

**UNIVERSIDAD MARIANO GÁLVEZ DE GUATEMALA**

**INGENIERÍA EN SISTEMAS**

**PROGRAMACIÓN II**

**ING. JHONNY MORALES**

**PROYECTO FINAL**

**MARVIN ALEXANDER PABLO MATÍAS**

**0904-19-19673**

**ESDRAS MIQUEAS HERNÁNDEZ MARROQUÍN**

**0904-15-11681**

**HUEHUETENANGO 30 DE OCTUBRE DE 2020**

## ÍNDICE

### Contenido

<b>PROCESO DE ELABORACIÓN DEL PROYECTO .....</b>	<b>3</b>
<b>TECNOLOGIAS UTILIZADAS:.....</b>	<b>29</b>
<b>Manual de usuario:.....</b>	<b>30</b>
<b>DIAGRAMA DE CLASES: .....</b>	<b>34</b>
<b>DIAGRAMA DE CLASES ER: .....</b>	<b>35</b>

# PROCESO DE ELABORACIÓN DEL PROYECTO

Se divide en 4 paquetes los cuales son:

## CRUD:

Aquí están las clases donde están los métodos de, crear, eliminar, actualizar, y listar registros en la base de datos.

Se utiliza una interfaz donde se colocaran los métodos que serán utilizados por los métodos de los CRUD.

Las entidades o tablas son: Clientes, Productos, Vendedor, Ventas y detalles ventas, Ventas y detalles ventas se estarán trabajando en la misma clase VentaCRUD.

## MODELO:

Aquí están las clases, donde se colocaran todos los campos utilizados en las tablas de la base de datos.

Se realizó el encapsulamiento, con los métodos getter y setter, además se utilizó un constructor vacío y uno con todas las variables.

## REPORTES:

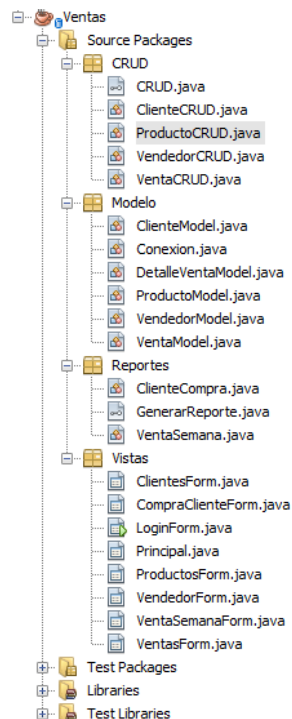
Aquí están las clases, donde se realizaran los métodos para generar los 2 reportes, para ventas por semana y ventas por cliente.

Se utiliza una interfaz donde está un método que será utilizado en las otras 2 clases.

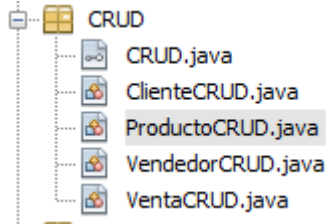
## VISTAS:

Están todos los formularios, Login y Principal son JFrame.

Los demás formularios son JInternalFrame, para que estos sean mostrados se necesita que el formulario Principal tenga un elemento que se llama jdesktop.



## ESTRUCTURA DE UNA CLASE DEL PAQUETE CRUD



Se utiliza una interfaz que se llamara CRUD.

Se instancian todos los métodos que se utilizaran en las otras clases

```
package CRUD;

import java.util.List;

// Interfaz con metodos abstractos utilizados para los CRUD de las entidades
public interface CRUD {
    public List listar();
    public int add(Object[] o);
    public int actualizar(Object[] o);
    public void eliminar(int id);
}
```

### Ejemplo de una clase del paquete del crud:

las clases ClienteCRUD, ProductoCRUD, VendedorCRUD, VentaCRUD tienen los mismos métodos ya que heredan de Conexión e implementan la misma interfaz.

### Ejemplo de Cliente:

Se importan librerías

```
] import Modelo.ClienteModel;
import Modelo.Conexion;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.ArrayList;
import java.util.List;
- import javax.swing.JOptionPane;
```

En la clase se hereda de la clase de Conexión y se implementa la interfaz CRUD

La herencia de la clase conexión es para que no se necesite crear un objeto de tipo conexión. Para que se pueda utilizar al instante

```
public class ClienteCRUD extends Conexion implements CRUD {
```

### Método listar:

Se crea un objeto lista de ClienteModel que es el modelo.

Se hace el string de la consulta que se llama sql.

Try y Catch para manejo de excepciones.

En el try se coloca todo el proceso.

Ps es una variable y rs que se crean al inicio de la clase.

En ps se asigna el valor de la conexión, connection es parte de la clase conexión que se hereda.

En rs se asigna, la ejecución de la consulta.

Se hace un ciclo while para asignar al objeto ClienteModel que es el modelo.

Los setId, setDpi... son los setter que se crearon en ese modelo, para obtener lo que se devolvió en la consulta.

En el catch. Se atrapa si existe un error y en lugar de cerrar todo el sistema solo se mostrar un mensaje.

```
// Listar registros que se persisten en la base de datos
@Override
public List listar() {
    List<ClienteModel> lista = new ArrayList<>();
    String sql = "select * from Cliente";
    try {
        ps = connection.prepareStatement(sql);
        rs = ps.executeQuery();

        while (rs.next()) {
            ClienteModel c = new ClienteModel();
            c.setId(rs.getInt(1));
            c.setDpi(rs.getString(2));
            c.setNit(rs.getString(3));
            c.setNombres(rs.getString(4));
            c.setTelefono(rs.getString(5));
            c.setDireccion(rs.getString(6));
            lista.add(c);
        }
    } catch (Exception e) {
        JOptionPane.showMessageDialog(null, "No se pudo listar los registros: " + e.getMessage());
    }

    return lista;
}
```

**Método agregar:**

es un método retornar un entero,

sql es la variable donde se guarda el string de consulta.

Try y catch lo mismo de la vez pasada.

Ps se le asigna para realizar la conexión.

Con ps se establece que en los signos de interrogación que están en el valor de la variable sql

Lo que nosotros asignamos en el formulario de cliente o de los otros.

Se coloquen según su posición.

Ps.setObject(1, o[0]); 1 es la posición del signo de interrogación, o[0] es la posición del dato que se está enviando en el formulario.

En la variable r = se asigna lo de la ejecución de esa inserción.

Catch lo mismo para manejar excepciones.

Retorna r;

```
// metodo para agregar registros a la tabla
@Override
public int add(Object[] o) {
    int r = 0;
    String sql = "insert into Cliente(Dpi, Nit, Nombres, Telefono, Direccion) values(?,?,?,?,?)";

    try {
        ps = connection.prepareStatement(sql);
        ps.setObject(1, o[0]);
        ps.setObject(2, o[1]);
        ps.setObject(3, o[2]);
        ps.setObject(4, o[3]);
        ps.setObject(5, o[4]);
        r = ps.executeUpdate();
    } catch (Exception e) {
        JOptionPane.showMessageDialog(null, "No se pudo crear el registro: " + e.getMessage());
    }
    return r;
}
```

### Método Actualizar:

Es lo mismo a insertar.

En la variable sql, set es para colocar los nuevos valores en su respectivo campo, y where id es que el id que se envía a través del formulario se igual al que este en la bd.

```
// metodo para actualizar registros a la bd
@Override
public int actualizar(Object[] o) {
    int r = 0;
    String sql = "update Cliente set Dpi=?, Nit=?, Nombres=?, Telefono=?, Direccion=? where id = ? ";

    try {
        ps = connection.prepareStatement(sql);
        ps.setObject(1, o[0]);
        ps.setObject(2, o[1]);
        ps.setObject(3, o[2]);
        ps.setObject(4, o[3]);
        ps.setObject(5, o[4]);
        ps.setObject(6, o[5]);
        r = ps.executeUpdate();
    } catch (Exception e) {
        JOptionPane.showMessageDialog(null, "No se pudo actualizar el registro: " + e.getMessage());
    }

    return r;
}
```

### Método eliminar:

el formulario o la vista de cliente se enviara el id del cliente y este método lo recibirá para que en la variable sql = delete from cliente where id = signo se coloque el id que se este enviando.

Para eso se entra en el try y catch. Y en ps.setInt(1, id) se pone el id devuelto.

Se ejecuta la consulta y lo elimina.

```
// eliminar registro por id del registro
@Override
public void eliminar(int id) {
    String sql = "delete from Cliente where id = ? ";

    try {
        ps = connection.prepareStatement(sql);
        ps.setInt(1, id);
        ps.executeUpdate();
    } catch (Exception e) {
        JOptionPane.showMessageDialog(null, "No se pudo eliminar el registro: " + e.getMessage());
    }
}
```

## CLASE CONEXIÓN

Aquí se establece las credenciales para establecer la conexión.

```
package Modelo;

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

// clase de conexion donde se establece las credenciales de la base de datos y el driver de conexion
public class Conexion {
    public static final String USERNAME = "root";
    public static final String PASSWORD = "C0ntr@s3n@";
    public static final String HOST = "localhost";
    public static final String PORT = "3306";
    public static final String DATABASE = "ProyectoVentas";
    public static final String CLASSNAME = "com.mysql.cj.jdbc.Driver";
    public static final String URL = "jdbc:mysql://" + HOST + ":" + PORT + "/" + DATABASE;

    public Connection connection;

    // Constructor
    public Conexion() {
        try {
            Class.forName(CLASSNAME);
            connection = DriverManager.getConnection(URL, USERNAME, PASSWORD);
        } catch (ClassNotFoundException e) {
            System.err.println("Error de clase: " + e.toString());
        } catch (SQLException e) {
            System.err.println("Error de sql: " + e.toString());
        }
    }
}
```

Las variables con public static final SON LAS CREDENCIALES DEL BASE DE DATOS.

En el constructor de conexión se realiza la conexión.

La public connection connection es la variable que se usa en las otras clases donde se hereda este modelo.

En try se realiza el proceso de conexión a la bd y catch maneja la excepción si aparece.



## EJEMPLO DE UNA CLASE DE MODELO

Se utilizar el modelo de cliente.

Se declaran las variables, las misma que se tienen el bd en esa entidad de cliente

Se crean 2 constructores, uno vacio y uno con todas las variables.

Se crean getter y setter esto es conocido como encapsulación.

```
package Modelo;

// Se establece campos de la tabla para luego se utilizado como objeto
public class ClienteModel {
    int id;
    String dpi;
    String nit;
    String nombres;
    String telefono;
    String direccion;

    public ClienteModel() {
    }

    public ClienteModel(int id, String dpi, String nit, String nombres, String telefono, String direccion) {
        this.id = id;
        this.dpi = dpi;
        this.nit = nit;
        this.nombres = nombres;
        this.telefono = telefono;
        this.direccion = direccion;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }
}
```

## EJEMPLO DE REPORTES

Interfaz utilizada GenerarReporte

```
package Reportes;

import java.sql.ResultSet;

//Interfaz para que se implemente en los reportes
public interface GenerarReporte {
    // metodo abstracto que se utilizara en otra clase
    public ResultSet generar();
}
```

## Reporte cliente compra

```
// se implemente una interfaz con metodos que se utilizaran para generar el reporte
public class ClienteCompra extends Conexion implements GenerarReporte {

    // Metodo de la interfaz de GenerarReportes;
    @Override
    public ResultSet generar() {
        ResultSet filas = null;
        Statement st = null;
        try {
            // se crea conexion que se heredo de la clase Conexion
            st = connection.createStatement();
            // String de consulta
            String sql = "select c.Nombres as Cliente, c.Nit, sum(v.Monto) as Total, count(*) Compras from Ventas"
                + "inner join cliente c on v.IdCliente = c.id\n"
                + "inner join vendedor ve on v.IdVendedor = ve.id\n"
                + "group by c.Nombres\n"
                + "order by Compras desc;";
            filas = st.executeQuery(sql);
            // manejo de expeciones
        } catch (SQLException e) {
            System.err.println("Error en la carga del driver: " + e.getMessage());
        } catch (Exception e) {
            System.err.println("Error en: " + e.getMessage());
        }
        // retorna resultado obtenido
        return filas;
    }
}
```

Clase cliente compra hereda de conexión e implementa interfaz generar reporte.

Un método de tipo resultset para recibir los datos obtenidos de la consulta que se ejecutara.

La variable sql = tiene toda la consulta para generar el reporte.

Nombre del cliente, nit del cliente, total de la venta, el total de compras realizada.

Se hace un innerjoin con todas la tablas cliente y vendedor para obtener esos datos.

```

public class VentaSemana extends Conexion implements GenerarReporte {

    PreparedStatement ps;
    ResultSet rs;

    // Metodo de la interfaz de GenerarReportes;
    @Override
    public ResultSet generar() {
        ResultSet filas = null;
        Statement st = null;
        try {
            // se crea conexion que se heredo de la clase Conexion
            st = connection.createStatement();
            // String de consulta
            String sql = "select v.FechaVentas, v.NumeroSerie, c.Nombres as Cliente, c.Nit, ve.Nombres as Vendedor, v.Cantidad as Cantidad\n"
                + "inner join cliente c on v.IdCliente = c.id\n"
                + "inner join vendedor ve on v.IdVendedor = ve.id\n"
                + "where week(now(),1)-1 = week(v.FechaVentas)";
            filas = st.executeQuery(sql);
            // Manejo de excepciones
        } catch (SQLException e) {
            System.err.println("Error en la carga del driver: " + e.getMessage());
        } catch (Exception e) {
            System.err.println("Error en: " + e.getMessage());
        }
        // envia resultado de la consulta
        return filas;
    }
}

```

Clase VentaSemana hereda de conexión e implementa interfaz generar reporte.

Un método de tipo resultset para recibir los datos obtenidos de la consulta que se ejecutara.

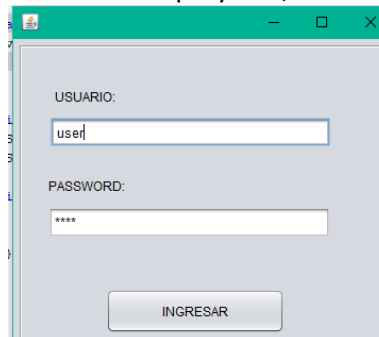
La variable sql = tiene toda la consulta para generar el reporte.

Devolverá fecha de venta, numero de serie, cliente, nit, vendedor, ventas de esta semana de domingo a sábado.

## VISTAS

### LOGIN

es la primera vista que se mostrara al inicio del proyecto, cuando se inicia sesión



USUARIO:

user

PASSWORD:

\*\*\*\*

INGRESAR

```
VendedorCRUD vcrud = new VendedorCRUD();
VendedorModel mv = new VendedorModel();
VentasForm vf = new VentasForm();
```

se crean objetos;

```
private void btnIngresarActionPerformed(java.awt.event.ActionEvent evt) {
    validar();
}

public void validar(){
    String nombre = username.getText();
    String pass = password.getText();

    if(username.getText().equals("") || password.getText().equals("")){
        JOptionPane.showMessageDialog(this, "Debe ingresar datos");
        username.requestFocus();
    }else{
        mv = vcrud.validar(nombre, pass);
        if(mv.getUser() != null && mv.getPassword() != null){
            Principal p = new Principal();
            vf.usuario = mv.getNombres();
            vf.usuarioId = mv.getId();
            p.setVisible(true);
            dispose();
        }else{
            JOptionPane.showMessageDialog(this, "Debe ingresar datos validos");
            username.requestFocus();
        }
    }
}
```

Los botones tienen un evento llamado ActioPerformed todos los botones y menus de este proyecto utilizan este evento.

En código este es un método, lo llamamos como btnIngresar

Se crea un método aparte para la lógica del login.

Username y password son las cajas de texto, para obtener el texto de una caja se utiliza el método getText().

El if evalúa que esas cajas de texto no estén vacías,

Si lo están tirará un mensaje que debe ingresar datos.

Sino, se crea un objeto con la clase de crud del vendedor que en este caso es nuestro usuario.

Con eso se envía el nombre y el password al método de validar que está en la clase vendedorCRUD

Se hace un segundo if si estos son diferentes de null, se abrirá el formulario principal y se cerrará el login.

Sino se enviará un mensaje para que ingrese usuarios válidos.

## FORMULARIO DE CREAR VENTAS

Opciones Ventas Clientes Productos Vendedor Reportes

Ventas

No. Serie: 000008

2020-10-28

Nit:

Buscar

Cliente:

Producto:

Buscar

Producto:

Precio:

Agregar

Stock:

Cantidad:

Vendedor: prueba

No.	Codigo	Producto	Cantidad	Precio Unitario	Total
-----	--------	----------	----------	-----------------	-------

Cancelar

Crear Venta

Total a pagar:

Es el formulario que tiene mas lógica dentro de el

La caja de texto de no serie se generara automáticamente.

La de la fecha también.

El Nit la caja de texto debe ingresar un nit del cliente y presionar el boto buscar de la par y automáticamente aparecer el nombre de cliente en la caja de texto Cliente.

En la caja de texto de Producto se coloca el código de producto y cuando se presione el botón de buscar que está a la par se mostrar en el txt de producto el nombre, en la caja de stock la cantidad del producto en existencia, el precio en la caja de texto de precio.

Se ingresa la cantidad del producto seleccionado en la caja de texto de cantidad.

Cuando se presione el botón de agregar, se agregar el ítem a la tabla que esta abajo  
Se agregara el código del producto, el nombre y la cantidad y el precio unitario y total.

Cuando se presione el botón de crear venta, se guardar en la tabla de ventas el encabezado, y lo que esta en la tabla se guardara en detalle de ventas.

El total de ventas es la suma total de la fila de total de la tabla.

The screenshot shows a Java Swing window titled "Ventas" with a close button in the top right corner. The window contains the following elements:

- No. Serie:** A text field containing "000008".
- 2020-10-28:** A date field.
- Nit:** A text field containing "CF".
- Buscar:** A button next to the Nit field.
- Cliente:** A text field containing "Consumidor Final".
- Producto:** A text field containing "8".
- Buscar:** A button next to the Producto field.
- Producto:** A text field containing "Cacahuates".
- Precio:** A text field containing "100.43".
- Agregar:** A button next to the Precio field.
- Stock:** A text field containing "29".
- Cantidad:** A text field containing "1" with a spinner control.
- Vendedor:** A text field containing "prueba".

Below the input fields is a table with the following data:

No.	Codigo	Producto	Cantidad	Precio Unitario	Total
1	8	Cacahuates	1	100.43	100.43

At the bottom of the window, there are two buttons: "Cancelar" and "Crear Venta". To the right of these buttons is a text field labeled "Total a pagar:" containing the value "100.43".

## METODOS UTILIZADOS EN FORMULARIO DE GENERAR VENTAS

Todas las clases de formularios heredan de javax.swing dependiendo si son JFrame o JInternalFrame que se han usado en este momento.

Se crean los objetos que se utilizaran en los métodos.

```
public class VentasForm extends javax.swing.JInternalFrame {

    ClienteCRUD ccrud = new ClienteCRUD();
    ProductoCRUD pcrud = new ProductoCRUD();
    VentaCRUD vcrud = new VentaCRUD();
    ProductoModel producto = new ProductoModel();
    VentaModel venta = new VentaModel();
    DetalleVentaModel dVenta = new DetalleVentaModel();
    ClienteModel cliente = new ClienteModel();
    VendedorModel vendedor = new VendedorModel();
    public static String usuario;
    public static int usuarioId;

    DefaultTableModel modelo = new DefaultTableModel();
    int idp;
    int cantidad;
    double precio;
    double tPago;
```



/\*\*

El defaulttablemodel es para manejar la tabla.

```
*/
public VentasForm() {
    initComponents();
    generarSerie();
    fecha();
    nombreVendedor();
}
```

Constructor, se inician los métodos para generar la serie, la fecha de la venta, y el nombre del vendedor.

```
void fecha() {
    Calendar calendar = new GregorianCalendar();
    txtFecha.setText("" + calendar.get(Calendar.YEAR) + "-" + (calendar.get(Calendar.MONTH) + 1) + "-" + ca
}
```

Se instancia objeto de tipo calendar para generar la fecha.

```

void generarSerie() {
    String serie = vcrud.NroSerieVentas();

    if (serie == null) {
        txtNoFactura.setText("0000001");
    } else {
        int increment = Integer.parseInt(serie);
        increment = increment + 1;
        txtNoFactura.setText("00000" + increment);
    }
}

```

Para generar la serie de factura, se hizo un método en el clase de ventaCRUD para generar el numero de serie es una consulta sql para obtener el ultimo registro y así generarlo si no existe y si existe solo es a concatenar mas una variable que va aumentando.

```

private void btnClienteActionPerformed(java.awt.event.ActionEvent evt) {
    buscarCliente();
}

```

```

private void btnProductoActionPerformed(java.awt.event.ActionEvent evt) {
    buscarProducto();
}

```

```

private void btnAgregarActionPerformed(java.awt.event.ActionEvent evt) {
    agregarProducto();
}

```

```

private void btnCrearActionPerformed(java.awt.event.ActionEvent evt) {
    if (txtTotal.getText().equals("")) {
        JOptionPane.showMessageDialog(this, "Debe crear una venta");
    } else {
        guardarVenta();
        guardarDetalleVenta();
        actualizarStock();
        JOptionPane.showMessageDialog(this, "Venta Creada Correctamente");
        limpiarTabla();
        nuevo();
        generarSerie();
    }
}

```

Estos son los botones con el evento actio perfomed, en esto se llama a los métodos necesarios para el proceso.

En el botón de crear se tiene un if que si la caja de texto txtTotal esta vacio se debe crear una y si no esta vacio se llamaran a los métodos para el proceso de generar la venta.



```

void limpiarTabla() {
    for (int i = 0; i < modelo.getRowCount(); i++) {
        modelo.removeRow(i);
        i = i - 1;
    }
}

```

Limpia la tabla cada vez que se termine de generar la venta para generar una nueva.

```

void actualizarStock() {
    for (int i = 0; i < modelo.getRowCount(); i++) {
        ProductoModel producto = new ProductoModel();
        idp = Integer.parseInt(tabla.getValueAt(i, 1).toString());
        cantidad = Integer.parseInt(tabla.getValueAt(i, 3).toString());
        producto = pcrud.listarID(idp);
        int existencia = producto.getStock() - cantidad;

        pcrud.actualizarStock(existencia, idp);
    }
}

```

Actualiza el stock del producto cuando se termine de generar la venta,

En el ciclo for se hace la instancia de un objeto de productoModel.

El pcrud es una variable que se instancio arriba en la clase con los atributos de la clase

ProductoCRUD que tiene un método para actualizar el stock el cual debe recibir dos parámetros, la existencia, y el id de producto que se enviara, la existencia se va obtener del stock que tiene producto menos la cantidad que es la columna de la tabla es por eso que se usa el ciclo for.

### Guardar venta

```

void guardarVenta() {
    int idVendedor = usuarioId;
    int idCliente = cliente.getId();
    String serie = txtNoFactura.getText();
    String fecha = txtFecha.getText();
    double monto = tPago;
    String estado = "Activo";

    venta.setIdCliente(idCliente);
    venta.setIdVendedor(idVendedor);
    venta.setSerie(serie);
    venta.setFecha(fecha);
    venta.setMonto(monto);
    venta.setEstado(estado);
    vcrud.GuardarVentas(venta);
}

```

el id de vendedor se pasa a través de una variable pública y estática instanciada en la parte arriba de la clase, la cual se le asigna un valor cuando el usuario inicia sesión. Es lo mismo con el nombre del vendedor.

Los demás campos son con las cajas de texto;

Se usa el objeto venta que se instanció arriba en la clase para poder utilizar los métodos getter y setter.

Ya con `vcrud` se guarda la venta con ese método que se hizo en el `VentasCRUD` recibe un objeto de tipo venta (`ventaModel`).

```
void guardarDetalleVenta() {
    String idv = vcrud.IdVentas();
    int idve = Integer.parseInt(idv);

    for (int i = 0; i < tabla.getRowCount(); i++) {
        int idp = Integer.parseInt(tabla.getValueAt(i, 1).toString());
        int cantidad = Integer.parseInt(tabla.getValueAt(i, 3).toString());
        double precio = Double.parseDouble(tabla.getValueAt(i, 4).toString());
        dVenta.setIdVentas(idve);
        dVenta.setIdProducto(idp);
        dVenta.setCantidad(cantidad);
        dVenta.setPrecioVenta(precio);

        vcrud.GuardarDetalleVentas(dVenta);
    }
}
```

Guardar detalle venta es un for para recorrer la tabla y obtener los ítems y guardarlos con el método de guardar detalle ventas que está en `VentasCRUD`.

```

void agregarProducto() {
    double total;
    modelo = (DefaultTableModel) tabla.getModel();
    int item = 0;
    item = item + 1;
    idp = producto.getId();
    String nomProducto = verProducto.getText();
    precio = Double.parseDouble(txtPrecio.getText());
    cantidad = Integer.parseInt(txtCantidad.getValue().toString());
    int stock = Integer.parseInt(verStock.getText());
    total = cantidad * precio;
    ArrayList lista = new ArrayList();
    if (stock > 0) {
        lista.add(item);
        lista.add(idp);
        lista.add(nomProducto);
        lista.add(cantidad);
        lista.add(precio);
        lista.add(total);
        Object[] ob = new Object[6];
        ob[0] = lista.get(0);
        ob[1] = lista.get(1);
        ob[2] = lista.get(2);
        ob[3] = lista.get(3);
        ob[4] = lista.get(4);
        ob[5] = lista.get(5);
        modelo.addRow(ob);
        tabla.setModel(modelo);
    }
}

```

Este método se utiliza en el formulario cuando se presiona el botón de agregar, se guardará en la tabla, se instancia de tipo object para guardarlo en la tabla ya que este acepta solo de tipo object, se agrega la fila con addrow(ob), si el stock es mayor a cero realiza el proceso y si no mostrará un mensaje que no hay existencias.

```

void calculaTotal() {
    tPago = 0;

    for (int i = 0; i < tabla.getRowCount(); i++) {
        cantidad = Integer.parseInt(tabla.getValueAt(i, 3).toString());
        precio = Double.parseDouble(tabla.getValueAt(i, 4).toString());
        tPago = tPago + (cantidad * precio);
    }

    txtTotal.setText("" + tPago);
}

```

Método para hacer el total de las ventas en la parte de abajo del formulario, es un for, para hasta que sea el total de filas de la tabla, esto se mostrara en la caja de texto de total que esta abajo del formulario.

```
void buscarCliente() {
    String cod = txtCliente.getText();
    int r;
    if (txtCliente.getText().equals("")) {
        JOptionPane.showMessageDialog(this, "Debe ingresar el cliente");
    } else {
        cliente = ccrud.listarID(cod);
        if (cliente.getNit() != null) {
            verCliente.setText(cliente.getNombres());
            txtProducto.requestFocus();
        } else {
            r = JOptionPane.showConfirmDialog(this, "Cliente no Registrado, Desea Registrarlo?");
            if (r == 0) {
                ClientesForm cf = new ClientesForm();
                Principal.desktop.add(cf);
                cf.setVisible(true);
            }
        }
    }
}
```

Este método busca el cliente con el nit, en el if si es igual a vacío el mensaje tirara que debe buscar un cliente,

Si el cliente es diferente que vacío se hace una segunda condición donde se evalúa con el método de getNit que es un método de encapsulación del modelo cliente.

Esto llena la caja de texto del nombre del cliente.

Sino se cumple eso, mostrara un mensaje que el cliente no está registrado y tirará opciones si desea registrarlo si lo desea mostrar el formulario para crear el cliente.

```
,
void buscarProducto() {

    if (txtProducto.getText().equals("")) {
        JOptionPane.showMessageDialog(this, "Debe ingresar un Código de producto");
    } else {
        int id = Integer.parseInt(txtProducto.getText());
        producto = pcrud.listarID(id);
        if (producto.getId() != 0) {
            verProducto.setText(producto.getNombre());
            txtPrecio.setText("" + producto.getPrecio());
            verStock.setText("" + producto.getStock());
        } else {
            JOptionPane.showMessageDialog(this, "Producto no Registrado");
            txtProducto.requestFocus();
        }
    }
}
```

Buscar producto es casi lo mismo, si encuentra el producto llenará las cajas de texto de nombre del producto, el precio y el stock y sino mostrará un mensaje.

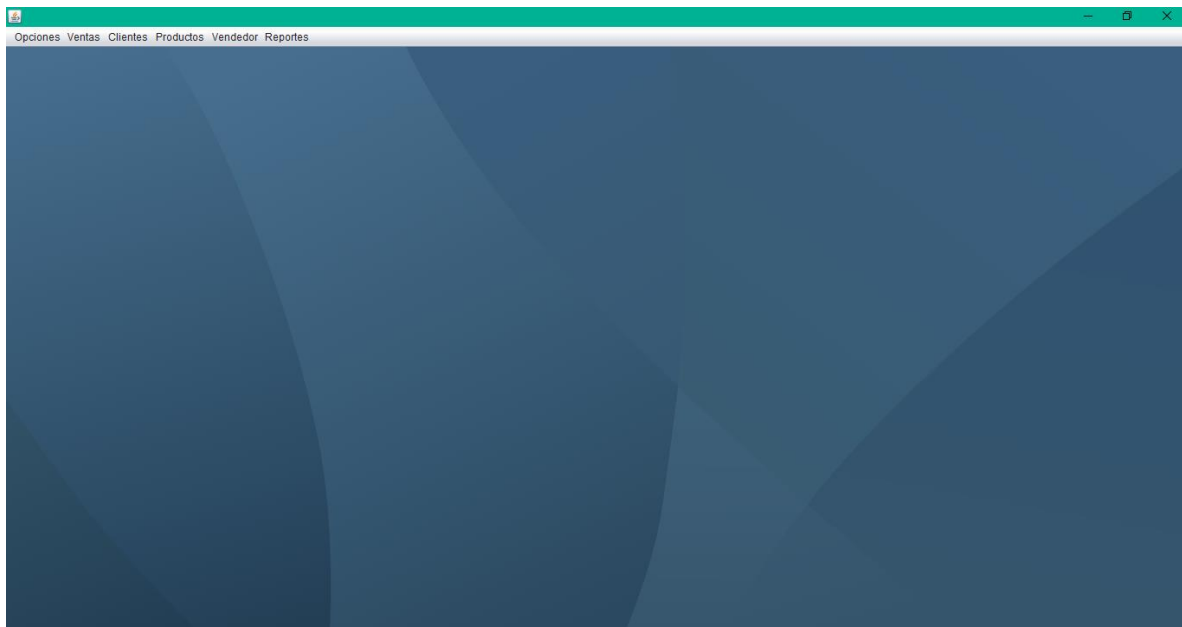
```

void nuevo() {
    txtCliente.setText("");
    txtProducto.setText("");
    txtFecha.setText("");
    txtNoFactura.setText("");
    txtPrecio.setText("");
    txtTotal.setText("");
    verCliente.setText("");
    verProducto.setText("");
    verStock.setText("");
    verVendedor.setText("");
    txtCantidad.setValue(1);
    txtCliente.requestFocus();
}

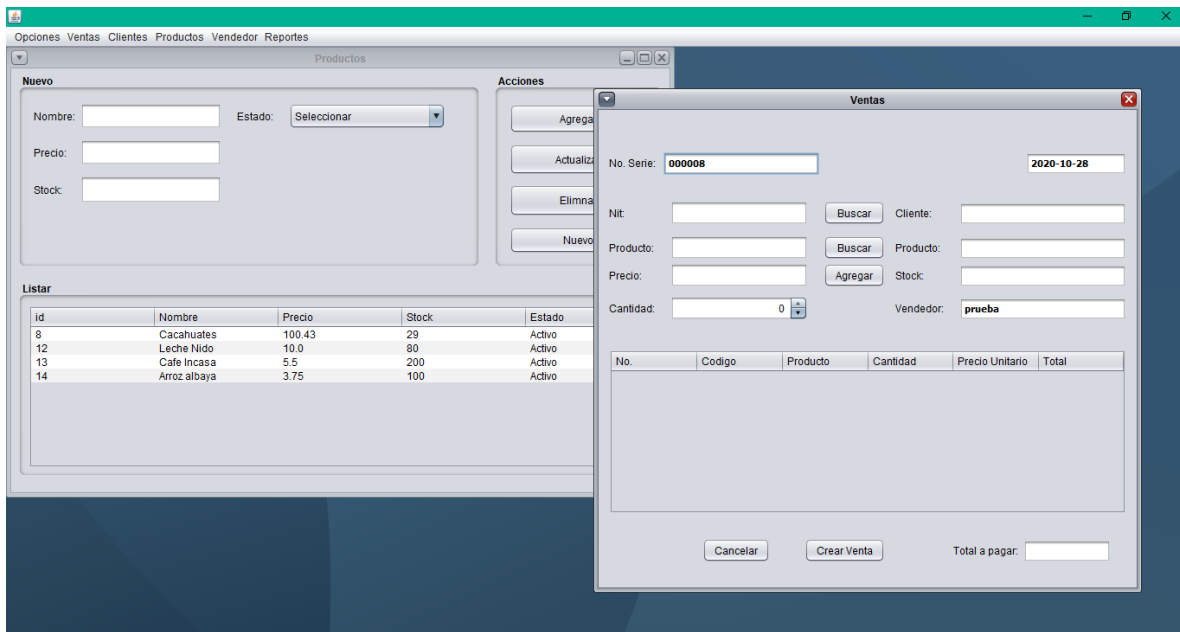
```

Método que Limpiara todas las cajas de texto.

## VISTA PRINCIPAL



LA VISTA PRINCIPAL SE COMPONE DE UN menú y de un elemento llamado desktop pane, el cual es utilizado con los jinternal frame , que cuando se seleccionen por ejemplo producto o genera una venta se muestre dentro de ese desktoppane



```

private void opRVentaActionPerformed(java.awt.event.ActionEvent evt) {
    VentasForm vf = new VentasForm();
    Centrar(vf);
}

private void opClienteActionPerformed(java.awt.event.ActionEvent evt) {
    ClientesForm vf = new ClientesForm();
    Centrar(vf);
}

private void opProductoActionPerformed(java.awt.event.ActionEvent evt) {
    ProductosForm vf = new ProductosForm();
    Centrar(vf);
}

private void opVendedorActionPerformed(java.awt.event.ActionEvent evt) {
    VendedorForm vf = new VendedorForm();
    Centrar(vf);
}

```

Estos métodos tienen evento actionPerformed, estos son los elementos del menú.

Se crea un objeto por ejemplo de venta form.

Ese objeto se envía a un método centrar, el cual despliega el formulario según la opción del menú que se selecciona.

```

void Centrar(JInternalFrame frame) {
    desktop.add(frame);
    frame.show();
}

```

Método para desplegar cualquier jinternal frame

## EJEMPLO DE UN FORMULARIO

id	Nombre	Precio	Stock	Estado
8	Cacahuates	100.43	29	Activo
12	Leche Nido	10.0	80	Activo
13	Cafe Incasa	5.5	200	Activo
14	Arroz albaya	3.75	100	Activo

La estructura de los formulario será siempre igual excepto para los reportes y generar venta.

Este formulario tiene cajas de texto nombre, precio, stock, y un combo box seleccionar que tiene los estados de un producto activo o inactivo, en acciones están los botones agregar actualizar, eliminar y nuevo.

Y las tablas.

```
public class ProductosForm extends javax.swing.JInternalFrame {

    ProductoCRUD crud = new ProductoCRUD();
    ProductoModel md = new ProductoModel();
    DefaultTableModel modelo = new DefaultTableModel();
    int id;

    /**
     * Creates new form ProductosForm
     */
    public ProductosForm() {
        initComponents();
        listar();
    }
}
```

Clase que hereda de javax.swing internal frame.

Se instancian objeto para trabajar.

Producto crud es donde están los métodos para listar, eliminar, guardar, y editar.

El modelo de producto en este caso.

Defaultable model para trabajar con la tabla.

Y el constructor con el método de listar para inicializar las tablas.

```
void listar() {  
    List<ProductoModel> lista = crud.listar();  
    modelo = (DefaultTableModel) tabla.getModel();  
    Object[] ob = new Object[5];  
    for (int i = 0; i < lista.size(); i++) {  
        ob[0] = lista.get(i).getId();  
        ob[1] = lista.get(i).getNombre();  
        ob[2] = lista.get(i).getPrecio();  
        ob[3] = lista.get(i).getStock();  
        ob[4] = lista.get(i).getEstado();  
        modelo.addRow(ob);  
    }  
    tabla.setModel(modelo);  
}
```

Método para listar ítems que devuelve el método listar del objeto crud.

Se instancia la tabla.

Se hace un for para agregar las filas que devuelva la tabla en la base de datos.

Se agrega a la tabla del formulario.



```

private void btnAgregarActionPerformed(java.awt.event.ActionEvent evt) {
    agregar();
    cleanTabla();
    listar();
    nuevo();
}

private void btnActualizarActionPerformed(java.awt.event.ActionEvent evt) {
    actualizar();
    cleanTabla();
    listar();
    nuevo();
}

private void btnEliminarActionPerformed(java.awt.event.ActionEvent evt) {
    eliminar();
    cleanTabla();
    listar();
    nuevo();
}

private void btnNuevoActionPerformed(java.awt.event.ActionEvent evt) {
    nuevo();
}

```

Son los botones que se mencionaron anteriormente, con los métodos que se utilizarán.

```

private void tablaMouseClicked(java.awt.event.MouseEvent evt) {
    int fila = tabla.getSelectedRow();
    if (fila == -1) {
        JOptionPane.showMessageDialog(this, "Debe Seleccionar una Fila");
    } else {
        id = Integer.parseInt(tabla.getValueAt(fila, 0).toString());
        String nombre = tabla.getValueAt(fila, 1).toString();
        String precio = tabla.getValueAt(fila, 2).toString();
        String stock = tabla.getValueAt(fila, 3).toString();
        String estado = tabla.getValueAt(fila, 4).toString();

        txtNombre.setText(nombre);
        txtPrecio.setText(precio);
        txtStock.setText(stock);
        sltEstado.setSelectedItem(estado);
    }
}

```

Este evento se ejecutará, cuando se seleccione una columna de la tabla del formulario, si no se ha seleccionado se mostrará el mensaje que debe seleccionar una fila, Si se selecciona, estos valores de la fila pasarán a las cajas de texto para que se actualicen o se eliminen.

```

void agregar() {
    String nombre = txtNombre.getText();
    String precio = txtPrecio.getText();
    String stock = txtStock.getText();
    String estado = sltEstado.getSelectedItem().toString();

    Object[] ob = new Object[4];
    ob[0] = nombre;
    ob[1] = precio;
    ob[2] = stock;
    ob[3] = estado;
    crud.add(ob);
}

```

Agregar los registros que se coloquen en las cajas de texto, se crea un objeto que se enviara al método del crud (productoCRUD) para guardarlo, recordar que este método recibe como parámetro un tipo object.

```

void actualizar() {
    int fila = tabla.getSelectedRow();
    if (fila == -1) {
        JOptionPane.showMessageDialog(this, "Debe Seleccionar una Fila");
    } else {
        String nombre = txtNombre.getText();
        String precio = txtPrecio.getText();
        String stock = txtStock.getText();
        String estado = sltEstado.getSelectedItem().toString();
        Object[] obj = new Object[5];
        obj[0] = nombre;
        obj[1] = precio;
        obj[2] = stock;
        obj[3] = estado;
        obj[4] = id;
        crud.actualizar(obj);
    }
}

```

Es lo mismo que agregar solo que se envía el id para saber que registro se actualizara. Si no se ha seleccionado una fila al presionar, este mostrara un mensaje donde dice que debe seleccionar fila.

```

void eliminar() {
    int fila = tabla.getSelectedRow();
    if (fila == -1) {
        JOptionPane.showMessageDialog(this, "Debe Seleccionar una Fila");
    } else {
        crud.eliminar(id);
    }
}
}

```

Eliminar, método donde se enviara el id del registro, mismo if que antes, sino se ha seleccionado una fila se mostrar un mensaje.

## REPORTES

**Reporte de Ventas de esta Semana**

Fecha	No. Serie	Cliente	Nit	Vendedor	Total
2020-10-26	000002	Luciano Vignato	456	prueba	100.43
2020-10-26	000003	Luciano Vignato	456	prueba	0.0
2020-10-27	000006	Luciano Vignato	456	prueba	1104.30000000...
2020-10-27	000007	Consumidor FI...	CF	prueba	502.150000000...

**Compras Realizadas por Clientes**

Cliente	Nit	Total	Total de Compras
Luciano Vignato	456	1706.88	4
Andres Iniesta	2338234-1	127.5	2
Consumidor Final	CF	502.150000000000003	1

Se mostraran según la opción que se coloque.

### Ejemplo

```

public class VentaSemanaForm extends javax.swing.JInternalFrame {

    DefaultTableModel modelo = new DefaultTableModel();
    VentaSemana vs = new VentaSemana();
}

```

Lo mismo se genera objeto de tipo defaulttablemodel para las tablas.

Objeto de tipo ventasemana donde esta el método que genera el reporte o la consulta de sql.

```

public VentaSemanaForm() throws SQLException {
    initComponents();
    listar();
}

void listar() throws SQLException {
    ResultSet listar = vs.generar();
    modelo = (DefaultTableModel) tabla.getModel();
    Object[] ob = new Object[6];

    while (listar.next()){
        ob[0] = listar.getString(1);
        ob[1] = listar.getString(2);
        ob[2] = listar.getString(3);
        ob[3] = listar.getString(4);
        ob[4] = listar.getString(5);
        ob[5] = listar.getString(6);
        modelo.addRow(ob);
    }
    tabla.setModel(modelo);
}

```

Construeto y método listar.

El método listar tiene el throw sqlexceptio, como se esta manejando con resultset.

Se hace un while de lo que se obtenga del método de generar y esto se mostrara en la tabla del formulario. Será igual en el otro reporte.

## **TECNOLOGIAS UTILIZADAS:**

### **NETBEANS:**

La plataforma NetBeans permite que las aplicaciones sean desarrolladas a partir de un conjunto de componentes de software llamados *módulos*. Un módulo es un archivo Java que contiene clases de java escritas para interactuar con las API de NetBeans y un archivo especial (manifest file) que lo identifica como módulo. Las aplicaciones construidas a partir de módulos pueden ser extendidas agregándole nuevos módulos. Debido a que los módulos pueden ser desarrollados independientemente, las aplicaciones basadas en la plataforma NetBeans pueden ser extendidas fácilmente por otros desarrolladores de software.

El NetBeans IDE permite el desarrollo de todos los tipos de aplicación Java (J2SE, web, EJB y aplicaciones móviles). Entre sus características se encuentra un sistema de proyectos basado en Ant, control de versiones y refactoring.

### **Java:**

Java es un lenguaje de programación orientado a objetos que se incorporó al ámbito de la informática en los años noventa. La idea de Java es que pueda realizarse programas con la posibilidad de ejecutarse en cualquier contexto, en cualquier ambiente, siendo así su portabilidad uno de sus principales logros. Fue desarrollado por Sun Microsystems, posteriormente adquirido por Oracle. En la actualidad puede utilizarse de modo gratuito, pudiéndose conseguir sin problemas un paquete para desarrolladores que oriente la actividad de programar en este lenguaje. Puede ser modificado por cualquiera, circunstancia que lo convierte en lo que comúnmente se denomina "código abierto".

### **MySQL:**

MySQL es un sistema de gestión de bases de datos relacional desarrollado bajo licencia dual: Licencia pública general/Licencia comercial por Oracle Corporation y está considerada como la base de datos de código abierto más popular del mundo,<sup>12</sup> y una de las más populares en general junto a Oracle y Microsoft SQL Server, todo para entornos de desarrollo web.

MySQL fue inicialmente desarrollado por MySQL AB (empresa fundada por David Axmark, Allan Larsson y Michael Widenius). MySQL AB fue adquirida por Sun Microsystems en 2008, y ésta a su vez fue comprada por Oracle Corporation en 2010, la cual ya era dueña desde 2005 de Innobase Oy, empresa finlandesa desarrolladora del motor InnoDB para MySQL.

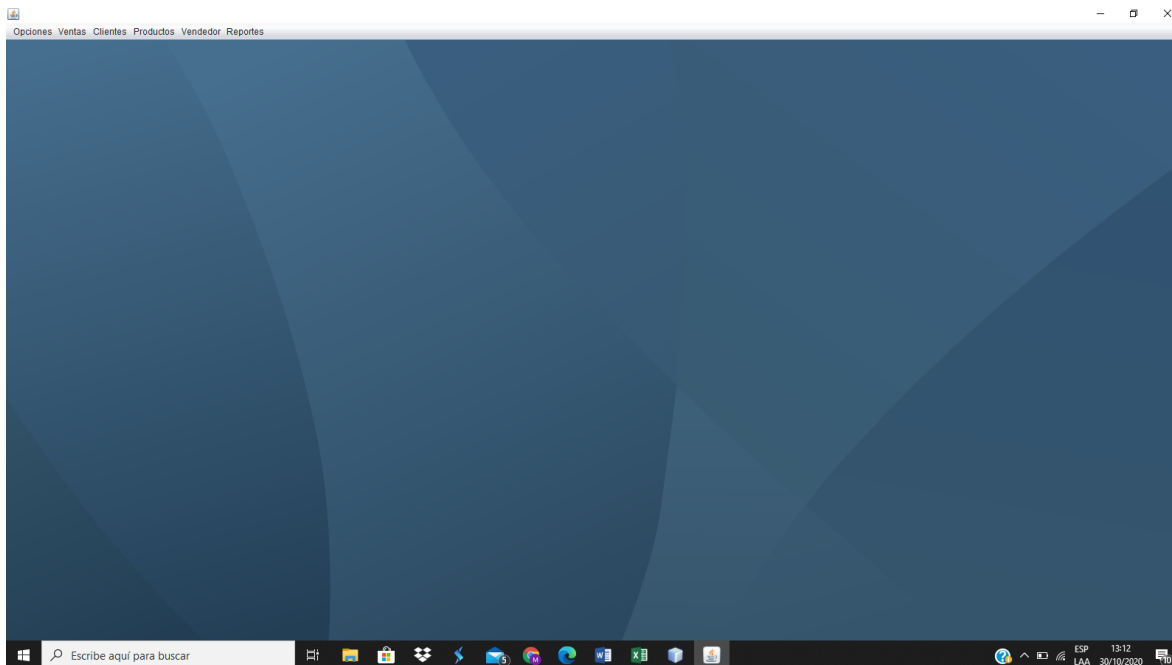
### **GitHub:**

GitHub es un sistema de gestión de proyectos y control de versiones de código, así como una plataforma de red social diseñada para desarrolladores. ¿Pero para qué se usa GitHub? Bueno, en general, permite trabajar en colaboración con otras personas de todo el mundo, planificar proyectos y realizar un seguimiento del trabajo. GitHub es también uno de los repositorios online más grandes de trabajo colaborativo en todo el mundo.

### **GIT:**

Git es un sistema de control de versiones desarrollado por Linus Torvalds (el hombre que creó Linux).

## Manual de usuario:



Esta es la ventana principal del programa, en donde se despliega el menú en la parte superior para que el usuario pueda escoger que desea realizar.



Cada opción tiene una función diferente, por ejemplo si desea realizar una venta debe dar clic en la opción de “realizar venta” y se le desplegará la siguiente ventana:

This is the 'Ventas' form window. It contains the following fields and controls:

- No. Serie: 000002
- 2020-10-30
- Nit: [Empty field] with a 'Buscar' button
- Cliente: [Empty field]
- Producto: [Empty field] with a 'Buscar' button
- Precio: [Empty field] with an 'Agregar' button
- Stock: [Empty field]
- Cantidad: 0 with a '+' button
- Vendedor: Usuario de Prueba

Below these fields is a table with the following columns: No., Codigo, Producto, Cantidad, Precio Unitario, Total. At the bottom of the form are three buttons: 'Cancelar', 'Crear Venta', and 'Total a pagar: [Empty field]'.

En donde salen varios campos que hay que llenar, por ejemplo en nit deberá colocar el nit del cliente, o ya sea CF como consumidor final, en producto debe colocar el código del producto a

vender por ejemplo es 1 para Leche y después deberá dar click en el botón buscar para que automáticamente se llenen los demás campos ya que el programa identificará el producto y su precio etc. El cual quedaría así:

No.	Codigo	Producto	Cantidad	Precio Unitario	Total
1	1	Leche Nido 50...	2	125.75	251.5

Cancel Crear Venta Total a pagar: 251.5

Id	Nombres	Nit	Dpi	Telefono	Direccion
1	Consumidor Final	cf			Ciudad
2	Consumidor Final	CF			Ciudad
3	Consumidor Final	C/F			Ciudad
4	Juan Osorio	12345678	3333333333333...	45678900	zona 1

Para la opción de “Clientes” se despliega esta ventana en la que se puede agregar un nuevo cliente con todos los datos que debe llevar como nombre, tel, dpi, dirección y nit.

Id	Nombre	Precio	Stock	Estado
1	Leche Nido 500 g	125.75	97	Activo
2	Cafe Incasa	2.5	100	Activo
3	Huevos 30 unidades	30.75	20	Activo
4	Salchichas Fud	0.5	100	Activo
5	Consome de pollo Mal...	3.5	100	Activo
6	Consome de pollo Ital...	3.5	100	Activo
7	arroz	3.5	25	Activo

En la opción de productos se despliega esta ventana que muestra todos los productos disponibles en la tienda y nos da la opción de agregar un nuevo producto y/o eliminar alguno que esté ahí, nos muestra detalles como precio del producto, código del producto, y cuantos hay en existencia.

Id	Nombres	Telefono	Estado	User	Password
1	Usuario de Pr...	11111111	Activo	userPrueba	12345678

En la siguiente opción que es “vendedor” podremos ver la lista de vendedores con los que cuenta la tienda, en este caso solo existe un vendedor que es un vendedor de prueba para la ejecución del programa. En esta ventana nos da varias opciones como agregar un nuevo vendedor, eliminar vendedor, y actualizar tabla.



Opciones Ventas Clientes Productos Vendedor Reportes

### Reporte de Ventas de esta Semana

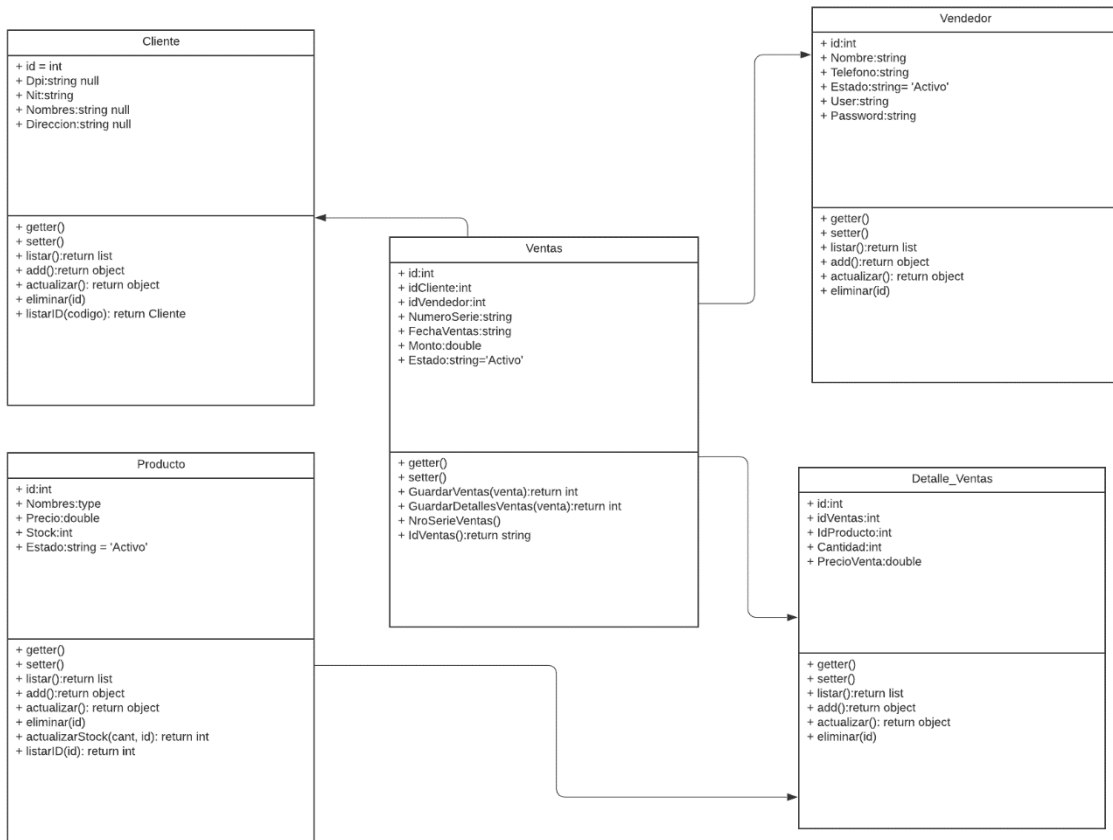
Fecha	No. Serie.	Cliente	Nit	Vendedor	Total
2020-10-30	000001	Consumidor Fl...	C/F	Usuario de Pru...	377.25

### Compras Realizadas por Clientes

Cliente	Nit	Total	Total de Compras
Consumidor Final	C/F	377.25	1

En la última opción “Reportes” tenemos dos opciones, la primera es “Reporte de ventas de esta semana” la cual nos muestra el reporte de las ventas que se han realizado durante esta semana. Y la otra opción “Compras realizadas por cliente” que muestra cuantas compras ha realizado un cliente específico.

## DIAGRAMA DE CLASES:



## DIAGRAMA DE CLASES ER:

