UNIVERSIDAD DE LAS FUERZAS ARMADAS-ESPE

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN - DCCO-SS

CARRERA DE INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN

PERIODO	:	PREGRADO OCTUBRE 2025 - MARZO 2026
ASIGNATURA	:	Estructura de datos
TEMA	:	Laboratorio1Torres de Hanoi con Interfaz gráfica en java
ESTUDIANTE	:	Angel Steven Rodriguez Chavez
NIVEL-PARALELO - NRC	:	30748
DOCENTE	:	Ing. Margoth Guaraca M.
FECHA DE ENTREGA	:	17/10/2025

SANTO DOMINGO – ECUADOR

Tabla de contenido:

Tabla de Figuras

1. Introducción

El juego de las Torres de Hanoi implica mover un conjunto de discos apilados de forma decreciente (del más grande en la base al más pequeño en la parte superior) desde una torre inicial (Torre A) hasta una torre de destino (Torre C), utilizando una tercera torre auxiliar (Torre B) y siguiendo estas reglas:

1. Mover un disco a la vez: Solo se puede mover un disco en cada movimiento.
2. Regla de tamaño: No se puede colocar un disco más grande sobre un disco más pequeño.
3. Usar la torre auxiliar: La torre auxiliar ayuda en el proceso de transferencia de los discos entre la torre de origen y la torre de destino.

El objetivo es mover todos los discos de la torre de origen a la torre de destino en el menor número de movimientos posible, que es 2n−1, donde n es el número de discos.

Solución Recursiva para las Torres de Hanoi La solución recursiva del problema se basa en descomponer el problema de mover n discos en problemas más pequeños de mover n−1 discos:

1. Caso base (n = 1): Si solo hay un disco, se mueve directamente de la torre de origen a la torre de destino.
2. Paso recursivo (n > 1):
   - Paso 1: Mover los primeros n−1 discos desde la torre de origen (A) hasta la torre auxiliar (B), usando la torre de destino (C) como apoyo.
   - Paso 2: Mover el disco más grande (el n-ésimo disco) directamente desde la torre de origen (A) hasta la torre de destino (C).
   - Paso 3: Mover los n−1n-1n−1 discos desde la torre auxiliar (B) hasta la torre de destino (C), utilizando la torre de origen (A) como apoyo.

Esta lógica de dividir el problema en partes más pequeñas se aplica recursivamente hasta que solo queda un disco, que es el caso base.

2. Objetivos

2.1. Objetivo General:

- Crear una aplicación gráfica en Java para simular el problema de las Torres de Hanoi,

2.2. Objetivos Específicos:

- Comprender la finalidad de las soluciones recursivas

- Integrar en la solución interfaz gráfica.
- Visualizar los discos con colores únicos.
- Controlar la velocidad de la animación.

3. Desarrollo / Marco Teórico/ Práctica

3.1. Definición y Fundamentos de la Recursividad

La **recursividad** es un concepto fundamental en informática y matemáticas donde la solución de un problema depende de las soluciones a instancias más pequeñas del mismo problema. En programación, se implementa mediante una **función o método que se llama a sí mismo** de forma repetida.

La recursividad permite resolver problemas complejos (como las Torres de Hanoi) de una manera muy **elegante y concisa**, modelando la solución de forma natural a la definición del problema.

Para la realización de esta práctica, se siguió la metodología descrita en la guía de laboratorio, desarrollando una aplicación en Java que simula la resolución del problema de las Torres de Hanoi mediante un algoritmo recursivo y una interfaz gráfica de usuario (GUI).

El proyecto se estructuró en tres paquetes para separar responsabilidades, tal como se solicitó:

1. **logicarecursividad**: Contiene la clase TorresDeHanoi, que encapsula la lógica recursiva para resolver el problema.
2. **interfaz**: Incluye la clase TorresDeHanoiPanel, responsable de toda la representación gráfica de las torres y los discos.
3. **ejecutable**: Contiene la clase principal TorresDeHanoiApp, que construye la ventana de la aplicación, inicializa los componentes de la interfaz y maneja los eventos del usuario, como iniciar o reiniciar la simulación.

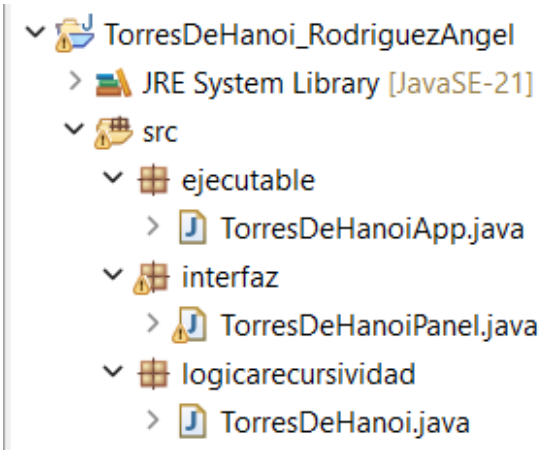

Figura 3.1 Estructura de paquetes y clases en Eclipse.

El flujo de la aplicación comienza en TorresDeHanoiApp, donde se crea la ventana principal y un panel de control. Este panel permite al usuario introducir el número de discos, ajustar la velocidad de la animación y comenzar la simulación. Al presionar el botón "Iniciar", se crea una instancia de TorresDeHanoi y se invoca al método resolver en un nuevo hilo para no bloquear la interfaz gráfica durante la animación.

Clase	**Paquete**	**Función Principal**	**Métodos Clave**
TorresDeHanoiApp	ejecutable	Crea y configura la ventana principal (JFrame) y los controles de la GUI. Inicia la simulación.	main()
TorresDeHanoiPanel	interfaz	Dibuja las torres y los discos en un panel. Mantiene el	paintComponent(), moverDisco(), reiniciar()

Clase	Paquete	Función Principal	Métodos Clave
		estado de las pilas de discos.	
TorresDeHanoi	logicarecursividad	Implementa el algoritmo recursivo para resolver el problema.	resolver(numDiscos, origen, auxiliar, destino, delay)

- Código de la clase **TorresDeHanoi.java:**

Esta clase contiene el **cerebro del algoritmo recursivo**. El método resolver es el encargado de ejecutar la lógica para mover los discos. Recibe como parámetros el número de discos a mover, las torres de origen, auxiliar y destino, y un retardo (delay) para controlar la velocidad de la animación. La recursión se detiene cuando numDiscos llega a 0. Dentro de este método se realizan las dos llamadas recursivas y la llamada a panel.moverDisco para actualizar la interfaz gráfica, siguiendo la estrategia teórica del problema.

TorresDeHanoi.java

```java
package logicarecursividad;
import interfaz.TorresDeHanoiPanel;
public class TorresDeHanoi {
private TorresDeHanoiPanel panel;

public TorresDeHanoi(TorresDeHanoiPanel panel) {
    this.panel=panel;}
public void resolver(int numDiscos, int origen, int auxiliar,int destino, int delay) {
    if(numDiscos>0) {
        //Paso 1: Mover todos los discos menos el ultimo del origen al auxiliar
        resolver (numDiscos-1, origen, destino, auxiliar, delay);
        //Paso 2: Mover el disco mas grande del origen al destino
        panel.moverDisco(origen, destino);
        try {
            Thread.sleep (delay);
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
        //Paso 3: Mover los discos en auxiliar al destino
        resolver (numDiscos-1, auxiliar, origen, destino, delay);
    }
}
}
```

Figura 3.2 Código de la clase TorreDeHanoi.

- Código de la clase **TorresDeHanoiPanel.java**:

Esta clase, que hereda de JPanel, es la **responsable de toda la parte visual** de la simulación. Utiliza una estructura de datos de tipo Stack para representar cada una de las tres torres y almacenar los discos (representados por enteros). El método paintComponent se encarga de dibujar el estado actual de las torres y los discos en la ventana cada vez que se produce un cambio. El método moverDisco actualiza el estado de las torres (haciendo pop del origen y push al destino) y llama a repaint() para forzar el redibujado. También se encarga de asignar colores únicos a cada disco para una mejor visualización.

TorresDeHanoiPanel.java

```java
package interfaz;
import javax.swing.*;
import java.awt.*;
import java.util.ArrayList;
import java.util.List;
import java.util.Stack;

public class TorresDeHanoiPanel extends JPanel {
    private Stack<Integer>[] torres;
    private List<Color> discoColores;
    private int movimientos;
    public TorresDeHanoiPanel(int numDiscos) {
        torres = new Stack[3];
        discoColores = new ArrayList<>();
        movimientos = 0;
        for (int i = 0; i < 3; i++) {
            torres[i] = new Stack<>();
        }
        for(int i=numDiscos; i> 0; i--) {
            torres[0].push(i);
            discoColores.add(new Color((int)(Math.random() * 0x1000000)));
        }
        setPreferredSize(new Dimension(600, 300));
        setBackground(Color.WHITE);

    }
    public void moverDisco(int origen, int destino) {
        int disco= torres[origen].pop();
        torres[destino].push(disco);
        movimientos++;
        repaint();
    }
```

Figura 3.3Código de la clase TorresDeHanoiPanel.

TorresDeHanoiPanel.java

```
33      @Override
34      protected void paintComponent (Graphics g) {
35          super.paintComponent(g);
36
37          int torreWidth = getWidth() / 4;
38          int torreHeight = getHeight() / 2;
39          int discoHeight = 20;
40
41          for(int i = 0; i<3;i++) {
42          int torreX= (i+1)* torreWidth - torreWidth / 2;{
43          int torreY= getHeight() - torreHeight;
44          g.setColor(Color.BLACK);
45          g.fillRect(torreX, torreY, 10, torreHeight);
46
47          int discoY= getHeight() - discoHeight;
48          for(int disco: torres[i]) {
49              int discoWidth= disco *20;
50              int discoX= torreX - discoWidth / 2 + 5;
51              g.setColor(discoColores.get(disco - 1));
52              g.fillRect(discoX, discoY, discoWidth, discoHeight);
53              discoY -= discoHeight;
54          }
55          }
56          g.setColor(Color.BLACK);
57          g.drawString("Movimientos: " + movimientos, 10, 20);
58          }}
59          public void reiniciar(int numDiscos) {
60          movimientos=0;
61          for (int i=0; i<3;i++) {
62              torres[i].clear();
63          }
```

Figura 3.4 Código de la clase TorresDeHanoiPanel part2.

```
64          discoColores.clear();
65          for(int i=numDiscos; i> 0; i--) {
66              torres[0].push(i);
67              discoColores.add(new Color((int)(Math.random() * 0x100000)));
68          }
69          repaint();
70      }
71  }
```

Figura 3.5 Código de la clase TorresDeHanoiPanel part3.

Código de la clase **TorresDeHanoiApp.java**:

Esta es la **clase principal que inicia la aplicación**. En su método main, se crea la ventana (JFrame) y se configuran todos los componentes de la interfaz de usuario, como etiquetas, un campo de texto para el número de discos, un deslizador (JSlider) para la velocidad y los botones de "Iniciar" y "Reiniciar". Lo más importante es que aquí se definen los ActionListener para los botones. El ActionListener del botón "Iniciar" captura la entrada del usuario, crea un nuevo hilo (Thread) para ejecutar el algoritmo resolver y así evita que la interfaz gráfica se congele durante la animación.

TorresDeHanoiApp.java

```java
package ejecutable;
import interfaz.TorresDeHanoiPanel;
import logicarecursividad.TorresDeHanoi;
import javax.swing.*;
import java.awt.*;
public class TorresDeHanoiApp {
    public static void main(String[] args) {
        TorresDeHanoiPanel panel = new TorresDeHanoiPanel(3);
        JFrame frame=new JFrame("Torres de Hanoi");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLayout(new BorderLayout());

        JPanel controlPanel = new JPanel();
        JLabel label = new JLabel("Numero de Discos:");
        JTextField numDiscosField = new JTextField(5);
        JButton startButton = new JButton("Iniciar");
        JButton resetButton = new JButton("Reiniciar");

        JLabel velocidadLabel = new JLabel("Velocidad:");
        JSlider velocidadSlider = new JSlider(1, 10, 5);

        controlPanel.add(label);
        controlPanel.add(numDiscosField);
        controlPanel.add(velocidadLabel);
        controlPanel.add(velocidadSlider);
        controlPanel.add(startButton);
        controlPanel.add(resetButton);

        frame.add(controlPanel, BorderLayout.NORTH);
        frame.add(panel, BorderLayout.CENTER);
        frame.pack();
        frame.setLocationRelativeTo(null);
        frame.setVisible(true);

```

Figura 3.6 Código de la clase TorresDeHanoi.

```java
        startButton.addActionListener(e ->{
            try {
                int numDiscos = Integer.parseInt (numDiscosField.getText());
                panel.reiniciar(numDiscos);

                TorresDeHanoi hanoi = new TorresDeHanoi(panel);
                int delay= 1000 / velocidadSlider.getValue();
                new Thread(() -> hanoi.resolver(numDiscos, 0, 1, 2, delay)).start();
            }catch(NumberFormatException ex) {
                JOptionPane.showMessageDialog(frame, "Por favor ingrese un numero valido de discos.", "Entrada invalida", JOptionPane.ERROR_MESSAGE);
            }
        });
        resetButton.addActionListener (e -> {
            int numDiscos = Integer.parseInt(numDiscosField.getText());
            panel.reiniciar(numDiscos);
        });
    }
}
```

Figura 3.7 Código de la clase TorresDeHanoi part2.

3.2. Ejecución del código:

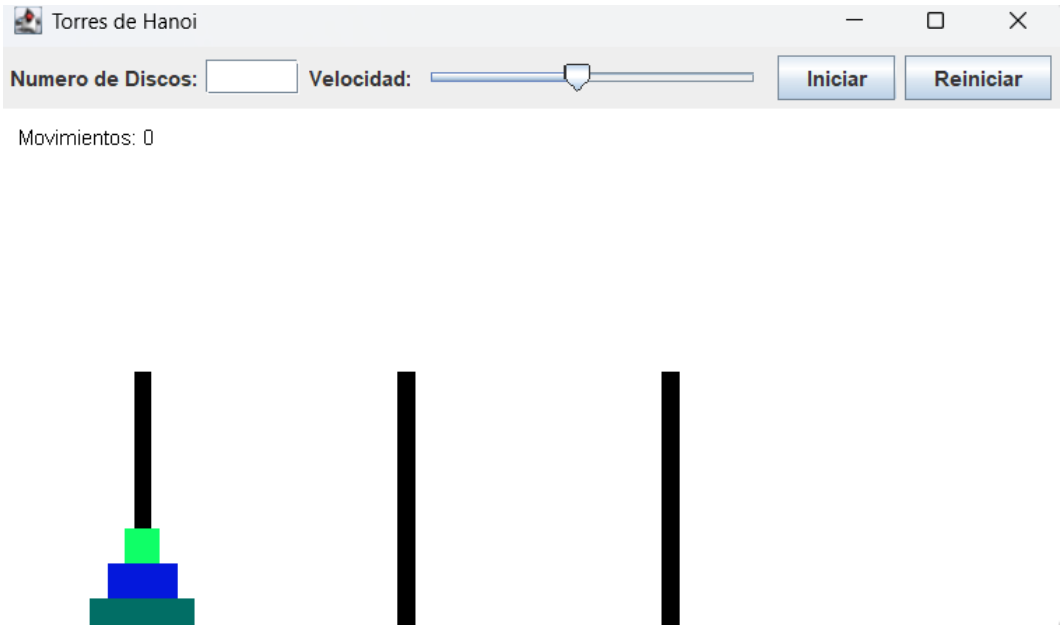

Figura 3.8 Ventana principal de la aplicación al iniciar.

Ejecución de la simulación con 5 discos

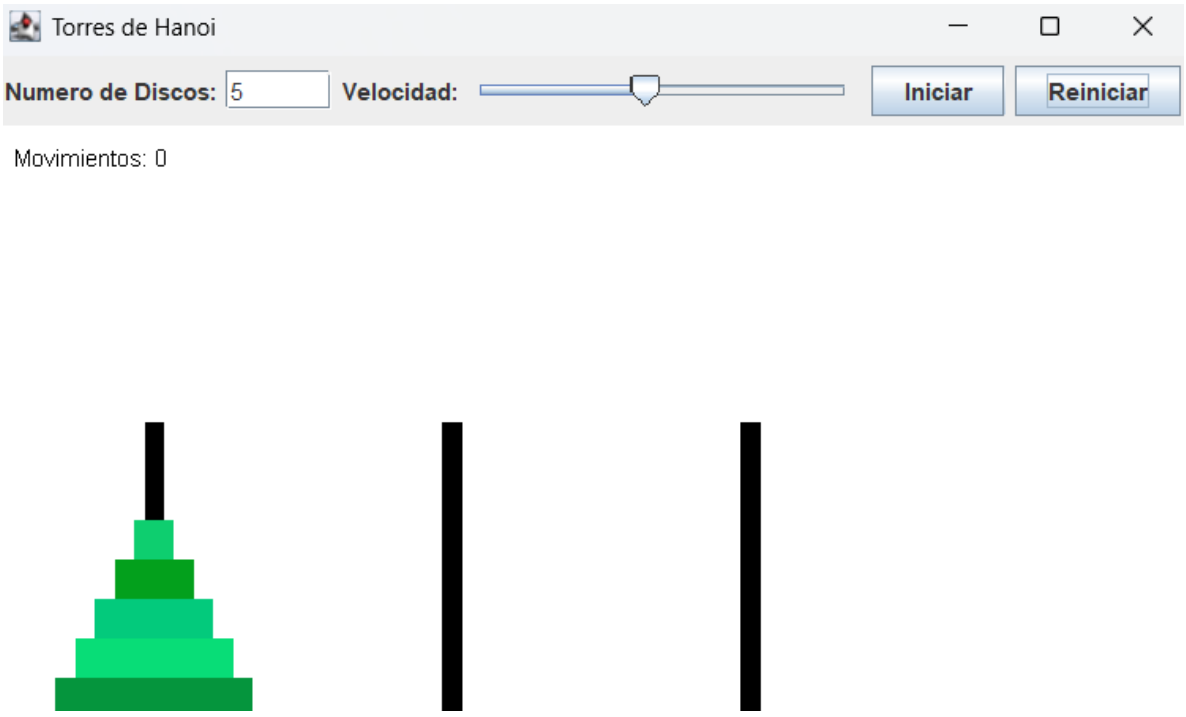

Figura 3.9 Ejecución con 5 discos (inicio).

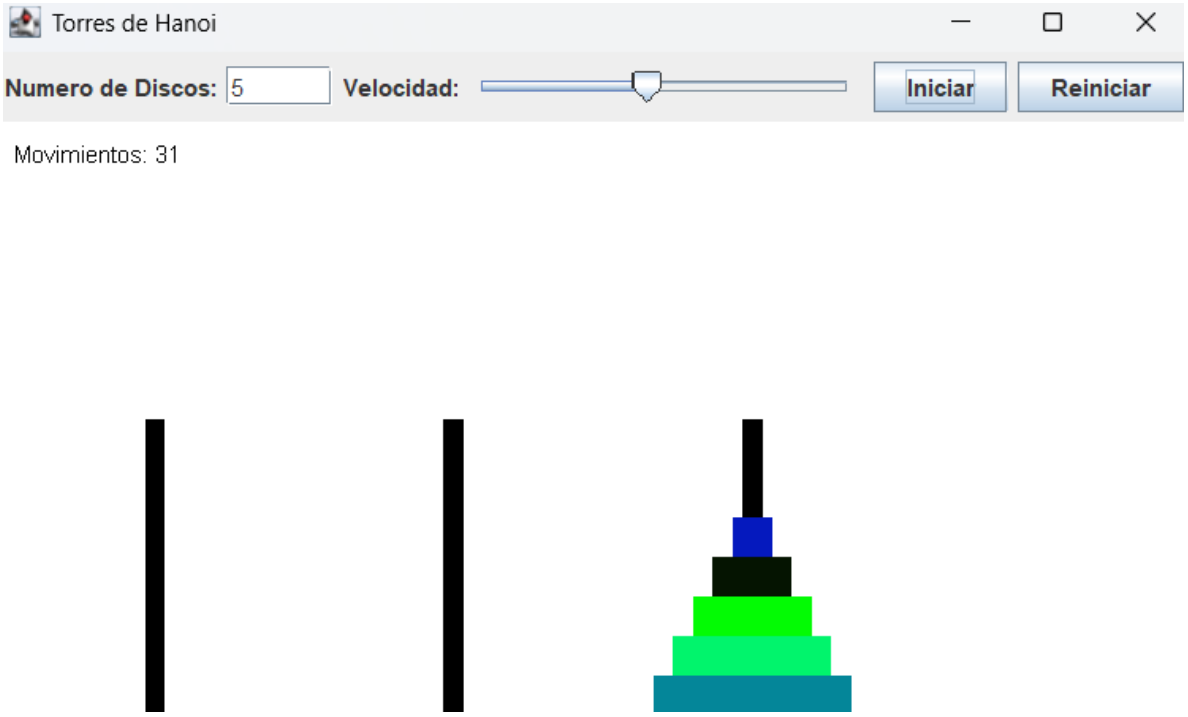

Figura 3.10 Ejecución con 5 discos (final).

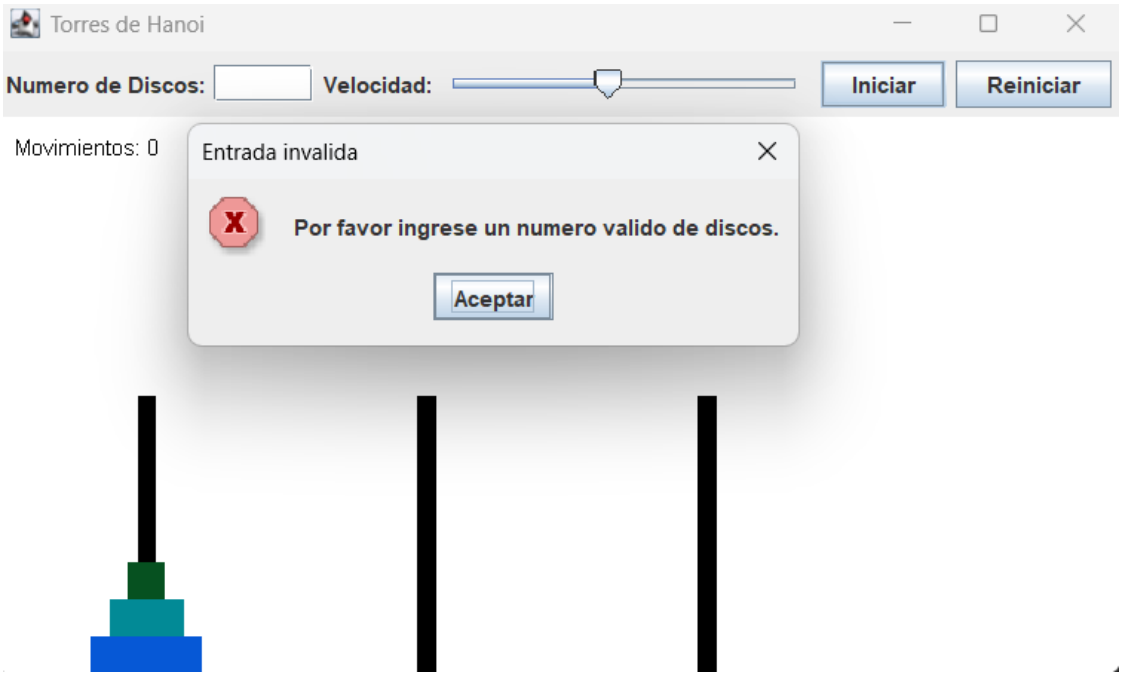

Figura 3.11 Ejecución entrada invalida (dato vacio).

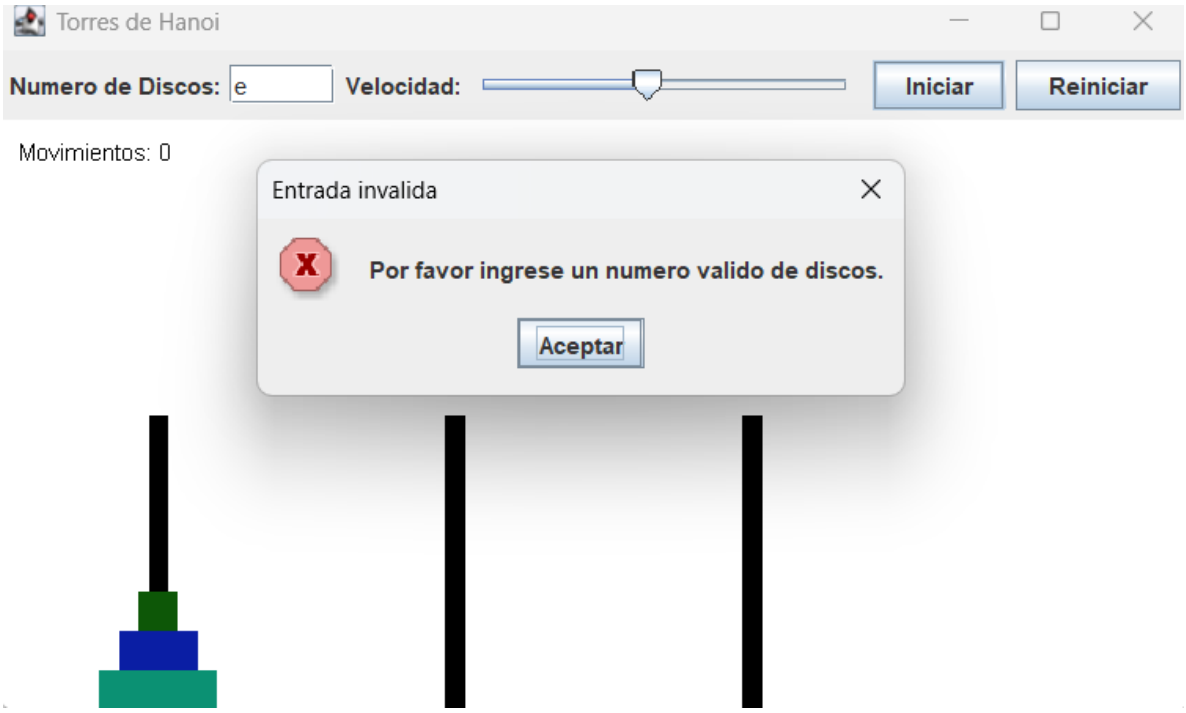

Figura 3.12 Ejecución entrada invalida (dato no numérico).

3.3. Análisis de recursividad

A continuación, se detalla el análisis de la solución recursiva implementada, como se solicita en el **Paso 6** de la guía.

1. Caso Base Utilizado en el Aplicativo

El caso base en la implementación del método resolver de la clase TorresDeHanoi es la condición que detiene las llamadas recursivas. En este código, la recursión se ejecuta dentro de la condición if(numDiscos > 0). Por lo tanto, el caso base se alcanza cuando numDiscos es igual a 0. Cuando esto ocurre, el método no realiza ninguna acción y simplemente retorna, finalizando esa rama de la recursión.

2. Método Recursivo y su Funcionamiento

El método recursivo se llama **resolver** y se encuentra en la clase TorresDeHanoi.

Firma del método: public void resolver(int numDiscos, int origen, int auxiliar,int destino, int delay)

Funcionamiento: Este método está diseñado para mover un número determinado de discos (numDiscos) desde una torre de origen a una torre de destino, utilizando una torre auxiliar. Su lógica se basa en el principio de "divide y vencerás":

1. **Llamada Recursiva 1**: Primero, mueve numDiscos - 1 discos desde la torre de origen a la torre auxiliar. Para esta llamada, la torre de destino original actúa como auxiliar.
2. **Acción Base**: Mueve el disco más grande (el que quedó en la torre de origen) directamente a la torre de destino. Esto se logra llamando a panel.moverDisco(origen, destino), que actualiza la interfaz gráfica. Después, hace una pausa (Thread.sleep(delay)) para controlar la velocidad de la animación.
3. **Llamada Recursiva 2:** Finalmente, mueve los numDiscos - 1 discos que había dejado en la torre auxiliar hacia la torre de destino final, usando la torre de origen como apoyo.

Este proceso se repite hasta que se cumple el caso base (numDiscos == 0).

3. Comparativa de Soluciones Recursivas

A continuación, se presenta una comparativa entre la solución recursiva teórica (presentada en la introducción de la guía) y la solución implementada en el proyecto.

- Similitudes:
    - Ambas soluciones se basan en la misma **estrategia recursiva de tres pasos**: mover n-1 discos a la torre auxiliar, mover el disco base al destino y, finalmente, mover los n-1 discos de la auxiliar al destino.
    - El objetivo en ambas es descomponer el problema principal de mover n discos en subproblemas más pequeños de mover n-1 discos.
- Diferencias:
    - **Definición del Caso Base**: La introducción teórica define el caso base como n = 1, donde el único disco se mueve directamente. La implementación en el código, sin embargo, utiliza numDiscos == 0 como la condición de parada. Aunque conceptualmente diferentes, el resultado es el mismo: la acción de mover un solo disco ocurre cuando a resolver se le llama con numDiscos = 1, y su llamada recursiva posterior con numDiscos = 0 es la que detiene la cadena.
    - **Contexto**: La solución de la introducción es **teórica y abstracta**. En cambio, la solución del proyecto es una **implementación concreta y visual** que incluye elementos prácticos como la manipulación de una interfaz gráfica (panel.moverDisco), la gestión de hilos para la animación y el control de velocidad (delay).
- **Analogía:** La solución teórica es como tener un **manual de instrucciones** que dice: "Para armar el juguete, primero ensamble la sub-parte A, luego una la pieza principal B, y después ensamble la sub-parte C". El manual te da la

estrategia general. La solución implementada en el proyecto es como **ver un video tutorial** que no solo te dice los pasos, sino que te muestra visualmente cómo se ensambla cada sub-parte, pieza por pieza, y te permite pausar o acelerar el video para entenderlo mejor. El video (la aplicación) hace tangible y observable el proceso descrito en el manual (la teoría).

4. Conclusiones

- Se logró desarrollar con éxito una aplicación de escritorio en Java que simula la solución al problema de las Torres de Hanoi, cumpliendo con el objetivo principal de integrar una lógica recursiva con una interfaz gráfica para visualizar su funcionamiento.
- La visualización gráfica del algoritmo fue fundamental para comprender de manera práctica el concepto de recursividad. Observar cómo el problema se descompone en subproblemas más pequeños y cómo las llamadas recursivas se apilan y resuelven facilitó la asimilación de una teoría que puede ser abstracta.
- El uso de un hilo de ejecución separado para la lógica del algoritmo (new Thread(...)) demostró ser una técnica efectiva y necesaria para mantener una interfaz de usuario fluida y receptiva mientras se ejecuta una tarea prolongada como la animación de los movimientos.

5. Recomendaciones

- Implementar una validación de entrada más robusta para el campo de "Número de Discos". Actualmente, la aplicación solo previene errores

si la entrada no es un número, pero sería útil añadir una comprobación para que no se permitan valores negativos o el cero, ya que no son válidos para el problema.

- Para mejorar la experiencia del usuario, se podría añadir una etiqueta que muestre el número mínimo de movimientos teóricos necesarios para resolver el puzzle (2^n - 1). Esto permitiría al usuario comparar la eficiencia del algoritmo con el resultado óptimo esperado.

6. Bibliografía/ Referencias

- Deitel, P., & Deitel, H. (2012). *Cómo programar en Java*. Pearson Educación.
- Guía de Laboratorio ED-U1-LAB 1.1. (2024). *Universidad de las Fuerzas Armadas - ESPE.*