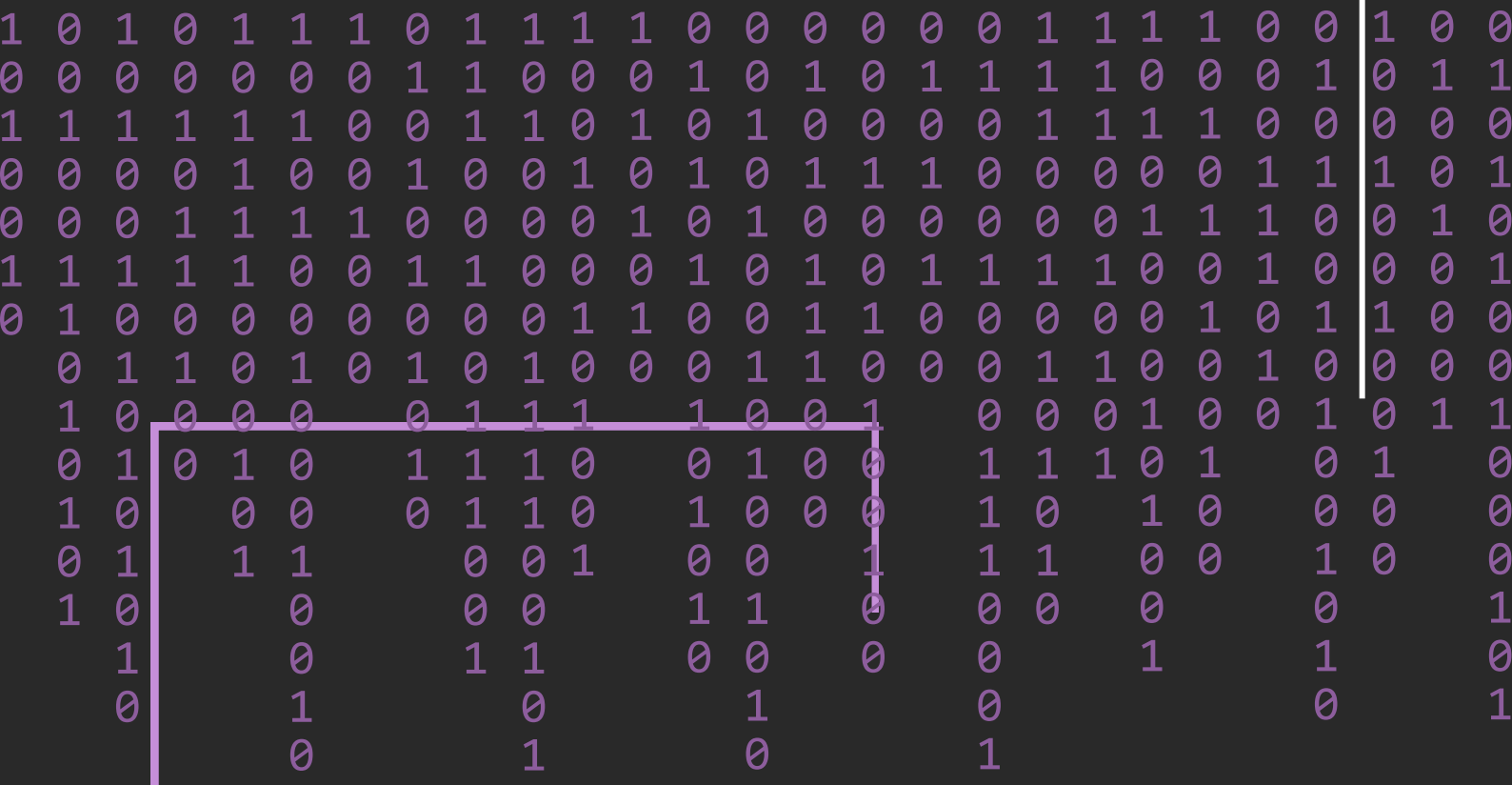


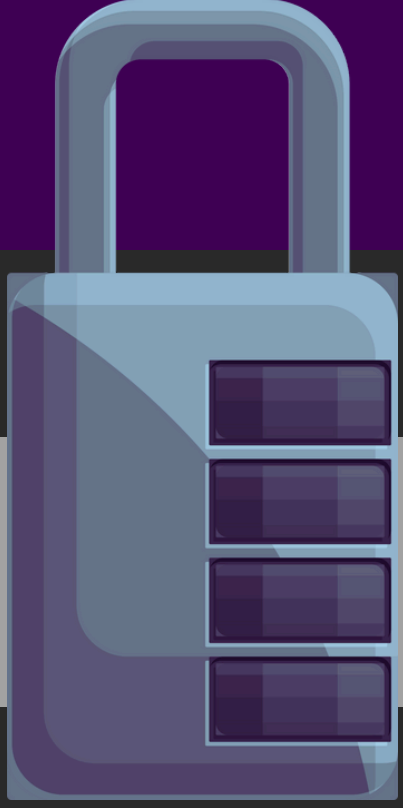
MANUAL DE SISTEMA
ENCRIPT-A

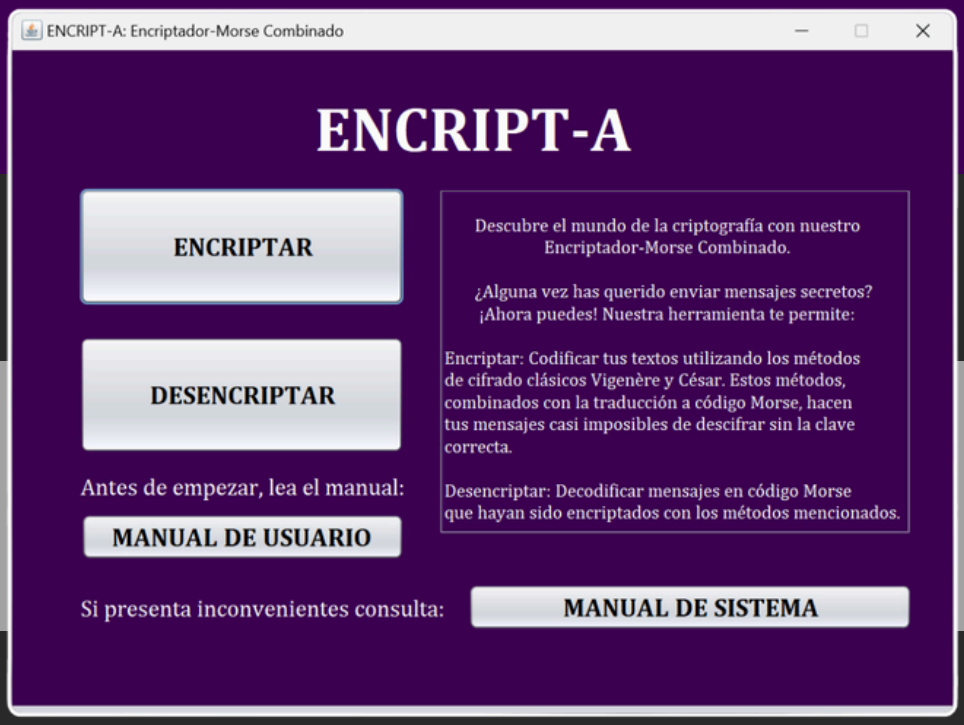
Segundo Lab: Encriptador-Morse

ALGORITMIA Y PROGRAMACIÓN II

2024

Katherin Barrera y Valeria Florez





INTRODUCCIÓN

ENCRIPT-A es un programa diseñado para permitir a los usuarios convertir mensajes de texto en código Morse y viceversa. Este software facilita la encriptación de texto, proporcionando una herramienta sencilla y rápida para codificar y decodificar información, preservando su privacidad o simplemente explorando el fascinante lenguaje del código Morse.

CARACTERÍSTICAS PRINCIPALES:

TEXTO DESCIFRADO ⇄ **MORSE CIFRADO**

CONVERSIÓN DE TEXTO A CÓDIGO MORSE:

Permite a los usuarios escribir mensajes y convertirlos en código Morse cifrado de manera instantánea.

CONVERSIÓN DE CÓDIGO MORSE A TEXTO:

Ofrece la opción de decodificar mensajes escritos en código Morse, transformándolos en texto legible.

¿A QUIÉN ESTÁ DIRIGIDO?

El programa está dirigido a usuarios de todos los niveles de experiencia, desde principiantes hasta personas con conocimientos avanzados en codificación. Es una herramienta ideal tanto para fines educativos, como para usuarios que buscan una forma divertida y práctica de encriptar mensajes.

ENCRIPTAR

```
String[] abecedario={"A","B","C","D","E","F","G","H","I","J","K","L","M","N","O","P","Q",
```

Antes que cualquier otro metodo se trabajo con un Vector abecedario el cual seguiremos usando alrededor del codigo, esto es exactamente igual en la parte de **desencriptar.**

Las imagenes a continuacion pertenecen a las acciones del boton **ENCRIPTAR**

```
String clav = clave.getText().toUpperCase();
if (!validarEntrada( texto: text)) {
    return;
}
String[] Texto_claro = new String[text.length()];
for (int i = 0; i < text.length(); i++) {
    Texto_claro[i] = text.substring( beginIndex: i, i+1);
}
```

Se le asignan los dos textos recolectado y en mayuscula a dos variables.

Se validan las entradas con una funcion que veremos detalladamente mas adelante.

Se separa cada una de las letras de nuestro texto principal(el cual esta en el TextArea) y cada una se le asigna a una posicion de un vector lladamo "Texto_Claro", esto nos ayudara mas adelante con el metodo Cesar

ENCRIPTAR

2

```
switch (metodo.getSelectedIndex()) {
    case 1 ->                   {
            // Método César
            String C = metodoCesar(Texto_claro, 1:text.length());
            Texto.append("\nCriptograma: " + C);
            traducirMorse( Criptograma: C);
        }
```

Si el usuario selecciona el metodo Cesar, este se llama y el TextArea se mostrara el critograma(resultado de esta metodo) y su traduccion al codigo morse (que se detallara mas adelante).

Metodo César

```
String metodoCesar(String[] Texto_claro, int l){
String Criptograma = ""; // Inicializa el Criptograma

    for(int i = 0; i < l; i++){
        if (Texto_claro[i].equals( anObject:" ")) {  // Verifica si es un espacio
            Criptograma += " ";  // Agrega un espacio directamente
            continue;  // Salta al siguiente carácter
        }
```

Inicializamos el criptograma y verificamos la aparicion de espacios para evitar errores y poder trabajar varias palabras

```
int j = 0;
while ((j < 26) && (!abecedario[j].equals(Texto_claro[i]))) {
    j++;
}
```

Se busca la posicion de cada letra del texto con respecto del abecedario.

ENCRIPTAR

3

Metodo César

2

```
    if (j < 26) {  // Si se encontró una letra v
        Criptograma += abecedario[(j + 3) % 26];
    }
}
return Criptograma;
```

Si la letra fue encontrada, agrega al criptograma la letra desplazada tres posiciones hacia adelante en el alfabeto. El (j + 3) % 26 asegura que el índice se mantenga dentro de los límites de abecedario (si j + 3 supera 25, "vuelve" al principio del alfabeto).

```
case 2 ->                    {
        // Método Vigenére
        if (!validarEntrada( texto: clav)) {
            return;
        }           String C = metodoVignere(text, clav);
        Texto.append("\nCriptograma: " + C);
        traducirMorse( Criptograma: C);
    }
```

Si el usuario selecciona el metodo Vigenére este se llama y se valida la clave, luego se le asinga a "C" al llamado de metodo y en el Text Area se escribe el criptograma y su traduccion.

ENCRIPTAR

4

Metodo Vigenére

```
String metodoVignere(String text, String clav){
    String mensaje = text.toUpperCase();
    String cl = clav.toUpperCase();
    int longitud = mensaje.length();

    String sub = ""; // Clave repetida
    String criptograma = ""; // Texto cifrado
```

Se asignan y declaran las variables necesarias.

```
    // Construir la clave repetida
    for (int i = 0; i < longitud; i++) {
        sub += cl.substring(i % cl.length(), (i % cl.length()) + 1);
    }
```

Se repite la clave la longitud del texto.

```
for (int i = 0; i < longitud; i++) {
    String letraMensaje = mensaje.substring(i, i + 1);
```

Se inicia un ciclo en donde asinamos una variable con una subcadena que contiene cada letra del texto.

```
    if (letraMensaje.equals(" ")) {  // Si es un espacio, simplemente añadirlo
        criptograma += " ";
        continue;
    }
```

Si el texto tiene un espacio entonces simplemente lo agrega tambien como un espacio.

ENCRIPTAR

5

Metodo Vigenére

2

```
// Procesar cada letra del mensaje
for (int i = 0; i < longitud; i++) {
    String letraMensaje = mensaje.substring( beginIndex: i, i + 1);

    if (letraMensaje.equals( anObject: " ")) {   // Si es un espacio, simple
        criptograma += " ";
        continue;
    }
```

Si existe un espacio, simplemente se coloca

Se inicia un ciclo en donde asignamos una variable con una subcadena que contiene cada letra del texto.

```
String letraClave = sub.substring( beginIndex: i, i + 1);

int posMensaje = posicion( ab: abecedario,  c: letraMensaje);
int posClave = posicion( ab: abecedario,  c: letraClave);
```

Asignamos una variable con una subcadena que contiene cada letra de la clave.

Se le asigna a las dos variables el llamado de la funcion "posicion", sin embargo, una con mensaje y otra con la clave

ENCRIPTAR

6

Metodo Vigenére

3

```
    if (posMensaje != -1 && posClave != -1) {
        criptograma += abecedario[(posMensaje + posClave) % 26];
    }
}

return criptograma;
```

Si ambas letras son válidas (las posiciones no son -1), calcula la posición cifrada sumando posMensaje y posClave, aplicando % 26 par.

```
public static int posicion(String[]ab, String c){
    for (int i = 0; i < ab.length+1; i++) {
        if (ab[i].equals( anObject: c)){
            return i;
        }
    }
     return -1;
}
```

Recorre cada elemento de ab. Si encuentra una coincidencia entre ab[i] y c, devuelve i, la posición de c. Sino retorna -1.

DESENCRIPTAR

```
String b = Texto.getText();
String C = traducirMorse( Morse: b);
 if (!validarEntrada( texto: C)) {
    return;
}
```

```
 Texto.append("\nCriptograma: " + C);
String[] Cripto = new String[C.length()];
for (int i = 0; i < C.length(); i++) {
    Cripto[i] = C.substring( beginIndex: i, i+1);
}
```

Se le asigna a el texto a una variable y luego se traduce a las letras del abecedario.

Se escribe la traduccion en el TextArea y luego se separan cada una de las letras y se colocan en cada una de las posiciones de un vector llamado "Cripto".

```
switch (metodo.getSelectedIndex()) {
    case 1 -> {
        // Método César
        String MC = metodoCesarInverso(Cripto,  1: C.length());
        Texto.append("\nMensaje descifrado: " + MC.toLowerCase());
    }
```

Si el metodo Cesar es seleccionado, se le llama al procedimiento y se le asigan a una variable, para luego escribirlo en el text area.

DESENCRIPTAR 2

Metodo César

```
String metodoCesarInverso(String[] Cripto, int l) {
    String Texto_claro = "";

    for (int i = 0; i < l; i++) {
        if (Cripto[i].equals( anObject: " ")) {
            Texto_claro += " "; // Agrega espacio directamente
        } else {
            int j = posicion( ab: abecedario, Cripto[i]);
            if (j != -1) {
                Texto_claro += abecedario[(j - 3 + 26) % 26];
            }
        }
    }

    return Texto_claro;
}
```

El metodo Cesar de la parte de desencriptar es exactamente igual al metodo Cesar de la parte de incriptar, si deseeas una informacion mas detallada puedes devolverte a donde se explica paso a paso

DESENCRIPTAR

3

Metodo Vigenére

```
String metodoVignere(String text, String clav){
    String mensaje = text.toUpperCase();
    String cl = clav.toUpperCase();
    int longitud = mensaje.length();

    String sub = ""; // Clave repetida
    String criptograma = ""; // Texto cifrado

    // Construir la clave repetida
    for (int i = 0; i < longitud; i++) {
        sub += cl.substring(i % cl.length(), (i % cl.length()) + 1);
    }

    // Procesar cada letra del mensaje
    for (int i = 0; i < longitud; i++) {
        String letraMensaje = mensaje.substring( beginIndex: i, i + 1);

        if (letraMensaje.equals( anObject: " ")) {  // Si es un espacio, simple
            criptograma += " ";
            continue;
        }

        String letraClave = sub.substring( beginIndex: i, i + 1);

        int posMensaje = posicion( ab: abecedario,  c: letraMensaje);
        int posClave = posicion( ab: abecedario,  c: letraClave);

        if (posMensaje != -1 && posClave != -1) {
            criptograma += abecedario[(posMensaje + posClave) % 26];
        }
```

El metodo Vigenere de la parte de desencriptar es exactamente igual al metodo Vigenere de la parte de incriptar, si deseeas una informacion mas detallada puedes devolverte a donde se explica paso a paso

CODIGO MORSE

Encriptar

```
String Morse = "";
String[] codigoMorse = {".-", "-...", "-.-.", "-..", ".", "..-.", "
                        ".-..", "--", "-.", "---", ".--.", "--.-",
                        ".--", "-..-", "-.--", "--.."};
```

Creamos una cadena vacia donde se irá construyendo el mensaje en código Morse. Y un arreglo que contiene el código Morse correspondiente a cada letra del alfabeto, en orden (A-Z).

```
for (int i = 0; i < Criptograma.length(); i++) {
    String letra = Criptograma.substring( beginIndex: i, i + 1);

    if (letra.equals( anObject: " ")) {  // Si es un espacio, agrega
        Morse += "  ";
    } else {
        int j = posicion( ab: abecedario,  c: letra);
        if (j != -1) {
            Morse += " " + codigoMorse[j];
        }
    }
}
Texto.append("\nCódigo Morse: " + Morse.toLowerCase());
```

Creamos una subcadena que extrae cada letra para comparar

Si es un espacio, agregar dos espacios para separarlo en Morse

Se usa una función llamada posicion para encontrar la posición de letra en el arreglo abecedario.Si letra es una letra válida (es decir, si posicion no devuelve -1), se añade el código Morse correspondiente desde codigoMorse[j] a Morse, precedido de un espacio. Por ultimo se escribe en en text area.

CÓDIGO MORSE 2

Desencriptar

```
String traducirMorse(String Morse){
    String Criptograma="";
    String[] codigoMorse = {".-", "-...", "-.-.", "-..", ".", "
                            ".-..", "--", "-.", "---", ".--.",
                            ".--", "-..-", "-.--", "--.."};
    int em = 0;
```

Creamos una variable almacenará el texto traducido, un arreglo contiene los códigos Morse correspondientes a cada letra del alfabeto inglés, en orden (A-Z) y una variable que almacenara el índice que marca el inicio de cada código Morse en el mensaje.

```
while (em < Morse.length()) {
    int ter = em;
    while (ter < Morse.length() && !Morse.substring( beginIndex: ter, ter + 1).equals( anObject: " ")) {
        ter++;
    }

  String letra = Morse.substring( beginIndex: em,  endIndex: ter);
```

Aqui tenemos bucle recorre todo el mensaje Morse. Tambien una variable ter que es un índice que marca el final del código Morse actual. Asi mismo con el segundo while: Avanza ter hasta el próximo espacio en Morse, que indica el final de un código Morse para una letra. Esto permite aislar cada letra en código Morse.

CÓDIGO MORSE

3

Desencriptar

2

```
String letra = Morse.substring( beginIndex: em,  endIndex: ter);


 boolean encontrado = false;
 for (int i = 0; i < codigoMorse.length; i++) {
     if (letra.equals(codigoMorse[i])) {
         Criptograma += abecedario[i];
         encontrado = true;
     }
 }


 em = ter + 1;
```

Primero creamos la variable " letra" que extraerael símbolo Morse (para una sola letra) desde em hasta ter. Luego busca la letra correspondiente en codigoMorse,

usando un ciclo para comparar letra con cada elemento de codigoMorse.Cuando encuentra una coincidencia, añade la letra correspondiente en abecedario[i] a Criptograma. Luego mueve "em" al siguiente carácter después del símbolo Morse procesado.

CÓDIGO MORSE

4

Desencriptar

3

```
    if (em < Morse.length() - 1 && Morse.substring( beginIndex: em, em + 1).e
        Criptograma += " "; // Añadir espacio entre palabras
        em += 2; // Saltar los dos espacios
    }
}

return Criptograma; // Devolver el texto traducido
```

Si hay dos espacios consecutivos, esto indica un espacio entre palabras. En ese caso añade un espacio en Criptograma para separar palabras y avanza em dos posiciones para saltar ambos espacios. Luego retorna el criptograma traducido.

VALIDACIONES

Encriptar

```
private boolean validarEntrada(String texto) {
    if (texto.equals( anObject: "")) {
        mensaje.setText( text: "El texto no puede estar vacío.");
        return false;
    }

    String textol = texto.toUpperCase();
    if (texto.equals( anObject: "")) {
        mensaje.setText( text: "El texto no puede estar vacío.");
        return false;
    }
    for (int i = 0; i < textol.length(); i++) {
        String letra = textol.substring( beginIndex: i, i+1);

        if (!letra.equals( anObject: " ") && posicion( ab: abecedario,  c: letra) == -1) {
            mensaje.setText( text: "Esta palabra no es válida");
            return false;
        }
    }

    mensaje.setText( text: "");
    return true;
}
```

Mediante condicionales validamos que ninguna entrada puede estar vacia y que en las cadenas no puede colocar algo diferente de las letras del abecedario.

VALIDACIONES 2

```
private boolean validarEntrada(String texto) {
  if (texto.equals( anObject: "")) {
        mensaje.setText( text: "El texto no puede estar vacío.");
        return false;
    }

    String textol = texto.toUpperCase();
    for (int i = 0; i < textol.length(); i++) {
        String caracter = textol.substring( beginIndex: i, i + 1);
        if (!caracter.equals( anObject: " ")) { // Ignora los espacios
            boolean letraValida = false;
            for (String letra : abecedario) {
                if (caracter.equals( anObject: letra)) {
                    letraValida = true;

                }
            }
            if (!letraValida) {
                mensaje.setText( text: "Esta palabra no es válida");
                return false;
            }
        }
    }

    mensaje.setText( text: "");
    return true;
}
```

E Verifica que el texto ingresado no esté vacío y contenga únicamente letras válidas (basadas en un arreglo abecedario que representa las letras permitidas). Si el texto está vacío, muestra un mensaje indicando que no puede estar vacío. Luego, convierte el texto a mayúsculas y recorre cada carácter, ignorando los espacios. Para cada letra, verifica si está en abecedario; si alguna letra no es válida, muestra un mensaje indicando que la palabra no es válida y devuelve false. Si todas las letras son válidas, limpia el mensaje de error y devuelve true.

CONTACTO

Si tienes preguntas sobre el funcionamiento del software o requieres asistencia técnica, puedes ponerte en contacto con las creadoras del sistema o el equipo de soporte de Almacenes de La Prosperidad a través de los siguientes medios:

Creadoras del Software

Este software fue desarrollado por:

Katherin Barrera y Valeria Florez.

Correo Electrónico de las Creadoras:

Para consultas técnicas o información sobre el desarrollo, puedes escribir a:

- lkatherin@uninorte.edu.co
- florezvaleria@uninorte.edu.co