Universidad Tecnológica de Panamá

Facultad de Ingeniería de Sistemas Computacionales

Centro Regional de Panamá Oeste

Carrera Licenciatura en Desarrollo de Software

"Informe del Proyecto Final"

Asignatura

Desarrollo de Software

Nombre de los estudiantes

Rosario Cabrera
Brayan Rodríguez
Milagros Alonso
Artemio Rodríguez
Nobel De Gracia
Lohanys Mena
Jesús Claros
Jaime Rojas
Javier Chong
Iván Ovalles

Grupo

9LS121

Fecha de entrega

26 de julio del 2023

Periodo

I Semestre

2023

Índice

Introducción

En este informe del proyecto presentaremos y documentaremos el desarrollo del aplicativo de gestión de clientes y compras, desarrollado como proyecto final. Este proyecto representa una experiencia valiosa para poner en práctica los conocimientos teóricos adquiridos en el ámbito del desarrollo de software, permitiéndonos explorar y comprender los procesos involucrados en la creación de una solución informática integral y funcional.

El objetivo principal de este proyecto es construir un aplicativo integral que facilite la administración de clientes y compras para una empresa hipotética. A través de este proceso, aprenderemos cómo interactuar con una base de datos, crear una interfaz gráfica intuitiva y garantizar la práctica de habilidades fundamentales para cualquier desarrollador de software.

El aplicativo de gestión de clientes y compras tiene como finalidad ofrecer una solución práctica y eficiente para la administración de una empresa ficticia. La aplicación contempla una base de datos que almacena información detallada de los clientes, clasificándolos en tres categorías: Cliente VIP, Cliente Frecuente y Cliente Ocasional. Además, permite registrar y gestionar los productos disponibles para la venta y llevar un control exhaustivo de todas las compras realizadas por los clientes.

Es por lo que en este informe se generara una explicación de la lógica detrás de este proyecto y la estructuración del mismo.

ArchivosUsuarios Package

Archivo

package com.mycompany.ArchivosUsuarios; es el paquete que se está utilizando.

import com.mycompany.Entity.Usuariosdb; se importa la clase de Usuariosdb.

import java.io.File; import java.io.FileNotFoundException; se usan para trabajar con los ficheros.

import java.io.FileReader; import java.io.FileWriter; se usan para poder leer y escribir los ficheros.

import java.io.IOException; se usa para trabajar con las excepciones.

import java.util.Scanner; se usa para leer los datos dentro de la clase.

import java.util.logging.Level; import java.util.logging.Logger; se usa para registrar los datos.

import javax.swing.JOptionPane; se usa para mostrarnos ventanas de dialogo.

public class Archivo en esta clase se realiza la creación, lectura y el ingreso de datos de un archivo.

Comenzando con la creación de una variable ruta de tipo String, la cual le asignamos una propiedad de nuestra maquina con la llave “user.home”, y ubicándola en el escritorio de nuestra computadora.

public boolean crearArchivo() Se crea un método público de tipo booleano para crear el archivo donde utilizamos un **try,** en el try, se creó el archivo “milota”, asignándole como parámetro la variable ruta, para que al final nos devuelva el archivo, de lo contrario no se complete la acción anterior, se pasará al **catch** donde mandará una excepción “IOException” la cual nos comunicará que hubo un error.

public String leerArchivo (String nombre) se crea otro método para poder leer el archivo creado.

Se crea un file asignándole como parámetro la variable ruta, a su ves se crea un objeto "db". Se vuelve a utilizar el **try** donde con el scanner leemos el archivo anteriormente creado, se utiliza un bucle **while** con la condición de que mientras haya elementos que leer en nuestro archivo, guarde estos elementos en una variable de tipo String y a su ves con el método ".split" nos separe las cadenas de caracteres del archivo y los almacene en un array, dentro del while, se utiliza un bucle **for** con la condición de que si el tamaño del array es igual a 3 y el valor en la posición [2] del array es equivalente a la variable "nombre", entonces modifique el valor del "usuario" con el valor que este en la posición [0] del array; modifique el valor de la "contraseña" con el valor que este en la posición [1] del array; modifique el valor del "nombre" con el valor que este en la posición [2] del array (de Usuariosdb). Finalizando dichas acciones de no completarse nos mandará al **catch** con "Exception" comunicándonos que hubo un error.

Entity Package

ClienteOcasional

Esta es la una clase Padre de otras clases o subclases que se encuentran en el paquete Entity, en esta clase se aplicó la siguiente lógica:

1. Declaración de Variables de instancia

   En esta clase se declararon 9 variables de instancia, todas ellas siendo de tipo **'String',** esto para representar diferentes atributos de un cliente ocasional, como lo son su nombre, fecha de nacimiento, género, cédula, correo, teléfono, provincia, ciudad y corregimiento.

2. Constructor predeterminado 'ClienteOcasional()':

   Este es un constructor sin parámetros, el cual estamos utilizando para inicializar todas las variables de instancia con cadenas vacías (**""**).

3. Constructor con parámetros **‘ClienteOcasional(String nombre, String cedula, String genero, String fechaNacimiento, String telefono, String provincia, String ciudad, String corregimiento, String correo)’**:
   Este constructor acepta 9 parámetros que corresponden a los datos del cliente ocasional: **‘nombre’**, **‘cedula’**, **‘genero’**, **‘fechaNacimiento’**, **‘telefono’**, **‘provincia’**, ‘**ciudad’**, **‘corregimiento’** y **‘correo’**. Dentro del constructor, se asignan los valores proporcionados a las variables de instancia correspondientes. Además, hay una línea de código que intenta convertir el valor de teléfono que es **‘String’** a un número entero mediante **‘Integer.parseInt(telefono)’**.

4. Métodos getters y setters:
   También se agregaron los métodos getters y setters para accdesr y modificar las variables de instancia de forma controlada.

La clase **‘ClienteOcasional’** es una clase que representa a un cliente ocasional y almacena información relacionada con su identidad y ubicación. Los constructores permiten crear objetos de **‘ClienteOcasional’** con diferentes combinaciones de datos, y los métodos getters y setters ofrecen una forma segura de acceder y modificar los valores de sus atributos.

ClienteFrecuente

La clase **‘ClienteFrecuente’** se extiende de otra clase llamada **‘ClienteOcasional’**, a continuación, se explicará a detalle la estructuración de esta clase:

1. Declaración de variables de instancia:
   En esta clase se declararon cuatro variables de instancia para representar diferentes atributos de la clase cliente frecuente como los son: **‘numeroMembresia’** el cual es una variable tipo **‘int’**, **‘productoCompradoFrecuente’** este siendo una variable de tipo **‘String’ ‘cantidadPromedioGastado’** y **‘descuento’** siendo tipo **‘double’.**

2. Constructor con parámetros **‘ClienteFrecuente(int numeroMembresia, double cantidadPromedioGastado, String productoCompradoFrecuente, double descuento, String nombre, String cedula, String genero, String**

fechaNacimiento, String telefono, String provincia, String ciudad, String corregimiento, String correo)':

Este constructor acepta cuatro parámetros los cuales corresponden a los datos que específicamente solo posee un cliente frecuente, además de algunos datos que adicionales que hereda de la clase **'ClienteOcasional'**, utilizamos para la palabra clave **'super'** para llamar al constructor de la clase padre **'Cliente Ocasional'** y pasarle los parámetros a la clase **'nombre'**, **'cedula'**, **'genero'**, **'fechaNacimiento'**, **'telefono'**, **'provincia'**, **'ciudad'**, **'corregimiento'** y **'correo'**. Luego inicializamos las variables de instancia.

3. Métodos getters y setters:

   Se utilizan métodos getter y setters para para acceder y modificar las variables de instancia de forma controlada.

'ClienteFrecuente' es una clase que representa a un cliente frecuente y hereda características de la clase **'ClienteOcasional'**. Tiene la capacidad de almacenar información sobre el número de membresía, el promedio de gastos, el producto comprado frecuentemente y el descuento que se aplica al cliente en sus compras.

ClienteVIP

1. Importación de clases:

   En esta parte lo que se hizo es importar la clase **'IngresoDatos'** del paquete **'com.mycompany.HikariMetods.IngresoDatos.'** Esa clase contiene métodos relacionados con el ingreso de datos.

2. Declaración de Variables de instancia:

   Se declaran dos variables de instancia: **'asesor'** de tipo **'String'** y **'credito'** de tipo **'double'**. Estas variables representan el nombre del asesor asignado al cliente VIP y el crédito que tiene disponible el cliente VIP.

3. Constructor predeterminado **'ClienteVip()**':

Este constructor es un constructor sin parámetros, en este se inicializamos las variables de instancia **‘asesor’** y **‘credito’** con **‘String’** vacío (**“”**) y **‘double’** en **‘0.0’** correspondientemente.

4. Constructor con parámetros **‘ClienteVip(String asesor, double credito, int numeroMembresia, double cantidadPromedioGastado, String productoCompradoFrecuente, double descuento, String nombre, String cedula, String genero, String fechaNacimiento, String telefono, String provincia, String ciudad, String corregimiento, String correo)’**:
   En este constructor acepta una serie de parámetros para inicializar tanto los datos del cliente VIP como los datos heredados de la clase ClienteFrecuente. También Utiliza la palabra clave **‘super’** para llamar al constructor de la superclase **‘ClienteFrecuente’** y pasarle los parámetros correspondientes. Luego, asignamos los valores proporcionados a las variables de instancia **‘asesor’** y **‘credito’**.
5. Métodos getters y setters:
   Se utilizaron métodos getters y setters para acceder y modificar las variables de instancia de forma controlada.

La clase **‘ClienteVip’** es una subclase de la clase **‘ClienteFrecuente’** que representa a un cliente VIP y hereda características tanto de **‘ClienteFrecuente’** como de las clases superiores como lo es **‘ClienteOcasional’**. Tiene la capacidad de almacenar información sobre el asesor asignado al cliente y el crédito disponible para el cliente VIP.

Productos

1. Declaración de variables
   En esta clase se declararon cinco variables de instancia para representar diferentes atributos de la clase **‘Productos’** estas variables siendo: **‘nombrePro’** (tipo **‘String’**), **‘codigoBarra’** (tipo **‘String’**), **‘id’** (tipo **‘int’**), **‘precioStandart’** (tipo **‘double’**) y **‘precioDescuento’** (tipo **‘double’**).

2. Constructores:

   Constructor con parámetros **‘Productos(String nombrePro, String codigoBarra, double precioStandart, double precioDescuento)’**:

   Em este constructor se acepta cuatro parámetros que corresponden a los datos del producto: **‘nombrePro’**, **‘codigoBarra’**, **‘precioStandart’** y **‘precioDescuento’** y Asigna los valores proporcionados a las variables de instancia correspondientes.

3. Constructor predeterminado **‘Productos()’**:

   En este constructor sin parámetros, inicializamos las variables de instancia **‘nombrePro’** y **‘codigoBarra’** con cadenas vacías **("")**, y los valores numéricos **‘precioStandart’** y **‘precioDescuento’** con **‘0.0’**.

4. Métodos getters y setters:

   Utilizamos getters y setters para acceder y modificar las variables de instancia de forma controlada.

Compras

La clase Compras extiende la clase Productos. Esto significa que la clase Compras hereda todas las características y métodos de la clase Productos. Por lo tanto, la clase Compras tiene acceso a los atributos y métodos definidos en la clase Productos. A continuación, se explicará la lógica de esta clase:

1. Declaración de variables de instancia:

   En esta clase se declararon cinco variables de instancia para representar diferentes atributos de la clase **‘Compras’** como lo son: **‘numeroCompra’** (tipo **‘int’**) **‘costoTotal’** (tipo **‘double’**), **‘itbms’** (tipo **‘double’**), **‘fechaCompra’** (tipo **‘String’**) y **‘cantidadProducto’** (tipo **‘int’**).

2. Constructores con parámetros **‘Compras(int numeroCompra, double costoTotal, double itbms, String fechaCompra, int cantidadProducto, String nombrePro)’**:

En este constructor se cuenta con seis parámetros. Cinco de ellos corresponden a los datos de la clase **‘Compras’**, y el sexto, **‘nombrePro’**, se pasa a la clase base **‘Productos’** al invocar el constructor de la clase padre **‘Productos’** utilizando **‘super(nombrePro)’** para establecer el nombre del producto asociado a la compra.

3. **Métodos getters y setters:**
   Utilizamos métodos setters y getters para así poder acceder y modificar las variables de instancia de forma mas segura.

La clase **‘Compras’** representa una compra realizada y hereda de la clase padre **‘Productos’**. Tiene la capacidad de almacenar información sobre el número de compra, el costo total, el ITBMS, la fecha de compra y la cantidad de productos comprados en una transacción.

Ventas

La clase **‘Ventas’** extiende la clase **‘Productos’**. Esto significa que la clase **‘Ventas’** hereda todas las características y métodos de la clase **‘Productos’**. Por lo tanto, la clase **‘Ventas’** tiene acceso a los atributos y métodos definidos en la clase **‘Productos’**. A continuación, se explicará a detalle la estructuración de esta clase:

1. Declaración de Variables de Instancia:
   En esta clase se declararon tres variables de instancia para representar diferentes atributos de la clase estas siendo: **‘numeroVenta’** (tipo **‘int’**), **‘costoTotal’** (tipo **‘double’**), **‘itbms’** (tipo **‘double’**).

2. Constructor predeterminado **‘Ventas()’**:
   En este constructor sin parámetros inicializamos las variables de **‘numeroVenta’**, **‘costoTotal’** e **‘itbms’** con **‘0.0’**.

3. Constructor con parámetros **‘Ventas(int numeroVenta, double costoTotal, double itbms, String nombrePro, String codigoBarra, double precioStandart, double precioDescuento)’**:

Este constructor contiene siete parámetros. Tres de ellos corresponden a los datos de la clase **‘Ventas’** (**‘numeroVenta’**, **‘costoTotal’** e **‘itbms’**), y los otros invocando el constructor de la clase padre **‘Productos’** utilizando **‘super(nombrePro, codigoBarra, precioStandart, precioDescuento)’** para establecer los atributos heredados de Productos.

4. Métodos getters y setters:
   Utilizamos estos métodos para acceder y modificar las variables de instancia de forma controlada.

UsuariosDB

package com.mycompany.Entity; es el paquete que se esta utilizando para esta clase.

public class Usuariosdb Se inicia la clase “Usuariosdb” donde al principio se crean 2 variables privadas static de tipo String (usuario y contraseña) y una última variable privada también de tipo String (nombre).

public static String getUsuario() se crea un método público static en el cual nos devuelve el usuario retornándonos dicho valor.

public static void setUsuario(String usuario) un nuevo método público static que tiene como parámetro el usuario, en el cual se modifica el usuario asignándolo con la clase.

public static String getContraseña() se crea un método público static en el cual nos devuelve la contraseña retornándonos dicho valor.

public static void setContraseña(String contraseña) un nuevo método público static que tiene como parámetro la contraseña, en el cual se modifica la contraseña asignándola con la clase.

public String getNombre() se crea un método público en el cual nos devuelve el nombre retornándonos dicho valor.

public void setNombre(String nombre) un nuevo método público que tiene como parámetro el nombre, en el cual se modifica el nombre.

Hikari CP Package

ConexionHikari

package com.mycompany.HikariCP; es el paquete en donde se encuentra nuestra clase.

import com.mycompany.Entity.Usuariosdb; se importa la clase de Usuariosdb del paquete Entity.

import com.zaxxer.hikari.HikariConfig;

import com.zaxxer.hikari.HikariDataSource; se usan para poder realizar la conexión hikari.

import java.sql.Connection; se usa para la conexión con la base de datos.

import java.sql.SQLException; trabaja las excepciones relacionadas con sql la base de datos.

public class ConexionHikari es la clase donde se realiza la conexión a través de hikari con la base de datos.

Primero creando un método privado static final para realizar la configuración de la conexión.

Luego se creará la instancia del “datasource”.

static se crea un bloque static para que se inicialice una vez sea corrido el programa.

Colocando la dirección url de nuestra base de datos.

Colocando el atributo usuario que se ha recogido de la clase Usuariodb.

Colocando el atributo contraseña que se ha recogido de la clase Usuariodb.

Luego se le añade ciertas propiedades a la configuración.

Y se guardan las configuraciones en la variable “DataS”.

public static Connection getConnection() se crea un método público static que nos retorna los valores dentro del “DataS”.

HikariMetods Package

IngresoDatos

1. Método **ingreso:**
   - Este método se encarga de insertar un nuevo cliente en la base de datos.
   - Primero, se establece una conexión a la base de datos utilizando la clase **ConexionHikari**.
   - Luego, se prepara una consulta SQL para insertar un nuevo registro en la tabla **clientes**.
   - Los datos necesarios para el nuevo cliente, como su tipo, nombre, fecha de nacimiento, género, correo, etc., se asignan a los parámetros de la consulta SQL.
   - Además, este método realiza consultas adicionales a través de un objeto de la clase **Consultas** para obtener información adicional sobre el cliente, como el promedio gastado y los productos frecuentes. Estos datos también se asignan a los parámetros de la consulta SQL.
   - Después de asignar todos los valores, se ejecuta la consulta y el nuevo cliente se guarda en la base de datos.
   - Finalmente, se muestra una ventana emergente utilizando **JOptionPane**, informando que se ha guardado el cliente con éxito, junto con su nombre y tipo de cliente.

2. Método **ingresoProducto**:
   - Este método se encarga de insertar un nuevo producto en la base de datos.
   - Al igual que en el método **ingreso**, primero se establece una conexión a la base de datos utilizando la clase **ConexionHikari**.
   - Luego, se prepara una consulta SQL para insertar un nuevo registro en la tabla **Productos**.
   - Los datos necesarios para el nuevo producto, como su nombre, código de barras, precios, etc., se asignan a los parámetros de la consulta SQL.

- Después de asignar los valores, se ejecuta la consulta y el nuevo producto se guarda en la base de datos.
- Se muestra un mensaje emergente utilizando **JOptionPane**, informando que se ha guardado el producto con éxito, junto con su nombre.

3. Método **ingresoCompra**:
   - Este método se encarga de insertar una nueva compra en la base de datos.
   - Al igual que en los métodos anteriores, primero se establece una conexión a la base de datos utilizando la clase **ConexionHikari**.
   - Se prepara una consulta SQL para insertar un nuevo registro en la tabla **Compras**.
   - Los datos necesarios para la nueva compra, como el número de compra, cédula del cliente, nombre del producto, cantidad, costo total, etc., se asignan a los parámetros de la consulta SQL.
   - Después de asignar los valores, se ejecuta la consulta y la nueva compra se guarda en la base de datos.
   - Se muestra un mensaje emergente utilizando **JOptionPane**, informando que se ha guardado la compra con éxito, junto con el nombre del producto comprado.

La clase **IngresoDatos** se encarga de realizar operaciones de inserción en la base de datos utilizando consultas SQL. Los métodos de esta clase se ocupan de establecer la conexión a la base de datos, preparar las consultas con los datos correspondientes y ejecutarlas para guardar la información en la base de datos. Además, se utilizan mensajes emergentes para notificar al usuario sobre el éxito de las operaciones de inserción.

ActualizarDatos

1. Método '**actualizarDatos(String tipo, ClienteVip cliente)':**
   Este método se utiliza para actualizar la información de un cliente VIP en la base de datos. Este recibe dos parámetros **'tipo'** y **'ClienteVIP'**. En esta parte establecemos una conexión a la base de datos utilizando la clase **'ConexionHikari'**, además de preparar una SQL para actualizar los datos del cliente en la tabla clientes de la base de datos, asignamos los valores correspondientes del objeto cliente a los parámetros

de la consulta SQL, ejecutamos la consulta y actualiza los datos en la base de datos y cerramos la conexión a la base de datos después de completar la operación.

2. Método **'actualizarProducto(Productos producto)'**:
   Este método se utiliza para actualizar la información de un producto en la base de datos. Este recibe un parámetro 'producto' y establecemos una conexión a la base de datos utilizando la clase **'ConexionHikari'**, preparamos una consulta SQL para actualizar los datos del producto en la tabla Productos de la base de datos, asignamos los valores correspondientes del objeto producto a los parámetros de la consulta SQL, ejecutamos la consulta y actualizamos los datos en la base de datos, para así cerrar la conexión a la base de datos después de completar la operación.

3. Método **'actualizarCompra(String cedula, Compras venta)'**:
   Utilizamos este método para actualizar la información de una compra en la base de datos. Este recibe dos parámetros **'cedula'** y **'venta'**, también establecimos una conexión a la base de datos utilizando la clase ConexionHikari, preparamos una consulta SQL para actualizar los datos de la compra en la tabla compras de la base de datos, asignamos los valores correspondientes del objeto venta y la variable cedula a los parámetros de la consulta SQL para luego ejecutar la consulta y actualizar los datos en la base de datos y por ultimo cerrar la conexión a la base de datos después de completar la operación.

4. Método **actualizarOperacion() throws SQLException**:
   Utilizamos este método para actualizar la información relacionada con las operaciones de clientes que no son ocasionales, hicimos una instancia de la clase Consultas, establecimos una conexión a la base de datos utilizando la clase **'ConexionHikari'**, preparamos una consulta SQL para obtener el número de cédula de todos los clientes en la tabla clientes se ejecuta la consulta y obtiene el resultado en forma de un **'ResultSet'**. Iteramos sobre los resultados y para cada número de cédula, preparamos una consulta SQL para actualizar los datos del cliente en la tabla clientes de la base de datos, específicamente los campos

‘cantidad_promedio_gastada’ y **‘productos_frecuentes’**, ejecutamos la consulta para actualizar los datos del cliente en la base de datos y cerramos la conexión a la base de datos después de completar la operación.

La clase **‘ActualizarDatos’** proporciona métodos para actualizar información relacionada con clientes, productos y compras en una base de datos. Cada método establece una conexión a la base de datos, prepara y ejecuta consultas SQL para realizar las actualizaciones, y luego cierra la conexión.

Consultas

1. Método **Productosfrecuentes**:
   - Este método recibe como parámetro la cédula del cliente.
   - Se establece una conexión a la base de datos utilizando la clase **ConexionHikari**.
   - Se prepara una consulta SQL que busca el producto más frecuente comprado por el cliente con la cédula proporcionada.
   - La consulta calcula la cantidad máxima de productos comprados para cada producto y luego selecciona aquel producto que tenga la máxima cantidad comprada.
   - La consulta utiliza una función de agregación **MAX** junto con la cláusula **GROUP BY** para obtener el producto más frecuente.
   - Se ejecuta la consulta y se obtiene el resultado en un objeto **ResultSet**.
   - El nombre del producto más frecuente se almacena en el atributo **producto**.
   - El método retorna el nombre del producto más frecuente.

2. Método **PromedioGastado**:
   - Este método recibe como parámetro la cédula del cliente.
   - Al igual que en el método anterior, se establece una conexión a la base de datos utilizando la clase **ConexionHikari**.
   - Se prepara una consulta SQL que calcula el promedio del costo total de todas las compras realizadas por el cliente con la cédula proporcionada.
   - La consulta utiliza la función de agregación **AVG** para calcular el promedio.
   - Se ejecuta la consulta y se obtiene el resultado en un objeto **ResultSet**.

- El promedio gastado se almacena en el atributo **monto**.
- El método retorna el promedio gastado.

3. Método **ConsultarCliente**:
   - Este método recibe parámetros por la que se va a realizar la consulta, el nombre de la tabla, y el valor con el que se desea consultar.
   - Se establece una conexión a la base de datos utilizando la clase **ConexionHikari**.
   - Se prepara una consulta SQL dinámica que busca todos los registros de la tabla especificada donde el valor en la columna especificada coincida con el valor proporcionado.
   - Se ejecuta la consulta y se obtiene el resultado en un objeto **ResultSet**.
   - Los datos de cada registro encontrado se agregan a la lista **datos**.
   - El método retorna una lista con los datos de los registros encontrados en la base de datos.
4. Método **ConsultarCompra**:
   - Este método es similar al método **ConsultarCliente**, pero se utiliza para consultar compras específicas en la tabla **Compras**.
   - Recibe como parámetros el nombre de la tabla (**tabla**) y el valor con el que se desea consultar (**valor**), que en este caso sería el número de compra.
   - Se establece una conexión a la base de datos utilizando la clase **ConexionHikari**.
   - Se prepara una consulta SQL dinámica que busca todos los registros de la tabla **Compras** donde el número de compra coincida con el valor proporcionado.
   - Se ejecuta la consulta y se obtiene el resultado en un objeto **ResultSet**.
   - Los datos de cada registro encontrado se agregan a la lista **datos**.
   - El método retorna una lista con los datos de las compras encontradas en la base de datos.
5. Método **ConsultarProductos**:
   - Este método es similar a los métodos anteriores, pero se utiliza para consultar productos específicos en la tabla **Productos**.

- Recibe como parámetros el nombre de la tabla (**tabla**) y el valor con el que se desea consultar (**valor**), que en este caso sería el **ID** del producto.
- Se establece una conexión a la base de datos utilizando la clase **ConexionHikari**.
- Se prepara una consulta SQL dinámica que busca todos los registros de la tabla Productos donde el ID del producto coincida con el valor proporcionado.
- Se ejecuta la consulta y se obtiene el resultado en un objeto **ResultSet**.
- Los datos de cada registro encontrado se agregan a la lista **datos**.
- El método retorna una lista con los datos de los productos encontrados en la base de datos.

La clase **Consultas** proporciona métodos para consultar información específica de la base de datos. Estos métodos se encargan de establecer la conexión, preparar y ejecutar las consultas SQL y obtener los resultados de la base de datos para retornarlos en forma de listas con los datos encontrados.

Eliminar Compra

1. Método **eliminar:**

- Este método recibe como parámetro la cédula de un cliente que se desea eliminar de la tabla **Clientes**.
- Se establece una conexión a la base de datos utilizando la clase **ConexionHikari**.
- Se prepara una consulta SQL para eliminar el registro correspondiente al cliente con la cédula proporcionada en la tabla **Clientes**.
- Se ejecuta la consulta y se elimina el registro de la base de datos.
- El método al final imprime "Se borró." en la consola como mensaje informativo.

2. Método **eliminarProducto**:

- Este método es similar al método anterior, pero se utiliza para eliminar productos de la tabla **Productos**.
- Recibe como parámetro el ID del producto que se desea eliminar.
- Se establece una conexión a la base de datos utilizando la clase **ConexionHikari**.
- Se prepara una consulta SQL para eliminar el registro correspondiente al producto con el ID proporcionado en la tabla **Productos**.

- Se ejecuta la consulta y se elimina el registro de la base de datos.
- El método al final imprime "Producto eliminado." en la consola como mensaje informativo.

3. Método **EliminarCompra:**

- Este método es similar a los métodos anteriores, pero se utiliza para eliminar compras de la tabla **Compras**.
- Recibe como parámetro el número de compra que se desea eliminar.
- Se establece una conexión a la base de datos utilizando la clase **ConexionHikari**.
- Se prepara una consulta SQL para eliminar el registro correspondiente a la compra con el número proporcionado en la tabla **Compras**.
- Se ejecuta la consulta y se elimina el registro de la base de datos.
- El método al final imprime "Compra eliminada." en la consola como mensaje informativo.

La clase **EliminarDatos** proporciona métodos para eliminar registros específicos de las tablas Clientes, Productos y Compras en la base de datos.

Main Package

Semestral

Esta clase crea una instancia de la clase Archivo y verifica si un archivo ya existe. Si el archivo no existe, lo crea y permite al usuario ingresar datos en él. Luego, crea una ventana de la clase Usuarios y la muestra en el centro de la pantalla. Teniendo la siguiente estructura:

- Se crea un objeto de la clase Archivo.
- Se verifica si se pudo crear un archivo. Si es así, se permite al usuario ingresar datos en él.
- Si el archivo ya existe, se imprime un mensaje en la consola.
- Se crea un objeto de la clase Usuarios y se muestra su ventana en el centro de la pantalla.

Vistas Package

Principal

1. Inicialización de componentes:
   - La clase **Principal** hereda de **JFrame**, lo que significa que representa una ventana.
   - En el constructor **Principal()**, se inicializan y configuran todos los componentes visuales que se utilizarán en la ventana.

2. Componentes de la ventana:
   - La ventana contiene una serie de botones y etiquetas que forman parte del menú principal.
   - Los botones tienen diferentes acciones asociadas, como **"CONSULTAS"**, **"GUARDAR COMPRAS"**, **"REGISTRAR CLIENTE"**, **"SALIR"** y **"REGISTRAR PRODUCTO"**.
   - También incluimos una etiqueta de título **"MENU PRINCIPAL"** y una imagen de fondo y logotipo que se muestran en la ventana.

3. Acciones de los botones:
   - Cada botón tiene un método de acción asociado que se ejecutará cuando el botón sea presionado.
   - Por ejemplo, cuando el botón **"GUARDAR COMPRAS"** se presiona, se ejecuta el método **btn_GuardarComprasActionPerformed(evt)**.
   - En cada método de acción, se realiza una acción específica, como abrir una nueva ventana, cerrar la ventana actual o realizar una actualización en la base de datos.

4. Método **initComponents()**:
   - Este método se encarga de inicializar todos los componentes gráficos y configurar su apariencia, posición y acciones.
   - Se utiliza el **GroupLayout** para organizar los componentes en la ventana de manera ordenada.

5. Acciones de los botones:
   - Los métodos **btn_GuardarComprasActionPerformed**, **btn_RegistrarClienteActionPerformed**, **btn_SalirPrincipalActionPerformed**, **btn_RegistrarProductosActionPerformed** y **btn_ConsultasActionPerformed** son los controladores de eventos para los botones mencionados anteriormente.
   - Cada método realiza una acción específica cuando se presiona el botón correspondiente, como abrir nuevas ventanas relacionadas con las opciones seleccionadas.

6. Método **jButton1ActionPerformed()**:
   - Este método es el controlador de eventos para el botón "actualizar bd".
   - Cuando se presiona este botón, se llama al método actualizarOperacion() de la clase ActualizarDatos, que está fuera del alcance del código proporcionado.

La clase **Principal** define la ventana principal de una interfaz gráfica con un menú que contiene diferentes opciones y botones.

TiposCliente

1. Inicialización de componentes:
   - La clase **TiposCliente** hereda de JFrame, lo que significa que representa una ventana.
   - En el constructor **TiposCliente()**, se inicializan y configuran todos los componentes visuales que se utilizarán en la ventana.

2. Componentes de la ventana:
   - La ventana contiene tres botones: **"CLIENTE VIP"**, **"CLIENTE OCASIONAL"** y **"CLIENTE FRECUENTE"**.
   - También hay tiene de título **"ELIGA EL TIPO DE CLIENTE A REGISTRAR"** y una imagen de fondo.

3. Acciones de los botones:

- Cada botón tiene un método de acción asociado que se ejecutará cuando el botón sea presionado.
- Los métodos de acción son **btn_ClienteVIPActionPerformed**, **btn_ClienteOcasionalActionPerformed**, **btn_SalirTipoClienteActionPerformed** y **btn_ClienteFrecuenteActionPerformed**.
- Cuando se presiona un botón, se crea una nueva instancia de la clase **DatosCliente**, y se pasa el nivel del cliente seleccionado como parámetro.
- La ventana **DatosCliente** se muestra al usuario, permitiendo así el registro de los datos del cliente.

4. Método **initComponents()**:
   - Este método se encarga de inicializar todos los componentes gráficos y configurar su apariencia, posición y acciones.
   - Se utiliza el **GroupLayout** para organizar los componentes en la ventana de manera ordenada.

5. Getter y Setter de **nivelCliente**:
   - La clase tiene un atributo **nivelCliente** que se utiliza para almacenar el nivel de cliente seleccionado por el usuario.
   - Se proporcionan métodos **getNivelCliente()** y **setNivelCliente()** para acceder y modificar el valor de **nivelCliente**, respectivamente.

6. Métodos de acción de los botones:
   - Cada método de acción de los botones **btn_ClienteVIPActionPerformed**, **btn_ClienteOcasionalActionPerformed**, **btn_SalirTipoClienteActionPerformed** y **btn_ClienteFrecuenteActionPerformed** realiza lo siguiente:
     - Configura el valor de **nivelCliente** con el nivel correspondiente al tipo de cliente seleccionado.
     - Crea una nueva instancia de **DatosCliente** y la muestra al usuario para que pueda registrar los datos del cliente.

- En el caso del "**CLIENTE OCASIONAL"** y **"CLIENTE FRECUENTE"**, también muestra un panel adicional con el método **dc.panelXtra.setVisible(true);** que permite al usuario ingresar información adicional relacionada con el cliente ocasional o frecuente.

7. Método **btn_SalirTipoClienteActionPerformed:**
   - Este método se ejecuta cuando el botón **"VOLVER"** es presionado.
   - Crea una nueva instancia de la clase **Principa**l y la muestra al usuario, lo que permite volver al menú principal.

La clase **TiposCliente** define una ventana de interfaz gráfica que muestra al usuario tres opciones para elegir el tipo de cliente a registrar y dependiendo del tipo de cliente seleccionado, se abre una nueva ventana **DatosCliente** que permite al usuario registrar los datos del cliente.

Usuarios

1. Inicialización de componentes:
   - La clase **Usuarios** hereda de **JFrame**, lo que significa que representa una ventana.
   - En el constructor **Usuarios()**, se inicializan y configuran todos los componentes visuales que se utilizarán en la ventana.

2. Componentes de la ventana:
   - La ventana contiene un combo box **JComboBox** llamado **comboUser,** que muestra tres opciones: **"Administrador"**, **"Mario"** y **"usuarioRoot"**.
   - También hay dos botones: **"ACEPTAR"** y **"SALIR"**.
   - Los botones tienen métodos de acción asociados: **btnAceptarActionPerformed** y **btnSalirActionPerformed**.

3. Acción del botón **"ACEPTAR" (btnAceptarActionPerformed)**:
   - Cuando se hace clic en el botón **"ACEPTAR"**, se ejecuta este método de acción.

- Se crea una instancia de la clase **Archivo** y se llama al método **leerArchivo** pasando como parámetro el usuario seleccionado en el combo box **(String)this.comboUser.getSelectedItem().**
- Después de leer el archivo, se crea una nueva instancia de la clase **Principal** se muestra al usuario. Al mismo tiempo, se cierra la ventana actual **this.dispose().**

4. Acción del botón **"SALIR" (btnSalirActionPerformed)**:
   - Cuando se hace clic en el botón **"SALIR",** se ejecuta este método de acción.
   - La ventana actual **Usuarios** se cierra **this.dispose()**.

5. Método main:
   - El método main es el punto de entrada del programa. Establece el aspecto visual de la interfaz gráfica.
   - Se crea una instancia de la clase **Usuarios** y se muestra al usuario utilizando **EventQueue.invokeLater(new Runnable() { ... }**, lo que garantiza que se ejecute en el hilo de eventos de la interfaz gráfica para evitar problemas de concurrencia.

DatosCliente

1. Variables de instancia:

   La clase contiene varias variables de instancia que representan diferentes componentes de GUI, como botones, etiquetas, campos de texto, cajas de selección, etc. Estas variables se utilizan para crear e interactuar con los elementos gráficos.

2. Constructor:
   - El constructor **DatosCliente(int nivelClienteF)** inicializa los componentes de la GUI y configura el estado inicial de la ventana.
   - El método **initComponents()** se llama desde el constructor, el cual configura el diseño y la apariencia de los componentes de la GUI.

3. Componentes de la GUI:

- Los componentes de la GUI están organizados utilizando el gestor de diseño **GroupLayout**.
- Se agregan etiquetas, campos de texto y cajas de selección al panel principal **jPanel1**, que forma el núcleo de la ventana.
- La tabla **TablaCliente** se agrega a un panel de desplazamiento y se coloca en la ventana.

4. Aciones y Listeners:
   - Hay varios acciones listeners adjuntas a diferentes componentes de GUI (por ejemplo, botones, campos de texto, cajas de selección).
   - Estos listeners responden a las acciones del usuario, como hacer clic en un botón o seleccionar un elemento de una caja de selección, y activan acciones específicas definidas en sus métodos de manejo de eventos correspondientes.

5. Diseño:
   - Se utiliza **GroupLayout** para organizar y gestionar la disposición de los componentes de la GUI, asegurándose de que se muestren correctamente en la ventana.
   - Los componentes de la GUI se organizan en filas y columnas, y el diseño especifica cómo deben agruparse y alinearse.

6. **btn_SalirDatosClienteActionPerformed(ActionEvent evt):**
   - Este método se activa cuando se hace clic en el botón **"Salir".**
   - Su función es cerrar la ventana actual usando **this.dispose()**.
   - Luego, crea una instancia de la clase **TiposCliente**, que es otra ventana, y la muestra con **tc.setVisible(true)**.
   - Finalmente, la ventana se ubica en el centro de la pantalla con **tc.setLocationRelativeTo((Component)null).**

7. **btn_guardarDatosClienteActionPerformed(ActionEvent evt):**
   - Este método se activa cuando se hace clic en el botón **"Guardar"**.

- Su función es llamar al método **guardar()**.
- Después, se limpia la tabla **this.limpiarTabla(this.modelo)** y se vuelve a cargar la lista de clientes con el método **listar()**.

8. **listar():**
   - Este método se encarga de cargar los datos de los clientes desde la base de datos a la tabla de la interfaz gráfica.
   - Primero, se crea una consulta SQL para seleccionar todos los registros de la tabla **"clientes"**.
   - Luego, se establece una conexión a la base de datos, se ejecuta la consulta y se obtiene un conjunto de resultados **ResultSet**.
   - A continuación, se recorre el conjunto de resultados y se agregan los datos de cada cliente a la tabla mediante **this.modelo.addRow(persona)**.

9. **limpiarTabla(DefaultTableModel model):**
   Este método recibe un modelo de tabla **DefaultTableModel** como parámetro y se encarga de eliminar todas las filas de la tabla.

10. **BuscarDatosCliente():**
    Este método crea y devuelve un objeto **ClienteVip** que contiene los datos del cliente que se ingresaron en los campos de la interfaz gráfica.

11. **guardar():**
    - Este método utiliza la clase **IngresoDatos** para guardar los datos del cliente en la base de datos.
    - Primero, se llama a **BuscarDatosCliente()** para obtener el objeto **ClienteVip** con los datos ingresados.
    - Luego, utiliza un switch para determinar el nivel del cliente **nivelClienteF** y llama al método **ingreso()** de la clase **IngresoDatos,** pasando el tipo de cliente correspondiente y el objeto **ClienteVip**.

- Finalmente, se llama al método **listar()** para volver a cargar la lista de clientes en la tabla.

12. **actualizar():**

    Este método es similar a **guardar()**, pero utiliza la clase **ActualizarDatos** para actualizar los datos del cliente en la base de datos en lugar de insertar nuevos registros.

13. **eliminar():**

- Este método utiliza la clase **EliminarDatos** para eliminar los datos de un cliente de la base de datos.
- Obtiene el número de cédula del cliente desde el campo correspondiente, y luego llama al método **eliminar(cedulax)** de la clase **EliminarDatos** para eliminar el registro.

14. **TablaClienteMouseClicked(MouseEvent evt):**

- Este método se activa cuando el usuario hace clic en una fila de la tabla.
- Recupera el número de cédula del cliente seleccionado y utiliza esta información para buscar los datos completos del cliente en la base de datos.
- Luego, rellena los campos de la interfaz gráfica con los datos recuperados para que el usuario pueda ver y editar la información del cliente seleccionado.

15. **btn_eliminarDatosClienteActionPerformed(ActionEvent evt):**

- Este método se activa cuando se hace clic en el botón **"Eliminar".**
- Llama al método **eliminar()** para eliminar el cliente seleccionado, luego limpia la tabla y recarga la lista de clientes.

16. **boton_actualizarActionPerformed(ActionEvent evt):**

- Este método se activa cuando se hace clic en el botón **"Actualizar"**.
- Llama al método **actualizar()** para actualizar los datos del cliente seleccionado, luego limpia la tabla y recarga la lista de clientes.

17. **btn_eliminarDatosCliente1ActionPerformed(ActionEvent evt):**
    - Este método se activa cuando se hace clic en el botón **"Limpiar".**
    - Su función es simplemente restablecer todos los campos de entrada a sus valores predeterminados o vaciarlos.

DatosCompra

La clase **DatosCompra** extiende **JFrame**, lo que indica que es una ventana de GUI.

1. Variables de Instancia:
    - **modelo**: Un **DefaultTableModel** utilizado para mostrar datos en una **JTable**.
    - **preciosProductosMap**: Un **Map** para almacenar nombres de productos como claves y sus precios correspondientes como valores.
    - También hay otras variables como contador, **precioTotalAcumulado**, **contadorAgregados**, **itbms**, **precioTotal**, y listas como **productosList.**

2. Componentes de la Interfaz Gráfica (GUI):
   La clase define varios componentes de GUI, como botones **btn_anadir**, **jButton1**, **jButton2**, etc., etiquetas **jLabel1**, **jLabel2**, etc., cuadros combinados (**comboxproductos**), campos de texto (**text_cedulaCliente**, **text_fechaCompra**, etc.), y una tabla (**tablaCompra**) para mostrar datos.

3. Constructor **DatosCompra()**:
   El constructor inicializa los componentes de la GUI, configura listeners de acciones y llama al método **listar()** para poblar la tabla con datos de la base de datos.

4. Métodos de Manejo de acciones:
    - La clase define varios métodos con prefijos **btn_**, **jButton_**, o **combox_**, que corresponden a las acciones de los botones y cuadros combinados y sus respectivos manejadores de acciones.
    - Estos métodos manejan las interacciones del usuario y desencadenan varias acciones, como guardar, actualizar, eliminar o limpiar datos.

5. Otros Métodos:

- **listar():** Este método obtiene datos de la tabla de la base de datos compras y los muestra en la **JTable (tablaCompra)** utilizando el **DefaultTableModel**.
- **eliminar():** Este método elimina un registro de la tabla compras según el **NumeroCompra** seleccionado en la tabla.
- **actualizar():** Este método actualiza un registro en la tabla compras según los datos ingresados en los campos de la GUI.
- **BuscarDatosCompra():** Este método recopila los datos ingresados en los campos de la GUI y crea un objeto Compras con esos datos.
- **limpiarTabla(DefaultTableModel model):** Este método borra los datos mostrados en la JTable.
- **contar():** Este método genera un número de compra único consultando la base de datos y encontrando el valor máximo de **NumeroCompra**.
- **ObtenerProductosComboBox():** Este método obtiene nombres y precios de productos de la tabla de la base de datos **Productos** y los muestra en el cuadro combinado **comboxproductos**.
- **calcular_itbms():** Este método calcula el y el precio total en función de los precios acumulados de los productos y muestra los resultados en los campos de la GUI.
- **guardar():** Este método guarda los datos de la compra en la tabla compras llamando al método **IngresoDatos.ingresoCompra().**

DatosProducto

1. Variables y Componentes de la Interfaz Gráfica (GUI):

- La clase declara varias variables de instancia, que son componentes de la GUI, como botones (**btn_eliminarDatosCliente1**, **jButton1**, **jButton2**, etc.), etiquetas (**jLabel1**, **jLabel2**, etc.), campos de texto (**text_IdProducto**, **text_PrecioDescuento**, etc.), una tabla (**TablaProducto**), y un modelo de tabla (**DefaultTableModel**).

- Los componentes se inicializan y se configuran sus propiedades en el método i**nitComponents()**. La disposición de los componentes se define utilizando **GroupLayout**.

2. **Constructor DatosProducto():**

   El constructor de la clase inicializa los componentes de la GUI llamando al método **initComponents()** y crea un nuevo **DefaultTableModel** que se asigna a la variable modelo.

3. **Métodos de Manejo de Acciones:**

   La clase define varios métodos con prefijos **jButton_**, **text_**, y **TablaProducto_**, que corresponden a las acciones de los botones, campos de texto y la tabla, y sus respectivos manejadores de eventos. Estos métodos manejan las interacciones del usuario con la interfaz gráfica y desencadenan diversas acciones.

4. **Métodos de Funcionalidad:**

   - **guardar():** Este método se utiliza para guardar los datos de un nuevo producto en la base de datos utilizando la clase **IngresoDatos**.
   - **eliminar():** Este método elimina un producto de la base de datos mediante la clase **EliminarDatos**, tomando como referencia el ID del producto seleccionado en la tabla.
   - **listar():** Este método obtiene datos de la tabla de la base de datos Productos y los muestra en la **JTable (TablaProducto)** utilizando el **DefaultTableModel**.
   - **limpiarTabla(DefaultTableModel model):** Este método borra todos los datos mostrados en la **JTable (TablaProducto)** pasando como argumento el modelo de la tabla para eliminar todas las filas.
   - **BuscarDatosProductos():** Este método recopila los datos ingresados en los campos de la GUI y crea un objeto **Productos** con esos datos para ser utilizado en las operaciones de guardar y actualizar.

5. **Otros Métodos de la Interfaz:**

   Hay métodos para manejar las acciones de los botones y campos de texto, como **jButton1ActionPerformed(evt)**, **jButton2ActionPerformed(evt)**, etc. Estos métodos ejecutan acciones específicas cuando se interactúa con los componentes correspondientes.

ConsultasVistas

La clase **Consultas**Vistas extiende **JFrame**, lo que indica que es una ventana de GUI.

1. **Declaración de variables y componentes de la interfaz:**

   La clase **ConsultasVistas** tiene una serie de variables y componentes que representan botones, etiquetas, tablas, campos de texto, etc. Que se utilizaran dentro de la clase.

2. **Constructor ConsultasVistas():**

   Es el constructor de la clase y se encarga de inicializar los componentes de la interfaz gráfica. Llama al método **initComponents()** para realizar esta inicialización.

3. **Método initComponents():**

   - Este método crea, configura y organiza los componentes de la interfaz gráfica. Agrupa los componentes en pestañas con nombres **"clientes"**, **"productos"** y **"compras"** utilizando un objeto **JTabbedPane**.
   - Cada pestaña contiene una tabla, etiquetas, campos de texto y botones que permiten realizar consultas y eliminar registros de esas entidades.

4. **Métodos para acciones de los botones:**

   La clase contiene varios métodos para manejar las acciones que ocurren cuando los botones son presionados.

   Por ejemplo, al hacer clic en el botón **"buscar"** en cualquiera de las pestañas, se realizará una consulta en la base de datos y mostrará los resultados en la tabla correspondiente. Al hacer clic en el botón **"eliminar"**, se eliminarán registros de la tabla.

5. **Métodos para realizar consultas:**
   - En los métodos **btnBuscarClientesActionPerformed**, **btnBuscarProductosActionPerformed** y **btnBuscarComprasActionPerformed**, se llevan a cabo las consultas en la base de datos utilizando la clase **Consultas**
   - Los resultados de las consultas se agregan a la tabla correspondiente mediante el uso de **DefaultTableModel**.

6. **Métodos para limpiar tablas:**
   Los métodos **btnEliminarProductoActionPerformed**, **btnEliminarClienteActionPerformed** y **btnEliminarCompraActionPerformed** se utilizan para limpiar las tablas después de eliminar registros.

7. **Métodos para volver a la ventana principal:**
   Los métodos **jButton2ActionPerformed**, **jButton3ActionPerformed** y **jButton4ActionPerformed** se utilizan para cerrar la ventana actual **ConsultasVistas** y abrir una nueva instancia de la ventana **Principal**.

Base de Datos

Se crea en mariadb una **base de datos** con el nombre de “BaseFinal”.

Se declara **use** para determinar la base de datos que estaremos usando.

Se crea una primera **tabla** llamada “Clientes” que tiene como **llave primaria** “numero_cedula” de tipo **varchar**.

También se le agregan como **atributos** de tipo **varchar (**tipo_cliente, nombre, fecha_nacimiento, genero, correo, numero_telefono, provincia, ciudad, corregimiento, productos_frecuentes, asesor_asignado**)**, de tipo **int (**membresia**)** y de tipo **decimal (**cantidad_promedio_gastada, descuento, cantidad_credito**)**.

Se crea una segunda **tabla** llamada “Productos” que tiene como **llave primaria** “id” de tipo **int**.

También se le agregan como **atributos** de tipo **varchar** (nombre, codigo_barras), de tipo **decimal** (precio_estandar, precio_descuento).

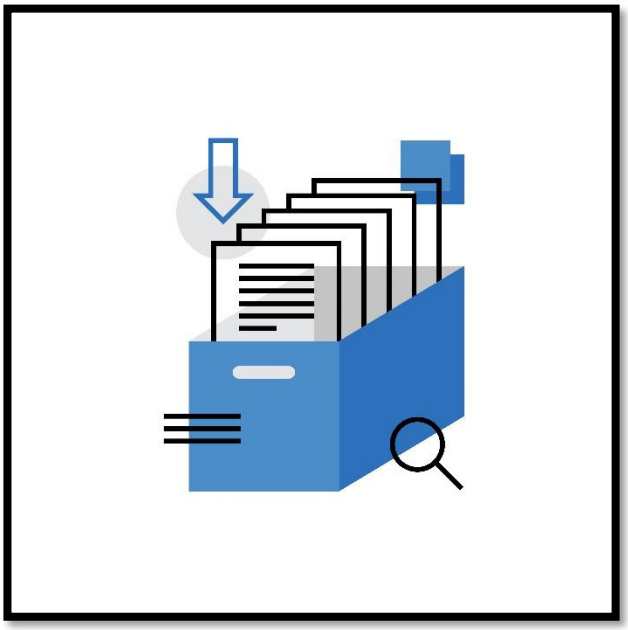
Se crea una tercera **tabla** llamada “Compras” que tiene como **llave primaria** “NumeroCompra” de tipo **int**.

También se le agregan como **atributos** de tipo **varchar** (cedula, nombre_producto, fecha_compra), de tipo **decimal** (costo_total, ITBMS) y de tipo **int** (cantidad_productos).

*__not null__ = que no debe estar vacio;

Repositorio

Para acceder al Repositorio de GitHub darle clic a la imagen de abajo.



Conclusiones

Este proyecto de desarrollo de este proyecto ha representado una oportunidad enriquecedora para aplicar nuestros conocimientos teóricos y aprender a enfrentar desafíos reales en el ámbito del desarrollo de software. El proceso de diseño, implementación y pruebas nos ha brindado una valiosa experiencia práctica y nos ha permitido mejorar nuestras habilidades en el desarrollo de aplicaciones funcionales y eficientes.