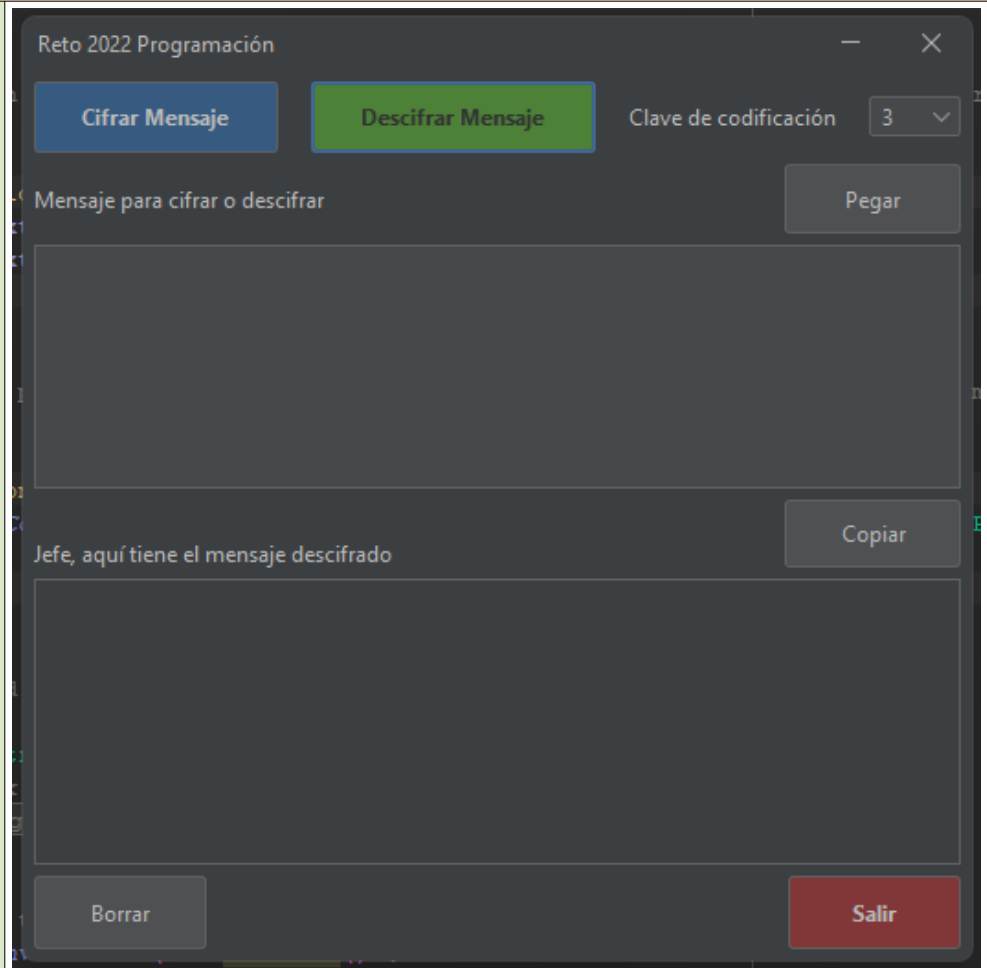
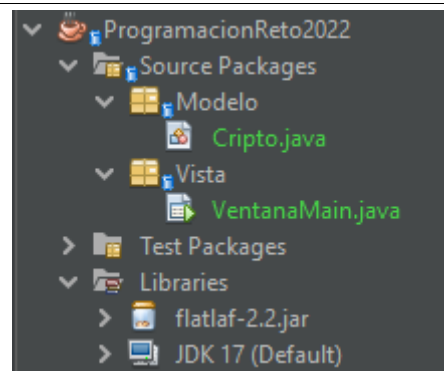


Reto de programación 2022

Aplicación →



El programa esta compuesto por dos clases Cripto y VentanaMain



Como elemento externo cabe mencionar que se ha utilizado las librerías Open Source llamadas flatlaf 2.2 que son un estilo lookAndFeel.

Url: <https://www.formdev.com/flatlaf/>

The screenshot shows the FlatLaf website for FormDev Software. The main heading is "FlatLaf - Flat Look and Feel". Below it, there's a "Star 1,511" badge and a description: "FlatLaf is a modern open-source cross-platform Look and Feel for Java Swing desktop applications. It looks almost flat (no shadows or gradients), clean, simple and elegant. FlatLaf comes with Light, Dark, IntelliJ and Darcula themes, scales on HiDPI displays and runs on Java 8 or newer. The look is heavily inspired by Darcula and IntelliJ themes from IntelliJ IDEA 2019.2+ and uses almost the same colors and icons." A screenshot of the FlatLaf Demo application is shown, displaying various UI components like buttons, checkboxes, and text fields. The demo is set to the "Flat Dark" theme. On the right side of the website, there's a "Page Contents" section with links to Features, Demo, Download, Getting started, Customizing, Themes, Theme Editor, License, Source code, and Issues. Below that is a "NEWS & BLOG" section with entries for FlatLaf 1.0, JFormDesigner 7.0.3, JFormDesigner 7.0.2, and JFormDesigner 7.0.1. At the bottom, there's a "TWEETS" section with a tweet from JFormDesigner.

Clase VentanaMain (VentanaMain.java)

Al comienzo de la aplicación se ha añadido el setLocationRelativeTo(null) para que la ventana salga en el centro de la pantalla.

```

public VentanaMain() {
    initComponents();
    setLocationRelativeTo(null); //Para que salga en el centro de la pantalla
}

```

En el método main antes que se visualice la ventana se hace una llamada al setLookAndFeel pasandole el nuevo estilo que hemos añadido al proyecto.

```

public static void main(String args[]) {
    /* Set the Nimbus look and feel */
    Look and feel setting code (optional)
    //</editor-fold>

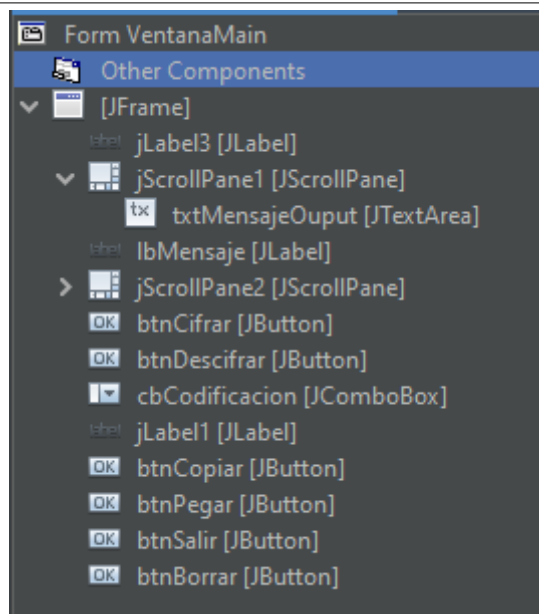
    /* Create and display the form */
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            try {
                UIManager.setLookAndFeel(new FlatDarkLaf());
            } catch (UnsupportedLookAndFeelException ex) {
                Logger.getLogger(VentanaMain.class.getName()).log(Level.SEVERE, null, ex);
            }
            new VentanaMain().setVisible(true);
        }
    });
}

```

En el editor de diseño desmarque la opción de resizable para que el tamaño de la ventana fuera estático.

The screenshot shows the Java Swing component inspector. The 'resizable' property is highlighted with a red line and is set to 'false'. Other properties visible include 'opacity' (1.0), 'opaque' (checked), 'preferredSize' ([517, 485]), 'shape' (<none>), 'size' (<Not Set>), 'state' (0), 'type' (NORMAL), and 'validateRoot' (checked).

Los componentes que forman la aplicación son estos que se pueden ver.
 Los JTextArea son donde se introduce el texto y donde sale el resultado.
 Los botones y el combobox son para interactuar el usuario



El método **clickCifrarMensaje** es llamado al pulsar el usuario el botón de “Cifrar Mensaje” y lo primero que hace es comprobar que hay texto en el JTextArea, en caso que no hubiera nada muestra un aviso con un JOptionPane.
 Si hay datos para cifrar obtiene el valor del ComoBox necesario para saber cual es el salto para el cifrado cesar.
 Después llama a la variable estática que hace el cifrado y el resultado lo mete directamente en el setText del jtextarea.
 Y por ultimo cambia un Label con una frase para que el usuario vea que su mensaje ha sido cifrado.

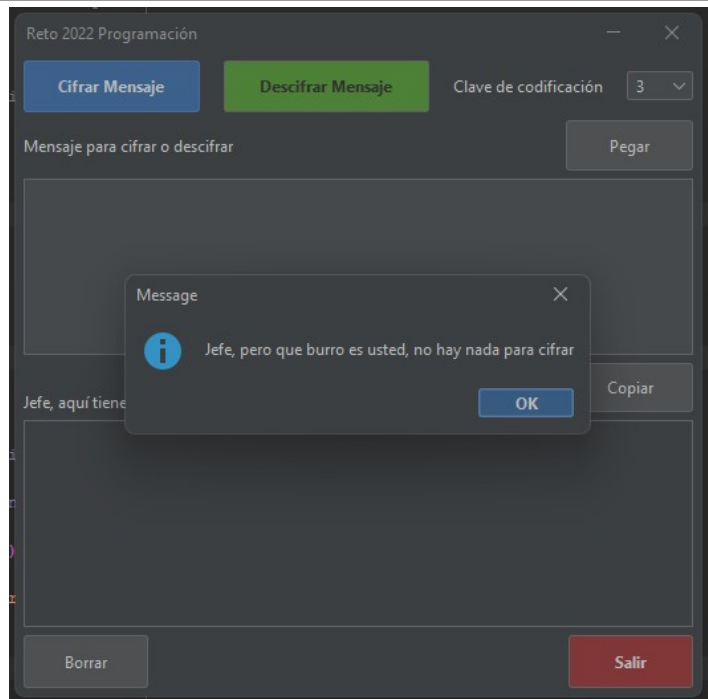
```
/**
 * Método que se lanza con el click en el botón de cifrarmensa
 * @param evt
 */
private void clickCifrarMensaje(java.awt.event.ActionEvent evt) {

    if (txtMensajeInput.getText().length() != 0) { //Se comprueba que hay texto que cifrar en el jTextArea

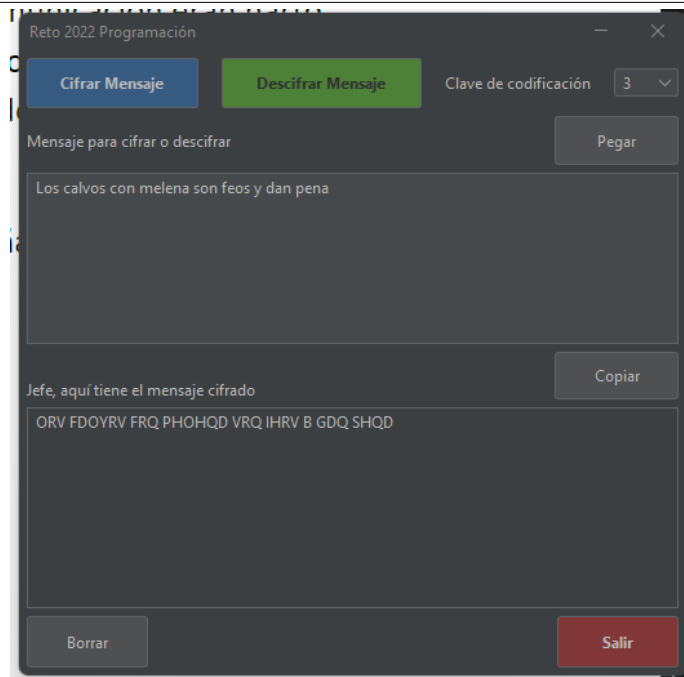
        int codificacion = Integer.valueOf((String) cbCodificacion.getSelectedItem()); //Obtenemos el valor del
        txtMensajeOutput.setText(Cripto.cifradoCesar(txtMensajeInput.getText(), codificacion));

        lbMensaje.setText("Jefe, aqui tiene el mensaje cifrado");
    } else {
        JOptionPane.showMessageDialog(this, "Jefe, pero que burro es usted, no hay nada para cifrar");
    }
}
```

Aviso en caso de no existir datos. →



Muestra de como cifra una de las frases dadas →



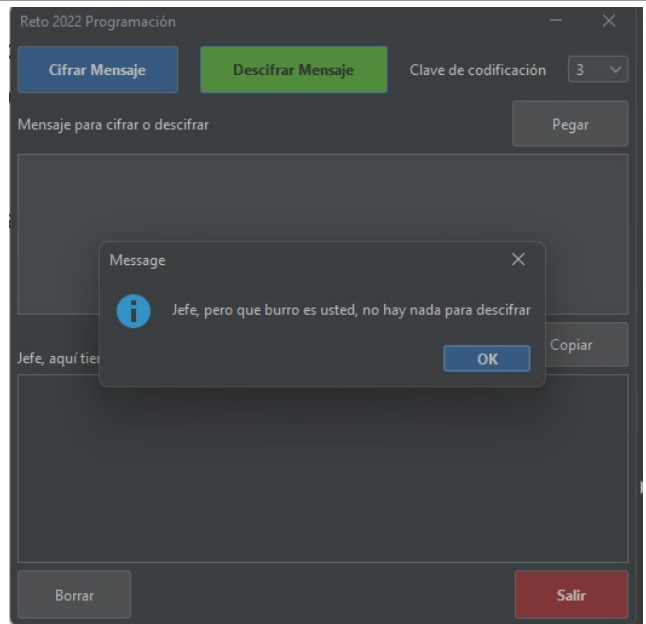
El método `clickDescifrarMensaje` se lanza cuando el usuario pulsa el botón de descifrar mensaje. El funcionamiento es muy parecido al de cifrar salvo por el detalle que el numero obtenido en el ComboBox lo que hago el invertirlo, multiplicándolo por menos uno (La explicación de esto cuando analicemos la clase `Cripto` y su método estático.

```
/**
 * Método que se lanza con el click en el botón de descifrarmensaje
 * @param evt
 */
private void clickDescifrarMensaje(java.awt.event.ActionEvent evt) {
    if (txtMensajeInput.getText().length() != 0) { //Se comprueba que hay texto que cifrar en el jTextArea

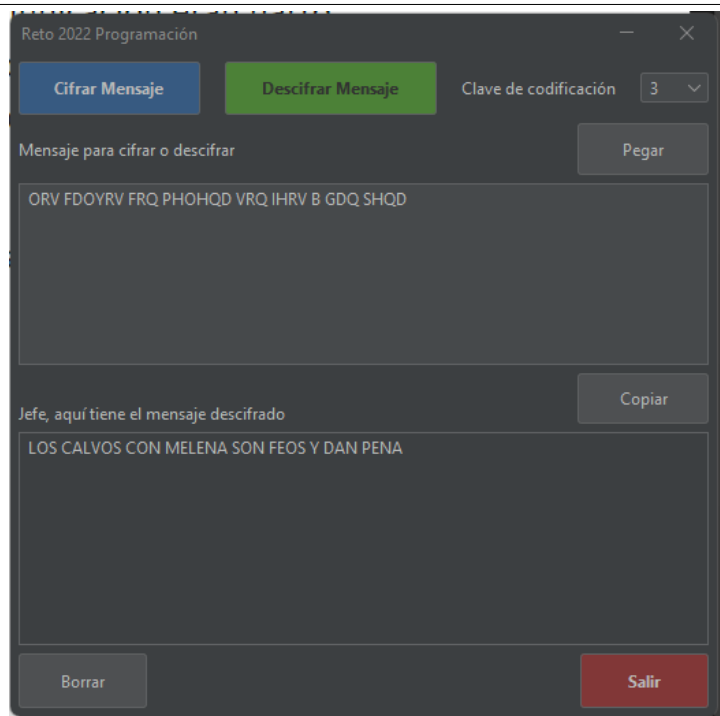
        int codificacion = Integer.valueOf((String) cbCodificacion.getSelectedItem()); //Obtenemos el valor del
        codificacion *= (-1); //Invierte el numero para decodificar
        txtMensajeOutput.setText(Cripto.cifradoCesar(txtMensajeInput.getText(), codificacion));

        lbMensaje.setText("Jefe, aquí tiene el mensaje descifrado");
    } else {
        JOptionPane.showMessageDialog(this, "Jefe, pero que burro es usted, no hay nada para descifrar");
    }
}
```

En caso de no existir datos →



Descifrando el mensaje cifrado anteriormente →



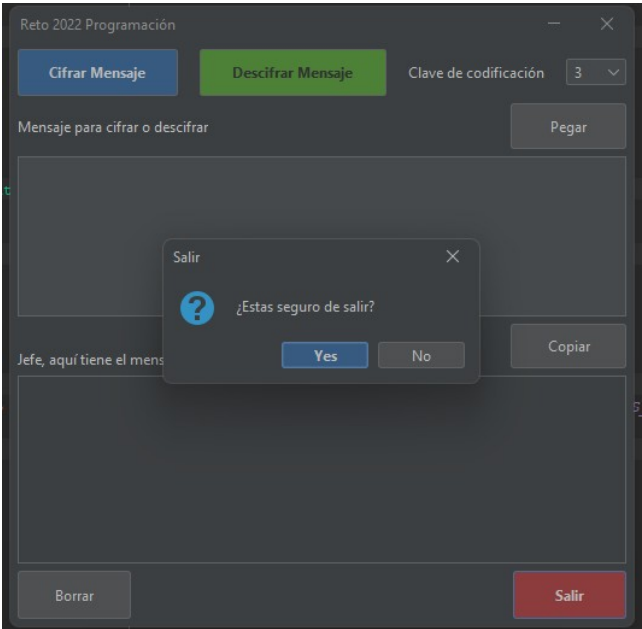
El método `btnBorrarActionPerformed` se lanza al pulsar el botón de borrar y lo que hace es cambiar los dos textarea a un string vacío.

```
/**
 * Método que se lanza con el click en el botón Borrar y lo que hace es
 * vaciar toda la información de los textarea.
 * @param evt
 */
private void btnBorrarActionPerformed(java.awt.event.ActionEvent evt) {
    txtMensajeInput.setText("");
    txtMensajeOutput.setText("");
}
```

El método `btnSalirActionPerformed` se lanza cuando el usuario pulsa el botón de Salir.

```
/**
 * Método que se lanza al pulsar el botón de salir y que lo que hace es finalizar el programa.
 * @param evt
 */
private void btnSalirActionPerformed(java.awt.event.ActionEvent evt) {
    if (JOptionPane.showConfirmDialog(rootPane, "¿Estas seguro de salir?", "Salir", JOptionPane.YES_NO_OPTION) == JOptionPane.YES_OPTION)
        System.exit(0);
}
```

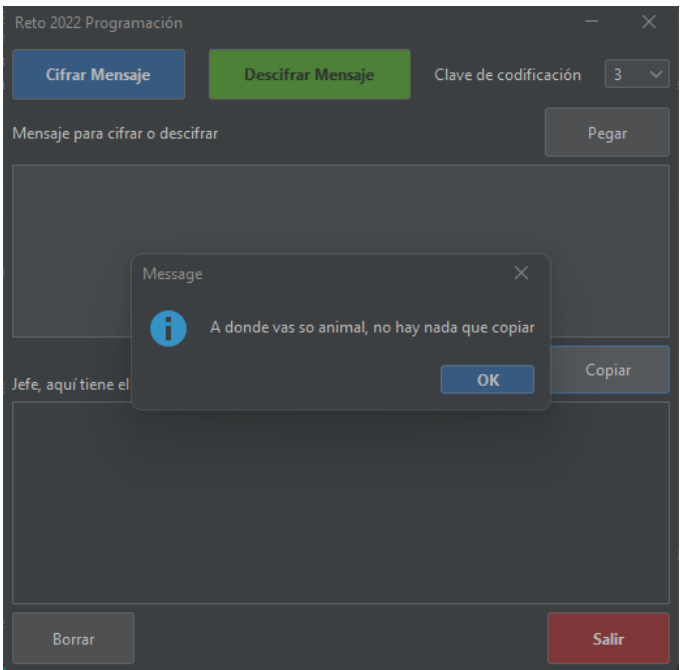
Lo que hace es preguntar al usuario si quiere salir, en caso que sea YES entonces llama al proceso de salir de la aplicación y se termina la ejecución de esta.



El método btnCopiarclickCopiar se lanza al pulsar el usuario en el botón de copiar y lo que hace es comprobar que en el textarea de salida hay datos, en caso de que haya texto lo que hace es copiar ese texto (String) en el portapapeles de windows.

```
/**
 * Método que se lanza en el botón click para copiar en el portapapeles el
 * texto del salida
 * @param evt
 */
private void btnCopiarclickCopiar(java.awt.event.ActionEvent evt) {
    if (txtMensajeInput.getText().length() != 0) { //Se comprueba que hay texto que cifrar en el JTextArea
        StringSelection stringSelection = new StringSelection(txtMensajeOutput.getText()); //Obtenemos el texto
        Clipboard clipboard = Toolkit.getDefaultToolkit().getSystemClipboard(); //Preparamos para copiar el portapapeles
        clipboard.setContents(stringSelection, null); //envia el texto al portapapeles
    } else {
        JOptionPane.showMessageDialog(rootPane, "A donde vas so animal, no hay nada que copiar");
    }
}
```

Sino hubiera datos lanza un mensaje →



El método btnPegarclickPegar se lanza al pulsar el botón de pegar y lo que hace es pegar cualquier texto que tenga en el portapapeles de windows. He añadido esta funcionalidad entendiendolo que es sobre todo muy cómodo a la hora de introducir un texto cifrado.

```
/**
 * Método que se lanza en el botón click para pegar en el jtextarea de entrada
 * el texto si lo hubiera en el portapapeles
 * @param evt
 */
private void btnPegarclickPegar(java.awt.event.ActionEvent evt) {
    Clipboard systemClipboard = Toolkit.getDefaultToolkit().getSystemClipboard();
    DataFlavor dataFlavor = DataFlavor.stringFlavor;

    if (systemClipboard.isDataFlavorAvailable(dataFlavor))
    {
        try {
            Object text = systemClipboard.getData(dataFlavor);
            txtMensajeInput.setText((String) text);

        } catch (UnsupportedFlavorException ex) {
            Logger.getLogger(VentanaMain.class.getName()).log(Level.SEVERE, null, ex);
        } catch (IOException ex) {
            Logger.getLogger(VentanaMain.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```

Clase Cripto (Cristo.java)

Lo primero que he defino en esta clase es una String con todas las letras que deseo utilizar.

Como se puede observar tenemos dos String uno esta comentado, el primero contiene todas las letras en mayúscula sin la letra Ñ, ya que es el método habitual. Y el segundo String que esta comentado contiene todas las letras incluida la Ñ más los números y algunos caracteres especiales como la arroba, las comas o los puntos y se podrían ir añadiendo más, consiguiendo así mayor dificultad de cifrado.

```
private static final String letras = "ABCDEFGHIJKLMNPOQRSTUVWXYZ"; //Solo letras SIN la Ñ
//private static final String letras = "ABCDEFGHIJKLMNPOQRSTUVWXYZ1234567890@.,": //Letras incluida la Ñ y numeros y simbolos que encripta o desencripta
```

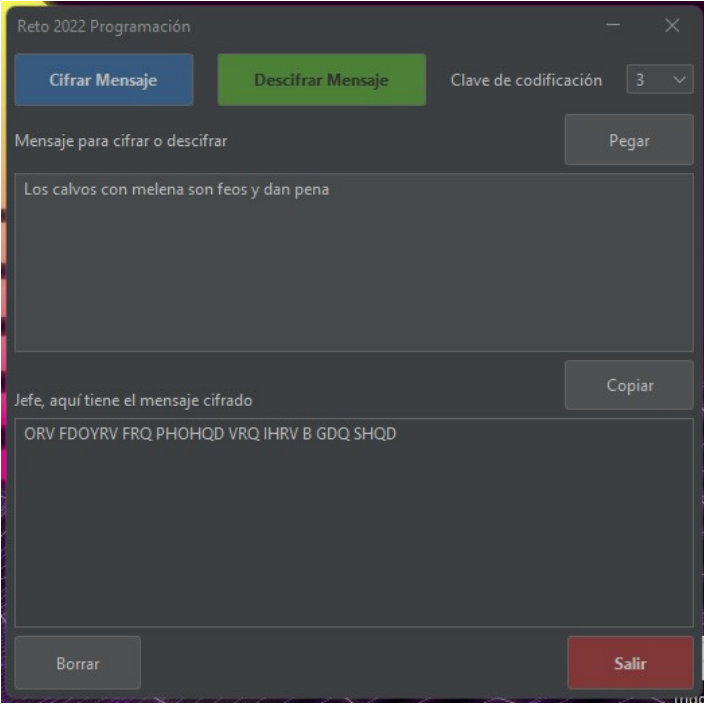
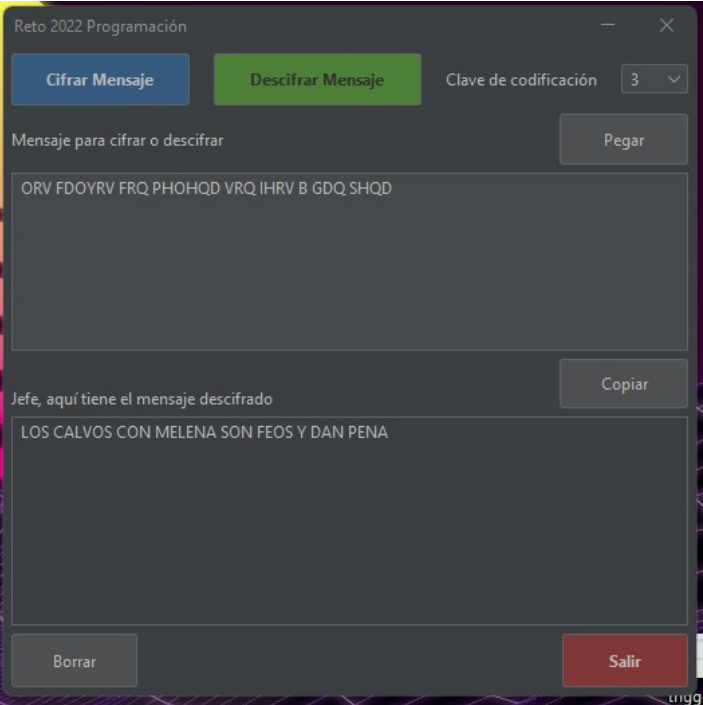
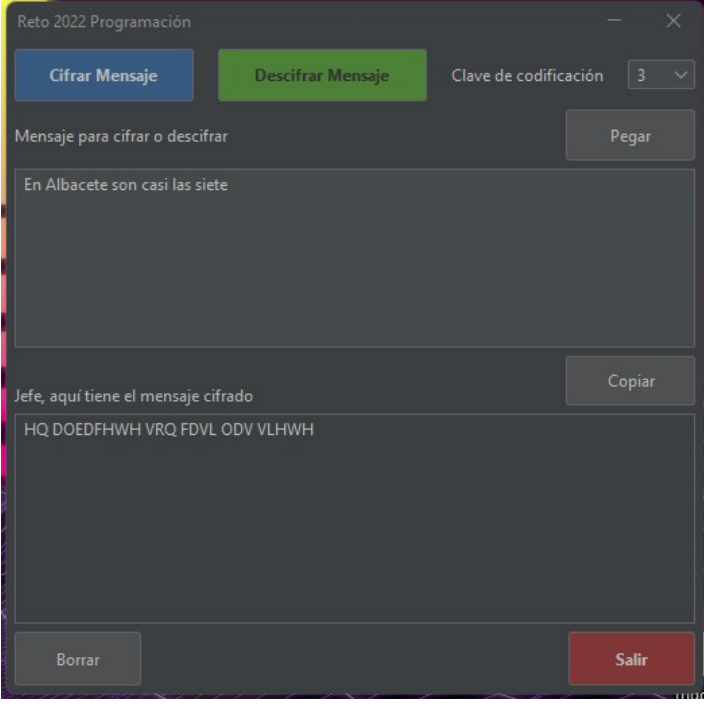
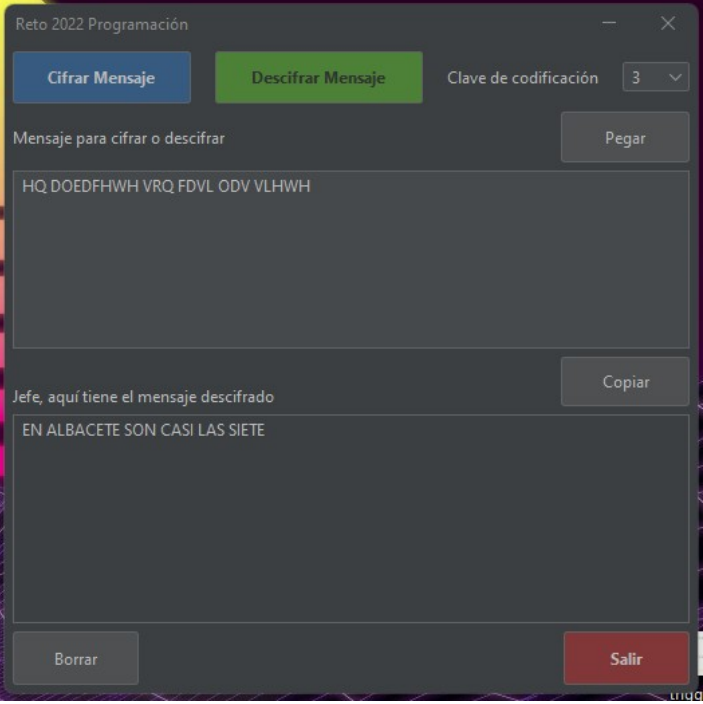
El método estático cifradoCesar es el método encargado de cifrar y descifrar el texto que se pasa por parámetro. El otro parámetro que se pasa es el numero de salto que se tiene que hacer para el descifrado en caso positivo y si es negativo hace el descifrado. El algoritmo esta creado para que sea cíclico, es decir si se pasa de las letras comienza por el principio y lo mismo pasa a la inversa si se pasa hacia atrás continua por el final.

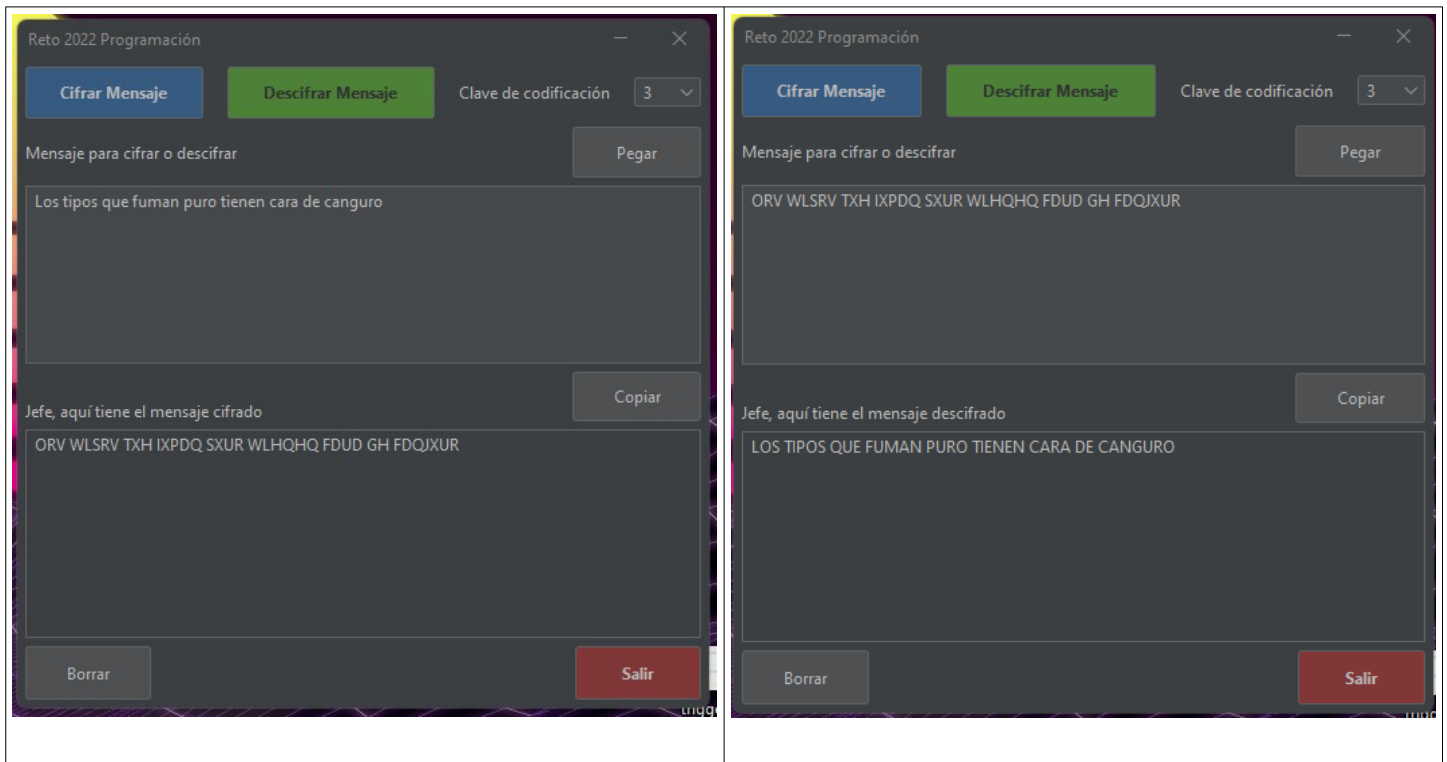
```
/**
 * Devuelve el texto cifrado o descifrado, en funcion del parametro salto
 *
 * @param text Texto que se desea encriptar o desencriptar
 * @param salto el salto determina el cifrado, es decir si para cifrar se uso un numero
 * para descifrar se necesita el mismo numero en negativo.
 * @return texto cifrado o descifrado
 */
public static String cifradoCesar(String text, int salto) {
    char textArray[] = text.toUpperCase().toCharArray();
    int posLetra;
    for (int i=0;i<textArray.length;i++) {
        if (letras.contains(String.valueOf(textArray[i]))) {
            posLetra = letras.indexOf(String.valueOf(textArray[i]).toUpperCase());

            if ((posLetra+salto)>(letras.length()-1)){
                textArray[i]=letras.charAt((posLetra + salto)-(letras.length()));
            } else if ((posLetra+salto)<0) {
                textArray[i]=letras.charAt((letras.length()) + (posLetra + salto));
            } else {
                textArray[i]=letras.charAt(posLetra+salto);
            }
        }
    }
    return String.copyValueOf(textArray);
}
```

Casos que se aportan para probar el programa.

- “Los calvos con melena son feos y dan pena”
- “En Albacete son casi las siete”
- “Los tipos que fuman puro tienen cara de canguro”

Cifrando	Descifrando
	
	



Y por último hago una prueba con el otro String que añado la Ñ, los números algún que otro símbolo he incluso los espacios, consiguiendo en el encriptado aparentemente sea más confuso.

- Mi dirección del correo electrónico de educantabria es jblancog03@educantabria.es

