



INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO



Cryptography

"Affine Cipher"

Abstract

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

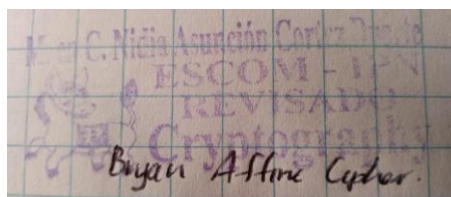
By:

Bryan Dominguez de la Rosa

Professor:

MSc. NIDIA ASUNCIÓN CORTEZ DUARTE

September 2018



Index

Contenido

Introduction:.....	1
Literature review:	1
Software (libraries, packages, tools):	2
Procedure:.....	3
Results:	4
Discussion:	8
Conclusions:.....	8
References:	8
Code.....	8

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 Affine-Cipher algorithm is to let the user select alpha and betha parameters. Alpha is going to multiplicate each letter of the message and betha is going to make the jumps, as in the Caesar algorithm. If alpha is 5 and betha is 10, that means that every letter of the message is going to be multiplied by 5 and then added with 10, the result is going to be applied with mod 26, because the English alphabet has only 26 letters.

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]:

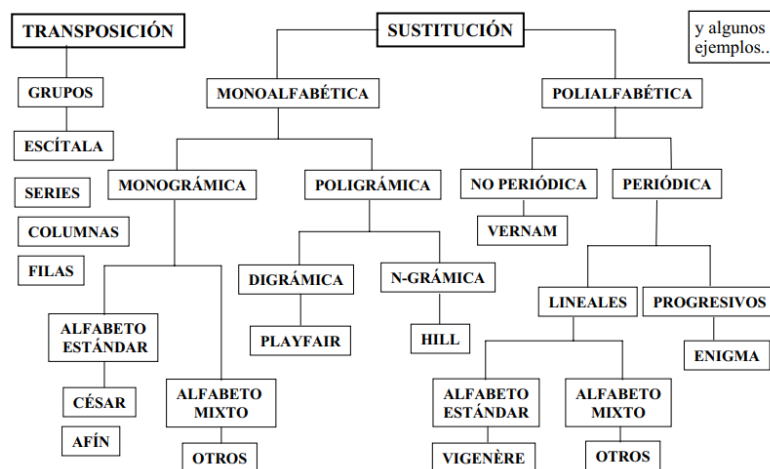


Fig. 1 Classification of classic cryptosystems

As can be seen, the main division is the transposition and substitution algorithms. The Affine 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 Affine Cipher, the encryption is performed by replacing each letter with another a number of places after (or before) 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 an affine cipher example

Software (libraries, packages, tools):

I use Java programming 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:

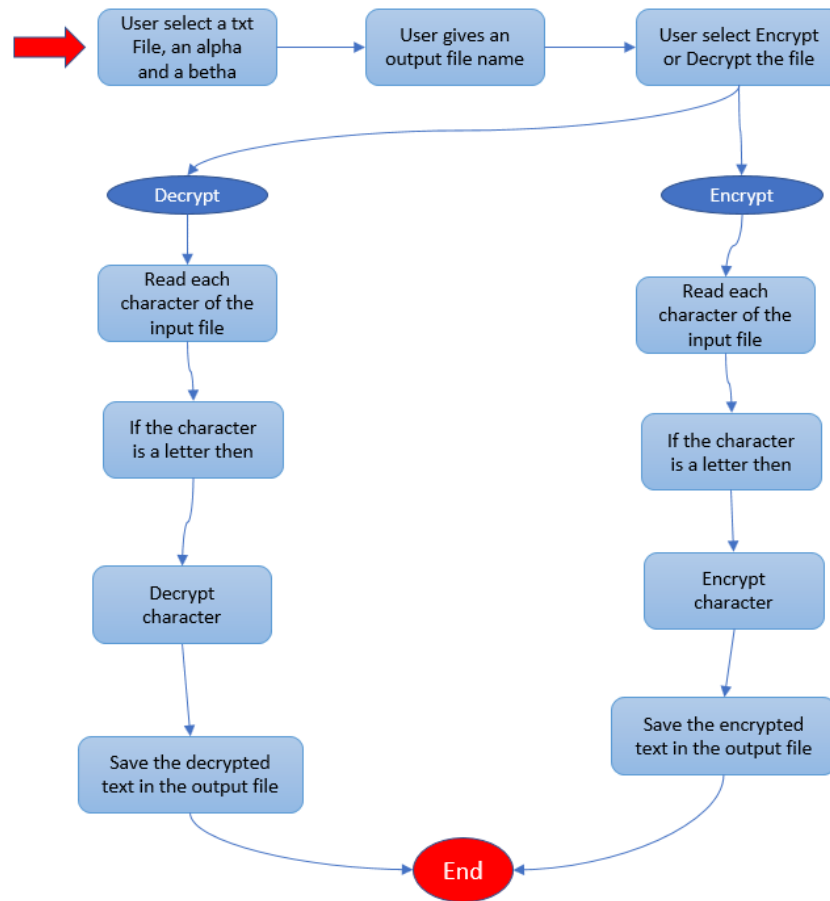


Fig. 3 Block diagram of the procedure of the Affine Cipher

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 select the alpha and type the betha value of the algorithm. 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:

$$C = (\alpha * p) + \beta \text{ mod } 26, \text{ where}$$

C = encrypted character

α = multiplicative value

p = original character

β = additive value

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 decrypted. This file must be in upper case. Then the user has to select the alpha and type the betha value of the algorithm. 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 alpha and inverse betha, that are necessary to apply the decryption formula. The inverse alpha means:

$$\alpha^{-1} * \alpha = 1$$

Then we have to calculate inverse betha, which means:

$$\beta^{-1} + \beta = 26$$

When we have the inverse values of alpha and betha we can apply the next formula:

$$p = \alpha^{-1}(C + \beta^{-1}) \bmod 26, \text{ where}$$

p = original character

α^{-1} = inverse multiplicative value

C = encrypted character

β^{-1} = inverse additive value

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.

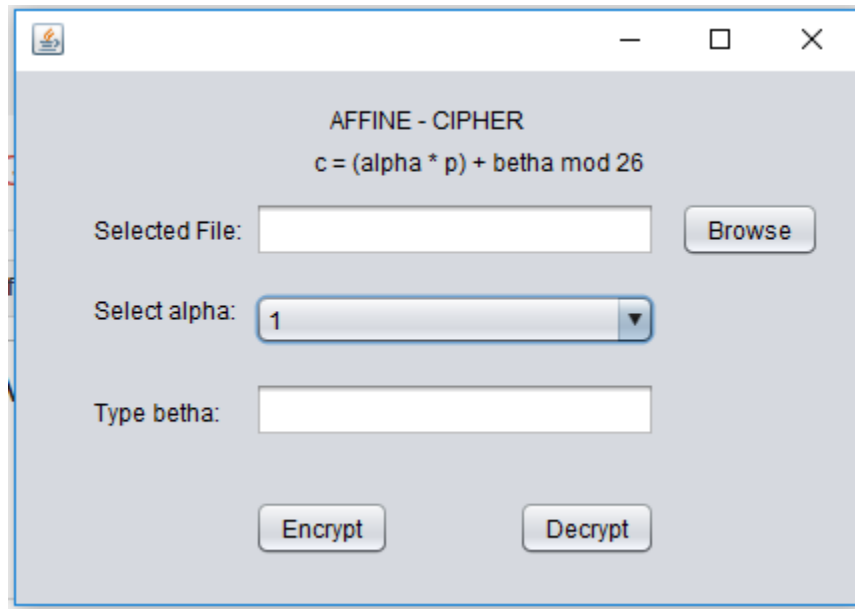


Fig. 4 The main interface of the program.

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.

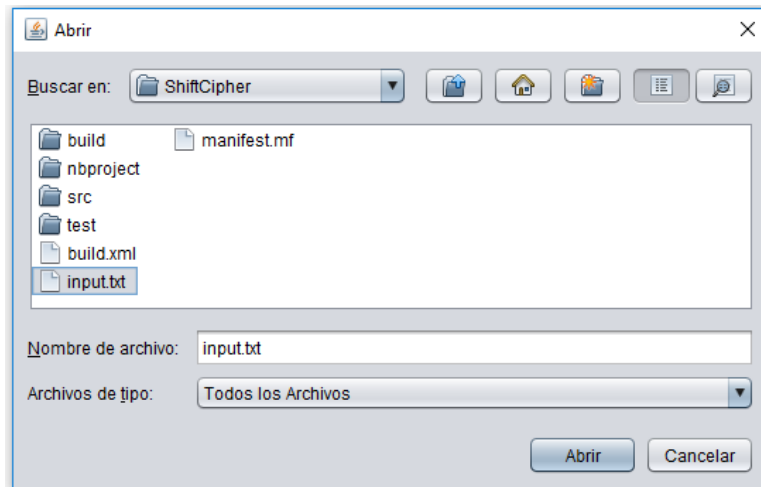


Fig. 5 The JFileChooser implemented in the program

Once the user selects the input file, he has to select the alpha and type a betha. 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.

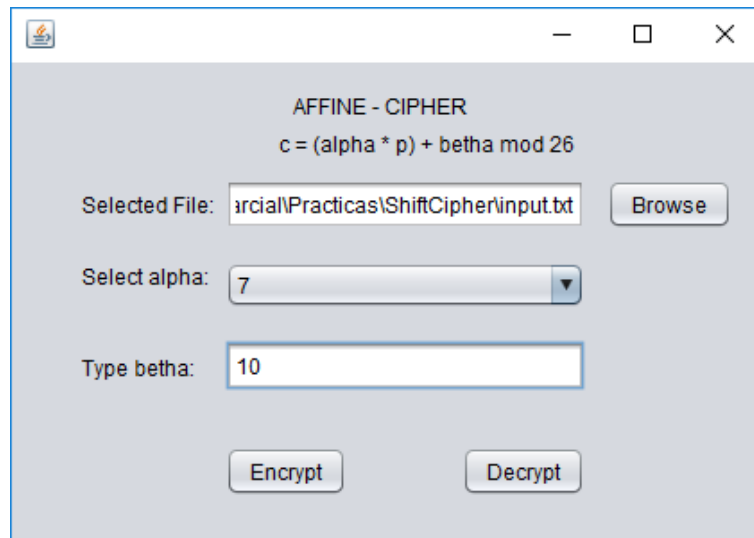


Fig. 6 How the program looks when it's ready

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.

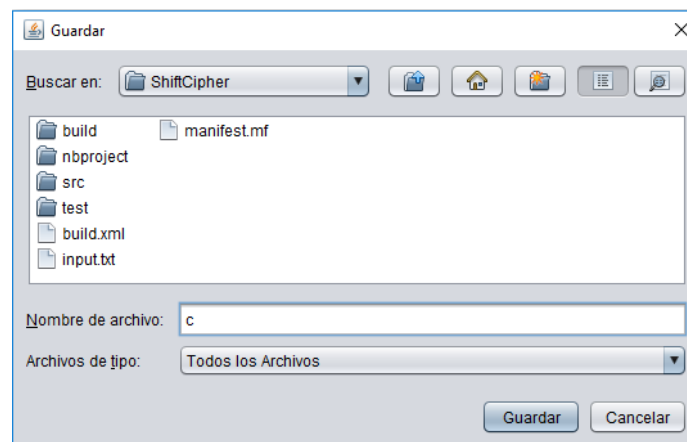


Fig. 7 The JFileChooser that let the user save the output file

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.

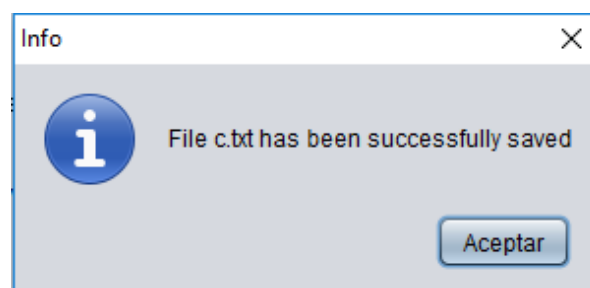
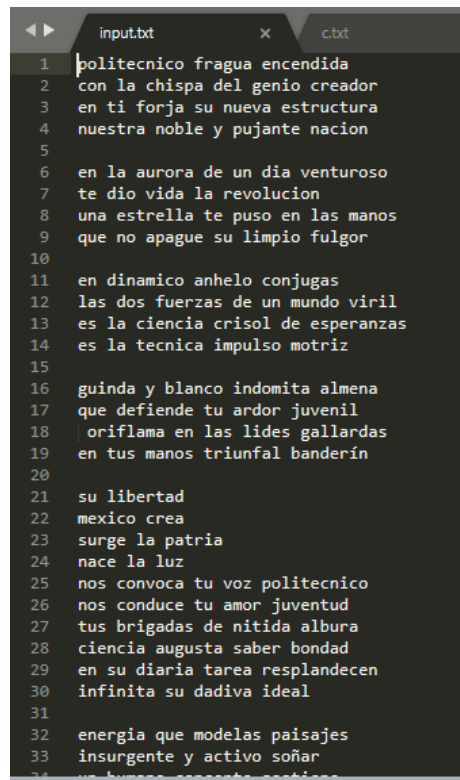


Fig. 8 Success message

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.



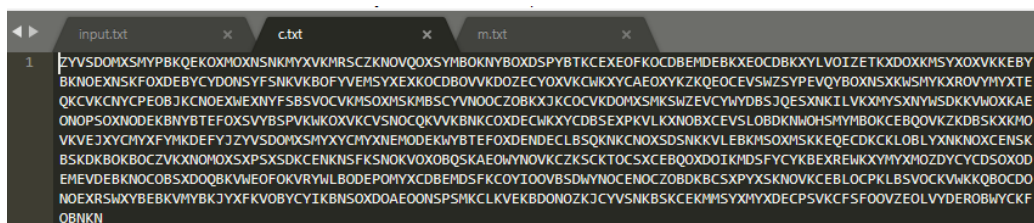
```

1 politecnico fragua encendida
2 con la chispa del genio creador
3 en ti forja su nueva estructura
4 nuestra noble y pujante nacion
5
6 en la aurora de un dia venturoso
7 te dio vida la revolucion
8 una estrella te puso en las manos
9 que no apague su limpio fulgor
10
11 en dinamico anhelo conjugas
12 las dos fuerzas de un mundo viril
13 es la ciencia crisol de esperanzas
14 es la tecnica impulso motriz
15
16 guinda y blanco indomita almena
17 que defiende tu ardor juvenil
18 oriflama en las lides gallardas
19 en tus manos triunfal banderín
20
21 su libertad
22 mexico crea
23 surge la patria
24 nace la luz
25 nos convoca tu voz politecnico
26 nos conduce tu amor juventud
27 tus brigadas de nitida albura
28 ciencia augusta saber bondad
29 en su diaria tarea resplandecen
30 infinita su dadiva ideal
31
32 energia que modelas paisajes
33 insurgente y activo soñar
34 un futuro concreto certidura

```

Fig. 9 IPN anthem ready to be encrypted

In the Fig. 10 we can see how the affine cipher 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.



```

1 zyvSDOMXSMYPBKQEKOMXNSNMXYVKMRSCZKNNOVQOXSVMBOKNYBOXDSPYBTKCEXEOKOCDBEMDEBKXEOCDBKXYLVOIZETKXDOXKMSYXOXVKKKEBY
BKNOEXNSKFOXDEBYCYDONSYFSNKVKBOFYVMSYEXKOCDBOVVKDOZECYXVKKXCYCAEOXYKZKQEOCEVSWZSYPEVQYBOXNSXKMSMYKXROVYHYXTE
QKCVKCNYPCEOBJKCNOEXNEXNYFSBSVOCVKMSOXMSKMBSCYVNOOCZOBKXJCKOCVKDOMXSMKSWZEVCMYDBSJQESXNKILVXOMYSXNYMSDKKVWXXKAE
ONOPSXNODKBNYBTEFOXSVYBSPVKXOVKVCVNOCQKVVKBKNCXOXCENKXCYCDBSEXPVLLKXNOBXCEVSLQBDKNWQHSMMYBOKCEBQOVKZKDBSKXKMO
VKVEJXYCMYFYMKDEFYJZYVSDOMXSMYXCMYXNEMODEKMYBTEFOXDENDECLBSQKNKCNOXSDSNKKVLEBKMSOXMSKKEQECCKLOBLXNKNNOXCENSK
BSKDKBOKBOCZVIXNOMOXSPXSXDKCENKNSFKSNOKVOXOBQSKAEOWYNNOVKCZSKCTOCSXCEBQOXDOIKMDSFYCYKBEXREWXYMYXMOZDYCYCDSOXOD
EMEVDKBNOCBSXDOQBKVWEFOKVRVYWLBODEPOMYXCDHEMDSFKCOYIOOVBSOWYNOCENOCZOBKBCSXYPYXSKNOVKCEBLOCPKLBSVOCKVNNKQBOCDO
NOEXRSWXYBEBKVMYBKJYXFKVOBYCYIKBNSOXDOAEONSPSHKCLKVEKBDOXKJCYVSNBKSCEKMMSYXMYXDECPVKCFSFQOVZEOVLVYDEROBWYCKF
OBNKN

```

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.

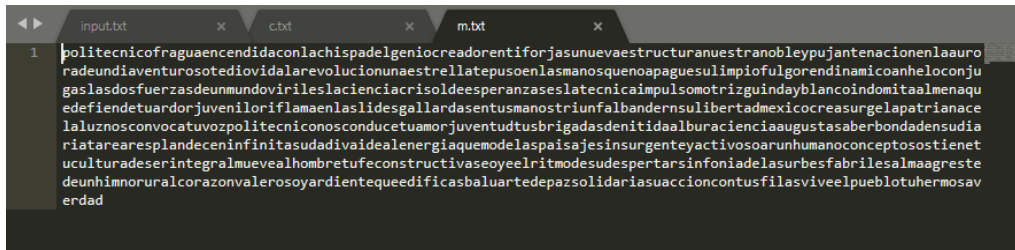


Fig. 11 The result of the decryption

Discussion:

The algorithm works correctly when we select and alpha that is coprime with 26 (or the alphabet that we are using), that means that there is no relation between alpha and the length of the alphabet. The results show 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. This is an encryption easy to do manually, but it's not as safer as the modern algorithms.

Conclusions:

Thanks to this practice I had to remember how to use modular arithmetic, because it is the key knowledge of the affine cipher algorithm. I realize that it's not that hard as I remembered. 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 select an alpha that where coprime with the alphabet. 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 AffineCipher.java

```
1. package com.affinecipher;
2.
```

```

3. public class AffineCipher {
4.
5.     private String message;
6.     private final int factor1 = 97;
7.     private final int factor2 = 65;
8.     private final int toUpper = 65;
9.     private final int toLow = 97;
10.    private final int alpha;
11.    private int inverseAlpha;
12.    private final int betha;
13.    private int inverseBetha;
14.
15.    public AffineCipher(String message, int alpha, int betha) {
16.        this.message = message;
17.        this.alpha = alpha;
18.        this.betha = betha;
19.        this.inverseAlpha = doInverseAlpha();
20.        this.inverseBetha = doInverseBetha();
21.    }
22.
23.    private int doInverseAlpha() {
24.        int alphabet = 26;
25.        int res;
26.        for (int i = 0; i < alphabet; i++) {
27.            res = (alpha * i) % alphabet;
28.            if (res == 1) {
29.                inverseAlpha = i;
30.                break;
31.            }
32.        }
33.        return inverseAlpha;
34.    }
35.
36.    private int doInverseBetha() {
37.        inverseBetha = 26 - (betha % 26);
38.        return inverseBetha;
39.    }
40.
41.    public String encrypt() {
42.
43.        String encMessage = new String();
44.
45.        for (char c : message.toCharArray()) {
46.            if (c > 96 && c < 123) {
47.                c -= factor1;
48.                c = (char) (((alpha * c) + betha) % 26);
49.                c += toUpper;
50.                encMessage += c;
51.            }
52.        }
53.
54.        return encMessage;
55.    }
56.
57.    public String decrypt() {
58.
59.        String decMessage = new String();
60.
61.        for (char c : message.toCharArray()) {
62.            if (c > 64 && c < 91) {
63.                c -= factor2;

```

```

64.         c = (char) ((inverseAlpha * (c + inverseBeta)) % 26);
65.         c += toLow;
66.         decMessage += c;
67.     }
68. }
69.     return decMessage;
70. }
71.
72. public String getMessage() {
73.     return message;
74. }
75.
76. public void setMessage(String message) {
77.     this.message = message;
78. }
79.
80. }

```

Class FileManager.java

```

1. package com.filemanager;
2.
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;
10.
11. public class FileManager {
12.
13.     private String dirPath = System.getProperty("user.dir");
14.     private File inputFile;
15.     String sourceData = "";
16.
17.     public FileManager() {
18.     }
19.
20.     public String openFile() {
21.         String aux = "";
22.
23.         try {
24.
25.             JFileChooser fileChooser = new JFileChooser(dirPath);
26.             fileChooser.showOpenDialog(null);
27.             inputFile = fileChooser.getSelectedFile();
28.
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){
43.
44.     }
45.     return sourceData;
46. }
47.
48. public void saveToFile(String encData) {
49.     try {
50.         JFileChooser fileChooser = new JFileChooser(dirPath);
51.         fileChooser.showSaveDialog(null);
52.         File outputFile = fileChooser.getSelectedFile();
53.
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 successful
ly 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. }
68.
69. public String getFileName() {
70.     return inputFile.getName();
71. }
72.
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;
2.
3. import com.affinecipher.AffineCipher;
4. import com.filemanager.FileManager;
5. import java.util.logging.Level;
6. import java.util.logging.Logger;
7. import javax.swing.JOptionPane;
8.
9. public class MainFrame extends javax.swing.JFrame {
10.
11.     private String inputText = "";
12.     String encData = null;
13.     String decData = null;
14.     char validBeta[] = {1,3,5,7,9,11,15,17,19,21,23,25};

```

```

15.
16.
17.     public MainFrame() {
18.         setLocation(200, 100);
19.         initComponents();
20.     }
21.
22.     /**
23.      * This method is called from within the constructor to initialize the form.
24.      * WARNING: Do NOT modify this code. The content of this method is always
25.      * regenerated by the Form Editor.
26.      */
27.     @SuppressWarnings("unchecked")
28.     // <editor-fold defaultstate="collapsed" desc="Generated Code">
29.     private void initComponents() {
30.
31.         titleLabel = new javax.swing.JLabel();
32.         inputTextField = new javax.swing.JTextField();
33.         browseButton = new javax.swing.JButton();
34.         jLabel1 = new javax.swing.JLabel();
35.         encryptButton = new javax.swing.JButton();
36.         decryptButton = new javax.swing.JButton();
37.         jLabel2 = new javax.swing.JLabel();
38.         bethaField = new javax.swing.JTextField();
39.         jLabel3 = new javax.swing.JLabel();
40.         alphaBox = new javax.swing.JComboBox<>();
41.         jLabel4 = new javax.swing.JLabel();
42.
43.         setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
44.
45.         titleLabel.setText("AFFINE - CIPHER");
46.
47.         inputTextField.setEditable(false);
48.         inputTextField.addActionListener(new java.awt.event.ActionListener() {
49.             public void actionPerformed(java.awt.event.ActionEvent evt) {
50.                 inputTextFieldActionPerformed(evt);
51.             }
52.         });
53.
54.         browseButton.setText("Browse");
55.         browseButton.addActionListener(new java.awt.event.ActionListener() {
56.             public void actionPerformed(java.awt.event.ActionEvent evt) {
57.                 browseButtonActionPerformed(evt);
58.             }
59.         });
60.
61.         jLabel1.setText("Selected File:");
62.
63.         encryptButton.setText("Encrypt");
64.         encryptButton.addActionListener(new java.awt.event.ActionListener() {
65.             public void actionPerformed(java.awt.event.ActionEvent evt) {
66.                 encryptButtonActionPerformed(evt);
67.             }
68.         });
69.
70.         decryptButton.setText("Decrypt");
71.         decryptButton.addActionListener(new java.awt.event.ActionListener() {
72.             public void actionPerformed(java.awt.event.ActionEvent evt) {
73.                 decryptButtonActionPerformed(evt);
74.             }

```



```

123.                .addGap(149, 149, 149)
124.                .addComponent(jLabel14, javax.swing.GroupLayout.PREFERRED_S
125.                IZE, 199, javax.swing.GroupLayout.PREFERRED_SIZE)
126.                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Sho
127.                rt.MAX_VALUE))
128.                );
129.                layout.setVerticalGroup(
130.                layout.createParallelGroup(javax.swing.GroupLayout.Alignment.L
131.                EADING)
132.                .addGroup(layout.createSequentialGroup())
133.                .addGap(16, 16, 16)
134.                .addComponent(titleLabel)
135.                .addGap(5, 5, 5)
136.                .addComponent(jLabel14)
137.                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacemen
138.                t.UNRELATED)
139.                .addGroup(layout.createParallelGroup(javax.swing.GroupLayo
140.                ut.Alignment.BASELINE)
141.                .addComponent(inputTextField, javax.swing.GroupLayout.PREFER
142.                RED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREF
143.                ERRED_SIZE)
144.                .addComponent(jLabel11)
145.                .addComponent(browseButton))
146.                .addGap(18, 18, 18)
147.                .addGroup(layout.createParallelGroup(javax.swing.GroupLayo
148.                ut.Alignment.LEADING)
149.                .addComponent(jLabel12)
150.                .addComponent(alphaBox, javax.swing.GroupLayout.PREFER
151.                RED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_
152.                SIZE))
153.                .addGroup(layout.createParallelGroup(javax.swing.GroupLayo
154.                ut.Alignment.LEADING)
155.                .addGroup(layout.createSequentialGroup())
156.                .addPreferredGap(javax.swing.LayoutStyle.Component
157.                Placement.RELATED, 25, Short.MAX_VALUE)
158.                .addComponent(jLabel13)
159.                .addGap(36, 36, 36))
160.                .addGroup(layout.createSequentialGroup())
161.                .addGap(18, 18, 18)
162.                .addComponent(bethaField, javax.swing.GroupLayout.PREFER
163.                RED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREF
164.                ERRED_SIZE)
165.                .addPreferredGap(javax.swing.LayoutStyle.Component
166.                Placement.RELATED, javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)))
167.                .addGroup(layout.createParallelGroup(javax.swing.GroupLayo
168.                ut.Alignment.BASELINE)
169.                .addComponent(encryptButton)
170.                .addComponent(decryptButton))
171.                .addGap(24, 24, 24))
172.                );
173.                pack();
174.            } // </editor-fold>
175.
176.            private void inputTextFieldActionPerformed(java.awt.event.ActionEvent
177.            evt) {
178.                // TODO add your handling code here:
179.            }
180.
181.            private void browseButtonActionPerformed(java.awt.event.ActionEvent ev
182.            t) {

```



```

166.         FileManager fm = new FileManager();
167.         inputText = fm.openFile();
168.         inputTextField.setText(fm.getFilePath());
169.     }
170.
171.     private void encryptButtonActionPerformed(java.awt.event.ActionEvent e
    vt) {
172.         if (!inputTextField.getText().isEmpty() && !bethaField.getText().is
    Empty()) {
173.             int alpha = Integer.parseInt(alphaBox.getSelectedItem().toStri
    ng());
174.             int betha = Integer.parseInt(bethaField.getText());
175.             if (alpha > 26){
176.                 alpha = alpha % 26;
177.                 JOptionPane.showMessageDialog(this, "The alpha value is in
    valid. The equivalent alpha is " + alpha, "Info", JOptionPane.INFORMATION_MESSAGE,
    null);
178.             }
179.             try {
180.                 AffineCipher c = new AffineCipher(inputText, alpha, betha)
    ;
181.                 encData = c.encrypt();
182.                 FileManager fm = new FileManager();
183.                 fm.saveToFile(encData);
184.             } catch (Exception ex) {
185.                 Logger.getLogger(MainFrame.class.getName()).log(Level.SEVERE,
    RE, null, ex);
186.             }
187.             clearForm();
188.
189.         } else {
190.             JOptionPane.showMessageDialog(this, "Select an input file, giv
    e an alpha and a betha value", "Error", JOptionPane.ERROR_MESSAGE, null);
191.         }
192.     }
193.
194.     private void decryptButtonActionPerformed(java.awt.event.ActionEvent e
    vt) {
195.         if (!inputTextField.getText().isEmpty() && !bethaField.getText().is
    Empty()) {
196.             int alpha = Integer.parseInt(alphaBox.getSelectedItem().toStri
    ng());
197.             int betha = Integer.parseInt(bethaField.getText());
198.             System.out.println(betha);
199.             if (alpha > 26){
200.                 alpha = alpha % 26;
201.                 JOptionPane.showMessageDialog(this, "The key value is inva
    lid. The equivalent key is " + alpha, "Info", JOptionPane.INFORMATION_MESSAGE, null
    );
202.             }
203.             try {
204.                 AffineCipher c = new AffineCipher(inputText, alpha, betha)
    ;
205.                 decData = c.decrypt();
206.                 FileManager fm = new FileManager();
207.                 fm.saveToFile(decData);
208.             } catch (Exception ex) {
209.                 Logger.getLogger(MainFrame.class.getName()).log(Level.SEVERE,
    RE, null, ex);
210.             }
211.             clearForm();

```

```

212.
213.         } else {
214.             JOptionPane.showMessageDialog(this, "Select an input file, give an alpha and a betha value", "Error", JOptionPane.ERROR_MESSAGE, null);
215.         }
216.     }
217.
218.     public void clearForm(){
219.         inputTextField.setText("");
220.         bethaField.setText("");
221.     }
222.
223.     private void bethaFieldActionPerformed(java.awt.event.ActionEvent evt)
224.     {
225.         // TODO add your handling code here:
226.     }
227.
228.     private void alphaBoxActionPerformed(java.awt.event.ActionEvent evt) {
229.         // TODO add your handling code here:
230.     }
231.
232.     /**
233.      * @param args the command line arguments
234.      */
235.     public static void main(String args[]) {
236.         /* Set the Nimbus look and feel */
237.         //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
238.         /* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
239.          * For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
240.          */
241.         try {
242.             for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {
243.                 if ("Nimbus".equals(info.getName())) {
244.                     javax.swing.UIManager.setLookAndFeel(info.getClassName());
245.                     break;
246.                 }
247.             } catch (ClassNotFoundException ex) {
248.                 java.util.logging.Logger.getLogger(MainFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
249.             } catch (InstantiationException ex) {
250.                 java.util.logging.Logger.getLogger(MainFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
251.             } catch (IllegalAccessException ex) {
252.                 java.util.logging.Logger.getLogger(MainFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
253.             } catch (javax.swing.UnsupportedLookAndFeelException ex) {
254.                 java.util.logging.Logger.getLogger(MainFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
255.             }
256.         } //</editor-fold>
257.
258.         /* Create and display the form */
259.         java.awt.EventQueue.invokeLater(new Runnable() {
260.             public void run() {

```

```
261.         new MainFrame().setVisible(true);
262.     }
263.     });
264. }
265.
266.     // Variables declaration - do not modify
267.     private javax.swing.JComboBox<String> alphaBox;
268.     private javax.swing.JTextField bethaField;
269.     private javax.swing.JButton browseButton;
270.     private javax.swing.JButton decryptButton;
271.     private javax.swing.JButton encryptButton;
272.     private javax.swing.JTextField inputTextField;
273.     private javax.swing.JLabel jLabel1;
274.     private javax.swing.JLabel jLabel2;
275.     private javax.swing.JLabel jLabel3;
276.     private javax.swing.JLabel jLabel4;
277.     private javax.swing.JLabel titleLabel;
278.     // End of variables declaration
279. }
```