Lance Baker (c3128034)

INFT3940 I.T. Applications

Assignment 2 Cryptography

Contents

Introduction

Feature Overview

1. **Scalable Letter Definitions**
   The system comprises the capability to have a language with any number of characters/ with any ordering of those characters, also supporting umlauts and foreign symbols. The languages letter frequencies corresponding to each letter are also stored, enabling the ability to detect when the message has been successfully decrypted.

2. **Recursive Keyword Decryption**
   The system allows a keyword file to be loaded, enabling the file contents to be stored in memory. The keywords in memory are used to decrypt the loaded cipher text file for each keyword, which is used in conjunction with character frequency analysis in order to determine the most probable keyword (being a sorted structure of the decrypted messages).

3. **Automatic Keyword Deducing**
   The system can automatically deduce possible used keywords from the encrypted cipher text in the circumstance that the keyword file was not supplied, which is accomplished using the Kasiski method (for deriving the key length) and through Interwoven Caesar Shifts partnered with character analysis. The possible keywords are then loaded into the system and the decryption process commences, decrypting the cipher text file (each time) for the possible keywords, with the results populated into a structure - being sorted based on the closest match to the language's frequency average.

4. **Detailed Comparisons**
   The decrypted messages are shown on the graphical user interface; facilitating the ability to select a message and view the corresponding decrypted text. In the circumstance that the language is unknown, the user can open a more detailed view – showing a comparison of the two frequencies together in an histogram, along with an even more detailed view displaying the compared frequency data in a grid (with the differences clearly shown).

Programming Style/ Conventions

- The application was written using the object-oriented Java programming language. The code is structured to provide the capability to be extended, with its detailed comments and object-orientated composition. The application is divided into two packages, enabling the system layer to be neatly separated from the graphical user interface; facilitating the system core to be easily transformed to another platform.

- The code was written with adherence to the Java coding conventions, allowing for the code to be comprehendible (more quickly and thoroughly) by new developers. The coding conventions cover aspects such as the appropriate indentation, programming practices, declarations, statements, and naming principles.
  Sun Microsystems (1997) 'Java Code Conventions'
  < *http://java.sun.com/docs/codeconv/CodeConventions.pdf* >.

- The code is commented with Java documentation, with the Javadoc files being additionally supplied. This enables the ability to browse through the classes (in the generated HTML) with detailed descriptions on the class and method level, assisting the developer to grasp how the application works. The Javadoc also provides a useful quick look-up tool allowing a developer to determine the usable class methods/ and functionality.

Programming References

- The algorithm and formulas used for the Kasiski method and Interwoven Caesar Shifts was adapted into this application, which was explained by Stinson (2006, pp. 32-36). The code was not copied, but however was written based on his explanation.

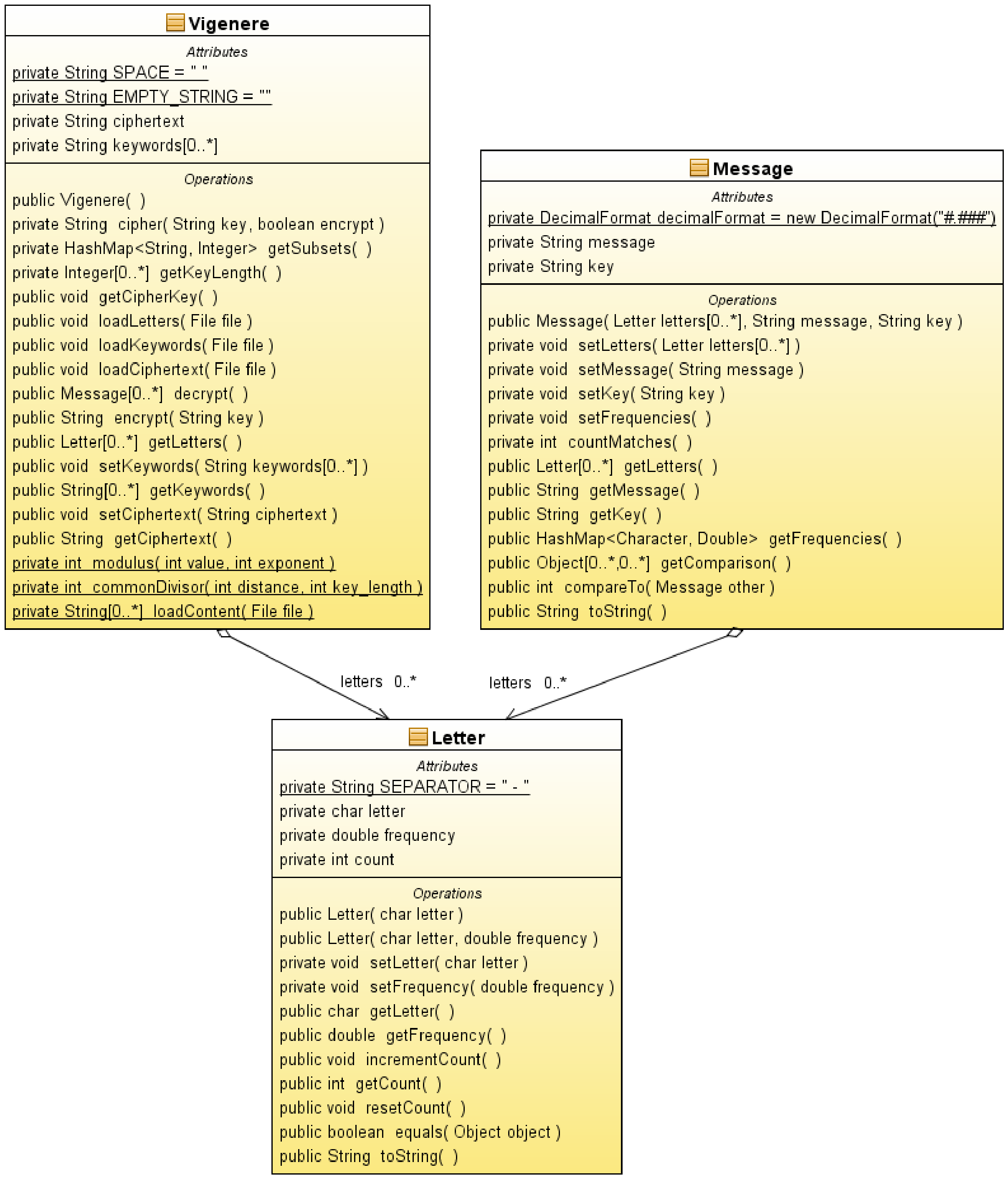
  Stinson, Douglas R 2006, *Cryptography – Theory and Practice (3rd edition),*
  Chapman & Hall/ CRC

- The application uses the JFreeChart Library, which enables the functionality to generate a histogram for the frequency comparison. The external Library is compiled within the Jar executable and since it is published under the GNU Lesser General Public Licence (LGPL) the usage is completely free.
  Object Refinery Limited (2009), 'JFreeChart' < *http://www.jfree.org/jfreechart/* >.

System Manual

Vigenere Package

UML Diagram


Vigenere
Attributes
private String SPACE = " "
private String EMPTY_STRING = ""
private String ciphertext
private String keywords[0..*]
Operations
public Vigenere( )
private String cipher(String key, boolean encrypt)
private HashMap<String, Integer> getSubsets( )
private Integer[0..*] getKeyLength( )
public void getCipherKey( )
public void loadLetters(File file)
public void loadKeywords(File file)
public void loadCiphertext(File file)
public Message[0..*] decrypt( )
public String encrypt(String key)
public Letter[0..*] getLetters( )
public void setKeywords(String keywords[0..*])
public String[0..*] getKeywords( )
public void setCiphertext(String ciphertext)
public String getCiphertext( )
private int modulus(int value, int exponent)
private int commonDivisor(int distance, int key_length)
private String[0..*] loadContent(File file)
Message
Attributes
private DecimalFormat decimalFormat = new DecimalFormat("#.###")
private String message
private String key
Operations
public Message(Letter letters[0..*], String message, String key)
private void setLetters(Letter letters[0..*])
private void setMessage(String message)
private void setKey(String key)
private void setFrequencies( )
private int countMatches( )
public Letter[0..*] getLetters( )
public String getMessage( )
public String getKey( )
public HashMap<Character, Double> getFrequencies( )
public Object[0..*,0..*] getComparison( )
public int compareTo(Message other)
public String toString( )
letters 0..*
letters 0..*
Letter
Attributes
private String SEPARATOR = " - "
private char letter
private double frequency
private int count
Operations
public Letter(char letter)
public Letter(char letter, double frequency)
private void setLetter(char letter)
private void setFrequency(double frequency)
public char getLetter( )
public double getFrequency( )
public void incrementCount( )
public int getCount( )
public void resetCount( )
public boolean equals(Object object)
public String toString( )

Class Descriptions

Class	Description
Vigenere	The Vigenere class contains the core functionality of the application. The class has methods used for loading the three text files (letter frequency, keywords, cipher text) and converts them into usable data structures. The class was designed to keep expansion in mind, and not only comprises the functionality to decrypt files - but to also encrypt them.
Letter	The letter class is used to store a languages alphabet character, and the frequency occurrence average in that language. The class is used in conjunction with a structure, and the ordering of the letters in the language is determined based on the element position. The class also contains a counter instance field, which is used to count the occurrence of a letter once a message has been decrypted.
Message	The Message class is used to contain data based on a decryption, which includes the keyword used, and the decrypted message based on that keyword. The class is also responsible for comparing the decrypted Messages based on the closest match to the languages frequency average. The closest match is determined based on a calculated rounded difference with each letter in the decryption, being compared to the languages average percentage; if the difference is zero, then a counter is incremented. The structure of messages is then sorted based on the counter.

Method Descriptions

Vigenere	
Method	**Description**
cipher	The cipher method is used to encrypt, and decrypt messages (the received Boolean indicates the formula which will be used). The Ciphertext String (which was loaded via a file) is iterated for each character value. A variable is used to keep track of the index position for the keyword; which is incremented, and then reset back to zero once the keyword length has been reached. This allows for the keyword to be repetitively cycled whilst iterating through each character in the Ciphertext, therefore each loop will be specifying a letter character, and the corresponding keyword character. Both character values are then found in the Letter structure, which grabs the position index - assigning it to their own local variables. The formula used to calculate the letter depends on the received Boolean. If the Boolean is true (being you are encrypting) then the letter position and key position is added together, otherwise the letter position is subtracted from the key position. The modulus of the resulting value is then calculated based on the Letter structure size, which is then used as the index position to fetch a Letter object (from the Letter structure). If the method is decrypting, then the Letter object invokes a method which increments a counter tally. The Letter object then fetches the inner character value, and appends it to a StringBuilder object (being essentially a wrapped character array). The resulting message is then returned as a String.
getSubsets	The getSubsets method is used in conjunction with the getKeyLength method in order to determine the possible key lengths used to encrypt the Ciphertext. A HashMap<String, Integer> is created (which is used to store the 3 character subsets and a counter value representing the occurrence). The method iterates an index value for the Ciphertext String length, and substrings the Ciphertext string based on the index position (plus 3 characters). The substring is then added to the HashMap (as a key) if it doesn't already exist, with starting counter of one. Otherwise if the substring is found, it increments the existing integer counter.

getKeyLength	The getKeyLength method uses the Kasiski approach in order to find the possible key lengths based on a encrypted Ciphertext message. This method partners with the getSubsets method (which finds repeating cipher subsets), and based on the max repeat count, the subsets which match are candidates to calculate the key length; since if there are more than one subset being repeated the same amount of times - the key length is less obvious. Therefore, with each subset iteration (and if its count is equal to the max count); the Ciphertext will be iterated for each occurrence of the repeated group - the distance between the subset is then calculated, and the greatest common denominator for all subset distances is calculated (which should be the key length). The key length is then added to a collection, which at the end of the iterations is sorted (based on lowest to highest) and returned.
getCipherKey	The getCipherKey method is the public interface which uses the getKeyLength method in order to decipher the Ciphertext in order to determine the keyword used. The method's purpose is to iterate the derived possible key lengths (using the Kasiski approach), and determine the keyword characters which were used to encrypt the Ciphertext through Interwoven Caesar Shifts; which uses letter frequency analysis to determine the most probable letter. It then populates the internal keyword structure with the (possible) derived keywords.
loadLetters	The loadLetters method is used to fetch the contents of a text file, interpret it, and add it to the internal data structures. First, the existing letters structure is cleared (which enables for letter frequency files to be reloaded) then the contents of the file is fetched based on the carriage returns using the static method loadContent() - with each line being a String element in the ArrayList. The content ArrayList is then iterated, and the data line is split into an array (based on the separating space); the first element being the character value, with the second being the frequency average. The char letter is checked with the static Character internal method to determine whether it’s a letter, and if so - a new instantiation of the Letter class is performed; passing the letter character and the converted frequency double in the constructor. The new object instance (with each iteration) is then added into the letters structure.

loadKeywords	The loadKeywords method fetches the contents of the keywords text-based File (separated on the carriage return),which is then added to a emptied keywords ArrayList<String>.
loadCiphertext	The loadCiphertext method receives the File object - which fetches the File content using the loadContent static method which returns a ArrayList<String> (each element being a line separated on the carriage return). A StringBuilder is instantiated, and the String lines in the content are iterated with each line being transformed into uppercased characters, and also being stripped of any white spaces. The resulting line with the iteration is appended to the StringBuilder object, and at the end of the loop the converted String value is assigned to the Ciphertext instance field.
decrypt	The decrypt method is used to decrypt the Ciphertext for each keyword. First, a collection of Message objects is instantiated, and then the keywords structure is iterated (if it’s not empty), with each iteration decrypting the message (based on the iterated key) by invoking the cipher method. The decrypted message is then added to a new Message instance - passing the letters structure (being an object reference), the decrypted message, and the key used to decrypt the message as arguments in the constructor. The new Message object is then added into the collection of messages, and the collection is sorted once the keywords have been fully iterated. The sort then orders the collection based on the messages closest letter frequency matches (compared with the frequency average).
encrypt	The encrypt method uses the loaded plaintext (which is inputted using the regular loadCiphertext method) and encrypts the text using the cipher method. The encrypted form is then iterated, with a white space character being appended once a counter has reached 5 (resetting the counter also back to zero). Therefore, the outputted String will be the encrypted Ciphertext formatted to have a space with every 5th character.
modulus	The modulus method is needed in order to perform a true mod operation. In the event that a negative number is encountered the % operator doesn't perform the mod, and the resulting number still remains negative. This is because Java’s internal % operator only finds the remainder.

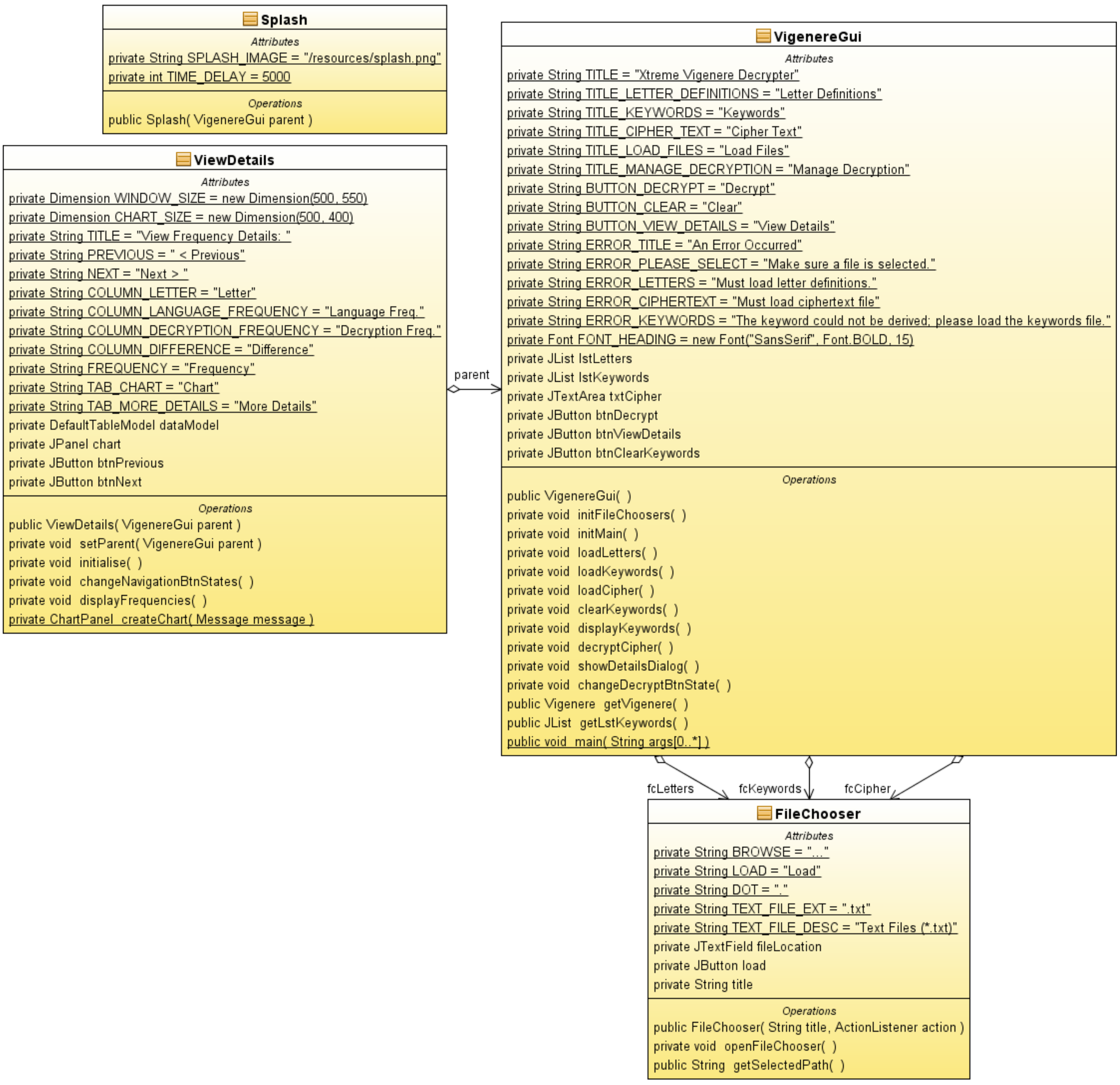
commonDivisor	The commonDivisor method is used recursively, and in conjunction with the getKeyLength method which is used to find the greatest common devisor between the subset character distance and the key length.
loadContent	The loadContent method is used by the three load file methods. It enables for the contents of a File to be retrieved without having to do the file handling in each method occurrence. The File object is placed in a FileReader object, which is then placed into a BufferedReader - which enables for the contents of the file to be iterated as its being read. The lines throughout the iteration are also placed as elements in an ArrayList<String> and returned once the end of the file has been reached.

Letter	
Method	**Description**
getLetter	Getter for the letter character.
getFrequency	Getter for the languages letter frequency.
incrementCount	Increments the counter by one. The method is used to count the occurrence of letters in a decrypted message.
getCount	Getter for the letter count.
resetCount	Resets the counter to zero.
equals	The overridden ‘equals’ method is used to compare Letter objects. The method is changed to only compare the inner letter character - which is used to quickly find the corresponding Letter in the main structure.
toString	The toString method is overridden to allow for a Letter to be represented in the form of a String.

Message	
Method	**Description**
setFrequencies	The setFrequency method iterates throughout the Letters structure, and calculates the percentage occurrence based on the Letter counter and message length. Stores the percentage in the frequencies HashMap with the Letter character used as the key index, and resets the Letter counter.
countMatches	The countMatches method iterates throughout the Letter Structure, and calculates a rounded difference between the messages Letter frequency and the languages average frequency. If there is no difference between the two values, then the counter is incremented.
getComparison	The getComparison method iterates the Letter structure; grabs the corresponding frequency for the decrypted messages, and calculates the difference between the two. It then adds the compared values to a Vector record, which is added to the row Vector. The method returns a Vector (containing the rows), with each row containing data for the comparison.
compareTo	The compareTo method is an implementation from the Comparable interface - which enables for the object to be sorted in a Collection. The method receives a Message object, which is then compared against the instance object. The comparison between both objects involves invoking the instance method countMatches() and comparing the results. If the instance Message is greater an positive integer is returned; otherwise if the received Message is greater an negative integer is returned. In the event that both objects match a zero is returned. The value returned then determines the direction the element is shuffled in the sorting process.

GUI Package

UML Diagram


Splash
Attributes
private String SPLASH_IMAGE = "/resources/splash.png"
private int TIME_DELAY = 5000
Operations
public Splash(VigenereGui parent)
ViewDetails
Attributes
private Dimension WINDOW_SIZE = new Dimension(500, 550)
private Dimension CHART_SIZE = new Dimension(500, 400)
private String TITLE = "View Frequency Details: "
private String PREVIOUS = " < Previous"
private String NEXT = "Next > "
private String COLUMN_LETTER = "Letter"
private String COLUMN_LANGUAGE_FREQUENCY = "Language Freq."
private String COLUMN_DECRYPTION_FREQUENCY = "Decryption Freq."
private String COLUMN_DIFFERENCE = "Difference"
private String FREQUENCY = "Frequency"
private String TAB_CHART = "Chart"
private String TAB_MORE_DETAILS = "More Details"
private DefaultTableModel dataModel
private JPanel chart
private JButton btnPrevious
private JButton btnNext
Operations
public ViewDetails(VigenereGui parent)
private void setParent(VigenereGui parent)
private void initialise( )
private void changeNavigationBtnStates( )
private void displayFrequencies( )
private ChartPanel createChart(Message message)
parent
VigenereGui
Attributes
private String TITLE = "Xtreme Vigenere Decrypter"
private String TITLE_LETTER_DEFINITIONS = "Letter Definitions"
private String TITLE_KEYWORDS = "Keywords"
private String TITLE_CIPHER_TEXT = "Cipher Text"
private String TITLE_LOAD_FILES = "Load Files"
private String TITLE_MANAGE_DECRYPTION = "Manage Decryption"
private String BUTTON_DECRYPT = "Decrypt"
private String BUTTON_CLEAR = "Clear"
private String BUTTON_VIEW_DETAILS = "View Details"
private String ERROR_TITLE = "An Error Occurred"
private String ERROR_PLEASE_SELECT = "Make sure a file is selected."
private String ERROR_LETTERS = "Must load letter definitions."
private String ERROR_CIPHERTEXT = "Must load ciphertext file"
private String ERROR_KEYWORDS = "The keyword could not be derived; please load the keywords file."
private Font FONT_HEADING = new Font("SansSerif", Font.BOLD, 15)
private JList lstLetters
private JList lstKeywords
private JTextArea txtCipher
private JButton btnDecrypt
private JButton btnViewDetails
private JButton btnClearKeywords
Operations
public VigenereGui( )
private void initFileChoosers( )
private void initMain( )
private void loadLetters( )
private void loadKeywords( )
private void loadCipher( )
private void clearKeywords( )
private void displayKeywords( )
private void decryptCipher( )
private void showDetailsDialog( )
private void changeDecryptBtnState( )
public Vigenere getVigenere( )
public JList getLstKeywords( )
public void main(String args[0..*])
fcLetters
fcKeywords
fcCipher
FileChooser
Attributes
private String BROWSE = "..."
private String LOAD = "Load"
private String DOT = "."
private String TEXT_FILE_EXT = ".txt"
private String TEXT_FILE_DESC = "Text Files (*.txt)"
private JTextField fileLocation
private JButton load
private String title
Operations
public FileChooser(String title, ActionListener action)
private void openFileChooser( )
public String getSelectedPath( )

Class Descriptions

Class	Description
VigenereGUI	The VigenereGui class is the main class within the graphical user interface. It extends JFrame, and contains all the graphical components used by the application. Upon instantiation, it creates two sections - the first section is used to display the three FileChooser objects (for the letter frequencies, the keyword file, and the cipher text file). The second section has two JList components (for the letter frequencies, and the keywords) and one JTextArea which is used to display the cipher text message. The keyword JList and the letter frequencies JList is populated once the load button is pressed on the corresponding FileChooser. The JTextArea is set to contain the contents of the encrypted message once it's loaded, however when the 'Decrypt' button is pressed the keywords JList is re-populated with Message objects (that corresponds to each keyword, hence still displaying the keyword) and a message once selected displays the decrypted message into the JTextArea.
Splash	The Splash class is essentially a JDialog which is displayed for a short duration. The JDialog has an image added, non re-sizable, and the system buttons (close, maximise, minimise) are set to undecorated - therefore the whole window title bar disappears, only leaving the contents to be displayed.
FileChooser	The file chooser class is used to create an object instance for each file chooser; which contains a JTextField, a 'Browse' JButton, and a 'Load' JButton that also receives the desired action to be performed. The class also extends JPanel, therefore each FileChooser instance is a JPanel with the components added to it. The Browse button uses the JFileChooser class, essentially being a dialog box that allows for directory file browsing, and once a file is selected - the JTextField is set to contain the selected file path.
ViewDetails	The ViewDetails class is used to show further details based on the message decryption; showing comparison between the decrypted message letter frequencies and the language average. The comparison between the letter frequencies are displayed in two approaches: The first being a bar graph overview - which shows the comparison for each letter, and the second being a more in-depth view showing the rounded differences (within a JTable).

Method Descriptions

VigenereGUI

Method	Description
initFileChoosers	The initFileChoosers method creates a JPanel with a single rowed 3 column GridLayout, adding three instances of the FileChooser class (which extends JPanel) to each column space. The instances are used by the three different types of files (the letter frequencies, the keywords, and the Ciphertext) that require loading by the application. The instantiations of the FileChooser also need a corresponding Action for the load button operation, which invokes other methods contained within this class. The created JPanel is then added to the north pane of the JFrame.
initMain	The initMain method is used to add the component-guts of the application. The method creates the letter freq. JList, the keywords JList, and the JTextArea for the Ciphertext. The method also creates the JButtons used to control the interface.
loadLetters	The loadLetters method is used to load the contents of the selected file into the system. The Vigenere object's loadLetters method accepts the File object to be loaded (which interprets the text within the document). The loaded letter structure is then converted to an Array, and loaded into the letters JList. The changeDecryptBtnState method is then invoked, which handles whether the decrypt button should be enabled - since the decryption functionality requires more than one file to be loaded.
loadKeywords	The loadKeywords method is used to load the keywords file. The path is grabbed from the keywords FileChooser, and added to a new File Object instance - which is then passed as an argument to the Vigenere object's loadKeywords system method. Once the keywords file is loaded - the displayKeywords method is then invoked.

loadCipher	The loadCipher method loads the chosen ciphertext file into the system. The selected file path, is added to a new File object, which is then passed as an argument to the Vigenere loadCiphertext() system method. The JTextArea is then set to display the contents of the encrypted message loaded, and the position of the scroll bar is reset to the top. The displayKeywords method is then invoked; which ensures that the keywords JList contains just String keywords (and not the decrypted Message objects). This allows for the Ciphertext to be reloaded (without issues) once a decryption has already taken place.
clearKeywords	The clearKeywords method is used to clear both the stored keywords in the Vigenere object, and the keywords DefaultListModel structure. Once both structures are cleared, it disables the Clear JButton (since it cannot be emptied again).
displayKeywords	The displayKeywords method grabs the DefaultListModel from the keywords JList, clears it, and re-adds the keywords by iterating throughout the Vigenere system keyword structure - adding each keyword String to the model. The clear JButton is then enabled if the keyword structure size has elements.
decryptCipher	The decryptCipher method is used to decrypt a given encrypted message. If the keywords list isn't loaded (or cleared), it will attempt to derive the keyword and populate the keyword structure with the results found. Once the system has keywords, the decryption process takes place; it decrypts the message for every given keyword, and orders the results based on the closest letter frequency matches (outputting it as a structure of Message(s)). The decrypted Message objects are then populated into an already cleared keywords JList model (which updates the GUI component). The clear button is then enabled, and the last index of the JList is selected (which should be the decrypted message) in doing so triggering the selected index changed event (which outputs the message to the user).

showDetailsDialog	The showDetailsDialog method prompts the ViewDetails JDialog - Only if the keywords JList has a selection, and the selected element in the JList is a Message object.
changeDecryptBtnState	Enables the Decrypt Button - if the Letters and Ciphertext file is loaded.

FileChooser

Method	Description
openFileChooser	The openFileChooser method uses the JFileChooser to prompt a dialog which enables for the user to browse file directories; which is filtered to only display folders, and text file documents. Once a user has selected a file - the file path is then added to the JTextField file location, and the load button is enabled.
getSelectedPath	A getter that returns the file location (being the text within the JTextField).

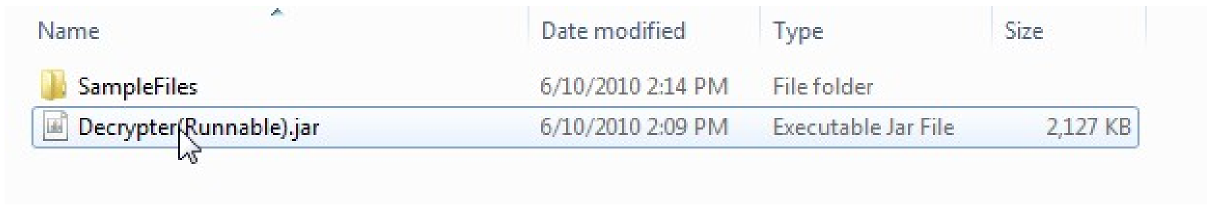
ViewDetails

Method	Description
initialise	The initialise method is pretty much the main method for the Dialog. It is invoked from the constructor before the dialog is shown (therefore the components are fully added before it's displayed). The main content pane is added with a JTabbedPane in the centred region, with the navigation buttons (JPanel) added to the south. The JTabbedPane is created with two tabs - the first being a JPanel (used as a container for the bar chart) and the second being a JTable.
changeNavigationBtnStates	This method is used to determine whether the navigation buttons should be enabled or disabled. It checks whether the buttons have enough room - being that the selected index is greater (or lesser) than the structures regions.
displayFrequencies	The displayFrequencies method shows the comparison between the letter frequencies of the language, and the frequencies occurred in the decrypted message. It enables for the user to understand whether the message has been properly decrypted (without being required to know the language). The method grabs the currently selected value from the keywords JList (which in this case is a Message object). The retrieved message object comparison is added to a new Chart (which replaces the previous chart in the Chart JPanel) and the JTable model is populated with the compared data.
createChart	The createChart method uses the external JFreeChart library in order to have the capability to create the chart. The method receives the Message object which is then used to get the comparison data.

User Manual

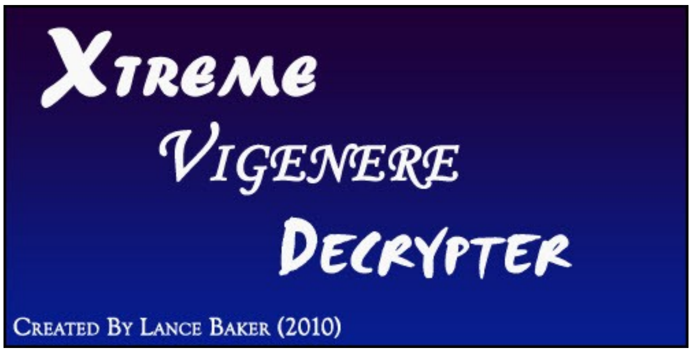
Opening the application

1) **Open** the *application* folder.
2) **Double click** on the *Decrypter(Runnable).jar* File.

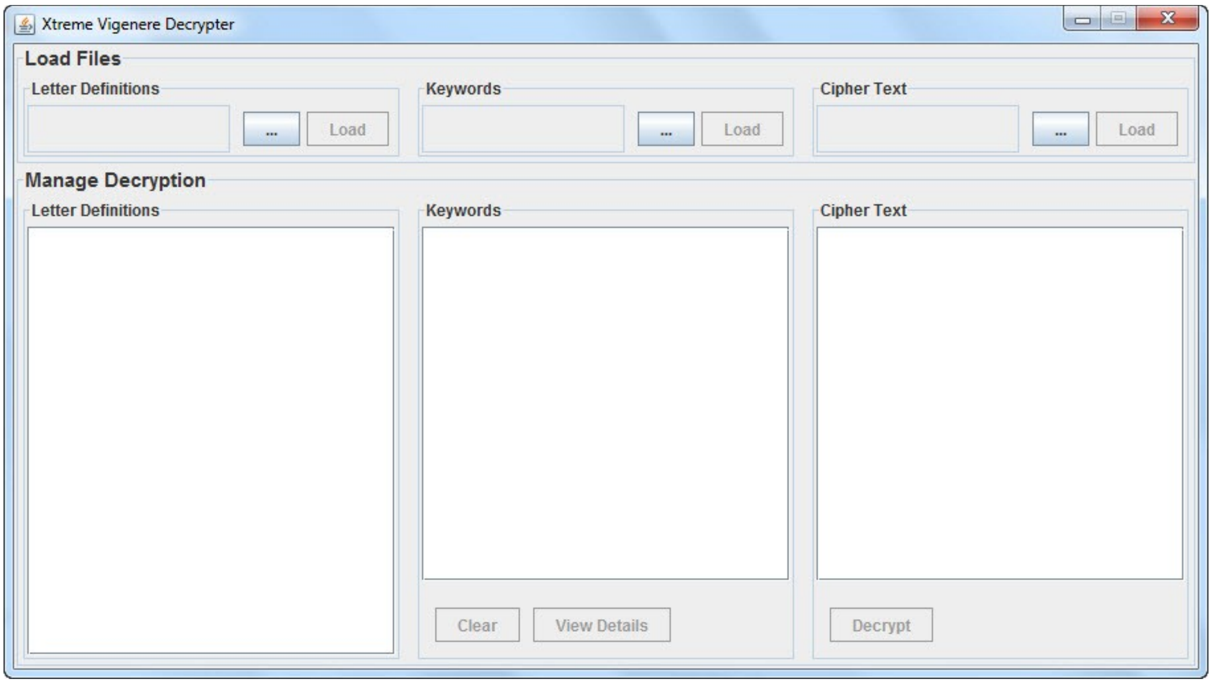


Name	Date modified	Type	Size
SampleFiles	6/10/2010 2:14 PM	File folder	
Decrypter(Runnable).jar	6/10/2010 2:09 PM	Executable Jar File	2,127 KB

i) [The splash screen will be displayed]



ii) [The application will load after five seconds]

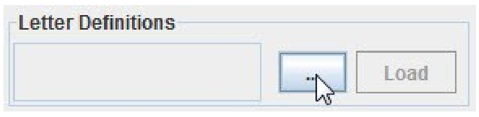


(Please note the disabled buttons - only allowing the user to proceed to select files to load).

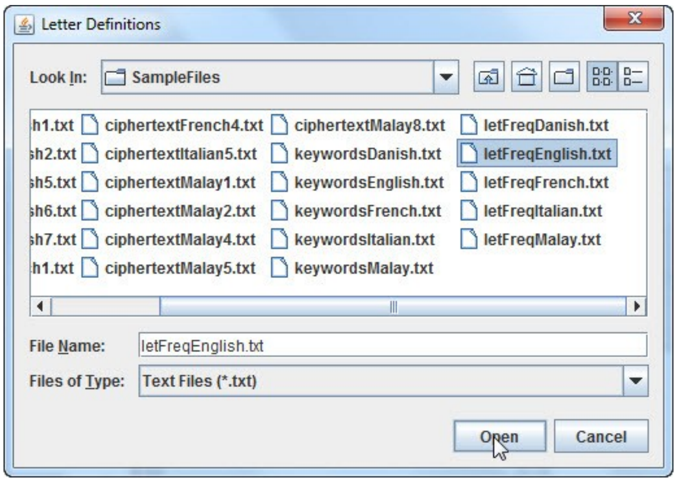
Loading Files

Letter Definitions

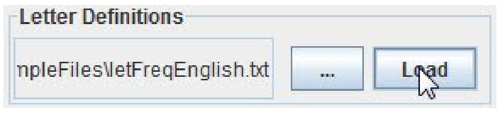
1. **Click** the *'…' Browse* Button.

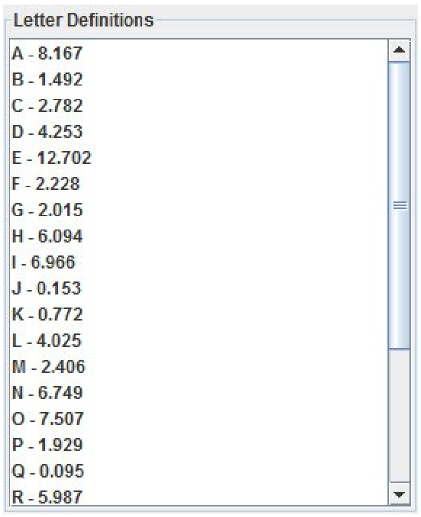


1. [The file browsing dialog box appears]
   Navigate to the desired Letter Frequency/Definition File.
2. Ensure the file is selected, and **Click** '*Open*'.



3. [The path is displayed in the textbox and the *Load* Button is enabled].
   Click the '*Load*' Button.





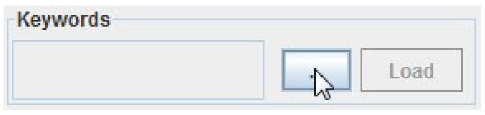
A - 8.167
B - 1.492
C - 2.782
D - 4.253
E - 12.702
F - 2.228
G - 2.015
H - 6.094
I - 6.966
J - 0.153
K - 0.772
L - 4.025
M - 2.406
N - 6.749
O - 7.507
P - 1.929
Q - 0.095
R - 5.987

The letter definitions file is then interpreted and loaded into the system - stored in a dynamic structure in memory. The loaded letters are then iterated, and displayed in a non-selectable List box; enabling the user to view the contents of the loaded file. The letter is also shown regardless of the language (supporting all Unicode characters).

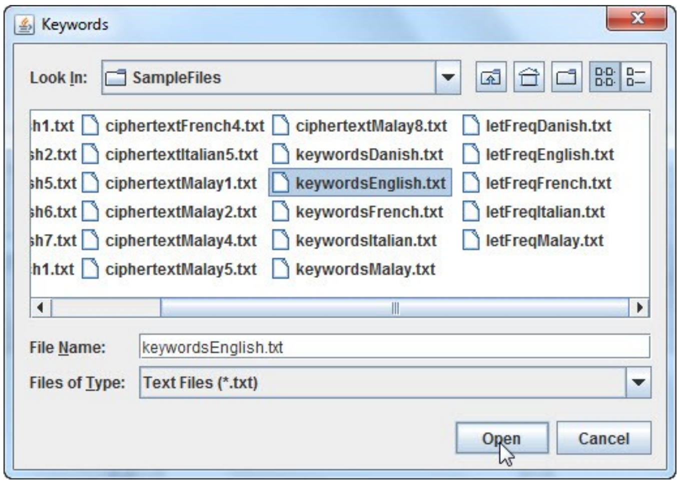
In order to load a different letter definition file - just browse and load another; which will clear, and repopulate the Letter structure with the new definitions.

Keywords

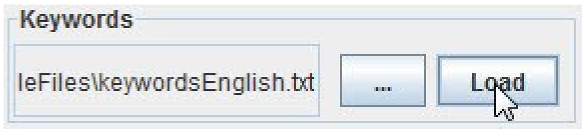
2. **Click** the *'…' Browse* Button.

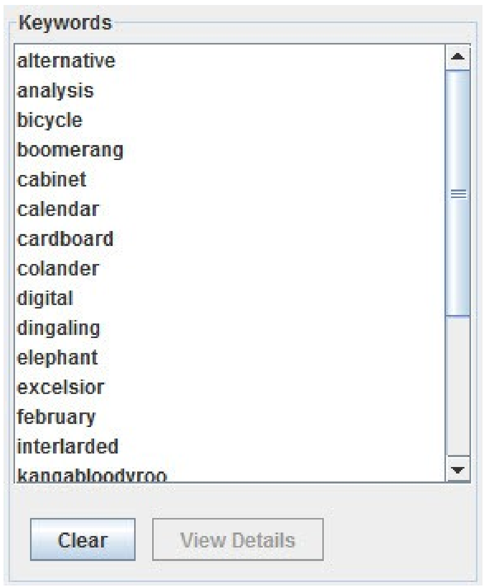


1. [The file browsing dialog box appears]
   Navigate to the desired Keyword File.
2. Ensure the file is selected, and **Click** '*Open*'.



3. [The path is displayed in the textbox and the *Load* Button is enabled].
   Click the '*Load'* Button.



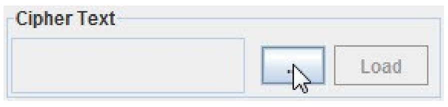


The keyword file is interpreted and loaded into an internal memory structure. The keywords are then displayed in their keyword List box as String objects – which have two buttons underneath. The Clear Button is enabled once the keywords are loaded (which once pressed clears the List box and the internal memory structure). The View Details Button is greyed out, which will become enabled once the file is decrypted.

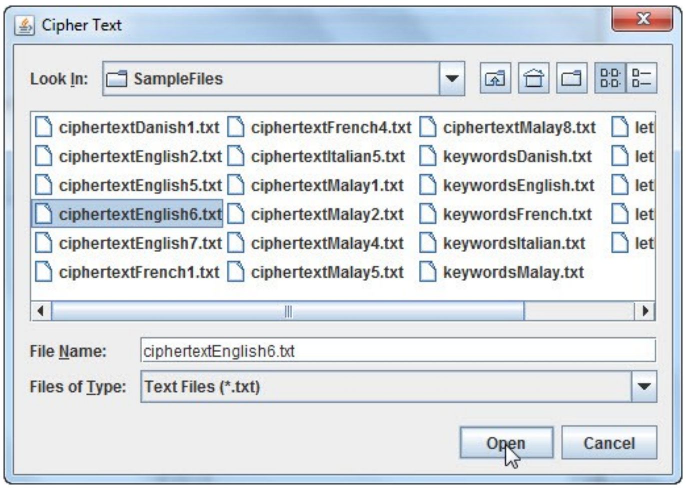
The keywords List box is shared with the decrypted messages (for each keyword) and once the file is decrypted it will be repopulated with a list of decrypted messages corresponding to the keyword used.

Cipher Text

3. **Click** the *'…' Browse* Button.



4. [The file browsing dialog box appears]
   Navigate to the desired Cipher Text File.
5. Ensure the file is selected, and **Click** '*Open*'.

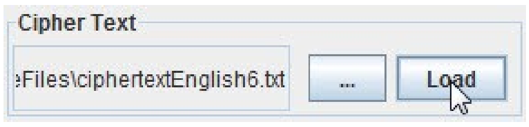


```
ciphertextDanish1.txt    ciphertextFrench4.txt    ciphertextMalay8.txt    let
ciphertextEnglish2.txt   ciphertextItalian5.txt   keywordsDanish.txt      let
ciphertextEnglish5.txt   ciphertextMalay1.txt     keywordsEnglish.txt     let
ciphertextEnglish6.txt   ciphertextMalay2.txt     keywordsFrench.txt      let
ciphertextEnglish7.txt   ciphertextMalay4.txt     keywordsItalian.txt     let
ciphertextFrench1.txt    ciphertextMalay5.txt     keywordsMalay.txt
```

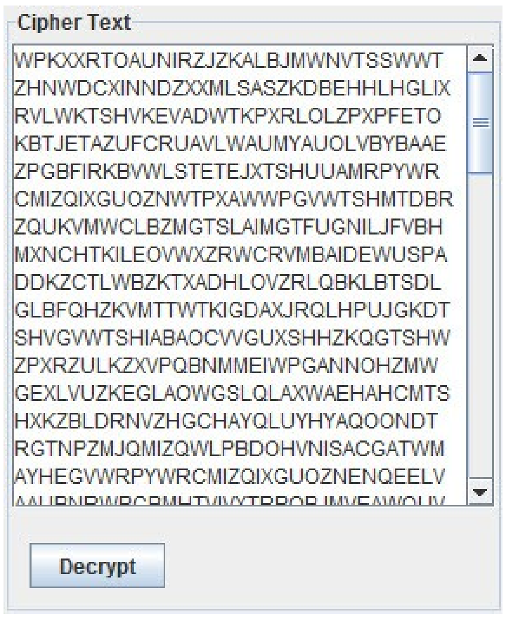
File Name: ciphertextEnglish6.txt

Files of Type: Text Files (*.txt)

6. [The path is displayed in the textbox and the *Load* Button is enabled].
   Click the '*Load*' Button.



```
:Files\ciphertextEnglish6.txt
```



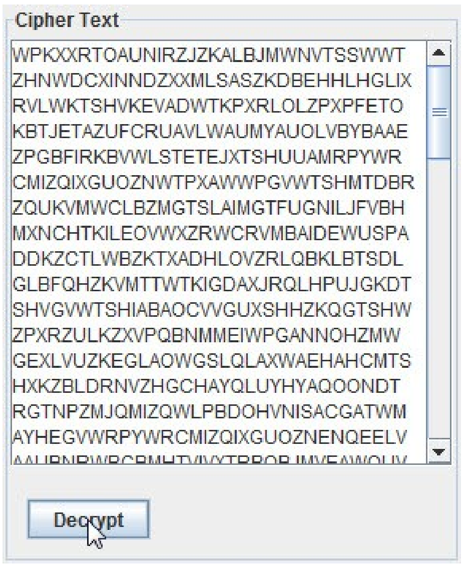
```
WPKXXRTOAUNIRZJZKALBJMWNVTSSWWT
ZHNWDCXINNDZXXMLSASZKDBEHHLHGLIX
RVLWKTSHVKEVADWTKPXRLOLZPXPFETO
KBTJETAZUFCRUAVLWAUMYAUOLVBYBAAE
ZPGBFIRKBVWLSTETEJXTSHUUAMRPYWR
CMIZQIXGUOZNWTPXAWWPGVWTSHMTDBR
ZQUKVMWCLBZMGTSLAIMGTFUGNILJFVBH
MXNCHTKILEOVWXZRWCRVMBAIDEWUSPA
DDKZCTLWBZKTXADHLOVZRLQBKLBTSDL
GLBFQHZKVMTTWTKIGDAXJRQLHPUJGKDT
SHVGWWTSHIABAOCVVGUXSHHZKQGTSHW
ZPXRZULKZXVPQBNMMEIWPGANNOHZMW
GEXLVUZKEGLAOWGSLQLAXWAEHAHCMTS
HXKZBLDRNVZHGCHAYQLUYHYAQOONDT
RGTNPZMJQMIZQWLPBDOHVNISACGATWM
AYHEGVWRPYWRCMIZQIXGUOZNENQEELV
```

The encrypted text from the selected file will be interpreted and stored into memory - capitalising all characters and removing any spaces, with the output being displayed in the scrollable read-only Text area.

The Decrypt Button becomes enabled once the required files (Letter Definitions and Cipher Text) have been loaded. The keywords file is not mandatory due to the extended functionality which attempts to break the cipher without being supplied the keywords list.

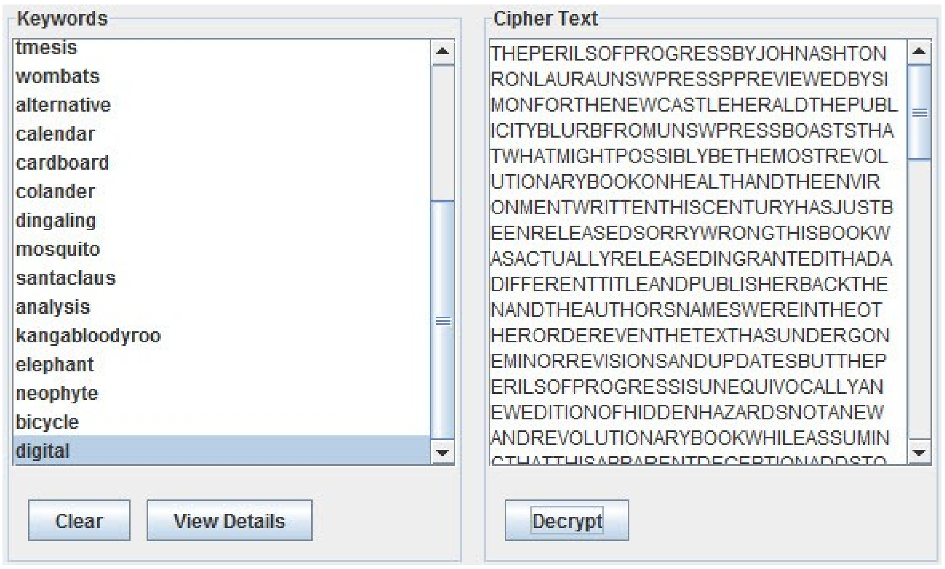
Decrypting Ciphertext

1. **Click** the *'Decrypt'* Button.
   (The Decrypt Button is enabled once the required files are loaded).



```
WPKXXRTOAUNIRZJZKALBJMWNVTSSWWT
ZHNWDCXINNDZXXMLSASZKDBEHHLHGLIX
RVLWKTSHVKEVADWTKPXRLOLZPXPFETO
KBTJETAZUFCRUAVLWAUMYAUOLVBYBAAE
ZPGBFIRKBVWLSTETEJXTSHUUAMRPYWR
CMIZQIXGUOZNWTPXAWWPGVWTSHMTDBR
ZQUKVMWCLBZMGTSLAIMGTFUGNILJFVBH
MXNCHTKILEOVWXZRWCRVMBAIDEWUSPA
DDKZCTLWBZKTXADHLOVZRLQBKLBTSDL
GLBFQHZKVMTTWTKIGDAXJRQLHPUJGKDT
SHVGVWTSHIABAOCVVGUXSHHZKQGTSHW
ZPXRZULKZXVPQBNMMEIWPGANNOHZMW
GEXLVUZKEGLAOWGSLQLAXWAEHAHCMTS
HXKZBLDRNVZHGCHAYQLUYHYAQOONDT
RGTNPZMJQMIZQWLPBDOHVNISACGATWM
AYHEGVWRPYWRCMIZQIXGUOZNENQEELV
```

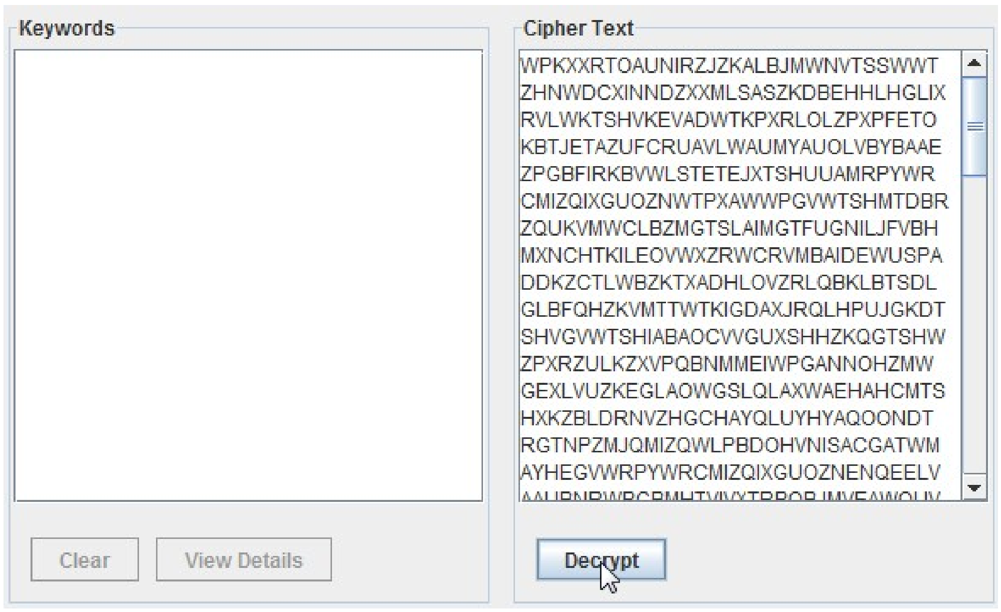
i) In the circumstance that the possible keyword file is supplied - the loaded encrypted text is then decrypted for each keyword loaded in memory; which is then sorted based on the closest match between the languages average letter frequency with the decrypted message letter frequency. The keyword List is populated with the sorted decrypted messages, and the last element in the List (which is most likely the decrypted message) is selected automatically. The Messages in the Keyword List updates the corresponding Cipher text-area with the decrypted message on selection. Therefore enabling the user to navigate between the messages. The 'View Details' Button provides the user with frequency comparison details (in case the user isn't too familiar with the language).



```
tmesis
wombats
alternative
calendar
cardboard
colander
dingaling
mosquito
santaclaus
analysis
kangabloodyroo
elephant
neophyte
bicycle
digital
```

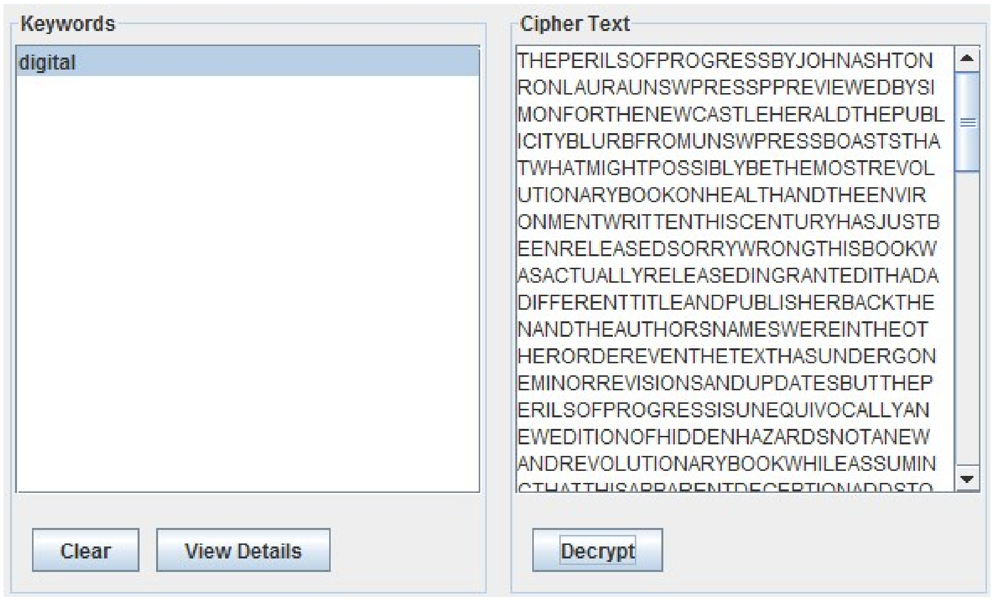
```
THEPERILSOFPROGRESSBYJOHNASHTON
RONLAURAUNSWPRESSPPREVIEWEDBYSI
MONFORTHENEWCASTLEHERALDTHEPUBL
ICITYBLURBFROMUNSWPRESSBOASTSTHA
TWHATMIGHTPOSSIBLYBETHEMOSTREVOL
UTIONARYBOOKONHEALTHANDTHEENVIR
ONMENTWRITTENTHISCENTURYHASJUSTB
EENRELEASEDSORRYWRONGTHISBOOKW
ASACTUALLYRELEASEDINGRANTEDITHADA
DIFFERENTTITLEANDPUBLISHERBACKTHE
NANDTHEAUTHORSNAMESWEREINTHEOT
HERORDEREVENTHETEXTHASUNDERGON
EMINORREVISIONSANDUPDATESBUTTHEP
ERILSOFPROGRESSISUNEQUIVOCALLYAN
EWEDITIONOFHIDDENHAZARDSNOTANEW
ANDREVOLUTIONARYBOOKWHILEASSUMIN
```

ii) If the keywords file was not loaded (or cleared) then the system will attempt to deduce the used keyword based on the encrypted text – using the Kasiski method to find the key length, then deriving the keyword through Interwoven Caesar Shifts with character analysis (based on the length found).



```
WPKXXRTOAUNIRZJZKALBJMWNVTSSWWT
ZHNWDCXINNDZXXMLSASZKDBEHHLHGLIX
RVLWKTSHVKEVADWTKPXRLOLZPXPFETO
KBTJETAZUFCRUAVLWAUMYAUOLVBYBAAE
ZPGBFIRKBVWLSTETEJXTSHUUAMRPYWR
CMIZQIXGUOZNWTPXAWWPGVWTSHMTDBR
ZQUKVMWCLBZMGTSLAIMGTFUGNILJFVBH
MXNCHTKILEOVWXZRWCRVMBAIDEWUSPA
DDKZCTLWBZKTXADHLOVZRLQBKLBTSDL
GLBFQHZKVMTTWTKIGDAXJRQLHPUJGKDT
SHVGVWTSHIABAOCVVGUXSHHZKQGTSHW
ZPXRZULKZXVPQBNMMEIWPGANNOHZMW
GEXLVUZKEGLAOWGSLQLAXWAEHAHCMTS
HXKZBLDRNVZHGCHAYQLUYHYAQOONDT
RGTNPZMJQMIZQWLPBDOHVNISACGATWM
AYHEGVWRPYWRCMIZQIXGUOZNENQEELV
```

Once the Decrypt Button is pressed the system will attempt to break the cipher; storing each possible keyword in the keyword structure. The keywords found will then be used to decrypt the loaded cipher text, outputting the decrypted message results into a sorted Message structure, which are then populated into the keywords List box.

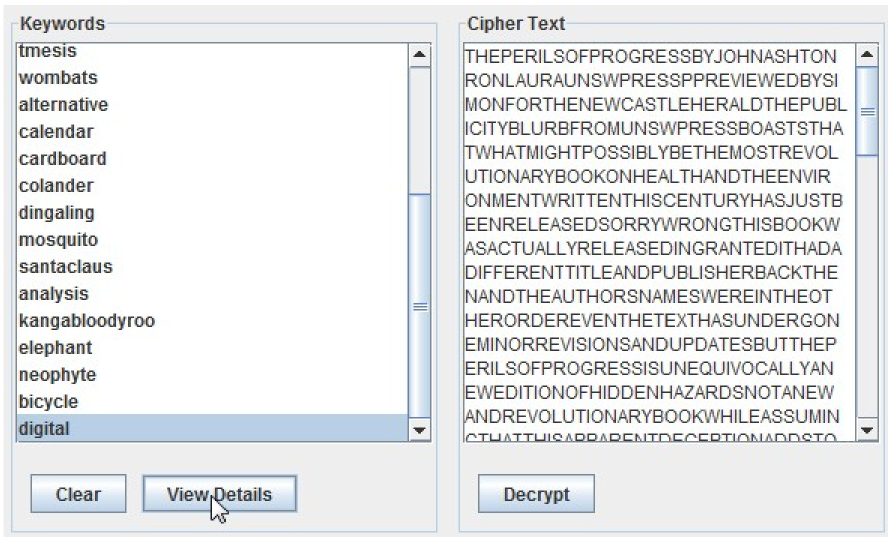


```
THEPERILSOFPROGRESSBYJOHNASHTON
RONLAURAUNSWPRESSPPREVIEWEDBYSI
MONFORTHENEWCASTLEHERALDTHEPUBL
ICITYBLURBFROMUNSWPRESSBOASTSTHA
TWHATMIGHTPOSSIBLYBETHEMOSTREVOL
UTIONARYBOOKONHEALTHANDTHEENVIR
ONMENTWRITTENTHISCENTURYHASJUSTB
EENRELEASEDSORRYWRONGTHISBOOKW
ASACTUALLYRELEASEDINGRANTEDITHADA
DIFFERENTTITLEANDPUBLISHERBACKTHE
NANDTHEAUTHORSNAMESWEREINTHEOT
HERORDEREVENTHETEXTHASUNDERGON
EMINORREVISIONSANDUPDATESBUTTHEP
ERILSOFPROGRESSISUNEQUIVOCALLYAN
EWEDITIONOFHIDDENHAZARDSNOTANEW
ANDREVOLUTIONARYBOOKWHILEASSUMIN
```

Comparing Results

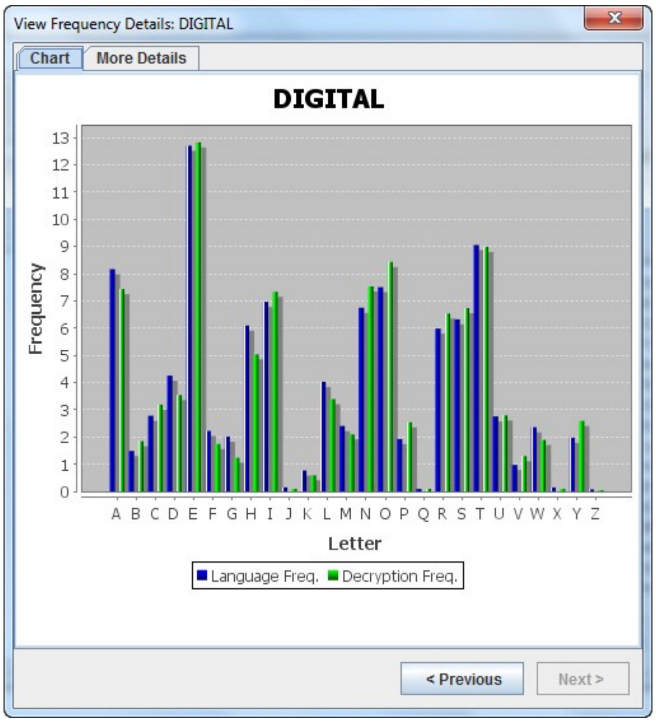
The decrypted messages once selected update the adjacent Cipher text area with its interpretation of the decrypted message based on the keyword used. If for instance the user wishes to compare the decrypted letter frequencies with the languages frequency average (in case they aren't too familiar with the language) then the 'View Details' Button provides an extended inspection for comparing the two frequencies.

1) **Click** on the '*View Details*' Button which corresponds to the selected message.



tmesis
wombats
alternative
calendar
cardboard
colander
dingaling
mosquito
santaclaus
analysis
kangabloodyroo
elephant
neophyte
bicycle
digital

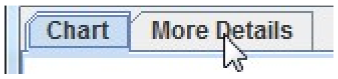
```
THEPERILSOFPROGRESSBYJOHNASHTON
RONLAURAUNSWPRESSPPREVIEWEDBYSI
MONFORTHENEWCASTLEHERALDTHEPUBL
ICITYBLURBFROMUNSWPRESSBOASTSTHA
TWHATMIGHTPOSSIBLYBETHEMOSTREVOL
UTIONARYBOOKONHEALTHANDTHEENVIR
ONMENTWRITTENTHISCENTURYHASJUSTB
EENRELEASEDSORRYWRONGTHISBOOKW
ASACTUALLYRELEASEDINGRANTEDITHADA
DIFFERENTTITLEANDPUBLISHERBACKTHE
NANDTHEAUTHORSNAMESWEREINTHEOT
HERORDEREVENTHETEXTHASUNDERGON
EMINORREVISIONSANDUPDATESBUTTHEP
ERILSOFPROGRESSISUNEQUIVOCALLYAN
EWEDITIONOFHIDDENHAZARDSNOTANEW
ANDREVOLUTIONARYBOOKWHILEASSUMIN
```



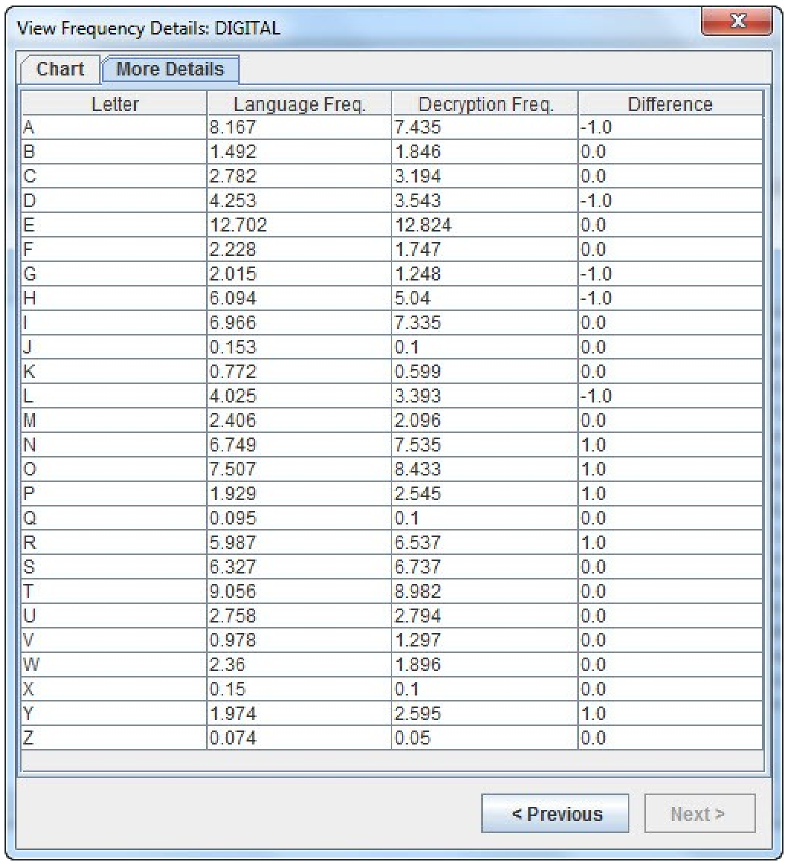
The 'View Details' dialog appears in front of the main application interface window (with greater focus) and consists of two tabbed regions. The first tab selected is comparing the languages frequency average with the message's frequency in a generated histogram.

The navigation buttons provide the user with the ability to cycle between the decrypted messages (which changes the selected index position of the parent window's keywords List) updating both the Cipher text area and the dialog contents.

2) **Click** on the '*More Details*' Tab.



The frequency comparison for each letter is shown in a data grid – with the letters being listed as a row; each letter displaying its character, language frequency average, the decrypted message frequency, and a rounded difference between the two frequency values. The navigation buttons still operate the same; allowing the user to cycle throughout the decrypted messages on this tabbed region (showing the updated data-grid).



Letter	Language Freq.	Decryption Freq.	Difference
A	8.167	7.435	-1.0
B	1.492	1.846	0.0
C	2.782	3.194	0.0
D	4.253	3.543	-1.0
E	12.702	12.824	0.0
F	2.228	1.747	0.0
G	2.015	1.248	-1.0
H	6.094	5.04	-1.0
I	6.966	7.335	0.0
J	0.153	0.1	0.0
K	0.772	0.599	0.0
L	4.025	3.393	-1.0
M	2.406	2.096	0.0
N	6.749	7.535	1.0
O	7.507	8.433	1.0
P	1.929	2.545	1.0
Q	0.095	0.1	0.0
R	5.987	6.537	1.0
S	6.327	6.737	0.0
T	9.056	8.982	0.0
U	2.758	2.794	0.0
V	0.978	1.297	0.0
W	2.36	1.896	0.0
X	0.15	0.1	0.0
Y	1.974	2.595	1.0
Z	0.074	0.05	0.0