UNIVERSIDAD TÉCNICA DEL NORTE

Resolución No. 001-073 CEAACES-2013-13

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

CARRERA DE SOFTWARE

ESTRUCTURA DE DATOS

Proyecto Estructura de Datos Pila

Docente: Ing.MSc. Pineda Manosalvas Carpio Agapito

Nombres:
Diego Cuaycal,
Lizbeth Quilumba y
Diego Recalde

Fecha: 22/05/2023

Periodo:
Abril - Agosto

Ibarra – Ecuador

Índice

INTRODUCCIÓN

Objetivos

Objetivo general

Comprender en profundidad el concepto, funcionamiento y aplicaciones de la estructura de datos Pila en la programación, desarrollando habilidades sólidas para su implementación eficiente y uso adecuado.

Objetivo específico

- Aplicar técnicas de implementación de la Pila utilizando diferentes enfoques: Aquí se trata de explorar y aprender cómo implementar una Pila utilizando distintas estructuras de datos subyacentes, listas enlazadas u otras estructuras.

- Desarrollar habilidades para identificar y resolver problemas relacionados con la implementación y el uso de la Pila: Aquí se trata de adquirir habilidades de depuración y resolución de problemas, para poder identificar y solucionar desafíos relacionados con la implementación y el uso de la Pila en programas y algoritmos.

- Estudiar y comprender las aplicaciones prácticas de la estructura de datos Pila: Este objetivo busca explorar y comprender cómo se utiliza la Pila en la solución de problemas comunes de programación, como el procesamiento de expresiones matemáticas, la reversión de cadenas o la administración de memoria, comprendiendo su utilidad y aplicabilidad en diferentes escenarios.

Materiales

Nombre	**Versión**
Apache NetBeans	14,15 y 16
Java	19 y 20
GitHub	-
GitHub Desktop	-

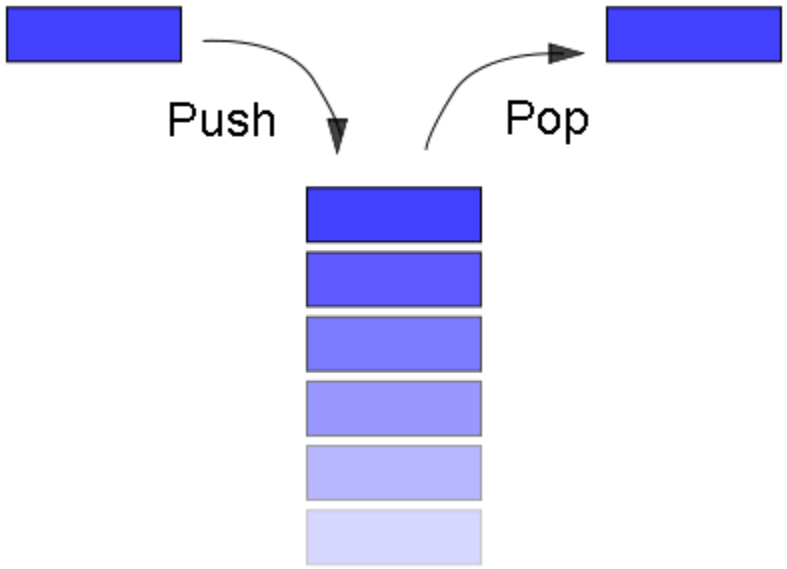
Concepto

¿Qué es Pila?

Una pila, es una estructura ordenada de datos en la que el modo de acceso a sus elementos es de tipo LIFO (Last In First Out, ultimo en entrar, primero en salir) que permite almacenar y recuperar datos. Esta es una estructura se aplica en multitud de ocasiones en el área de la informática debido a su simplicidad y ordenación implícita de la propia estructura.

Para el manejo cuenta con dos operaciones básicas: **Apilar(push),** que coloca un objeto en la pila y su operación inversa, **Desapilar(pop),** que retira el último elemento apilado.

En cada momento solamente se tiene acceso a la parte superior de la pila, es decir, al último objeto apilado. La operación **Desapilar** permite la obtención de este elemento, que es retirado de la pila permitiendo el acceso al interior (Elemento que haya sido apilado antes), que pasa a ser el último (Google.sities, s.f.).



En la pila se suele emplearse algunos contextos:

- Evaluación de expresiones en notación postfija
- Reconocedores sintácticos de lenguaje independientes de contexto.
- Implementación de recursividad.

Métodos principales que deben ser implementados cuando implementamos una pila:
Push: agregar un elemento a la pila.

Pop: eliminar el elemento superior.

Peek (también conocido como top): obtener el elemento superior sin eliminarlo.

Empty (también conocido como isEmpty): verificar si la pila está vacía o no.

Size: obtener el tamaño de la pila.

Aplicación de pila en el juego el Ahorcado

El juego del ahorcado es un juego de palabras en el que un jugador intenta adivinar una palabra o frase oculta. El objetivo del jugador es descubrir la palabra o frase antes de que se complete la representación de un ahorcado. El juego se juega con al menos dos personas, una que selecciona la palabra o frase y otra que intenta adivinarla.

El jugador que adivina propone letras una por una. Si la letra propuesta está en la palabra o frase, se revelan todas las ocurrencias de esa letra. Si la letra no está presente, se añade una parte del ahorcado. El jugador tiene un número limitado de intentos antes de que se complete el dibujo del ahorcado.

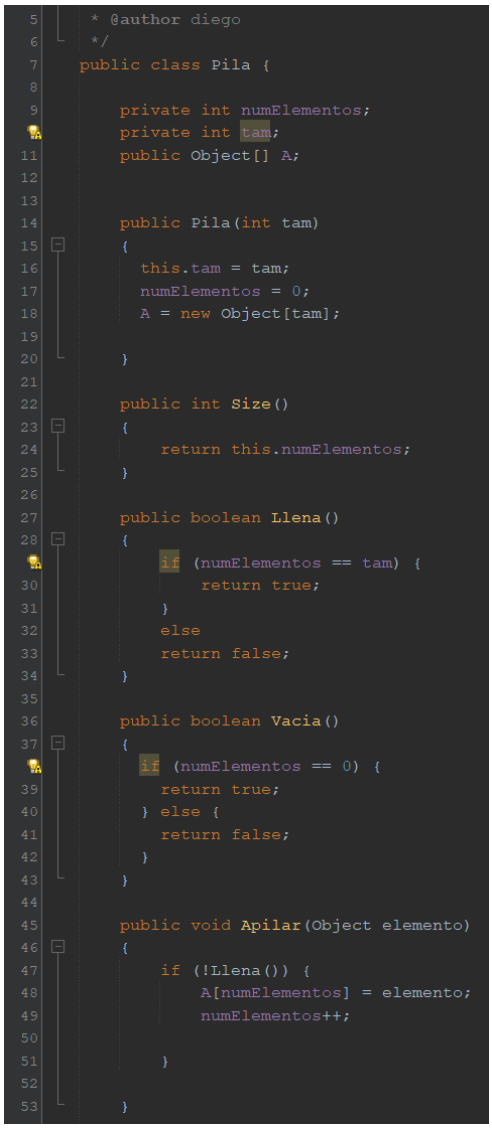
El juego continúa hasta que el jugador adivina la palabra o frase o hasta que se completa el dibujo del ahorcado. Es un juego divertido que pone a prueba las habilidades de adivinanza, el conocimiento de palabras y la estrategia del jugador.

El ahorcado es un juego clásico que se puede jugar en papel y lápiz, o en versiones digitales. Se puede adaptar para diferentes niveles de dificultad y temáticas. Es un juego popular para jugar en reuniones, fiestas o simplemente para pasar el tiempo de manera entretenida.

DESARROLLO:

Clase Pila

Utilizamos la clase pila implementada anteriormente.



```
 * @author diego
 */
public class Pila {

    private int numElementos;
    private int tam;
    public Object[] A;


    public Pila(int tam)
    {
      this.tam = tam;
      numElementos = 0;
      A = new Object[tam];

    }

    public int Size()
    {
        return this.numElementos;
    }

    public boolean Llena()
    {
        if (numElementos == tam) {
            return true;
        }
        else
        return false;
    }

    public boolean Vacia()
    {
      if (numElementos == 0) {
        return true;
      } else {
        return false;
      }
    }

    public void Apilar(Object elemento)
    {
        if (!Llena()) {
            A[numElementos] = elemento;
            numElementos++;

        }

    }
```

```
47
48      public Object Desapilar() {
49          if (!Vacia()) {
50              Object aux = A[numElementos - 1];
51              A[numElementos - 1] = null;
52              numElementos--;
53              return aux;
54          }
55
56          return false;
57      }
58
59      public Object Peek() {
60          if (!Vacia()) {
61              Object aux = A[numElementos - 1];
62              return aux;
63          }
64
65          return false;
66      }
67
68      @Override
69      public String toString() {
70          String salida = "";
71          for (int i = A.length - 1; i >= 0; i--) {
72              if (A[i] != null) {
73                  salida += A[i] + "\n";
74              }
75
76          }
77
78          return salida;
79      }
80
81      public String Imprimir() {
82          return "Cambios";
83      }
84  }
85
```

Clase Ahorcado

Esta clase nos permite jugar al juego del ahorcado, eligiendo una palabra aleatoria y administrando las letras adivinadas por el jugador, utilizando una pila para rastrear el historial y proporcionando métodos para mostrar el estado del juego.

Las variables y el constructor son utilizados para almacenar y gestionar las palabras, letras, historial de movimientos y estado del juego en la implementación del juego del ahorcado.

```
public class Ahorcado {

    private ArrayList<String> palabras;
    public String[] letras;  //arreglo de String que almacena el estado del juego, dicho arreglo se llenara de letra en letra hasta adivinar toda la palabra
    String[] palabraAux;  //arreglo de String para almacenar palabra a adivinar
    public Pila pila;  //uso de la clase Pila creada en clase
    public int intentoActual = 0;

    public Ahorcado()
    {
        palabras = new ArrayList<>();

        //this.intentoActual = intentoActual;

        //try y catch para leer archivo de texto que contiene 50 palabras, dichas palabras seran almacenadas en una arraylist
        try {
            Scanner sc = new Scanner(new File( pathname: "palabras.txt"));
            while (sc.hasNextLine()) {
                palabras.add( e: sc.nextLine());
            }

        } catch (Exception e)
        {
            System.out.println( x: e.getMessage());
            System.out.println( x: "Palabras...2");

        }
    }
```

En este método se utiliza para obtener una palabra aleatoria de la lista de palabras disponibles en el juego.

```
53
54      public String PalabraRandom() {
55          //Palabras del juego
56          //se obtiene una palabra aleatoria del arraylist
57          return palabras.get( index: new Random().nextInt( bound: palabras.size())).toUpperCase();
58
59      }
```

Es un método para iniciar el juego en el se obtiene una letra aleatoria de la palabra a adivinar y se la asigna en la primera jugada del juego.

```
74          */
75      public void Inicio()
76      {
77          palabraAux = PalabraRandom().split( regex: "");      //s
78          pila = new Pila( tam: palabraAux.length);
79          letras = new String[palabraAux.length];
80
81
82          Random ran = new Random();
83          int indiceRandom = ran.nextInt( bound: letras.length);
84          letras[indiceRandom] = palabraAux[indiceRandom];
85          pila.Apilar( elemento: letras.clone());
86           System.out.println("Pila inicial:\n"+PrintPila());
87
88      }
```

Es un método para retornar cadena de la jugada actual. Se genera una cadena que representa de forma visual la jugada. Se Asigna las letras en su respectiva posición y si la posición es null se asigna un guion bajo.

```
96      public String Palabra()
97      {
98
99          String[] operacion = (String[]) pila.Peek();
100         String r = "";
101
102         for (int i = 0; i < letras.length; i++) {
103
104             if (operacion[i] == null) {
105                 r+="_"+" "; // se asigna un guion bajo si la posicion esta vacia
106             }
107             else r += operacion[i]+" ";
108         }
109         return r;
110     }
```

Este método convierte un arreglo de cadenas de texto en una representación visual, utilizando guiones bajos para indicar posiciones vacías y letras para indicar posiciones ocupadas en el arreglo.

```
120         */
121     public String Palabra(String[] palabra)
122     {
123
124         String r = "";
125
127         for (int i = 0; i < palabra.length; i++) {
128
129             if (palabra[i] == null) {
130                 r+="_"+" "; // se asigna un guion bajo si la posicion esta vacia
131             }
132             else r += palabra[i]+" ";
133         }
134         return r;
135     }
```

Método de ayuda para imprimir un arreglo de String.

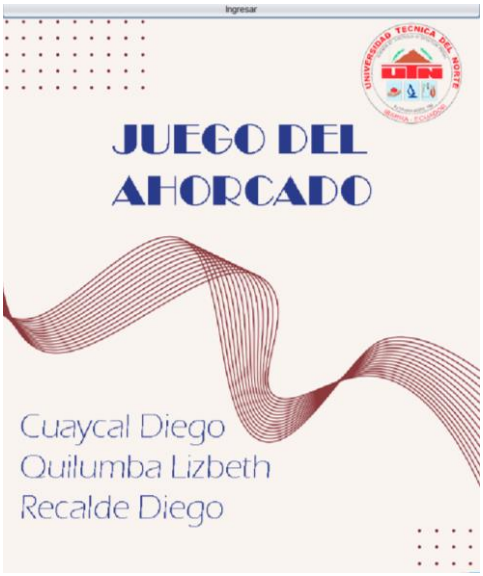
```
135
136     public String StringArray(String[] arreglo)
137     {
138         String resultado = "";
139         for (String r : arreglo) {
140             resultado += r+" ";
141         }
142         return resultado;
143     }
```

Método de ayuda para ver el estado de la pila. En la cual recorre los elementos de la pila y genera una representación visual del estado actual de la pila, donde cada elemento se muestra en una línea separada. Si un elemento es un arreglo de cadenas de texto, se muestra la representación visual del arreglo en lugar del objeto en sí.

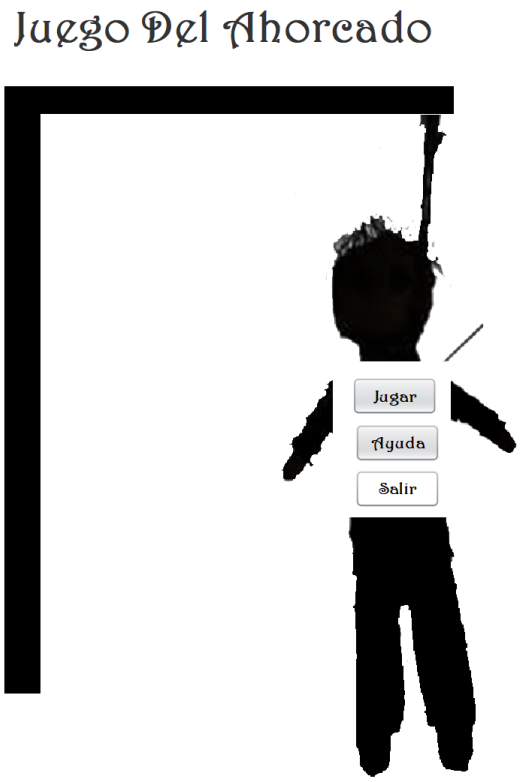
```
146     public String PrintPila()
147     {
148         String r = "";
149         for (int i = 0; i < pila.Size(); i++) {
150
152             if (pila.A[i] instanceof String[]) {
153
154                 String[] aux = (String[])pila.A[i];
155                 r+= this.StringArray( arreglo: aux) + "\n";
156             }
157             else
158                 r+=pila.A[i]+ "\n";
159         }
160         return r;
    }
```

Formularios

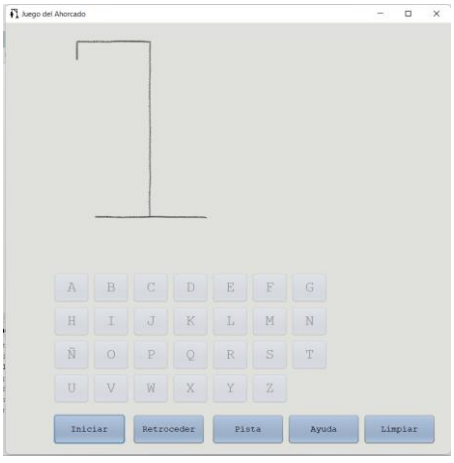
Formulario Portada



Formulario para elegir opciones



Formulario del juego



Clase Form

Variables: Estas variables se van a utilizar en toda la clase Form.

```
Ahorcado juego;
BufferedImage[] sprites;
ImageIcon icono;
int numIntentosMAX = 7;
boolean pistaSelect = false;
String[] pista;
```

Constructor

```
public Form() {
    initComponents();
    // Centrar el formulario en la pantalla
    setLocationRelativeTo( c: null);
    this.getContentPane().setBackground( c: Color.decode( nm: "#e0e1dd"));
    /*
    Al iniciar el juego el usuario no puede jugar mientras no haya dado clic en iniciar es por es
    desabilitacion de los botones
     */
    DeshabilitarBotones();

    lblAyuda.setVisible( aFlag: false);
    lblPista.setVisible( aFlag: false);

    try {

        /*Se utiliza el paquete ImageIO de Java para leer las imagenes almacenadas en la carpeta
        en un arreglo de sprites tipo BufferedImage
         */
        sprites = new BufferedImage[7];
        sprites[0] = ImageIO.read(new File( pathname: "sprites/1.png"));
        sprites[1] = ImageIO.read(new File( pathname: "sprites/2.png"));
        sprites[2] = ImageIO.read(new File( pathname: "sprites/3.png"));
        sprites[3] = ImageIO.read(new File( pathname: "sprites/4.png"));
        sprites[4] = ImageIO.read(new File( pathname: "sprites/5.png"));
        sprites[5] = ImageIO.read(new File( pathname: "sprites/6.png"));
        sprites[6] = ImageIO.read(new File( pathname: "sprites/7.png"));

        icono = new ImageIcon(sprites[0]); //Se utiliza el paquete ImageIcon para almacenar image
        ahorcado.setIcon( icon: icono);       //se utiliza el icono creado para aplicarlo en el labe
```

```
    DeshabilitarBotones();

    lblAyuda.setVisible( aFlag: false);
    lblPista.setVisible( aFlag: false);

    try {

        /*Se utiliza el paquete ImageIO de Java para leer las imagenes almac
        en un arreglo de sprites tipo BufferedImage
         */
        sprites = new BufferedImage[7];
        sprites[0] = ImageIO.read(new File( pathname: "sprites/1.png"));
        sprites[1] = ImageIO.read(new File( pathname: "sprites/2.png"));
        sprites[2] = ImageIO.read(new File( pathname: "sprites/3.png"));
        sprites[3] = ImageIO.read(new File( pathname: "sprites/4.png"));
        sprites[4] = ImageIO.read(new File( pathname: "sprites/5.png"));
        sprites[5] = ImageIO.read(new File( pathname: "sprites/6.png"));
        sprites[6] = ImageIO.read(new File( pathname: "sprites/7.png"));

        icono = new ImageIcon(sprites[0]); //Se utiliza el paquete ImageIcon
        ahorcado.setIcon( icon: icono);       //se utiliza el icono creado para

        ImageIcon formLogo = new ImageIcon( filename: "icono.png");     //se lee
        setIconImage( image: formLogo.getImage());                      //se esta

    } catch (Exception e) {
        System.out.println( x: e.getMessage());
```

Métodos privates

Método private void btnB

Se refiere a un evento que se ejecuta cuando se hace clic en el botón.

```
private void btnBActionPerformed(java.awt.event.ActionEvent evt) {
    Intento( letra: "B",  btn: btnB);
    txtArea.setText( text: juego.Palabra());


}

private void btnIActionPerformed(java.awt.event.ActionEvent evt) {
    Intento( letra: "I",  btn: btnI);
    txtArea.setText( text: juego.Palabra());


}
```

Método private void btnAyuda

Muestra la funcionalidad de mostrar la ayuda al jugador en el juego del ahorcado, si el juego ha sido iniciado previamente.

```
private void btnAyudaActionPerformed(java.awt.event.ActionEvent evt) {

    if (juego != null) {
        lblAyuda.setVisible( aFlag: true);
        lblAyuda.setText( text: Arrays.toString( a: juego.palabraAux));
    } else {
        JOptionPane.showMessageDialog( parentComponent: null,  message: "Ni siquiera ha empezado el juego...",  title: "Error",  messageType: JOptionPane.
    }

}
```

Método private void btnInicio

Se encarga de iniciar y preparar el juego del ahorcado, habilitando los botones de letras, ocultando las ayudas y pista, creando una nueva instancia del juego y actualizando la interfaz gráfica del juego.

```
private void btnIniciarActionPerformed(java.awt.event.ActionEvent evt) {
    JButton[] botonesLetras = {btnA, btnB, btnC, btnD, btnE, btnF, btnG, btnH, btnI, btnK, btnL, btnM, btnN, btnO, btnP, btnQ,
        btnR, btnS, btnT, btnU, btnW, btnX, btnY, btnZ, btnJ, btnÑ, btnV};

    // Se habilitan los botones de letras al iniciar
    for (JButton boton : botonesLetras) {
        boton.setEnabled( b: true);
        boton.setBackground( bg: Color.decode( nm: "#e0e1dd"));
    }
    lblAyuda.setVisible( aFlag: false);
    lblPista.setVisible( aFlag: false);
    juego = new Ahorcado();
    juego.Inicio();
    txtArea.setText( text: juego.Palabra());

    icono = new ImageIcon(sprites[0]);
    ahorcado.setIcon( icon: icono);

    pistaSelect = false;


}
```

Método private void btnRetroceder

Este método regresa a la jugada anterior de la partida. Se desapila un elemento de la pila para así regresar a la acción actual y se disminuye el número de intentos fallidos. No retorna nada.

```
private void btnRetrocederActionPerformed(java.awt.event.ActionEvent evt) {

    if (juego != null) {   //si la jugada actual es igual a la palabra a adivinar significa que el jugador ya ganó, entonces no podrá retr
        if (Arrays.equals( a: juego.palabraAux,  a2: juego.letras)) {
            JOptionPane.showMessageDialog( parentComponent: null,  message: "No se puede retroceder, ya ganaste...",  title: "Error",  messageType: JOptionP
        } else {   //si el número de intentos fallidos llegó a su máximo significa que el jugador ya perdió y no podrá retroceder.
            if (juego.intentoActual == numIntentosMAX) {
                JOptionPane.showMessageDialog( parentComponent: null,  message: "No se puede retroceder, ya perdiste...",  title: "Error",  messageType: JOp
            } else {
                if (juego.intentoActual > 0) {   //se controla que el número de intento actual sea mayor a cero para evitar errores

                    /* si el tamaño de la pila es mayor a uno significa que se ha realizado más de una jugada, entonces se retrocederá
                    a la jugada anterior almacenada en la pila */
                    if (juego.pila.Size() > 1) {
                        juego.pila.Desapilar();                                     //se retrocede de jugada al desapilar un elemento de l
                        juego.letras = ((String[]) juego.pila.Peek()).clone();      //la jugada actual se actualiza al último elemento de
                        txtArea.setText( text: juego.Palabra());                    //se actualiza el txtArea que contiene la palabra
                        System.out.println("Pila retrocedida: " + juego.PrintPila());
                    }

                    juego.intentoActual--;                          //se disminuye el número de intentos fallidos
                    icono = new ImageIcon(sprites[juego.intentoActual]);  //se actualiza la animación a su sprite anterior
                    ahorcado.setIcon( icon: icono);

                } else {
                    JOptionPane.showMessageDialog( parentComponent: null,  message: "No puede retroceder mas",  title: "Error",  messageType: JOptionPane.ER
                }
            }
        }
    } else {
        JOptionPane.showMessageDialog( parentComponent: null,  message: "Ni siquiera ha empezado el juego...",  title: "Error",  messageType: JOptionPane.ER
    }
}
```

Método private void btnPista

Este código permite generar y mostrar una pista en el juego del ahorcado si el juego aún no ha terminado y la pista no ha sido seleccionada previamente.

```
private void btnPistaActionPerformed(java.awt.event.ActionEvent evt) {

    if (juego != null) {

        if (!pistaSelect) {       //si la ayuda de pista no ha sido seleccic

            if (!juego.pila.Llena()) {  //si la pila está llena significa qu

                do {                    //se ejecuta el método de pista mier
                    Pista();
                } while (Arrays.equals( a:pista,  a2:juego.letras));
                //se termina el ciclo cuando la pista generada contiene una

                lblPista.setVisible( aFlag:true);
                lblPista.setText( text:juego.Palabra( palabra:pista));
                pistaSelect = true;          //se marca la pista como selecci
            }

        }
    }
}
```

Método private void btnLimpiar

Se encarga de reiniciar el juego del ahorcado al estado inicial, vaciando la pila, restableciendo las letras, reiniciando el contador de intentos, reiniciando la animación y ocultando las ayudas en la interfaz gráfica.

```
  private void btnLimpiarActionPerformed(java.awt.event.ActionEvent evt) {

      if (juego != null) {

          while (juego.pila.Size() > 1) {          //ciclo para vaciar la pi
              juego.pila.Desapilar();
          }
          juego.letras = ((String[]) juego.pila.Peek()).clone();      //el a
          juego.intentoActual = 0;                                    //se r
          icono = new ImageIcon(sprites[0]);                          //se r
          ahorcado.setIcon( icon:icono);
          txtArea.setText( text:juego.Palabra());                      //se a
          lblPista.setVisible( aFlag:false);                          //se o
          pistaSelect = false;
          lblAyuda.setVisible( aFlag:false);
      } else {
          JOptionPane.showMessageDialog( parentComponent:null,  message:"Ni siquie
      }
  }
```

Método Pista

Este método proporciona una pista al jugador. Se añade una letra al estado actual del juego para así dar una pequeña. Ayuda al jugador. Se obtiene una letra aleatoria de la palabra a adivinar, se comprueba que dicha letra no esté ya almacenada en la jugada actual y se añade esa letra nueva al arreglo de la pista el cual se imprime en un label. El método es auxiliar y no afecta como tal al estado del juego. No retorna nada.

```
public void Pista() {
    Random random = new Random();                          //se utiliza el paquete randc
    pista = ((String[]) juego.pila.Peek()).clone();        //se almacena en el arreglo d

    int indiceAleatorio = 0;                               //se declara una variable pa
    int j = 0;                                             //contador para recorrer el
    while (j <= pista.length) {                            //se realiza un ciclo mientr
        indiceAleatorio = random.nextInt( bound: pista.length);  //se genera un numero alea

        if (pista[j] != null && pista[j].equals(juego.palabraAux[indiceAleatorio])) //se
        {                                                                           //
            j++;                                                                    //s
        } else //si la letra de ayuda no se encuentra en la jugada....
        {
            pista[indiceAleatorio] = juego.palabraAux[indiceAleatorio];   //se asigna di
            break;
        }
    }
}
```

Método public void Intento

Este método realiza un intento del juego. Recibe como parámetros una cadena llamada letra y un botón. La letra recibida se buscará en la palabra a adivinar, si se encuentra se almacenará en la posición correspondiente en el arreglo de letras, se apilará dicho arreglo y se pintará el botón correspondiente a la letra acertada. Si no ha sido encontrada la letra, se incrementará el intento actual y se actualizará el sprite. Si el arreglo de letras es igual a la palabra a adivinar, el juego se termina. Si el número de intentos ha llegado a su máximo, el juego termina. No retorna nada.

- letra Letra que representa el intento del jugador.
- btn Boton correspondiente a la letra que se desea intentar adivinar.

```
public void Intento(String letra, JButton btn) {

    boolean encontrado = false;       //variable boolean auxiliar para verif
    for (int i = 0; i < juego.letras.length; i++) {
        if (juego.palabraAux[i].equals( anObject: letra)) { //si la palabra a a
            juego.letras[i] = letra;                        //las posiciones o
            encontrado = true;
        }

    }

    if (!encontrado) {      //si la letra NO ha sido encontrada...

        juego.intentoActual++;
        if (juego.intentoActual == numIntentosMAX) {

            JOptionPane.showMessageDialog( parentComponent: null, "Perdiste! La p
            DeshabilitarBotones();

        } else {

            icono = new ImageIcon(sprites[juego.intentoActual]);
            ahorcado.setIcon( icon: icono);

        }
    } else {            //si la letra SI ha sido encontrada...

        btn.setBackground( bg: Color.decode( nm: "#415a77"));                  //
        if (Arrays.equals( a: juego.palabraAux,  a2: juego.letras)) { //si el a
```

Método public void Desabilitar

Este método deshabilita todos los botones

Método utilizado al momento de que el jugador falle o gane la partida y así evitar errores. También se utiliza al cargar el formulario. No retorna nada

```
public void DeshabilitarBotones() {
    JButton[] botonesLetras = {btnA, btnB, btnC, btnD, btnE, btnF, btnG, btnH, btnI, btnK, btnL, btnM,
        btnN, btnO, btnP, btnQ, btnR, btnS, btnT, btnU, btnW, btnX, btnY, btnZ,
        btnJ, btnÑ, btnV};                                  //arreglo que contiene todos los botones representativos al alfabeto

    for (JButton boton : botonesLetras) {
        boton.setEnabled( b: false);                //se deshabilita cada botón
    }
}
```

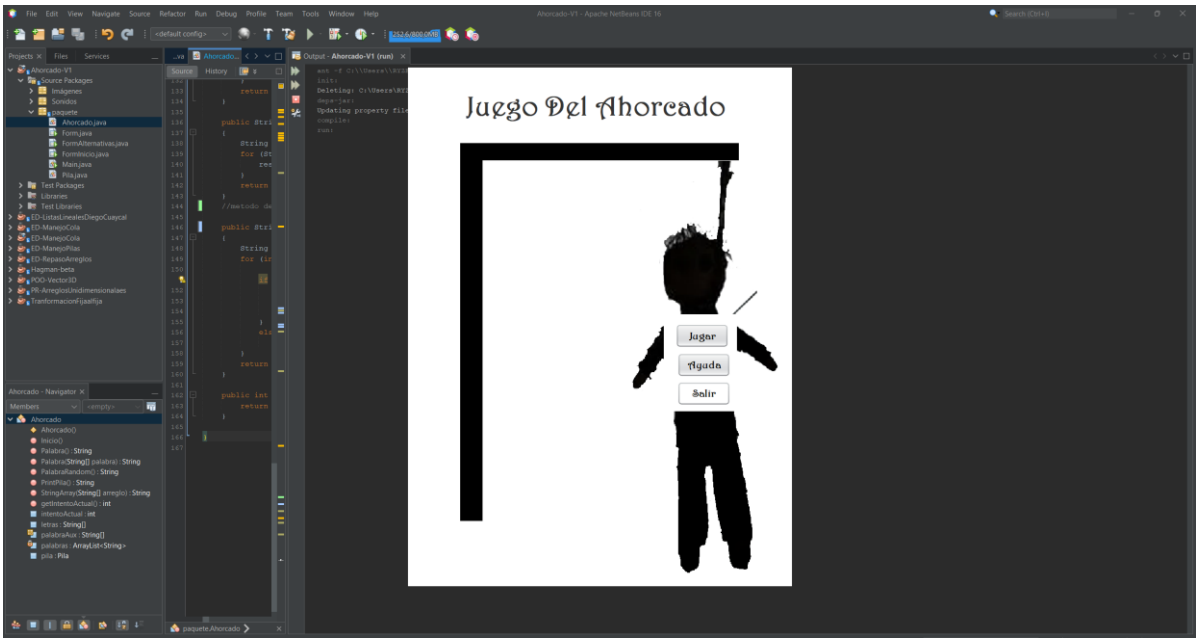
CONCLUSIONES:

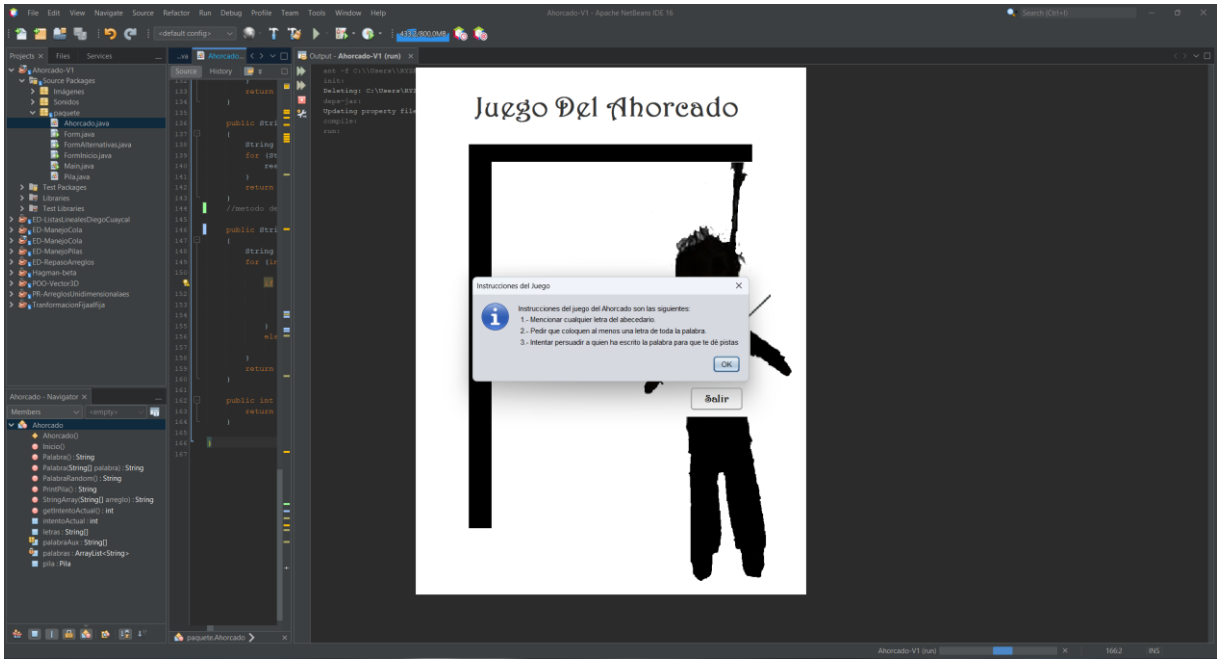
- La implementación de la estructura de datos pila en el juego del ahorcado destaca la importancia de mantener el estado y la coherencia de los datos en una

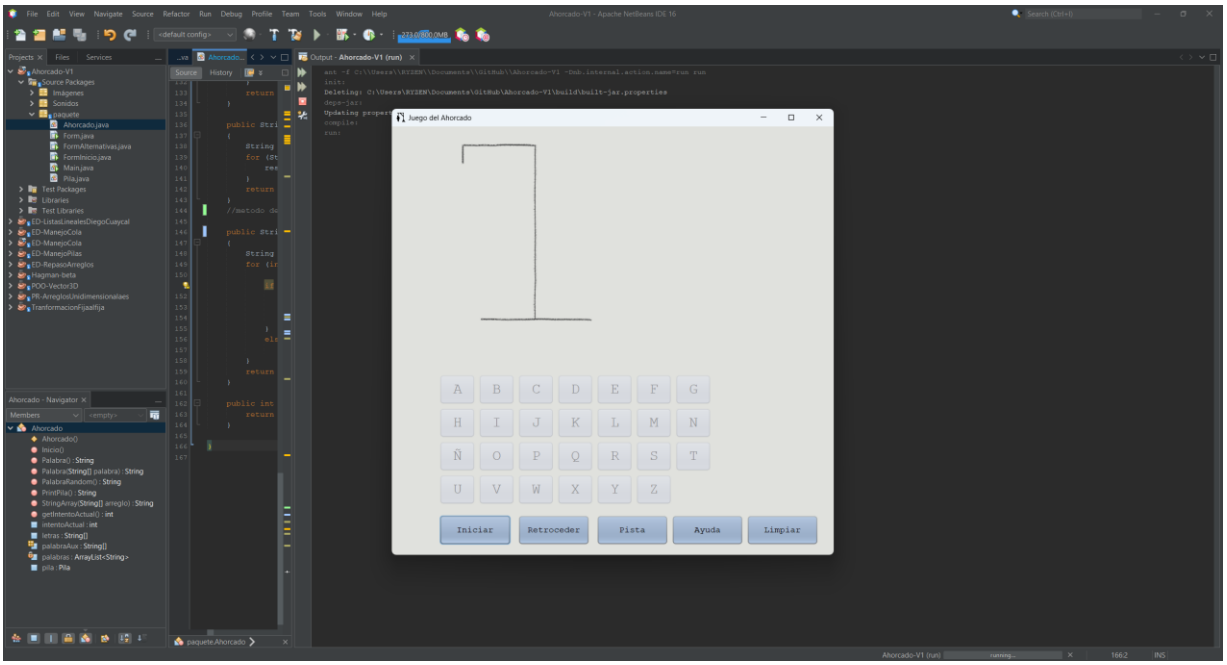
aplicación interactiva. La pila asegura que las letras adivinadas se mantengan en el orden correcto y se puedan recuperar en el momento adecuado.

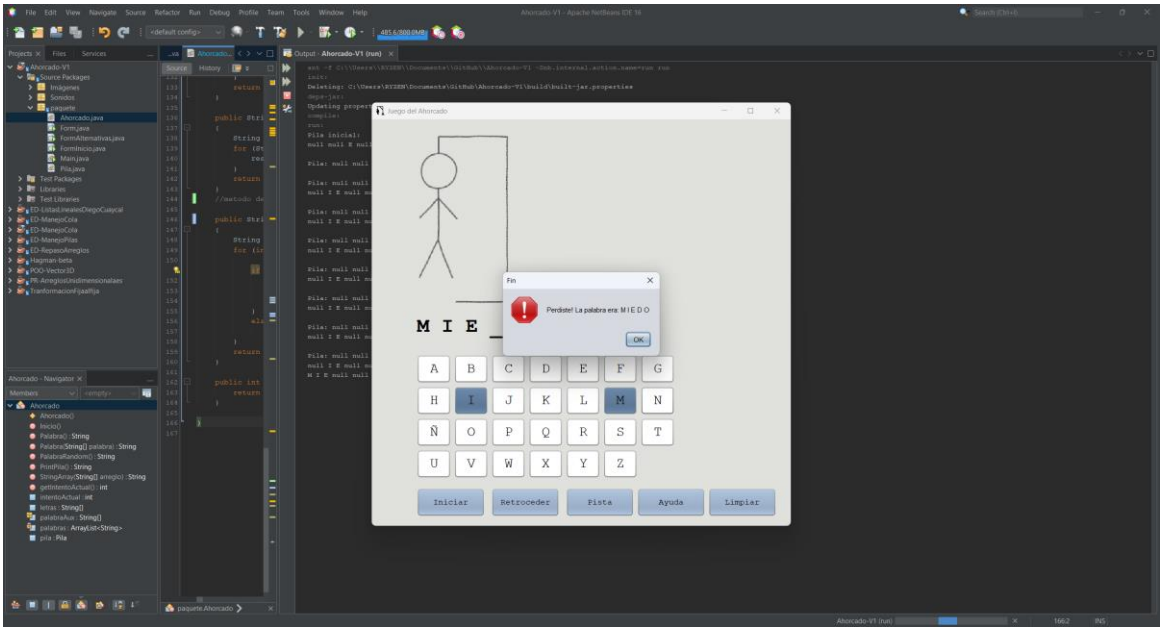
- La implementación de la estructura de datos pila en el juego del ahorcado en Apache NetBeans demuestra la importancia de elegir la estructura de datos adecuada para resolver un problema específico. En este caso, la pila resulta ser una elección apropiada debido a su naturaleza LIFO (último en entrar, primero en salir), que se ajusta al comportamiento del juego.
  Al utilizar una pila, se logra un seguimiento eficiente de las letras adivinadas en el juego del ahorcado. Cada letra adivinada se inserta en la pila, lo que permite un fácil acceso a la última letra adivinada y su eliminación en caso de un error. Esto simplifica la lógica del juego y facilita el manejo de las adivinanzas del jugador.

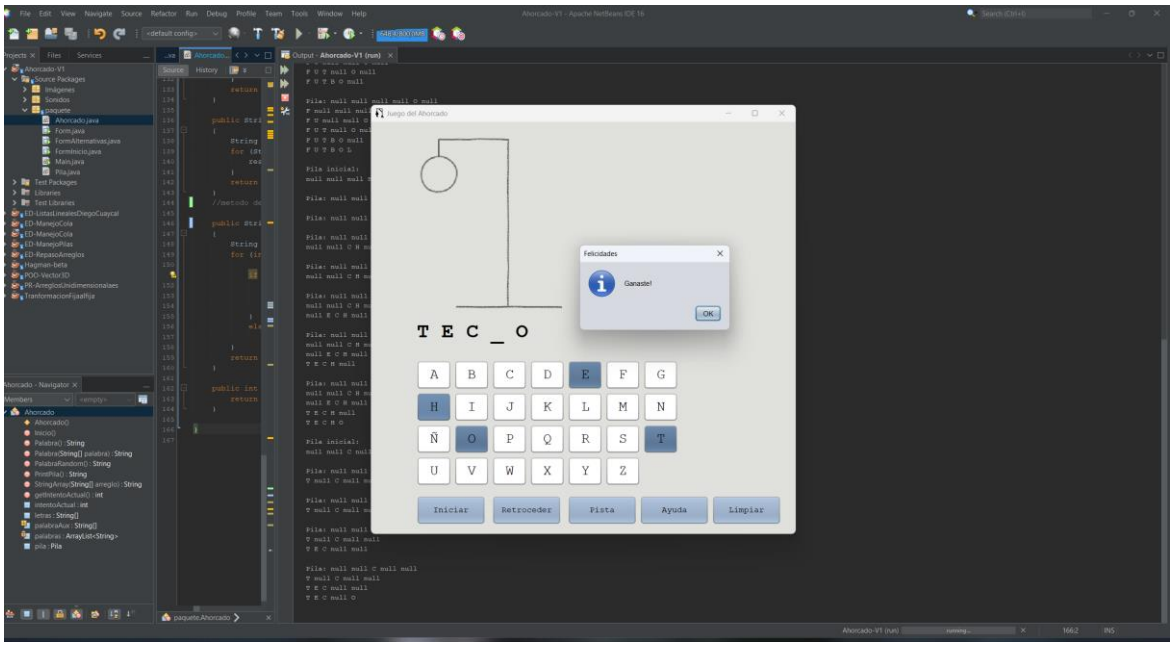
ANEXOS:


Juego Del Ahorcado
Jugar
Ayuda
Salir


Juego Del Ahorcado
Instrucciones del Juego
Instrucciones del juego del Ahorcado son las siguientes:
1.- Mencionar cualquier letra del abecedario.
2.- Pedir que coloquen al menos una letra de toda la palabra.
3.- Intentar persuadir a quien ha escrito la palabra para que te dé pistas
OK
Salir


Juego del Ahorcado
A B C D E F G
H I J K L M N
Ñ O P Q R S T
U V W X Y Z
Iniciar
Retroceder
Pista
Ayuda
Limpiar


Juego del Ahorcado
M I E _
Fin
Perdiste! La palabra era: M I E D O
OK
A B C D E F G
H I J K L M N
Ñ O P Q R S T
U V W X Y Z
Iniciar
Retroceder
Pista
Ayuda
Limpiar



BIBLIOGRAFÍAS:

Que Son Las Pilas estructura de Datos - EstructuraDeDatos (no date) *Google Sites: Sign-in*. Available at: https://sites.google.com/site/miguelestructura/que-son-las-pilas-en-estructuras-de-datos (Accessed: 19 May 2023).

Pila (Informática) (2023) *Wikipedia*. Available at: https://es.wikipedia.org/wiki/Pila_%28inform%C3%A1tica%29 (Accessed: 19 May 2023).

Free APA citation generator & format: CITE this for me (2021) *Cite this for Me | Free Reference Generator – Harvard, APA, MLA, Chicago...* Available at: https://www.citethisforme.com/citation-generator/apa (Accessed: 19 May 2023).

Stack Data Structure (no date) *Programiz*. Available at: https://www.programiz.com/dsa/stack (Accessed: 19 May 2023).