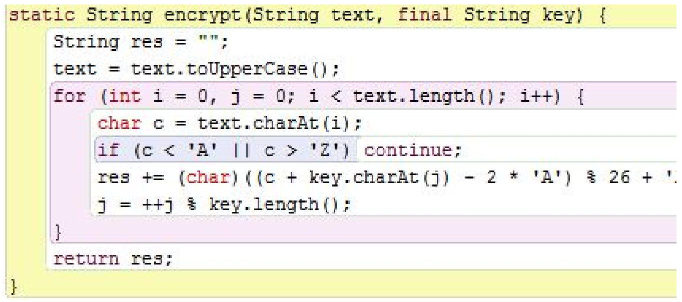
```
for(int i = 0; i<caesar.length; i++)
{
    caesarDup.add(caesar[i]);

}//CREATES AN ARRAYLIST FOR SHIFTING THE CIPHER

int shiftDe = 0;
for( int j = 0; j < caesar.length; j++)
    {

        if(keyDe==(caesar[j]))
        {
            shiftDe = j;
        }
    }//SHIFTER METHOD FOR KEY IN REVERSE

    for(int m = 0; m<enmsg.length(); m++)
{
    msgHolderDe.add(enmsg.charAt(m));
}//CREATES THE STRING ARRAYLIST

 Collections.rotate(caesarDup, -shiftDe);

for(int l = 0; l<msgHolderDe.size(); l++)
{

    for(int k = 0; k<caesarDup.size(); k++)
    {
        if(msgHolderDe.get(l) == caesarDup.get(k))
        {

        msgHolderDe.set( l,caesar[k]);
        break;


    }
    }
}//DECRYPTS THE MESSAGE

StringBuffer deToString = new StringBuffer();
for(char msgH : msgHolderDe){
    deToString.append(msgH);
} //ARRAY to STRING

 System.out.println("Decrypted: " + deToString);
 Collections.rotate(caesarDup, shiftDe);
```

Source Code for Vigenere Cipher Class:

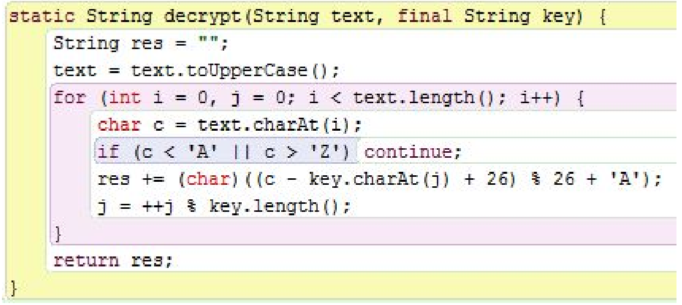
IMPORTANT NOTICE: I do not own this source code completely. The `encrypt` and `decrypt` methods are provided by: https://rosettacode.org/wiki/Vigen%C3%A8re_cipher

```
import java.util.Arrays;
import java.io.*;
import java.util.*;
import java.lang.Object;
import java.util.Collections;
import java.util.ArrayList;

public class VigenereCipher {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter a key phrase: ");
        String key = sc.nextLine();
        key = key.toUpperCase();

        Scanner sc1 = new Scanner(System.in);
        System.out.println("Enter your message: ");
        String oriEn = sc1.nextLine();
        String enc = encrypt(oriEn, key);
        System.out.println("Encrypted: " + enc);

        Scanner sc2 = new Scanner(System.in);
        System.out.println("Enter a key phrase: ");
        String keyDe = sc2.nextLine();
        keyDe = keyDe.toUpperCase();

        Scanner sc3 = new Scanner(System.in);
        System.out.println("Enter your message: ");
        String oriDe = sc3.nextLine();
        String de = decrypt(oriDe, keyDe);
        System.out.println("Decrypted: " + de);
    }

    static String encrypt(String text, final String key) {
        String res = "";
        text = text.toUpperCase();
        for (int i = 0, j = 0; i < text.length(); i++) {
            char c = text.charAt(i);
            if (c < 'A' || c > 'Z') continue;
            res += (char)((c + key.charAt(j) - 2 * 'A') % 26 + 'A');
            j = ++j % key.length();
        }
        return res;
    }

    static String decrypt(String text, final String key) {
        String res = "";
        text = text.toUpperCase();
        for (int i = 0, j = 0; i < text.length(); i++) {
            char c = text.charAt(i);
            if (c < 'A' || c > 'Z') continue;
            res += (char)((c - key.charAt(j) + 26) % 26 + 'A');
            j = ++j % key.length();
        }
        return res;
    }
}
```

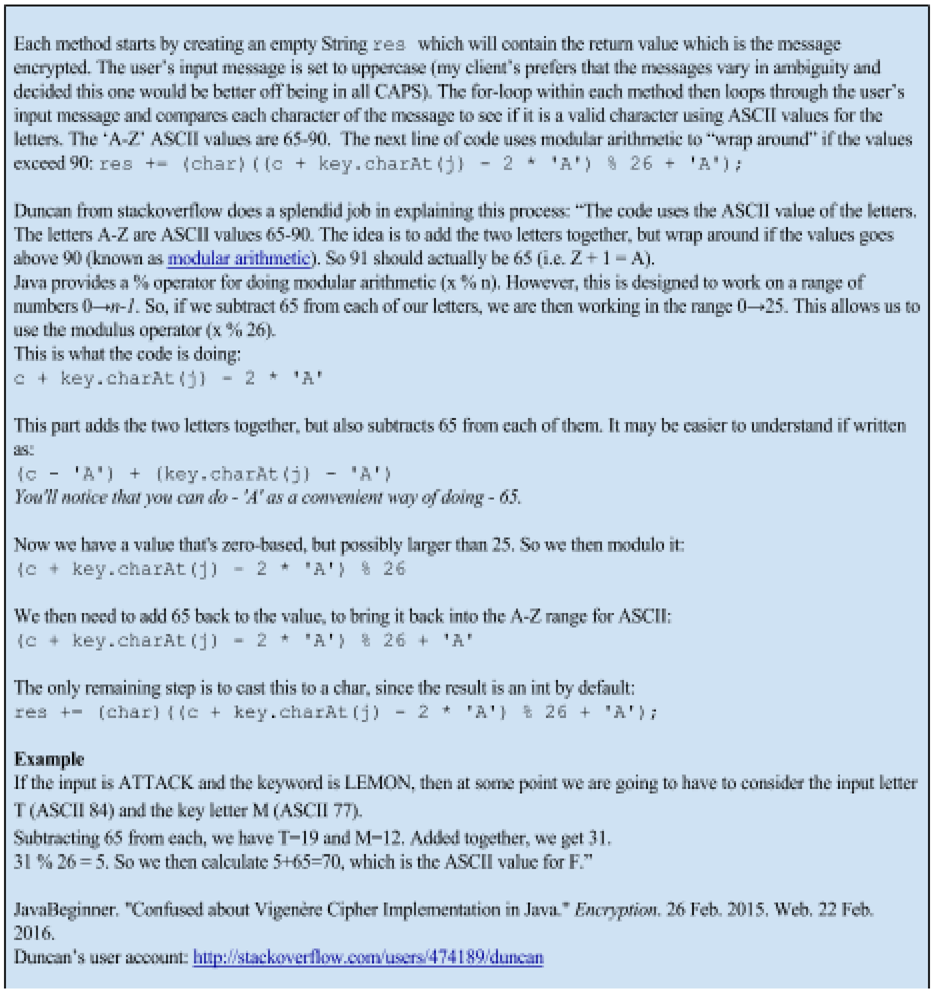
Similar to the Caesar Class, two methods are used: `encrypt` and `decrypt`. The two methods return a String type and accept two String-type parameters: `text` and `key`. The technique used in this method is shown below.



```
static String encrypt(String text, final String key) {
    String res = "";
    text = text.toUpperCase();
    for (int i = 0, j = 0; i < text.length(); i++) {
        char c = text.charAt(i);
        if (c < 'A' || c > 'Z') continue;
        res += (char)((c + key.charAt(j) - 2 * 'A') % 26 + '.
        j = ++j % key.length();
    }
    return res;
}
```



```
static String decrypt(String text, final String key) {
    String res = "";
    text = text.toUpperCase();
    for (int i = 0, j = 0; i < text.length(); i++) {
        char c = text.charAt(i);
        if (c < 'A' || c > 'Z') continue;
        res += (char)((c - key.charAt(j) + 26) % 26 + 'A');
        j = ++j % key.length();
    }
    return res;
}
```



Each method starts by creating an empty String `res` which will contain the return value which is the message encrypted. The user's input message is set to uppercase (my client's prefers that the messages vary in ambiguity and decided this one would be better off being in all CAPS). The for-loop within each method then loops through the user's input message and compares each character of the message to see if it is a valid character using ASCII values for the letters. The 'A-Z' ASCII values are 65-90. The next line of code uses modular arithmetic to "wrap around" if the values exceed 90: `res += (char)((c + key.charAt(j) - 2 * 'A') % 26 + 'A');`

Duncan from stackoverflow does a splendid job in explaining this process: "The code uses the ASCII value of the letters. The letters A-Z are ASCII values 65-90. The idea is to add the two letters together, but wrap around if the values goes above 90 (known as modular arithmetic). So 91 should actually be 65 (i.e. Z + 1 = A).
Java provides a % operator for doing modular arithmetic (x % n). However, this is designed to work on a range of numbers 0→*n-1*. So, if we subtract 65 from each of our letters, we are then working in the range 0→25. This allows us to use the modulus operator (x % 26).
This is what the code is doing:

```
c + key.charAt(j) - 2 * 'A'
```

This part adds the two letters together, but also subtracts 65 from each of them. It may be easier to understand if written as:

```
(c - 'A') + (key.charAt(j) - 'A')
```

You'll notice that you can do - 'A' as a convenient way of doing - 65.

Now we have a value that's zero-based, but possibly larger than 25. So we then modulo it:

```
(c + key.charAt(j) - 2 * 'A') % 26
```

We then need to add 65 back to the value, to bring it back into the A-Z range for ASCII:

```
(c + key.charAt(j) - 2 * 'A') % 26 + 'A'
```

The only remaining step is to cast this to a char, since the result is an int by default:

```
res += (char)((c + key.charAt(j) - 2 * 'A') % 26 + 'A');
```

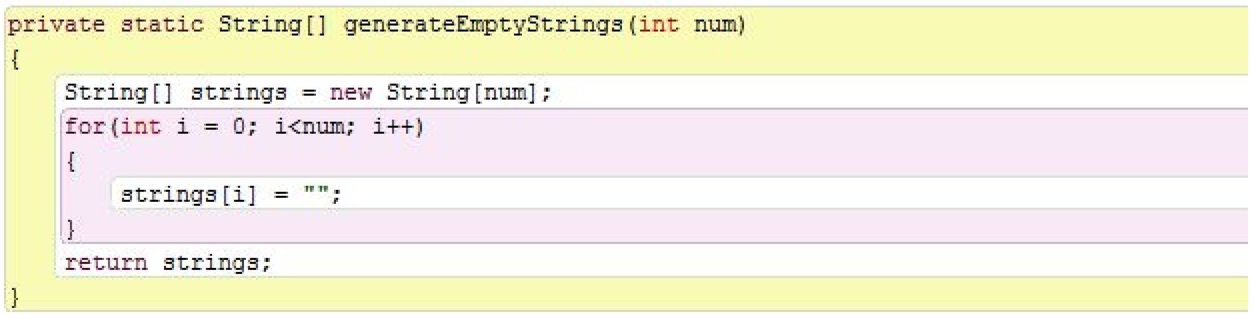
Example
If the input is ATTACK and the keyword is LEMON, then at some point we are going to have to consider the input letter T (ASCII 84) and the key letter M (ASCII 77).
Subtracting 65 from each, we have T=19 and M=12. Added together, we get 31.
31 % 26 = 5. So we then calculate 5+65=70, which is the ASCII value for F."

JavaBeginner. "Confused about Vigenère Cipher Implementation in Java." *Encryption*. 26 Feb. 2015. Web. 22 Feb. 2016.
Duncan's user account: http://stackoverflow.com/users/474189/duncan

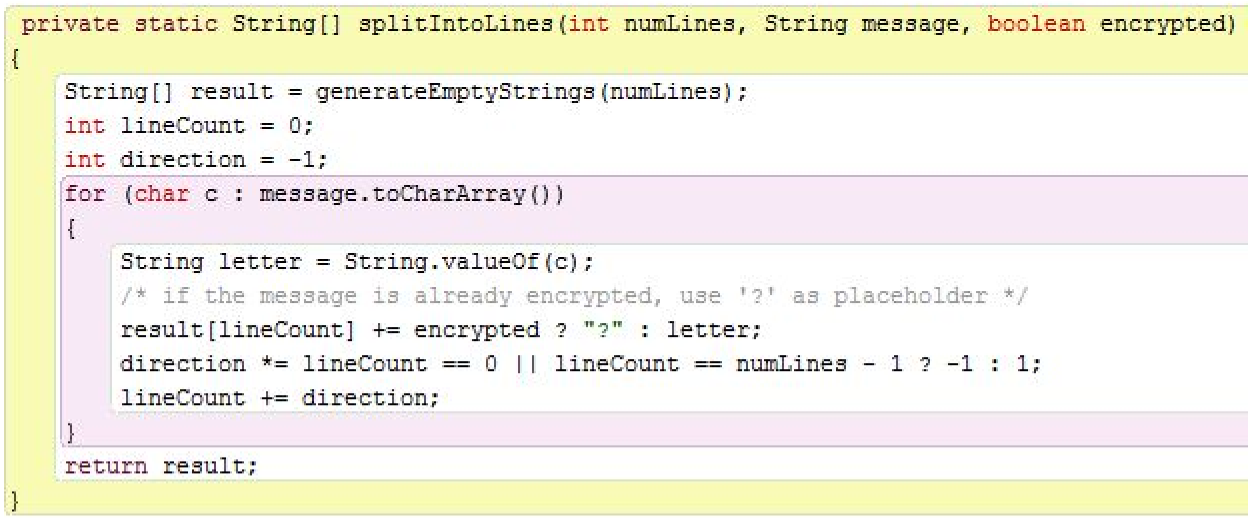
<u>Source Code for ZigZag Class:</u>
IMPORTANT NOTICE: I do not own this source code completely. The methods below except the `main` method belong to **spfy** https://www.reddit.com/user/spfy.

```
import java.util.Arrays;
import java.io.*;
import java.util.*;
import java.lang.Object;
import java.util.Collections;
import java.util.ArrayList;

public class ZigZag
{
    public static void main(String args[])
    {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter how many rows you would like: ");
        int rowsEn = sc.nextInt();

        Scanner sc1 = new Scanner(System.in);
        System.out.println("Enter your message: ");
        String inputEnMes = sc1.nextLine();
        System.out.println(ZigZagEn(rowsEn, inputEnMes));

        Scanner sc2 = new Scanner(System.in);
        System.out.println("Enter the key row number: ");
        int rowsDe = sc2.nextInt();

        Scanner sc3 = new Scanner(System.in);
        System.out.println("Enter your encrypted message: ");
        String inputDeMes = sc3.nextLine();
        System.out.println(ZigZagDe(rowsDe, inputDeMes));

    }

    private static String[] generateEmptyStrings(int num)
    {
        String[] strings = new String[num];
        for(int i = 0; i<num; i++)
        {
            strings[i] = "";
        }
        return strings;
    }

     private static String[] splitIntoLines(int numLines, String message, boolean encrypted)
    {
        String[] result = generateEmptyStrings(numLines);
        int lineCount = 0;
        int direction = -1;
        for (char c : message.toCharArray())
        {
            String letter = String.valueOf(c);
            /* if the message is already encrypted, use '?' as placeholder */
            result[lineCount] += encrypted ? "?" : letter;
            direction *= lineCount == 0 || lineCount == numLines - 1 ? -1 : 1;
            lineCount += direction;
        }
        return result;
    }

    public static String ZigZagEn(int key, String inputMes){

         String[] lines = splitIntoLines(key, inputMes, false);
        String result = "";
        for (String line : lines)
        {
            result += line;
        }
        return result;

    }

    public static String ZigZagDe(int key, String inputMes){

        String[] lines = splitIntoLines(key, inputMes, true);

        /* replace ?s with the actual letter */
        int charCount = 0;
        for (int i = 0; i < key; ++i)
        {
            while (lines[i].contains("?"))
            {
                String letter = String.valueOf(inputMes.charAt(charCount));
                lines[i] = lines[i].replaceFirst("\\?", letter);
                charCount++;
            }
        }

        /* condense zig-zag array into normal string */
        String result = "";
        int lineCount = 0;
        int direction = -1;
        for (int i = 0; i < inputMes.length(); ++i)
        {
            /* Add first letter to result by removing it from the line */
            String letter = String.valueOf(lines[lineCount].charAt(0));
            lines[lineCount] = lines[lineCount].substring(1);
            result += letter;
            direction *= lineCount == 0 || lineCount == key - 1 ? -1 : 1;
            lineCount += direction;
        }
        return result;

    }

}
```

The `ZigZagEn` and `ZigZagDe` methods accept an `key` of an int type, and a String `inputMes`. These methods are then ran into the `splitIntoLines` method which creates a String array `lines`. The `splitIntoLines` method decides if the message is encrypted already or not using a boolean `encrypted`. It then takes the number of lines generated by the `generateEmptyStrings` method and creates multiple String arrays. A for- loop then iterates through the user's input message and separates each character between *n* amount of arrays (*n* representing the number of "rows" generated) and. The `ZigZagEn` method then replaces the empty String result by iterating through each array generated containing the now scrambled message and appending it into a String `result` which is then returned. The first for-loop within the `ZigZagDe` is just used to replace '?' located within an array with a letter.



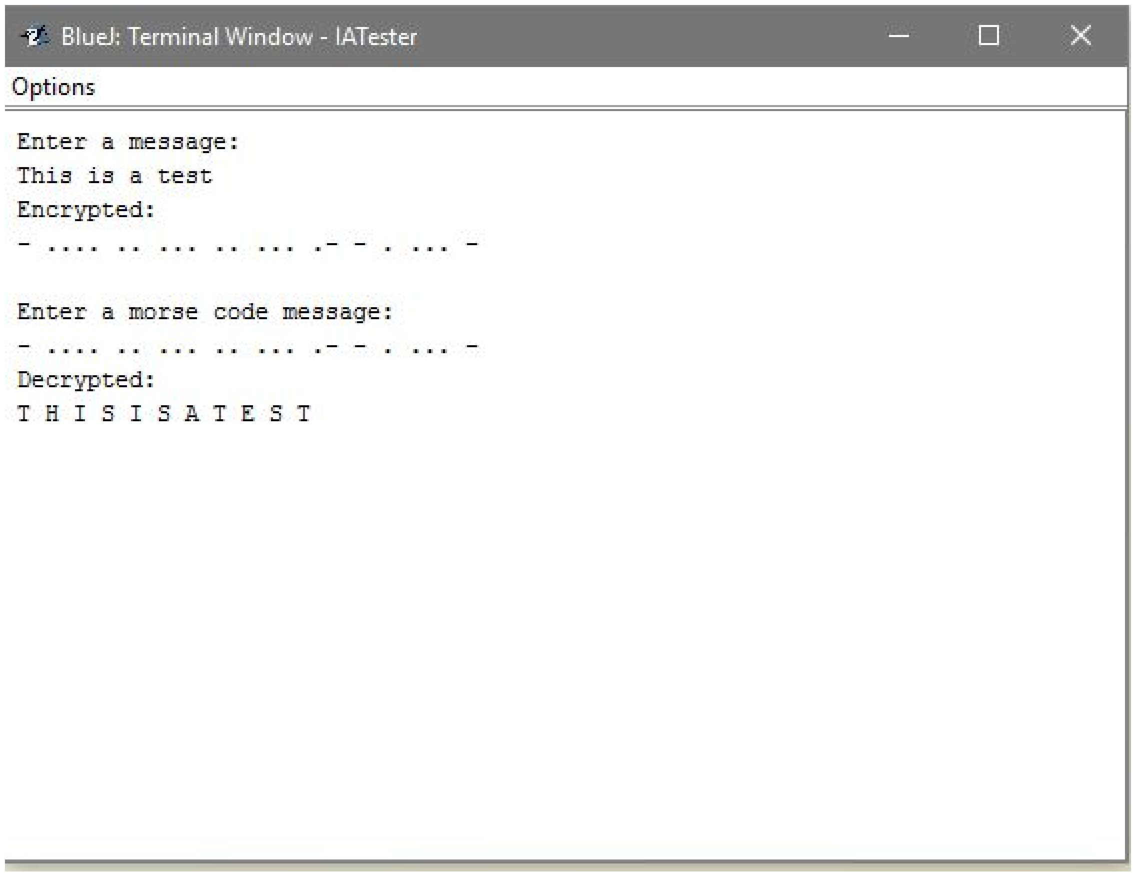
```
private static String[] generateEmptyStrings(int num)
{
    String[] strings = new String[num];
    for(int i = 0; i<num; i++)
    {
        strings[i] = "";
    }
    return strings;
}
```



```
private static String[] splitIntoLines(int numLines, String message, boolean encrypted)
{
    String[] result = generateEmptyStrings(numLines);
    int lineCount = 0;
    int direction = -1;
    for (char c : message.toCharArray())
    {
        String letter = String.valueOf(c);
        /* if the message is already encrypted, use '?' as placeholder */
        result[lineCount] += encrypted ? "?" : letter;
        direction *= lineCount == 0 || lineCount == numLines - 1 ? -1 : 1;
        lineCount += direction;
    }
    return result;
}
```

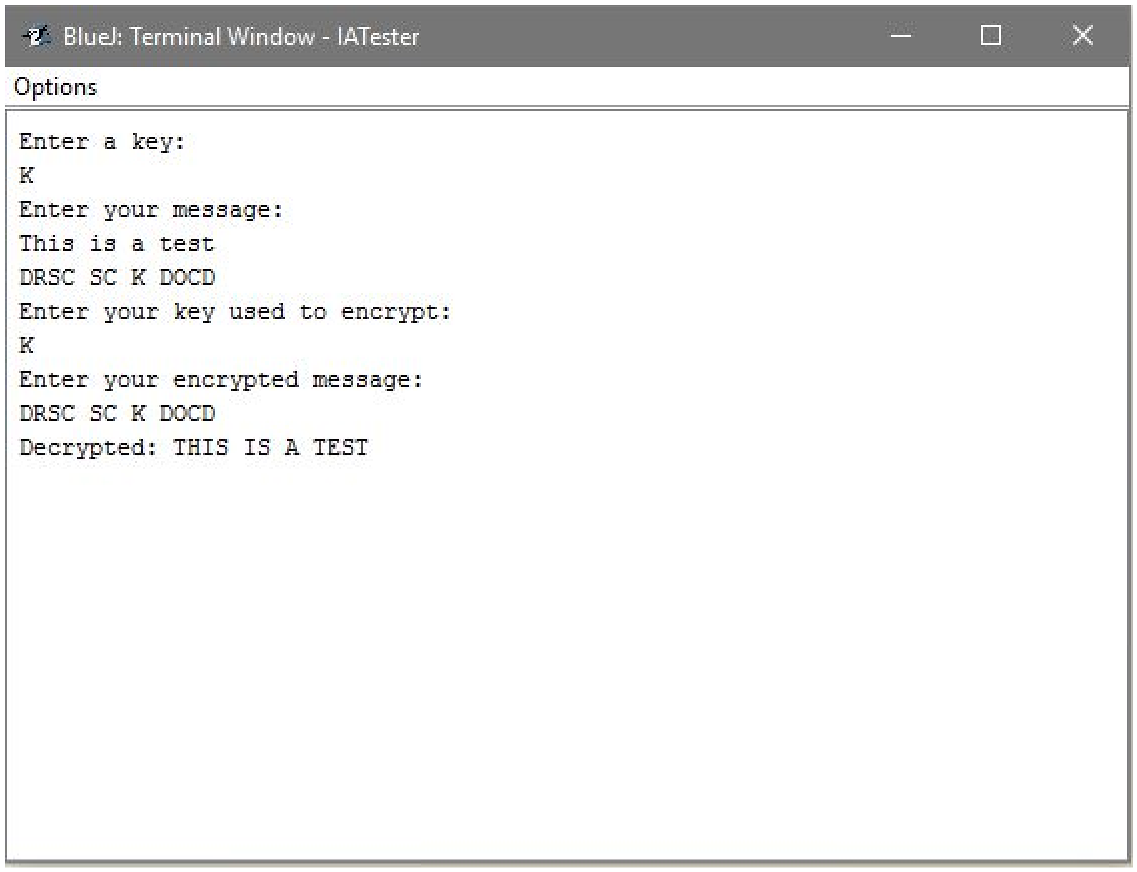
The second for-loop is responsible for decrypting a message by going through the same steps as the `ZigZagEn` method, but instead of printing each element of each array created line by line, the elements are drawn out of each array in order and put into a String `letter` and then printed out as `result`.

Example Morse Code console form:



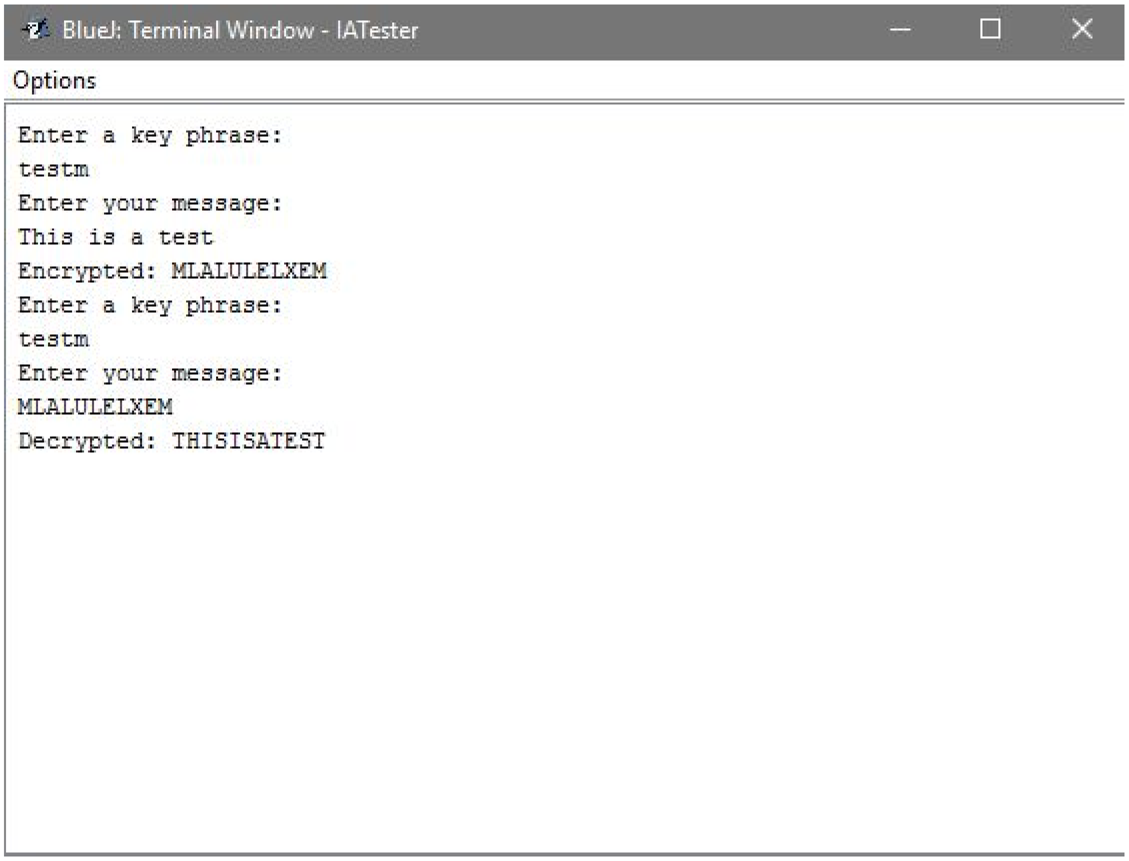
```
Enter a message:
This is a test
Encrypted:
- .... .. ... .. ... .- - . ... -

Enter a morse code message:
- .... .. ... .. ... .- - . ... -
Decrypted:
T H I S I S A T E S T
```

Example Caesar Cipher console form:



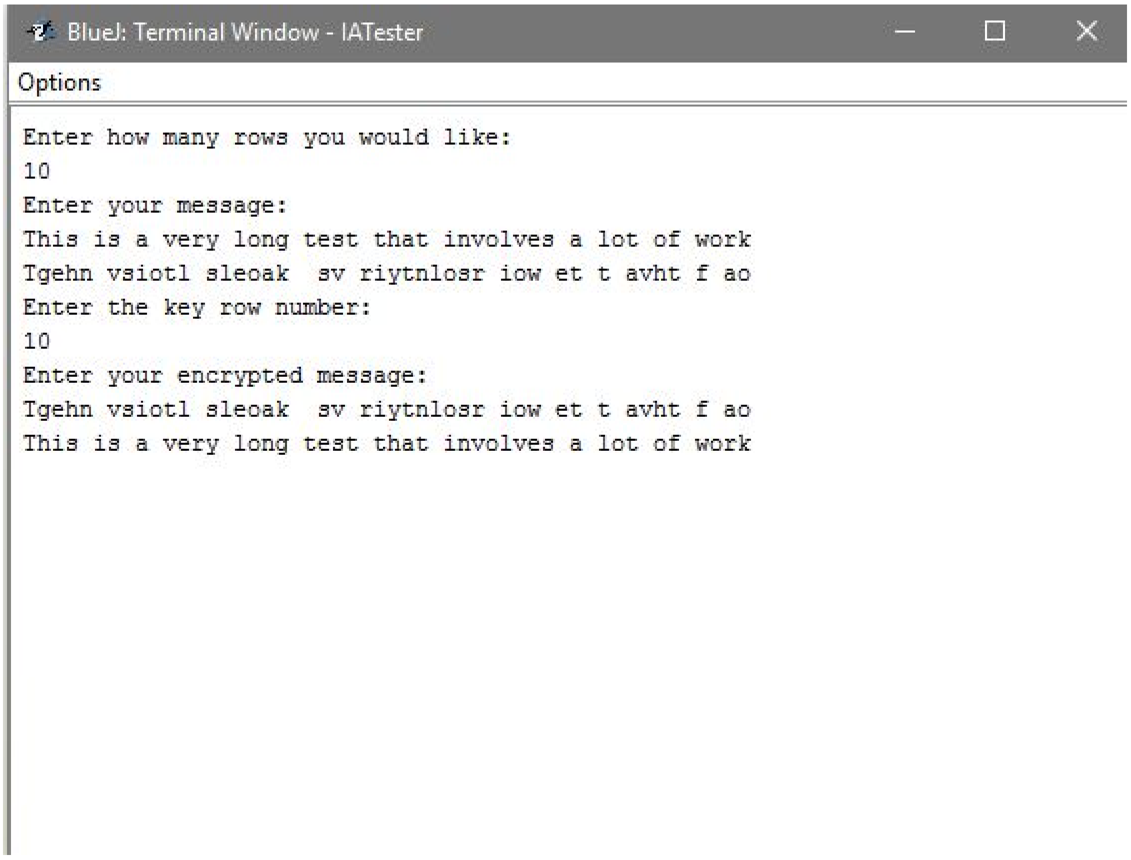
```
Enter a key:
K
Enter your message:
This is a test
DRSC SC K DOCD
Enter your key used to encrypt:
K
Enter your encrypted message:
DRSC SC K DOCD
Decrypted: THIS IS A TEST
```

Example Vigenere Cipher console form:



```
Enter a key phrase:
testm
Enter your message:
This is a test
Encrypted: MLALULELXEM
Enter a key phrase:
testm
Enter your message:
MLALULELXEM
Decrypted: THISISATEST
```

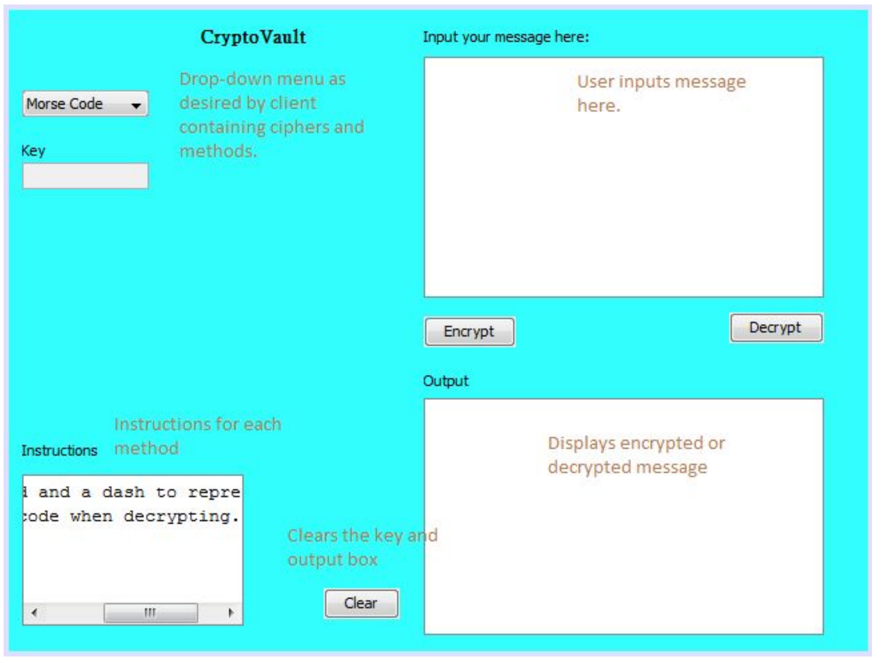
Example of ZigZag Cipher console form:

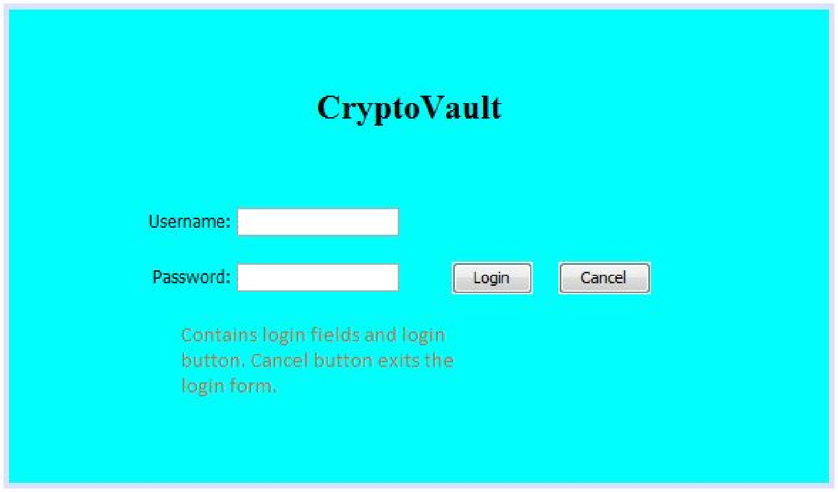


```
Enter how many rows you would like:
10
Enter your message:
This is a very long test that involves a lot of work
Tgehn vsiotl sleoak  sv riytnlosr iow et t avht f ao
Enter the key row number:
10
Enter your encrypted message:
Tgehn vsiotl sleoak  sv riytnlosr iow et t avht f ao
This is a very long test that involves a lot of work
```

Implementing a Graphical User Interface:

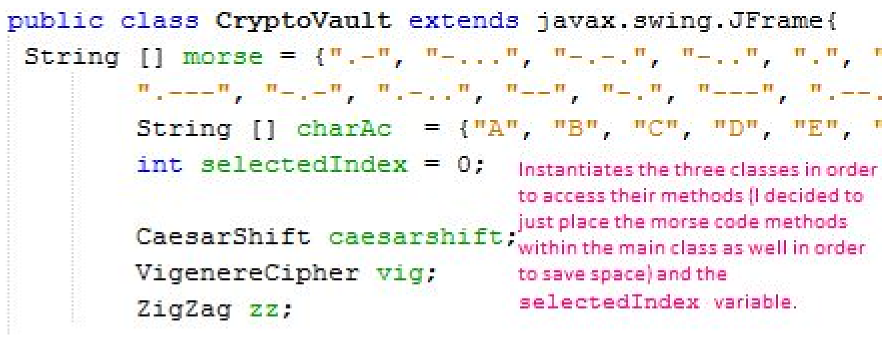
I imported the classes into NetBeans and removed the `main` method from each class leaving just the methods for use since the GUI will act as the interactive device. I then made use of NetBeans form creator to develop the GUI.


CryptoVault
Drop-down menu as desired by client containing ciphers and methods.
Morse Code
Key
Input your message here:
User inputs message here.
Encrypt
Decrypt
Output
Displays encrypted or decrypted message
Instructions for each method
Instructions
d and a dash to repre
code when decrypting.
Clears the key and output box
Clear



Contains login fields and login button. Cancel button exits the login form.

The methods are used in an if-else statement bound to the `encrypt` and `decrypt` button using the an Action Event Listener that checks to see what index, which represents what cipher, the drop-down menu is at currently and sets the integer variable `selectedIndex` to the current index of the drop-down menu using the `.getSelectIndex()` method used for combo boxes(drop-down menus). The `encrypt` or `decrypt` then takes the user input from the first textbox, `inPut` and gets the key from the `key` textbox (used in every cipher except Morse Code), and runs them into either the encrypt method or decrypt method within each cipher class called from the `encrypt` and `decrypt` button.The `ciphersItemStateChanged(java.awt.event.ItemEvent evt)` is what determines the current selected index, as well as displays the instructions for the appropriate ciphers.



```
public class CryptoVault extends javax.swing.JFrame{
 String [] morse = {".-", "-...", "-.-.", "-..", ".", "
          ".---", "-.-", ".-..", "--", "-.", "---", ".--.
          String [] charAc  = {"A", "B", "C", "D", "E", "
          int selectedIndex = 0;

          CaesarShift caesarshift;
          VigenereCipher vig;
          ZigZag zz;
```

Instantiates the three classes in order to access their methods (I decided to just place the morse code methods within the main class as well in order to save space) and the `selectedIndex` variable.

<u>Source Code for the `encrypt` Button:</u>

```
private void EncryptActionPerformed(java.awt.event.ActionEvent evt) {
        if(selectedIndex==0){

        String en = inPut.getText();
        en = en.toUpperCase();
        String output = "";
        for(int x =0; x<en.length(); x++)
        {
            char c = en.charAt(x);
            String convertMor = Character.toString(c);
            for(int j=0; j<charAc.length; j++)
            {
                if(convertMor.equals(charAc[j]))
                {
                    output += morse[j] + " ";
                }
            }
        }
       outPut.setText(output);
        }
        else if(selectedIndex == 1){
            caesarshift = new CaesarShift();
            char keyIn = key.getText().charAt(0);
            keyIn = Character.toUpperCase(keyIn);
            String outPut1 = caesarshift.caesarCiphEn(inPut.getText().toUpperCase(), keyIn);
            outPut.setText(outPut1);
        }
        else if(selectedIndex == 2){
            vig = new VigenereCipher();
            String outPut1 = vig.encrypt(inPut.getText().toUpperCase(),
key.getText().toUpperCase());
            outPut.setText(outPut1);
        }
        else if(selectedIndex == 3)
        {
            zz =  new ZigZag();
            int keyIn = Integer.parseInt(key.getText());
            outPut.setText(zz.ZigZagEn(keyIn, inPut.getText()));
        }
    }
```

The `decrypt` button makes use of the same structure except the methods used are the deciphering methods called from each class.

<u>Source Code for `decrypt` Button:</u>

```
private void DecodeActionPerformed(java.awt.event.ActionEvent evt) {
        if(selectedIndex==0){
        String dec = inPut.getText();
        String [] dec_array = dec.split(" ");
        String output1 = "";
        for(int l = 0; l<dec_array.length; l++)
        {
            for(int p = 0; p<morse.length; p++)
            {
                if(dec_array[l].equals(morse[p]))
                {
                    output1 += charAc[p] + " " ;
                }
            }
        }
        outPut.setText(output1);
        }
        else if(selectedIndex ==1){
            caesarshift = new CaesarShift();
            char keyIn = key.getText().charAt(0);
            keyIn = Character.toUpperCase(keyIn);
            String outPut1 = caesarshift.caesarCiphDe(inPut.getText().toUpperCase(), keyIn);
            outPut.setText(outPut1);
        }
        else if(selectedIndex == 2){
            vig = new VigenereCipher();
            String outPut1 = vig.decrypt(inPut.getText().toUpperCase(),
key.getText().toUpperCase());
            outPut.setText(outPut1);
        }
        else if(selectedIndex == 3){
            zz =  new ZigZag();
            int keyIn = Integer.parseInt(key.getText());
            outPut.setText(zz.ZigZagDe(keyIn, inPut.getText()));
        }


    }
```

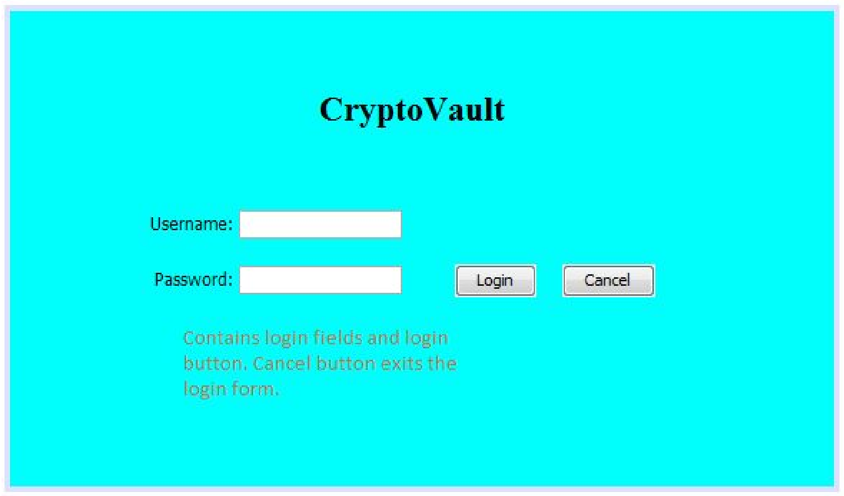
Source Code for `ciphersItemStateChanged(java.awt.event.ItemEvent evt)` method used with the drop-down menu Object `cipher`:

```
 private void ciphersItemStateChanged(java.awt.event.ItemEvent evt) {
        selectedIndex = ciphers.getSelectedIndex();

        if(selectedIndex == 0){
            inStruc.setText("Use a period and a dash to represent \nyour morse code when
decrypting.");
            key.setVisible(false);
            jLabel4.setVisible(false);
        }

        else if (selectedIndex == 1){
            key.setVisible(true);
            jLabel4.setVisible(true);
            inStruc.setText("Enter one letter for a key.");
        }
        else if(selectedIndex == 2){
            key.setVisible(true);
            jLabel4.setVisible(true);
            inStruc.setText("Enter a phrase for a key.");
        }
        else if(selectedIndex == 3){
            key.setVisible(true);
            jLabel4.setVisible(true);
            inStruc.setText("Enter the number of rows for your \nZigZag encryption.");
        }
        else if(selectedIndex == 4){
            key.setVisible(false);
            jLabel4.setVisible(false);
            inStruc.setText("");
        }


    }
```

Source Code for `clear` Button:

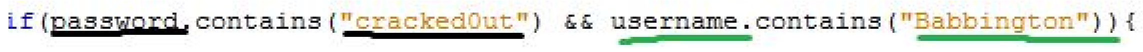
```
    private void clearActionPerformed(java.awt.event.ActionEvent evt) {
        outPut.setText("");
        key.setText("");
    }
```

Login Form:

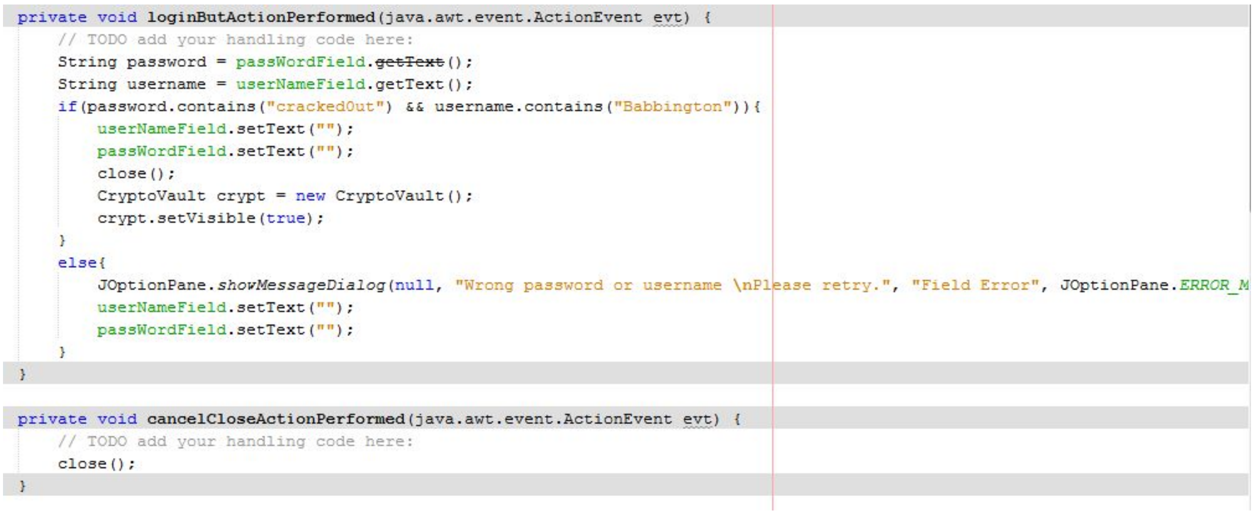


Contains login fields and login button. Cancel button exits the login form.

The username and passwords are "hard-coded" within the source code, meaning they aren't editable as requested by my client. The username is: Babbington. The password is: cracked0ut.

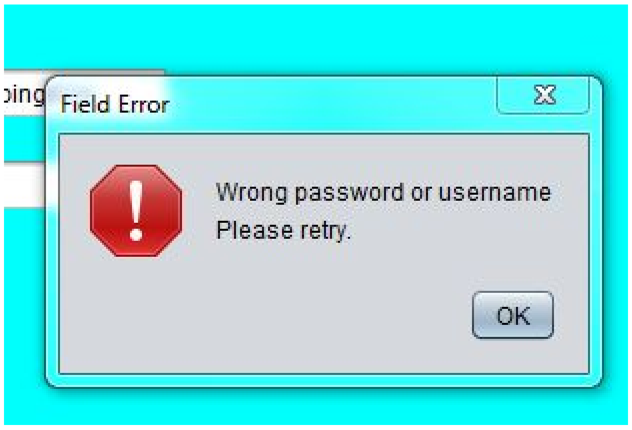


```
if(password.contains("cracked0ut")  && username.contains("Babbington")){
```



```
private void loginButActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    String password = passWordField.~~getText~~();
    String username = userNameField.getText();
    if(password.contains("crackedOut") && username.contains("Babbington")){
        userNameField.setText("");
        passWordField.setText("");
        close();
        CryptoVault crypt = new CryptoVault();
        crypt.setVisible(true);
    }
    else{
        JOptionPane.showMessageDialog(null, "Wrong password or username \nPlease retry.", "Field Error", JOptionPane.ERROR_M
        userNameField.setText("");
        passWordField.setText("");
    }
}

private void cancelCloseActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    close();
}
```

The login button, `loginBut`, works by checking if the `password` and `username` field contain the specific password and username using an if-else statement. If it does then the `CryptoVault` form is set visible loading the main menu. If the password or username is incorrect, an error message is displayed and the fields are cleared.



Field Error

Wrong password or username
Please retry.

Bibliography:

1. Strobl, Roman. "NetBeans Has Compile on Save! (Roumen's Weblog)." *NetBeans Has Compile on Save! (Roumen's Weblog)*. Blog for Roumen, 26 July 2008. Web. 17 Feb. 2016.
2. Vigenere Cipher source code: https://rosettacode.org/wiki/Vigen%C3%A8re_cipher
3. JavaBeginner. "Confused about Vigenère Cipher Implementation in Java." *Encryption*. 26 Feb. 2015. Web. 22 Feb. 2016.
4. Duncan's user account: http://stackoverflow.com/users/474189/duncan
5. Spfy's user account: //www.reddit.com/user/spfy
6. NetBean's website: https://netbeans.org/
7. BlueJ's wesbsite: http://www.bluej.org/