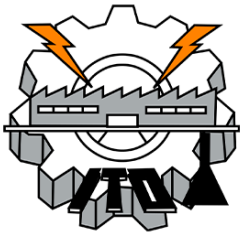


Proyecto final

Martinez Guzmán Julián

Guadalupe de Jesus Nicio Hernandez

Instituto Tecnológico de Oaxaca, Ingeniería en Sistemas Computacionales

Tópicos avanzados de programación

Adelina Martínez Nieto

1

12 de diciembre de 2024

Introducción

- Descripción general del proyecto o sistema.

Características Principales

1. **Inicio de Sesión:**
    - **Roles:** Se implementarán dos tipos de usuarios:
        - **Gerente:** Con acceso a las funcionalidades del sistema de gestión.
        - **Empleado:** Con acceso limitado a las funciones de inventario y consultas.
    - **Validación:** El inicio de sesión será seguro y validará las credenciales desde una base de datos.
2. **Gestión de Usuarios:**
    - El gerente puede crear, editar o eliminar cuentas de:
        - **Nuevos gerentes.**
        - **Empleados.**
    - Las contraseñas estarán encriptadas para garantizar la seguridad.
3. **Gestión de Inventarios:**
    - Registro de productos disponibles en la papelería, con información como:
        - Nombre del producto.
        - Precio unitario.
        - Stock actual.
        - Stock mínimo permitido.
    - Sistema de alertas que identifica los productos que han alcanzado o están por debajo del stock mínimo configurado.
4. **Pedidos a Proveedores:**
    - Cuando un producto esté por debajo del stock mínimo, el sistema permitirá realizar un pedido manualmente a los proveedores registrados.
    - Gestión de proveedores:
        - Registro de información de los proveedores: nombre, contacto.
        - Edición o eliminación de proveedores.

5. **Interfaz de Usuario:**
    - Gráfica y amigable desarrollada con **Java Swing**.
    - Paneles específicos para:
        - Inicio de sesión.
        - Gestión de usuarios.
        - Venta de productos
        - Inventario y productos.
        - Pedidos a proveedores.

- Propósito del código.

Desarrollar un sistema de gestión para una papelería utilizando Java en el entorno de desarrollo NetBeans. Este sistema permitirá la administración de inventarios, el registro de empleados y gerentes, la gestión de proveedores, y la realización de pedidos de productos con bajo stock.

- Público objetivo (desarrolladores, usuarios finales, etc.).

Este sistema está diseñado para atender las necesidades de propietarios, gerentes y empleados de pequeñas y medianas papelerías. A continuación, se detallan los perfiles específicos del público objetivo:

1. Propietarios de Papelerías

- **Descripción:**
  Personas que administran papelerías y buscan centralizar y optimizar las operaciones de su negocio.
- **Necesidades:**
    - Control del inventario para evitar pérdidas por desabasto o exceso de stock.
    - Gestión eficiente de empleados y proveedores.
    - Automatización de pedidos a proveedores.
    - Reportes claros para tomar decisiones informadas.
- **Beneficios del Sistema:**
    - Centraliza todas las operaciones del negocio.
    - Brinda alertas proactivas para evitar pérdidas.
    - Permite un mejor control de usuarios, asignando roles y responsabilidades.

2. Gerentes de Papelerías

- **Descripción:**
  Empleados encargados de administrar la operación diaria, incluida la supervisión de empleados, inventarios y proveedores.
- **Necesidades:**
  - Herramientas para delegar responsabilidades a los empleados.
  - Reportes claros sobre inventarios y pedidos realizados.
  - Capacidad para agregar nuevos empleados o modificar roles fácilmente.
- **Beneficios del Sistema:**
  - Les permite gestionar eficientemente al personal y supervisar las operaciones.
  - Facilita la toma de decisiones basadas en el control del stock y registro de pedidos.

3. Empleados de Papelerías

- **Descripción:**
  Personal de atención al cliente y operaciones que interactúa con el sistema para registrar ventas, consultar inventarios y hacer seguimiento de productos.
- **Necesidades:**
  - Acceso sencillo y limitado a inventarios y productos disponibles.
  - Herramientas para realizar tareas básicas de consulta y registro.
- **Beneficios del Sistema:**
  - Proporciona acceso rápido a información sobre disponibilidad de productos.
  - Restringe funcionalidades avanzadas, reduciendo riesgos de errores o manipulaciones indebidas.
  - Genera pedidos automatizados y organizados, facilitando su procesamiento.

4. Proveedores de Papelerías

- **Descripción:**
  Empresas o personas que suministran materiales y productos a las papelerías. Aunque no interactúan directamente con el sistema, este se diseñará para facilitar la comunicación entre proveedores y las papelerías.

- **Necesidades:**
    - Información clara y precisa sobre pedidos realizados.
    - Comunicación eficaz con la papelería para el cumplimiento de pedidos.
- **Beneficios del Sistema:**
    - Proporciona datos actualizados de contacto y productos disponibles.

Resumen del Público Objetivo

El sistema está orientado principalmente a **pequeñas y medianas papelerías** que desean profesionalizar sus operaciones mediante herramientas tecnológicas. Su diseño prioriza facilidad de uso, seguridad, y funcionalidades clave para propietarios y gerentes que buscan optimizar la gestión de su negocio.

Requisitos

- Herramientas necesarias (lenguaje de programación, bibliotecas, frameworks).

Requisitos Previos

1. **Software Necesario:**
    - **Java JDK 8 o superior.**
      (Descargar desde Oracle o usar OpenJDK).
    - **NetBeans IDE** (versión 12.x o superior).
    - **PostgreSQL** como sistema de gestión de bases de datos.
      Descarga desde la página oficial de PostgreSQL.
    - **Conector JDBC para PostgreSQL** (generalmente llamado postgresql-xx.xx.jar).
2. **Configuración Inicial de PostgreSQL:**
    - Instala PostgreSQL y crea una base de datos inicial llamada papeleria.
    - Crea un usuario para el sistema o usa el administrador (postgres).
3. **Código Fuente del Proyecto:**
    - Puedes acceder al proyecto desde un repositorio creado en GitHub llamado "Papelería".

Instalación y Configuración

- Pasos para instalar y configurar el entorno necesario.

Pasos para la Instalación

1. Instalación de PostgreSQL

- Descarga e instala PostgreSQL desde la página oficial.
- Durante la instalación:
  - Define una contraseña para el usuario postgres.
  - Asegúrate de instalar **pgAdmin** si deseas una interfaz gráfica para gestionar bases de datos.
- Inicia el servidor PostgreSQL y verifica que esté corriendo.

2. Configuración de la Base de Datos

- Accede a PostgreSQL a través de pgAdmin o la línea de comandos:

```
psql -U postgres
```

- Crea la base de datos:

```
CREATE DATABASE papeleria;
```

- Crea las tablas necesarias importando un archivo .sql incluido en el proyecto o diseñándolas manualmente:

```
CREATE SCHEMA ciber_action AUTHORIZATION postgres;

SET search_path TO ciber_action;


-- Crear la tabla Proveedor

CREATE TABLE Proveedor (

  Id_prov SERIAL PRIMARY KEY,

  Nombre VARCHAR(100) NOT NULL,

  Telefono VARCHAR(15),

  Correo_e VARCHAR(100),

  Direccion TEXT,

  RFC VARCHAR(13) NOT NULL

);
```

```
-- Crear la tabla Producto
CREATE TABLE Producto (
    Id_prod SERIAL PRIMARY KEY,
    Nombre VARCHAR(100) NOT NULL,
    Descripcion TEXT,
    Categoria VARCHAR(50),
    Precio DECIMAL(10, 2) NOT NULL
);

-- Crear la tabla Empleado
CREATE TABLE Empleado (
    Id_em SERIAL PRIMARY KEY,
    Nombre VARCHAR(100) NOT NULL,
    Telefono VARCHAR(15),
    Correo_e VARCHAR(100),
    Fecha_contratacion DATE NOT NULL
);

-- Crear la tabla Pedido
CREATE TABLE Pedido (
    Id_pedido SERIAL PRIMARY KEY,
    Id_prov INT NOT NULL,
    Estado VARCHAR(50),
    Fecha_P DATE NOT NULL,
    FOREIGN KEY (Id_prov) REFERENCES Proveedor(Id_prov) ON DELETE CASCADE
);
```

```
-- Crear la tabla Detalle_Pedido
CREATE TABLE Detalle_Pedido (
    Id_pedido INT NOT NULL,
    Id_prod INT NOT NULL,
    Cantidad INT NOT NULL,
    Monto_total DECIMAL(10, 2) NOT NULL,
    PRIMARY KEY (Id_pedido, Id_prod),
    FOREIGN KEY (Id_pedido) REFERENCES Pedido(Id_pedido) ON DELETE CASCADE,
    FOREIGN KEY (Id_prod) REFERENCES Producto(Id_prod) ON DELETE CASCADE
);

-- Crear la tabla Venta
CREATE TABLE Venta (
    Folio_v SERIAL PRIMARY KEY,
    Fecha DATE NOT NULL,
    Id_em INT NOT NULL,
    FOREIGN KEY (Id_em) REFERENCES Empleado(Id_em) ON DELETE CASCADE
);

-- Crear la tabla Detalle_Venta
CREATE TABLE Detalle_Venta (
    Folio_v INT NOT NULL,
    Id_prod INT NOT NULL,
    Cantidad INT NOT NULL,
    Monto_total DECIMAL(10, 2) NOT NULL,
    PRIMARY KEY (Folio_v, Id_prod),
    FOREIGN KEY (Folio_v) REFERENCES Venta(Folio_v) ON DELETE CASCADE,
    FOREIGN KEY (Id_prod) REFERENCES Producto(Id_prod) ON DELETE CASCADE
);
```

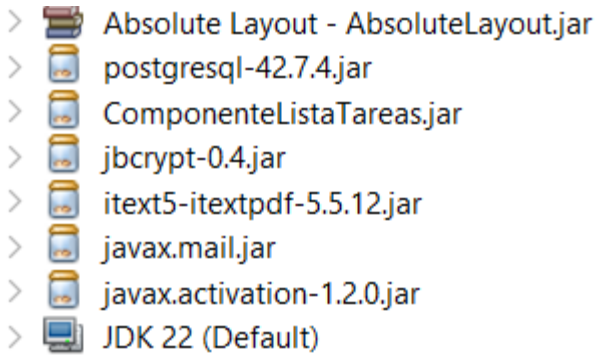
```
-- Crear la tabla Inventario
CREATE TABLE Inventario (
    Id_prod INT PRIMARY KEY,
    Cantidad_inv INT NOT NULL,
    Stock_minimo INT NOT NULL,
    FOREIGN KEY (Id_prod) REFERENCES Producto(Id_prod) ON DELETE CASCADE
);

-- Crear la tabla Usuario
CREATE TABLE Usuario (
    Id_usuario SERIAL PRIMARY KEY,
    Id_em INT NOT NULL,  -- Clave foránea que referencia a la tabla Empleado
    Nombre_usuario VARCHAR(50) NOT NULL UNIQUE,
    Contraseña VARCHAR(255) NOT NULL,
    Rol VARCHAR(20) NOT NULL CHECK (Rol IN ('Gerente', 'Empleado')),
    FOREIGN KEY (Id_em) REFERENCES Empleado(Id_em) ON DELETE CASCADE  -- Relación con la tabla Empleado
);

-- Crear la tabla Tareas
CREATE TABLE Tareas (
    Id_tarea SERIAL PRIMARY KEY,
    Id_pedido INT NOT NULL,
    Nombre VARCHAR(255) NOT NULL,
    FOREIGN KEY (Id_pedido) REFERENCES Pedido(Id_pedido) ON DELETE CASCADE
);
```

3. Configuración del Proyecto en NetBeans

1. **Importa el Proyecto:**
    - Abre NetBeans y selecciona **"File > Open Project"**.
    - Selecciona el directorio del proyecto descargado.
2. **Agregar Librerías JDBC:**
    - Descarga el conector de PostgreSQL (postgresql-xx.xx.jar) desde el sitio oficial.
    - En NetBeans, haz clic derecho sobre el proyecto y selecciona **"Properties > Libraries"**.
    - Agrega el archivo .jar descargado.



3. **Configura el Archivo de Conexión:**
    - Busca en el proyecto el archivo de configuración para la base de datos (puede ser un archivo .properties o en una clase específica).
    - Configura los parámetros de PostgreSQL:

db.url=jdbc:postgresql://localhost:5432/papeleria

db.user=postgres

db.password=tu_contraseña

4. Compilación y Ejecución

- En NetBeans, selecciona el proyecto, haz clic en **"Clean and Build"** y luego en **"Run"**.
- Inicia el sistema y verifica la conexión a la base de datos probando el inicio de sesión y operaciones básicas.

5. Verificación de Funcionalidades

- Crea usuarios y verifica que se guarden en PostgreSQL.
- Registra productos y verifica la gestión del stock.
- Realiza pruebas de pedidos a proveedores.

Estructura del Proyecto

- Descripción de los archivos y directorios principales.

Nuestro proyecto cuenta con distintos apartados y clases relacionados con cada una de las funcionalidades.

Clases relacionadas con la gestión de productos e inventario

1. **ActualizarProducto.java**
    - Clase encargada de actualizar los datos de un producto existente en la base de datos, como nombre, precio, stock o stock mínimo.
2. **AgregarAlInventario.java**
    - Permite registrar nuevos productos o actualizar las cantidades de productos existentes en el inventario.
3. **InventarioF.java**
    - Interfaz gráfica para mostrar el estado completo del inventario, incluyendo cantidades disponibles y niveles mínimos establecidos.
4. **InventarioEmpleado.java**
    - Una clase específica que brinda a los empleados acceso restringido a la vista del inventario y posibles operaciones.
5. **VerProductos.java**
    - Clase diseñada para listar y mostrar todos los productos registrados en el sistema, probablemente filtrados o categorizados según necesidad.
6. **AgregarProductos.java**
    - Permite registrar un nuevo producto en el inventario, almacenando atributos como nombre, precio y stock inicial.

Clases relacionadas con la gestión de empleados

8. **MenuEmpleado.java**
    - Una interfaz o lógica que define el menú de opciones disponible para los usuarios que inician sesión con el rol de empleado.
9. **MenuGerente.java**
    - Similar al anterior, pero adaptada para las funcionalidades disponibles al gerente, como acceso al sistema de pedidos o a la gestión de usuarios.
10. **VerEmpleados.java**
    - Clase para listar y visualizar todos los empleados registrados en el sistema.
11. **ActualizarEmpleado.java**
    - Se utiliza para modificar los datos de los empleados, como su nombre, rol o contraseña.
12. **AgregarEmpleados.java**
    - Clase para registrar nuevos empleados en el sistema y asociarlos a su rol correspondiente (gerente o empleado).

Clases relacionadas con la gestión de pedidos

13. **PedidoF.java**
    - Interfaz para gestionar pedidos realizados a los proveedores, permitiendo registrar nuevos pedidos o visualizar los ya existentes.
14. **VerPedidos.java**
    - Lista y permite consultar todos los pedidos realizados por el sistema, con detalles como fechas y cantidades.
15. **VerPedGerente.java**
    - Similar a **VerPedidos.java**, pero enfocada en la información relevante para el gerente.

Clases relacionadas con la gestión de proveedores

16. **AgregarProveedores.java**
    - Clase utilizada para registrar nuevos proveedores en la base de datos, almacenando información como nombre y contacto.

17. **ActualizarProveedor.java**
    - Permite modificar los datos de los proveedores existentes en el sistema.
18. **VerProveedores.java**
    - Clase diseñada para listar todos los proveedores registrados, incluyendo sus detalles y los productos que ofrecen.

Clases relacionadas con ventas

19. **Venta.java**
    - Esta clase diseñada para gestionar el proceso de venta de productos, como disminuir el stock o registrar transacciones.

Otros

20. **NewJFrame.java**
    - Clase la cual contiene el login de usuario, tanto como para gerente y empleados comunes.

- Explicación de cómo están organizados los módulos o componentes.

1. Módulo de Gestión de Productos e Inventarios

Objetivo:

Manejar el registro, actualización y control de inventarios de productos de la papelería, garantizando el abastecimiento adecuado.

Clases Involucradas:

- **ActualizarProducto.java**: Permite la edición de información de productos existentes.
- **AgregarAlInventario.java**: Incrementa el stock de productos ya registrados.
- **Fstock.java**: Probablemente monitorea niveles mínimos de inventario para alertar al gerente sobre necesidades de reposición.
- **InventarioF.java**: Visualiza el inventario completo.
- **InventarioEmpleado.java**: Da acceso restringido al inventario para empleados, solo permitiéndoles consultar información.
- **AgregarProductos.java**: Permite agregar nuevos productos a la base de datos.
- **VerProductos.java**: Muestra una lista de todos los productos disponibles.

Relación entre las Clases:

Estas clases trabajan juntas para manejar el ciclo completo del inventario, desde la introducción de nuevos productos, actualizaciones, consulta del estado actual y la detección de alertas relacionadas con niveles bajos de stock.

2. Módulo de Gestión de Empleados

Objetivo:

Facilitar la administración del personal que opera en la papelería, tanto gerentes como empleados, con diferentes roles y niveles de acceso.

Clases Involucradas:

- **MenuEmpleado.java**: Define la interfaz y funcionalidades accesibles exclusivamente para los empleados.
- **MenuGerente.java**: Ofrece un conjunto de herramientas ampliado para los gerentes, incluyendo gestión de empleados y productos.
- **ActualizarEmpleado.java**: Actualiza los datos de los empleados registrados.
- **AgregarEmpleados.java**: Permite registrar nuevos empleados, asignándoles sus roles y permisos.
- **VerEmpleados.java**: Muestra una lista de los empleados registrados.

Relación entre las Clases:

- **Los menús (empleado y gerente)** definen lo que cada rol puede hacer.
- **Las clases de gestión de empleados** permiten al gerente administrar los datos de los trabajadores de forma centralizada.

3. Módulo de Gestión de Proveedores

Objetivo:

Controlar el registro, actualización y consulta de los proveedores de productos, permitiendo un flujo adecuado de reposición de inventarios.

Clases Involucradas:

- **AgregarProveedores.java**: Añade nuevos proveedores a la base de datos.
- **ActualizarProveedor.java**: Edita la información de proveedores existentes.
- **VerProveedores.java**: Lista y consulta a los proveedores registrados.

Relación entre las Clases:

Este módulo conecta directamente con el sistema de productos para garantizar que los pedidos se dirijan a los proveedores adecuados, manteniendo una base de datos actualizada y accesible.

4. Módulo de Gestión de Pedidos

Objetivo:

Registrar y procesar pedidos de reabastecimiento hacia los proveedores cuando los niveles de inventario son bajos.

Clases Involucradas:

- **PedidoF.java**: Registra los pedidos que se realizan a proveedores.
- **VerPedidos.java**: Muestra los pedidos realizados y en proceso.
- **VerPedGerente.java**: Ofrece un desglose de pedidos más completo o específico para el gerente.

Relación entre las Clases:

La generación de pedidos puede estar vinculada a las alertas del inventario (clase **Fstock**), mientras que el seguimiento se realiza a través de las herramientas de visualización disponibles para el gerente.

5. Módulo de Ventas

Objetivo:

Registrar las ventas realizadas a los clientes y actualizar automáticamente el inventario en función de los productos vendidos.

Clase Involucrada:

- **Venta.java**: Realiza el proceso de ventas, asegurando que se descuenten los productos vendidos del inventario y que se guarde la transacción.

Relación con Otros Módulos:

Este módulo está vinculado principalmente con:

- **El inventario** para controlar la cantidad de productos.
- **El rol del empleado** que lleva a cabo las transacciones.

6. Módulo de Inicio de Sesión

Objetivo:

Controlar el acceso al sistema mediante el inicio de sesión para empleados y gerentes, asegurando que cada usuario acceda únicamente a las funcionalidades de acuerdo con su rol.

Clase Involucrada:

- **NewJFrame.java**
  Es la ventana principal del sistema y cumple las siguientes funciones:
  - Solicitar credenciales (usuario y contraseña).
  - Validar las credenciales contra la base de datos (tabla usuarios).
  - Redirigir al menú correspondiente según el rol del usuario:
    - **Empleado:** Acceso a **MenuEmpleado.java**.
    - **Gerente:** Acceso a **MenuGerente.java**.

Relación con otros Módulos:

1. **Interacción con la base de datos:**
   - Realiza consultas a la tabla usuarios de la base de datos PostgreSQL para autenticar usuarios.
2. **Redireccionamiento:**
   Una vez que el usuario ha sido autenticado, NewJFrame redirige a la ventana que corresponde al rol.

Flujo del Inicio de Sesión:

1. **Ingreso de Credenciales:** El usuario introduce el nombre de usuario y contraseña.
2. **Validación:** Se compara con los datos de la base de datos. Si coinciden:
   - Rol *empleado*: Se abre **MenuEmpleado**.
   - Rol *gerente*: Se abre **MenuGerente**.
3. **Gestión de Errores:** Si las credenciales no coinciden, se muestra un mensaje de error y se permite reintentar.

Manual de Uso

- Instrucciones para ejecutar el código.

Requisitos Previos

1. **Instalar Java:**
   Asegúrate de que tienes el **Java JDK 8 o superior** instalado en tu máquina. Puedes verificarlo con el comando:
2. **Instalar NetBeans:**
   Descarga e instala **NetBeans IDE**, que es el entorno en el cual está desarrollado este proyecto.
3. **Instalar PostgreSQL:**
   - Configura PostgreSQL e instala la base de datos.
   - Crea una base de datos llamada **papelería**.
   - Usa el siguiente script SQL para crear las tablas necesarias:
4. **Configura las Credenciales de PostgreSQL en el Código:**
   - En el archivo NewJFrame.java, reemplaza las credenciales por las tuyas en este bloque:

```
Connection conn = DriverManager.getConnection(
  "jdbc:postgresql://localhost:5432/papeleria", "postgres", "tu_contraseña");
```

Ejecución del Proyecto

1. Abre NetBeans y carga el proyecto:
   - Usa **Archivo > Abrir proyecto** y selecciona la carpeta donde está el proyecto.
2. Configura la biblioteca JDBC:
   - Añádelo al proyecto en **Propiedades > Bibliotecas > Añadir JAR**.
3. Ejecuta el proyecto:
   - Haz clic derecho sobre el archivo principal (NewJFrame.java) y selecciona **Ejecutar Archivo**.

- Ejemplos de entrada y salida.

1. Inicio de Sesión

- **Entrada:**
    - Usuario: gerente1
    - Contraseña: admin123
- **Salida esperada:**
    - Redirección a la ventana **MenuGerente** (donde se listan las opciones de gestión de empleados, productos, y pedidos).
- **Caso Fallido:**
    - Entrada incorrecta:
      Usuario: empleado1
      Contraseña: wrongpass
    - Salida esperada: Un mensaje emergente de error:

"Usuario o contraseña incorrectos"

2. Gestión de Productos

- Desde **MenuGerente**, selecciona "Agregar Producto".
- **Entrada:**
    - Nombre del producto: Cuaderno A4
    - Precio: 45.50
    - Stock inicial: 100
    - Stock mínimo: 10
- **Salida esperada:**
  Un mensaje que confirme el registro:

"Producto agregado exitosamente."

3. Verificar Alerta de Stock Mínimo

- **Entrada:**
    - Se consulta el inventario donde el stock actual está por debajo del mínimo.
- **Salida esperada:**
  Una lista en pantalla con productos como:

Producto: Cuaderno A4, Stock Actual: 5, Stock Mínimo: 10.

4. Realizar Pedido a Proveedores

- Desde **MenuGerente**, selecciona "Registrar Pedido".
- **Entrada:**
    - Producto: Cuaderno A4
    - Cantidad: 50
    - Proveedor: Proveedor Papelería ABC
- **Salida esperada:**

"Pedido registrado exitosamente."

- Explicación de las funcionalidades principales.

1. Inicio de Sesión

El archivo **NewJFrame.java** se utiliza para autenticar a los usuarios en la base de datos PostgreSQL. Según su rol (empleado o gerente), los redirige al menú correspondiente:

- **Empleados:** Acceden a opciones como ventas y consultas de inventario.
- **Gerentes:** Tienen acceso a todo el sistema, incluyendo gestión de empleados, proveedores, y productos.

2. Gestión de Productos

El gerente puede realizar las siguientes acciones sobre los productos:

- **Agregar productos nuevos** al inventario.
- **Actualizar información** de productos existentes.
- Consultar el inventario y detectar niveles bajos de stock.

3. Gestión de Empleados

Permite a los gerentes:

- Registrar nuevos empleados en el sistema.
- Modificar datos existentes.
- Consultar la lista de empleados registrados.

4. Gestión de Proveedores

Este módulo maneja el registro, edición, y consulta de proveedores que abastecen la papelería. Está vinculado directamente con el sistema de pedidos.

5. Sistema de Pedidos

Se realiza un pedido automáticamente o de manera manual cuando los niveles de stock son bajos. Cada pedido:

- Está asociado a un proveedor.
- Actualiza los registros en la base de datos para que se mantenga un historial.

6. Ventas

Los empleados pueden registrar ventas y descontar automáticamente el inventario de los productos vendidos. Estas operaciones están ligadas al sistema de control de stock.

Referencia del Código

- Descripción detallada de cada módulo, clase o función:

La clase NewJFrame.java es la ventana de inicio de sesión de la aplicación de papelería. Está diseñada para autenticar a los usuarios (empleados y gerentes) según su rol, proporcionando las credenciales para acceder a los respectivos menús.

Organización General

1. **Diseño visual**:
    - Usa la biblioteca javax.swing para construir una interfaz gráfica simple y funcional.
    - Contiene elementos como etiquetas (JLabel), botones de radio (JRadioButton), campos de texto y contraseña (JTextField, JPasswordField), y un botón para iniciar sesión.
2. **Componentes principales**:
    - Un botón de selección (JRadioButton) para elegir el **rol**: Gerente o Empleado.
    - Un campo de texto para ingresar el **nombre de usuario**.
    - Un campo de contraseña para capturar la **contraseña** del usuario.
    - Botón de acción (jButton1) que desencadena el proceso de autenticación.
3. **Uso de Base de Datos**:
    - Se conecta a una base de datos PostgreSQL para validar las credenciales y determinar el rol del usuario.
4. **Validación de contraseña**:

- Utiliza la función BCrypt.checkpw() para comparar contraseñas encriptadas en la base de datos.

Flujo de Ejecución

1. **Inicio de sesión**:
   - Cuando el usuario da clic en el botón **"Iniciar sesión"**, se recopila el nombre de usuario, la contraseña, y el rol seleccionado.
   - El código valida que se haya seleccionado un rol antes de proceder. Si no, muestra un mensaje de advertencia.
2. **Consulta en la base de datos**:
   - El sistema usa una conexión a PostgreSQL para buscar el usuario en la tabla Usuario de la base de datos.
   - **Consulta SQL**: Busca la contraseña encriptada (Contraseña) y el rol (Rol) asociado al usuario proporcionado.
3. **Verificación de credenciales**:
   - Compara la contraseña ingresada por el usuario (usando BCrypt.checkpw) con la almacenada en la base de datos.
   - Si la contraseña es válida y el rol coincide con el seleccionado, el sistema redirige al usuario al menú correspondiente:
     - **Gerente**: Se abre MenuGerente.
     - **Empleado**: Se abre MenuEmpleado.
4. **Manejo de errores**:
   - Si las credenciales no coinciden, el programa muestra un mensaje de error.
   - Maneja errores de conexión a la base de datos con un try-catch para evitar interrupciones del programa.

Aspectos Técnicos Clave

- **Personalización de la interfaz gráfica**:
  - Utiliza el esquema de colores y diseño visual mediante la clase Color para mantener un aspecto limpio y profesional.
  - Integra un ícono en la parte superior de la ventana.

- **Encapsulamiento de seguridad**:
    - La encriptación y verificación de contraseñas usando BCrypt asegura que las contraseñas no se almacenen en texto plano, cumpliendo con buenas prácticas de seguridad.
- **Interactividad**:
    - Los componentes como los botones de selección y los mensajes emergentes (JOptionPane) facilitan la interacción intuitiva.
- **Conexión a PostgreSQL**:
    - Establece la conexión con las credenciales predeterminadas:

String url = "jdbc:postgresql://localhost:5432/Papeleria";

String user = "postgres";

String dbPassword = "9656";

- **Esquema dinámico**:
    - Ajusta el esquema activo mediante la consulta SQL:

SET search_path TO ciber_action;

Ejemplo de Entrada y Salida

Entrada:

1. Usuario selecciona **"Gerente"**.
2. Ingresa:
    - Nombre de usuario: admin.
    - Contraseña: 1234.

Proceso Interno:

- Consulta en la tabla Usuario de la base de datos.
- Contrasta el rol (Gerente) y verifica la contraseña encriptada.

Salida:

- Si las credenciales son correctas, se abre el menú del gerente.
- Si las credenciales son incorrectas, muestra:

“Usuario o contraseña incorrectos.”

Principales Funcionalidades

1. **Inicio de sesión**:
    - Función principal que autentica a los usuarios basándose en credenciales y rol.
2. **Redirección según rol**:
    - Distintos menús para gerentes y empleados.
3. **Mensajes de error**:
    - Detecta errores como falta de selección de rol, usuario no existente o contraseñas incorrectas.

1. Estructura principal

- **Clase MenuGerente**: Es una ventana gráfica que actúa como el menú principal del gerente. Está diseñada para interactuar con otras partes del programa, como gestionar empleados, proveedores, productos y ver el historial de pedidos.
- **Método main**: Es el punto de entrada de la aplicación. Inicializa la ventana (aunque parece que llama inicialmente a otra clase NewJFrame).

---

2. Constructor

```
public MenuGerente() {
   initComponents();
   this.getContentPane().setBackground(new Color(0x86C7E7));
}
```

- Llama a initComponents, que inicializa todos los elementos gráficos (botones, etiquetas, paneles, etc.). Este método es generado automáticamente por el diseñador de interfaces de NetBeans y no debe ser modificado manualmente.
- Cambia el fondo de la ventana a un color específico (0x86C7E7, un tono azul claro).

---

3. Elementos gráficos

a) Panel principal (jPanel1)

- Es el contenedor principal que utiliza un layout de tipo AbsoluteLayout. Esto permite posicionar los componentes manualmente mediante coordenadas absolutas.

b) Botones

Los botones tienen acciones asociadas, y cada uno abre una ventana nueva específica:

- **Regresar**: Regresa al frame NewJFrame (probablemente el login o menú principal).

```
private void RegresarActionPerformed(java.awt.event.ActionEvent evt) {
   new NewJFrame().setVisible(true);
   this.dispose();
}
```

dispose cierra la ventana actual.

- **Historial de pedidos**: Abre la ventana VerPedGerente, probablemente para revisar los pedidos realizados.

```
private void RevHisPedActionPerformed(java.awt.event.ActionEvent evt) {
   VerPedGerente vf = new VerPedGerente();
   vf.setVisible(true);
   this.dispose();
}
```

- **Empleados**, **Proveedores**, y **Productos**: Cada botón abre su respectiva ventana (agregarEmpleados, agregarProovedores, agregarProductos).

c) Etiquetas (jLabel)

- Se usan para mostrar íconos (como imágenes representativas). Por ejemplo:

```
jLabel3.setIcon(new javax.swing.ImageIcon(getClass().getResource("/emp.png")));
```

d) Diseño visual

- El diseño usa colores personalizados y fuentes (Roboto). Las imágenes y estilos se cargan desde el proyecto (getResource()).

---

4. Interacción con otras clases

Los botones están conectados a otras ventanas (agregarEmpleados, agregarProovedores, agregarProductos, VerPedGerente). Estas clases no están incluidas aquí, pero parecen ser parte del sistema para gestionar las diferentes entidades.

La clase **MenuEmpleado** representa una ventana gráfica de Java Swing que sirve como el menú principal para un empleado en una aplicación de papelería

Componentes principales:

1. **Herencia de JFrame:** La clase extiende javax.swing.JFrame, lo que le permite comportarse como una ventana gráfica independiente.
2. **Método constructor (MenuEmpleado()):**
    - Llama al método initComponents() para inicializar los componentes gráficos.
    - Cambia el fondo del contenido principal a un color azul claro (new Color(0x86C7E7)).
3. **Método initComponents():**
    - Es generado automáticamente por el editor visual de Java Swing. Configura el diseño, agrega los botones, etiquetas e íconos, y define las acciones asociadas a cada botón.
4. **Componentes clave:**
    - **Botón "Vender" (venderButton)**: Abre la ventana Venta (presumiblemente relacionada con el proceso de ventas) y cierra la ventana actual.
    - **Botón "Revisar inventario" (RevInv)**: Abre la ventana InventarioEmpleado para ver los productos disponibles.
    - **Botón "Revisar pedidos" (RevPed)**: Abre la ventana PedidoF para consultar los pedidos realizados.
    - **Botón "Regresar" (jButton4)**: Retorna al NewJFrame (aparentemente el menú principal o una pantalla de inicio de sesión) y cierra la ventana actual.
    - **Etiquetas (jLabel1, jLabel2, jLabel4)**: Representan íconos o imágenes decorativas para identificar las opciones de forma gráfica.
5. **Métodos de acción:** Cada botón tiene asociado un evento ActionListener que define el comportamiento cuando se hace clic:
    - **Ejemplo (venderButtonActionPerformed)**:

java

Copiar código

```
private void venderButtonActionPerformed(java.awt.event.ActionEvent evt) {
   Venta agP = new Venta();
   agP.setVisible(true); // Muestra la ventana de ventas.
   this.dispose();      // Cierra la ventana actual.
}
```

Flujo de ejecución:

1. **Creación del formulario:** En el método main, se configura el tema gráfico (Look and Feel) y se crea una instancia de MenuEmpleado:

```
public static void main(String args[]) {
   java.awt.EventQueue.invokeLater(new Runnable() {
      public void run() {
         new MenuEmpleado().setVisible(true);
      }
   });
}
```

2. **Interacción del usuario:**
    - Cuando el usuario selecciona una acción (por ejemplo, "Vender"), se abre la ventana correspondiente, y la ventana del menú actual se cierra para evitar duplicidad.
3. **Estilo visual:**
    - Usa el gestor de diseño javax.swing.GroupLayout para organizar los componentes dentro de jPanel1.

Clase agregarEmpleados

Esta clase es una interfaz gráfica de usuario (GUI) diseñada para gestionar la adición de empleados a una base de datos en un sistema de papelería. Utiliza javax.swing para la creación de componentes gráficos y establece conexiones con una base de datos PostgreSQL para almacenar información.

Funcionalidad Principal

1. **Agregar Empleado**:
    - Recopila datos de entrada del usuario a través de los campos de texto (JTextPane) y un combo box para el rol.
    - Valida los campos, asegurándose de que no estén vacíos y que la fecha siga el formato YYYY-MM-DD.
    - Se conecta a una base de datos PostgreSQL y realiza dos inserciones:
        - En la tabla Empleado: Guarda la información básica como nombre, teléfono, correo electrónico, y fecha de contratación.
        - En la tabla Usuario: Crea un usuario asociado al empleado utilizando su ID generado y encriptando la contraseña con la biblioteca BCrypt.
2. **Ver Empleados**:
    - Redirige al usuario a una ventana distinta (VerEmpleados) que probablemente muestra una lista de empleados registrados.

3. **Regresar**:
    - Permite al usuario regresar al menú principal (MenuGerente).

Componentes Gráficos

- **Paneles (JPanel)**:
    - Dividen el diseño visual en secciones: Encabezado, formulario de datos y botones de acción.
- **Campos de Texto (JTextPane)**:
    - Capturan datos específicos del empleado como nombre, teléfono, correo electrónico, fecha de contratación, nombre de usuario, y contraseña.
- **Combo Box (JComboBox)**:
    - Permite seleccionar el rol del empleado (Gerente o Empleado).
- **Botones (JButton)**:
    - Botones interactivos para las acciones principales: "Añadir", "Ver empleados" y "Regresar".

Métodos Importantes

1. **AgregarEmpActionPerformed**:
    - Lógica principal para agregar empleados.
    - Incluye validaciones de entrada, procesamiento de fechas, encriptación de contraseñas y operaciones de base de datos.
2. **VerEmpleadosActionPerformed**:
    - Navega hacia la ventana de visualización de empleados registrados.
3. **RegresarbuttActionPerformed**:
    - Navega de regreso al menú principal del sistema.
4. **Clase Interna PasswordUtil**:
    - Proporciona métodos para encriptar contraseñas (hashPassword) y para validar contraseñas encriptadas (checkPassword).

Conexión con la Base de Datos

- **URL**: jdbc:postgresql://localhost:5432/Papeleria
- **Esquema**: Cambiado a ciber_action usando el comando SET search_path.
- **Tablas Afectadas**:
    - Empleado: Contiene datos básicos del empleado.

- Usuario: Almacena credenciales del empleado, incluyendo contraseña encriptada y rol.

Validaciones y Seguridad

- Validación de entrada:
  - Verifica que todos los campos estén llenos.
  - Asegura que la fecha ingresada siga el formato correcto (yyyy-MM-dd).
- Seguridad:
  - La contraseña se almacena en la base de datos usando encriptación BCrypt, lo que dificulta su descifrado en caso de acceso no autorizado.

Ejecución

La clase incluye un método main que inicializa y muestra la ventana agregarEmpleados. Usa la biblioteca Nimbus para mejorar la apariencia de la GUI si está disponible.

Elementos principales:

1. **Clase agregarProductos**:
   - Extiende javax.swing.JFrame para crear una ventana gráfica.
   - Define una interfaz gráfica que permite a los usuarios ingresar detalles de productos como nombre, descripción, categoría y precio.
2. **Componentes principales de la interfaz**:
   - **Campos de entrada**:
     - nombretext: Campo para ingresar el nombre del producto.
     - descriptext: Campo para la descripción del producto.
     - categtext: Campo para especificar la categoría del producto.
     - preciotext: Campo para ingresar el precio del producto.
   - **Botones**:
     - agregarProductos: Botón para guardar el producto en la base de datos.
     - verProd: Botón para ver una lista de productos.
     - Regresarbutt: Botón para regresar al menú principal.
3. **Conexión a la base de datos**:
   - Se utiliza un controlador JDBC para conectarse a una base de datos PostgreSQL.
   - Se establece el esquema a utilizar con el comando SQL SET search_path TO ciber_action.

- Se inserta el producto con la consulta INSERT INTO producto (nombre, descripcion, categoria, precio) VALUES (?, ?, ?, ?).

4. **Validaciones de entrada**:
   - Verifica que todos los campos estén completos.
   - Asegura que el precio sea un número válido utilizando BigDecimal.
   - Muestra mensajes de error o confirmación mediante cuadros de diálogo (JOptionPane).
5. **Gestión de acciones**:
   - El método agregarProductosActionPerformed maneja la acción de agregar un producto.
   - verProdActionPerformed muestra otra ventana con la lista de productos.
   - RegresarbuttActionPerformed cierra esta ventana y regresa al menú principal.
6. **Gestión de errores**:
   - Captura excepciones relacionadas con formato de precio (NumberFormatException).
   - Maneja errores de conexión o consultas SQL (SQLException)

Clase agregarProovedores

Esta clase es parte del paquete papeleria y representa una interfaz gráfica (GUI) para agregar proveedores a una base de datos. La clase utiliza la biblioteca **Swing** para construir la interfaz y conectar a una base de datos PostgreSQL.

Propósito

El objetivo principal de esta clase es permitir al usuario ingresar datos sobre un proveedor, como nombre, teléfono, correo electrónico, dirección, y RFC, y almacenarlos en la base de datos.

Componentes principales

1. **Interfaz gráfica:**
   - **Campos de entrada**:
     - nombretext: Para ingresar el nombre del proveedor (obligatorio).
     - telefonotext: Para ingresar el número de teléfono.
     - correotext: Para ingresar el correo electrónico.
     - direcciontext: Para ingresar la dirección.
     - rfctex: Para ingresar el RFC del proveedor (obligatorio).
   - **Botones**:
     - añadirProv: Botón que agrega el proveedor a la base de datos al hacer clic.

- VerProv: Botón que abre otra ventana para ver los proveedores registrados.
- Regresarbutt: Botón que regresa al menú principal.

2. **Base de datos:**
    - La conexión a la base de datos utiliza JDBC para interactuar con una base PostgreSQL.
    - Los datos del proveedor se insertan en una tabla llamada **Proveedor** dentro del esquema **ciber_action**.
3. **Métodos principales:**
    - **añadirProvActionPerformed**:
        - Valida que los campos obligatorios (nombre y RFC) estén completos.
        - Conecta a la base de datos, ajusta el esquema de búsqueda, y ejecuta un INSERT para guardar los datos del proveedor.
        - Muestra mensajes al usuario en caso de éxito o error.
    - **RegresarbuttActionPerformed**:
        - Abre la ventana principal del menú del gerente y cierra la ventana actual.
    - **VerProvActionPerformed**:
        - Abre la ventana de visualización de proveedores.
4. **Diseño visual:**
    - Usa **JPanel**, **JTextPane**, y **JButton** para organizar la interfaz.
    - Aplica estilos de color y tipografía para mejorar la experiencia visual.
    - La clase está diseñada con el editor de formularios de NetBeans, como se indica en los comentarios generados automáticamente.

Flujo de funcionamiento

1. **Inicialización:**
    - Al ejecutarse, el constructor agregarProovedores llama a initComponents, que inicializa y organiza todos los componentes de la interfaz.
    - Se establece un fondo de color azul claro para la ventana principal.
2. **Agregar proveedor:**
    - El usuario ingresa los datos en los campos correspondientes y presiona el botón "AÑADIR".
    - Los datos ingresados son validados para asegurar que los campos obligatorios estén completos.

- Se realiza la conexión a la base de datos y los datos son insertados.
- El sistema muestra un mensaje indicando si la operación fue exitosa o si ocurrió un error.

3. **Navegación:**
    - El botón "Regresar" permite volver al menú principal.
    - El botón "VerProovedores" abre una ventana que muestra los proveedores registrados.

Requisitos de configuración

- **Base de datos:**
    - Una base de datos PostgreSQL debe estar configurada en localhost:5432 con el nombre Papeleria.
    - El usuario de la base de datos debe ser postgres y la contraseña 9656.
    - La tabla Proveedor en el esquema ciber_action debe tener los campos: nombre, telefono, correo_e, direccion, rfc.
- **Dependencias de Java:**
    - Biblioteca JDBC para PostgreSQL.

La clase VerEmpleados hereda de javax.swing.JFrame y sirve como una ventana para gestionar empleados. Permite visualizar los datos en una tabla, eliminarlos y actualizarlos mediante la interacción con una base de datos.

Componentes Principales

1. Atributos:

- **EmTable**: Una tabla (JTable) donde se muestran los datos de los empleados.
- **Regresarbutt**: Botón para regresar a la ventana anterior.
- **actualizarEmp y borrarEmp**: Botones para actualizar y eliminar registros de empleados.
- **jPanel1 y jPanel2**: Paneles para organizar los componentes gráficos.
- **Conexión a la base de datos:** Configuración para conectarse a una base PostgreSQL localizada en localhost.

---

2. Métodos Principales:

a. Constructor (VerEmpleados)

Inicializa los componentes de la interfaz y genera la tabla con los datos de los empleados mediante el método generarTable().

---

b. generarTable()

Este método realiza las siguientes acciones:

1. Conecta a la base de datos PostgreSQL usando DriverManager.
2. Cambia el esquema activo a ciber_action mediante el comando SQL SET search_path.
3. Ejecuta una consulta SQL para obtener todos los registros de la tabla empleado.
4. Extrae los datos y los metadatos (nombres de columnas) del resultado.
5. Crea un modelo de tabla (DefaultTableModel) que es asignado a EmTable.

Excepción manejada: Si ocurre un error durante la conexión o consulta, se muestra un mensaje de error.

---

c. borrarEmp()

Permite eliminar un empleado seleccionado en la tabla:

1. Verifica si hay una fila seleccionada.
2. Extrae el ID del empleado desde la primera columna de la fila seleccionada.
3. Pide confirmación al usuario antes de eliminar.
4. Realiza una consulta SQL DELETE para eliminar el empleado por su ID.
5. Actualiza la tabla si la operación es exitosa.

Manejo de errores: Se muestra un mensaje si ocurre un error o si no se encuentra el empleado.

---

d. actualizarEmpActionPerformed()

Se ejecuta al hacer clic en el botón "Actualizar":

1. Verifica si hay una fila seleccionada.
2. Obtiene los valores de las columnas seleccionadas (ID, nombre, teléfono, correo, fecha).
3. Crea una instancia de la clase actualizarEmpleado y la abre para permitir la actualización de los datos.

---

e. RegresarbuttActionPerformed()

Regresa a la ventana anterior (agregarEmpleados) y cierra la ventana actual.

Gestión de Eventos

Los eventos son manejados por métodos ActionListener asociados a los botones:

- **RegresarbuttActionPerformed**: Regresa al menú principal.
- **actualizarEmpActionPerformed**: Llama a la lógica de actualización.
- **borrarEmpActionPerformed**: Llama a la lógica para eliminar un empleado.

Base de Datos

El esquema de la base de datos contiene una tabla empleado que se espera que tenga al menos las siguientes columnas:

- **ID (id_em)**: Identificador único del empleado.
- **Nombre, teléfono, correo, fecha**: Información básica del empleado.

Puntos de Mejora

1. **Parámetros de conexión**: Los valores de conexión (URL, usuario, contraseña) están codificados. Se recomienda moverlos a un archivo de configuración.
2. **Validación de datos**: Incluir validaciones más robustas para evitar errores en tiempo de ejecución.
3. **Manejo de excepciones**: Centralizar el manejo de errores y mostrar mensajes más detallados.

Clase: VerPedGerente

Descripción General

La clase VerPedGerente extiende javax.swing.JFrame y proporciona una interfaz gráfica para que el gerente visualice y gestione los pedidos almacenados en la base de datos.

Atributos

private javax.swing.JButton Regresarbutt;

private javax.swing.JButton jButton1;

private javax.swing.JLabel jLabel1;

private javax.swing.JPanel jPanel2;

private javax.swing.JScrollPane jScrollPane1;

private javax.swing.JTable pedidoTable;

Estos componentes son elementos visuales usados en la interfaz, tales como botones, etiquetas y tablas.

Constructor

```
public VerPedGerente() {
   initComponents();
   generarTable();
}
```

1. **initComponents()**: Inicializa los componentes visuales de la ventana. Este método es generado automáticamente por herramientas como NetBeans.
2. **generarTable()**: Carga y muestra los datos de los pedidos desde la base de datos en la tabla.

Métodos Principales

generarTable

public void generarTable() { ... }

Carga los datos de los pedidos desde una base de datos PostgreSQL y los muestra en la tabla.

- **Conexión a la base de datos**:

```
String url = "jdbc:postgresql://localhost:5432/Papeleria";
String user = "postgres";
String password = "9656";
Connection con = DriverManager.getConnection(url, user, password);
```

Se conecta a la base de datos Papeleria con las credenciales especificadas.

- **Consulta SQL**:

```
SELECT p.Id_pedido, p.Id_prov, p.Estado, p.Fecha_P,
    dp.Id_prod, dp.Cantidad, dp.Monto_total
FROM Pedido p
JOIN Detalle_Pedido dp ON p.Id_pedido = dp.Id_pedido;
```

Recupera datos relevantes de las tablas Pedido y Detalle_Pedido.

- **Construcción de la tabla**: Utiliza DefaultTableModel para organizar los datos obtenidos en un formato adecuado para mostrarlos en pedidoTable.
- **Excepciones**: Maneja errores con SQLException mostrando un mensaje emergente si ocurre un problema al cargar los datos.

RegresarbuttActionPerformed

private void RegresarbuttActionPerformed(java.awt.event.ActionEvent evt) { ... }

Este método se ejecuta cuando el usuario presiona el botón "Regresar".

- Abre la ventana MenuGerente y cierra la actual.

---

jButton1ActionPerformed

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) { ... }

Este método elimina los pedidos con estado "Recibido" de la base de datos, junto con sus dependencias.

- **Acciones clave**:
    1. **Transacción SQL**:

con.setAutoCommit(false);

Usa transacciones para asegurar que todos los cambios se realicen correctamente o ninguno en caso de error.

    2. **Eliminación de dependencias**:

DELETE FROM Tareas WHERE Id_pedido IN (SELECT Id_pedido FROM Pedido WHERE Estado = 'Recibido');

DELETE FROM Detalle_Pedido WHERE Id_pedido IN (SELECT Id_pedido FROM Pedido WHERE Estado = 'Recibido');

DELETE FROM Pedido WHERE Estado = 'Recibido';

Elimina registros en cascada en las tablas relacionadas.

    3. **Confirmación o reversión**:

con.commit();

con.rollback();

Aplica los cambios si todo es exitoso o los revierte en caso de error.

- **Mensajes al usuario**: Utiliza JOptionPane para informar al usuario del resultado de la operación.

Método main

public static void main(String args[]) { ... }

Inicia la aplicación gráfica, configurando la apariencia y mostrando la ventana principal.

Interfaz Gráfica

- **Tabla (pedidoTable)**: Muestra los pedidos y sus detalles.
- **Botones**:
  - **"Regresar"**: Vuelve al menú principal.
  - **"Eliminar historial"**: Borra pedidos con estado "Recibido".

Manejo de Errores

El programa maneja errores con bloques try-catch y muestra mensajes explicativos al usuario en caso de fallos de conexión o ejecución de consultas.

Clase: Verpedidos

Componentes del Código

Paquetes Importados

1. **Paquete de Swing:**
   - Se utiliza para construir la interfaz gráfica.
   - Componentes como JTable, JPanel, JScrollPane y JButton son empleados.
2. **Paquete de SQL:**
   - Para interactuar con la base de datos PostgreSQL, se importan clases como Connection, DriverManager, ResultSet, etc.
3. **Paquete Vector y DefaultTableModel:**
   - Vector: Almacena datos de forma dinámica para construir las filas y columnas de la tabla.
   - DefaultTableModel: Sirve para configurar el modelo de datos del componente JTable.

Atributos Principales

1. **pedidoTable:**
   - Un componente JTable donde se muestran los pedidos y sus detalles.

2. **Regresarbutt:**
    - Botón que permite regresar a una pantalla previa.

Constructor

El constructor de la clase realiza lo siguiente:

1. Llama a initComponents() para inicializar los componentes visuales.
2. Llama al método generarTable() para cargar los datos desde la base de datos y mostrarlos en el componente pedidoTable.

Inicialización de Componentes (initComponents)

Este método es generado automáticamente por el Editor de Formas de NetBeans. Crea los componentes visuales y define sus propiedades básicas, como tamaño, disposición y colores.

Método generarTable

Este es el método central de la funcionalidad. Realiza las siguientes tareas:

1. **Establecer Conexión con la Base de Datos:**
    - Se conecta a una base de datos PostgreSQL utilizando las credenciales especificadas.
2. **Cambiar el Esquema Activo:**
    - Usa la instrucción SQL SET search_path para definir el esquema que contiene las tablas necesarias.
3. **Ejecutar Consulta SQL:**
    - La consulta selecciona datos de las tablas Pedido y Detalle_Pedido mediante un JOIN.
4. **Obtener y Procesar Resultados:**
    - Los datos se estructuran en vectores para ser utilizados en el modelo de la tabla.
5. **Asignar Modelo al JTable:**
    - El modelo generado dinámicamente se aplica al componente pedidoTable.
6. **Manejo de Errores:**
    - Se captura cualquier excepción SQL y se muestra un mensaje de error al usuario mediante JOptionPane.

Acción del Botón Regresar

El botón Regresarbutt permite volver a una pantalla previa llamada PedidoF. Lo hace al instanciar esta clase, mostrar su ventana y cerrar la ventana actual.

Método main

El método principal inicializa la aplicación con el *look-and-feel* de Nimbus (si está disponible) y crea una instancia de la clase VerPedidos, configurándola como visible.

Clase VerProductos

Esta clase es una extensión de javax.swing.JFrame y representa la ventana principal donde se visualizan los productos almacenados en la base de datos.

Métodos Principales

1. **generarTable()**
    - Este método se encarga de cargar los datos de la tabla producto desde una base de datos PostgreSQL y mostrarlos en una tabla (JTable) en la interfaz gráfica.
    - **Pasos principales:**
        1. **Establece una conexión con la base de datos usando DriverManager.getConnection.**
        2. **Cambia el esquema activo a ciber_action utilizando SET search_path.**
        3. **Ejecuta una consulta SQL (SELECT * FROM producto) para recuperar los datos.**
        4. **Procesa los resultados:**
            - Obtiene los nombres de las columnas mediante ResultSetMetaData.
            - Crea un Vector para las filas y las columnas.
        5. **Asigna un DefaultTableModel al JTable para mostrar los datos.**
    - **Manejo de errores:**
        1. Si ocurre un error de SQL, muestra un mensaje de error en un cuadro de diálogo.
2. **actualizarProdActionPerformed()**
    - Este método se ejecuta cuando se presiona el botón "Actualizar".

- Verifica si se ha seleccionado un producto en la tabla y abre un formulario para actualizarlo.
- Pasos importantes:
    1. Obtiene el ID y otros valores del producto seleccionado en la tabla.
    2. Llama al constructor de una nueva ventana (ActualizarProducto) pasando estos valores como parámetros.

3. **borrarProdActionPerformed()**
    - Este método elimina un producto seleccionado de la base de datos.
    - **Pasos principales:**
        1. Verifica que se haya seleccionado una fila en la tabla.
        2. Pide confirmación al usuario para eliminar el producto.
        3. Ejecuta una consulta SQL DELETE para borrar el producto usando su ID.
        4. Refresca la tabla invocando nuevamente generarTable().
4. **VerInvActionPerformed()**
    - Abre una nueva ventana (AgregarAlInventario) que permite gestionar el inventario relacionado con los productos.
5. **RegresarbuttActionPerformed()**
    - Permite regresar a una ventana anterior (agregarProductos).

Componentes de la Interfaz

- Panel Principal (jPanel1): Contiene la tabla y el botón "Ver inventario".
- Panel Lateral (jPanel2): Contiene los botones:
    - "Regresar"
    - "Actualizar"
    - "Borrar"
- Tabla de Productos (ProductosTable): Se utiliza para visualizar los datos obtenidos de la base de datos.

Gestión de la Conexión a la Base de Datos

La conexión utiliza las siguientes credenciales:

- URL: jdbc:postgresql://localhost:5432/Papeleria

- Usuario: postgres
- Contraseña: 9656

Esquema utilizado: ciber_action.

Consideraciones de Seguridad

- Validación de Entrada: Se verifica si se ha seleccionado una fila antes de realizar operaciones como actualizar o eliminar.
- Gestión de Errores: Los errores SQL se manejan con cuadros de diálogo para notificar al usuario.

Método main

Define el punto de entrada de la aplicación, configurando el aspecto gráfico y creando una instancia de VerProductos

Clase VerProovedores

La clase VerProovedores pertenece al paquete papeleria y representa una ventana gráfica que permite visualizar, actualizar y eliminar proveedores desde una base de datos PostgreSQL. Se implementa utilizando la biblioteca Swing para la interfaz gráfica.

Componentes Principales:

1. Propiedades de la Interfaz:

- Panel Principal (jPanel1): Contiene todos los elementos de la interfaz.
- Tabla (ProvTable): Muestra los datos de los proveedores obtenidos desde la base de datos.
- Botones:
    - actualizarTable: Abre un formulario para actualizar el proveedor seleccionado.
    - borrarProv: Elimina al proveedor seleccionado tras una confirmación.
    - Regresarbutt: Regresa a la ventana principal.

---

2. Métodos Principales:

a) generarTable()

- Función: Pobla la tabla ProvTable con los datos obtenidos de la base de datos.
- Detalles de Implementación:

- Se conecta a la base de datos PostgreSQL usando el esquema ciber_action.
- Ejecuta una consulta SQL para obtener todos los registros de la tabla proveedor.
- Extrae los nombres de las columnas y las filas para construir un modelo de tabla (DefaultTableModel).
- Asigna el modelo a la tabla gráfica ProvTable.

b) borrarProv()

- Función: Elimina al proveedor seleccionado en la tabla.
- Detalles de Implementación:
  - Verifica que haya una fila seleccionada en ProvTable.
  - Obtiene el id_prov del proveedor desde la tabla.
  - Solicita confirmación al usuario antes de proceder.
  - Ejecuta una consulta SQL para eliminar el proveedor de la base de datos.
  - Actualiza la tabla tras la eliminación con una nueva llamada a generarTable().

c) actualizarTableActionPerformed()

- Función: Permite actualizar la información del proveedor seleccionado.
- Detalles:
  - Extrae los datos de la fila seleccionada en la tabla.
  - Abre una nueva ventana (actualizarProovedor) para editar la información.
  - **Cierra la ventana actual.**

d) RegresarbuttActionPerformed()

- Función: Cierra la ventana actual y regresa al menú principal (agregarProovedores).

---

3. Inicialización de Componentes:

- El constructor VerProovedores:
  - Llama al método initComponents() generado por el Editor de GUI.
  - Invoca generarTable() para mostrar los datos al inicio.

Consideraciones Técnicas:

1. Gestión de Base de Datos:
   - Usa un esquema específico ciber_action.

- Las conexiones y declaraciones (Connection, Statement, PreparedStatement) son cerradas para evitar fugas de recursos.

2. Interfaz Gráfica:
    - Utiliza el diseño absoluto (AbsoluteLayout) en jPanel1.
    - Los colores y estilos están definidos para una apariencia moderna.
3. Manejo de Excepciones:
    - Captura de excepciones SQL (SQLException) para manejar errores de conexión o consultas.
4. Validaciones:
    - Verifica que el usuario seleccione una fila antes de actualizar o eliminar.
    - Incluye confirmaciones para acciones críticas como eliminar proveedores.

Uso:

- Ejecutar la Aplicación: El método main lanza la ventana con los datos cargados.
- Funciones Disponibles:
    - Visualizar todos los proveedores.
    - Actualizar datos de un proveedor.
    - Eliminar un proveedor con confirmación.
    - Regresar al menú principal.

Clase ActualizarEmpleado

La clase actualizarEmpleado es una interfaz gráfica desarrollada en Java Swing para actualizar los datos de un empleado en una base de datos PostgreSQL. A continuación se describe en detalle su estructura y funcionalidad:

Propósito de la Clase

Esta clase proporciona una ventana para:

1. Mostrar los datos existentes de un empleado.
2. Permitir al usuario modificar información como el nombre, teléfono, correo electrónico y fecha de contratación.
3. Guardar los cambios actualizando los datos en la base de datos.

Atributos Principales

- idEmpleado: Identificador único del empleado a actualizar.

- Componentes Swing como nombretext, telefonotext, correotext, fechactext para introducir los datos del empleado.
- Botones como ActualizarEmp y Regresarbutt para ejecutar acciones específicas.

Constructores

1. actualizarEmpleado()
    - Inicializa los componentes gráficos.
    - Este constructor se utiliza si no se proporcionan datos iniciales.
2. actualizarEmpleado(int id, String nombre, String telefono, String correo, String fechac)
    - Recibe datos iniciales del empleado.
    - Asigna estos valores a los campos de texto y guarda el ID del empleado.

Métodos

1. actualizarDatoEmpleado()
    - Actualiza los datos del empleado en la base de datos.
    - Pasos principales:
        - Valida la fecha ingresada para garantizar el formato YYYY-MM-DD.
        - Conecta con la base de datos usando JDBC.
        - Ejecuta una consulta UPDATE para modificar los datos del empleado.
        - Muestra mensajes de confirmación o error según el resultado.
2. initComponents()
    - Generado automáticamente por el entorno de desarrollo (NetBeans).
    - Configura la interfaz gráfica, añadiendo etiquetas, campos de texto y botones.
3. RegresarbuttActionPerformed()
    - Navega de vuelta a la ventana principal o lista de empleados (VerEmpleados).
4. ActualizarEmpActionPerformed()
    - Llama al método actualizarDatoEmpleado cuando el usuario hace clic en el botón "Actualizar".

Flujo Principal

1. El usuario selecciona un empleado para actualizar.
2. La ventana se inicializa con los datos del empleado.
3. El usuario edita los campos y presiona "Actualizar".

4. El programa valida los datos y actualiza el registro en la base de datos.

Puntos Clave

- Validación de Datos: Garantiza que la fecha ingresada cumpla con el formato requerido antes de enviarla a la base de datos.
- Conexión con la Base de Datos: Cambia el esquema a ciber_action antes de realizar la operación.
- Manejo de Errores: Informa al usuario si ocurre un error en la conexión o durante la actualización.

Clase actualizarProovedor

Esta clase forma parte de un sistema de gestión de una papelería y está diseñada para actualizar la información de un proveedor en la base de datos. La interfaz gráfica de usuario (GUI) está desarrollada con **Swing** y permite modificar los datos del proveedor seleccionado.

Propiedades principales

1. **idProveedor**
   Un entero que almacena el ID del proveedor cuya información se va a actualizar. Este ID se utiliza para identificar al proveedor dentro de la base de datos.

Constructores

1. **actualizarProovedor()**
   Constructor por defecto que inicializa los componentes de la interfaz gráfica.
2. **actualizarProovedor(int id, String nombre, String telefono, String correo, String direccion, String rfc)**
   Constructor sobrecargado que permite prellenar los campos de la interfaz con los datos de un proveedor existente. Esto facilita la edición al mostrar la información actual.

Método principal

1. **actualizarDatoProveedor()**
   Este método realiza la actualización de los datos del proveedor en la base de datos.

Pasos clave del método:

- Conecta a la base de datos PostgreSQL usando DriverManager.
- Cambia el esquema activo a ciber_action usando un Statement.
- Ejecuta una sentencia SQL preparada para actualizar los datos del proveedor.

- Captura y maneja cualquier excepción relacionada con SQL, mostrando mensajes al usuario según corresponda.
- Muestra un mensaje de éxito o error según el resultado de la operación.

Componentes de la interfaz gráfica

1. **Panel principal (jPanel3)**
   Contiene todos los elementos gráficos, organizados en secciones:
   - **Etiquetas (JLabel)**: Muestran los nombres de los campos como "Nombre", "Teléfono", etc.
   - **Campos de texto (JTextPane)**: Permiten al usuario ingresar o modificar información.
   - **Botón "Actualizar" (ActualizarProv)**: Invoca el método actualizarDatoProveedor() cuando se hace clic.
   - **Botón "Regresar" (Regresarbutt)**: Cierra esta ventana y regresa al listado de proveedores.
2. **Diseño**
   Utiliza AbsoluteLayout para organizar los componentes con posiciones específicas.

Eventos importantes

1. **ActualizarProvActionPerformed**
   Llamado cuando el botón "Actualizar" es presionado. Este método invoca actualizarDatoProveedor().
2. **RegresarbuttActionPerformed**
   Llamado cuando el botón "Regresar" es presionado. Navega de regreso al listado de proveedores cerrando la ventana actual.

Ejecución principal

En el método main, se configura el *Look and Feel* de la interfaz gráfica y se inicializa el formulario. Este código se ejecuta en un hilo separado para mantener la interfaz responsiva.

Clase ActualizarProducto

Propósito

La clase ActualizarProducto proporciona una interfaz gráfica para actualizar los datos de un producto existente en una base de datos. Permite editar el nombre, la descripción, la categoría y el precio de un producto especificado por su ID.

Componentes principales

1. **Atributos de la clase**:
    - idProducto: Almacena el identificador único del producto que se va a actualizar.

2. **Constructores**:
    - **ActualizarProducto()**: Constructor por defecto que inicializa los componentes gráficos.
    - **ActualizarProducto(int id, String nombre, String descripcion, String categoria, String precio)**: Constructor sobrecargado que permite inicializar el formulario con los datos actuales de un producto específico.
3. **Métodos clave**:
    - **initComponents()**: Método generado automáticamente por el editor de interfaces gráficas de Java. Configura los componentes visuales de la ventana.
    - **actualizarDatoProducto()**:
        - Conecta con la base de datos PostgreSQL.
        - Cambia el esquema de trabajo al correspondiente.
        - Ejecuta una consulta SQL para actualizar el registro del producto especificado.
        - Maneja excepciones como errores de conexión o entrada inválida para el precio.
    - **Eventos**:
        - **ActProdActionPerformed()**: Llama al método actualizarDatoProducto() cuando el usuario presiona el botón "Actualizar".
        - **RegresarbuttActionPerformed()**: Abre el formulario VerProductos y cierra la ventana actual.

4. **Validaciones**:
    - Se asegura de que el precio sea un número válido (BigDecimal).
    - Controla errores de conexión a la base de datos y problemas con el ID del producto.
5. **Base de datos**:
    - Conexión a un esquema llamado ciber_action en la base de datos Papeleria.
    - Los parámetros de conexión (url, user, password) están embebidos en el código.

Funcionamiento General

1. El usuario abre el formulario de actualización desde otra sección de la aplicación.
2. Se cargan los datos actuales del producto en los campos correspondientes.
3. El usuario edita los valores deseados y pulsa el botón "Actualizar".
4. El sistema valida los datos y actualiza el registro en la base de datos.
5. Si la operación es exitosa, muestra un mensaje de confirmación; de lo contrario, informa del error.

Clase Fstock

La clase Fstock extiende javax.swing.JFrame y representa una interfaz gráfica (GUI) para gestionar el stock mínimo de productos en el inventario.

Propósito

Proporcionar una interfaz para:

1. Registrar nuevos productos junto con su stock mínimo.
2. Navegar de regreso a la pantalla de inventario.

Componentes Principales

1. **Paneles (jPanel1, jPanel2, jPanel3)**
    - Organizan la distribución y diseño de los elementos de la interfaz.
    - jPanel1: Contiene un encabezado con el título "Stock".
    - jPanel2: Panel principal donde se introducen los datos del producto y su stock mínimo.
    - jPanel3: Barra lateral con opciones de navegación y etiquetas descriptivas.
2. **Etiquetas (jLabel1, jLabel2, jLabel5, jLabel6)**
    - jLabel1: Texto "Stock mínimo".
    - jLabel2: Texto "Producto".
    - jLabel5: Texto "Datos".

- jLabel6: Texto del encabezado principal.

3. **Campos de texto**
    - Productotext: Permite al usuario ingresar el nombre del producto.
    - cantidadtext1: Permite al usuario definir el stock mínimo.
4. **Botones**
    - agregarIvn: Botón para añadir los datos ingresados.
    - Regresarbutt: Botón para regresar a la pantalla anterior.

Métodos

1. **Constructor Fstock()**
    - Llama al método initComponents() para inicializar los componentes de la interfaz gráfica.
2. **RegresarbuttActionPerformed**
    - Navega a la pantalla de inventario (AgregarAlInventario) y cierra la ventana actual.
3. **agregarIvnActionPerformed**
    - Método vacío destinado a implementar la lógica para añadir el producto y el stock mínimo.
4. **Método main**
    - Configura el aspecto visual de la aplicación con el *Look and Feel* Nimbus (si está disponible).
    - Inicializa y muestra la ventana de la clase Fstock.

Atributos

Los elementos de la interfaz gráfica están definidos como atributos privados, declarados automáticamente por el editor visual de NetBeans:

- Botones: Regresarbutt, agregarIvn.
- Etiquetas: jLabel1, jLabel2, jLabel3, jLabel5, jLabel6.
- Paneles: jPanel1, jPanel2, jPanel3.
- Campos de texto: Productotext, cantidadtext1.

Clase InventarioEmpleado

Componentes Principales

1. **Método cargarInventario**
   Este método conecta con la base de datos PostgreSQL, realiza una consulta para obtener

información del inventario (como el ID del producto, nombre, cantidad, precio y categoría) y muestra estos datos en una tabla (JTable) dentro de la interfaz gráfica.

- **Conexión a la base de datos:**
  Utiliza DriverManager.getConnection para establecer la conexión con las credenciales y URL proporcionadas.
- **Cambio de esquema:**
  Cambia el esquema activo a ciber_action con el comando SET search_path.
- **Consulta SQL:**
  Realiza una consulta INNER JOIN para combinar las tablas Inventario y Producto y obtener los datos necesarios.
- **Población de la tabla:**
  Crea un modelo (DefaultTableModel) a partir de los datos obtenidos y lo asigna al componente Tinventario.

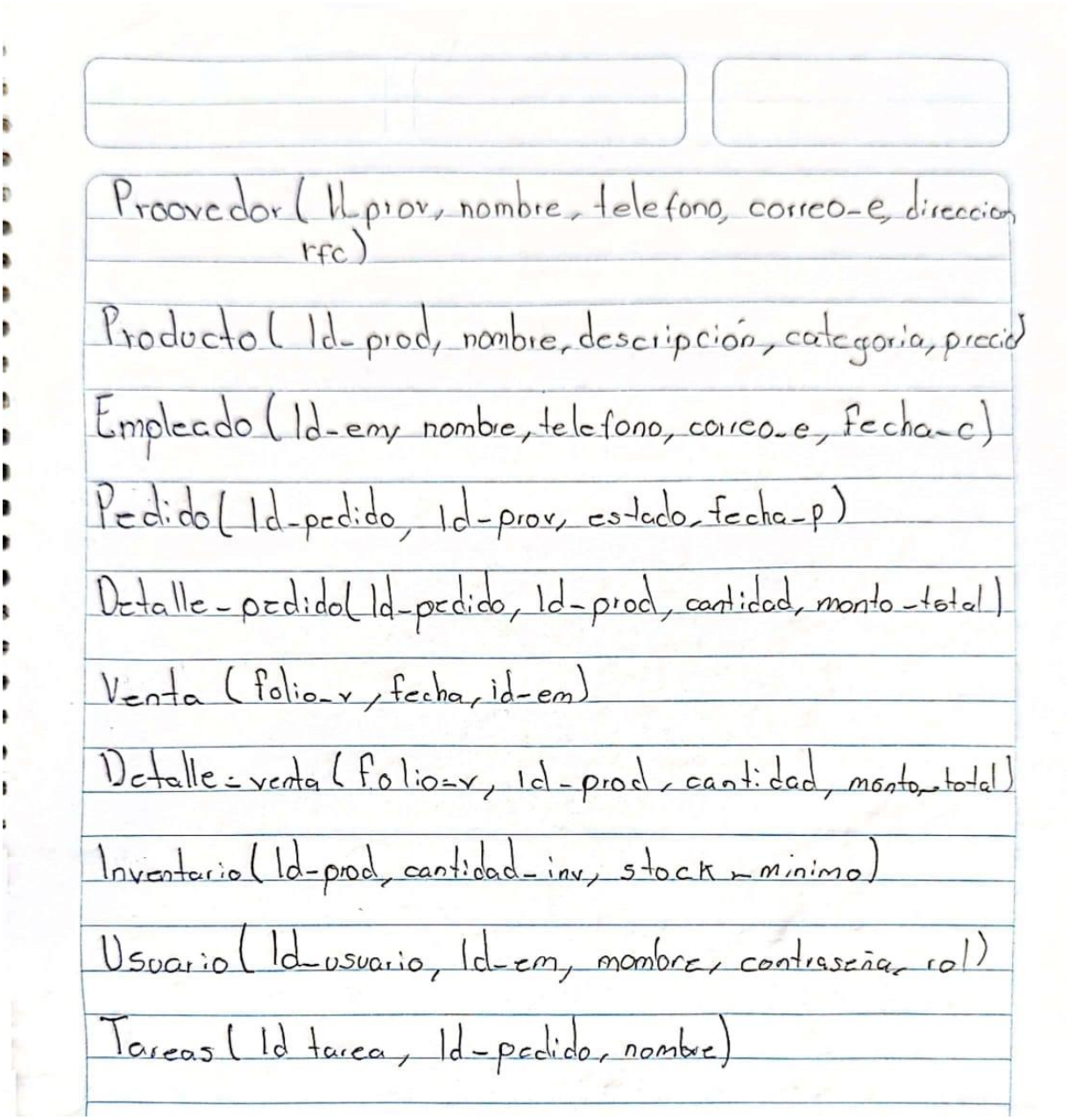
2. **Método actualizarInventario**
   Este método permite modificar la cantidad de un producto en el inventario.
   - **Consulta SQL de actualización:**
     Utiliza una consulta parametrizada (PreparedStatement) para actualizar la cantidad de un producto específico en la base de datos.
   - **Validación:**
     Muestra mensajes al usuario indicando si la operación fue exitosa o si el producto no se encontró.
3. **Interfaz Gráfica**
   La interfaz gráfica se divide en tres paneles:
   - **jPanel1:** Contiene un encabezado con el título "Inventario" y un botón para regresar al menú principal.
   - **jPanel2:** Contiene la tabla de inventario y un botón para agregar o modificar productos.
   - **jPanel3:** Panel contenedor general que organiza los elementos.
4. **Método agregarIvnActionPerformed**
   Este método se ejecuta al presionar el botón "AÑADIR". Solicita al usuario ingresar el ID del producto y la cantidad mediante cuadros de diálogo, y luego llama a los métodos actualizarInventario y cargarInventario para reflejar los cambios en la tabla.
5. **Gestión de Eventos**
   Se manejan eventos de clic en los botones mediante ActionListener. Por ejemplo:
   - Al presionar "Regresar", se cierra la ventana actual y se abre el menú principal (MenuEmpleado).
6. **Conexión con la Base de Datos**
   La interacción con la base de datos se realiza mediante Connection, Statement, y

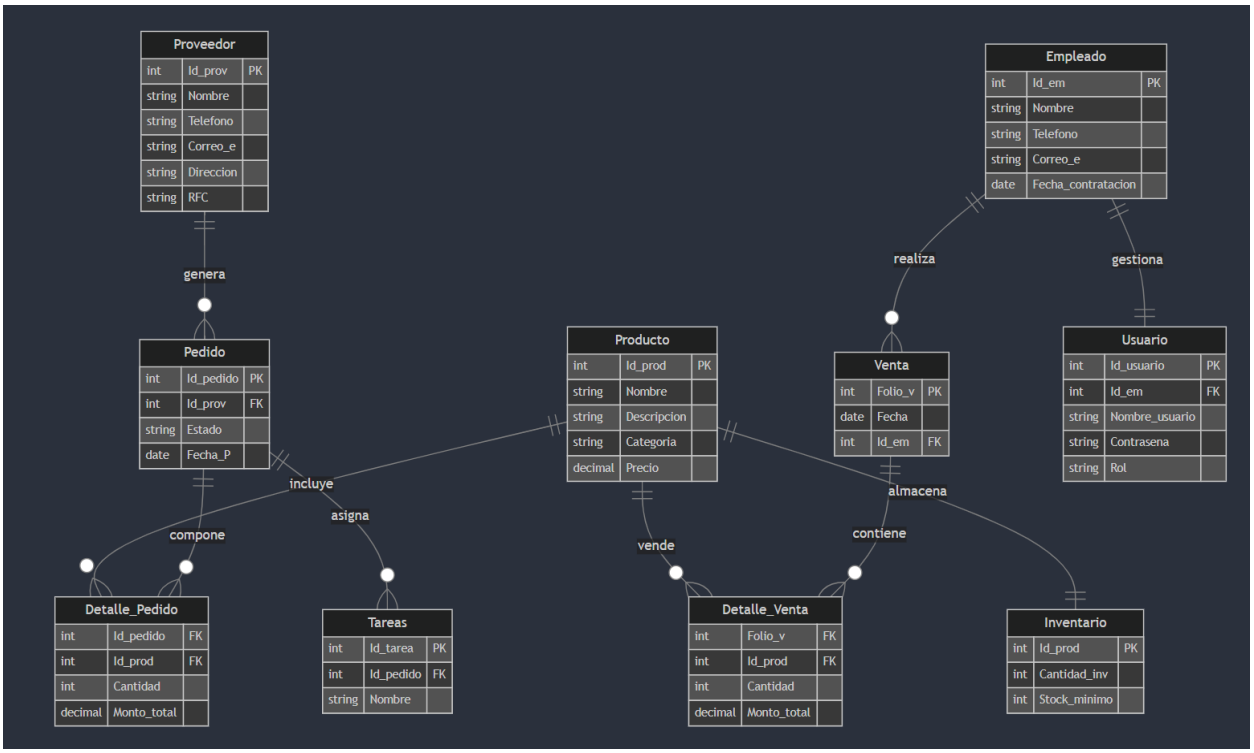
PreparedStatement, asegurando seguridad en las consultas parametrizadas y evitando ataques como inyecciones SQL.

7. **Clase Principal (main)**
   Configura el estilo visual (LookAndFeel) y lanza la ventana principal (InventarioEmpleado).

Apéndices

- Recursos adicionales como diagramas de flujo, diagramas de arquitectura, etc.




Proveedor
int Id_prov PK
string Nombre
string Telefono
string Correo_e
string Direccion
string RFC
genera
Pedido
int Id_pedido PK
int Id_prov FK
string Estado
date Fecha_P
incluye
asigna
compone
Detalle_Pedido
int Id_pedido FK
int Id_prod FK
int Cantidad
decimal Monto_total
Tareas
int Id_tarea PK
int Id_pedido FK
string Nombre
Producto
int Id_prod PK
string Nombre
string Descripcion
string Categoria
decimal Precio
vende
contiene
almacena
Detalle_Venta
int Folio_v FK
int Id_prod FK
int Cantidad
decimal Monto_total
Empleado
int Id_em PK
string Nombre
string Telefono
string Correo_e
date Fecha_contratacion
realiza
gestiona
Venta
int Folio_v PK
date Fecha
int Id_em FK
Usuario
int Id_usuario PK
int Id_em FK
string Nombre_usuario
string Contrasena
string Rol
Inventario
int Id_prod PK
int Cantidad_inv
int Stock_minimo