# Ejercicios prácticos

|  |
| --- |
| **Asignatura**: Desarrollo de Interfaces |
| **Unidades** |

## Guía

* Se plantean diferentes ejercicios para realizarlos en Java.
* En cada ejercicio añadir el código resultante final en el cuadro correspondiente.
* Los ejercicios pueden tener más de una solución, busca la óptima, teniendo en cuenta el número de instrucciones, sentencias y estructuras, etc.

Entregables:

Los ejercicios se entregarán mediante un documento Word donde se inserte el código del proyecto e incluya fotos que describan las distintas vistas de la interfaz.

## Ejercicio 1: Reloj

Desarrolla una aplicación en Java que presente un reloj digital estéticamente atractivo en una vista que ocupe el 60% de la pantalla. La aplicación deberá incluir tres modos: cronómetro, alarma y cambio de hora.

Requisitos Principales:

1. **Interfaz de Reloj Digital Atractiva:** Diseña una interfaz de reloj digital que sea visualmente atractiva y ocupe el 60% de la vista en la pantalla.
2. **Modo Cronómetro:** Implementa una vista de cronómetro en la que se muestra el tiempo transcurrido en formato 0:0:00. Debe incluir botones de inicio, pausa y parada para el cronómetro.
3. **Modo Alarma:** Permite a los usuarios configurar una hora específica para que suene una alarma cuando se alcance dicha hora.
4. **Modo Cambio de Hora:** Ofrece la posibilidad de modificar la hora del reloj en la aplicación.

**Consideraciones Importantes:**

* Inicialmente, el reloj debe tomar la hora del sistema del dispositivo.
* La aplicación debe funcionar de manera intuitiva y eficiente para el usuario.

**Añadidos Opcionales:**

1. **Integración de Modos en una Sola Vista:** Opcionalmente, puedes implementar los tres modos en una sola vista de manera que estén bien integrados en el diseño. Puedes incluir transiciones o cambios en la organización de la interfaz según tu diseño.
2. **Cambio de Diseño (Modo Noche / Modo Día):** Opcionalmente, añade la funcionalidad de cambiar el diseño de la aplicación en función de la hora del reloj, ofreciendo un modo noche y un modo día.
3. **Personalización del Tema del Reloj:** Opcionalmente, permite a los usuarios cambiar el tema del reloj y personalizar su apariencia.

## Ejercicio 2: Juego adivinanza.

Desarrolla una aplicación que presente un menú de opciones con las siguientes funcionalidades: iniciar juego, histórico y ajustes. El juego consiste en que el ordenador elige un número aleatorio y el usuario debe adivinarlo en un número limitado de intentos, recibiendo pistas (mayor que, menor que) del juego. Los mejores intentos se guardarán en el histórico, y los ajustes permitirán personalizar el número de intentos y la dificultad del juego.

**Requisitos Principales:**

1. **Menú de Opciones:** Diseña un menú que incluya las siguientes opciones: "Iniciar Juego," "Histórico" y "Ajustes."
2. **Juego de Adivinanza de Números:** Implementa un juego en el que el ordenador elija un número aleatorio y el usuario tenga que adivinarlo en un número limitado de intentos. Proporciona pistas (mayor que, menor que) en función del número introducido por el usuario.
3. **Histórico de Mejores Intentos:** Almacena y muestra los mejores intentos de los usuarios para adivinar el número en sesiones anteriores.
4. **Ajustes Personalizables:** Permite a los usuarios cambiar parámetros como el número de intentos y la dificultad, que incluye la elección del rango de números (por ejemplo, del 0 al 10 o del 0 al 100).

**Consideraciones Importantes:**

* Diseña la interfaz de manera atractiva y amigable para fomentar la participación y la repetición del juego.
* Asegúrate de que el juego sea intuitivo y brinde una experiencia satisfactoria al usuario.

Ejercicio 3: Aplicación de IMC

Crea una aplicación que permita a los usuarios introducir datos como su edad, género, peso y altura.   
Basándose en estos datos, la aplicación calculará el Índice de Masa Corporal (IMC) y mostrará un gráfico de colores (verde, amarillo y rojo) que represente el valor obtenido en relación con el IMC ideal. Además, la aplicación debe permitir al usuario guardar la información y consultarla, incluyendo el nombre de la persona. También debe brindar la opción de editar y añadir nuevas lecturas a una etiqueta existente, así como visualizar un histórico de las mediciones.

**Requisitos Principales:**

1. **Entrada de Datos de Usuario:** Diseña una interfaz de usuario en la que los usuarios puedan introducir su edad, género, peso y altura de manera clara y fácil de entender.
2. **Cálculo del IMC:** Implementa la funcionalidad para calcular el IMC en función de los datos proporcionados por el usuario.
3. **Gráfico de Colores:** Muestra un gráfico de colores (verde, amarillo y rojo) que refleje el IMC calculado en comparación con el IMC ideal. Usa colores claros que den un aspecto saludable.
4. **Almacenamiento de Datos:** Permite a los usuarios guardar su información, incluyendo el nombre, en una etiqueta. Los datos deben poder consultarse y editarse en cualquier momento.
5. **Histórico de Mediciones:** Proporciona una función que permita al usuario ver el histórico de mediciones previas y añadir nuevas lecturas a una etiqueta existente.

**Consideraciones Importantes:**

* La presentación de la información debe ser clara y fácil de entender para fomentar el uso continuo de la aplicación.
* Los colores utilizados en el gráfico deben ser visualmente atractivos y transmitir un mensaje de salud.

**EJERCICIO 1**

**Estructura del Proyecto:**

**C:.**

**├───src**

**│ └───main**

**│ └───java**

**│ └───com**

**│ └───relojdigital**

**│ ├───back**

**│ └───interfaces**

**└───target**

**├───classes**

**│ └───com**

**│ └───relojdigital**

**│ ├───back**

**│ └───interfaces**

**└───test-classes**

**CÓDIGO:**

package com.relojdigital.back;

import java.io.File;

import java.util.Calendar;

import javax.sound.sampled.AudioInputStream;

import javax.sound.sampled.AudioSystem;

import javax.sound.sampled.*Clip*;

import javax.swing.JFileChooser;

import javax.swing.JOptionPane;

public class AlarmHandler {

    private Calendar alarmTime;

    public AlarmHandler() {

*this*.alarmTime = Calendar.getInstance();

*this*.alarmTime.set(Calendar.SECOND, 0);

    }

    public *void* setAlarmTime(*int* *hour*, *int* *minute*) {

*this*.alarmTime.set(Calendar.HOUR\_OF\_DAY, *hour*);

*this*.alarmTime.set(Calendar.MINUTE, *minute*);

    }

    public *void* checkAlarm() {

        Calendar currentTime = Calendar.getInstance();

        System.out.println("Checking Alarm - Current Time: " + currentTime.getTime());

        System.out.println("Checking Alarm - Alarm Time: " + *this*.alarmTime.getTime());

        if (currentTime.get(Calendar.HOUR\_OF\_DAY) == *this*.alarmTime.get(Calendar.HOUR\_OF\_DAY) &&

                currentTime.get(Calendar.MINUTE) == *this*.alarmTime.get(Calendar.MINUTE)) {

            System.out.println("Alarm Triggered!");

            triggerAlarm();

        }

    }

    private *void* triggerAlarm() {

        System.out.println("Alarm Triggered - Showing Message");

        JOptionPane.showMessageDialog(null, "Alarm Triggered!"); // Asegúrate de que este mensaje se muestre

        selectAndPlayAlarmSound();

    }

    // Damos al usuario la posibilidad de seleccionar un sonido de alarma

    private *void* selectAndPlayAlarmSound() {

        JFileChooser fileChooser = **new** JFileChooser();

        fileChooser.setDialogTitle("Selecciona el sonido de la alarma");

        // Para filtrar por archivos de sonido

        fileChooser.setFileFilter(**new** javax.swing.filechooser.FileFilter() {

            @Override

            public *boolean* accept(File *f*) {

                return *f*.isDirectory() || *f*.getName().toLowerCase().endsWith(".wav");

            }

            @Override

            public String getDescription() {

                return "Archivos de sonido (\*.wav)";

            }

        });

*int* result = fileChooser.showOpenDialog(null);

        if (result == JFileChooser.APPROVE\_OPTION) {

            File selectedFile = fileChooser.getSelectedFile();

            playAlarmSound(selectedFile);

        }

    }

    // Reproducimos el sonido de alarma seleccionado

    private *void* playAlarmSound(File *soundFile*) {

        try {

            AudioInputStream audioIn = AudioSystem.getAudioInputStream(*soundFile*);

*Clip* clip = AudioSystem.getClip();

            clip.open(audioIn);

            clip.start();

        } catch (Exception *e*) {

            e.printStackTrace();

        }

    }

}

package com.relojdigital.back;

import javax.swing.JLabel;

import javax.swing.SwingUtilities;

public class ChronometerHandler implements *Runnable* {

    private volatile *boolean* running, paused;

    private *long* elapsedTime;

    private JLabel timeLabel;

    public ChronometerHandler(JLabel *timeLabel*) {

        if (*timeLabel* == null) {

            throw **new** IllegalArgumentException("timeLabel cannot be null");

        }

*this*.timeLabel = *timeLabel*;

*this*.running = true;

*this*.paused = false;

*this*.elapsedTime = 0;

    }

    @Override

    public *void* run() {

        while (running) {

            if (!paused) {

                elapsedTime++;

                SwingUtilities.invokeLater(() *->* updateTimeLabel());

                try {

                    Thread.sleep(1000);

                } catch (InterruptedException *e*) {

                    e.printStackTrace();

                }

            }

        }

    }

    public *void* pause() {

        paused = true;

    }

    public *void* resume() {

        paused = false;

    }

    public *void* stop() {

        running = false;

    }

    public *boolean* isPaused() {

        return paused;

    }

    private *void* updateTimeLabel() {

        // Verificamos si timeLabel es null antes de intentar actualizarlo

        if (timeLabel != null) {

*long* hours = elapsedTime / 3600;

*long* minutes = (elapsedTime % 3600) / 60;

*long* seconds = elapsedTime % 60;

            String time = String.format("%02d:%02d:%02d", hours, minutes, seconds);

            SwingUtilities.invokeLater(() *->* timeLabel.setText(time));

        }

    }

}

package com.relojdigital.back;

import java.text.SimpleDateFormat;

import javax.swing.JLabel;

public class ClockHandler implements *Runnable* {

    private SimpleDateFormat timeFormat;

    private JLabel timeLabel;

    private AlarmHandler alarmHandler;

    public ClockHandler(JLabel *timeLabel*, AlarmHandler *alarmHandler*) {

*this*.timeLabel = *timeLabel*;

*this*.timeFormat = **new** SimpleDateFormat("hh:mm:ss a");

*this*.alarmHandler = *alarmHandler*;

    }

    @Override

    public *void* run() {

        while (true) {

            updateTime();

            checkAlarm();

            try {

                Thread.sleep(1000);

            } catch (InterruptedException *e*) {

                e.printStackTrace();

            }

        }

    }

    private *void* updateTime() {

        String time = timeFormat.format(**new** java.util.Date().getTime());

        timeLabel.setText(time);

    }

    private *void* checkAlarm() {

        // Verificamos si la alarma está activada

        if (alarmHandler != null) {

            alarmHandler.checkAlarm();

        }

    }

}

package com.relojdigital.back;

import java.io.File;

public class SoundSystem {

    public static File getSystemSound() {

        String os = System.getProperty("os.name").toLowerCase();

        if (os.contains("win")) {

            return **new** File("C:\\Windows\\Media\\ding.wav"); // Ruta al sonido predeterminado de Windows

        } else {

            // Aquí, simplemente devolvemos null para que no se reproduzca ningún sonido por

            // defecto en otros sistemas.

            return null;

        }

    }

}

package com.relojdigital.interfaces;

import javax.swing.JButton;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JOptionPane;

import javax.swing.JPanel;

import javax.swing.JTextField;

import com.relojdigital.back.AlarmHandler;

import java.awt.BorderLayout;

import java.awt.Color;

import java.awt.GradientPaint;

import java.awt.Graphics;

import java.awt.Graphics2D;

import java.awt.GridBagConstraints;

import java.awt.GridBagLayout;

import java.awt.Insets;

public class AlarmFrame extends JFrame {

    private AlarmHandler alarmHandler;

    private JTextField alarmHourTextField;

    private JTextField alarmMinuteTextField;

    public AlarmFrame(AlarmHandler *alarmHandler*) {

*this*.alarmHandler = *alarmHandler*;

        setTitle("Configurar Alarma");

        setSize(400, 200);

        setLocationRelativeTo(null);

        setDefaultCloseOperation(JFrame.DISPOSE\_ON\_CLOSE);

        // Panel para contenido con fondo gradiente

        JPanel panel = **new** JPanel() {

            @Override

            protected *void* paintComponent(Graphics *g*) {

*super*.paintComponent(*g*);

                Graphics2D g2d = (Graphics2D) *g*;

                GradientPaint gradient = **new** GradientPaint(0, 0, **new** Color(30, 30, 30), getWidth(), getHeight(),

**new** Color(10, 10, 10));

                g2d.setPaint(gradient);

                g2d.fillRect(0, 0, getWidth(), getHeight());

            }

        };

        panel.setLayout(**new** BorderLayout());

        // Uso GridBagLayout para más control sobre el diseño

        panel.setLayout(**new** GridBagLayout());

        GridBagConstraints gbc = **new** GridBagConstraints();

        gbc.gridx = 0;

        gbc.gridy = 0;

        gbc.insets = **new** Insets(10, 10, 10, 10); // Espacio entre componentes

        // Agregar campos de texto y etiquetas

        JLabel hourLabel = **new** JLabel("Hora:");

        panel.add(hourLabel, gbc);

        gbc.gridx++;

        alarmHourTextField = **new** JTextField(2);

        panel.add(alarmHourTextField, gbc);

        gbc.gridx++;

        JLabel minuteLabel = **new** JLabel("Minuto:");

        panel.add(minuteLabel, gbc);

        gbc.gridx++;

        alarmMinuteTextField = **new** JTextField(2);

        panel.add(alarmMinuteTextField, gbc);

        gbc.gridy++;

        gbc.gridx = 0;

        gbc.gridwidth = 4;

        JButton setAlarmButton = ButtonsStyler.customizeButton("Configurar Alarma", **new** Color(255, 165, 0), 16, 210,

                40);

        panel.add(setAlarmButton, gbc);

        setAlarmButton.addActionListener(*e* *->* configureAlarm());

        // Agregar el panel al frame

        add(panel);

        // Hacer visible el frame

        setVisible(true);

    }

    private *void* configureAlarm() {

        // Configurar la alarma

        try {

*int* hour = Integer.parseInt(alarmHourTextField.getText());

*int* minute = Integer.parseInt(alarmMinuteTextField.getText());

            if (hour >= 0 && hour < 24 && minute >= 0 && minute < 60) {

                alarmHandler.setAlarmTime(hour, minute);

                JOptionPane.showMessageDialog(null, "Alarma configurada para " + hour + ":" + minute, "Alarma",

                        JOptionPane.INFORMATION\_MESSAGE);

                dispose(); // Cerrar la ventana después de configurar la alarma

            } else {

                JOptionPane.showMessageDialog(null, "Ingrese una hora y minutos válidos", "Error",

                        JOptionPane.ERROR\_MESSAGE);

            }

        } catch (NumberFormatException *ex*) {

            JOptionPane.showMessageDialog(null, "Ingrese valores numéricos para hora y minutos", "Error",

                    JOptionPane.ERROR\_MESSAGE);

        }

    }

}

package com.relojdigital.interfaces;

import java.awt.Color;

import java.awt.Dimension;

import java.awt.Font;

import javax.swing.JButton;

public class ButtonsStyler {

    public static JButton customizeButton(String *text*, Color *backgroundColor*, *int* *fontSize*, *int* *buttonWidth*,

*int* *buttonHeight*) {

        JButton button = **new** JButton(*text*);

        button.setBackground(*backgroundColor*);

        button.setForeground(Color.BLACK); // Texto en negro

        button.setFont(**new** Font("Arial", Font.BOLD, *fontSize*));

        // Establecer el tamaño preferido del botón

        button.setPreferredSize(**new** Dimension(*buttonWidth*, *buttonHeight*));

        return button;

    }

}

package com.relojdigital.interfaces;

import java.awt.BorderLayout;

import java.awt.Color;

import java.awt.Font;

import java.awt.GradientPaint;

import java.awt.Graphics;

import java.awt.Graphics2D;

import javax.swing.BorderFactory;

import javax.swing.JButton;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JPanel;

import com.relojdigital.back.ChronometerHandler;

public class ChronometerFrame extends JFrame {

    private JLabel timeLabel;

    private ChronometerHandler chronometerHandler;

    public ChronometerFrame(MyFrame *parentFrame*) {

        setTitle("Cronómetro");

        setSize(*parentFrame*.getWidth(), *parentFrame*.getHeight());

        setLocationRelativeTo(*parentFrame*);

        setDefaultCloseOperation(JFrame.DISPOSE\_ON\_CLOSE); // Cerrar solo esta ventana al salir

        // Panel para contenido con fondo gradiente

        JPanel panel = **new** JPanel() {

            @Override

            protected *void* paintComponent(Graphics *g*) {

*super*.paintComponent(*g*);

                Graphics2D g2d = (Graphics2D) *g*;

                GradientPaint gradient = **new** GradientPaint(0, 0, **new** Color(30, 30, 30), getWidth(), getHeight(),

**new** Color(10, 10, 10));

                g2d.setPaint(gradient);

                g2d.fillRect(0, 0, getWidth(), getHeight());

            }

        };

        panel.setLayout(**new** BorderLayout());

        // Agregar el timeLabel al panel

        timeLabel = **new** JLabel("00:00:00");

        timeLabel.setFont(**new** Font("Arial", Font.BOLD, 100));

        timeLabel.setForeground(Color.WHITE);

        timeLabel.setHorizontalAlignment(JLabel.CENTER);

        // Ubica la etiqueta en el centro del panel

        panel.add(timeLabel, BorderLayout.CENTER);

        // Panel para botones del cronómetro

        JPanel buttonPanel = **new** JPanel();

        // Colores y estilo del panel de botones

        buttonPanel.setBackground(**new** Color(20, 20, 20));

        buttonPanel.setBorder(BorderFactory.createEmptyBorder(50, 50, 50, 50));

        // Botón para pausar/reanudar el cronómetro

        JButton pauseChronometerButton = ButtonsStyler.customizeButton("Pausar / Reanudar", **new** Color(50, 205, 50), 16,

                210, 40);

        buttonPanel.add(pauseChronometerButton);

        pauseChronometerButton.addActionListener(*e* *->* {

            if (chronometerHandler.isPaused()) {

                chronometerHandler.resume();

            } else {

                chronometerHandler.pause();

            }

        });

        // Botón para detener el cronómetro

        JButton stopChronometerButton = ButtonsStyler.customizeButton("Detener", **new** Color(50, 205, 50), 16,

                210, 40);

        buttonPanel.add(stopChronometerButton);

        stopChronometerButton.addActionListener(*e* *->* stopChronometer(*parentFrame*));

        // Agregar el panel de botones al sur del panel de contenido

        panel.add(buttonPanel, BorderLayout.SOUTH);

        // Agregar el panel de contenido al frame

        add(panel);

        // Iniciar el cronómetro

        startChronometer();

    }

    private *void* startChronometer() {

        chronometerHandler = **new** ChronometerHandler(timeLabel);

**new** Thread(chronometerHandler).start();

    }

    private *void* stopChronometer(MyFrame *parentFrame*) {

        // Detener el cronómetro y cerrar la ventana del cronómetro

        chronometerHandler.stop();

        dispose();

        // Mostrar nuevamente el reloj en la ventana principal

*parentFrame*.setVisible(true);

    }

}

package com.relojdigital.interfaces;

import java.awt.BorderLayout;

import java.awt.Color;

import java.awt.Font;

import java.awt.GradientPaint;

import java.awt.Graphics;

import java.awt.Graphics2D;

import java.awt.GridBagConstraints;

import java.awt.GridBagLayout;

import java.awt.Insets;

import java.awt.Toolkit;

import javax.swing.BorderFactory;

import javax.swing.JButton;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JPanel;

import com.relojdigital.back.AlarmHandler;

import com.relojdigital.back.ClockHandler;

public class MyFrame extends JFrame {

    private JLabel timeLabel;

    private AlarmHandler alarmHandler;

    public MyFrame() {

        setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

        setTitle("Reloj Digital");

        // Obtenemos el tamaño de la pantalla

*java*.*awt*.Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();

        // Calcular el ancho de la ventana para que ocupe el 60% de la pantalla

*int* windowWidth = (*int*) (screenSize.width \* 0.6);

        setSize(windowWidth, windowWidth); // Tamaño de la ventana

        setResizable(true); // No se puede redimensionar

        setLocationRelativeTo(null); // Centrar la ventana

        // Panel para contenido con fondo gradiente

        JPanel panel = **new** JPanel() {

            @Override

            protected *void* paintComponent(Graphics *g*) {

*super*.paintComponent(*g*);

                Graphics2D g2d = (Graphics2D) *g*;

                GradientPaint gradient = **new** GradientPaint(0, 0, **new** Color(30, 30, 30), getWidth(), getHeight(),

**new** Color(10, 10, 10));

                g2d.setPaint(gradient);

                g2d.fillRect(0, 0, getWidth(), getHeight());

            }

        };

        panel.setLayout(**new** BorderLayout());

        // Agregar el timeLabel al panel

        timeLabel = **new** JLabel("00:00:00");

        timeLabel.setFont(**new** Font("Arial", Font.BOLD, 100));

        timeLabel.setForeground(Color.WHITE);

        timeLabel.setHorizontalAlignment(JLabel.CENTER);

        // Ubica la etiqueta en el centro del panel

        panel.add(timeLabel, BorderLayout.CENTER);

        // Hacer visible la ventana después de iniciar el reloj

        setVisible(true);

        // Panel para botones del cronómetro

        JPanel buttonPanel = **new** JPanel();

        JButton startChronometerButton = ButtonsStyler.customizeButton("Iniciar Cronómetro", **new** Color(50, 205, 50), 16,

                210, 40);

        buttonPanel.add(startChronometerButton);

        startChronometerButton.addActionListener(*e* *->* startChronometer());

        // Colores y estilo del panel de botones

        buttonPanel.setBackground(**new** Color(20, 20, 20));

        buttonPanel.setBorder(BorderFactory.createEmptyBorder(50, 50, 50, 50));

        // Uso GridBagLayout para más control sobre el diseño

        buttonPanel.setLayout(**new** GridBagLayout());

        GridBagConstraints gbc = **new** GridBagConstraints();

        gbc.gridx = 0;

        gbc.gridy = 0;

        gbc.insets = **new** Insets(10, 10, 10, 10); // Espacio entre botones

        // Agregar el panel de botones al sur del panel de contenido

        panel.add(buttonPanel, BorderLayout.SOUTH);

        JButton setAlarmButton = ButtonsStyler.customizeButton("Configurar Alarma", **new** Color(255, 165, 0), 16, 210,

                40);

        buttonPanel.add(setAlarmButton);

        setAlarmButton.addActionListener(*e* *->* **new** AlarmFrame(alarmHandler));

        // Agregar el panel de contenido al frame

        add(panel);

        // Hacer visible la ventana después de iniciar el reloj

*this*.setVisible(true);

        // Iniciar el reloj digital

        startClock();

        alarmHandler = **new** AlarmHandler();

    }

    private *void* startClock() {

        // Crear una instancia del manejador del reloj y ejecutarlo en un hilo separado

        ClockHandler clockHandler = **new** ClockHandler(timeLabel, alarmHandler);

**new** Thread(clockHandler).start();

    }

    private *void* startChronometer() {

        // Crear una instancia de ChronometerFrame al iniciar el cronómetro

        ChronometerFrame chronometerFrame = **new** ChronometerFrame(*this*);

        chronometerFrame.setVisible(true);

    }

}

package com.relojdigital;

import com.relojdigital.interfaces.MyFrame;

public class App {

    public static *void* main(String[] *args*) {

        // Punto de entrada de la aplicación

        launchApp();

    }

    private static *void* launchApp() {

        // Inicializa y muestra la ventana principal

**new** MyFrame();

    }

}

**VISTAS:**

**Captura de pantalla de computadora

Descripción generada automáticamente**

Captura de pantalla de computadora

Descripción generada automáticamente

Interfaz de usuario gráfica, Sitio web

Descripción generada automáticamente

**EJERCICIO 2:**

**Estructura del Proyecto:**

**C:.**

**├───.vscode**

**├───src**

**│ └───main**

**│ └───java**

**│ └───com**

**│ └───guessgame**

**│ ├───bestgamerecord**

**│ ├───gui**

**│ └───logicgame**

**└───target**

**├───classes**

**│ └───com**

**│ └───guessgame**

**│ ├───bestgamerecord**

**│ ├───gui**

**│ └───logicgame**

**├───maven-archiver**

**├───maven-status**

**│ └───maven-compiler-plugin**

**│ ├───compile**

**│ │ └───default-compile**

**│ └───testCompile**

**│ └───default-testCompile**

**└───test-classes**

**CÓDIGO:**

package com.guessgame.bestgamerecord;

import java.util.ArrayList;

import java.util.Date;

import java.util.*List*;

public class BestGameRecord {

    private String difficulty;

    private *int* attempts;

    private Date date; // Usamos java.util.Date para guardar la fecha de la partida

    private static *List*<BestGameRecord> bestGames = **new** ArrayList<>();

    private static final *int* MAX\_BEST\_GAMES\_TO\_KEEP = 5;

    public BestGameRecord(String *difficulty*, *int* *attempts*, Date *date*) {

*this*.difficulty = *difficulty*;

*this*.attempts = *attempts*;

*this*.date = *date*;

    }

    public String getDifficulty() {

        return difficulty;

    }

    public *void* setDifficulty(String *difficulty*) {

*this*.difficulty = *difficulty*;

    }

    public *int* getAttempts() {

        return attempts;

    }

    public *void* setAttempts(*int* *attempts*) {

*this*.attempts = *attempts*;

    }

    public Date getDate() {

        return date;

    }

    public *void* setDate(Date *date*) {

*this*.date = *date*;

    }

    public static *int* getMaxBestGamesToKeep() {

        return MAX\_BEST\_GAMES\_TO\_KEEP;

    }

    @Override

    public String toString() {

        return "Fecha: " + date + ". Dificultad: " + difficulty + ", intentos: " + attempts + ".";

    }

    public static *List*<BestGameRecord> getBestGames() {

        return bestGames;

    }

}

package com.guessgame.gui;

import javax.swing.*\**;

import com.formdev.flatlaf.FlatDarkLaf;

import com.guessgame.bestgamerecord.BestGameRecord;

import java.awt.*\**;

import java.awt.event.ActionEvent;

import java.awt.event.*ActionListener*;

import java.util.*List*;

/\*\*

 \* La clase DifficultyMenu representa un componente de interfaz gráfica de

 \* usuario que permite al usuario seleccionar el nivel de dificultad para un

 \* juego.

 \* Extiende la clase JPanel y proporciona métodos para obtener la dificultad

 \* seleccionada.

 \*/

public class DifficultyMenu extends JPanel {

    private JFrame frame;

    private JComboBox<String> difficultyComboBox;

    private JTextArea historyTextArea;

    public DifficultyMenu(*Runnable* *onSelectDifficulty*) {

        // Configuramos FlatLaf para utilizar un tema oscuro

        try {

            UIManager.setLookAndFeel((**new** FlatDarkLaf()));

        } catch (UnsupportedLookAndFeelException *e*) {

            e.printStackTrace();

        }

        // Personaliza los colores

        Color backgroundColor = **new** Color(255, 231, 186); // Amarillo claro

        Color textColor = **new** Color(34, 49, 63); // Azul oscuro

        Color buttonColor = **new** Color(242, 120, 75); // Naranja claro

        // Configuración de la ventana principal

        frame = **new** JFrame("¡Adivina el Número!");

        frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

        frame.setLayout(**new** BorderLayout());

        frame.getContentPane().setBackground(backgroundColor);

        // Configuración del panel principal

        JPanel mainPanel = **new** JPanel();

        mainPanel.setLayout(**new** BoxLayout(mainPanel, BoxLayout.Y\_AXIS));

        mainPanel.setBorder(BorderFactory.createEmptyBorder(20, 20, 20, 20)); // Borde para espacio

        mainPanel.setBackground(backgroundColor);

        // Configuración del título del juego

        JLabel tituloJuego = **new** JLabel("¡BIENVENIDO A \"ADIVINA EL NÚMERO!\"");

        tituloJuego.setFont(**new** Font("Arial", Font.BOLD, 18));

        tituloJuego.setAlignmentX(Component.CENTER\_ALIGNMENT); // Título centrado horizontalmente

        tituloJuego.setForeground(textColor);

        // Configuración de la etiqueta de selección de dificultad

        JLabel label = **new** JLabel("Selecciona la dificultad:");

        label.setAlignmentX(Component.CENTER\_ALIGNMENT);

        label.setBorder(BorderFactory.createEmptyBorder(10, 0, 10, 0));

        label.setForeground(textColor);

        label.setFont(**new** Font("Arial", Font.BOLD, 14));

        try {

            // Configuración del ComboBox de dificultad

            difficultyComboBox = **new** JComboBox<>(

**new** String[] { "Fácil (8 vidas)", "Intermedio (5 vidas)", "Difícil (3 vidas)" });

            difficultyComboBox.setFont(**new** Font("Arial", Font.BOLD, 14));

            difficultyComboBox.setMaximumSize((**new** Dimension(200, 30)));

            difficultyComboBox.setAlignmentX(Component.CENTER\_ALIGNMENT); // ComboBox centrado horizontalmente

            difficultyComboBox.setBackground(buttonColor);

            difficultyComboBox.setForeground(textColor);

            // Crea un renderizador personalizado para centrar el texto

            DefaultListCellRenderer renderer = **new** DefaultListCellRenderer();

            renderer.setHorizontalAlignment(DefaultListCellRenderer.CENTER);

            difficultyComboBox.setRenderer(renderer);

            difficultyComboBox.setAlignmentX(Component.CENTER\_ALIGNMENT); // ComboBox centrado horizontalmente

        } catch (Exception *e*) {

            e.printStackTrace();

        }

        // Configuración del botón de inicio

        JButton startButton = **new** JButton("Comenzar");

        startButton.setAlignmentX(Component.CENTER\_ALIGNMENT); // Botón centrado horizontalmente

        startButton.setBackground(buttonColor);

        startButton.setForeground(textColor);

        // Acción al hacer clic en el botón de inicio

        startButton.addActionListener(**new** ActionListener() {

            @Override

            public *void* actionPerformed(ActionEvent *e*) {

*onSelectDifficulty*.run();

                frame.dispose();

            }

        });

        // Configuración del botón de historial

        JButton historyButton = **new** JButton("Mejores Partidas");

        historyButton.setAlignmentX(Component.LEFT\_ALIGNMENT); // Botón alineado a la izquierda

        // Agregar el botón de historial al panel principal

        historyButton.addActionListener(**new** ActionListener() {

            @Override

            public *void* actionPerformed(ActionEvent *e*) {

                updateHistory(BestGameRecord.getBestGames());

            }

        });

        historyTextArea = **new** JTextArea();

        historyTextArea.setEditable(false);

        historyTextArea.setBackground(backgroundColor);

        historyTextArea.setFont(**new** Font("Arial", Font.PLAIN, 12));

        JPanel historyPanel = **new** JPanel();

        historyPanel.setLayout(**new** FlowLayout(FlowLayout.LEFT, 0, 0));

        historyPanel.add(historyButton);

        historyPanel.add(historyTextArea);

        historyPanel.setBackground(backgroundColor);

        // Agregar componentes al panel principal

        mainPanel.add(tituloJuego);

        mainPanel.add(Box.createVerticalStrut(10));

        mainPanel.add(label);

        mainPanel.add(difficultyComboBox);

        mainPanel.add(Box.createVerticalStrut(10));

        mainPanel.add(startButton);

        mainPanel.add(Box.createVerticalStrut(30));

        mainPanel.add(historyPanel);

        // Agregar panel principal a la ventana

        frame.getContentPane().add(BorderLayout.CENTER, mainPanel);

        frame.setSize(GuiConstants.FRAME\_WIDTH, GuiConstants.FRAME\_HEIGHT); // Aumentamos el tamaño de la ventana

        frame.setLocationRelativeTo(null); // Centrar la ventana en la pantalla

        frame.setVisible(true);

    }

    /\*\*

     \* Recupera la dificultad seleccionada del difficultyComboBox.

     \*

     \* @return La dificultad seleccionada como una cadena de texto.

     \*/

    public String getSelectedDifficulty() {

        return (String) difficultyComboBox.getSelectedItem();

    }

    public *void* updateHistory(*List*<BestGameRecord> *bestGames*) {

        historyTextArea.setText("Historial de Mejores Partidas:\n");

        for (BestGameRecord bestGame : *bestGames*) {

            historyTextArea.append("Fecha: " + bestGame.getDate() +

                    ", Dificultad: " + bestGame.getDifficulty() +

                    ", Intentos: " + bestGame.getAttempts() + "\n");

        }

    }

}

package com.guessgame.gui;

import java.awt.BorderLayout;

import java.awt.Color;

import java.awt.FlowLayout;

import java.awt.Font;

import java.awt.GridBagConstraints;

import java.awt.GridBagLayout;

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.JOptionPane;

import javax.swing.JPanel;

import javax.swing.JTextField;

import javax.swing.*SwingConstants*;

import com.guessgame.bestgamerecord.BestGameRecord;

import com.guessgame.logicgame.GuessNumberGame;

/\*\*

 \* La clase GameFrame representa la ventana principal del juego de adivinar el

 \* número.

 \* Contiene métodos para crear y mostrar el menú de dificultad, iniciar el juego

 \* con una dificultad seleccionada,

 \* crear y mostrar la interfaz gráfica del juego, y manejar los eventos del

 \* botón de adivinar y el diálogo de fin de juego.

 \*/

public class GameFrame {

    private JFrame frame;

    private JTextField textField;

    private JLabel resultLabel;

    private GuessNumberGame game;

    private DifficultyMenu difficultyMenu;

    private JLabel livesLabel;

    /\*\*

     \* Crea y muestra el menú de dificultad.

     \*/

    public *void* createAndShowDifficultyMenu() {

        difficultyMenu = **new** DifficultyMenu(*this*::startGameWithDifficulty);

    }

    /\*\*

     \* Inicia el juego con la dificultad seleccionada por el usuario.

     \*/

    public *void* startGameWithDifficulty() {

        String selectedDifficulty = difficultyMenu.getSelectedDifficulty();

        switch (selectedDifficulty) {

            case "Fácil (8 vidas)":

                game = **new** GuessNumberGame(8, selectedDifficulty);

                break;

            case "Intermedio (5 vidas)":

                game = **new** GuessNumberGame(5, selectedDifficulty);

                break;

            case "Difícil (3 vidas)":

                game = **new** GuessNumberGame(3, selectedDifficulty);

                break;

        }

        // Inicia el juego

        createAndShowGameGUI();

    }

    /\*\*

     \* Esta función se encarga de crear y configurar los componentes de la interfaz

     \* gráfica,

     \* como el marco principal, los paneles, las etiquetas, los botones y los campos

     \* de texto.

     \* También establece los estilos y colores de los elementos de la interfaz.

     \* Finalmente, muestra la ventana en la pantalla.

     \*/

    public *void* createAndShowGameGUI() {

        // Personaliza los colores

        Color backgroundColor = **new** Color(255, 231, 186); // Amarillo claro

        Color textColor = **new** Color(34, 49, 63); // Azul oscuro

        Color buttonColor = **new** Color(242, 120, 75); // Naranja claro

        // Crea y configura el marco principal de la ventana

        frame = **new** JFrame("Adivina el número");

        frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

        frame.getContentPane().setLayout(**new** BorderLayout());

        frame.getContentPane().setBackground(backgroundColor); // Establece el color de fondo del marco como amarillo

                                                               // claro

        // Etiqueta de instrucción

        JLabel instructionLabel = **new** JLabel("Adivina el número entre 1 y 100.");

        instructionLabel.setHorizontalAlignment(*SwingConstants*.CENTER); // Centra el texto horizontalmente

        instructionLabel.setForeground(textColor); // Establece el color del texto como azul oscuro

        // Etiqueta de vidas restantes

        livesLabel = **new** JLabel("Vidas restantes: " + game.getMaxAttempts()); // Etiqueta para mostrar las vidas

        livesLabel.setForeground(textColor); // Establece el color del texto como azul oscuro

        livesLabel.setHorizontalAlignment(*SwingConstants*.CENTER); // Centra el texto horizontalmente

        // Etiqueta de resultado. Se muestra cuando el usuario adivina el número.

        resultLabel = **new** JLabel(""); // Etiqueta para mostrar el resultado del juego restantes

        // Crea el panel principal y lo configura

        JPanel mainPanel = **new** JPanel(**new** GridBagLayout());

        mainPanel.setBackground(backgroundColor); // Establece el color de fondo del panel

        // Configura las restricciones para la etiqueta de instrucción

        GridBagConstraints gbc = **new** GridBagConstraints();

        gbc.gridx = 0;

        gbc.gridy = 0;

        gbc.gridwidth = 1;

        gbc.anchor = GridBagConstraints.CENTER;

        mainPanel.add(instructionLabel, gbc);

        // Configura las restricciones para la etiqueta de vidas restantes

        gbc.gridy = 1;

        mainPanel.add(livesLabel, gbc);

        // Configura las restricciones para la etiqueta de resultado

        gbc.gridy = 2;

        mainPanel.add(resultLabel, gbc);

        // Fuentes y estilos

        Font labelFont = **new** Font("Arial", Font.BOLD, 16);

        instructionLabel.setFont(labelFont);

        resultLabel.setFont(labelFont);

        livesLabel.setFont(labelFont);

        // Crea el campo de texto para ingresar el número

        textField = **new** JTextField(10); // Campo de texto para ingresar el número

        textField.setBackground(Color.white); // Establece el color de fondo del campo de texto como blanco

        textField.setForeground(textColor); // Establece el color del texto del campo de texto como azul oscuro

        textField.setFont(labelFont);

        textField.setHorizontalAlignment(JTextField.CENTER); // Centra el texto horizontalmente

        frame.getContentPane().add(mainPanel, BorderLayout.CENTER); // Agrega el panel principal al marco

        // Crea un panel de entrada

        JPanel inputPanel = **new** JPanel();

        JButton guessButton = **new** JButton("Adivinar");

        guessButton.addActionListener(**new** GuessButtonListener()); // Evento del usuario para comprobar el número

                                                                  // introducido

        guessButton.setBackground(buttonColor); // Establece el color de fondo del botón como naranja claro

        guessButton.setForeground(textColor); // Establece el color del texto del botón como blanco

        guessButton.setFont(labelFont);

        JPanel menuButtonPanel = **new** JPanel();

        menuButtonPanel.setLayout(**new** FlowLayout(FlowLayout.RIGHT));

        JButton menuButton = **new** JButton("Volver al menú");

        menuButton.setBackground(buttonColor); // Establece el color de fondo del botón como naranja claro

        menuButton.setForeground(textColor); // Establece el color del texto del botón como blanco

        menuButton.addActionListener(**new** MenuButtonListener()); // Evento del usuario para volver al menú

        menuButton.setFont(labelFont.deriveFont(Font.BOLD, 12));

        menuButtonPanel.add(menuButton);

        // Configura el diseño y agrega los componentes al panel de entrada

        inputPanel.setLayout(**new** FlowLayout());

        inputPanel.add(textField);

        inputPanel.add(guessButton);

        // Agrega los paneles al marco principal

        frame.getContentPane().add(mainPanel, BorderLayout.CENTER);

        frame.getContentPane().add(inputPanel, BorderLayout.SOUTH);

        frame.getContentPane().add(menuButtonPanel, BorderLayout.NORTH);

        frame.setSize(GuiConstants.FRAME\_WIDTH, GuiConstants.FRAME\_HEIGHT); // Aumentamos el tamaño de la ventana

        frame.setLocationRelativeTo(null); // Centrar la ventana en la pantalla

        frame.setVisible(true);

    }

    // Nuevo ActionListener para el botón de menú

    private class MenuButtonListener implements *ActionListener* {

        @Override

        public *void* actionPerformed(ActionEvent *e*) {

            // Volver al menú

            createAndShowDifficultyMenu();

            frame.dispose(); // Cierra la ventana actual

        }

    }

    // Escucha de eventos para el botón de adivinar

    private class GuessButtonListener implements *ActionListener* {

        @Override

        public *void* actionPerformed(ActionEvent *e*) {

            try {

*int* userGuess = Integer.parseInt(textField.getText()); // Obtiene el número ingresado por el usuario

                game.incrementAttempts(); // Incrementa el contador de intentos del juego

                if (game.checkGuess(userGuess)) { // Verifica si el número ingresado es correcto

                    resultLabel.setText("¡Correcto! Has adivinado el número en " + game.getAttempts() + " intentos.");

                    textField.setEditable(false); // Deshabilita la edición del campo de texto

                } else {

                    resultLabel.setText("Intenta con un número "

                            + (userGuess < game.getNumberToGuess() ? "más grande." : "más pequeño."));

                    livesLabel.setText("Vidas restantes: " + (game.getMaxAttempts() - game.getAttempts()));

                    textField.setText(""); // Limpia el campo de texto

                }

                if (game.isGameOver()) { // Verifica si el juego ha terminado

                    textField.setEditable(false); // Deshabilita la edición del campo de texto

                    showGameOverDialog(); // Muestra el diálogo de fin de juego

                    difficultyMenu.updateHistory(BestGameRecord.getBestGames()); // Actualiza el historial de partidas

                }

            } catch (NumberFormatException *ex*) {

                resultLabel.setText("Por favor, ingresa un número válido."); // Muestra un mensaje de error si el número

                                                                             // ingresado no es válido

            }

        }

    }

    // Muestra un diálogo de fin de juego con dos opciones: "Volver a jugar" y

    // "Salir"

    private *void* showGameOverDialog() {

        Object[] options = { "Volver a jugar", "Salir" };

        // Muestra el diálogo de opción con el mensaje de juego terminado y el número a

        // adivinar

*int* choice = JOptionPane.showOptionDialog(

                frame,

                "Has perdido. El número era " + game.getNumberToGuess(),

                "¡Juego terminado!",

                JOptionPane.DEFAULT\_OPTION,

                JOptionPane.INFORMATION\_MESSAGE,

                null,

                options,

                options[0]);

        if (choice == 0) {

            createAndShowDifficultyMenu();

            // Volver a jugar

            frame.dispose(); // Cerrar la ventana actual

            // Volver a la pantalla de selección de dificultad

            difficultyMenu.setVisible(true);

        } else if (choice == 1) {

            // Salir de la aplicación

            System.exit(0);

        }

    }

}

package com.guessgame.gui;

public abstract class GuiConstants {

    public static final *int* FRAME\_WIDTH = 450;

    public static final *int* FRAME\_HEIGHT = 250;

}

package com.guessgame.logicgame;

import java.util.ArrayList;

import java.util.*Comparator*;

import java.util.Date;

import java.util.*List*;

import com.guessgame.bestgamerecord.BestGameRecord;

public class GuessNumberGame {

    private *int* numberToGuess;

    private *int* attempts;

    private *int* maxAttempts;

    private String difficulty;

    private *List*<BestGameRecord> bestGamesHistory = **new** ArrayList<>();

    public GuessNumberGame(*int* *maxAttempts*, String *difficulty*) {

*this*.numberToGuess = generateRandomNumber();

*this*.attempts = 0;

*this*.maxAttempts = *maxAttempts*;

*this*.difficulty = *difficulty*;

*this*.bestGamesHistory = **new** ArrayList<>();

    }

    private *int* generateRandomNumber() {

        return (*int*) (Math.random() \* 100) + 1;

    }

    public *int* getNumberToGuess() {

        return numberToGuess;

    }

    public *void* incrementAttempts() {

        attempts++;

    }

    public *int* getAttempts() {

        return attempts;

    }

    public *void* setAttempts(*int* *attempts*) {

*this*.attempts = *attempts*;

    }

    public *void* setMaxAttempts(*int* *maxAttempts*) {

*this*.maxAttempts = *maxAttempts*;

    }

    public *void* setNumberToGuess(*int* *numberToGuess*) {

*this*.numberToGuess = *numberToGuess*;

    }

    public *int* getMaxAttempts() {

        return maxAttempts;

    }

    public *boolean* checkGuess(*int* *userGuess*) {

        return *userGuess* == numberToGuess;

    }

    public *boolean* isGameOver() {

        return attempts >= maxAttempts;

    }

    public *void* gameCompletedSuccessfully() {

*int* maxBestGamesToKeep = BestGameRecord.getMaxBestGamesToKeep();

        if (bestGamesHistory.size() < maxBestGamesToKeep) {

            // Agrega el registro a la lista de mejores partidas si no se alcanzó el límite

            BestGameRecord record = **new** BestGameRecord(difficulty, attempts, **new** Date());

            bestGamesHistory.add(record);

        } else {

            // Encuentra la partida con el mayor número de intentos y compárala con la

            // partida actual

            BestGameRecord maxAttemptsRecord = bestGamesHistory.stream()

                    .max(*Comparator*.comparingInt(BestGameRecord::getAttempts))

                    .orElse(null);

            if (maxAttemptsRecord != null && attempts < maxAttemptsRecord.getAttempts()) {

                // Reemplaza la partida con el mayor número de intentos con la actual

                bestGamesHistory.remove(maxAttemptsRecord);

                BestGameRecord record = **new** BestGameRecord(difficulty, attempts, **new** Date());

                bestGamesHistory.add(record);

            }

        }

    }

}

package com.guessgame;

import com.guessgame.gui.GameFrame;

public class App {

    public static *void* main(String[] *args*) {

        javax.swing.SwingUtilities.invokeLater(() *->* {

            GameFrame frame = **new** GameFrame();

            frame.createAndShowDifficultyMenu();

        });

    }

}

**VISTAS:**

**Interfaz de usuario gráfica

Descripción generada automáticamente**

**Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente**

**Interfaz de usuario gráfica

Descripción generada automáticamente**

**Interfaz de usuario gráfica

Descripción generada automáticamente**

**Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente**

**EJERCICIO 3**

**Estructura del Proyecto:**

**C:.**

**├───.vscode**

**├───src**

**│ ├───main**

**│ │ └───java**

**│ │ └───com**

**│ │ └───imcapp**

**│ │ ├───gui**

**│ │ │ ├───guiprincipal**

**│ │ │ └───historialgui**

**│ │ └───logica**

**│ └───test**

**│ └───java**

**│ └───com**

**│ └───imcapp**

**└───target**

**├───classes**

**│ └───com**

**│ └───imcapp**

**│ ├───gui**

**│ │ ├───guiprincipal**

**│ │ └───historialgui**

**│ └───logica**

**└───test-classes**

**└───com**

**└───imcapp**

**CÓDIGO:**

package com.imcapp.gui.guiprincipal;

import java.awt.Color;

import java.awt.Dimension;

import java.util.List;

import javax.swing.JButton;

import javax.swing.JOptionPane;

import javax.swing.JPanel;

import org.jfree.chart.ChartFactory;

import org.jfree.chart.ChartPanel;

import org.jfree.chart.JFreeChart;

import org.jfree.data.category.CategoryDataset;

import org.jfree.data.category.DefaultCategoryDataset;

import org.jfree.chart.plot.CategoryPlot;

import org.jfree.chart.renderer.category.LineAndShapeRenderer;

import com.imcapp.logica.MedicionIMC;

public class GraficoComparativo extends *JPanel* {

    private static final *String* IMC = "IMC";

    private *GUI* gui;

    private *CategoryDataset* dataset;

    private *JFreeChart* chart;

    private *ChartPanel* chartPanel;

    public GraficoComparativo(*GUI* *gui*) {

*this*.gui = gui;

*this*.dataset = createDataset();

*this*.chart = ChartFactory.createBarChart(

                "Comparación IMC",

                "Usuario",

                IMC,

                dataset);

*this*.chart.setBackgroundPaint(Color.white);

*this*.chartPanel = **new** ChartPanel(chart);

*this*.chartPanel.setPreferredSize(**new** Dimension(800, 400));

        add(chartPanel);

*JButton* closeButton = **new** JButton("Cerrar");

        closeButton.addActionListener(e *->* setVisible(false));

        add(closeButton);

    }

    private *CategoryDataset* createDataset() {

*DefaultCategoryDataset* dataset = **new** DefaultCategoryDataset();

*double* imcUsuarioActual = obtenerIMCUsuarioActual();

        dataset.addValue(imcUsuarioActual, "Usuario Actual", IMC);

        return dataset;

    }

    private *double* obtenerIMCUsuarioActual() {

*double* peso;

*double* altura;

        try {

            peso = gui.getTxtPesoValue();

            altura = gui.getTxtAlturaValue();

        } catch (*NumberFormatException* *e*) {

            JOptionPane.showMessageDialog(*this*, "Por favor, introduce un número válido.", "Error",

                    JOptionPane.ERROR\_MESSAGE);

            return 0.0;

        }

        return calcularIMC(peso, altura);

    }

    private *double* calcularIMC(*double* *peso*, *double* *altura*) {

        return peso / (altura \* altura);

    }

    public *double* calcularMediaGlobal(*List*<MedicionIMC> *historial*) {

        return historial.stream()

                .mapToDouble(MedicionIMC::getIMC)

                .average()

                .orElse(0.0);

    }

    public *void* representacionDatos(*List*<MedicionIMC> *historial*) {

*DefaultCategoryDataset* newDataset = **new** DefaultCategoryDataset();

*double* imcUsuarioActual = obtenerIMCUsuarioActual();

        newDataset.addValue(imcUsuarioActual, "Tu IMC", IMC);

*double* mediaGlobal = calcularMediaGlobal(historial);

        newDataset.addValue(mediaGlobal, "Media Global", IMC);

        for (*int* i = 0; i < historial.size(); i++) {

*MedicionIMC* medicion = historial.get(i);

            newDataset.addValue(medicion.getIMC(), "Resto de Usuarios", "" + (i + 1));

        }

        chart.getCategoryPlot().setDataset(newDataset);

*CategoryPlot* plot = chart.getCategoryPlot();

*LineAndShapeRenderer* renderer = **new** LineAndShapeRenderer();

        renderer.setSeriesLinesVisible(1, true);

        plot.setRenderer(renderer);

        renderer.setSeriesPaint(0, **new** Color(50, 205, 50));

        renderer.setSeriesPaint(1, Color.RED);

        renderer.setSeriesPaint(2, (**new** Color(0, 0, 255)));

    }

}

package com.imcapp.gui.guiprincipal;

import java.awt.BorderLayout;

import java.awt.Color;

import java.awt.Font;

import java.awt.GridBagConstraints;

import java.awt.GridBagLayout;

import java.awt.GridLayout;

import java.awt.Insets;

import java.awt.event.*ActionListener*;

import java.util.*List*;

import javax.swing.BorderFactory;

import javax.swing.JButton;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JPanel;

import javax.swing.JTextField;

import javax.swing.*SwingConstants*;

import javax.swing.SwingUtilities;

import com.imcapp.gui.StyledButton;

import com.imcapp.gui.historialgui.HistorialFrame;

import com.imcapp.logica.Calculadora;

import com.imcapp.logica.MedicionIMC;

public class GUI extends JFrame {

    private JTextField txtNombre, txtEdad;

    JTextField txtPeso;

    JTextField txtAltura;

    private JLabel lblResultado;

    private JButton btnCalcular, btnGuardar, btnHistorial;

    private GraficoComparativo graficoComparativo;

    private JPanel panelGrafico;

    public GUI() {

        setTitle("Calculadora de IMC");

        setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

        setExtendedState(JFrame.MAXIMIZED\_BOTH);

        // Panel superior con los campos de texto y botones para calcular y guardar

        JPanel panelSuperior = **new** JPanel(**new** GridLayout(7, 2, 10, 10));

        panelSuperior.setBackground(Color.WHITE);

        panelSuperior.setBorder(BorderFactory.createEmptyBorder(0, 10, 0, 10));

        Font labelFont = **new** Font("Arial", Font.BOLD, 20);

        Font textFieldFont = **new** Font("Helvetica", Font.BOLD, 20);

        panelSuperior.add(createLabeledTextField("Nombre:", labelFont, textFieldFont, "txtNombre"));

        panelSuperior.add(createLabeledTextField("Edad:", labelFont, textFieldFont, "txtEdad"));

        panelSuperior.add(createLabeledTextField("Peso (Kg):", labelFont, textFieldFont, "txtPeso"));

        panelSuperior.add(createLabeledTextField("Altura (en metros):", labelFont, textFieldFont, "txtAltura"));

        lblResultado = createResultLabel();

        btnCalcular = createStyledButton("Calcular IMC", **new** Color(0, 128, 0), *e* *->* calcularIMC());

        btnGuardar = createStyledButton("Guardar Datos", **new** Color(0, 0, 128), *e* *->* guardarDatos());

        btnHistorial = createStyledButton("Ver Historial", **new** Color(64, 0, 64), *e* *->* verHistorial());

        panelSuperior.add(btnCalcular);

        panelSuperior.add(lblResultado);

        panelSuperior.add(btnGuardar);

        panelSuperior.add(btnHistorial);

        // Crear el panel para la mitad inferior con el gráfico comparativo

        JPanel panelInferior = **new** JPanel(**new** BorderLayout());

        panelInferior.setBackground(Color.WHITE);

        panelGrafico = **new** JPanel();

        panelInferior.add(panelGrafico, BorderLayout.CENTER);

        graficoComparativo = **new** GraficoComparativo(*this*);

        graficoComparativo.setVisible(false);

        panelGrafico.add(graficoComparativo);

        // Usar BorderLayout para organizar los dos paneles en el JFrame

        setLayout(**new** BorderLayout());

        add(panelSuperior, BorderLayout.CENTER);

        add(panelInferior, BorderLayout.SOUTH);

    }

    private JPanel createLabeledTextField(String *labelText*, Font *labelFont*, Font *textFieldFont*, String *fieldName*) {

        JPanel panel = **new** JPanel(**new** GridBagLayout());

        GridBagConstraints gbc = **new** GridBagConstraints();

        gbc.gridx = 0;

        gbc.gridy = 0;

        gbc.anchor = GridBagConstraints.WEST;

        gbc.insets = **new** Insets(0, 10, 0, 10);

        gbc.ipady = 10;

        JLabel label = **new** JLabel(*labelText*);

        label.setFont(*labelFont*);

        panel.add(label, gbc);

        gbc.gridx = 1;

        gbc.fill = GridBagConstraints.HORIZONTAL;

        gbc.weightx = 1.0;

        JTextField textField = **new** JTextField();

        textField.setFont(*textFieldFont*);

        textField.setOpaque(false);

        textField.setBorder(BorderFactory.createEmptyBorder());

        panel.add(textField, gbc);

        switch (*fieldName*) {

            case "txtNombre":

                txtNombre = textField;

                break;

            case "txtEdad":

                txtEdad = textField;

                break;

            case "txtPeso":

                txtPeso = textField;

                break;

            case "txtAltura":

                txtAltura = textField;

                break;

            default:

                break;

        }

        return panel;

    }

    private JLabel createResultLabel() {

        JLabel resultLabel = **new** JLabel("");

        resultLabel.setFont(**new** Font("Helvetica", Font.BOLD, 24));

        resultLabel.setHorizontalAlignment(*SwingConstants*.CENTER);

        return resultLabel;

    }

    private JButton createStyledButton(String *text*, Color *color*, *ActionListener* *actionListener*) {

        StyledButton button = **new** StyledButton(*text*, *color*);

        button.addActionListener(*actionListener*);

        return button;

    }

    private *void* calcularIMC() {

        try {

            if (txtPeso == null || txtAltura == null) {

                throw **new** Exception("Los campos de texto no están inicializados.");

            }

*double* peso = Double.parseDouble(txtPeso.getText());

*double* altura = Double.parseDouble(txtAltura.getText());

            if (peso <= 0 || altura <= 0) {

                throw **new** ArithmeticException("Peso y altura deben ser mayores a cero.");

            }

*double* imc = peso / (altura \* altura);

            String estadoIMC = Calculadora.calcularIMC(peso, altura);

            Color colorIMC = Calculadora.determinarColorIMC(peso, altura);

            mostrarResultado(estadoIMC, colorIMC, imc);

            verGrafico();

        } catch (NumberFormatException *e*) {

            lblResultado.setText("Error: Ingrese valores numéricos para peso y altura.");

        } catch (ArithmeticException *e*) {

            lblResultado.setText("Error: " + e.getMessage());

        } catch (Exception *e*) {

            lblResultado.setText("Error: " + e.getMessage());

        }

    }

    private *void* verGrafico() {

        // Obtener el historial de mediciones

*List*<MedicionIMC> historial = Calculadora.obtenerHistorial();

        // Actualizar el historial y recalcular el gráfico

        graficoComparativo.representacionDatos(historial);

        // Mostrar u ocultar el gráfico según sea necesario

        graficoComparativo.setVisible(!graficoComparativo.isVisible());

        // Actualizar la GUI

*this*.revalidate();

*this*.repaint();

    }

    private *void* mostrarResultado(String *estadoIMC*, Color *colorIMC*, *double* *imc*) {

        String mensaje = String.format("%s (IMC: %.1f)", *estadoIMC*, *imc*);

        SwingUtilities.invokeLater(() *->* {

            lblResultado.setText(mensaje);

            lblResultado.setForeground(*colorIMC*);

        });

    }

    private *void* guardarDatos() {

        try {

            String nombre = txtNombre.getText();

            String edad = txtEdad.getText();

*double* peso = Double.parseDouble(txtPeso.getText());

*double* altura = Double.parseDouble(txtAltura.getText());

            Calculadora.guardarMedicion(nombre, edad, peso, altura);

            lblResultado.setText("Datos guardados con éxito para el usuario " + nombre + ".");

        } catch (NumberFormatException *e*) {

            lblResultado.setText("Error: Ingrese valores numéricos para peso y altura.");

        }

    }

    private *void* verHistorial() {

*List*<MedicionIMC> historial = Calculadora.obtenerHistorial();

        HistorialFrame historialFrame = **new** HistorialFrame(historial);

        historialFrame.setLocationRelativeTo(null);

        historialFrame.setVisible(true);

    }

*double* getTxtPesoValue() {

        try {

            return Double.parseDouble(txtPeso.getText());

        } catch (NumberFormatException *e*) {

            e.printStackTrace();

            return 0.0;

        }

    }

    public *double* getTxtAlturaValue() {

        try {

            return Double.parseDouble(txtAltura.getText());

        } catch (NumberFormatException *e*) {

            e.printStackTrace();

            return 0.0; // Valor predeterminado si hay un error

        }

    }

}

package com.imcapp.gui.historialgui;

import java.awt.BorderLayout;

import java.awt.Color;

import java.awt.FlowLayout;

import java.awt.Font;

import java.util.ArrayList;

import java.util.*List*;

import javax.swing.JButton;

import javax.swing.JFrame;

import javax.swing.JPanel;

import javax.swing.JScrollPane;

import javax.swing.JTable;

import javax.swing.border.EmptyBorder;

import javax.swing.table.DefaultTableCellRenderer;

import javax.swing.table.DefaultTableModel;

import javax.swing.table.TableColumn;

import com.imcapp.gui.StyledButton;

import com.imcapp.logica.Calculadora;

import com.imcapp.logica.MedicionIMC;

public class HistorialFrame extends JFrame {

    // Componentes de la interfaz

    private JTable tabla;

    private JButton btnBorrarSeleccionados;

    private JButton btnBorrarTodos;

    private *List*<MedicionIMC> historialActualizado;

    private ModeloTablaSeleccionable modeloTabla;

    // Constructor

    public HistorialFrame(*List*<MedicionIMC> *historial*) {

        setTitle("Historial de Mediciones");

        setExtendedState(JFrame.MAXIMIZED\_BOTH); // Abre la ventana en tamaño completo

        // Crear la tabla

        String[] nombresColumnas = { "Seleccionar", "Nombre", "Edad", "Peso (kg)", "Altura (en metros)", "IMC",

                "Estado" };

        modeloTabla = **new** ModeloTablaSeleccionable(**new** Object[][] {}, nombresColumnas);

        tabla = **new** JTable(modeloTabla); // Crear la tabla

        // Cambiar la fuente de los campos de texto

        Font fuente = **new** Font("Arial", Font.PLAIN, 20); // Crear una nueva fuente

        tabla.setFont(fuente); // Establecer la nueva fuente para la tabla

        // Cambiar el tamaño de los títulos de las columnas

        tabla.getTableHeader().setFont(**new** Font("Arial", Font.BOLD, 18));

        // Cambiar el ancho de las columnas

        TableColumn columna;

        for (*int* i = 0; i < tabla.getColumnCount(); i++) {

            columna = tabla.getColumnModel().getColumn(i);

            columna.setPreferredWidth(200); // Cambiar el ancho preferido de la columna

        }

        // Centrar el texto en las celdas

        DefaultTableCellRenderer centroRender = **new** DefaultTableCellRenderer();

        centroRender.setHorizontalAlignment(DefaultTableCellRenderer.CENTER);

        tabla.setDefaultRenderer(Object.class, centroRender);

        tabla.getColumnModel().getColumn(1).setCellRenderer(centroRender);

        tabla.getColumnModel().getColumn(2).setCellRenderer(centroRender);

        tabla.getColumnModel().getColumn(3).setCellRenderer(centroRender);

        tabla.getColumnModel().getColumn(4).setCellRenderer(centroRender);

        // Agregar las filas

        JScrollPane scrollPanel = **new** JScrollPane(tabla); // Crear el scroll pane

        add(scrollPanel); // Agregar el scroll pane a la ventana

        // Organizar la interfaz con BorderLayout

        setLayout((**new** BorderLayout()));

        add(scrollPanel, BorderLayout.CENTER); // Agregar el scroll pane a la ventana (al panel

        // Panel para los botones en la parte superior derecha

        JPanel panelBotones = **new** JPanel(**new** FlowLayout(FlowLayout.RIGHT));

        panelBotones.setBorder(**new** EmptyBorder(10, 10, 10, 10));

        btnBorrarSeleccionados = **new** StyledButton("Borrar Seleccionados", **new** Color(77, 77, 77));

        btnBorrarSeleccionados.addActionListener(*e* *->* {

            borrarRegistrosSeleccionados();

        });

        panelBotones.add(btnBorrarSeleccionados); // Agregar el botón a la ventana (al panel)

        btnBorrarTodos = **new** StyledButton("Borrar Todos", **new** Color(204, 0, 0));

        btnBorrarTodos.addActionListener(*e* *->* {

            borrarTodosRegistros();

        });

        panelBotones.add(btnBorrarTodos); // Agregar el botón a la ventana (al panel)

        add(panelBotones, BorderLayout.NORTH); // Agregar el panel de botones a la ventana (al panel)

        // Guardar copia del historial original

        historialActualizado = **new** ArrayList<>(*historial*);

        cargarHistorial(*historial*); // Cargar el historial

    }

    // Método para cargar el historial

    private *void* cargarHistorial(*List*<MedicionIMC> *historial*) {

        // Borrar las filas para evitar duplicados

        modeloTabla.setRowCount(0);

        // Agregar las filas

        for (MedicionIMC medicion : *historial*) {

            Object[] datosFila = { // Crear un arregwlo con los datos de la fila

                    false, // Checkbox

                    medicion.getNombre(),

                    medicion.getEdad(),

                    medicion.getPeso(),

                    medicion.getAltura(),

                    medicion.getIMCRedondeado(),

                    medicion.getEstadoIMC()

            };

            modeloTabla.addRow(datosFila); // Agregar la fila al modelo

        }

        // Cambiar el alto de las filas

        for (*int* i = 0; i < tabla.getRowCount(); i++) {

            tabla.setRowHeight(i, 70);

        }

        // Cambiar el espaciado entre las celdas

        tabla.setIntercellSpacing(**new** java.awt.Dimension(2, 0));

    }

    private *void* borrarTodosRegistros() {

        DefaultTableModel modelo = (DefaultTableModel) tabla.getModel(); // Obtener el modelo de la tabla

        modelo.setRowCount(0); // Borrar todas las filas

        historialActualizado.clear();

    }

    private *void* borrarRegistrosSeleccionados() {

*int* conteoFilas = modeloTabla.getRowCount(); // Obtener el número de filas

        for (*int* i = conteoFilas - 1; i >= 0; i--) {

            Boolean seleccionado = (Boolean) modeloTabla.getValueAt(i, 0);

            if (seleccionado) {

                modeloTabla.removeRow(i);

                MedicionIMC medicion = historialActualizado.get(i);

                Calculadora.borrarMedicion(medicion);

                historialActualizado.remove(i);

            }

        }

    }

    // Mantener este código para posible futura implementación

    // private List<MedicionIMC> getHistorialActualizado() {

    // return new ArrayList<>(historialActualizado);

    // }

}

package com.imcapp.gui.historialgui;

import javax.swing.table.DefaultTableModel;

public class ModeloTablaSeleccionable extends DefaultTableModel {

    private static final *long* serialVersionUID = 1L; // Número de versión para la serialización

    private Class<*?*>[] tiposClases = **new** Class<*?*>[] { Boolean.class, String.class, Integer.class, Double.class, Double.class, String.class, String.class };

    private *boolean*[] editable = **new** *boolean*[] { true, false, false, false, false, false };

    public ModeloTablaSeleccionable(Object[][] *dato*, Object[] *nombresColumnas*) {

*super*(*dato*, *nombresColumnas*);

    }

    @Override

    public Class<*?*> getColumnClass(*int* *columnIndex*) {

        return tiposClases[*columnIndex*];

    }

    @Override

    public *boolean* isCellEditable(*int* *rowIndex*, *int* *columnIndex*) {

        return editable[*columnIndex*];

    }

}

package com.imcapp.gui;

import java.awt.Color;

import javax.swing.BorderFactory;

import javax.swing.JButton;

public class StyledButton extends JButton {

    public StyledButton(String *text*, Color *color*) {

        // Llamar al constructor de la clase padre

*super*(*text*);

        // Establecer el estilo del botón

        setBorder(BorderFactory.createCompoundBorder(BorderFactory.createLineBorder(Color.BLACK, 2),

                (BorderFactory.createEmptyBorder(5, 10, 5, 10)))); // Establecer el borde del botón con un grosor de 2

        setContentAreaFilled(false);

        // Establecer el color de fondo y el color de primer plano

        setOpaque(true);

        // Establecer el color de fondo y el color de primer plano

        setBackground(*color*);

        // Establecer el color de primer plano

        setForeground(Color.WHITE);

        // Cambiar el tamaño de la fuente

        setFont(getFont().deriveFont(20f));

    }

}

package com.imcapp.logica;

import java.awt.Color;

import java.io.File;

import java.io.FileInputStream;

import java.io.FileOutputStream;

import java.io.IOException;

import java.io.ObjectInputStream;

import java.io.ObjectOutputStream;

import java.io.*Serializable*;

import java.util.ArrayList;

import java.util.*List*;

public class Calculadora implements *Serializable* {

    // Constantes

    // Aviso: Este campo serialVersionUID se declara para garantizar la

    // compatibilidad de la serialización

    @SuppressWarnings("unused") // Suprimir advertencia de variable no utilizada

    private static final *long* SERIAL\_VERSION\_UID = 1L; // Numero de version para la serializacion

    // Atributos

    private static *List*<MedicionIMC> historial = **new** ArrayList<>();

    // Método para calcular el IMC

    public static String calcularIMC(*double* *peso*, *double* *altura*) {

*double* imc = *peso* / (*altura* \* *altura*);

        return determinarEstadoIMC(imc);

    }

    // Método para determinar el estado del IMC

    static String determinarEstadoIMC(*double* *imc*) {

        if (*imc* < 18.5) {

            return "Bajo peso";

        } else if (*imc* < 25) {

            return "Peso normal";

        } else {

            return "Sobrepeso";

        }

    }

    // Método para determinar el color del IMC

    public static Color determinarColorIMC(*double* *peso*, *double* *altura*) {

*double* imc = *peso* / (*altura* \* *altura*);

        if (imc < 18.5) {

            return Color.RED;

        } else if (imc < 24.9) {

            return Color.GREEN;

        } else {

            return **new** Color(255, 165, 0);

        }

    }

    // Método para guardar una medición

    public static *void* guardarMedicion(String *nombre*, String *edad*, *double* *peso*, *double* *altura*) {

        MedicionIMC medicion = **new** MedicionIMC(*nombre*, *edad*, *peso*, *altura*);

        historial.add(medicion);

        guardarHistorial(); // Guardar el historial en un archivo

    }

    // Método para borrar una medición

    public static *void* borrarMedicion(MedicionIMC *medicion*) {

        historial.remove(*medicion*);

        guardarHistorial(); // Guardar el historial en un archivo

    }

    // Método para obtener el historial

    public static *List*<MedicionIMC> obtenerHistorial() {

        return historial;

    }

    // Método para guardar el historial en un archivo

    public static *void* guardarHistorial() {

        try (ObjectOutputStream oos = **new** ObjectOutputStream(**new** FileOutputStream("historial.dat"))) {

            oos.writeObject(historial); // Escribir el objeto en el flujo

        } catch (IOException *e*) { // Manejamos IOException que es la excepción que puede lanzar FileOutputStream

            e.printStackTrace();

            System.out.println("Error al guardar el historial.");

        }

    }

    // Método para cargar el historial desde un archivo

    @SuppressWarnings("unchecked") // Suprimir advertencia de unchecked cast.

    public static *void* cargarHistorial() {

        File file = **new** File("historial.dat"); // Crear el archivo

        if (file.exists()) { // Verificar si el archivo existe

            try (ObjectInputStream ois = **new** ObjectInputStream(**new** FileInputStream("historial.dat"))) {

                historial = (*List*<MedicionIMC>) ois.readObject(); // Leer el objeto del flujo

            } catch (IOException | ClassNotFoundException *e*) {

                e.printStackTrace();

            }

        }

    }

}

package com.imcapp.logica;

import java.io.*Serializable*;

public class MedicionIMC implements *Serializable* {

    // Atributos

    private String nombre;

    private String edad;

    private *double* peso;

    private *double* altura;

    // Constructor

    public MedicionIMC(String *nombre*, String *edad*, *double* *peso*, *double* *altura*) {

*this*.nombre = *nombre*;

*this*.edad = *edad*;

*this*.peso = *peso*;

*this*.altura = *altura*;

    }

    // Getters y setters

    public String getNombre() {

        return nombre;

    }

    public *void* setNombre(String *nombre*) {

*this*.nombre = *nombre*;

    }

    public String getEdad() {

        return edad;

    }

    public *void* setEdad(String *edad*) {

*this*.edad = *edad*;

    }

    public *double* getPeso() {

        return peso;

    }

    public *void* setPeso(*double* *peso*) {

*this*.peso = *peso*;

    }

    public *double* getAltura() {

        return altura;

    }

    public *void* setAltura(*double* *altura*) {

*this*.altura = *altura*;

    }

    // Método para obtener el IMC

    public *double* getIMC() {

        return peso / (altura \* altura);

    }

    // Método para obtener el IMC redondeado a un decimal

    public *double* getIMCRedondeado() {

        return Math.round(getIMC() \* 10) / 10.0;

    }

    // Método para obtener el estado del IMC

    public String getEstadoIMC() {

        return Calculadora.determinarEstadoIMC(getIMC());

    }

}

package com.imcapp;

import java.awt.event.WindowAdapter;

import java.awt.event.WindowEvent;

import javax.swing.SwingUtilities;

import com.imcapp.gui.guiprincipal.GUI;

import com.imcapp.logica.Calculadora;

public class App {

    // Método main

    public static *void* main(String[] *args*) {

        // Cargar el historial al inicio

        Calculadora.cargarHistorial();

        // Crear la interfaz

        SwingUtilities.invokeLater(() *->* { // Ejecutar la interfaz

            GUI gui = **new** GUI(); // Crear la interfaz

            gui.setVisible(true); // Mostrar la interfaz

            // Agregar el evento para guardar el historial al cerrar la ventana

            gui.addWindowListener(**new** WindowAdapter() {

                @Override

                public *void* windowClosing(WindowEvent *e*) {

                    Calculadora.guardarHistorial();

                    System.exit(0); // Cerrar la aplicación

                }

            });

        });

    }

}

**VISTAS:**

**Escala de tiempo

Descripción generada automáticamente**

**Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente**

**Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente**

Tabla

Descripción generada automáticamente