|  |  |  |
| --- | --- | --- |
| **ipn** | **INSTITUTO POLITÉCNICO NACIONAL**  **ESCUELA SUPERIOR DE CÓMPUTO** |  |

**Cryptography**

**“Shift - Cipher”**

Abstract

One of the most popular classic ciphers is the Caesar Cipher. In this paper I explain the procedure to apply the shift-cipher algorithm, which is an improvement of the Caesar. The user interface and algorithm were made using Java programing language­.

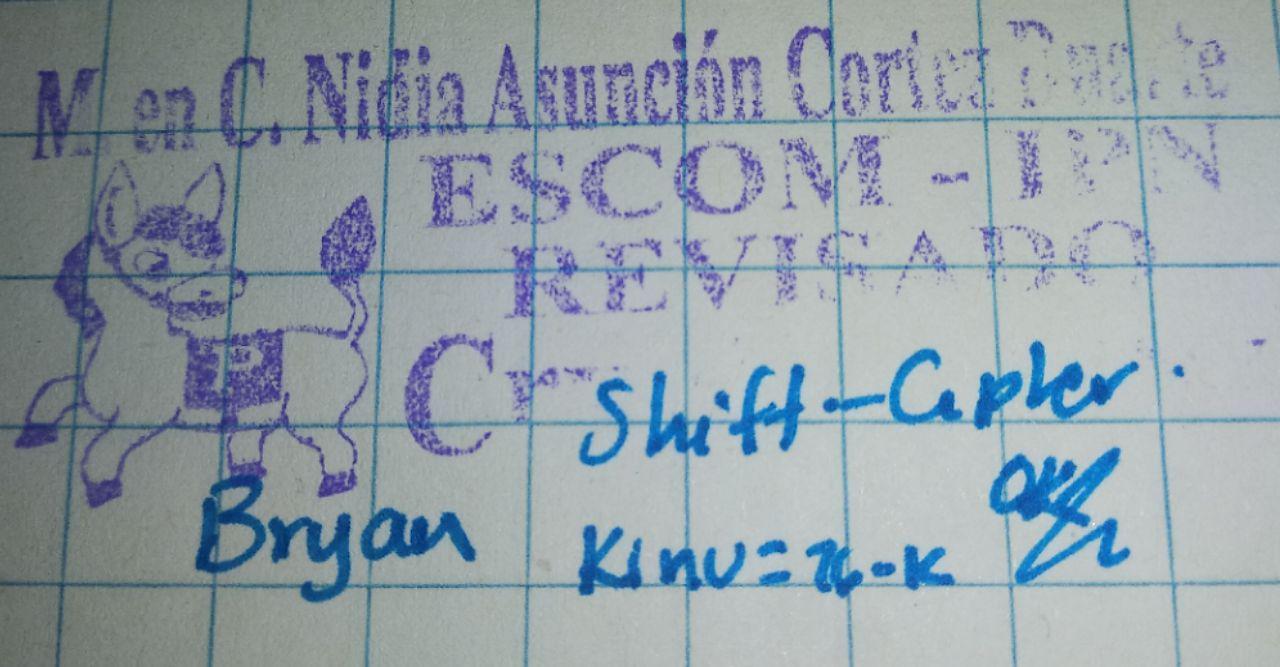
**By:**

**Bryan Dominguez de la Rosa**

Professor:

MSc. NIDIA ASUNCIÓN CORTEZ DUARTE

August 2018



**Index**

Contenido

[Introduction: 1](#_Toc522776215)

[Literature review: 1](#_Toc522776216)

[Software (libraries, packages, tools): 2](#_Toc522776217)

[Procedure: 3](#_Toc522776218)

[Results: 4](#_Toc522776219)

[Discussion: 8](#_Toc522776220)

[Conclusions: 8](#_Toc522776221)

[References: 8](#_Toc522776222)

[Code 9](#_Toc522776223)

# Introduction:

The Caesar Cipher consist in replacing each letter of the alphabet for its corresponding three “jumps” after, for examples, letter A corresponds to letter D, letter B corresponds to letter E and so on. The aim of the Shift-Cipher algorithm is to let the user to decide the number of jumps, i.e. the key, that the alphabet is going to have. If the key given is 5, then the letter A corresponds to the letter F and so on. The main idea of the encryption and decryption algorithm is to use the ASCII code of each letter of the English alphabet (26 symbols) and modular arithmetic to find the correspondence. The user is going to select a plane text file and type the key, then is going to give the name for the encrypted output file. The decryption process is the same.

# Literature review:

The cryptology is the science that deal with theoretical problems related with security in encrypted messages exchange from a sender to a receiver through a channel of communication (in informatic terms, this channel is usually a computer network). [1]

The use of cryptographic techniques is almost as old as the cultures of the first villages of our planet. The major advances were achieved in the First and Second World War, especially during and after the last one. The countries in conflict had real companies with a great number of mathematicians whose function was to break the encrypted messages of the tickers exchanged by their enemies.

The Fig. 1 shows the classification of classic cryptosystems [2]:

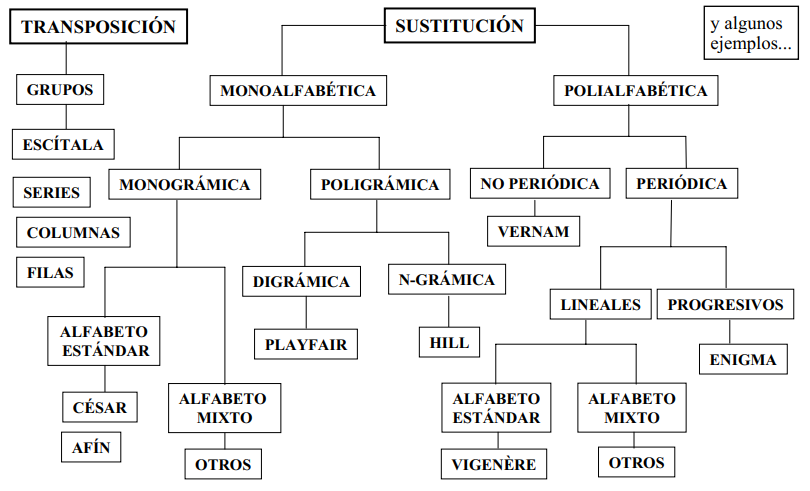


Fig. 1 Classification of classic cryptosystems

As can be seen, the main division is the transposition and substitution algorithms. The Shift-Cipher is in the category of the monoalphabetic substitution, because every single letter of the message that we are encrypting will be mapped to another letter of the alphabet that we are using, usually a 26 letter alphabet.

In the Shift Cipher, the encryption is performed by replacing each letter by the letter a certain number of places on in the alphabet. One technique of breaking this algorithm is using frequency analysis. As can be seen in Fig.2, one graph looks almost like a shift of the other graph. The plain text is given in red and the cipher text is given in blue. [3]

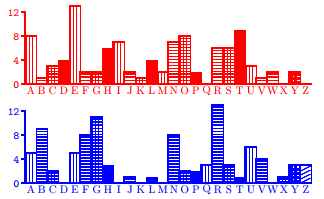


Fig. 2 Comparison of plaintext and ciphertext frequencies for a shift cipher example

# Software (libraries, packages, tools):

I use Java programing language to develop this practice, so the tools that I used were:

* NetBeans IDE 8.2
* Java Development Kit (jdk) 8
* All libraries used were of the Java Standard

# Procedure:

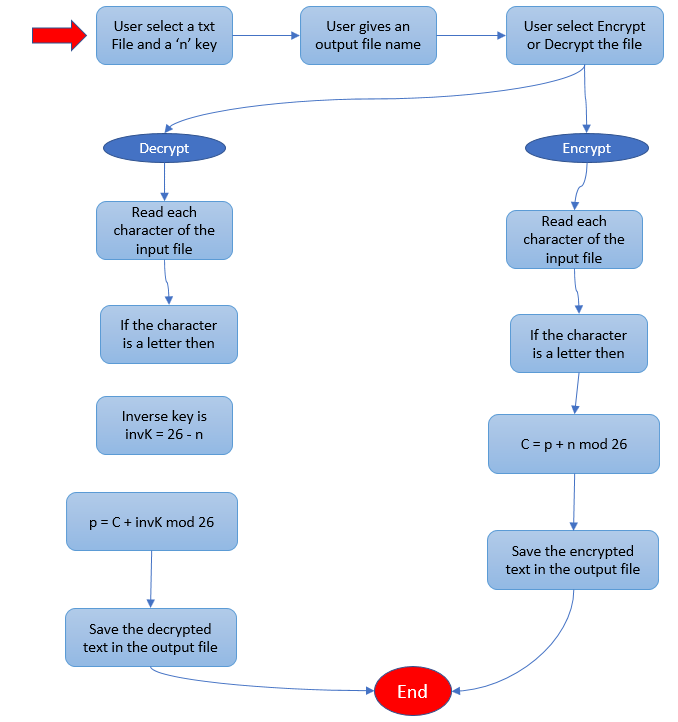


Fig. 3 Block diagram of the procedure of the Shift-Cipher

**For the encryption process:**

The user must select the input file, which is the one that is going to be encrypted. This file must be in lower case. Then the user has to type the key of the algorithm, i.e. the number of jumps. The user also has to type the name of the output file.

The program is going to read each character of the file, and if the character is a lower-case letter, then it is going to apply the formula:

, where

C = encrypted character

p = original character

n = key or number of jumps

When the program has finished to encrypt the original message, it is going to save the result in the output txt file.

**For the decryption process:**

The user must select the input file, which is the one that is going to be encrypted. This file must be in upper case. Then the user has to type the key of the algorithm, i.e. the number of jumps. The user also has to type the name of the output file.

The program is going to read each character of the file, and if the character is an upper-case letter, then it is going to calculate the inverse key, which is necessary to apply the decryption formula. The inverse key is calculated as in , where n is the original key.

When we have the inverse key then we can apply the next formula:

, where

C = encrypted character

p = original character

invK = inverse Key

When the program has finished to encrypt the original message, it is going to save the result in the output txt file.

# Results:

The user interface is shown in Fig. 4. It is the main interface where the user can select the file to encrypt by clicking the Browse button.

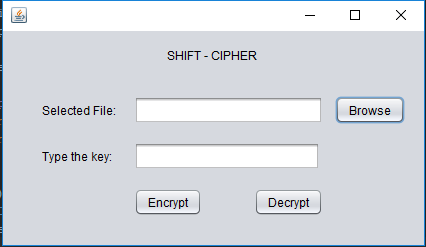


Fig. 4 The main interface of the program.

To make easier the use of the program, I implemented the JFileChooser component of Java (see Fig. 5), it allows the user to navigate through the files in the system and to select the one than he wants.

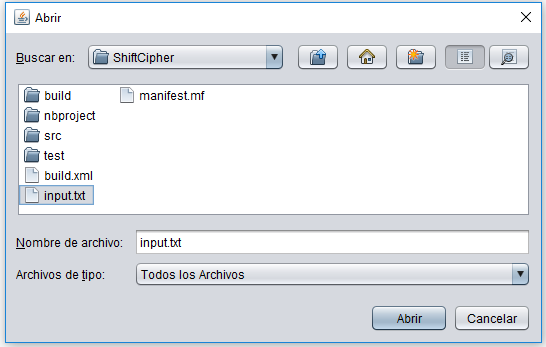


Fig. 5 The JFileChooser implemented in the program

Once the user selects the input file, he has to type the key, or the number of jumps that a letter is going to do. As can be seen in Fig. 6, it’s ready to start the encryption or decryption. In this program the input files are going to be successfully encrypted if they are in lower-case. And if the user wants to decrypt a file, that file has to be in upper-case.

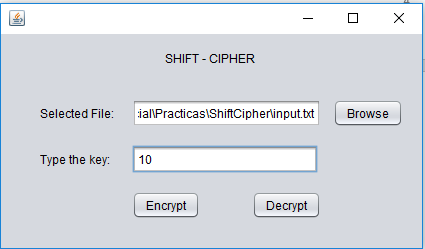


Fig. 6 How the program looks when it’s ready

When the user decide to encrypt the file, he has to select the path and the name of the file that is going to be the output. That JFileChooser is shown in Fig. 7.

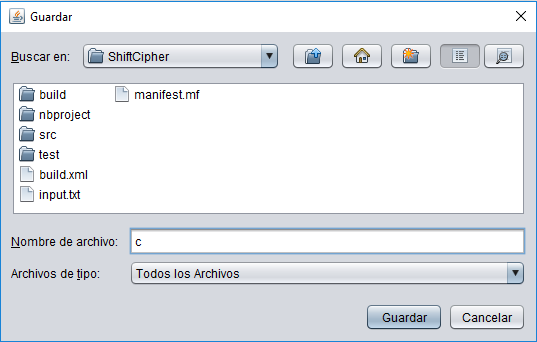


Fig. 7 The JFileChooser that let the user save the output file

When the program finish, it sends an alert to the user saying that everything has been successfully encrypted and save, as can be seen in Fig. 8.

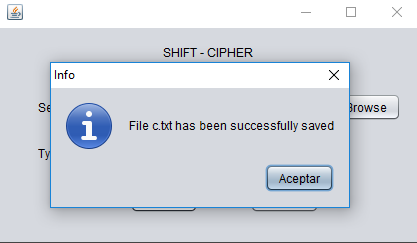


Fig. 8 Success message

The Fig. 9 shows the original text file, the message inside is the IPN anthem. We can see that every single character is a lower-case one, because the program ignores any character that is not a lower-case letter, and if we put an upper-case letter or a special symbol, we are going to have loss of information.

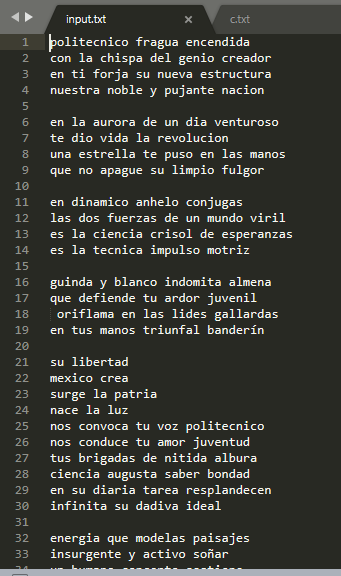


Fig. 9 IPN anthem ready to be encrypted

In the Fig. 10 we can see how the shift-cipher with a 10 key looks. The program omits all the spaces and line breaks and give a unique string. This make the cipher safer, because we cannot identify if we had 2, 3 or 4 letters words.

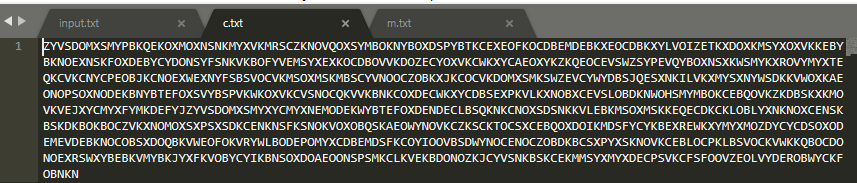


Fig. 10 The result of the encryption with a key of 10

The thing is, that if we want to decrypt the previous file, we cannot achieve a 100% fidelity with the original text, because we have lost the spaces and line breaks, but even if it is a unique string, every person could identify and separate the words to read the original message. This is shown in Fig. 11.

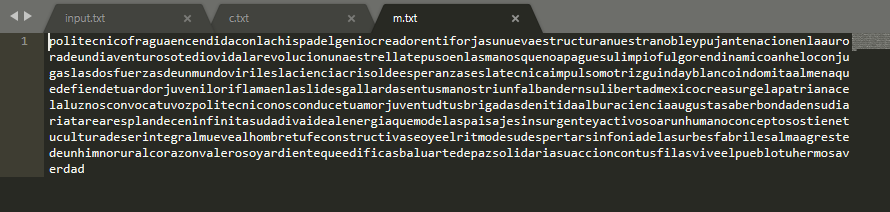


Fig. 11 The result of the decryption

# Discussion:

The algorithm works correctly with any key. Every number bigger than 26 has an equivalent key that is find by calculating for all n > 26. The results shows that we can find the encrypted message as a single string, which is good because if someone intercepts our channel of communication will not be able to understand the word in the first look. The problem I find is that the interceptor could make the inverse process, i.e. he could be able to jump before every single letter of the message and it will take 26 tries to find the original message.

If a person with cryptography knowledge intercepts the message, he could apply the frequency analysis [3], as we have seen before, and he could decrypt the message even faster than the previous interceptor.

To make this algorithm harder to decrypt, The *Substitution Cipher* was invented. [3].

# Conclusions:

Thanks to this practice I had to remember how to use modular arithmetic, because it is the key of the sfhift-cipher algorithm. I realize that it’s not that hard as I remember. I learned how to cipher a message using a substitution algorithm, which can be used to send secret messages between friends in the middle of the class, for example.

One of the things that we had to consider was to find an equivalent key if the user give one bigger than 26. Another possible improvement could be to transform the message received to lower-case, in that case we could ensure that we are not having lost of information, at least for all the letters.

# References:

[1] S. Fernández, "La criptografía clásica", Sigma, no. 24, pp. 119-142, 2004.

[2] J. Ramió, “Seguridad Informática y Criptografía”, Sistemas de Cifra Clásicos, Cap. 9, Universidad Politécnica de Madrid, pp. 343-383, 2006.

[3] N. Smart, Cryptography. London: McGraw-Hill, 2003.

# Code

Class ShiftCipher.java

1. **package** com.shiftcipher;
3. **public** **class** ShiftCipher {
5. **private** String message;
6. **private** **int** key;
7. **private** **final** **int** keyInv;
8. **private** **final** **int** factor1 = 97;
9. **private** **final** **int** factor2 = 65;
10. **private** **final** **int** toUpper = 65;
11. **private** **final** **int** toLow = 97;
13. **public** ShiftCipher(String message, **int** key) {
14. **this**.message = message;
15. **this**.key = key;
16. **this**.keyInv = (26 - key);
17. }
19. **public** String encrypt() {
21. String decMessage = **new** String();
23. **for** (**char** c : message.toCharArray()) {
24. **if** (c >= 97 && c <= 122) {
25. decMessage += (**char**) (((c - factor1 + key) % 26) + toUpper);
26. }
27. }
28. **return** decMessage;
29. }
31. **public** String decrypt() {
33. String decMessage = **new** String();
35. **for** (**char** c : message.toCharArray()) {
36. **if** (c >= 65 && c <= 90) {
37. c -= factor2;
38. c = (**char**) ((c+keyInv)%26);
39. decMessage += (**char**) (c + toLow);
40. }
41. }
42. **return** decMessage;
43. }
45. **public** String getMessage() {
46. **return** message;
47. }
49. **public** **void** setMessage(String message) {
50. **this**.message = message;
51. }
53. **public** **int** getKey() {
54. **return** key;
55. }
57. **public** **void** setKey(**int** key) {
58. **this**.key = key;
59. }
61. }

Class FileManager.java

1. **package** com.filemanager;
3. **import** java.io.BufferedReader;
4. **import** java.io.File;
5. **import** java.io.FileReader;
6. **import** java.io.FileWriter;
7. **import** java.io.IOException;
8. **import** javax.swing.JFileChooser;
9. **import** javax.swing.JOptionPane;
11. **public** **class** FileManager {
13. **private** String dirPath = System.getProperty("user.dir");
14. **private** File inputFile;
15. String sourceData = "";
17. **public** FileManager() {
18. }
20. **public** String openFile() {
21. String aux = "";
23. **try** {
25. JFileChooser fileChooser = **new** JFileChooser(dirPath);
26. fileChooser.showOpenDialog(**null**);
27. inputFile = fileChooser.getSelectedFile();
29. **if** (inputFile != **null**) {
30. FileReader archivos = **new** FileReader(inputFile);
31. BufferedReader br = **new** BufferedReader(archivos);
32. **while** ((aux = br.readLine()) != **null**) {
33. sourceData += aux + "\n";
34. }
35. br.close();
36. }
37. } **catch** (IOException ex) {
38. JOptionPane.showMessageDialog(**null**, ex + ""
39. + "\nNo se ha encontrado el archivo",
40. "ADVERTENCIA!!!", JOptionPane.WARNING\_MESSAGE);
41. }
42. **catch**(NullPointerException n){
44. }
45. **return** sourceData;
46. }
48. **public** **void** saveToFile(String encData) {
49. **try** {
50. JFileChooser fileChooser = **new** JFileChooser(dirPath);
51. fileChooser.showSaveDialog(**null**);
52. File outputFile = fileChooser.getSelectedFile();
54. **if** (outputFile != **null**) {
55. FileWriter fw = **new** FileWriter(outputFile + ".txt");
56. fw.write(encData);
57. fw.close();
58. JOptionPane.showMessageDialog(**null**,
59. "File " + outputFile.getName() + ".txt has been successfully saved",
60. "Info", JOptionPane.INFORMATION\_MESSAGE);
61. }
62. } **catch** (IOException ex) {
63. JOptionPane.showMessageDialog(**null**,
64. "Su archivo no se ha outputFiledo",
65. "Advertencia", JOptionPane.WARNING\_MESSAGE);
66. }
67. }
69. **public** String getFileName() {
70. **return** inputFile.getName();
71. }
73. **public** String getFilePath() {
74. **try** {
75. **return** inputFile.getAbsolutePath();
76. } **catch** (Exception e) {
77. }
78. **return** **null**;
79. }
80. }

Class MainFrame.java

1. **package** com.gui;
3. **import** com.shiftcipher.ShiftCipher;
4. **import** com.filemanager.FileManager;
5. **import** java.util.logging.Level;
6. **import** java.util.logging.Logger;
7. **import** javax.swing.JOptionPane;
9. **public** **class** MainFrame **extends** javax.swing.JFrame {
11. **private** String inputText = "";
12. String encData = **null**;
13. String decData = **null**;

16. **public** MainFrame() {
17. setLocation(200, 100);
18. initComponents();
19. }
21. /\*\*
22. \* This method is called from within the constructor to initialize the form.
23. \* WARNING: Do NOT modify this code. The content of this method is always
24. \* regenerated by the Form Editor.
25. \*/
26. @SuppressWarnings("unchecked")
27. // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:initComponents
28. **private** **void** initComponents() {
30. titleLabel = **new** javax.swing.JLabel();
31. inputTextField = **new** javax.swing.JTextField();
32. browseButton = **new** javax.swing.JButton();
33. jLabel1 = **new** javax.swing.JLabel();
34. encryptButton = **new** javax.swing.JButton();
35. decryptButton = **new** javax.swing.JButton();
36. jLabel2 = **new** javax.swing.JLabel();
37. keyTextField = **new** javax.swing.JTextField();
39. setDefaultCloseOperation(javax.swing.WindowConstants.EXIT\_ON\_CLOSE);
41. titleLabel.setText("SHIFT - CIPHER");
43. inputTextField.setEditable(**false**);
44. inputTextField.addActionListener(**new** java.awt.event.ActionListener() {
45. **public** **void** actionPerformed(java.awt.event.ActionEvent evt) {
46. inputTextFieldActionPerformed(evt);
47. }
48. });
50. browseButton.setText("Browse");
51. browseButton.addActionListener(**new** java.awt.event.ActionListener() {
52. **public** **void** actionPerformed(java.awt.event.ActionEvent evt) {
53. browseButtonActionPerformed(evt);
54. }
55. });
57. jLabel1.setText("Selected File:");
59. encryptButton.setText("Encrypt");
60. encryptButton.addActionListener(**new** java.awt.event.ActionListener() {
61. **public** **void** actionPerformed(java.awt.event.ActionEvent evt) {
62. encryptButtonActionPerformed(evt);
63. }
64. });
66. decryptButton.setText("Decrypt");
67. decryptButton.addActionListener(**new** java.awt.event.ActionListener() {
68. **public** **void** actionPerformed(java.awt.event.ActionEvent evt) {
69. decryptButtonActionPerformed(evt);
70. }
71. });
73. jLabel2.setText("Type the key:");
75. keyTextField.addActionListener(**new** java.awt.event.ActionListener() {
76. **public** **void** actionPerformed(java.awt.event.ActionEvent evt) {
77. keyTextFieldActionPerformed(evt);
78. }
79. });
81. javax.swing.GroupLayout layout = **new** javax.swing.GroupLayout(getContentPane());
82. getContentPane().setLayout(layout);
83. layout.setHorizontalGroup(
84. layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
85. .addGroup(layout.createSequentialGroup()
86. .addGap(39, 39, 39)
87. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
88. .addComponent(jLabel1)
89. .addComponent(jLabel2))
90. .addGap(18, 18, 18)
91. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
92. .addGroup(layout.createSequentialGroup()
93. .addComponent(keyTextField)
94. .addGap(104, 104, 104))
95. .addGroup(layout.createSequentialGroup()
96. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
97. .addGroup(layout.createSequentialGroup()
98. .addComponent(encryptButton)
99. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 52, Short.MAX\_VALUE)
100. .addComponent(decryptButton))
101. .addComponent(inputTextField, javax.swing.GroupLayout.Alignment.LEADING))
102. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
103. .addComponent(browseButton)
104. .addGap(19, 19, 19))))
105. .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()
106. .addContainerGap(javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)
107. .addComponent(titleLabel)
108. .addGap(167, 167, 167))
109. );
110. layout.setVerticalGroup(
111. layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
112. .addGroup(layout.createSequentialGroup()
113. .addGap(16, 16, 16)
114. .addComponent(titleLabel)
115. .addGap(33, 33, 33)
116. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
117. .addComponent(inputTextField, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)
118. .addComponent(jLabel1)
119. .addComponent(browseButton))
120. .addGap(18, 18, 18)
121. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
122. .addComponent(jLabel2)
123. .addComponent(keyTextField, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE))
124. .addGap(18, 18, 18)
125. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
126. .addComponent(encryptButton)
127. .addComponent(decryptButton))
128. .addContainerGap(29, Short.MAX\_VALUE))
129. );
131. pack();
132. }// </editor-fold>//GEN-END:initComponents
134. **private** **void** inputTextFieldActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_inputTextFieldActionPerformed
135. // TODO add your handling code here:
136. }//GEN-LAST:event\_inputTextFieldActionPerformed
138. **private** **void** browseButtonActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_browseButtonActionPerformed
139. FileManager fm = **new** FileManager();
140. inputText = fm.openFile();
141. inputTextField.setText(fm.getFilePath());
142. }//GEN-LAST:event\_browseButtonActionPerformed
144. **private** **void** encryptButtonActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_encryptButtonActionPerformed
145. **if** (!inputTextField.getText().isEmpty()&& !keyTextField.getText().isEmpty()) {
146. **int** keyValue = Integer.parseInt(keyTextField.getText());
147. **if** (keyValue > 26){
148. keyValue = keyValue % 26;
149. JOptionPane.showMessageDialog(**this**, "The key value is invalid. The equivalent key is "+ keyValue, "Info", JOptionPane.INFORMATION\_MESSAGE, **null**);
150. }
151. **try** {
152. ShiftCipher c = **new** ShiftCipher(inputText, keyValue);
153. encData = c.encrypt();
154. FileManager fm = **new** FileManager();
155. fm.saveToFile(encData);
156. } **catch** (Exception ex) {
157. Logger.getLogger(MainFrame.**class**.getName()).log(Level.SEVERE, **null**, ex);
158. }
159. clearForm();
161. } **else** {
162. JOptionPane.showMessageDialog(**this**, "Select an input file and give a key value", "Error", JOptionPane.ERROR\_MESSAGE, **null**);
163. }
164. }//GEN-LAST:event\_encryptButtonActionPerformed
166. **private** **void** decryptButtonActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_decryptButtonActionPerformed
167. **if** (!inputTextField.getText().isEmpty() && !keyTextField.getText().isEmpty()) {
168. **int** keyValue = Integer.parseInt(keyTextField.getText());
169. **if** (keyValue > 26){
170. keyValue = keyValue % 26;
171. JOptionPane.showMessageDialog(**this**, "The key value is invalid. The equivalent key is "+ keyValue, "Info", JOptionPane.INFORMATION\_MESSAGE, **null**);
172. }
173. **try** {
174. ShiftCipher c = **new** ShiftCipher(inputText, keyValue);
175. decData = c.decrypt();
176. FileManager fm = **new** FileManager();
177. fm.saveToFile(decData);
178. } **catch** (Exception ex) {
179. Logger.getLogger(MainFrame.**class**.getName()).log(Level.SEVERE, **null**, ex);
180. }
181. clearForm();
183. } **else** {
184. JOptionPane.showMessageDialog(**this**, "Select an input file and give a key value", "Error", JOptionPane.ERROR\_MESSAGE, **null**);
185. }
186. }//GEN-LAST:event\_decryptButtonActionPerformed
188. **public** **void** clearForm(){
189. inputTextField.setText("");
190. keyTextField.setText("");
191. }
193. **private** **void** keyTextFieldActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_keyTextFieldActionPerformed
194. // TODO add your handling code here:
195. }//GEN-LAST:event\_keyTextFieldActionPerformed
197. /\*\*
198. \* @param args the command line arguments
199. \*/
200. **public** **static** **void** main(String args[]) {
201. /\* Set the Nimbus look and feel \*/
202. //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
203. /\* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
204. \* For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
205. \*/
206. **try** {
207. **for** (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {
208. **if** ("Nimbus".equals(info.getName())) {
209. javax.swing.UIManager.setLookAndFeel(info.getClassName());
210. **break**;
211. }
212. }
213. } **catch** (ClassNotFoundException ex) {
214. java.util.logging.Logger.getLogger(MainFrame.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
215. } **catch** (InstantiationException ex) {
216. java.util.logging.Logger.getLogger(MainFrame.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
217. } **catch** (IllegalAccessException ex) {
218. java.util.logging.Logger.getLogger(MainFrame.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
219. } **catch** (javax.swing.UnsupportedLookAndFeelException ex) {
220. java.util.logging.Logger.getLogger(MainFrame.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
221. }
222. //</editor-fold>
224. /\* Create and display the form \*/
225. java.awt.EventQueue.invokeLater(**new** Runnable() {
226. **public** **void** run() {
227. **new** MainFrame().setVisible(**true**);
228. }
229. });
230. }
232. // Variables declaration - do not modify//GEN-BEGIN:variables
233. **private** javax.swing.JButton browseButton;
234. **private** javax.swing.JButton decryptButton;
235. **private** javax.swing.JButton encryptButton;
236. **private** javax.swing.JTextField inputTextField;
237. **private** javax.swing.JLabel jLabel1;
238. **private** javax.swing.JLabel jLabel2;
239. **private** javax.swing.JTextField keyTextField;
240. **private** javax.swing.JLabel titleLabel;
241. // End of variables declaration//GEN-END:variables
242. }