TORRES DE HANOI

MANUAL TÉCNICO

PROGRAMA REALIZADO POR EL ALUMNO ALLEN GIANKARLO ROMÁN VÁSQUEZ CARNET 202004745 PARA EL CURSO INTRODUCCIÓN A LA PROGRAMACIÓN Y COMPUTACIÓN I DE LA ESCUELA DE CIENCIAS Y SISTEMAS DE LA FACULTAD DE INGENIERÍA DE LA UNIVERSIDAD SAN CARLOS DE GUATEMALA

CONTENIDO

Identificación del documento----------2

Lugar, fecha y responsables de la elaboración ----------2

Objetivos y alcances del sistema ----------2

Especificación Técnica ----------3

Lógica del programa ----------4

IDENTIFICACIÓN DEL DOCUMENTO

El presente documento constituye el manual técnico de la práctica 2 “Torres de Hanoi” para entretenimiento, en el cual se proporciona al lector los aspectos que se consideraron para su elaboración.

Es importante destacar que este manual no es un curso de aprendizaje de las funciones de programación utilizadas para el desarrollo del programa, sino es una herramienta que provee los aspectos a conocer para la forma correcta de operación y aplicación de este.

Programa realizado por el alumno Allen Giankarlo Román Vásquez carné 202004745 para el curso Introducción a la Programación de Computadoras 1 de la escuela de ciencias y sistemas de la Facultad de Ingeniería de la Universidad san Carlos de Guatemala

LUGAR, FECHA Y RESPONSABLES DE LA ELABORACIÓN

El programa se elaboró en Huehuetenango para la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, durante la última semana del mes de septiembre y la primera semana de octubre del año 2021 durante el segundo semestre de este y el programa fue realizado por el alumno Allen Giankarlo Román Vásquez carné 202004745 para el curso Introducción a la Programación y Computación 1 de la Escuela de Ciencias y Sistemas de la Facultad de Ingeniería de la Universidad San Carlos de Guatemala.

OBJETIVOS Y ALCANCES DEL SISTEMA

Proveer de una herramienta de entretenimiento educativo a todo el que posea el programa.

Disponer de una interfaz libre y sencilla de utilizar para la utilización de cualquier persona en la que puedan almacenar datos de la jugabilidad minimizando el riesgo de pérdida de información.

Gestionar la configuración para poder tener una experiencia personalizada.

Proveer información de la manera óptima de la resolución de este puzzle ejecutando un algoritmo recursivo.

ESPECIFICACIÓN TÉCNICA (HARDWARE Y SOFTWARE)

Los requisitos para que el sistema pueda ser ejecutado adecuadamente son:

- Procesador: Intel Pentium III 800 MHz (800MHz Intel Pentium III u otro equivalente)
- RAM: 512 MB
- Espacio en disco: 750 MB
- Sistema Operativo: Windows 7, Windows XP, Windows Vista) Windows XP Profesional SP3/Vista SP1/Windows 7 Professional)
- Resolución gráfica: 1024 x 728
- Navegador de internet: Google Chrome, Microsoft Edge, Mozilla Firefox, Vivaldi u Opera
- Herramientas: Java y algún IDE que corra el mismo, en este caso se utilizo NetBeans 8.2.

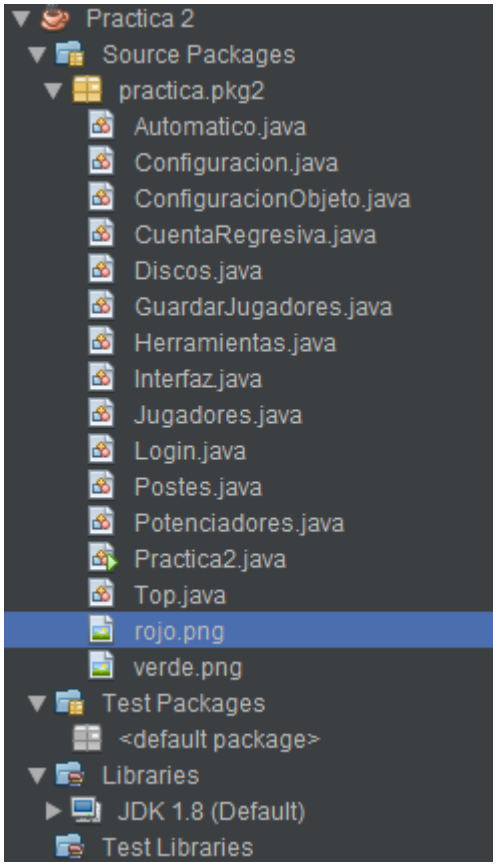
LÓGICA DEL PROGRAMA

Configuración del sistema

El juego de Torres de Hanoi está desarrollado bajo un formato Java de nombre NetBeans en su versión 8.2, esto da la facilidad de estructurar el sistema de manera que se facilita el mantenimiento a dicha solución, a continuación, se describe la estructura básica del sistema y se enfatiza en los archivos y directorios relevantes para su configuración y adaptación.

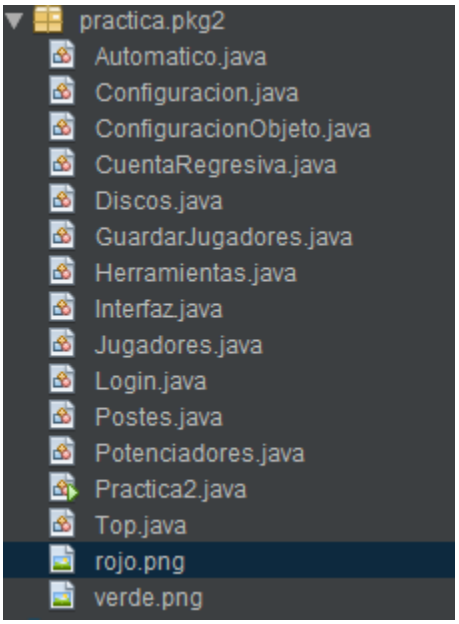
Estructura

La práctica tiene la siguiente estructura.



Directorio

El proyecto tiene la siguiente estructura.



Package proyecto1: en el que se encuentran todas las clases utilizadas para la realización del proyecto.

1. **Práctica 2 (main):** En esta pestaña se deserializa si es el caso y se llama a la pestaña de inicio para desplegar todas las funciones.

```
package practica.pkg2;

import java.io.File;
import java.io.IOException;

/**
 *Practica Torres de Hanoi
 * @author Allen Román
 */
public class Practica2 {

    static int segundos;
    public static Jugadores[] jugadores = new Jugadores[50];
    public static ConfiguracionObjeto[] config = new ConfiguracionObjeto[1];

    public static void main(String[] args) throws IOException {
        segundos = 120;
        Configuracion.n1=3;
        if (new File("Jugadores.bin").exists()) {
            Practica2.jugadores = (Jugadores[]) Herramientas.deserialize("Jugadores.bin");
        }
        if (new File("Configuracion.bin").exists()) {
            Practica2.config = (ConfiguracionObjeto[]) Herramientas.deserialize("Configuracion.bin")
            for (int i = 0; i < 1; i++) {
                segundos = config[i].getsegundos1();
                Configuracion.n1 = config[i].getn1();
            }
        }

        Login ventana = new Login();
        ventana.setVisible(true);

    }

}
```

2. Interfaz: En esta clase se encuentra las principales funcionalidades del juego, entre ellas:

```
package practica.pkg2;

import java.awt.Color;
import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.IOException;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.WindowConstants;
import static practica.pkg2.Login.potenciadores;
import static practica.pkg2.Login.temporizador;
import static practica.pkg2.Practica2.config;
import static practica.pkg2.Practica2.segundos;

public class Interfaz extends JFrame implements ActionListener {

    static int aros;
    static int arosJuego;

    //postes
    static Postes torre1;
    static Postes torre2;
    static Postes torre3;

    public static JLabel lbl_time = new JLabel();
    public static JButton boton = new JButton();
    public static int alturaBoton = 50;

    static JLabel pasosEchos;
    JLabel numeroDeAros;
    JLabel nombreTorre1;
    JLabel nombreTorre2;
    JLabel nombreTorre3;

    JButton b1;
    JButton b2;
    JButton b3;
    JButton b4;
    JButton b5;
    JButton b6;
    JButton SalirInterfaz;
```

Las declaraciones

```
public Interfaz() throws IOException {

    this.setLayout(null);
    this.setTitle("Nuevo Juego");
    this.setSize(1000, 500);
    this.setLocationRelativeTo(null);
    this.getContentPane().setBackground(new Color(174, 214, 241));

    numeroDePasos = 0;
    aros = 3;
    aroMover = null;
    seleccionado = false;
    arosJuego = 3;

    this.setBackground(Color.WHITE);
    this.setLayout(null);


    pasosEchos = new JLabel("Movimientos: ");
    pasosEchos.setBounds(125, 0, 150, 25);
    pasosEchos.setFont(new Font("Century Gothic", 1, 15));
    this.add(pasosEchos);

    lbl_time.setVisible(true);
    lbl_time.setText("Tiempo: ");
    lbl_time.setBounds(10, 0, 125, 25);
    lbl_time.setFont(new Font("Century Gothic", 1, 15));
    setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
    this.add(lbl_time);

    boton.setVisible(true);
    boton.setBounds(25, alturaBoton, 25, 25);
    boton.addActionListener(this);
    setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
    this.add(boton);
```

Declaraciones del entorno gráfico

```
torre1 = new Postes();
torre1.setBounds(100, 75, 220, 300);
switch (Configuracion.n1) {
    case 3:
        agregarAros(3);
        aros=3;
        break;
    case 4:
        agregarAros(4);
        aros=4;
        break;
    case 5:
        agregarAros(5);
        aros=5;
        break;
    case 6:
        agregarAros(6);
        aros=6;
        break;
    case 7:
        agregarAros(7);
        aros=7;
        break;
    default:
        agregarAros(3);
        break;
}
this.add(torre1);
```

Agregar aros al primer poste

```java
public static void agregarAros(int n) {
    Configuracion.n1 = n;
    Discos aro = new Discos();
    aro.setBounds(90, 250, 45, 20);
    torre1.add(aro);
    torre1.updateUI();
    for (int i = 1; i <= n - 1; i++) {
        torre1.add(new Discos());
    }
    organizar(n);
    torre1.updateUI();
}

public static void organizar(int n) {
    Configuracion.n1 = n;
    if (n >= 0) {
        for (int j = 1; j <= n - 1; j++) {
            //panel Anterior
            JPanel anterior = (JPanel) torre1.getComponent(j - 1);
            //posiciones y tamaño del aro anterior
            int x = anterior.getX();
            int y = anterior.getY();
            int w = anterior.getWidth();
            int h = anterior.getHeight();
            //Panel que se va a modificar
            JPanel aroA = (JPanel) torre1.getComponent(j);
            aroA.setBounds(x, y - h, w, h);
            anterior.setBounds(x - 10, y, w + 20, h);
            torre1.setComponentZOrder(aroA, j);
            torre1.setComponentZOrder(anterior, j - 1);
        }
        organizar(n - 1);
    }
}
```

Métodos con los que se agregan y ordenan los aros.

```
if (ae.getSource() == b1) {
    if (torre1.getComponentCount() > 0) {
        if (seleccionado == false) {
            //aro que voy a mover
            aroMover = (Discos) torre1.getComponent(torre1.getComponentCount() - 1);
            seleccionado = true;
            //elimino el aro de la torre
            torre1.remove(torre1.getComponentCount() - 1);
            torre1.updateUI();
        }
    }
    if (seleccionado == true) {
        if (aroMover != null) {

            //guardar el ancho y alto para acomodar
            int x = aroMover.getX();//posicion en x
            int h = aroMover.getHeight();//altura
            int w = aroMover.getWidth();//anchura
            if (torre3.getComponentCount() == 0) {

                //acomodar el aro que entra
                aroMover.setBounds(x, 250, w, h);

                torre3.add(aroMover);
                torre3.updateUI();
                aroMover = null;

                //contador que aumento para contar el numero de pasos del usuario
                numeroDePasos++;

                //Para actualizar el Label de numero de pasos que hace el jugador
                pasosEchos.setText("Movimientos: " + numeroDePasos);
                if (verificarFinalJuego(aros, torre3.getComponentCount()) == true) {
                    GuardarJugadores ventana = new GuardarJugadores();
                    ventana.setVisible(true);
                    this.dispose();
                }
                seleccionado = false;

            } else {
```

Para cada botón que mueve un aro se hacen comprobaciones de primero el aro que se mueve y luego el aro que se elimina.

```
if (seleccionado == true) {
    if (aroMover != null) {

        //guardar el ancho y alto para acomodar
        int x = aroMover.getX();//posicion en x
        int h = aroMover.getHeight();//altura
        int w = aroMover.getWidth();//anchura
        if (torre3.getComponentCount() == 0) {

            //acomodar el aro que entra
            aroMover.setBounds(x, 250, w, h);

            torre3.add(aroMover);
            torre3.updateUI();
            aroMover = null;

            //contador que aumento para contar el numero de pasos del usuario
            numeroDePasos++;

            //Para actualizar el Label de numero de pasos que hace el jugador
            pasosEchos.setText("Movimientos: " + numeroDePasos);
            if (verificarFinalJuego(aros, torre3.getComponentCount()) == true) {
                GuardarJugadores ventana = new GuardarJugadores();
                ventana.setVisible(true);
                this.dispose();
            }
            seleccionado = false;

        } else {
```

Guardar las dimensiones y acomodarlo en el aro correspondiente a la posición y agregar al contador un paso, actualizar el número de pasos, y si es un movimiento inválido se regresa a la posición anterior en este caso al poste.

```
if (ae.getSource() == SalirInterfaz) {
    temporizador.resume();
    Practica2.segundos = 0;
    potenciadores=0;
    this.dispose();
    Login ventana = new Login();
    ventana.setVisible(true);
}

if (ae.getSource() == boton) {
    if (segundos % 2 == 0) {
        segundos=segundos-10;
    } else {
        segundos=segundos+10;
    }
}
```

Botones para salir de la interfaz y para activar los potenciadores.

```
public boolean verificar(Discos aroPresente, Discos aroAMover) {
    int w = aroPresente.getWidth();
    int w2 = aroAMover.getWidth();
    if (w > w2) {
        return true;
    } else {
        return false;
    }
}

public static void setTime(String text) {
    lbl_time.setText(text);
}

public boolean verificarFinalJuego(int n, int numeroArosTorre3) {
    return n == numeroArosTorre3;
}
```

Métodos para verificar el ancho de los aros, setear el tiempo y verificar el juego.

3. Discos

```java
package practica.pkg2;

import java.awt.Color;
import java.util.Random;
import javax.swing.JPanel;
import javax.swing.border.BevelBorder;
import javax.swing.border.Border;
import javax.swing.border.TitledBorder;

/**
 *
 * @author Allen Román
 */
public class Discos extends JPanel{

    public Discos (){

        Random rand = new Random();

        //tres colores bases
        float red = rand.nextFloat();
        float green = rand.nextFloat();
        float blue = rand.nextFloat();

        Color colorAnillo = new Color(red, green, blue);
         //Linea 1
        Border bordejpanel = new TitledBorder(new BevelBo

        //Linea 2
        this.setBorder(bordejpanel);
        this.setBackground(colorAnillo);

    }

}
```

Clase discos en la que se agregan los discos y se elijen un color al azar.

4. Postes

```java
package practica.pkg2;

import java.awt.Color;
import java.awt.Graphics;
import javax.swing.JPanel;

/**
 *
 * @author Allen Román
 */
public class Postes extends JPanel {

    public Postes(){
        this.setLayout(null);
    }

        @Override
    public void paintComponent(Graphics g) {

        super.paintComponent(g);
        this.setBackground(new Color(174, 214, 241));

        g.setColor(Color.BLACK);

        //base
        g.fillRect(10, 270, 200,5);

        //asta
        g.fillRect(110, 30, 5, 240);

    }

}
```

Clase postes en la que se realizan los postes.

5. Cuenta Regresiva

```java
package practica.pkg2;

import javax.swing.JOptionPane;
import static practica.pkg2.Practica2.segundos;
import static practica.pkg2.Login.Juego;

/**
 *
 * @author Allen Román
 */
public class CuentaRegresiva extends Thread {

    private String tiempo;


    @Override
    public void run() {
       try {
            while(true && segundos>=0){
                tiempo = ("Tiempo: "+ segundos );
                Interfaz.lbl_time.setText(tiempo);
               Thread.sleep(1000);
                segundos--;
                if (segundos == 0) {
                    finalizar();
                }
            }

       } catch (InterruptedException ex) {
           ex.printStackTrace();
       }

    }

    public void finalizar() {
        JOptionPane.showMessageDialog(null, "Ha finalizado el juego, ingrese de nuevo por favor");
        System.exit(0);
    }

}
```

Hilo que funciona de temporizador.

6. Potenciadores

```java
package practica.pkg2;

import java.awt.Color;
import static practica.pkg2.Interfaz.alturaBoton;
import static practica.pkg2.Login.potenciadores;
import static practica.pkg2.Practica2.segundos;

/**
 *
 * @author Allen Román
 */
public class Potenciadores extends Thread {

    @Override
    public void run() {
        try {
            while (true && potenciadores >= 0) {

                Thread.sleep(1000);
                alturaBoton=alturaBoton+3;
                potenciadores--;
                if (potenciadores % 2 == 0) {
                    Interfaz.boton.setBackground(new Color(241, 148, 138));
                    Interfaz.boton.setBounds(25, alturaBoton, 25, 25);
                }
                else{
                    Interfaz.boton.setBackground(new Color(130, 224, 170));
                    Interfaz.boton.setBounds(25, alturaBoton, 25, 25);
                }

            }

        } catch (InterruptedException ex) {
            ex.printStackTrace();
        }
    }
}
```

Hilo que funciona para ir variando los potenciadores.

7. Jugadores

```java
package practica.pkg2;

import java.io.Serializable;

/**
 *
 * @author Allen Román
 */
public class Jugadores implements Serializable {
    private int MovimientosJugador;
    private int TiempoJugador;
    private String NombreJugador;

    public Jugadores(int MovimientosJugador, int TiempoJugador, String NombreJugador){
        this.MovimientosJugador = MovimientosJugador;
        this.TiempoJugador= TiempoJugador;
        this.NombreJugador= NombreJugador;

    }

    public int getMovimientosJugador() {
        return MovimientosJugador;
    }

    public void setMovimientosJugador(int MovimientosJugador) {
        this.MovimientosJugador = MovimientosJugador;
    }

    public int getTiempoJugador() {
        return TiempoJugador;
    }

    public void setTiempoJugador(int TiempoJugador) {
        this.TiempoJugador = TiempoJugador;
    }

    public String getNombreJugador() {
        return NombreJugador;
    }

    public void setNombreJugador(String NombreJugador) {
        this.NombreJugador = NombreJugador;
    }


}
```

Clase para guardar el objeto de jugadores.

8. GuardarJugadores

```java
package practica.pkg2;

import java.awt.Color;
import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JTextField;

/**
 *
 * @author Allen Román
 */
public class GuardarJugadores extends JFrame implements ActionListener {

    JTextField ingreso1;
    JButton agregar;
    static int TiempoJugador;

    public GuardarJugadores(){
        TiempoJugador = Practica2.segundos;
        this.setTitle("Guardar Jugador");
        this.setBounds(1000, 500, 500, 250);
        this.setLocationRelativeTo(null);
        this.getContentPane().setBackground(new Color(174, 214, 241));
        this.setLayout(null);

        JLabel titulo = new JLabel();
        titulo.setText("¡Felicitaciones ha ganado el juego!");
        titulo.setBounds(80, 25, 300, 25);
        titulo.setFont(new Font("Century Gothic", 1, 15));
        this.add(titulo);

        JLabel titulo1 = new JLabel();
        titulo1.setText("Guarda tu puntuación");
        titulo1.setBounds(125, 65, 200, 25);
        titulo1.setFont(new Font("Century Gothic", 1, 15));
        this.add(titulo1);

        JLabel label1 = new JLabel();
        label1.setText("Nombre");
        label1.setBounds(50, 100, 75, 25);
        label1.setFont(new Font("Century Gothic", 1, 15));
        this.add(label1);
```

Clase para guardar jugadores, en la primera parte se levanta el entorno gráfico.

```
    @Override
    public void actionPerformed(ActionEvent ae) {

        if (ae.getSource() == agregar) {
            Practica2.jugadores[10] = null;
            for (int i = 0; i < Practica2.jugadores.length; i++) {
                if (Practica2.jugadores[i] == null) {
                    int MovimientosJugador = Interfaz.numeroDePasos;
                    String NombreJugador = ingresol.getText();
                    Jugadores nuevo = new Jugadores(MovimientosJugador, TiempoJugador, NombreJugador);
                    Practica2.jugadores[i] = nuevo;
                    break;
                }
            }
            this.dispose();
            Herramientas.serialize("Jugadores.bin", Practica2.jugadores);
            Herramientas.BurbujaDesc1(Practica2.jugadores);
            Login ventana = new Login();
            ventana.setVisible(true);
        }

    }
```

Y se guarda el objeto.

9. Configuración

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JLabel;
import javax.swing.JTextField;

/**
 *
 * @author Allen Román
 */
class Configuracion extends JFrame implements ActionListener {
    JComboBox NumeroDiscos;
    JTextField Tiempo;
    JButton GuardarConfiguracion;
    JButton SalirConfiguracion;
    static int n1;
    static int segundos1;


    public Configuracion() {...59 lines }
    @Override
    public void actionPerformed(ActionEvent ae) {
        if(ae.getSource()==SalirConfiguracion){
        this.dispose();
        Login ventana = new Login();
        ventana.setVisible(true);
        }

        if(ae.getSource() == GuardarConfiguracion) {

            n1 = (int) NumeroDiscos.getSelectedItem();
            String tiempoconfig = Tiempo.getText();
            segundos1 = Integer.parseInt(tiempoconfig);
            Practica2.segundos = Configuracion.segundos1;
            for (int i = 0; i < Practica2.config.length; i++) {
                ConfiguracionObjeto nuevo1 = new ConfiguracionObjeto(n1, segundo
                Practica2.config[i] = nuevo1;
                Herramientas.serialize("Configuracion.bin", Practica2.config);

            }

        }
    }
```

Clase en la que se guarda la configuración preferida por el usuario.

10. ConfiguraciónObjeto

```java
package practica.pkg2;

import java.io.Serializable;

/**
 *
 * @author Allen Román
 */
public class ConfiguracionObjeto implements Serializable {

    private int nl;
    private int segundosl;

    public ConfiguracionObjeto(int nl, int segundosl) {
        this.nl = nl;
        this.segundosl = segundosl;

    }

    public int getnl() {
        return nl;
    }

    public void setnl(int nl) {
        this.nl = nl;
    }

    public int getsegundosl() {
        return segundosl;
    }

    public void setsegundosl(int segundosl) {
        this.segundosl = segundosl;
    }

}
```

Clase que guarda la configuración.

11. Top

```java
 */
public class Top extends JFrame implements ActionListener {

    static JTextField tabla;
    static JTable tablajugadores;
    JButton SalirTop;

    public Top() {...28 lines }

    public static void TablaJugadores() {
        tabla.removeAll();
        String[] columnas = {"Nombre", "Movimientos", "Tiempo (s)"};
        Object[][] datos = new Object[5][3];

        for (int i = 0; i < Practica2.jugadores.length; i++) {
            if (Practica2.jugadores[i] != null) {

                datos[i][0] = Practica2.jugadores[i].getNombreJugador();
                datos[i][1] = Practica2.jugadores[i].getMovimientosJugador();
                datos[i][2] = Practica2.jugadores[i].getTiempoJugador();

            }

        }
        tablajugadores = new JTable(datos, columnas);
        tablajugadores.setFont(new Font("Century Gothic", 1, 15));
        tablajugadores.setEnabled(false);
        JScrollPane scroll = new JScrollPane(tablajugadores);
        scroll.setBounds(0, 0, 550, 675);
        scroll.setVisible(true);
        tabla.add(scroll);
        Practica2.jugadores[10] = null;

    }

    @Override
    public void actionPerformed(ActionEvent ae) {
        if (ae.getSource() == SalirTop) {
            this.dispose();
            Login ventana = new Login();
            ventana.setVisible(true);
        }
    }
}
```

Clase para imprimir el Top de Jugadores.

12. Automático

```java
package practica.pkg2;

import java.awt.Color;
import java.awt.Font;
import java.awt.TextArea;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;

/**
 *
 * @author Allen Román
 */
public class Automatico extends JFrame implements ActionListener {

    int contador;
    JButton SalirAutomatico;
    TextArea Procedimiento;

    public Automatico() {...39 lines }

    @Override
    public void actionPerformed(ActionEvent ae) {
        if (ae.getSource() == SalirAutomatico) {
            this.dispose();
            Login ventana = new Login();
            ventana.setVisible(true);
        }

    }

    public void Hanoi(int n, int origen, int auxiliar, int destino) {
        if (n == 1) {
            Procedimiento.append("Mover disco "+n+ " de poste " + origen + " a poste " + destino);
            Procedimiento.append(System.getProperty("line.separator"));
            contador++;

        } else {
            Hanoi(n - 1, origen, destino, auxiliar);
            Procedimiento.append("Mover disco "+n+ " de poste " + origen + " a poste " + destino);
            Procedimiento.append(System.getProperty("line.separator"));
            Hanoi(n - 1, auxiliar, origen, destino);
            contador++;
        }
    }

}
```

Clase que imprime y realiza las Torres de Hanoi utilizando un algoritmo recursivo.

13. Herramientas

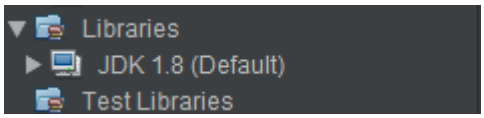
```java
public static void BurbujaDesc1(Jugadores[] arr) {
    int n = 0;
    for (int i = 0; i < arr.length; i++) {
        if (arr[i] != null) {
            n++;
        }
    }
    Jugadores temp;
    for (int i = 0; i < n; i++) {
        for (int j = 1; j < (n - i); j++) {
            if (arr[j - 1].getMovimientosJugador() > arr[j].getMovimientosJugador() && arr[j].getMovimientosJugador() != 0 && arr[j - 1].getMovimientosJugador() != 0) { //Solo se realiza el
                temp = arr[j - 1];
                arr[j - 1] = arr[j];
                arr[j] = temp;
            }
        }

    }
}

    public static void serialize(String pathname, Object object) {
    // Serializar un objeto
    try {
        ObjectOutputStream objectOutputStream = new ObjectOutputStream(new FileOutputStream(pathname));
        objectOutputStream.writeObject(object);
        objectOutputStream.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

public static Object deserialize(String pathname) {
    // Leer un objeto serializado
    try {
        ObjectInputStream objectInputStream = new ObjectInputStream(new FileInputStream(pathname));
        Object data = objectInputStream.readObject();
        objectInputStream.close();
        return data;
    } catch (IOException | ClassNotFoundException e) {
        e.printStackTrace();
    }
    return null;
}
```

Clase en la que se serializa, deserializa y se usa ordenamiento.



Libraries: en las que se encuentran la librería por default.