Proyecto Programación

Juan Felipe Mena

¡Vamos!

Contexto de mi programa.

Mi programa se basa en un gestor de restaurante, en el cual puedes añadir, actualizar o eliminar productos y proveedores según sus campos.

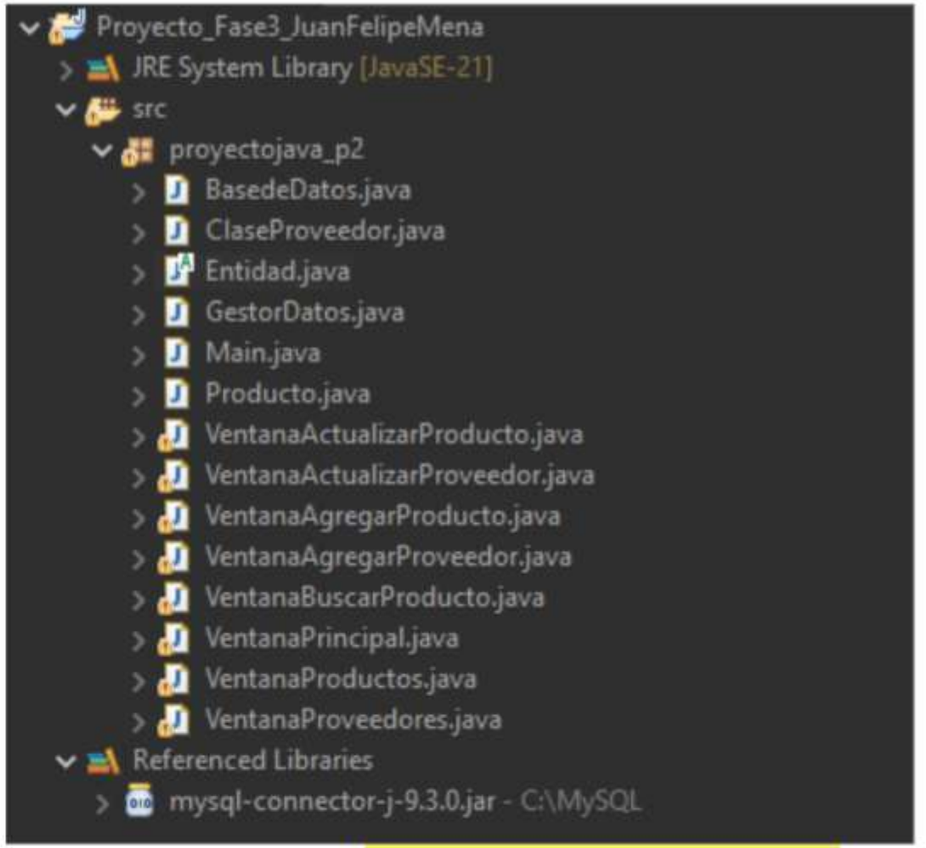
Continuar

Problemática a resolver

En las etapas anteriores, el sistema permitía la gestión básica de productos y proveedores. Sin embargo, habían limitaciones importantes.

Objetivo de esta parte:

Resolver estas limitaciones y añadir mejoras.



Continuar

MEJORAS

Base de Datos - Interfaz Gráfica - Nuevas clases - Excepciones - Herencia

Continuar

CÓDIGO

Entidad - GestorDatos - BasedeDatos - ClaseProveedor - Main - Producto - VentanaActualizarProducto - VentanaActualizarProveedor - VentanaAgregarProducto - VentanaAgregarProveedor - VentanaBuscarProducto - VentanaPrincipal - VentanaProductos - VentanaProveedores.

Continuar

Clase Entidad

Actúa como una clase base abstracta para las entidades Producto y Proveedor, es decir, define una estructura común que ambas comparten y sirve como plantilla para estas.

Define un método abstracto **mostrarInformacion()** que obliga a las subclases (Producto y Proveedor) a implementar su propia forma de mostrar información.

Utilizo protected para que las clases accedan directamente a estos campos sin necesidad de getters y setters.

```java
package proyectojava_p2;
public abstract class Entidad {
    protected String id;
    protected String nombre;

    public Entidad(String id, String nombre) {
        this.id = id;
        this.nombre = nombre;
    }

    public String getId() {
        return id;
    }

    public String getNombre() {
        return nombre;
    }

    public abstract void mostrarInformacion();
}
```

Continuar

Clase Producto

Representa un producto del restaurante dentro del inventario. Sirve para almacenar y gestionar toda la información relevante sobre ese producto y puede estar asociada a un proveedor específico. Al heredar de **Entidad** garantiza una estructura común para la identificación y nombre de cada producto.

```java
public class Producto extends Entidad {
    private double precio;
    private int cantidad;
    private boolean disponible;
    private String categoria;
    private int calorias;
    private ClaseProveedor proveedor;
```

Inicio todos los productos

```java
public Producto(String id, String
    super(id, nombre);
    this.precio = precio;
    this.cantidad = cantidad;
    this.disponible = disponible;
    this.categoria = categoria;
    this.calorias = calorias;
    this.proveedor = proveedor;
}
```

Método que actualiza el producto con todos sus atributos.

```java
public void actualizarProducto(Str
    this.nombre = nombre;
    this.precio = precio;
    this.cantidad = cantidad;
    this.disponible = disponible;
    this.categoria = categoria;
    this.calorias = calorias;
    this.proveedor = proveedor;
}
```

Continuar

Clase Proveedor

```java
public class ClaseProveedor extends Entidad {
    private String contacto;

    public ClaseProveedor(String id, String nombre, String contacto) {
        super(id, nombre);
        this.contacto = contacto;
    }

    public String getContacto() {
        return contacto;
    }

    public void setContacto(String contacto) {
        this.contacto = contacto;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    @Override
    public void mostrarInformacion() {
        System.out.printf("ID: %s | Nombre: %s | Contacto: %s\n",
                id, nombre, contacto);
    }
}
```

Representa un producto del restaurante dentro del inventario. Sirve para almacenar y gestionar toda la información relevante sobre ese producto y puede estar asociada a un proveedor específico.

Continuar

Clase GestorDatos

Es una parte clave de la aplicación, ya que actúa como un intermediario entre la base de datos y el resto del programa.

Permite: Agregar, buscar, eliminar y actualizar productos o proveedores, y buscar productos por nombre o categoría.

```java
package proyectojava_p2;

import java.sql.*;

public class GestorDatos {
    private static GestorDatos instancia;
    private BasedeDatos baseDatos;

    private GestorDatos() {
        baseDatos = new BasedeDatos();
    }

    public static GestorDatos getInstancia() {
        if (instancia == null) {
            instancia = new GestorDatos();
        }
        return instancia;
    }
```

← Métodos →

🔷 Métodos para Productos

Método	Descripción
`boolean agregarProducto(Producto producto)`	Inserta un nuevo producto en la tabla `productos`. Devuelve `true` si se inserta correctamente.
`Producto buscarProducto(String id)`	Busca un producto por su `id`, incluyendo información del proveedor asociado si existe.
`boolean eliminarProducto(String id)`	Elimina un producto por su `id` en la base de datos. Retorna `true` si fue eliminado.
`Producto[] obtenerTodosLosProductos()`	Devuelve un array con todos los productos, incluyendo los proveedores vinculados.
`ArrayList<Producto> buscarProductosPorNombre(String nombre)`	Devuelve una lista de productos cuyo nombre contiene el texto indicado (búsqueda parcial con `LIKE`).
`ArrayList<Producto> buscarProductosPorCategoria(String categoria)`	Devuelve una lista de productos filtrados por su categoría (`LIKE`).
`boolean actualizarProducto(Producto producto)`	Actualiza todos los datos de un producto existente en la base de datos usando su `id` como referencia.

← Continuar →

🔷 Métodos para Proveedores

Método	Descripción
`boolean agregarProveedor(ClaseProveedor proveedor)`	Inserta un nuevo proveedor en la base de datos. Retorna `true` si se agregó correctamente.
`ClaseProveedor buscarProveedor(String id)`	Busca un proveedor por su `id`. Si lo encuentra, devuelve un objeto `ClaseProveedor`.
`boolean eliminarProveedor(String id)`	Elimina un proveedor por su `id`. Retorna `true` si fue eliminado.
`ClaseProveedor[] obtenerTodosLosProveedores()`	Devuelve un array con todos los proveedores registrados en la base de datos.

El método getInstancia() sirve para que todas las partes del programa que necesiten acceder al objeto GestorDatos usen el mismo objeto en lugar de estar creando uno nuevo cada vez.

```
public static GestorDatos getInstancia() {
    if (instancia == null) {
        instancia = new GestorDatos();
    }
    return instancia;
}
```

Continuar

VentanaActualizarProducto

```
public class VentanaActualizarProducto extends JFrame {
    private GestorDatos gestor;
    private VentanaProductos ventanaPadre;
    private Producto producto;
    private JTextField txtNombre, txtPrecio, txtCantidad, txtCategoria, txtCalorias;
    private JCheckBox chkDisponible;
    private JComboBox<String> cmbProveedor;
```

Permite modificar los datos de un producto ya existente, como nombre, precio, cantidad, categoría, calorías, disponibilidad y proveedor.

- `GestorDatos gestor` : para acceder a la lógica de negocio y datos.
- `VentanaProductos ventanaPadre` : para actualizar la tabla principal cuando se guarda.
- `Producto producto` : el producto que se va a editar.
- Campos del formulario: `txtNombre` , `txtPrecio` , `txtCantidad` , `txtCategoria` , `txtCalorias` , `chkDisponible` , `cmbProveedor` .

VentanaActualizarProveedor

- `GestorDatos gestor` : acceso a lógica de negocio.
- `VentanaProveedores ventanaPadre` : ventana que muestra la lista de proveedores.
- `ClaseProveedor proveedor` : proveedor que será editado.
- Campos del formulario: `txtNombre` , `txtContacto` .

```
public class VentanaActualizarProveedor extends JFrame {
    private GestorDatos gestor;
    private VentanaProveedores ventanaPadre;
    private ClaseProveedor proveedor;
    private JTextField txtNombre, txtContacto;
```

← Métodos →

VentanaActualizarProducto

Método	Tipo	Propósito
`VentanaActualizarProducto(...)`	Constructor	Inicializa la ventana, carga datos del producto y proveedores.
`componentes`	Privado	Crea e inicializa todos los componentes gráficos (formulario, botones).
`cargarDatosProducto()`	Privado	Carga los valores actuales del producto en los campos del formulario.
`cargarProveedores()`	Privado	Llena el `JComboBox` con la lista de proveedores disponibles.
`actualizarProducto()`	Privado	Valida, actualiza el producto, guarda cambios en `GestorDatos`.

```
public class VentanaActualizarProveedor extends JFrame {
    private GestorDatos gestor;
    private VentanaProveedores ventanaPadre;
    private ClaseProveedor proveedor;
    private JTextField txtNombre, txtContacto;
```

/entanaActualizarProveedor

do	Tipo	Propósito
`VentanaActualizarProveedor(...)`	Constructor	Inicializa la ventana y carga los datos del proveedor a actualizar.
`componentes()`	Privado	Crea e inicializa los elementos gráficos del formulario.
`cargarDatosProveedor()`	Privado	Llena los campos del formulario con los datos del proveedor.
`actualizarProveedor()`	Privado	Valida entradas, actualiza el proveedor en `GestorDatos`, y cierra.

Continuar

VentanaAgregarProducto

`VentanaAgregarProducto(VentanaProductos)`	Constructor que inicializa la ventana.
`componentes()`	Configura e inicializa los componentes gráficos de la ventana.
`cargarProveedores()`	Carga los proveedores en el ComboBox desde el gestor de datos.
`guardarProducto()`	Valida los datos ingresados y guarda el producto en el sistema.

Sirve para crear una ventana gráfica donde el usuario puede ingresar los datos de un nuevo producto y guardarlo.

```
public class VentanaAgregarProducto extends JFrame {
    private GestorDatos gestor;
    private VentanaProductos ventanaPadre;
    private JTextField txtId, txtNombre, txtPrecio, txtCantidad, txtCategoria, txtCalorias;
    private JCheckBox chkDisponible;
    private JComboBox<String> cmbProveedor;
```

VentanaAgregarProveedor

Permite al usuario ingresar datos para crear un nuevo proveedor y agregarlo al sistema.

`VentanaAgregarProveedor(VentanaProveedores)`	Constructor que inicializa la ventana y el gestor.
`componentes()`	Crea y organiza los componentes gráficos.
`guardarProveedor()`	Valida los datos ingresados y guarda el proveedor.

← Continuar →

VentanaBuscarProducto

Permite buscar productos registrados, ya sea por nombre o por categoría, mostrando los resultados en una tabla emergente. Se comunica con GestorDatos para obtener los resultados.

```
public class VentanaBuscarProducto extends JFrame {
    private GestorDatos gestor;
    private VentanaProductos ventanaPadre;
    private JRadioButton rbNombre, rbCategoria;
    private JTextField txtBusqueda;
    private JButton btnBuscar;
```

`VentanaBuscarProducto(VentanaProductos)`	Constructor que inicializa la ventana.
`componentes()`	Crea y configura todos los elementos gráficos.
`buscarProductos()`	Ejecuta la lógica de búsqueda en base a la opción seleccionada.
`mostrarResultados(ArrayList<Producto>)`	Muestra los resultados en un diálogo con tabla.

Continuar

VentanaPrincipal

Desde aquí se puede acceder a las ventanas de gestión de productos y proveedores, o salir del sistema.

```
public class VentanaPrincipal extends JFrame {
    private GestorDatos gestor;

    public VentanaPrincipal() {
        gestor = GestorDatos.getInstancia();
        componentes();
    }
```

`VentanaPrincipal()`	Constructor que inicializa la ventana principal.
`componentes()`	Configura y posiciona todos los elementos gráficos. (Puede renombrarse a `componentes()`)

Continuar

VentanaProductos

Permite gestionar productos del sistema: agregar, actualizar, eliminar, buscar y visualizar todos los productos registrados en forma de tabla.

```
public class VentanaProductos extends JFrame {
    private GestorDatos gestor;
    private JTable tablaProductos;
    private DefaultTableModel modeloTabla;
```

`VentanaProductos()`	Constructor que inicializa la ventana y carga los productos.
`componentes()`	Crea y organiza los componentes gráficos.
`cargarProductos()`	Llena la tabla con todos los productos registrados.
`abrirVentanaAgregarProducto()`	Abre la ventana para agregar un nuevo producto.
`actualizarProductoSeleccionado()`	Abre la ventana de actualización para el producto seleccionado.
`eliminarProductoSeleccionado()`	Elimina el producto seleccionado de la tabla.
`abrirVentanaBusqueda()`	Abre la ventana para buscar productos.
`actualizarTabla()`	Refresca la tabla después de realizar una operación.

Continuar

VentanaProveedores

Permite gestionar proveedores: agregar, actualizar, eliminar y visualizar los proveedores existentes en una tabla.

```
public class VentanaProveedores extends JFrame {
    private GestorDatos gestor;
    private JTable tablaProveedores;
    private DefaultTableModel modeloTabla;
```

`componentes()`	Crea y organiza todos los elementos gráficos de la interfaz.
`cargarProveedores()`	Carga todos los proveedores en la tabla.
`abrirVentanaAgregarProveedor()`	Abre la ventana para agregar un nuevo proveedor.
`actualizarProveedorSeleccionado()`	Abre la ventana de edición para el proveedor seleccionado.
`eliminarProveedorSeleccionado()`	Elimina el proveedor seleccionado tras confirmación.
`actualizarTabla()`	Refresca la tabla con la lista actual de proveedores.

Continuar

Clase Main

```java
package proyectojava_p2;

public class Main {
    public static void main(String[] args) {
        VentanaPrincipal ventana = new VentanaPrincipal();
        ventana.setVisible(true);
    }
}
```

Desde aquí se lanza la aplicación y su objetivo es mostrar los menús principales, manejar la interacción con el usuario y coordinar las llamadas a los distintos métodos del sistema.

Clase BasedeDatos

Cumple la función de almacenamiento centralizado de los datos del sistema. Permite almacenar, consultar, modificar y eliminar datos de forma persistente.

```java
public class BasedeDatos {
    // Atributos de la clase
    private String database = "restaurante";
    private String hostname = "127.0.0.1"; // Host
    private String port = "3306"; // Puerto
    private String login = "root";
    private String pwd = "mysql";
    private String url = "jdbc:mysql://" + hostname + ":" + port + "/" + d
    private Connection conexion;
```

Continuar

BASE DE DATOS

```sql
CREATE TABLE proveedores (
    id VARCHAR(20) PRIMARY KEY,
    nombre VARCHAR(100) NOT NULL,
    contacto VARCHAR(100)
);
```

```sql
CREATE TABLE productos (
    id VARCHAR(20) PRIMARY KEY,
    nombre VARCHAR(100) NOT NULL,
    precio DECIMAL(10,2) NOT NULL,
    cantidad INT NOT NULL,
    disponible BOOLEAN NOT NULL,
    categoria VARCHAR(50),
    calorias INT,
    proveedor_id VARCHAR(20),
    FOREIGN KEY (proveedor_id) REFERENCES proveedores(id)
);
```

Continuar

Uso de IA.

La IA me ha ayudado sobretodo a conectar la base de datos con mi programa y la clase GestorDatos o proporcionarme pequeños elementos visuales como showConfirmDialog.

```
public Producto[] obtenerTodosLosProductos() {
    ArrayList<Producto> lista = new ArrayList<>();
    String sql = "SELECT p.*, pr.nombre as proveedor_nombre, pr.contacto as proveedor_contacto " +
                 "FROM productos p LEFT JOIN proveedores pr ON p.proveedor_id = pr.id";

    try (Connection conn = baseDatos.getConexion();
         Statement stmt = conn.createStatement();
         ResultSet rs = stmt.executeQuery(sql)) {

        while (rs.next()) {
            ClaseProveedor proveedor = null;
            if (rs.getString("proveedor_id") != null) {
                proveedor = new ClaseProveedor(
                    rs.getString("proveedor_id"),
                    rs.getString("proveedor_nombre"),
                    rs.getString("proveedor_contacto")
                );
            }

            Producto producto = new Producto(
                rs.getString("id"),
                rs.getString("nombre"),
                rs.getDouble("precio"),
                rs.getInt("cantidad"),
                rs.getBoolean("disponible"),
                rs.getString("categoria"),
                rs.getInt("calorias"),
                proveedor
            );
            lista.add(producto);
        }

    } catch (SQLException e) {
        System.out.println("Error al obtener productos: " + e.getMessage());
    }

    return lista.toArray(new Producto[0]);
}
```

Me recomendó usar así GestorDatos para asegurar que solo exista una instancia, lo que evita errores y mejora el rendimiento. También me orientó a separar esta lógica de la interfaz gráfica, porque así el código es más claro y más fácil de mantener

Continuar