**UCSM Esc. Prof. de Ingeniería de Sistemas**

**Noviembre - 2024**

INFORME DE PRÁCTICAS

Lenguaje de Programación

Práctica N° 09: JAVA SWING BÁSICO: INTRODUCCIÓN Y CONTROLES

## Elaborado por:

## 4 razones para aprender programación- Bruno Torres Ana Paula

**GRUPO N° 04**

**PRÁCTICAS DE LENGUAJE DE PROGRAMACIÓN**

Presentado por: Bruno Torres Ana Paula

**RECONOCIMIENTOS**

Queremos expresar nuestros más sinceros reconocimientos a Diego Alonso Iquira Becerra, quien ha sido un pilar fundamental en la enseñanza de los lenguajes de programación en este curso de Lenguaje de Programación III. Su dedicación y compromiso en la formación de los estudiantes, especialmente en la comprensión de conceptos complejos en Java, han dejado una huella significativa en nuestra comunidad académica.

**PALABRAS CLAVES**

Interfaces Gráficas de Usuario (GUI),Java Swing,Java, Componentes de Swing,Clases Adaptadoras, Desarrollo Visual

**ÍNDICE**

1. [RESÚMEN 1](#_bookmark0)
2. [INTRODUCCIÓN 1](#_bookmark1)
3. [MARCO TEÓRICO 2](#_bookmark2)
4. [ACTIVIDADES 3](#_bookmark3)
5. [EJERCICIOS 7](#_bookmark4)
6. [CONCLUSIONES DE LA PRÁCTICA 16](#_bookmark5)
7. [BIBLIOGRAFÍA 17](#_bookmark6)
8. **RESÚMEN**

Este informe aborda el desarrollo de interfaces gráficas de usuario (GUI) utilizando la biblioteca Java Swing, destacando su relevancia en la creación de aplicaciones interactivas. Se analiza la diferencia entre Swing y AWT, resaltando que Swing ofrece componentes ligeros, personalizables y con apariencia uniforme en distintas plataformas. También se describen los principales componentes de Swing, como JButton, JLabel, y JTextField, esenciales para construir GUIs funcionales.

El manejo de eventos se aborda desde el modelo de delegación, explicando cómo los eventos generados por los componentes son procesados por manejadores que implementan interfaces como ActionListener o utilizan clases adaptadoras. Asimismo, se profundiza en los administradores de esquemas, como FlowLayout, BorderLayout y GridLayout, que permiten organizar los elementos visuales de forma eficiente.

Finalmente, se destaca el uso de herramientas de diseño visual en IDEs como NetBeans, que facilitan el desarrollo de GUIs mediante operaciones de "arrastrar y soltar". El informe complementa los conceptos teóricos con actividades prácticas que refuerzan las habilidades en la creación de interfaces interactivas y organizadas.

# INTRODUCCIÓN

Las interfaces gráficas de usuario (GUI) son un elemento fundamental en el desarrollo de aplicaciones modernas, ya que permiten a los usuarios interactuar de manera visual e intuitiva con los sistemas. En el lenguaje de programación Java, la biblioteca Swing se destaca por ofrecer herramientas versátiles para la creación de GUIs, permitiendo el diseño de aplicaciones visuales con componentes ligeros, personalizables y uniformes en diversas plataformas.

Este informe presenta los conceptos básicos relacionados con el desarrollo de GUIs utilizando Java Swing. Se abordarán temas como los principales componentes gráficos, el manejo de eventos mediante modelos de delegación y la organización de elementos visuales a través de administradores de esquemas. Además, se explorarán herramientas visuales para la construcción de interfaces y se realizarán ejercicios prácticos para aplicar los conceptos aprendidos.

El objetivo es proporcionar una introducción clara y estructurada al uso de Java Swing, permitiendo a los desarrolladores implementar interfaces gráficas funcionales y eficientes en sus aplicaciones.

# MARCO TEÓRICO

### ¿Qué son las interfaces gráficas de usuario (GUI)?

Una interfaz gráfica de usuario (GUI) es un medio visual y amigable que permite al usuario interactuar con una aplicación a través de elementos como botones, menús y cuadros de texto. En Java, las GUI se desarrollan utilizando bibliotecas específicas como AWT y Swing, siendo esta última más robusta y versátil al ofrecer componentes ligeros y uniformes en todas las plataformas (Guevara et al., 2024).

### Diferencias entre AWT y Swing

AWT (Abstract Window Toolkit) fue la primera biblioteca para GUI en Java. Sus componentes dependen del sistema operativo, lo que significa que su apariencia varía según la plataforma. Por otro lado, Swing, que pertenece al paquete javax.swing, es una biblioteca completamente escrita en Java, lo que garantiza una apariencia uniforme y personalizable. Además, Swing incluye funcionalidades avanzadas como soporte de temas, nemónicos y cuadros de herramientas.

### Componentes básicos de Swing

Swing proporciona un conjunto de componentes visuales para construir aplicaciones interactivas. Algunos de los más utilizados son:

* + - **JLabel:** Etiquetas de texto o imágenes.
    - **JButton:** Botones para ejecutar acciones al ser presionados.
    - **JTextField:** Campos de texto para ingreso de datos.
    - **JPasswordField:** Campos de texto con protección para contraseñas.
    - **JPanel:** Contenedores para organizar components (Guevara et al., 2024).

### Administradores de esquemas

Los administradores de esquemas en Java determinan la disposición de los componentes en una ventana. Algunos de los más comunes son:

* + - **FlowLayout**: Organiza los componentes horizontalmente, ajustándose al tamaño del contenedor.
    - **BorderLayout:** Divide el contenedor en cinco regiones: norte, sur, este, oeste y centro.
    - **GridLayout:** Distribuye los componentes en una cuadrícula uniforme.

.



*Ilustración 1: Interfaz de usuario.*

# ACTIVIDADES

### EXPERIENCIA DE PRÁCTICA N° 01:

Según el código visto en el marco teórico implementen tres ventanas que utilicen diferentes tipos de administradores de esquemas FlowLayout, BorderLayout y GridLayout. Para la implementación de cada uno de los administradores, creen ventanas con títulos diferentes, agréguenles diferentes funcionalidades según los temas vistos en el marco teórico.

Hagan capturas de pantalla por cada implementación de administrador de esquema de tal forma que se evidencie cómo se organiza el contenido y las interacciones que se puede realizar en cada ventana.

**package** E;

**import** javax.swing.\*;

**import** java.awt.\*;

**import** java.awt.event.ActionEvent; **import** java.awt.event.ActionListener; **import** java.util.Random;

**public class** Aplicaciones {

**public static void** main(String[] args) { SwingUtilities.*invokeLater*(() -> **new** JuegoAdivinaNumero()); SwingUtilities.*invokeLater*(() -> **new** Calculadora()); SwingUtilities.*invokeLater*(() -> **new** FrasesMotivacionales());

}

}

//CON GRIDLAYOUT

**class** JuegoAdivinaNumero **extends** JFrame {

**private int** numeroSecreto;

**private int** intentos;

**public** JuegoAdivinaNumero() { **super**("Adivina el Número"); setLayout(**new** GridLayout(3, 1, 5, 5));

// Generar número RANDM Y TAMBIEN SECRETO Random random = **new** Random(); numeroSecreto = random.nextInt(100) + 1; intentos = 0;

JLabel mensaje = **new** JLabel("Adivina un número entre 1 y 100", JLabel.***CENTER***); add(mensaje);

JTextField entrada = **new** JTextField(); add(entrada);

JButton verificar = **new** JButton("Intentar"); verificar.addActionListener(e -> {

**try** {

**int** adivinanza = Integer.*parseInt*(entrada.getText()); intentos++;

**if** (adivinanza == numeroSecreto) {

mensaje.setText("¡Correcto! Lo lograste en " + intentos + " intentos."); verificar.setEnabled(**false**);

} **else if** (adivinanza < numeroSecreto) { mensaje.setText("Demasiado bajo. ¡Intenta de nuevo!");

} **else** {

mensaje.setText("Demasiado alto. ¡Intenta de nuevo!");

}

} **catch** (NumberFormatException ex) {

mensaje.setText("Por favor, ingresa un número válido.");

}

});

add(verificar);

setSize(300, 200); setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***); setVisible(**true**);

}

}

//BORDERLAYOUT EN ESTE CASO CALCULADORA

**class** Calculadora **extends** JFrame {

**private** String operador;

**private double** numero1;

**public** Calculadora() { **super**("Calculadora "); setLayout(**new** BorderLayout(5, 5));

// Panel de entrada

JTextField pantalla = **new** JTextField(); pantalla.setHorizontalAlignment(JTextField.***RIGHT***);

pantalla.setEditable(**false**); // No se puede escribir directamente en la pantalla add(pantalla, BorderLayout.***NORTH***);

// Panel de botones

JPanel panelBotones = **new** JPanel(); panelBotones.setLayout(**new** GridLayout(4, 4, 5, 5));

String[] botones = {"7", "8", "9", "/",

"4", "5", "6", "\*",

"1", "2", "3", "-",

"C", "0", "=", "+"};

**for** (String texto : botones) {

JButton boton = **new** JButton(texto); panelBotones.add(boton);

boton.addActionListener(**new** ActionListener() { @Override

**public void** actionPerformed(ActionEvent e) { String comando = boton.getText();

**if** ("0123456789".contains(comando)) {

// Agregar dígitos a la pantalla pantalla.setText(pantalla.getText() + comando);

} **else if** (comando.equals("C")) {

// Borrar la pantalla y reiniciar valores pantalla.setText("");

operador = **null**; numero1 = 0;

} **else if** (comando.equals("=")) {

// operacion

**try** {

**double** numero2 = Double.*parseDouble*(pantalla.getText());

**double** resultado = realizarOperacion(numero1, numero2, operador); pantalla.setText(String.*valueOf*(resultado));

} **catch** (NumberFormatException ex) { pantalla.setText("Error");

}

} **else** {

// Guardar el operador y el primer número

**try** {

numero1 = Double.*parseDouble*(pantalla.getText()); operador = comando;

pantalla.setText(""); // ESTO ES PARA: limpiar la pantalla para el segundo número

} **catch** (NumberFormatException ex) { pantalla.setText("Error");

}

}

}

});

}

add(panelBotones, BorderLayout.***CENTER***);

setSize(300, 400);

setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***); setVisible(**true**);

}

**private double** realizarOperacion(**double** numero1, **double** numero2, String operador) {

**return switch** (operador) {

**case** "+" -> numero1 + numero2; **case** "-" -> numero1 - numero2; **case** "\*" -> numero1 \* numero2;

**case** "/" -> numero2 != 0 ? numero1 / numero2 : Double.***NaN***; // Manejo de división entre cero

**default** -> 0;

};

}

}

// CON FLOWLAYOUT

**class** FrasesMotivacionales **extends** JFrame {

**public** FrasesMotivacionales() { **super**("Frases Motivacionales"); setLayout(**new** FlowLayout());

// Campo para el nombre

JTextField nombreField = **new** JTextField(15); add(**new** JLabel("Ingresa tu nombre:")); add(nombreField);

JLabel fraseLabel = **new** JLabel(""); add(fraseLabel);

JButton[] botonesFrases = {

**new** JButton("Frase 1"),

**new** JButton("Frase 2"),

**new** JButton("Frase 3")

};

String[] frases = {

"NUNCA TE RINDAS, %s Puedes lograr grandes cosas.",

"Confia en tus habilidades, %s El mundo esta lleno de oportunidades.", "CADA DIA ES UN NUEVO COMIENZO, %s Aprovecha al maximo."

};

**for** (**int** i = 0; i < botonesFrases.length; i++) { **int** index = i; botonesFrases[i].addActionListener(e -> {

String nombre = nombreField.getText();

**if** (!nombre.isEmpty()) { fraseLabel.setText(String.*format*(frases[index], nombre));

} **else** {

fraseLabel.setText("....................No ingresaste tu nombre, ingresa tu

nombre \n");

}

});

add(botonesFrases[i]);

}

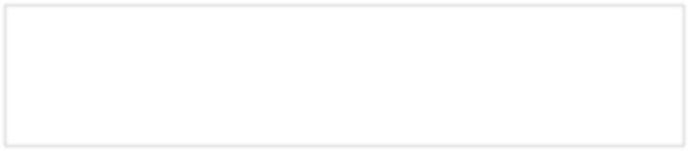
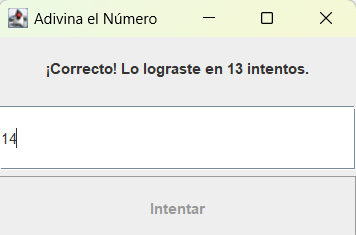
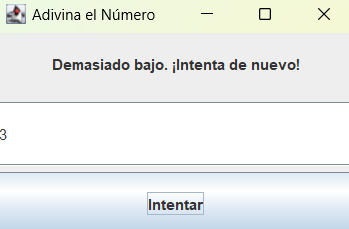
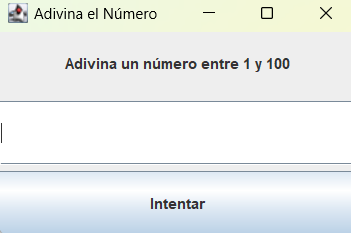
setSize(350, 200); setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***); setVisible(**true**);

}

}

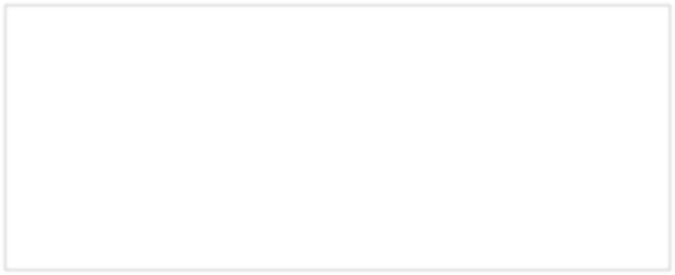
POR PANTALLA

GRINDLAYOUT



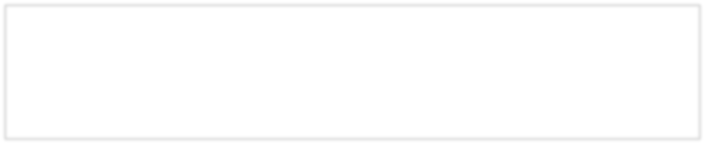
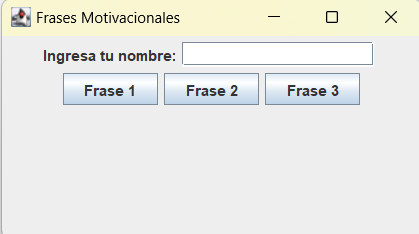
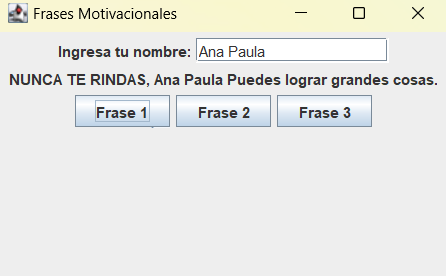
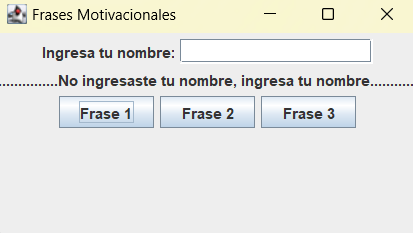
*Ilustración 2: Ventana de GrindLayout.*

BORDERLAYOUT



*Ilustración 3: Ventana de BorderLayout*

FLOWLAYOUT



*Ilustración 4: Ventana de FlowLayout*

### EXPLICACIÓN

Este programa Java presenta tres ventanas gráficas utilizando diferentes administradores de diseño **(LayoutManager).** La primera es un juego de adivinanza de números donde el usuario intenta adivinar un número entre 1 y 100, con un límite de intentos. La segunda es una calculadora básica que permite realizar operaciones aritméticas con una interfaz de botones. La tercera aplicación muestra frases motivacionales personalizadas que incluyen el nombre del usuario, usando un diseño más simple con FlowLayout. Cada aplicación está organizada en clases separadas y se ejecuta dentro del mismo hilo utilizando **SwingUtilities.invokeLater.**

# EJERCICIOS

### EJERCICIO 01

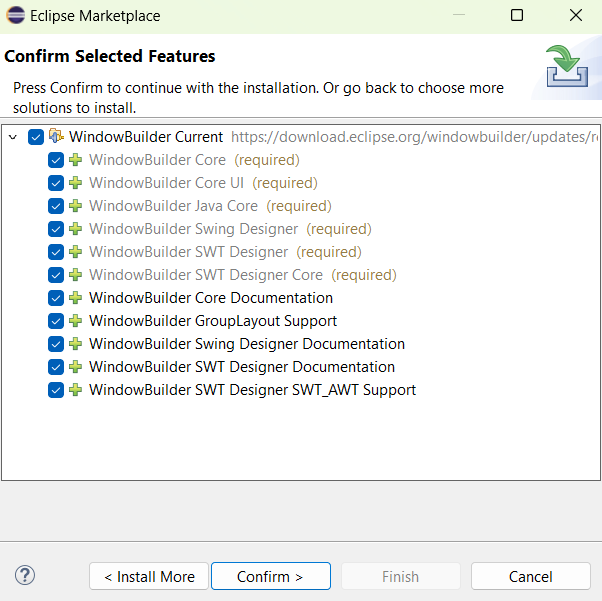
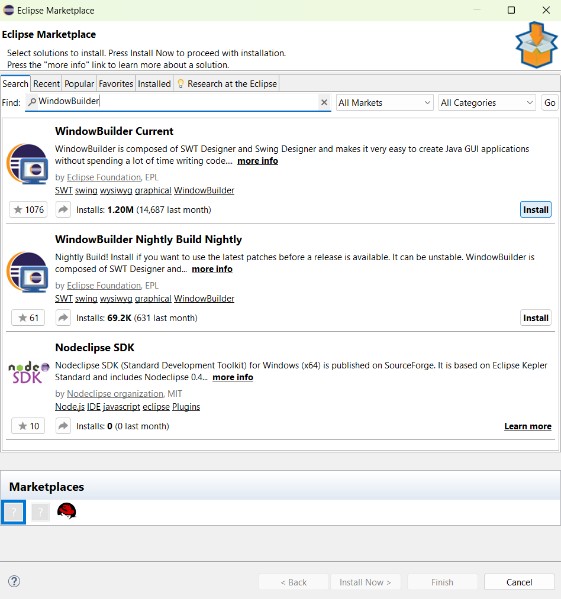
Investigue en el IDE de su preferencia cómo se puede utilizar herramientas visuales que permitan crear componentes GUI Swing a través de operaciones “soltar y arrastrar” realizadas con el mouse de tal manera que sólo se tenga que agregar código a las clases y manejadores ya creados de manera automática por el IDE.

Explique brevemente lo siguiente:

### - Como es el proceso de creación de una GUI utilizando herramientas de “soltar y arrastrar” y haga capturas de pantalla que evidencien el mismo.

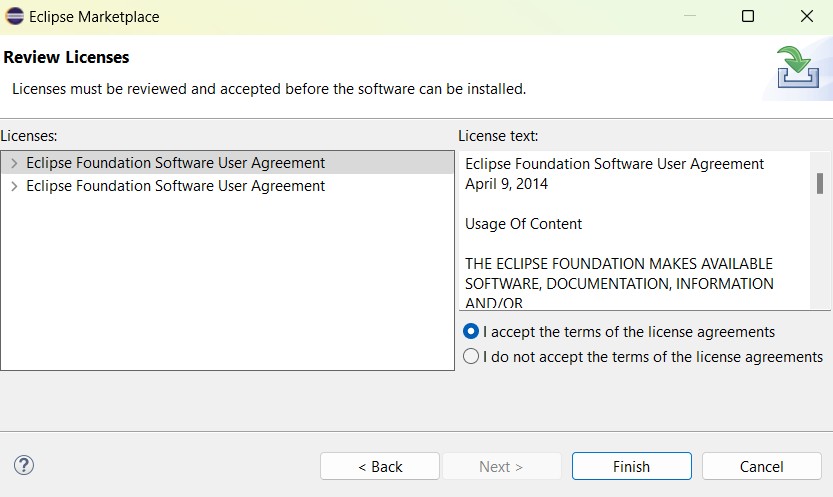
### PASO 1

Para instalar WindowBuilder en Eclipse, se debe abrir el entorno de desarrollo y acceder al menú "Help" (Ayuda). Luego, se selecciona la opción "Eclipse Marketplace" y, en el campo de búsqueda, se escribe "WindowBuilder". Al aparecer en los resultados, se debe hacer clic en el botón "Go", seguido de la opción "Install" para iniciar la instalación.

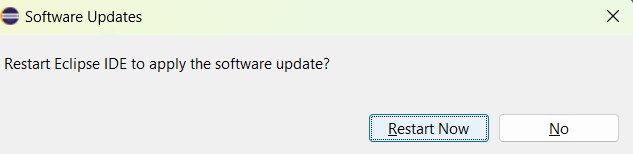


*Ilustración 5: Eclipse Marketplace. Ilustración 6: Confirmación de la instalación y sus*

*características.*



*Ilustración 7: Aceptamos términos y condiciones.*

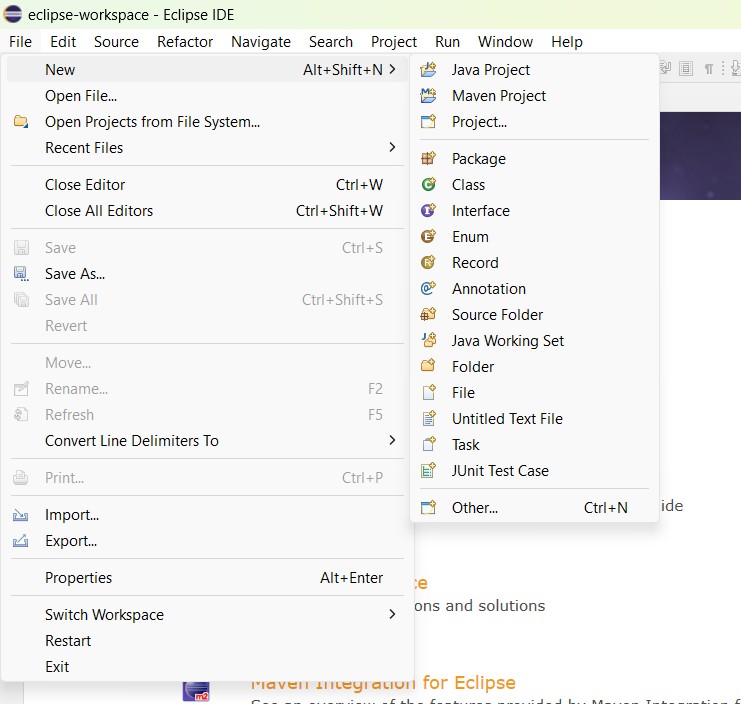


*Ilustración 8: Reiniciamos Eclipse.*

### PASO 2

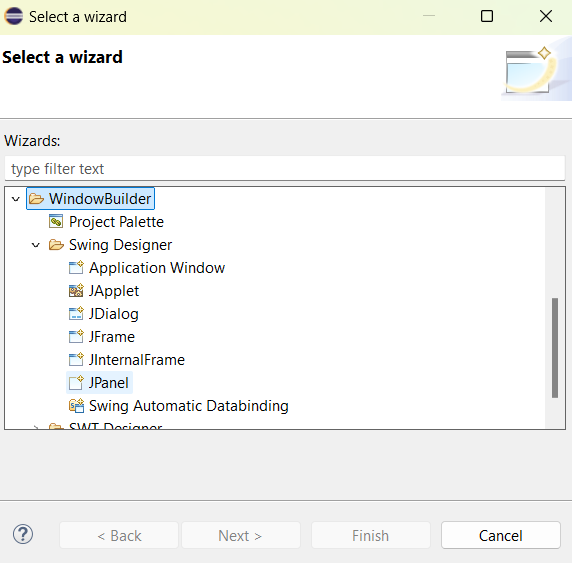
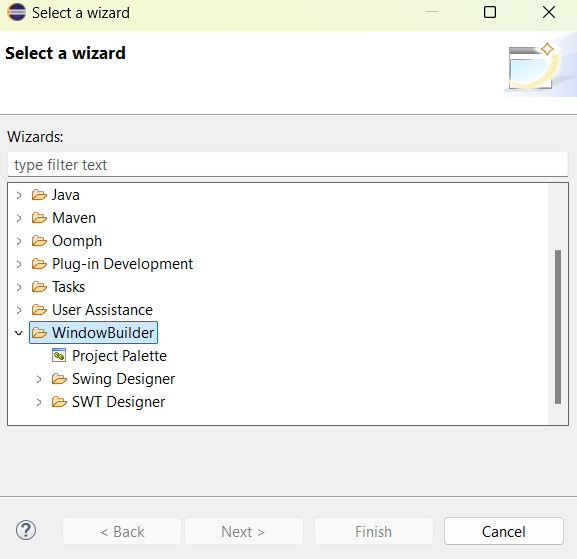
Se debe ir a File > New > Java Project para crear un nuevo proyecto.

Luego, se **selecciona File > New > Other > WindowBuilder** > Swing Designer > JFrame para crear una nueva clase JFrame, que será la ventana principal de la interfaz.

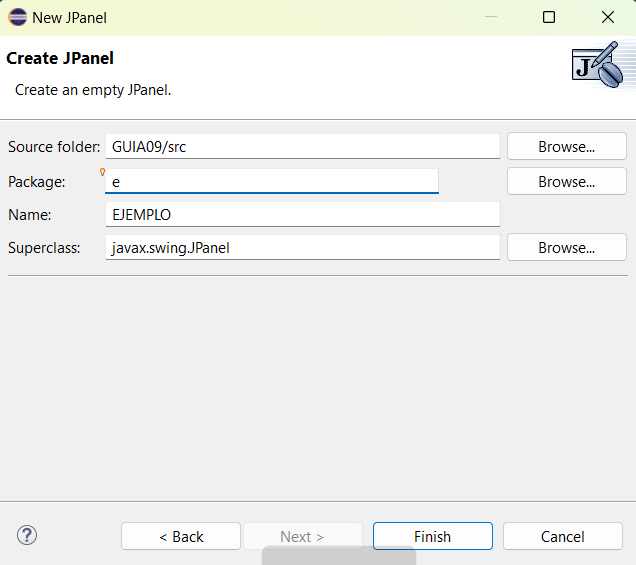


*Ilustración 9: Dentro de Project poner other.*

-

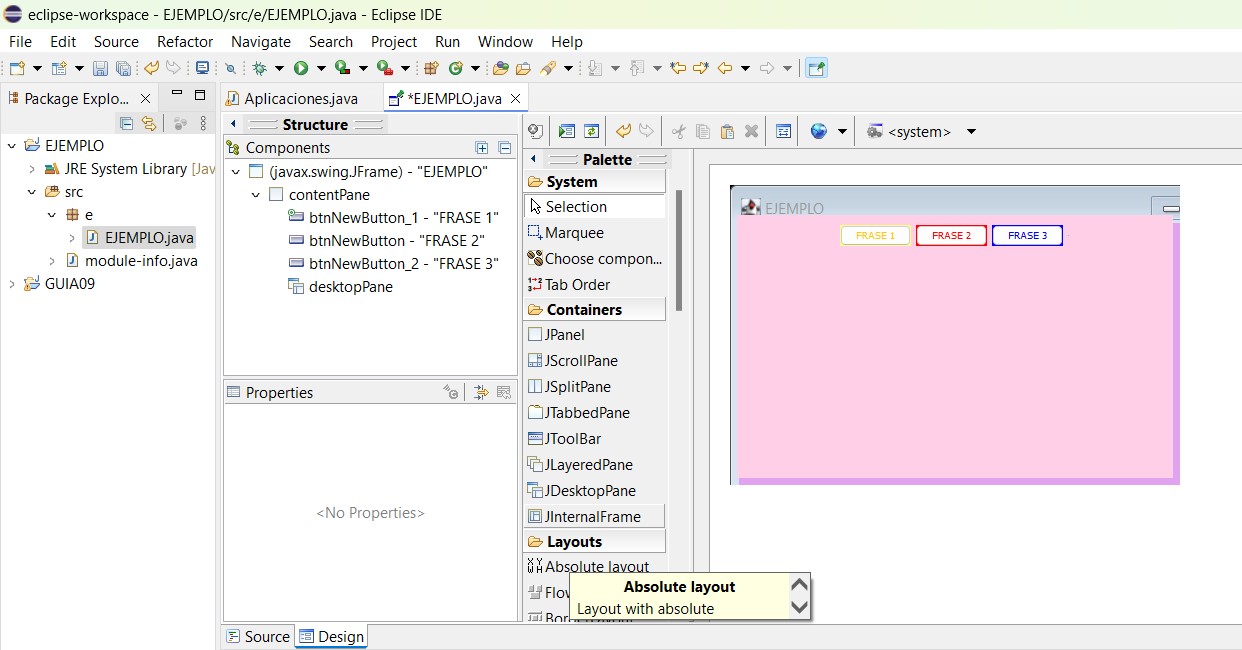


*Ilustración 10: Seleccionamos WindowBuilder. Ilustración 11: Seleccionamos JFrame como ejemplo.*



*Ilustración 12: Nombramos JPanel.*

Una vez creada la clase JFrame, Eclipse abrirá una vista de diseño con el editor visual de Swing. Este editor permite arrastrar y soltar componentes gráficos (botones, etiquetas, cuadros de texto, etc.) directamente sobre el área de diseño.



*Ilustración 13: Prueba de arrastrar y soltar*

### CODIGO GENERADO

**package** e;

**import** java.awt.EventQueue;

**import** javax.swing.JFrame;

**import** javax.swing.JPanel;

**import** javax.swing.border.EmptyBorder;

**import** javax.swing.JButton;

**import** java.awt.event.ActionListener; **import** java.awt.event.ActionEvent; **import** java.awt.Color;

**import** javax.swing.JLabel;

**import** javax.swing.SwingConstants; **import** javax.swing.JDesktopPane; **import** javax.swing.JMenu;

**import** javax.swing.JSpinner;

**import** java.awt.FlowLayout;

**public class** EJEMPLO **extends** JFrame {

**private static final long *serialVersionUID*** = 1L;

**private** JPanel contentPane;

/\*\*

\* Launch the application.

\*/

**public static void** main(String[] args) { EventQueue.*invokeLater*(**new** Runnable() {

**public void** run() {

**try** {

EJEMPLO frame = **new** EJEMPLO(); frame.setVisible(**true**);

} **catch** (Exception e) {

e.printStackTrace();

}

}

});

}

/\*\*

\* Create the frame.

\*/

**public** EJEMPLO() {

setTitle("EJEMPLO");

setBackground(**new** Color(225, 163, 239)); setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***); setBounds(100, 100, 450, 300);

contentPane = **new** JPanel();

FlowLayout flowLayout = (FlowLayout) contentPane.getLayout(); contentPane.setBackground(**new** Color(255, 206, 231));

contentPane.setBorder(**new** EmptyBorder(5, 5, 5, 5)); setContentPane(contentPane);

JButton btnNewButton\_1 = **new** JButton("FRASE 1"); btnNewButton\_1.setForeground(Color.***ORANGE***); btnNewButton\_1.setBackground(**new** Color(255, 255, 128)); btnNewButton\_1.addActionListener(**new** ActionListener() {

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

}

});

contentPane.add(btnNewButton\_1);

JButton btnNewButton = **new** JButton("FRASE 2"); btnNewButton.setVerticalAlignment(SwingConstants.***BOTTOM***); btnNewButton.setHorizontalAlignment(SwingConstants.***TRAILING***); btnNewButton.setForeground(Color.***RED***); btnNewButton.setBackground(Color.***RED***); contentPane.add(btnNewButton);

JButton btnNewButton\_2 = **new** JButton("FRASE 3"); btnNewButton\_2.setForeground(Color.***BLUE***); btnNewButton\_2.setBackground(Color.***BLUE***); contentPane.add(btnNewButton\_2);

JDesktopPane desktopPane = **new** JDesktopPane(); contentPane.add(desktopPane);

}

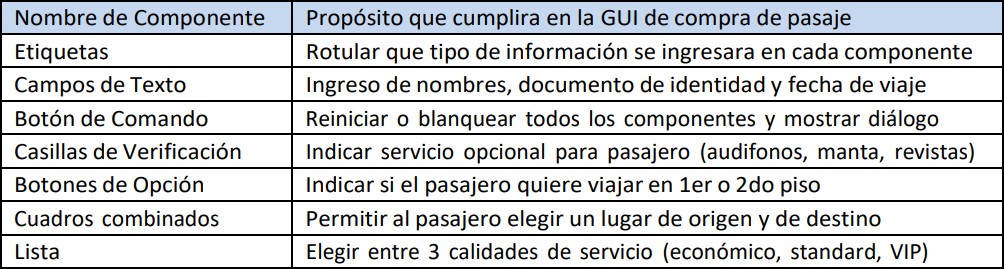
}

### - Cuáles serían algunas ventajas y desventajas de crear una GUI desde código y de hacerlo con una herramienta de “soltar y arrastrar”

Crear una GUI utilizando una herramienta de "soltar y arrastrar" ofrece ventajas como facilidad de uso, ahorro de tiempo, una interfaz intuitiva y una menor probabilidad de errores al evitar la escritura manual del código. Sin embargo, su principal desventaja es que el código generado puede ser menos eficiente, más difícil de mantener y ofrecer menor flexibilidad para personalizaciones avanzadas. Por otro lado, crear una GUI desde código proporciona un mayor control y flexibilidad, permitiendo escribir un código más limpio y optimizado, y una mejor comprensión del funcionamiento interno de la interfaz. No obstante, esta opción requiere más tiempo y esfuerzo, y la ausencia de una vista previa gráfica puede hacer que el proceso sea más propenso a errores y más difícil de visualizar el resultado final.

### EJERCICIO 02

Implemente una pequeña aplicación que simule el ingreso de datos para la compra de pasajes de una empresa de transporte terrestre. Para esto deberá utilizar todos los conceptos y componentes ya vistos. Una vez que se ingresen todos los datos del pasajero, se deberá mostrar un resumen de sus datos en un cuadro de Dialogo luego de presionar un botón. A continuación, se muestran cómo se utilizarían los componentes para ingresar un tipo de información específica:



*Tabla 1: Componentes a ingresar*

CODIGO IMPLEMENTADO

**package** e;

**import** java.awt.GridLayout;

**import** java.awt.event.ActionEvent;

**import** java.awt.event.ActionListener;

**import** javax.swing.ButtonGroup; **import** javax.swing.JButton; **import** javax.swing.JCheckBox; **import** javax.swing.JComboBox; **import** javax.swing.JFrame; **import** javax.swing.JLabel; **import** javax.swing.JOptionPane; **import** javax.swing.JPanel; **import** javax.swing.JRadioButton; **import** javax.swing.JTextField; **import** java.awt.Color;

**import** java.awt.Font;

**import** java.awt.Toolkit;

**public class** EJEMPLO **extends** JFrame {

**private** JTextField txtNombre, txtDocumento, txtFechaViaje; **private** JCheckBox chkAudifonos, chkManta, chkRevistas; **private** JRadioButton rdbPrimero, rdbSegundo;

**private** JComboBox<String> cmbOrigen, cmbDestino;

**private** JComboBox<String> cmbCalidad;

**private** JButton btnReiniciar, btnResumen;

**public** EJEMPLO() {

setIconImage(Toolkit.*getDefaultToolkit*().getImage("C:\\Users\\ESPERANZA\\Downloads\\854faee24f83d86 e28785d1044596609-removebg-preview.png"));

getContentPane().setBackground(**new** Color(204, 185, 255));

setBackground(**new** Color(156, 119, 255)); setTitle("Compra de Pasajes"); setSize(700, 480);

setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***); getContentPane().setLayout(**new** GridLayout(10, 2, 5, 5));

JLabel label\_1 = **new** JLabel("Nombre del Pasajero:");

label\_1.setFont(**new** Font("Lucida Calligraphy", Font.***PLAIN***, 14)); getContentPane().add(label\_1);

txtNombre = **new** JTextField();

txtNombre.setFont(**new** Font("Lucida Calligraphy", Font.***PLAIN***, 10)); getContentPane().add(txtNombre);

JLabel label = **new** JLabel("Documento de Identidad:"); label.setFont(**new** Font("Lucida Calligraphy", Font.***PLAIN***, 14)); label.setBackground(**new** Color(255, 255, 128)); getContentPane().add(label);

txtDocumento = **new** JTextField();

txtDocumento.setFont(**new** Font("Lucida Calligraphy", Font.***PLAIN***, 10)); getContentPane().add(txtDocumento);

JLabel label\_2 = **new** JLabel("Fecha de Viaje (dd/mm/yyyy):"); label\_2.setFont(**new** Font("Lucida Calligraphy", Font.***PLAIN***, 13)); getContentPane().add(label\_2);

txtFechaViaje = **new** JTextField();

txtFechaViaje.setFont(**new** Font("Lucida Calligraphy", Font.***PLAIN***, 10)); getContentPane().add(txtFechaViaje);

ButtonGroup grupoPiso = **new** ButtonGroup(); rdbPrimero = **new** JRadioButton("1er Piso");

rdbPrimero.setFont(**new** Font("Lucida Calligraphy", Font.***PLAIN***, 10)); rdbSegundo = **new** JRadioButton("2do Piso");

rdbSegundo.setFont(**new** Font("Lucida Calligraphy", Font.***PLAIN***, 10)); grupoPiso.add(rdbPrimero);

grupoPiso.add(rdbSegundo);

JLabel label\_3 = **new** JLabel("Seleccione Piso:"); label\_3.setFont(**new** Font("Lucida Calligraphy", Font.***PLAIN***, 14)); getContentPane().add(label\_3);

JPanel panelPiso = **new** JPanel(); panelPiso.add(rdbPrimero); panelPiso.add(rdbSegundo); getContentPane().add(panelPiso);

JLabel label\_4 = **new** JLabel("Servicios Opcionales:"); label\_4.setFont(**new** Font("Lucida Calligraphy", Font.***PLAIN***, 14)); getContentPane().add(label\_4);

JPanel panelServicios = **new** JPanel(); chkAudifonos = **new** JCheckBox("Audífonos");

chkAudifonos.setFont(**new** Font("Lucida Calligraphy", Font.***PLAIN***, 10)); chkManta = **new** JCheckBox("Manta");

chkManta.setFont(**new** Font("Lucida Calligraphy", Font.***PLAIN***, 10)); chkRevistas = **new** JCheckBox("Revistas");

chkRevistas.setFont(**new** Font("Lucida Calligraphy", Font.***PLAIN***, 10)); panelServicios.add(chkAudifonos);

panelServicios.add(chkManta); panelServicios.add(chkRevistas); getContentPane().add(panelServicios);

JLabel label\_5 = **new** JLabel("Seleccione Origen:"); label\_5.setFont(**new** Font("Lucida Calligraphy", Font.***PLAIN***, 14)); getContentPane().add(label\_5);

cmbOrigen = **new** JComboBox<>(**new** String[]{"Lima", "Arequipa", "Cusco"}); cmbOrigen.setFont(**new** Font("Lucida Calligraphy", Font.***PLAIN***, 10)); getContentPane().add(cmbOrigen);

JLabel label\_6 = **new** JLabel("Seleccione Destino:"); label\_6.setFont(**new** Font("Lucida Calligraphy", Font.***PLAIN***, 14)); getContentPane().add(label\_6);

cmbDestino = **new** JComboBox<>(**new** String[]{"Trujillo", "Tacna", "Piura", "Cusco"}); cmbDestino.setFont(**new** Font("Lucida Calligraphy", Font.***PLAIN***, 10)); getContentPane().add(cmbDestino);

JLabel label\_7 = **new** JLabel("Calidad de Servicio:"); label\_7.setFont(**new** Font("Lucida Calligraphy", Font.***PLAIN***, 14)); getContentPane().add(label\_7);

cmbCalidad = **new** JComboBox<>(**new** String[]{"Económico", "Standard", "VIP"}); cmbCalidad.setFont(**new** Font("Lucida Calligraphy", Font.***PLAIN***, 10)); getContentPane().add(cmbCalidad);

btnReiniciar = **new** JButton("Reiniciar Datos"); btnReiniciar.setFont(**new** Font("Lucida Calligraphy", Font.***PLAIN***, 10)); btnResumen = **new** JButton("Mostrar Resumen");

btnResumen.setFont(**new** Font("Lucida Calligraphy", Font.***PLAIN***, 10)); getContentPane().add(btnReiniciar); getContentPane().add(btnResumen);

btnReiniciar.addActionListener(**new** ActionListener() { @Override

**public void** actionPerformed(ActionEvent e) { txtNombre.setText(""); txtDocumento.setText(""); txtFechaViaje.setText(""); grupoPiso.clearSelection(); chkAudifonos.setSelected(**false**); chkManta.setSelected(**false**); chkRevistas.setSelected(**false**); cmbOrigen.setSelectedIndex(0); cmbDestino.setSelectedIndex(0); cmbCalidad.setSelectedIndex(0);

JOptionPane.*showMessageDialog*(**null**, "Datos Reiniciados", "Reiniciar", JOptionPane.***INFORMATION\_MESSAGE***);

}

});

btnResumen.addActionListener(**new** ActionListener() { @Override

**public void** actionPerformed(ActionEvent e) { String nombre = txtNombre.getText(); String documento = txtDocumento.getText();

String fechaViaje = txtFechaViaje.getText();

String piso = rdbPrimero.isSelected() ? "1er Piso" : (rdbSegundo.isSelected() ? "2do Piso"

: "No Seleccionado");

String servicios = "";

**if** (chkAudifonos.isSelected()) servicios += "Audífonos, ";

**if** (chkManta.isSelected()) servicios += "Manta, ";

**if** (chkRevistas.isSelected()) servicios += "Revistas, ";

**if** (servicios.isEmpty()) servicios = "Ninguno";

**else** servicios = servicios.substring(0, servicios.length() - 2);

String origen = (String) cmbOrigen.getSelectedItem(); String destino = (String) cmbDestino.getSelectedItem(); String calidad = (String) cmbCalidad.getSelectedItem();

}

});

String mensaje = "Resumen de compras:\n"

+ "Nombre: " + nombre + "\n"

+ "Documento: " + documento + "\n"

+ "Fecha de Viaje: " + fechaViaje + "\n"

+ "Piso: " + piso + "\n"

+ "Servicios Opcionales: " + servicios + "\n"

+ "Origen: " + origen + "\n"

+ "Destino: " + destino + "\n"

+ "Calidad de Servicio: " + calidad;

JOptionPane.*showMessageDialog*(**null**, mensaje, "Resumen", JOptionPane.***INFORMATION\_MESSAGE***);

setVisible(**true**);

}

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

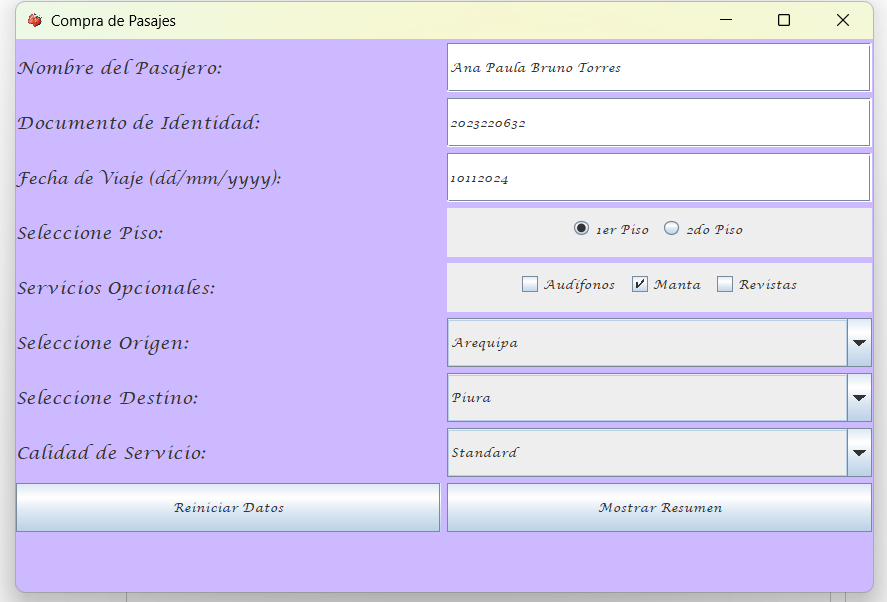
// Creating an instance of the window and making it visible

**new** EJEMPLO();

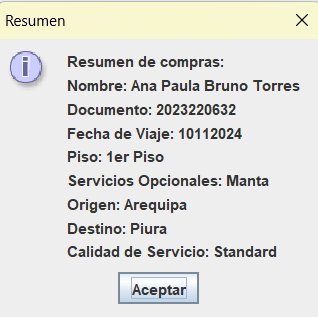
}

}

### POR PANTALLA



*Ilustración 14: Ventana de Compra de pasajes*



*Ilustración 15: Ventana de Mostrar Resumen*

### EXPLICACION

Este programa usa componentes de Swing como JTextField, JCheckBox, JRadioButton, JComboBox, y JButton organizados en un diseño de GridLayout. Los usuarios ingresan datos personales, seleccionan opciones de viaje (piso, origen, destino, calidad de servicio) y servicios opcionales. Dos botones permiten reiniciar los datos o mostrar un resumen de la información ingresada, utilizando ActionListener para manejar eventos. Además, el diseño incluye elementos visuales personalizados como fuentes, colores, y un ícono de ventana.

Además de eso tenemos a el botón "Mostrar Resumen" que genera un resumen personalizado con todos los datos del pasajero y las selecciones realizadas, como servicios adicionales, destino y calidad del servicio, mostrando el resultado en un cuadro de diálogo (JOptionPane).

# CONCLUSIONES DE LA PRÁCTICA:

* La práctica ha permitido entender la importancia de las interfaces gráficas en la interacción del usuario con las aplicaciones. Se ha aprendido que una GUI proporciona una apariencia visual que facilita la interacción con el software, haciendo que las aplicaciones sean más amigables y accesibles.
* La práctica ha resaltado la importancia de la interacción del usuario a través de cuadros de diálogo (como JOptionPane) para obtener información o mostrar mensajes. Esto mejora la experiencia del usuario al proporcionar un medio simple y directo para comunicarse con la aplicación.
* Se ha explorado el uso de componentes de Swing, como JFrame, JLabel, JButton, JTextField, y JPasswordField. Estos componentes son fundamentales para construir interfaces efectivas y permiten a los desarrolladores crear aplicaciones robustas y visualmente atractivas.

# BIBLIOGRAFÍA

Deitel, P., & Deitel, H. (2012). Cómo Programar en Java (9ª ed.). Prentice Hall.

Guevara, K., Montesinos, Á., Santillana, M., Iquira, D., Galarza, M., & Villaverde, N. (2024). Guía de Laboratorio N°09: Java Swing Básico. Universidad Católica de Santa María.

# ANEXO: ADJUNTAR ARCHIVO

GITHUB

<https://github.com/Ana11245/Repositorio_UCSM/tree/main/LP3/GUIA09>