FACULTAD DE CIENCIAS PURAS Y NATURALES

CARRERA INFORMÁTICA

GRUPO 4

TEMA: SISTEMA DE GESTION DE RESTAURANTE

Materia: INF-121

Integrantes:
1.Jhonny Duran Paco
2.Noel Mena Condori
3.Lenin Quispe Mamani
4.Brandon Apaza Amaru
5.Boris Miranda Mamani
6. Franklin Ramos Acarapi

Docente: Lic. Rosalía López M.
Fecha: 28 de diciembre de 2025

Introducción

Este documento presenta el desarrollo de un Sistema de Gestión de Restaurante basado en los principios de la Programación Orientada a Objetos (POO). A través del uso de diagramas UML, se modelan las principales entidades del sistema, incluyendo Clientes, Empleados, Pedidos y Platos, aplicando conceptos como herencia, polimorfismo, agregación, composición y genericidad para una estructura flexible y escalable.

El modelo permite gestionar de manera eficiente los pedidos, calcular precios y administrar empleados, optimizando el funcionamiento del restaurante. Además, se establecen relaciones entre las clases para mejorar la organización y facilitar la implementación del sistema en un entorno de desarrollo.

1. Definición del Proyecto

El Sistema de Gestión de Restaurante es una aplicación diseñada para administrar de manera eficiente los procesos dentro de un restaurante, incluyendo la gestión de pedidos, el control de empleados y la organización del menú. Este sistema permite a los clientes realizar pedidos de diferentes tipos de platos, mientras que los empleados pueden gestionar y registrar las órdenes de manera estructurada. Para ello, se ha aplicado el enfoque de Programación Orientada a Objetos (POO), utilizando herencia, polimorfismo, abstracción y composición para garantizar un diseño modular y reutilizable.

El sistema se ha modelado utilizando diagramas UML, permitiendo visualizar la relación entre las distintas entidades, como `Cliente`, `Empleado`, `Pedido` y `Plato`. La implementación se ha desarrollado en Java, integrando una interfaz gráfica para facilitar la interacción con el usuario. Además, el uso de listas genéricas y estructuras dinámicas mejora la escalabilidad del sistema, haciendo que sea adaptable a distintos tamaños de restaurantes.

1.1 Descripción General

El proyecto consiste en un Sistema de Gestión de Restaurante, que permite administrar eficientemente los procesos internos de un establecimiento gastronómico. Este sistema facilita la gestión de pedidos, el registro de clientes y empleados, y la organización de los diferentes platos del menú. Además, busca optimizar el flujo de trabajo en el restaurante, permitiendo que los empleados registren pedidos de manera estructurada y calculen el costo total de las órdenes de forma automatizada.

El sistema también cuenta con funcionalidades para categorizar los platos en diferentes tipos, como platos principales, postres y bebidas, asegurando una mejor organización del menú. Asimismo, se integra una interfaz gráfica que facilita la interacción con los usuarios, permitiendo un acceso intuitivo a las opciones de gestión. Gracias al uso de Programación Orientada a Objetos (POO) y la aplicación de diagramas UML, el diseño del sistema es modular, escalable y adaptable a las necesidades del restaurante.

1.2 Objetivos

Aplicar los conceptos fundamentales de POO en el desarrollo del Sistema de Gestión de Restaurante, utilizando principios como herencia, polimorfismo, encapsulamiento y abstracción para estructurar el código de manera eficiente y reutilizable.

Diseñar un sistema modular y escalable, organizando las clases de forma que cada una cumpla con una responsabilidad específica. Se implementan clases como Cliente, Empleado, Pedido y Plato, siguiendo un modelo basado en diagramas UML, lo que facilita futuras mejoras y adaptaciones.

Implementar patrones de diseño para mejorar la flexibilidad y mantenibilidad del código. Se busca optimizar la interacción entre las diferentes clases y módulos, asegurando que el sistema sea fácil de extender y mantener. Además, se integrará una interfaz gráfica que permita una gestión intuitiva de pedidos y empleados dentro del restaurante.

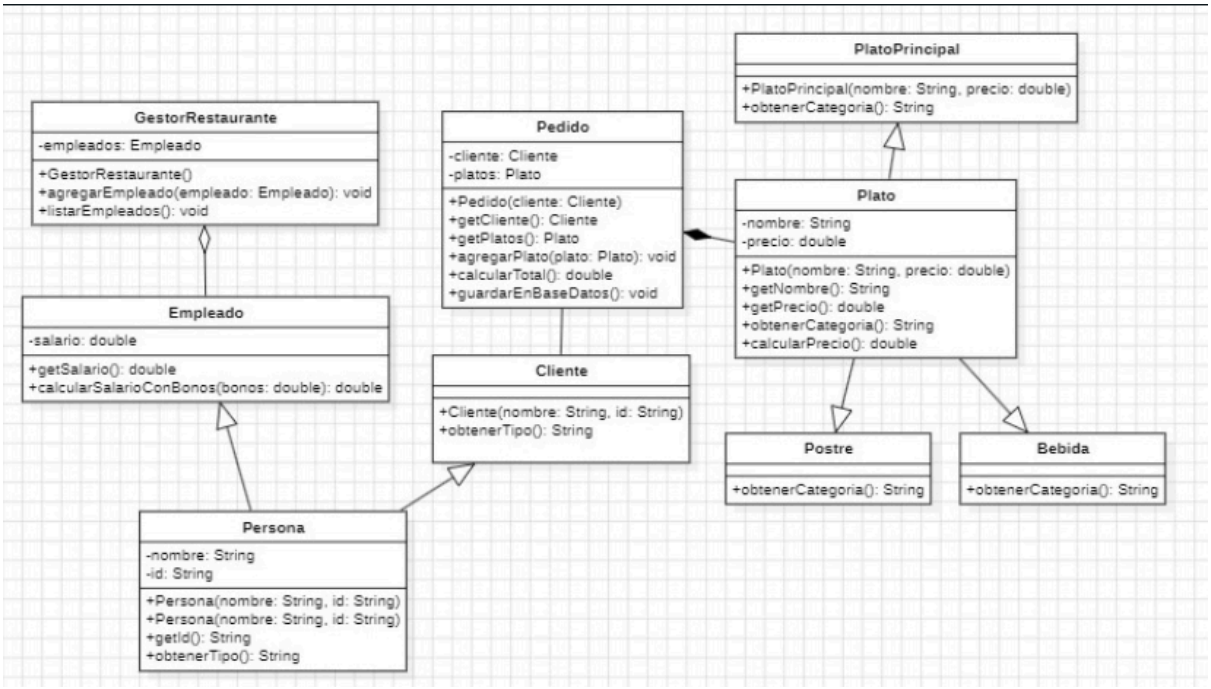
2. Análisis y Diseño

El Sistema de Gestión de Restaurante se diseñó utilizando los principios de la Programación Orientada a Objetos (POO) y haciendo uso de patrones de diseño para garantizar un sistema modular, escalable y mantenible. Durante la etapa de análisis, se identificaron las principales entidades del sistema: Clientes, Empleados, Pedidos y Platos, cada una con sus atributos y métodos específicos. Estas entidades se estructuraron a través de un diagrama UML, que permitió modelar las relaciones entre clases y establecer su responsabilidad dentro del sistema, asegurando una arquitectura clara y funcional.

En la etapa de diseño, se implementaron patrones de diseño, para garantizar una única instancia del `GestorRestaurante`; Decorator, para añadir características adicionales a los platos de manera dinámica; y Observer, para notificar automáticamente los cambios en los pedidos. Estas soluciones técnicas mejoran la flexibilidad del código y permiten que el sistema sea fácilmente adaptable a nuevas funcionalidades, como agregar nuevos tipos de platos o gestionar inventarios en tiempo real. Además, la integración de una interfaz gráfica facilita la interacción entre los usuarios y el sistema, mejorando la experiencia de uso y la gestión de operaciones del restaurante.

2.1 Diagrama UML

El sistema incluirá los siguientes diagramas UML:



2.2 Principios de Diseño

Herencia

La herencia se aplica en la jerarquía de clases, como Plato siendo una clase abstracta de la cual heredan PlatoPrincipal, Postre y Bebida. Esto permite reutilizar atributos comunes como nombre y precio, mientras cada subclase puede implementar comportamientos específicos, como definir la categoría del plato.

Composición

La clase Pedido incluye una colección de objetos de tipo Plato, lo que permite asociar múltiples platos a un pedido sin perder la modularidad. Esto facilita el cálculo del costo total y el manejo de los platos en cada orden.

Agregación

GestorRestaurante se encarga de gestionar una lista de empleados, utilizando agregación para asociar objetos de tipo Empleado. Esta relación permite que los empleados existan independientemente del gestor, manteniendo una conexión flexible.

Genericidad

Se hace uso de colecciones genéricas (List) para manejar listas dinámicas de objetos, como empleados y platos. Esto mejora la escalabilidad del sistema, ya que permite agregar y gestionar nuevas instancias sin restricciones específicas.

Interfaces

En el proyecto se incluyen formularios (interfaz.java e InterRestaurante.java) que probablemente actúan como interfaces gráficas para las operaciones CRUD (Crear, Leer, Actualizar y Eliminar). Estas interfaces facilitan la interacción del usuario con el sistema, manteniendo una capa separada para la lógica de negocio.

Patrones de diseño

- **Singleton**: Aplicado en GestorRestaurante para garantizar una única instancia que centralice la administración de empleados.
- **Decorator**: Útil para añadir características adicionales a los platos (como personalización de ingredientes o descuentos).
- **Observer**: Recomendado para notificar cambios en los pedidos, asegurando que otras partes del sistema respondan automáticamente a estos cambios.

3. Implementación en Java

3.1 Estructura del Proyecto

El proyecto está organizado en tres paquetes principales para una mejor separación de responsabilidades:

- modelo: Contiene las clases que representan el dominio del sistema, como `Persona`, `Plato`, `Cliente`, `Empleado` y `Pedido`. Estas clases definen la estructura de los datos y los métodos específicos de cada entidad.
- controlador: Implementa la lógica de negocio y el programa principal. La clase `GestorRestaurante` centraliza la gestión del sistema, mientras que otras clases manejan tareas específicas.
- vista: Incluye las interfaces gráficas (`InterRestaurante` e `interfaz`), facilitando la interacción entre el usuario y el sistema.

3.2 Código Fuente

```
class Pedido {

    private Cliente cliente;
    private Plato[] platos;
    private int cantidadPlatos;

    public Pedido(Cliente cliente, int
maxPlatos) {
        this.cliente = cliente;
        this.platos = new Plato[maxPlatos];
        this.cantidadPlatos = 0;
    }

    public Cliente getCliente() {
        return cliente;
    }

    public Plato[] getPlatos() {
        return platos;
    }
```

```
abstract class Plato {

    private int idPlato;
    private String nombre;
    private double precio;

    public Plato(String nombre, double
precio) {
        this.nombre = nombre;
        this.precio = precio;
    }

    public String getNombre() {
        return nombre;
    }

    public double getPrecio() {
        return precio;
    }
```

```
    public void agregarPlato(Plato plato) {
        if (cantidadPlatos < platos.length) {
            platos[cantidadPlatos++] = plato;
        } else {
            System.out.println("No se pueden
agregar más platos a este pedido.");
        }
    }

    public double calcularTotal() {
        double total = 0;
        for (int i = 0; i < cantidadPlatos; i++) {
            total += platos[i].getPrecio();
        }
        return total;
    }
}
```

```
    public int getIdPlato() {
        return idPlato;
    }

    public abstract String obtenerCategoria();
}
```

```
class Cliente extends Persona {
    public Cliente(String nombre, String id) {
        super(nombre, id);
    }

    @Override
    public String obtenerTipo() {
        return "Cliente";
    }
}
```

```
class Empleado extends Persona {
    private double salario;

    public Empleado(String nombre, String
id, double salario) {
        super(nombre, id);
        this.salario = salario;
    }

    public double getSalario() {
        return salario;
    }

    @Override
    public String obtenerTipo() {
        return "Empleado";
    }

    public double
calcularSalarioConBonos(double bonos) {
        return salario + bonos;
    }
}
```

```
class Postre extends Plato {
    public Postre(String nombre, double
precio) {
        super(nombre, precio);
    }

    @Override
    public String obtenerCategoria() {
```

```
class Bebida extends Plato {
    public Bebida(String nombre, double
precio) {
        super(nombre, precio);
    }

    @Override
    public String obtenerCategoria() {
```

```
        return "Postre";
    }
}
```

```
        return "Bebida";
    }
}
```

```
class PlatoPrincipal extends Plato {
    public PlatoPrincipal(String nombre,
double precio) {
        super(nombre, precio);
    }

    @Override
    public String obtenerCategoria() {
        return "Plato Principal";
    }
}
```

```
public class GestorRestaurante {
    private static final String URL =
"jdbc:mysql://sql10.freesqldatabase.com:3306/sql10759871";
    private static final String USER = "sql10759871";
    private static final String PASSWORD = "DVC3dgtF3E";

    public static Connection getConnection() throws SQLException {
        return DriverManager.getConnection(URL, USER, PASSWORD);

    }
    public void MostrarDetallesPedido() {
        String query = "SELECT * FROM pedidos"; // Asegúrate de que el nombre de la tabla
sea correcto
        try (Connection conexion = getConnection();
             Statement statement = conexion.createStatement();
             ResultSet rs = statement.executeQuery(query)) {

            System.out.println("Detalles de los pedidos:");
            while (rs.next()) {
                String id = rs.getString("id");
                String clienteId = rs.getString("cliente_id");
                double total = rs.getDouble("total");

                System.out.println("ID Pedido: " + id);
                System.out.println("ID Cliente: " + clienteId);
                System.out.println("Total: " + total);
                System.out.println("--------------------------");
            }

        } catch (SQLException e) {
            System.out.println("Error al mostrar los detalles de los pedidos: " +
e.getMessage());
        }
    }

    private Connection conexion;
```

```
// Constructor: Establece la conexión a la base de datos
public GestorRestaurante() {
    try {
        conexion = DriverManager.getConnection(URL, USER, PASSWORD);
        System.out.println("Conexión exitosa con la base de datos");
    } catch (SQLException e) {
        System.err.println("Error al conectar con la base de datos: " + e.getMessage());
    }
}

// Método para cerrar la conexión
public void cerrarConexion() {
    try {
        if (conexion != null) {
            conexion.close();
            System.out.println("Conexión cerrada correctamente");
        }
    } catch (SQLException e) {
        System.err.println("Error al cerrar la conexión: " + e.getMessage());
    }
}

// Función para registrar un cliente
public void registrarCliente(String nombre, String telefono) {
    String sql = "INSERT INTO clientes (nombre, telefono) VALUES (?, ?)";
    try (PreparedStatement stmt = conexion.prepareStatement(sql)) {
        stmt.setString(1, nombre);
        stmt.setString(2, telefono);
        stmt.executeUpdate();
        System.out.println("Cliente registrado exitosamente");
    } catch (SQLException e) {
        System.err.println("Error al registrar cliente: " + e.getMessage());
    }
}

// Función para mostrar todos los clientes
public void mostrarClientes() {
    String sql = "SELECT * FROM clientes";
    try (Statement stmt = conexion.createStatement();
         ResultSet rs = stmt.executeQuery(sql)) {
        while (rs.next()) {
            System.out.println("ID: " + rs.getInt("id_cliente") +
                    ", Nombre: " + rs.getString("nombre") +
                    ", Teléfono: " + rs.getString("telefono"));
        }
    } catch (SQLException e) {
        System.err.println("Error al mostrar clientes: " + e.getMessage());
    }
}

// Función para registrar un plato
public void registrarPlato(String nombre, double precio, String categoria) {
    String sql = "INSERT INTO platos (nombre, precio, categoria) VALUES (?, ?, ?)";
```

```
        try (PreparedStatement stmt = conexion.prepareStatement(sql)) {
            stmt.setString(1, nombre);
            stmt.setDouble(2, precio);
            stmt.setString(3, categoria);
            stmt.executeUpdate();
            System.out.println("Plato registrado exitosamente");
        } catch (SQLException e) {
            System.err.println("Error al registrar plato: " + e.getMessage());
        }
    }

    // Función para mostrar todos los platos
    public void mostrarPlatos() {
        String sql = "SELECT * FROM platos";
        try (Statement stmt = conexion.createStatement();
             ResultSet rs = stmt.executeQuery(sql)) {
            while (rs.next()) {
                System.out.println("ID: " + rs.getInt("id_plato") +
                        ", Nombre: " + rs.getString("nombre") +
                        ", Precio: " + rs.getDouble("precio") +
                        ", Categoría: " + rs.getString("categoria"));
            }
        } catch (SQLException e) {
            System.err.println("Error al mostrar platos: " + e.getMessage());
        }
    }

    // Función para registrar un pedido
    public void registrarPedido(int idCliente, Plato[] platos, int[] cantidades) {
        String sqlPedido = "INSERT INTO pedidos (id_cliente, total) VALUES (?, ?)";
        String sqlDetalle = "INSERT INTO detalle_pedido (id_pedido, id_plato, cantidad)
VALUES (?, ?, ?)";

        try (PreparedStatement stmtPedido = conexion.prepareStatement(sqlPedido,
Statement.RETURN_GENERATED_KEYS)) {
            // Calcular el total
            double total = 0;
            for (int i = 0; i < platos.length; i++) {
                total += platos[i].getPrecio() * cantidades[i];
            }

            // Registrar el pedido
            stmtPedido.setInt(1, idCliente);
            stmtPedido.setDouble(2, total);
            stmtPedido.executeUpdate();

            // Obtener el ID del pedido generado
            ResultSet rs = stmtPedido.getGeneratedKeys();
            if (rs.next()) {
                int idPedido = rs.getInt(1);

                // Registrar el detalle del pedido
                try (PreparedStatement stmtDetalle = conexion.prepareStatement(sqlDetalle)) {
                    for (int i = 0; i < platos.length; i++) {
```

```
                    stmtDetalle.setInt(1, idPedido);
                    stmtDetalle.setInt(2, platos[i].getIdPlato());
                    stmtDetalle.setInt(3, cantidades[i]);
                    stmtDetalle.executeUpdate();
                }
            }
        }
        System.out.println("Pedido registrado exitosamente");
    } catch (SQLException e) {
        System.err.println("Error al registrar pedido: " + e.getMessage());
    }
}

// Función para mostrar todos los pedidos
public void mostrarPedidos() {
    String sql = "SELECT p.id_pedido, c.nombre AS cliente, p.total " +
                 "FROM pedidos p JOIN clientes c ON p.id_cliente = c.id_cliente";
    try (Statement stmt = conexion.createStatement();
         ResultSet rs = stmt.executeQuery(sql)) {
        while (rs.next()) {
            System.out.println("ID Pedido: " + rs.getInt("id_pedido") +
                    ", Cliente: " + rs.getString("cliente") +
                    ", Total: " + rs.getDouble("total"));
        }
    } catch (SQLException e) {
        System.err.println("Error al mostrar pedidos: " + e.getMessage());
    }
}
}
```

CODIGO INTERFAZ

```
public class interfaz extends javax.swing.JFrame {

    private Connection connection;

    /**
     * Creates new form interfaz
     */
    public interfaz() {
        this.connection = connection;
        initUI();
    }

    private void initUI() {
        JFrame frame = new JFrame("Gestión de Restaurante");
        frame.setSize(700, 500);

frame.setDefaultCloseOperation(JFrame.E
```

```
        /* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
         * For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
         */
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {

javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
```

```
XIT_ON_CLOSE);
        frame.setLayout(new BorderLayout());

        // Crear barra de menú
        JMenuBar menuBar = new
JMenuBar();

        // Menú Cliente
        JMenu menuCliente = new
JMenu("Cliente");
        JMenuItem
itemCambiarNombreCliente = new
JMenuItem("Cambiar Nombre de Cliente");
        JMenuItem itemVerClientes = new
JMenuItem("Ver Clientes");
        JMenuItem itemEliminarCliente = new
JMenuItem("Eliminar Cliente");
        JMenuItem itemBuscarCliente = new
JMenuItem("Buscar Cliente");
        JMenuItem itemSalir = new
JMenuItem("Salir");


itemCambiarNombreCliente.addActionListe
ner(e -> cambiarNombreCliente());
        itemVerClientes.addActionListener(e
-> verClientes());

itemEliminarCliente.addActionListener(e ->
eliminarCliente());
        itemBuscarCliente.addActionListener(e
-> buscarCliente());
        itemSalir.addActionListener(e ->
System.exit(0));


menuCliente.add(itemCambiarNombreClien
te);
        menuCliente.add(itemVerClientes);
        menuCliente.add(itemEliminarCliente);
        menuCliente.add(itemBuscarCliente);
        menuCliente.addSeparator();
        menuCliente.add(itemSalir);

      // Menú Empleado
        JMenu menuEmpleado = new
JMenu("Empleado");
        JMenuItem
itemCambiarNombreEmpleado = new
JMenuItem("Cambiar Nombre de
Empleado");
        JMenuItem itemVerEmpleados = new
JMenuItem("Ver Empleados");
        JMenuItem itemEliminarEmpleado =
```

```
            }
        }
    } catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(interfaz.c
lass

.getName()).log(java.util.logging.Level.SEV
ERE, null, ex);

    } catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(interfaz.c
lass

.getName()).log(java.util.logging.Level.SEV
ERE, null, ex);

    } catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(interfaz.c
lass

.getName()).log(java.util.logging.Level.SEV
ERE, null, ex);

    } catch
(javax.swing.UnsupportedLookAndFeelExc
eption ex) {


private void verClientes() {
    Connection conn = null;
    try {
        // Conexión a la base de datos
        conn =
GestorRestaurante.getConnection();

        // Consulta SQL para obtener los
datos de los clientes
        String query = "SELECT * FROM
clientes";
        Statement stmt =
conn.createStatement();
        ResultSet rs =
stmt.executeQuery(query);

        // Crear un JTable con modelo por
defecto
        JTable table = new JTable();
        DefaultTableModel model = new
DefaultTableModel();
```

```
new JMenuItem("Eliminar Empleado");
        JMenuItem itemBuscarEmpleado =
new JMenuItem("Buscar Empleado");


itemCambiarNombreEmpleado.addActionLi
stener(e -> cambiarNombreEmpleado());

itemVerEmpleados.addActionListener(e ->
verEmpleados());

itemEliminarEmpleado.addActionListener(e
-> eliminarEmpleado());

itemBuscarEmpleado.addActionListener(e
-> buscarEmpleado());


menuEmpleado.add(itemCambiarNombreE
mpleado);

menuEmpleado.add(itemVerEmpleados);

menuEmpleado.add(itemEliminarEmpleado
);

menuEmpleado.add(itemBuscarEmpleado);

        // Añadir menús a la barra de menú
        menuBar.add(menuCliente);
        menuBar.add(menuEmpleado);

        // Panel central con botones
        JPanel centralPanel = new JPanel();
        centralPanel.setLayout(new
GridLayout(5, 2, 10, 10)); // 5 filas, 2
columnas con espaciado

        JButton btnCuentaTotalMesa = new
JButton("Cuenta Total de Mesa");
        JButton btnAgregarDescuento = new
JButton("Agregar Descuento a Cuenta");
        JButton btnListarPlatosVendidos =
new JButton("Listar Platos Más Vendidos");
        JButton btnListarMesasDisponibles =
new JButton("Listar Mesas Disponibles");


btnCuentaTotalMesa.addActionListener(e
-> calcularCuentaMesaX());

btnAgregarDescuento.addActionListener(e
-> agregarDescuento());
```

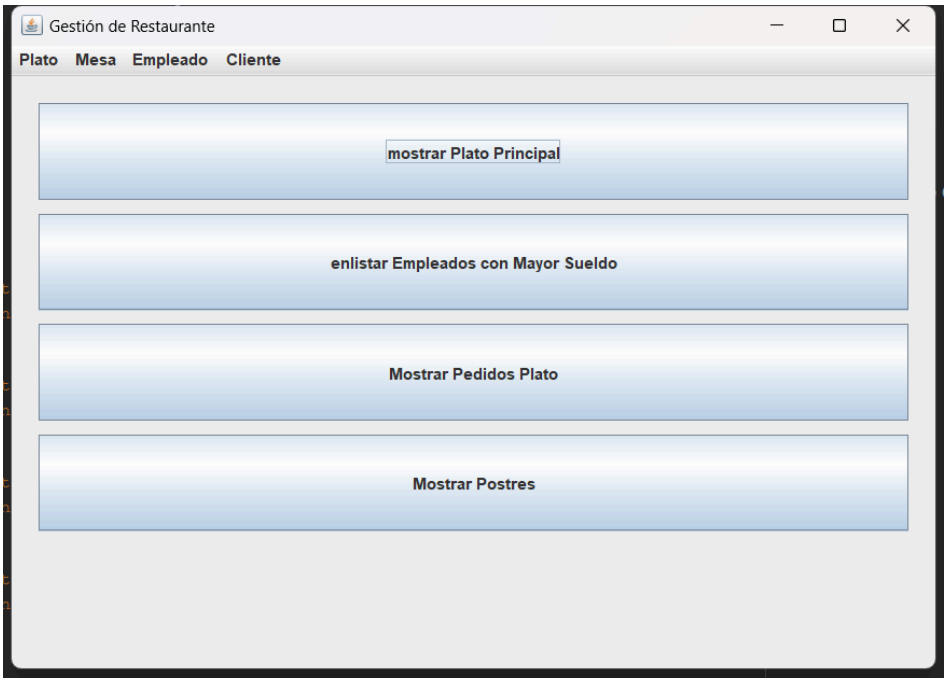
```
            // Añadir las columnas al modelo
            model.addColumn("ID");
            model.addColumn("Nombre");

            // Rellenar las filas del modelo con
los datos del ResultSet
            while (rs.next()) {
                model.addRow(new Object[]{
                    rs.getInt("id"), // ID del cliente
                    rs.getString("nombre") //
Nombre del cliente
                });
            }

            // Establecer el modelo al JTable
            table.setModel(model);

            // Mostrar el JTable dentro de un
JOptionPane con scroll
            JOptionPane.showMessageDialog(
                    null,
                    new JScrollPane(table),
                    "Clientes",

JOptionPane.INFORMATION_MESSAGE
            );
        } catch (SQLException e) {
            // Mostrar error si ocurre algún
problema con la base de datos
            JOptionPane.showMessageDialog(
                    null,
                    "Error al consultar clientes: " +
e.getMessage(),
                    "Error",

JOptionPane.ERROR_MESSAGE
            );
        } finally {
            try {
                if (conn != null) {
                    conn.close(); // Cerrar la
conexión
                }
            } catch (SQLException e) {

JOptionPane.showMessageDialog(
                        null,
                        "Error al cerrar la conexión: "
+ e.getMessage(),
                        "Error",

JOptionPane.ERROR_MESSAGE
                );
            }
```

```
btnListarPlatosVendidos.addActionListener(
e -> listarPlatosMasVendidos());

btnListarMesasDisponibles.addActionListen
er(e -> listarMesasDisponibles());


centralPanel.add(btnCuentaTotalMesa);

centralPanel.add(btnAgregarDescuento);

centralPanel.add(btnListarPlatosVendidos);

centralPanel.add(btnListarMesasDisponible
s);

        // Contenedor principal
        JPanel container = new JPanel(new
BorderLayout());

container.setBorder(BorderFactory.createE
mptyBorder(20, 20, 20, 20));
        container.add(centralPanel,
BorderLayout.CENTER);

        frame.add(container,
BorderLayout.CENTER);
        frame.setJMenuBar(menuBar);
        frame.setVisible(true);
    }
/**
     * This method is called from within the
constructor to initialize the form.
     * WARNING: Do NOT modify this code.
The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed"
desc="Generated Code">
    private void initComponents() {


setDefaultCloseOperation(javax.swing.Wind
owConstants.EXIT_ON_CLOSE);

        javax.swing.GroupLayout layout = new
javax.swing.GroupLayout(getContentPane()
);
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.Gro
upLayout.Alignment.LEADING)
        }
    }

    private void verRestaurante() {

    }

    private void buscarCliente() {
        throw new
UnsupportedOperationException("Not
supported yet.");
    }

    private void eliminarCliente() {
        throw new
UnsupportedOperationException("Not
supported yet.");
    }

    private void agregarDescuento() {
        throw new
UnsupportedOperationException("Not
supported yet.");
    }

    private void listarMesasDisponibles() {
        throw new
UnsupportedOperationException("Not
supported yet.");
    }

    private void calcularCuentaMesaX() {
        throw new
UnsupportedOperationException("Not
supported yet.");
    }

    private void listarPlatosMasVendidos() {
        throw new
UnsupportedOperationException("Not
supported yet.");
    }

    private void cambiarNombreCliente() {
        String idCliente =
JOptionPane.showInputDialog("Ingrese el
ID del Cliente:");
        String nuevoNombre =
JOptionPane.showInputDialog("Ingrese el
nuevo nombre:");

        String query = "UPDATE Cliente SET
nombre = ? WHERE id_cliente = ?";
        try (PreparedStatement stmt =
```

```
                    .addGap(0, 333,
Short.MAX_VALUE)
        );
        layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.Gro
upLayout.Alignment.LEADING)
                    .addGap(0, 174,
Short.MAX_VALUE)
        );

        pack();
    }// </editor-fold>

    private void
RegistrarPedidoActionPerformed(java.awt.e
vent.ActionEvent evt) {
        //id autoincremental, cliente_id, total
        String query = "INSERT INTO pedidos
(cliente_id, total) V LUES (?, ?)";
        try (Connection conn =
GestorRestaurante.getConnection();
PreparedStatement stmt =
conn.prepareStatement(query)) {
            stmt.setInt(1, 1); // Assuming the
client ID is 1
            stmt.setDouble(2, 0.0); // Assuming
the total is 0.0
            stmt.executeUpdate();

JOptionPane.showMessageDialog(null,
"Pedido registrado", "Éxito",
JOptionPane.INFORMATION_MESSAGE);
        } catch (SQLException e) {

JOptionPane.showMessageDialog(null,
"Error registrando pedido: " +
e.getMessage(), "Error",
JOptionPane.ERROR_MESSAGE);
        }
    }

    /**
     * @param args the command line
arguments
     */
    public static void main(String args[]) {
        /* Set the Nimbus look and feel */
        //<editor-fold defaultstate="collapsed"
desc=" Look and feel setting code (optional)
">
```

```
connection.prepareStatement(query)) {
            stmt.setString(1, nuevoNombre);
            stmt.setString(2, idCliente);
            int rowsAffected =
stmt.executeUpdate();
            if (rowsAffected > 0) {

JOptionPane.showMessageDialog(null,
"Nombre actualizado correctamente.");
            } else {

JOptionPane.showMessageDialog(null,
"Cliente no encontrado.");
            }
        } catch (SQLException e) {

JOptionPane.showMessageDialog(null,
"Error al actualizar el nombre: " +
e.getMessage());
        }
    }

    private void cambiarNombreEmpleado() {
        throw new
UnsupportedOperationException("Not
supported yet.");
    }

    private void verEmpleados() {
        throw new
UnsupportedOperationException("Not
supported yet.");
    }

    private void buscarEmpleado() {
        throw new
UnsupportedOperationException("Not
supported yet.");
    }

    private void eliminarEmpleado() {
        throw new
UnsupportedOperationException("Not
supported yet.");
    }
}
```

3.3 Diseño de Interfaces



3.4 Manejo de Archivos

El manejo de archivos y la persistencia de datos son aspectos fundamentales en el desarrollo de sistemas de software modernos. Estos conceptos permiten que los datos generados por un sistema se almacenen de forma permanente y puedan recuperarse posteriormente, incluso después de que el programa haya terminado su ejecución. En el contexto de este proyecto, que consiste en un sistema de gestión para restaurantes, se ha

implementado la persistencia de datos utilizando MySQL, un sistema de gestión de bases de datos relacional ampliamente reconocido por su eficiencia y escalabilidad.

Este informe detalla el enfoque técnico y teórico utilizado para integrar la persistencia de datos mediante MySQL en un proyecto basado en Java, haciendo uso de la tecnología JDBC (Java Database Connectivity). Además, se destacan los principios, beneficios y mejores prácticas que aseguran un manejo de archivos y datos seguro y eficiente.

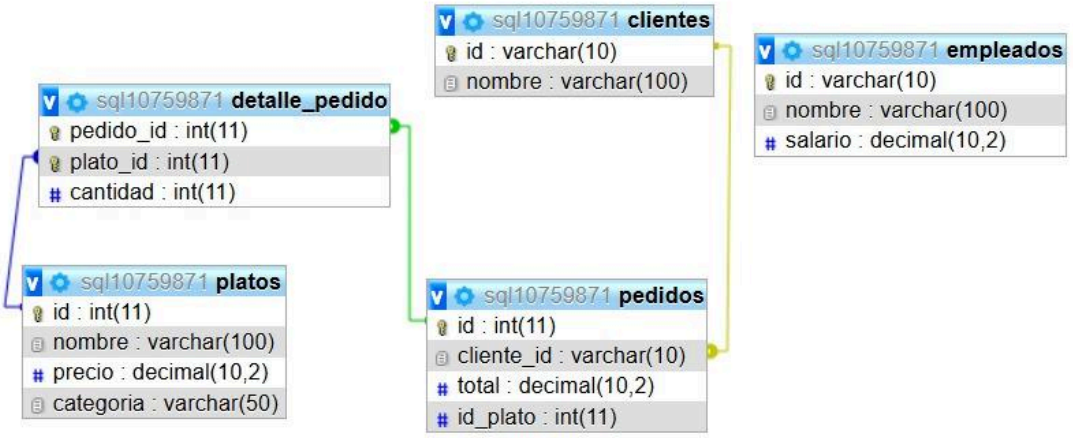
MySQL: Elección para la Persistencia de Datos

MySQL es un sistema de gestión de bases de datos relacional (RDBMS) que se utiliza para almacenar y organizar datos en tablas relacionadas mediante claves primarias y foráneas. Fue elegido para este proyecto debido a sus siguientes características:

- **Eficiencia**: Diseñado para manejar grandes volúmenes de datos con alto rendimiento.
- **Soporte Transaccional**: Compatible con el modelo ACID (Atomicidad, Consistencia, Aislamiento, Durabilidad), garantizando la integridad de las operaciones.
- **Compatibilidad con JDBC**: Se integra fácilmente con Java mediante el conector de MySQL.
- **Seguridad**: Ofrece autenticación de usuarios y control de acceso para proteger los datos.

Estructura de la Base de Datos

Para modelar las entidades del sistema de gestión de restaurante, se diseñaron las siguientes tablas:



Estas tablas están relacionadas mediante claves foráneas, lo que garantiza la integridad referencial entre los datos.

Implementación de la Conexión con MySQL en Java

La conexión entre el sistema y la base de datos MySQL se realiza mediante **JDBC**, una API que permite ejecutar consultas SQL desde una aplicación Java. El proceso general incluye:

1. **Registrar el controlador JDBC**: Se utiliza el conector MySQL para establecer la comunicación entre Java y la base de datos.
2. **Establecer la conexión**: Se crea un objeto `Connection` que representa la conexión activa con la base de datos.
3. **Ejecutar consultas SQL**: Mediante objetos como `Statement` o `PreparedStatement`, se ejecutan operaciones como SELECT, INSERT, UPDATE y DELETE.
4. **Cerrar la conexión**: Para evitar fugas de recursos, es importante cerrar la conexión al finalizar su uso.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class ConexionBD {
    private static final String URL = "jdbc:mysql://localhost:3306/restaurante";
    private static final String USUARIO = "root";
    private static final String PASSWORD = "admin";

    public static Connection conectar() {
        try {
            return DriverManager.getConnection(URL, USUARIO, PASSWORD);
        } catch (SQLException e) {
            System.out.println("Error al conectar con la base de datos: " + e.getMessage());
            return null;
        }
    }
}
```

Operaciones CRUD con JDBC

Las operaciones CRUD (Create, Read, Update, Delete) son fundamentales para interactuar con la base de datos. Estas operaciones permiten gestionar la información almacenada de manera eficiente.

INSERTAR: DATOS
String sql = "INSERT INTO clientes (id, nombre, telefono) VALUES (?, ?, ?)";
PreparedStatement stmt = conexion.prepareStatement(sql);
stmt.setInt(1, cliente.getId());
stmt.setString(2, cliente.getNombre());
	CONSULTAR: DATOS
	String sql = "SELECT * FROM pedidos WHERE cliente_id = ?";
	PreparedStatement stmt = conexion.prepareStatement(sql);
	stmt.setInt(1, clienteId);
	ResultSet rs = stmt.executeQuery();

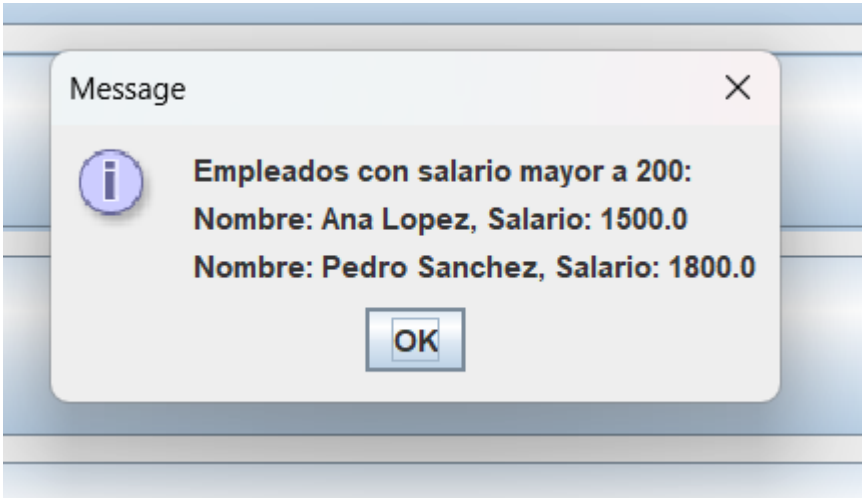
stmt.setString(3, cliente.getTelefono());
stmt.executeUpdate();
	while (rs.next()) {
	System.out.println("Pedido ID: " + rs.getInt("id"));
	System.out.println("Fecha: " + rs.getDate("fecha"));
	System.out.println("Total: " + rs.getDouble("total"));
	}

ACTUALIZAR: DATOS
String sql = "UPDATE clientes SET telefono = ? WHERE id = ?";
PreparedStatement stmt = conexion.prepareStatement(sql);
stmt.setString(1, nuevoTelefono);
stmt.setInt(2, clienteId);
stmt.executeUpdate();String sql = "UPDATE clientes SET telefono = ? WHERE id = ?";
PreparedStatement stmt = conexion.prepareStatement(sql);
stmt.setString(1, nuevoTelefono);
stmt.setInt(2, clienteId);
stmt.executeUpdate();
	ELIMINAR: DATOS
	String sql = "DELETE FROM platos WHERE id = ?";
	PreparedStatement stmt = conexion.prepareStatement(sql);
	stmt.setInt(1, platoId);
	stmt.executeUpdate();

Buenas Prácticas en el Manejo de Archivos y Persistencia

1. **Validaciones Previas**: Verificar los datos antes de enviarlos a la base de datos para evitar errores o inconsistencias.
2. **Consultas Parametrizadas**: Usar PreparedStatement para prevenir ataques de inyección SQL.
3. **Transacciones**: Implementar transacciones para garantizar que las operaciones críticas sean completas y consistentes.
4. **Cierre de Recursos**: Asegurar que las conexiones y otros recursos sean cerrados correctamente para evitar fugas.

4. Pruebas del Sistema



5. Conclusión

El desarrollo del Sistema de Gestión de Restaurante demuestra cómo la aplicación de los principios de la Programación Orientada a Objetos (POO) y el uso de patrones de diseño pueden contribuir significativamente a la creación de un sistema modular, escalable y eficiente. A través de la implementación de clases bien estructuradas, como Pedido, Cliente, Empleado y Plato, y el uso de patrones como Singleton, Decorator y Observer, se logró diseñar una solución que facilita la administración de las operaciones clave del restaurante, incluyendo la gestión de pedidos, empleados y menú.

El uso de diagramas UML permitió definir una arquitectura clara y comprensible, mientras que la integración de una interfaz gráfica mejoró la experiencia del usuario final, haciendo el sistema intuitivo y funcional. Además, la inclusión de conceptos como herencia, composición y agregación asegura que el código sea reutilizable y adaptable a futuros cambios o expansiones del sistema. En conclusión, este proyecto no solo cumple con los objetivos planteados, sino que también sirve como base para el desarrollo de sistemas más complejos en el ámbito de la gestión empresarial.