1 AssEx2

```java
/**
 * 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);
  }
}
```

2 CipherGUI

```java
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import java.io.*;
import java.util.Scanner;

/**
 * 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 MonoCipher mcipher;
  private VCipher vcipher;

  //Keyword, filename and last letter of filename are variables that must be accessible by
      multiple methods, so are declared here.
  private String key;
  private String fileIn;
  private String fileOut;
  private char ender;
  /**
   * 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 characters
  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(getKeyword() && processFileName()) {
    if (e.getSource() == monoButton) {
      boolean success = processFile(false);
    }
    else if(e.getSource() == vigenereButton) {
      boolean success = processFile(true);
    }
    System.exit(0);
  }
}


/**
 * Obtains cipher keyword
 * If the keyword is invalid, a message is produced
 * @return whether a valid keyword was entered
 */
private boolean getKeyword()
{
  key = keyField.getText();
  if(key.isEmpty() || !key.equals(key.toUpperCase()) || !checkRepeated(key) ) {
    JOptionPane.showMessageDialog(null, "Require a valid keyword");
    keyField.setText("");
    return false;
  }
  return true;
}

private boolean checkRepeated(String keyword) {
  char [] characters = new char[keyword.length()];
  for(int i=0; i < keyword.length(); i++) {
    characters[i] =keyword.charAt(i);
    //System.out.println(keyword.charAt(i));
    for(int j=0; j<i; j++) {
      if(characters[i] == characters[j]) {
        return false;
      }
    }
  }
  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()
{
  fileIn = messageField.getText();
  ender = fileIn.charAt(fileIn.length() - 1);
  fileOut = fileIn.substring(0, fileIn.length()-1);
  if(ender == 'P') {
    fileOut+="C";
    return true;
  }
  if(ender == 'C') {
    fileOut+="D";
    return true;
  }
  else{
    JOptionPane.showMessageDialog(null, "Require a P or C file excluding .txt extension.");
    messageField.setText("");
    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 (false)
 * @return whether the I/O operations were successful
 */
private boolean processFile(boolean vigenere)
{
  try {
    FileReader readFile = new FileReader(fileIn+".txt");
    Scanner scan = new Scanner(readFile);
    String readLine;
    FileWriter writeFile = new FileWriter(fileOut+".txt");
    writeFile.flush();
    LetterFrequencies freqCounter = new LetterFrequencies();
    if(!vigenere) {
      mcipher = new MonoCipher(key);
    }
    else {
      vcipher = new VCipher(key);
    }
    while(scan.hasNextLine()) {
      readLine = scan.nextLine();
      char[] charsIN = new char[readLine.length()];
      char[] charsOUT = new char[readLine.length()];
      StringBuilder outstring = new StringBuilder();
      if (!vigenere) {
        for(int i=0; i<readLine.length(); i++) {
          charsIN[i]= readLine.charAt(i);
          if (ender == 'P') {
            charsOUT[i]=mcipher.encode(charsIN[i]);
          }
          else if(ender == 'C') {
            charsOUT[i]=mcipher.decode(charsIN[i]);
          }
          freqCounter.addChar(charsOUT[i]);
          outstring.append(charsOUT[i]);
        }
        //System.out.println(readLine);
        //System.out.println(outstring);
        writeFile.write(outstring+"\n");
      }

      else {
        for(int i=0; i<readLine.length(); i++) {
          charsIN[i]= readLine.charAt(i);
          if (ender == 'P') {
            charsOUT[i]=vcipher.encode(charsIN[i]);
          }
          else if(ender == 'C') {
            charsOUT[i]=vcipher.decode(charsIN[i]);
          }
```

```
            freqCounter.addChar(charsOUT[i]);
            outstring.append(charsOUT[i]);
          }
          //System.out.println(readLine);
          System.err.println(outstring);
          writeFile.write(outstring+"\n");
        }

      }
      FileWriter freqReport = new FileWriter((fileIn.substring(0, fileIn.length()-1))+"F.txt");
      freqReport.write(freqCounter.getReport());
      writeFile.close();
      freqReport.close();
    } catch (FileNotFoundException fnf) {
      JOptionPane.showMessageDialog(null, "File Could Not Be Found. Make sure it is in src
           directory.");
      messageField.setText("");
      return false;
    } catch (IOException e) {
      e.printStackTrace();
    }

    return true;

  }
}
```

3 Letter Frequencies

```
/**
 * 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 = new int[SIZE];

  /** The alphabet */
  private char [] alphabet = new char[SIZE];

  /** 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;

  /**
 * Instantiates a new letterFrequencies object.
 */
public LetterFrequencies()
{
  totChars=0;
  for(int i=0; i<SIZE;i++) {
    alphabet[i]=(char)( 'A' + i );
    alphaCounts[i]=0;
  }
}

/**
 * Increases frequency details for given character
 * @param ch the character just read
 */
public void addChar(char ch)
{
  totChars++;
  for(int i=0; i< SIZE; i++) {
    if(ch == alphabet[i]) {
      alphaCounts[i]++;
    }
  }
}

/**
 * Gets the maximum frequency
 * @return the maximum frequency
 */
private double getMaxPC()
{
  double Max = 0;
  for(int i =0; i<SIZE; i++) {
    if(alphaCounts[i] > Max) {
      Max = alphaCounts[i];
      maxCh = alphabet[i];
    }
  }
  return Max;
}

/**
 * Returns a String consisting of the full frequency report
 * @return the report
 */
public String getReport()
{
  String output = new String("LETTER ANALYSIS \r\n\r\n");
  output+=String.format("%-20s %-20s %-20s %-20s %-20s \r\n", "Letter", "Freq", "Freq%",
      "AvgFreq%", "Diff");
  double[] freqP = new double[SIZE];
  for(int i=0; i<SIZE;i++) {
    freqP[i] = 100*(alphaCounts[i])/(double)totChars;
    double diff = freqP[i] - avgCounts[i];
    output+=String.format("%-20c %-20d %-20.1f %-20.1f %-20.1f \r\n", alphabet[i],
        alphaCounts[i], freqP[i], avgCounts[i], diff);
  }
  double maxFq = this.getMaxPC();
  output+=String.format("\r\nThe most frequent letter is '%c' at %2.1f%%", maxCh,maxFq);
```

```
    return output;
  }
}
```

4 Monoalphabetic Cipher

```
/**
 * Programming AE2
 * Contains monoalphabetic cipher and methods to encode and decode a character.
 */
public class MonoCipher
{
  /** The size of the alphabet. */
  private final int SIZE = 26;

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

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

  /**
   * 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
    // create remainder of cipher from the remaining characters of the alphabet
    // print cipher array for testing and tutors
    cipher = new char [SIZE];
    int skip = 0;
    for(int i=0;i<SIZE;i++) {
      if(i<keyword.length()) {
        cipher[i] = keyword.charAt(i);
      }
      else {
        cipher[i] = (char)('Z'- (i-(keyword.length()-skip)));
        for( int k = 0; k < keyword.length(); k++) {
          for( int j = 0; j < keyword.length(); j++ ) {
            if (cipher[i] == keyword.charAt(j)) {
              ++skip;
              cipher[i] = (char)('Z'- (i-(keyword.length()-skip)));
              //System.out.println(skip);
            }
          }
        }
      }
      System.err.println(cipher[i]);
    }
  }

  /**
   * Encode a character
   * @param ch the character to be encoded
   * @return the encoded character
   */
  public char encode(char ch)
  {
    for (int i = 0; i < SIZE; i++) {
      if(ch == alphabet[i]) {
        return cipher[i];
      }
    }
    //System.err.println("unexpected character");
    return ch;
  }

  /**
   * Decode a character
   * @param ch the character to be encoded
   * @return the decoded character
   */
  public char decode(char ch)
  {
    for (int i = 0; i < SIZE; i++) {
      if(ch == cipher[i]) {
        return alphabet[i];
      }
    }
    //System.err.println("unexpected character");
    return ch;
  }
}
```

5 Vigenere Cipher

```
/**
 * 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 [][] cipher;
  private int keyLength;
  private int encodeCounter;
  private int decodeCounter;

  /**
   * The constructor generates the cipher
   * @param keyword the cipher keyword
   */
  public VCipher(String keyword)
```

```java
{
  encodeCounter =0;
  decodeCounter =0;
  String checkString = "";
  alphabet = new char [SIZE];
  for (int i = 0; i < SIZE; i++) {
    alphabet[i] = (char)('A' + i);
  }
  keyLength=keyword.length();
  cipher = new char [SIZE][keyLength]; //keyLength redundant here but necessary for encoding.
  for(int i=0;i<keyword.length();i++) {
    for (int j=0;j<SIZE;j++) {
      if(j==0) {
        cipher[j][i] = keyword.charAt(i);
      }
      else {
        if (cipher[0][i]+j <= 'Z') {
        cipher[j][i] = (char) (cipher[0][i]+j);
        }
        else {
          int arrayPositionZ=0;
          for(int k=0;k<j;k++) {
            if(cipher[k][i] == 'Z')
              arrayPositionZ = k;
          }
          cipher[j][i] = (char) ('A'+j-(arrayPositionZ+1));
        }
      }
      checkString+=cipher[j][i]+" ";
    }
    checkString+="\n";
  }
  System.out.println(checkString);
}
/**
 * Encode a character
 * @param ch the character to be encoded
 * @return the encoded character
 */
public char encode(char ch)
{
  if( encodeCounter >= keyLength ) {
    encodeCounter =0;
  }
  for (int i = 0; i < SIZE; i++) {
    if(ch == alphabet[i]) {
      return cipher[i][encodeCounter++];
    }
  }
  //System.err.println("unexpected character");
  return ch;

}

/**
 * Decode a character
 * @param ch the character to be decoded
 * @return the decoded character
 */
```

```
  public char decode(char ch)
  {
    if( decodeCounter >= keyLength ) {
      decodeCounter =0;
    }
    for (int i = 0; i < SIZE; i++) {
      if(ch == cipher[i][decodeCounter]) {
        decodeCounter++;
        return alphabet[i];
      }
    }
    //System.err.println("unexpected character");
    return ch;
  }
}
```