```
/**
 * Programming AE2
 * Creates and shows the cipher GUI
 */
public class AssEx2
{
	/**
	 * The main method
	 * @param args the arguments
	 */
	public static void main(String [] args)
	{
		CipherGUI CipherGUI = new CipherGUI();
		CipherGUI.setVisible(true);
	}
}
```



```
import java.awt.*;

import javax.swing.*;

import java.awt.event.*;
import java.io.*;
import java.util.*;

/**
 * Programming AE2
 * Class to display cipher GUI and listen for events
 */
public class CipherGUI extends JFrame implements ActionListener
{
	//instance variables which are the components
	private JPanel top, bottom, middle;
	private JButton monoButton, vigenereButton;
	private JTextField keyField, messageField;
	private JLabel keyLabel, messageLabel;

	//application instance variables
	//including the 'core' part of the textfile filename
	//some way of indicating whether encoding or decoding is to be done
	private static final int SIZE = 26;
	private static final int KEYWORDCOUNT = 1;
	private static final int WHITESPACE = 2;
	private MonoCipher mcipher;
	private VCipher vcipher;
	private LetterFrequencies letterCount;
	private String inputFileName, outputFileName, inputKeyword;
	private Scanner textIn;
	private char [] coDec = new char [1];
	private char [] outputChars;
	private int [] repeatKeywordChars = new int [SIZE];
	private FileReader textInput;
	private FileWriter textOutput;
	private PrintWriter reportOutput;
	private boolean fileFound = true;
	private boolean codeType = true;

	/**
	 * The constructor adds all the components to the frame
	 */
	public CipherGUI()
	{
		this.setSize(400,150);
		this.setLocation(100,100);
		this.setTitle("Cipher GUI");
		this.setDefaultCloseOperation(EXIT_ON_CLOSE);
		this.layoutComponents();
	}

	/**
	 * Helper method to add components to the frame
	 */
	public void layoutComponents()
	{
		//top panel is yellow and contains a text field of 10 characters
		top = new JPanel();
		top.setBackground(Color.yellow);
		keyLabel = new JLabel("Keyword:");
		top.add(keyLabel);
```

```
                keyField = new JTextField(10);
                top.add(keyField);
                this.add(top,BorderLayout.NORTH);

                //middle panel is yellow and contains a text field of 10 charact
ers
                middle = new JPanel();
                middle.setBackground(Color.yellow);
                messageLabel = new JLabel("Message file : ");
                middle.add(messageLabel);
                messageField = new JTextField(10);
                middle.add(messageField);
                this.add(middle,BorderLayout.CENTER);

                //bottom panel is green and contains 2 buttons

                bottom = new JPanel();
                bottom.setBackground(Color.green);
                //create mono button and add it to the top panel
                monoButton = new JButton("Process Mono Cipher");
                monoButton.addActionListener(this);
                bottom.add(monoButton);
                //create vigenere button and add it to the top panel
                vigenereButton = new JButton("Process Vigenere Cipher");
                vigenereButton.addActionListener(this);
                bottom.add(vigenereButton);
                //add the top panel
                this.add(bottom,BorderLayout.SOUTH);
        }

        /**
         * Listen for and react to button press events
         * (use helper methods below)
         * @param e the event
         */
        public void actionPerformed(ActionEvent e)
        {       if (e.getSource() == monoButton)
                {       monoCipherMethod();
                }
                else if (e.getSource() == vigenereButton)
                {       vCipherMethod();
                }
        }

        /**
         * Helper method to process monocipher
         */
        private void monoCipherMethod()
        {       /*Determine whether to proceed to encoding/decoding
                */
                if (processFileName() && getKeyword())
                {       findFile();
                        if (processFile(false))
                        {       //Prints letter frequencies file if all is well
                                printReportFile();

                        }
                        else //This in the event of I/O operations failing
                        {       System.out.println("Something went wrong..");
                        }
                }
                else
```



```
                {       clearTextFields();
                }
        }

        private void vCipherMethod()
        {       if (processFileName() && getKeyword())
                {       findFile();
                        if (processFile(true))
                        {       //Prints letter frequencies file if all is well
                                printReportFile();
                        }
                        else //This in the event of I/O operations failing
                        {       System.out.println("Something went wrong..");
                        }
        }
                else
                {       clearTextFields();
                }
        }

        /**
         * Obtains cipher keyword
         * If the keyword is invalid, a message is produced
         * @return whether a valid keyword was entered
         */
        private boolean getKeyword()

                //obtain keyword from textfield
        {       inputKeyword = keyField.getText().trim();

                //create integer for use as an index in a while loop
                int index = 0;

                //Create boolean to aid in case checking
                boolean keywordCaseCheck = true;

                /*While loop that checks the case, in turn,
                 * of each character in the keyword using
                 * another boolean helper method and
                 * that there are no repeat characters
                 */
                while (keywordCaseCheck && index < (inputKeyword.length())-1)
                {       if (!checkCase(index))
                        {       //procedure in event of invalid keyword
                                JOptionPane.showMessageDialog(null, "Your Keyword i
s invalid", "Error", JOptionPane.ERROR_MESSAGE);
                                clearTextFields();
                                //reset repeat character array
                                resetKeyword();
                                //set boolean
                                keywordCaseCheck = false;
                                /*Break to prevent infinite loop
                                 * as index is not incremented if
                                 * a lower case letter is found
                                 */
                                break;
                        }
                        else
                        {       keywordCaseCheck = true;
                                index++;
                        }
                }
```

```
                //Test for keyword validity
                if (keywordCaseCheck)
                {       return true;
                }

                else
                {       return false;
                }
        }

        /*Helper method to check keyword
         * is upper case and not a repeat
         * and return a boolean
         */
        private boolean checkCase(int index)
        {       if (inputKeyword.charAt(index) >= 'A' && inputKeyword.charAt(ind
ex) <= 'Z' && keywordRepeats(inputKeyword.charAt(index))) //If keyword is capita
l and not repeated
                {       return true;
                }
                else
                {       return false;

                }

        }

        //Helper method to check repeat keyword chars
                private boolean keywordRepeats(char repeat)
                {       //Integer to store value of incoming char
                        int index = repeat - 'A';

                        //Increment char count to be checked in array
                        repeatKeywordChars[index] = repeatKeywordChars[index] +
KEYWORDCOUNT;
                        //Test to see if the char has been seen before (i.e. ind
ex count is greater than 1)
                        if (repeatKeywordChars[index] > KEYWORDCOUNT)
                        {       return false;
                        }
                        else
                        {       return true;
                        }

                }

        /**
         * Obtains filename from GUI
         * The details of the filename and the type of coding are extracted
         * If the filename is invalid, a message is produced
         * The details obtained from the filename must be remembered
         * @return whether a valid filename was entered
         */
        private boolean processFileName()
        {       //Obtain filename from user input
                inputFileName = messageField.getText().trim();

                /*Truncate and store filename so we can
                 * add the correct letter ('C' or 'D', and 'F')
                 */
                outputFileName = inputFileName.substring(0, (inputFileName.lengt
h())-1);
```



```
                //test if user entered a filename
                if (inputFileName.isEmpty())

                /*give user an error message if no filename was entered
                 * and return false to method that called this method
                 * to prevent further processing
                 */
                {       JOptionPane.showMessageDialog(null, "You did not enter a filenam
e",
                        "Error", JOptionPane.ERROR_MESSAGE);
                        return false;
                }
                else
                {       //Get length of filename to find last letter
                        int fileNameLength = inputFileName.length();

                        //Store last letter of filename
                        String endFileName = "" + inputFileName.charAt(fileNameL
ength-1);

                        //Store filename
                        inputFileName = inputFileName + ".txt";

                /*Check the last letter of the filename
                 * boolean "codeType" is used to determine
                 * whether to encode or decode the input file
                 */

                        if (endFileName.equals("P"))
                        {       codeType = true;
                                return true;
                        }
                        else if (endFileName.equals("C"))
                                {       codeType = false;
                                        return true;
                                }
                                else
                                {       //Error message if filename was not appr
opriate
                                        JOptionPane.showMessageDialog(null, "You
 did not enter a valid filename",
                                                        "Error", JOptionPane.ERRO
R_MESSAGE);
                                        return false;
                                }
                }
        }
        /**
         * Reads the input text file character by character
         * Each character is encoded or decoded as appropriate
         * and written to the output text file
         * @param vigenere whether the encoding is Vigenere (true) or Mono (fals
e)
         * @return whether the I/O operations were successful
         */
        private boolean processFile(boolean vigenere)
        {       /*Conditional test to determine whether to
                 * use mono or Vigenere cipher
                 */

                //Process MonoAlphabetic encryption
```

```
                if (fileFound && vigenere == false && codeType == true)
                {       //Read in text
                        readFileContents();

                        //Instantiate MonoCipher and coDec array
                        outputArray(false);

                        //Pass output filename to helper methods
                        fileOutput("..\\AE2\\"+ outputFileName + "C");
                        reportOutputFile("..\\AE2\\"+ outputFileName + "F");

                        //Instantiate letter counting object
                        letterCount = new LetterFrequencies();

                        //Pass each character to the encode method in mcipher ob
ject
                        try
                        {       for (int index = 0; index < coDec.length; index+
+)
                                {       outputChars[index] =  mcipher.encode(coD
ec[index]);
                                        letterCount.addChar(outputChars[index]);

                                        //Write output to text file
                                        textOutput.write(outputChars[index]);
                                }
                                vigenere = true;
                        }

                        catch (IOException c)
                        {       System.out.println("File not found: " + c);
                                vigenere = false;
                        }
                }

                //Process MonoAlphabetic decryption
                else if (fileFound && vigenere == false && codeType == false)
                {       readFileContents();

                        //Instantiate MonoCipher object and text array
                        outputArray(false);

                        //Pass output filename to helper methods
                        fileOutput("..\\AE2\\"+ outputFileName + "D");
                        reportOutputFile("..\\AE2\\"+ outputFileName + "F");

                        //Instantiate letter counting object
                        letterCount = new LetterFrequencies();

                        //Pass each character to the decode method in mcipher ob
ject
                        try
                        {       for (int index = 0; index < coDec.length; index+
+)
                                {       outputChars[index] =  mcipher.decode(coD
ec[index]);
                                        letterCount.addChar(outputChars[index]);

                                        //Write output to text file
                                        textOutput.write(outputChars[index]);
                                }
```



```
                                vigenere = true;
                        }


                        catch (IOException c)
                        {       System.out.println("File not found: " + c);
                                vigenere = false;
                        }

                }


                //Encrypt plain text using vigenere cipher
                else if (fileFound && vigenere == true && codeType == true)
                {       readFileContents();

                        //Instantiate VCipher object
                        outputArray(true);

                        //Pass output filename to helper methods
                        fileOutput("..\\AE2\\"+ outputFileName + "C");
                        reportOutputFile("..\\AE2\\"+ outputFileName + "F");

                        //Instantiate letter counting object
                        letterCount = new LetterFrequencies();

                        //Pass each character to the encode method in vcipher ob
ject
                        try
                        {       for (int index = 0; index < coDec.length; index+
+)
                                {       outputChars[index] =  vcipher.encode(coD
ec[index]);
                                        letterCount.addChar(outputChars[index]);

                                        //Write output to text file
                                        textOutput.write(outputChars[index]);
                                }
                                vigenere = true;
                        }

                        catch (IOException c)
                        {       System.out.println("File not found: " + c);
                                vigenere = false;
                        }
                }

                //Decrypt cipher text using vigenere cipher
                else if (fileFound && vigenere == true && codeType == false)
                {       readFileContents();

                        //Instantiate VCipher object
                        outputArray(true);


                        //Pass output filename to helper methods
                        fileOutput("..\\AE2\\"+ outputFileName + "D");
                        reportOutputFile("..\\AE2\\"+ outputFileName + "F");

                        letterCount = new LetterFrequencies();

                        //Pass each character to the decode method in vcipher ob
```

```
ject
                        try
                        {       for (int index = 0; index < coDec.length; index+
+)
                                {       outputChars[index] =  vcipher.decode(coD
ec[index]);
                                        letterCount.addChar(outputChars[index]);

                                        //Write output to text file
                                        textOutput.write(outputChars[index]);
                                }

                                vigenere = true;
                        }


                        catch (IOException c)
                        {       System.out.println("File not found: " + c);
                                vigenere = false;
                        }

                }
                        //Reset instance variables to let user try again with va
lid input
                        else if (!fileFound)
                        {       clearTextFields();
                                resetKeyword();
                                resetFileName();
                        }
        return vigenere;
        }



        /*Helper method to open the file and
         * catch file not found exceptions
         */
        private void findFile()

                //Open the requested file
        {       try
                {       textInput = new FileReader("..\\AE2\\"+ inputFileName);
                        fileFound = true;
                }

                /*Catch IOException if file doesn't exist
                 * let user go again
                 */
                catch (IOException e)
                {       JOptionPane.showMessageDialog(null, "No file with that name foun
d, anywhere...", "Error", JOptionPane.ERROR_MESSAGE);
                        System.err.println("File not found!");
                        resetFileName();
                        fileFound = false;
                }



        }

        /*Method to read text contained in file
         * and store as an array of chars.
         */
```



```
        private void readFileContents()
        {       textIn = new Scanner(textInput);

                //While loop to create array
                String inputText = "";
                while (textIn.hasNext())
                {       //Create a string to store the next string from our inpu
t file
                        inputText = textIn.next();

                        /*Create a new array to grow it
                         * by the length of the next string
                         * plus a little bit to account for
                         * whitespace, punctuation and numbers.
                         */
                        char [] coDec2 = new char [inputText.length() + coDec.le
ngth + WHITESPACE];

                        //Copy the contents of the old array to the new one
                        System.arraycopy(coDec, 0, coDec2, 0, coDec.length);

                        //Change pointer of old array to point at new one, old a
rray disappears
                        coDec = coDec2;



                }
                try
                {       /*New filereader from file to read characters
                        *as initial filereader has been exhausted
                        */
                        FileReader charsIn = new FileReader ("..\\AE2\\"+ inputFile
Name);
                        for (int index = 0; index < coDec.length; index++)
                        {       int s = charsIn.read();
                                if (s > -1)
                                {       char t = (char) s;
                                        coDec [index] = t;
                                }
                        }
                        charsIn.close();
                }

                catch (IOException i)
                {       System.err.println("IOException, Dopey" + i);
                }
        }



        /*Helper method to clear textfields in the event
         * of input/output errors or incorrect data entry
         * by user, or any other eventuality
         */
        private void clearTextFields()
        {       messageField.setText("");
                keyField.setText("");
        }

        /*Helper method to instantiate
         * MonoCipher object and an
```

```
         * array to store encoded/decoded characters
         */
        private void outputArray(boolean cipherType)
        {       if (!cipherType)
                {       mcipher = new MonoCipher(inputKeyword);
                }

                else
                {       vcipher = new VCipher(inputKeyword);
                }

                outputChars = new char[coDec.length];
        }


        /*Helper method to process text
         * file output
         */
        private void fileOutput(String fileName)
        {       try
                {       String outFileName = fileName + ".txt";
                        textOutput = new FileWriter(outFileName);
                }
                catch (IOException o)
                {       System.out.println("Cipher/Plain text file output error" + o);

                }

        }

        //Method to process letter frequency report file
        private void reportOutputFile(String reportName)
        {       try
                {       String reportFileName = reportName + ".txt";
                        reportOutput = new PrintWriter(reportFileName);
                }
                catch (IOException r)
                {       System.out.println("Frequency output file error" + r);

                }
        }

        private void printReportFile()
        {       try
                {       textOutput.close();
                        String s = letterCount.getReport();

                        reportOutput.println(s);
                        reportOutput.close();
                        System.exit(0);
                }

                catch (IOException to)
                {       System.out.println("Something went wrong" + to);
                }
        }

        private void resetKeyword()
        {       //reset repeat character array
                int [] resetRepeatKeywords = new int[SIZE];
                repeatKeywordChars = resetRepeatKeywords;
        }
```



```
        private void resetFileName()
        {       inputFileName = "";
                outputFileName = "";

        }
}
```

```
/**
 * Programming AE2
 * Processes report on letter frequencies
 */

public class LetterFrequencies
{
        /** Size of the alphabet */
        private final int SIZE = 26;

        /** Count for each letter */
        private int [] alphaCounts;

        //Frequencies for each letter
        private double [] charFrequencies;

        /** The alphabet */
        private char [] alphabet;


        /** Average frequency counts */
        private double [] avgCounts = {8.2, 1.5, 2.8, 4.3, 12.7, 2.2, 2.0, 6.1,
7.0,
                                                                0.2, 0.8, 4.0, 2.
4, 6.7, 7.5, 1.9, 0.1, 6.0,
                                                                    6.3, 9.1, 2.8
, 1.0, 2.4, 0.2, 2.0, 0.1};

        /** Character that occurs most frequently */
        private char maxCh;

        /** Total number of characters encrypted/decrypted */
        private int totChars, inputChar, maxFreqIndex;

        //Used to create report string in reportColumns method
        private String columns = "";


        /**
         * Instantiates a new letterFrequencies object.
         */
        public LetterFrequencies()
        {       /*initialise array to store letter frequencies
                *default for int arrays is to set values at all indices
                *to 0, so we dont need to do anything else
                */
                alphaCounts = new int [SIZE];

                //start count of total characters
                totChars = 0;

                //Initialise alphabet array (copied from MonoCipher...)
                alphabet = new char [SIZE];
                for (int i = 0; i < SIZE; i++)
                        alphabet[i] = (char)('A' + i);

        }

        /**
         * Increases frequency details for given character
         * @param ch the character just read
         */
```



```
        public void addChar(char ch)
        {       //store incoming char in instance variable
           if (ch >= 'A' && ch <= 'Z')
           {    inputChar = ch- 'A';

            //Call method to increment alphacount array
            countLetter(inputChar);
           }
        }

        /*Method to count individual
         * and total alphabetic character
         * frequencies
         */
        private void countLetter(int indexValue)
                //Count individual character frequencies
        {       alphaCounts[indexValue] = alphaCounts[indexValue] + 1;

                //Count total characters
                totChars++;
        }

        /**
         * Gets the maximum frequency
         * @return the maximum frequency
         */
        private double getMaxPC()
    {   double maxPercent = charFrequencies[maxFreqIndex];
            return maxPercent;
        }

        /**
         * Returns a String consisting of the full frequency report
         * @return the report
         */
        //As above...
        public String getReport()
        {       inputCharFrequencies();
                mostFrequent();
                reportColumns();
                String report = "" + String.format("LETTER ANALYSIS%n%n") + "Lett
er " +
                                        "Freq " + "Freq% " + "AvgFreq% " + String.format("
Diff%n") +
                                        columns + "The most frequent letter is " + maxCh +
                                        " with a percent frequency of " + String.format("%4.1f", g
etMaxPC());

            return report;
        }

        //Calculate and store character frequencies
        private void inputCharFrequencies()
        {       charFrequencies = new double [SIZE];
                for (int index = 0; index < SIZE; index++)
                {       //Calculate and store percent frequencies of input chars
                        charFrequencies[index] =  (alphaCounts[index]/(double) t
otChars)*100;
                }

        }
```

```
        /*Calculate most frequent character
         * and store its index
         */
        private void mostFrequent()
        {       //Initialise most frequent char variable
                maxCh = alphabet[0];

                /*Integer to store index
                 * of most frequent char so far
                 */
                int maxIndex = 0;

                /*Initialise integer to store index
                 * of most frequent char to use in getMaxPC method
                 */
                maxFreqIndex = 0;

                //Loop to find most frequent char and its index
                for (int i = 0; i < SIZE; i++)
                {       if (alphaCounts[i] > maxIndex)
                        {       maxCh = alphabet[i];
                                maxIndex = alphaCounts[i];
                                maxFreqIndex = i;
                        }
                }

        }

        /*Create formatted string with letters
         * and frequencies for report
         */
        private void reportColumns()
        {       //String builder for appending strings of array values
                StringBuilder letterData = new StringBuilder();
                for (int i = 0; i < SIZE; i++)
                {       //Build string iteratively so columns can be created
                        columns = String.format("" + letterData.append(" " + al
phabet[i] + "  " + String.format("%4d", alphaCounts[i]) + "  "
                                +String.format("%4.1f",charFrequencies[i]) + "
" + "" + avgCounts[i] + "  " +
                                String.format("%4.1f", (avgCounts[i])-charFreque
ncies[i]) + "%n"));

                }

        }
}
```



```
/**
 * Programming AE2
 * Contains monoalphabetic cipher and methods to encode and decode a character.
 */

//import java.util.*;
public class MonoCipher
{
        /** The size of the alphabet. */
        private final int SIZE = 26;

        /** The alphabet. */
        private char [] alphabet;

        /** The cipher array. */
        private char [] cipher;

        //More instance variables
        private String keywordIn;
        private int cipherIndex, alphaIndex, cIndex;
        private char encodeLetter, decodedLetter;


        /**
         * Instantiates a new mono cipher.
         * @param keyword the cipher keyword
         */
        public MonoCipher(String keyword)
        {
                //create alphabet
                alphabet = new char [SIZE];
                for (int i = 0; i < SIZE; i++)
                        alphabet[i] = (char)('A' + i);

                // create first part of cipher from keyword
                cipher = new char [SIZE];
                keywordIn = keyword.trim();
                cipherIndex = 0;
                alphaIndex = (SIZE-1);
                /*Loop to insert cipher characters into
                 * first part of array
                 */
                try
                {       //Set integer for maximum iterations in while loop
                        int len = keywordIn.length();

                        //Loop to insert keyword characters into cipher array
                        while (cipherIndex < len)
                        {       cipher[cipherIndex] = keywordIn.charAt(cipherInd
ex);
                                cipherIndex++;
                        }

                }
                catch (StringIndexOutOfBoundsException s)
                {       System.err.println("StringIndexEXCEPTION: " + cipherIndex);
                }
                /* create remainder of cipher from the remaining characters of t
he alphabet
                */
                try
                {       for (int counter = SIZE-1; counter >= 0; counter--)
```

```
                        {       //Start at 'Z'
                                char nextChar = (char) ('A' + counter);

                                //Search the keyword for the alphabetic characte
r
                                if (keywordIn.indexOf(nextChar) == -1)//Replace
if not found
                                {       cipher[cipherIndex] = nextChar;
                                        cipherIndex++;
                                }

                        }

                }
                catch (ArrayIndexOutOfBoundsException a)
                { System.err.println(a);
                }

                // print cipher array for testing and tutors
                System.out.println(alphabet);
                System.out.println(cipher);
        }



        /**
         * Encode a character
         * @param ch the character to be encoded
         * @return the encoded character
         */
        public char encode(char ch)
        {       //Store character to be encoded
                encodeLetter = ch;

                //Find the index of the character in the alphabet
                        if (encodeLetter >= 'A' && encodeLetter <='Z')
                        {       decodedLetter = cipher[encodeLetter - 'A'];
                                ch = decodedLetter;
                        }
                        /*If not an upper case letter
                         * return original value as must be
                         * punctuation or non-alphabet character
                         */
                        else
                        {       ch = encodeLetter;
                        }

                return ch;
}

        /**
         * Decode a character
         * @param ch the character to be encoded
         * @return the decoded character
         */
        public char decode(char ch)
        {       //Store character to be encoded
                encodeLetter = ch;

                //Find the index of the character in the alphabet
                boolean encodeCharFound = false;
```



```
                alphaIndex = 0;
                while (!encodeCharFound && alphaIndex < SIZE)
                {       if (encodeLetter == cipher[alphaIndex])
                        {       encodeCharFound = true;
                                decodedLetter = alphabet[alphaIndex];
                                ch = decodedLetter;
                        }
                        else
                        {       alphaIndex++;
                        }
            }
                return ch;  // replace with your code
        }


}
```

```
import java.util.Arrays;

/**
 * Programming AE2
 * Class contains Vigenere cipher and methods to encode and decode a character
 */
public class VCipher
{
        private char [] alphabet;   //the letters of the alphabet
        private final int SIZE = 26;

    // more instance variables
        private char [][] vcipher;
        private String cipherKey;
        private char encodeLetter, decodedLetter;

        /*Variable to keep track of row position in cipher array
         * will be set to 0 when object is instantiated
         */
        private int cipherArrayRow = 0;
        /*Integer to store length of keyword
         * needs to be available for multiple methods
         */
        private int rowsNo = 0;

        /**
         * The constructor generates the cipher
         * @param keyword the cipher keyword
         */
        public VCipher(String keyword)
        {       //Store keyword
                cipherKey = keyword;

                //Create alphabet array
                alphabet = new char [SIZE];
                for (int aIndex = 0; aIndex < SIZE; aIndex++)
                {       alphabet[aIndex] = (char)('A' + aIndex);
                }

                /*Create 2D array with
                 * keyword length number of rows
                 * and 26 columns
                 */

                //Store length of keyword
                rowsNo = cipherKey.length();

                //Initialise cipher array
                vcipher = new char [rowsNo][SIZE];

                //Loop to create each new row in cipher array
                for (int rowIndex = 0; rowIndex < rowsNo; rowIndex++)
                {       //set the first char of each row
                        vcipher [rowIndex][0] = cipherKey.charAt(rowIndex);


                        //index to keep track of cipher characters
                        int alphaIndex =  alphabet[(cipherKey.charAt(rowIndex)+1
) - 'A'];


                        //Then loop through the remaining columns in each row to
 set the remaining alphabetic characters
                        for (int colIndex = 1; colIndex < SIZE; colIndex++)
```



```
                        {       vcipher[rowIndex][colIndex] = alphabet[((alphaIn
dex)-'A')];
                                alphaIndex++;


                                //Test to see if character is at Z yet (otherwis
e increment character index)
                                if ((alphaIndex -'A') > ('Z' - 'A'))
                                        {       alphaIndex = 'A';
                                        }
                        }
                }

                //TRYING TO PRINT ARRAYS.....
                for (int i = 0; i < rowsNo; i++)
                {       System.out.println(Arrays.toString(vcipher[i]));
                }
                System.out.println(Arrays.toString(alphabet));
        }
        /**
         * Encode a character
         * @param ch the character to be encoded
         * @return the encoded character
         */
        public char encode(char ch)
        {       //Store character to be encoded
                encodeLetter = ch;

                        //Test to determine if char is alphabetic
                        if (encodeLetter >= 'A' && encodeLetter <='Z')
                        {       //Find the index of the character in the alphabe
t
                                int cipherArrayCol = encodeLetter - 'A';


                                //Test to see if we have reached the end of the
cipher array rows
                                if (cipherArrayRow >= rowsNo) //Reached the last
 row, so start at row 0
                                {
                                        cipherArrayRow = 0;
                                        decodedLetter = vcipher[cipherArrayRow][
cipherArrayCol];
                                        ch = decodedLetter;
                                        cipherArrayRow++;
                                }
                                else
                                {       decodedLetter = vcipher[cipherArrayRow][
cipherArrayCol];
                                        cipherArrayRow++;
                                        ch = decodedLetter;
                                }
                        }

                        /*If not an upper case letter
                         * return original value as must be
                         * punctuation, lower case, or non-alphabet character
                         */
                        else
                        {       ch = encodeLetter;
                        }

            return ch;
        }
```

```
        /**
         * Decode a character
         * @param ch the character to be decoded
         * @return the decoded character
         */
        public char decode(char ch)
        {       //Store character to be decoded
                encodeLetter = ch;

                //Test to determine if char is alphabetic
                if (encodeLetter >= 'A' && encodeLetter <='Z')
                {       //Search each column in the correct row for the input ch
ar

                        boolean encodeCharFound = false;
                        int alphaIndex = 0;
                        while (!encodeCharFound)
                                {       //search for char in cipher
                                        if (encodeLetter == vcipher[cipherArrayR
ow][alphaIndex])
                                        {       encodeCharFound = true;


                                                //read decoded char from the alp
habet
                                                decodedLetter = alphabet[alphaIn
dex];
                                                ch = decodedLetter;
                                                cipherArrayRow++;
                                        }
                                        else
                                        {       alphaIndex++;
                                        }
                                }
                }

                //Test to determine whether cipher rows have been exhausted
                if (cipherArrayRow == rowsNo)
                {       cipherArrayRow = 0;
                }

            return ch;
        }
}
```