Universidad Peruana Los Andes
Facultad de Ingeniería
Escuela Profesional de Sistemas y Computación

Proyecto Final

Asignatura: **Taller de Aplicaciones I**
Código: S01231J Apellidos y Nombres: Aquino Alejo Angelo Percy
Código: S01309J Apellidos y Nombres: villoslada Vilca Sergio Gianfranco
Fecha: 22.10.24

1. Enunciado del proyecto

Se requiere un sistema bancario que permita a los usuarios acceder a diversas funcionalidades según su rol, optimizando los procesos relacionados con la gestión de empleados, clientes, transacciones y préstamos. El sistema debe permitir un control efectivo de los registros mediante operaciones CRUD, asegurando que las operaciones bancarias se realicen de forma rápida, segura y con una interfaz intuitiva.

2. Identificación de requerimientos

Nombre:	R1: Login
Resumen:	Verificar si los campos estan vacios para retornar una alerta, si los campos estan completos buscara esas credenciales en la base de datos el usuario y su contraseña
Entradas:	usuario: Texto y números (no permite caracteres especiales) contraseña: cualquier tipo de Texto
Resultados :	Nos mandara a la interfaz del Sistema, pero habrán 2 Roles y mostrará lo siguiente Gerente : Las opciones y interfaces de Empleados , Clientes, Transacciones, Préstamos Cajero : Las opciones y interfaces de Clientes, Transacciones, Préstamos

Nombre:	R2: Crear nuevo empleado
Resumen:	Verificar si los campos estan vacios para retornar una alerta, si los campos estan completos guardará al nuevo empleado en la base de datos
Entradas:	Nombre - Apellido - Usuario : Texto y números (no permite caracteres especiales) Contraseña: cualquier tipo de Texto

	DNI - SALARIO - N°CELULAR : Ingreso de solo números y “. “
Resultados :	Se guardará el nuevo empleado en la base de datos y se mostrará en la tabla de empleado que se ubica en la parte inferior

Nombre:	R3: Eliminar Empleado
Resumen:	Eliminará al empleado mediante su DNI
Entradas:	DNI: Solo se permite números hasta 8 digitos
Resultados :	Eliminará al empleado de la base de datos y de la tabla que está ubicado en la parte inferior

Nombre:	R3: Nuevo Empleado
Resumen:	Limpia los campos para crear un nuevo usuario
Entradas:	
Resultados :	Limpia todos los campos que se requieren para crear un nuevo usuario

Nombre:	R4: Modificar Empleado
Resumen:	Se ingresa el DNI del empleado para modificar alguna característica (N° celular - nombre , etc)
Entradas:	DNI: Solo se permite números hasta 8 digitos
Resultados	Modificara al empleado seleccionado mediante su DNI

Nombre:	R5: Crear nuevo cliente
Resumen:	Verificar si los campos estan vacios para retornar una alerta, si los campos estan completos guardará al nuevo empleado en la base de datos
Entradas:	Nombre - Apellido - Dirección : Texto y números (no permite caracteres especiales) Email: cualquier tipo de Texto DNI - SALDO - N°CELULAR - N° de cuenta: Ingreso de solo números
Resultados :	Se guardará al nuevo cliente en la base de datos y se mostrará en la tabla que se ubica en la parte inferior

Nombre:	R6: Eliminar Cliente
Resumen:	Eliminará al cliente mediante su DNI
Entradas:	DNI: Solo se permite números hasta 8 digitos

Resultados :	Eliminará al cliente de la base de datos y de la tabla que está ubicado en la parte inferior

Nombre:	R7: Nuevo Cliente
Resumen:	Limpia los campos para crear un nuevo cliente
Entradas:	
Resultados :	Limpia todos los campos que se requieren para crear un nuevo cliente

Nombre:	R8: Modificar Cliente
Resumen:	Se ingresa el DNI del Cliente para modificar alguna característica (N° celular - nombre , etc)
Entradas:	DNI: Solo se permite números hasta 8 digitos
Resultados	Modificara al cliente seleccionado mediante su DNI

Nombre:	R9: RETIRO
Resumen:	Se ingresa un número de cuenta válido y el monto que se va retirar en sus respectivos campos
Entradas:	N° de cuenta: Solo acepta valores numéricos hasta 20 digitos) Monto : Solo acepta valores numéricos
Resultados :	Se retirará monto si la cuenta es válida, si no saldrá una alerta de que la cuenta no existe

Nombre:	R10 : Deposito
Resumen:	Se ingresa un número de cuenta válido y el monto que se va retirar en sus respectivos campos
Entradas:	N° de cuenta: Solo acepta valores numéricos hasta 20 digitos) Monto : Solo acepta valores numéricos
Resultados :	Se hará el depósito si la cuenta es válida, si no saldrá una alerta de que la cuenta no existe

Nombre:	R11 : Realizar Préstamo
Resumen:	Se realiza un préstamo rellenando los campos requeridos
Entradas:	N° de cuenta: Solo acepta valores numéricos (hasta 20 dígitos) Monto : Solo acepta valores numéricos Plazo en meses: Solo acepta valores numéricos (hasta 2 dígitos)
Resultados :	Se realizará el préstamo y se guardará en la base de datos, en la tabla que se ubica en la parte inferior también nos mostrará nuestros datos de dicho préstamo que realizamos

Nombre:	R12: Nuevo Préstamo
Resumen:	Limpia los campos para realizar un nuevo préstamo
Entradas:	
Resultados:	Se limpia todos los campos que se requieren para realizar un préstamo

Nombre:	R13: Desactivar Préstamo
Resumen:	Desactiva el préstamo que se realizo con un cliente mediante su N° de cuenta
Entradas:	N° de cuenta: solo acepta valores numéricos (solo 20 digitos)
Resultados:	El estado del préstamo que se muestra en la tabla inferior cambia a “No activo”

3. **Enlace del proyecto en GitHub**

   https://github.com/LynxSVV/PROYECTO.git

4. **Diseño de Base de Datos (Script) la base de datos puede estar alojada** (es opcional) **en un servidor de la nube (**puede utilizar Microsoft Azure, Google Firebase, u otro) , mínimo tres tablas

   BASE DE DATOS

```
/****** Object:  Database [probankDB]    Script Date: 24/10/2024 22:34:46 ******/
CREATE DATABASE [probankDB] (EDITION = 'GeneralPurpose', SERVICE_OBJECTIVE = 'GP_Gen5_2', MAXSIZE = 32 GB) WITH CATALOG_COLLATION = SQL_Latin1_General_CP1_CI_AS, LEDGER = OFF;
GO
ALTER DATABASE [probankDB] SET COMPATIBILITY_LEVEL = 160
GO
ALTER DATABASE [probankDB] SET ANSI_NULL_DEFAULT OFF
GO
ALTER DATABASE [probankDB] SET ANSI_NULLS OFF
GO
ALTER DATABASE [probankDB] SET ANSI_PADDING OFF
GO
ALTER DATABASE [probankDB] SET ANSI_WARNINGS OFF
GO
ALTER DATABASE [probankDB] SET ARITHABORT OFF
GO
```

```
ALTER DATABASE [probankDB] SET AUTO_SHRINK OFF
GO
ALTER DATABASE [probankDB] SET AUTO_UPDATE_STATISTICS ON
GO
ALTER DATABASE [probankDB] SET CURSOR_CLOSE_ON_COMMIT OFF
GO
ALTER DATABASE [probankDB] SET CONCAT_NULL_YIELDS_NULL OFF
GO
ALTER DATABASE [probankDB] SET NUMERIC_ROUNDABORT OFF
GO
ALTER DATABASE [probankDB] SET QUOTED_IDENTIFIER OFF
GO
ALTER DATABASE [probankDB] SET RECURSIVE_TRIGGERS OFF
GO
ALTER DATABASE [probankDB] SET AUTO_UPDATE_STATISTICS_ASYNC OFF
GO
ALTER DATABASE [probankDB] SET ALLOW_SNAPSHOT_ISOLATION ON
GO
ALTER DATABASE [probankDB] SET PARAMETERIZATION SIMPLE
GO
ALTER DATABASE [probankDB] SET READ_COMMITTED_SNAPSHOT ON
GO
ALTER DATABASE [probankDB] SET  MULTI_USER
GO
ALTER DATABASE [probankDB] SET ENCRYPTION ON
GO
ALTER DATABASE [probankDB] SET QUERY_STORE = ON
GO
ALTER DATABASE [probankDB] SET QUERY_STORE (OPERATION_MODE = READ_WRITE, CLEANUP_POLICY = (STALE_QUERY_THRESHOLD_DAYS = 30), DATA_FLUSH_INTERVAL_SECONDS = 900, INTERVAL_LENGTH_MINUTES = 60, MAX_STORAGE_SIZE_MB = 100, QUERY_CAPTURE_MODE = AUTO, SIZE_BASED_CLEANUP_MODE = AUTO, MAX_PLANS_PER_QUERY = 200, WAIT_STATS_CAPTURE_MODE = ON)
GO
/*** The scripts of database scoped configurations in Azure should be executed inside the target database connection. ***/
GO
-- ALTER DATABASE SCOPED CONFIGURATION SET MAXDOP = 8;
GO
```

```
DBO CLIENTES
/****** Object:  Table [dbo].[cliente]    Script Date: 24/10/2024 22:34:47 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[cliente](
	[ClienteID] [int] IDENTITY(1,1) NOT NULL,
	[Nombre] [nvarchar](50) NOT NULL,
	[Apellido] [nvarchar](50) NOT NULL,
	[DNI] [char](8) NOT NULL,
	[Direccion] [nvarchar](255) NOT NULL,
	[Telefono] [nvarchar](15) NOT NULL,
	[Email] [nvarchar](100) NULL,
	[FechaRegistro] [date] NULL,
PRIMARY KEY CLUSTERED
(
	[ClienteID] ASC
)WITH (STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY],
UNIQUE NONCLUSTERED
(
	[DNI] ASC
)WITH (STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[cuentabancaria](
	[CuentaID] [int] IDENTITY(1,1) NOT NULL,
	[ClienteID] [int] NULL,
	[NumeroCuenta] [nvarchar](20) NOT NULL,
	[TipoCuenta] [nvarchar](20) NOT NULL,
	[Saldo] [decimal](15, 2) NOT NULL,
	[FechaApertura] [date] NULL,
PRIMARY KEY CLUSTERED
(
	[CuentaID] ASC
)WITH (STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY],
```

```
UNIQUE NONCLUSTERED
(
	[NumeroCuenta] ASC
)WITH (STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[empleado](
	[EmpleadoID] [int] IDENTITY(1,1) NOT NULL,
	[Nombre] [nvarchar](50) NOT NULL,
	[Apellido] [nvarchar](50) NOT NULL,
	[DNI] [char](8) NOT NULL,
	[Cargo] [nvarchar](20) NOT NULL,
	[FechaContratacion] [date] NOT NULL,
	[Salario] [decimal](10, 2) NOT NULL,
	[NumeroCelular] [nvarchar](15) NOT NULL,
PRIMARY KEY CLUSTERED
(
	[EmpleadoID] ASC
)WITH (STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY],
UNIQUE NONCLUSTERED
(
	[DNI] ASC
)WITH (STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

TABLA CLIENTES
/****** Object:  Table [dbo].[prestamos]    Script Date: 24/10/2024 22:34:47 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[prestamos](
	[PrestamoID] [int] IDENTITY(1,1) NOT NULL,
	[ClienteID] [int] NULL,
	[MontoPrestamo] [decimal](15, 2) NOT NULL,
	[Interes] [decimal](5, 2) NOT NULL,
```

```
	[Plazo] [int] NOT NULL,
	[FechaDesembolso] [date] NOT NULL,
	[Estado] [nvarchar](20) NOT NULL,
	[EmpleadoID] [int] NULL,
	[NumeroCuenta] [nvarchar](20) NULL,
PRIMARY KEY CLUSTERED
(
	[PrestamoID] ASC
)WITH (STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

TABLA TRANSSACCIONES

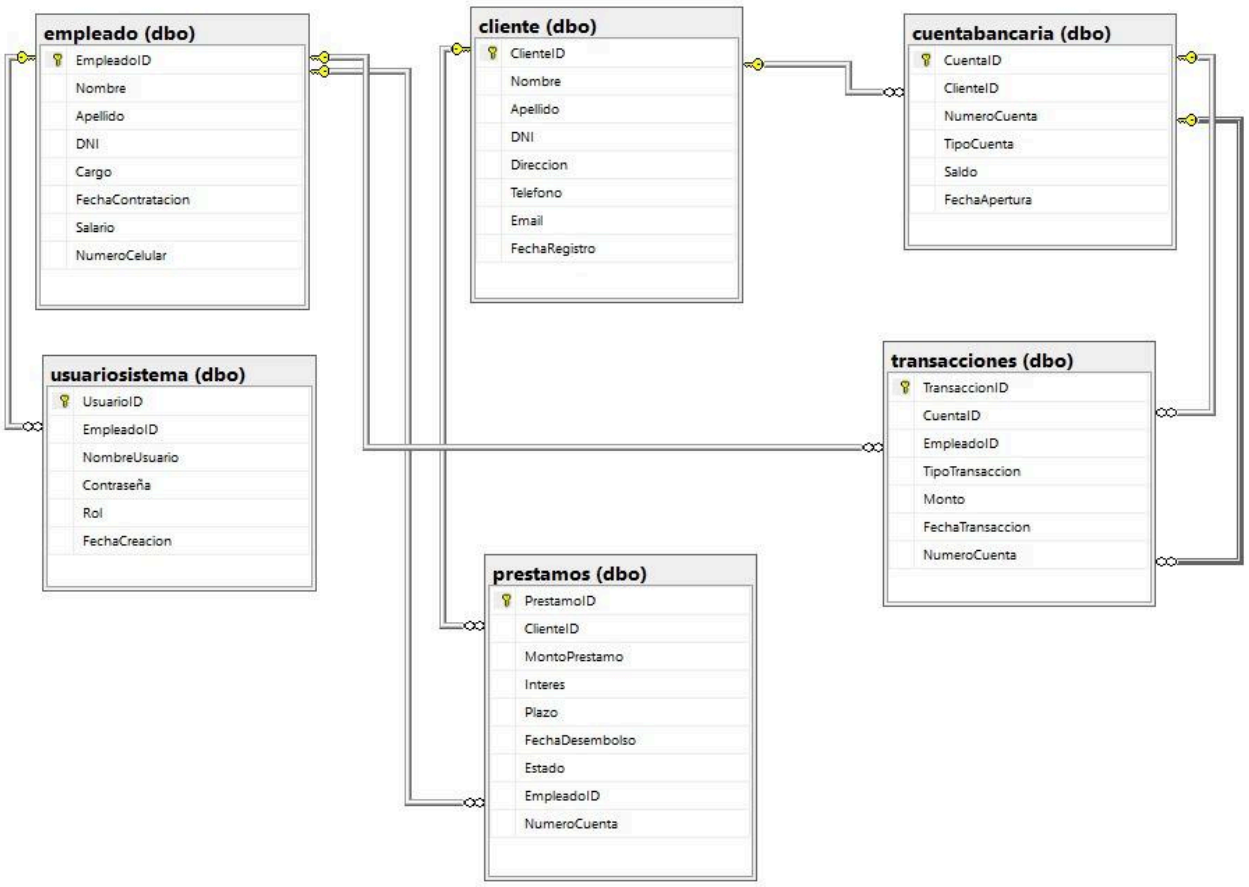
```
/****** Object:  Table [dbo].[transacciones]    Script Date: 24/10/2024 22:34:47 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[transacciones](
	[TransaccionID] [int] IDENTITY(1,1) NOT NULL,
	[CuentaID] [int] NULL,
	[EmpleadoID] [int] NULL,
	[TipoTransaccion] [nvarchar](50) NOT NULL,
	[Monto] [decimal](15, 2) NOT NULL,
	[FechaTransaccion] [datetime] NULL,
	[NumeroCuenta] [nvarchar](20) NULL,
PRIMARY KEY CLUSTERED
(
	[TransaccionID] ASC
)WITH (STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

USUARIO SISTEMAS

```
/****** Object:  Table [dbo].[usuariosistema]    Script Date: 24/10/2024 22:34:47 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[usuariosistema](
	[UsuarioID] [int] IDENTITY(1,1) NOT NULL,
	[EmpleadoID] [int] NULL,
```

```
	[NombreUsuario] [nvarchar](50) NOT NULL,
	[Contraseña] [nvarchar](255) NOT NULL,
	[Rol] [nvarchar](20) NOT NULL,
	[FechaCreacion] [datetime] NULL,
PRIMARY KEY CLUSTERED
(
	[UsuarioID] ASC
)WITH (STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY],
UNIQUE NONCLUSTERED
(
	[NombreUsuario] ASC
)WITH (STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[cliente] ADD  DEFAULT (getdate()) FOR [FechaRegistro]
GO
ALTER TABLE [dbo].[cuentabancaria] ADD  DEFAULT (getdate()) FOR [FechaApertura]
GO
ALTER TABLE [dbo].[transacciones] ADD DEFAULT (getdate()) FOR
[FechaTransaccion]
GO
ALTER TABLE [dbo].[usuariosistema] ADD  DEFAULT (getdate()) FOR [FechaCreacion]
GO
ALTER TABLE [dbo].[cuentabancaria]  WITH CHECK ADD FOREIGN KEY([ClienteID])
REFERENCES [dbo].[cliente] ([ClienteID])
GO
ALTER TABLE [dbo].[prestamos]  WITH CHECK ADD FOREIGN KEY([ClienteID])
REFERENCES [dbo].[cliente] ([ClienteID])
GO
ALTER TABLE [dbo].[prestamos]  WITH CHECK ADD FOREIGN KEY([EmpleadoID])
REFERENCES [dbo].[empleado] ([EmpleadoID])
GO
ALTER TABLE [dbo].[transacciones]  WITH CHECK ADD FOREIGN KEY([CuentaID])
REFERENCES [dbo].[cuentabancaria] ([CuentaID])
GO
ALTER TABLE [dbo].[transacciones]  WITH CHECK ADD FOREIGN KEY([EmpleadoID])
REFERENCES [dbo].[empleado] ([EmpleadoID])
GO
ALTER TABLE [dbo].[transacciones] WITH CHECK ADD CONSTRAINT
[FK_NumeroCuenta] FOREIGN KEY([NumeroCuenta])
REFERENCES [dbo].[cuentabancaria] ([NumeroCuenta])
GO
```

```
ALTER TABLE [dbo].[transacciones] CHECK CONSTRAINT [FK_NumeroCuenta]
GO
ALTER TABLE [dbo].[usuariosistema] WITH CHECK ADD FOREIGN KEY([EmpleadoID])
REFERENCES [dbo].[empleado] ([EmpleadoID])
GOALTER DATABASE [probankDB] SET  READ_WRITE
GO
```



5. Código explicado del proyecto

PAQUETE CONTROLLER

CLASE CLIENTECONTROLLER

```
package Controller;
import Model.Cliente;
import Model.Empleado;
import Model.UsuarioDAO;
import java.sql.Date;
import java.util.ArrayList;
import java.util.List;
public class ClienteController {
    private UsuarioDAO usuarioDao;
   public ClienteController() {
```

```
        usuarioDao = new UsuarioDAO();
    }
    //Insertamos el Cliente y Retornamos el ClienteID
    public int agregarCliente(String nombre, String apellido, String dni, String direccion,
String celular, String email, Date fechaContratacion){
        return usuarioDao.insertarCliente(nombre, apellido, dni, direccion, celular, email,
fechaContratacion);
    }
    //Retornar todos los clientes
    public List<Cliente> obtenerTodosLosClientes() {
    List<Cliente> clientes = new ArrayList<>();
    try {
        clientes = usuarioDao.obtenerClientes();
    } catch (Exception e) {
            System.err.println("Error al obtener empleados desde el controlador: " +
e.getMessage());
    }
    return clientes;
    }
    //Obtener el ClienteID por el dni
    public int obtenerClienteIDPorDni(String dni) {
        int ClienteID = -1;
        try {
            ClienteID = usuarioDao.obtenerClienteIDPorDNI(dni);
        } catch (Exception e) {
            System.err.println("Error al obtener el Cliente: " + e.getMessage());
        }
        return ClienteID;
    }
    //Eliminar el cliiente por DNI
    public void eliminarClientePorDNI(String dni) {
        try {
            usuarioDao.eliminarClientePorDNI(dni);
        } catch (Exception e) {
            System.err.println("Error al eliminar el Cliente: " + e.getMessage());
        }
    }
    //Obtener datos del Cliente y habilitar para modificar
    public Cliente obtenerClientePorDniModificar(String dni) {
        return usuarioDao.obtenerClientePorDNIModificar(dni);
    }
    //Aplicar cambios en modificar
     public boolean actualizarCliente(String dni, String nombre, String apellido, String
Direccion, String numeroCelular,String email) {
```

```
            return usuarioDao.actualizarCliente(dni, nombre, apellido, Direccion,
numeroCelular, email);
    }
}
```

CLASE CUENTABANCARIACONTROLLER

```
package Controller;
import Model.CuentaBancaria;
import Model.UsuarioDAO;
import Model.UsuarioSistema;
import java.sql.Date;
import java.util.ArrayList;
import java.util.List;
public class CuentaBancariaController {
    private UsuarioDAO usuarioDao;
    public CuentaBancariaController() {
        usuarioDao = new UsuarioDAO();
    }
    //Insertamos la Cuenta Bancaria del cliente Referente al ID del Cliente y retornamos
true o false
    public boolean agregarCuentaBancaria(int ClienteID, String numCuenta, String
tipoCuenta, double Saldo, Date fechaApertura) {
        return usuarioDao.insertarCuentaBancaria(ClienteID, numCuenta, tipoCuenta,
Saldo, fechaApertura);
    }
    //Retornamos todas las cuentas bancarias
    public List<CuentaBancaria> obtenerTodasLasCuentas() {
    List<CuentaBancaria> cuentas = new ArrayList<>();
    try {
        cuentas = usuarioDao.obtenerCuentasBancarias();
    } catch (Exception e) {
        System.err.println("Error al obtener empleados desde el controlador: " +
e.getMessage());
    }
    return cuentas;
    }
    //Eliminar cuenta bancaria por EmpleadoID
    public void eliminarCuentaBancariaPorEmpleadoID(int empleadoID) {
        try {
            usuarioDao.eliminarCuentaBancariaPorClienteID(empleadoID);
        } catch (Exception e) {
            System.err.println("Error al eliminar el usuario: " + e.getMessage());
        }
    }
    //Obtener la cuenta bancaria para habilitar el modificar Mediante el dni
```

```
    public CuentaBancaria obtenerCuentaBancariaPorDniModificar(String dni) {
        return usuarioDao.obtenerCuentaBancariaPorDNIModificar(dni);
    }
    //Aplicar cambios de modificar
    public boolean actualizarCuentaBancaria(String dni, String tipoCuenta) {
        return usuarioDao.actualizarCuentaBancaria(dni, tipoCuenta);
    }
}
```

CLASE EMPLEADOCONTROLLER

```
package Controller;
import Model.Empleado;
import Model.UsuarioDAO;
import Model.UsuarioSistema;
import java.sql.Date;
import java.util.ArrayList;
import java.util.List;
public class EmpleadoController {
   private UsuarioDAO empleadoDao;
    public EmpleadoController() {
        this.empleadoDao = new UsuarioDAO();
    }
    //Insertamos el Empleado y Retornamos el EmpleadoID
     public int agregarEmpleado(String nombre, String apellido, String dni, String cargo,
Date fechaContratacion, double salario, String numeroCelular) {
                    return empleadoDao.insertarEmpleado(nombre, apellido, dni, cargo,
fechaContratacion, salario, numeroCelular);
    }
   //Retornamos los empleados y su ID para la tabla Empleados
    public List<Empleado> obtenerTodosLosEmpleados() {
    List<Empleado> empleados = new ArrayList<>();
    try {
        empleados = empleadoDao.obtenerEmpleados();
    } catch (Exception e) {
            System.err.println("Error al obtener empleados desde el controlador: " +
e.getMessage());
    }
    return empleados;
    }
    //Obtener el EmpleadoID base al DNI para aplicar eliminar usuario por EmpleadoID
    public int obtenerEmpleadoIDPorDni(String dni) {
        int empleadoID = -1;
        try {
            empleadoID = empleadoDao.obtenerEmpleadoIDPorDNI(dni);
        } catch (Exception e) {
```

```
            System.err.println("Error al obtener el EmpleadoID: " + e.getMessage());
        }
        return empleadoID;
    }
    //Eliminar el Empleado por DNI
    public void eliminarEmpleadoPorDNI(String dni) {
        try {
            empleadoDao.eliminarEmpleadoPorDNI(dni);
        } catch (Exception e) {
            System.err.println("Error al eliminar el empleado: " + e.getMessage());
        }
    }
    //Obtenemos el empleado Para habilitar el modificar mediante el DNI
    public Empleado obtenerEmpleadoPorDniModificar(String dni) {
        return empleadoDao.obtenerEmpleadoPorDNIModificar(dni);
    }
    //Actualizamos el Empleado mediante el DNI
    public boolean actualizarEmpleado(String dni, String nombre, String apellido, String
cargo, double salario, String numeroCelular) {
            return empleadoDao.actualizarEmpleado(dni, nombre, apellido, cargo, salario,
numeroCelular);
    }
}
```

CLASE PRESTAMOSCONTROLLER

```
package Controller;
import Model.Cliente;
import Model.Prestamos;
import Model.UsuarioDAO;
import java.util.List;
public class PrestamosController {
    private UsuarioDAO usuarioDao;
    public PrestamosController() {
        this.usuarioDao = new UsuarioDAO();
    }
    public Integer obtenerClienteIDPorNumeroCuenta(String numeroCuenta) {
        return usuarioDao.obtenerClienteIDPorNumeroCuenta(numeroCuenta);
    }
    public double calcularInteres(int plazoMeses, double monto) {
        return usuarioDao.calcularInteres(plazoMeses, monto);
    }
        public boolean insertarPrestamo(int clienteID, double montoPrestamo, double
interes, int plazo, java.sql.Date fechaDesembolso, int empleadoID, String
numeroCuenta){
```

```
        return usuarioDao.insertarPrestamo(clienteID, montoPrestamo, interes, plazo, fechaDesembolso, empleadoID, numeroCuenta);
    }
    public Integer obtenerEmpleadoIdPorUsuario(String usuario){
        return usuarioDao.obtenerEmpleadoIDporUsuario(usuario);
    }
    public boolean existePrestamoPorNumeroCuenta(String numeroCuenta) {
        return usuarioDao.existePrestamoPorNumeroCuenta(numeroCuenta);
    }
    public boolean desactivarPrestamo(String numeroCuenta) {
        return usuarioDao.desactivarPrestamo(numeroCuenta);
    }
    public List<Prestamos> obtenerTodosLosPrestamos(){
        return usuarioDao.obtenerTodosLosPrestamos();
    }
    public List<Cliente> obtenerTodosLosClientes(){
        return usuarioDao.obtenerTodosLosClientesNombreApellido();
    }
}
```

TRANSACCIONESCONTROLLER

```
package Controller;
import Model.Transacciones;
import Model.UsuarioDAO;
public class TransaccionesController {
    private UsuarioDAO usuarioDao;
    public TransaccionesController() {
        usuarioDao = new UsuarioDAO();
    }
    public Integer obtenerEmpleadoIdPorUsuario(String usuario){
        return usuarioDao.obtenerEmpleadoIDporUsuario(usuario);
    }
    //obtener CuentaID por numero de cuenta
    public Integer obtenerCuentaIdPorNumeroCuenta(String numeroCuenta) {
        return usuarioDao.obtenerCuentaIDPorNumeroCuenta(numeroCuenta);
    }
    //aplicar retiro
    public boolean restarMontoDeCuenta(int cuentaID, double monto){
        return usuarioDao.restarMontoDeCuenta(cuentaID, monto);
    }
        public boolean insertarTransaccionRetiro(int cuentaID, int empleadoID, String tipoTransaccion, double monto, String numeroCuenta){
                    return usuarioDao.insertarTransaccionRetiro(cuentaID, empleadoID, tipoTransaccion, monto, numeroCuenta);
    }
```

```
    public boolean SumarMontoDeCuenta(int cuentaID, double monto){
        return usuarioDao.sumarMontoACuenta(cuentaID, monto);
    }
      public boolean insertarTransaccionDeposito(int cuentaID, int empleadoID, String
tipoTransaccion, double monto, String numeroCuenta){
                    return  usuarioDao.insertarTransaccionDeposito(cuentaID,  empleadoID,
tipoTransaccion, monto, numeroCuenta);
    }
    public String obtenerNombrePorNumeroCuenta(String numeroCuenta) {
        return usuarioDao.obtenerNombrePorNumeroCuenta(numeroCuenta);
    }
    public Transacciones obtenerUltimaOperacionConFecha() {
    Transacciones operacion = usuarioDao.obtenerUltimaOperacionConFecha();
    if (operacion == null) {
        // Manejo de casos en que no hay operaciones
        System.out.println("No se encontraron transacciones.");
    }
    return operacion; // Devuelve el objeto Transacciones o null si no hay operaciones
    }
}
```

CLASE USUARIOSISTEMACONTROLLER

```
package Controller;
import Model.Empleado;
import Model.UsuarioDAO;
import Model.UsuarioSistema;
import java.sql.Date;
import java.util.ArrayList;
import java.util.List;
public class UsuarioSistemaController {
    private UsuarioDAO usuarioDao;
    public UsuarioSistemaController() {
        usuarioDao = new UsuarioDAO();
    }
    //Para el ingreso del usuario al sistema
    public boolean verificarCredenciales(String nombreUsuario, String contraseña) {
        try {
            return usuarioDao.verificarCredenciales(nombreUsuario, contraseña);
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }
    //Retarnamos el rol de Usuario para Mostrarlo en la interface
    public String verificarRol(String usuarioId) {
```

```
        String rol = usuarioDao.obtenerRol(usuarioId);
        if (rol != null) {
            if (rol.equals("Gerente")) {
                return "Gerente";
            } else if (rol.equals("Cajero")) {
                return "Cajero";
            } else {
                System.out.println("Rol no reconcido");
                return null;
            }
        } else {
            System.out.println("No se puede obtener el rol de usuario");;
            return null;
        }
    }
    //Retornamos el nombre y apellido para mostrarlo en la interface
    public String obtenerNombreCompleto(String nombreUsuario) {
    return usuarioDao.obtenerNombreCompleto(nombreUsuario);
}
    //Insertamos el usuario Referente al ID del Empleado y retornamos true o false
        public boolean agregarUsuario(int empleadoID, String nombreUsuario, String
contraseña, String rol, Date fechaCreacion) {
         return usuarioDao.insertarUsuario(empleadoID, nombreUsuario, contraseña, rol,
fechaCreacion);
    }
    //Retornamos los usuarios y su ID para la tabla Empleados
    public List<UsuarioSistema> obtenerTodosLosUsuarios() {
    List<UsuarioSistema> usuarioSistem = new ArrayList<>();
    try {
        usuarioSistem = usuarioDao.obtenerUsuarios();
    } catch (Exception e) {
            System.err.println("Error al obtener empleados desde el controlador: " +
e.getMessage());
    }
    return usuarioSistem;
    }
    //Eliminar usuario por EmpleadoID
    public void eliminarUsuarioPorEmpleadoID(int empleadoID) {
        try {
            usuarioDao.eliminarUsuarioPorEmpleadoID(empleadoID);
        } catch (Exception e) {
            System.err.println("Error al eliminar el usuario: " + e.getMessage());
        }
    }
```

```
    //Obtener el Usuario para habilitar el modificar Mediante el dni
    public UsuarioSistema obtenerUsuarioPorDniModificar(String dni) {
        return usuarioDao.obtenerUsuarioPorDNIModificar(dni);
    }
    //Actualizamos el Usuario mediante el DNI
        public boolean actualizarUsuario(String dni, String nombreUsuario, String contraseña) {
        return usuarioDao.actualizarUsuario(dni, nombreUsuario, contraseña);
    }
}
```

PAQUETE “MODEL”

CLASE CLIENTE

```
package Model;
public class Transacciones {
    private int transaccionID;
    private int cuentaID;
    private int empleadoID;
    private String tipoTransaccion; // 'Depósito', 'Retiro', etc.
    private double monto;
    private String fechaTransaccion;

    public Transacciones() {
    }
    public Transacciones(int transaccionID, String fechaTransaccion) {
        this.transaccionID = transaccionID;
        this.fechaTransaccion = fechaTransaccion;
    }
        public Transacciones(int transaccionID, int cuentaID, int empleadoID, String tipoTransaccion, double monto, String fechaTransaccion) {
        this.transaccionID = transaccionID;
        this.cuentaID = cuentaID;
        this.empleadoID = empleadoID;
        this.tipoTransaccion = tipoTransaccion;
        this.monto = monto;
        this.fechaTransaccion = fechaTransaccion;
    }
    public int getTransaccionID() {
```

```
        return transaccionID;
    }
    public void setTransaccionID(int transaccionID) {
        this.transaccionID = transaccionID;
    }
    public int getCuentaID() {
        return cuentaID;
    }
    public void setCuentaID(int cuentaID) {
        this.cuentaID = cuentaID;
    }
    public int getEmpleadoID() {
        return empleadoID;
    }
    public void setEmpleadoID(int empleadoID) {
        this.empleadoID = empleadoID;
    }
    public String getTipoTransaccion() {
        return tipoTransaccion;
    }
    public void setTipoTransaccion(String tipoTransaccion) {
        this.tipoTransaccion = tipoTransaccion;
    }
    public double getMonto() {
        return monto;
    }
    public void setMonto(double monto) {
        this.monto = monto;
    }
    public String getFechaTransaccion() {
        return fechaTransaccion;
    }

    public void setFechaTransaccion(String fechaTransaccion) {
        this.fechaTransaccion = fechaTransaccion;
    }
}
```

CLASE CUENTABANCARIA

```
package Model;
public class CuentaBancaria {
    // Atributos privados para encapsular la información de la cuenta bancaria
    private int cuentaID; // Identificador único de la cuenta bancaria
    private int clienteID; // Identificador del cliente asociado a la cuenta
    private String numeroCuenta; // Número de cuenta bancaria (único)
```

```
    private String tipoCuenta; // Tipo de cuenta: 'Ahorros' o 'Corriente'
    private double saldo; // Saldo actual de la cuenta
    private String fechaApertura; // Fecha en que se abrió la cuenta
    // Constructor por defecto (sin parámetros)
    public CuentaBancaria() {
    }
    // Constructor con parámetros para inicializar todos los atributos de la clase
        public CuentaBancaria(int cuentaID, int clienteID, String numeroCuenta, String
tipoCuenta, double saldo, String fechaApertura) {
        this.cuentaID = cuentaID; // Asigna el identificador de la cuenta
        this.clienteID = clienteID; // Asigna el identificador del cliente asociado
        this.numeroCuenta = numeroCuenta; // Asigna el número de cuenta bancaria
        this.tipoCuenta = tipoCuenta; // Asigna el tipo de cuenta ('Ahorros' o 'Corriente')
        this.saldo = saldo; // Asigna el saldo inicial de la cuenta
        this.fechaApertura = fechaApertura; // Asigna la fecha de apertura de la cuenta
    }
    // Métodos getter y setter para cada atributo
    // Permiten acceder y modificar los valores de los atributos de la clase
    public int getCuentaID() {
        return cuentaID; // Devuelve el ID de la cuenta
    }
    public void setCuentaID(int cuentaID) {
        this.cuentaID = cuentaID; // Asigna el ID de la cuenta
    }
    public int getClienteID() {
        return clienteID; // Devuelve el ID del cliente
    }
    public void setClienteID(int clienteID) {
        this.clienteID = clienteID; // Asigna el ID del cliente
    }
    public String getNumeroCuenta() {
        return numeroCuenta; // Devuelve el número de cuenta bancaria
    }
    public void setNumeroCuenta(String numeroCuenta) {
        this.numeroCuenta = numeroCuenta; // Asigna el número de cuenta bancaria
    }
    public String getTipoCuenta() {
        return tipoCuenta; // Devuelve el tipo de cuenta ('Ahorros' o 'Corriente')
    }
    public void setTipoCuenta(String tipoCuenta) {
        this.tipoCuenta = tipoCuenta; // Asigna el tipo de cuenta
    }
    public double getSaldo() {
        return saldo; // Devuelve el saldo de la cuenta
```

```
    }
    public void setSaldo(double saldo) {
        this.saldo = saldo; // Asigna el saldo de la cuenta
    }
    public String getFechaApertura() {
        return fechaApertura; // Devuelve la fecha de apertura de la cuenta
    }
    public void setFechaApertura(String fechaApertura) {
        this.fechaApertura = fechaApertura; // Asigna la fecha de apertura de la cuenta
    }
}
```

CLASE EMPLEADO

```
package Model;

public class Empleado {

    // Atributos privados para encapsular la información del empleado
    private int empleadoID; // Identificador único del empleado
    private String nombre; // Nombre del empleado
    private String apellido; // Apellido del empleado
    private String dni; // Documento Nacional de Identidad del empleado
    private String cargo; // Cargo del empleado: 'Cajero' o 'Gerente'
    private String fechaContratacion; // Fecha de contratación del empleado
    private double salario; // Salario del empleado
    private String numeroCelular; // Número de celular del empleado

    // Constructor por defecto (sin parámetros)
    public Empleado() {
    }

    // Constructor con parámetros para inicializar todos los atributos de la clase
     public Empleado(int empleadoID, String nombre, String apellido, String dni, String
cargo, String fechaContratacion, double salario, String numeroCelular) {
        this.empleadoID = empleadoID; // Asigna el identificador del empleado
        this.nombre = nombre; // Asigna el nombre del empleado
        this.apellido = apellido; // Asigna el apellido del empleado
        this.dni = dni; // Asigna el DNI del empleado
        this.cargo = cargo; // Asigna el cargo del empleado (Cajero o Gerente)
        this.fechaContratacion = fechaContratacion; // Asigna la fecha de contratación del
empleado
        this.salario = salario; // Asigna el salario del empleado
        this.numeroCelular = numeroCelular; // Asigna el número de celular del empleado
    }
```

```java
// Métodos getter y setter para cada atributo
// Permiten acceder y modificar los valores de los atributos de la clase

public int getEmpleadoID() {
    return empleadoID; // Devuelve el ID del empleado
}

public void setEmpleadoID(int empleadoID) {
    this.empleadoID = empleadoID; // Asigna el ID del empleado
}

public String getNombre() {
    return nombre; // Devuelve el nombre del empleado
}

public void setNombre(String nombre) {
    this.nombre = nombre; // Asigna el nombre del empleado
}

public String getApellido() {
    return apellido; // Devuelve el apellido del empleado
}

public void setApellido(String apellido) {
    this.apellido = apellido; // Asigna el apellido del empleado
}

public String getDni() {
    return dni; // Devuelve el DNI del empleado
}

public void setDni(String dni) {
    this.dni = dni; // Asigna el DNI del empleado
}

public String getCargo() {
    return cargo; // Devuelve el cargo del empleado (Cajero o Gerente)
}

public void setCargo(String cargo) {
    this.cargo = cargo; // Asigna el cargo del empleado
}

public String getFechaContratacion() {
```

```
        return fechaContratacion; // Devuelve la fecha de contratación del empleado
    }

    public void setFechaContratacion(String fechaContratacion) {
        this.fechaContratacion = fechaContratacion; // Asigna la fecha de contratación del
empleado
    }

    public double getSalario() {
        return salario; // Devuelve el salario del empleado
    }

    public void setSalario(double salario) {
        this.salario = salario; // Asigna el salario del empleado
    }

    public String getNumeroCelular() {
        return numeroCelular; // Devuelve el número de celular del empleado
    }

    public void setNumeroCelular(String numeroCelular) {
        this.numeroCelular = numeroCelular; // Asigna el número de celular del empleado
    }
}
```

CLASE PRÉSTAMO

```
package Model;

public class Prestamos {

    // Atributos privados para encapsular la información del préstamo
    private int prestamoID; // Identificador único del préstamo
    private int clienteID; // Identificador del cliente asociado al préstamo
    private double montoPrestamo; // Monto total del préstamo otorgado
    private double interes; // Porcentaje de interés aplicado al préstamo
    private int plazo; // Número de meses para pagar el préstamo
        private String fechaDesembolso; // Fecha en que se realizó el desembolso del
préstamo
    private String estado; // Estado del préstamo: 'Activo' o 'Pagado'
    private int empleadoID; // Identificador del empleado que aprobó el préstamo
    private String numeroCuenta; // Número de cuenta asociada al préstamo
```

```
    // Constructor por defecto (sin parámetros)
    public Prestamos() {
    }

    // Constructor con parámetros para inicializar todos los atributos de la clase
        public Prestamos(int prestamoID, int clienteID, double montoPrestamo, double interes, int plazo, String fechaDesembolso, String estado, int empleadoID, String numeroCuenta) {
        this.prestamoID = prestamoID; // Asigna el identificador del préstamo
        this.clienteID = clienteID; // Asigna el identificador del cliente
        this.montoPrestamo = montoPrestamo; // Asigna el monto del préstamo
        this.interes = interes; // Asigna el interés del préstamo
        this.plazo = plazo; // Asigna el plazo de pago del préstamo en meses
        this.fechaDesembolso = fechaDesembolso; // Asigna la fecha de desembolso
        this.estado = estado; // Asigna el estado del préstamo
        this.empleadoID = empleadoID; // Asigna el identificador del empleado
          this.numeroCuenta = numeroCuenta; // Asigna el número de cuenta asociada al préstamo
    }

    // Métodos getter y setter para cada atributo
    // Permiten acceder y modificar los valores de los atributos de la clase

    public int getPrestamoID() {
        return prestamoID; // Devuelve el identificador del préstamo
    }

    public void setPrestamoID(int prestamoID) {
        this.prestamoID = prestamoID; // Asigna el identificador del préstamo
    }

    public int getClienteID() {
        return clienteID; // Devuelve el identificador del cliente
    }

    public void setClienteID(int clienteID) {
        this.clienteID = clienteID; // Asigna el identificador del cliente
    }

    public double getMontoPrestamo() {
        return montoPrestamo; // Devuelve el monto del préstamo
    }

    public void setMontoPrestamo(double montoPrestamo) {
```

```
    this.montoPrestamo = montoPrestamo; // Asigna el monto del préstamo
}

public double getInteres() {
    return interes; // Devuelve el interés aplicado al préstamo
}

public void setInteres(double interes) {
    this.interes = interes; // Asigna el interés del préstamo
}

public int getPlazo() {
    return plazo; // Devuelve el plazo en meses para el pago del préstamo
}

public void setPlazo(int plazo) {
    this.plazo = plazo; // Asigna el plazo del préstamo en meses
}

public String getFechaDesembolso() {
    return fechaDesembolso; // Devuelve la fecha de desembolso del préstamo
}

public void setFechaDesembolso(String fechaDesembolso) {
    this.fechaDesembolso = fechaDesembolso; // Asigna la fecha de desembolso
}

public String getEstado() {
    return estado; // Devuelve el estado del préstamo ('Activo' o 'Pagado')
}

public void setEstado(String estado) {
    this.estado = estado; // Asigna el estado del préstamo
}

public int getEmpleadoID() {
    return empleadoID; // Devuelve el identificador del empleado
}

public void setEmpleadoID(int empleadoID) {
    this.empleadoID = empleadoID; // Asigna el identificador del empleado
}

public String getNumeroCuenta() {
```

```
        return numeroCuenta; // Devuelve el número de cuenta asociada al préstamo
    }

    public void setNumeroCuenta(String numeroCuenta) {
        this.numeroCuenta = numeroCuenta; // Asigna el número de cuenta asociada al
préstamo
    }
}
```

CLASE TRANSACCIONES

```
package Model;

public class Transacciones {

    // Atributos privados para encapsular la información de la transacción
    private int transaccionID; // Identificador único de la transacción
    private int cuentaID; // Identificador de la cuenta asociada a la transacción
    private int empleadoID; // Identificador del empleado que realizó la transacción
    private String tipoTransaccion; // Tipo de transacción: 'Depósito', 'Retiro', etc.
    private double monto; // Monto de la transacción
    private String fechaTransaccion; // Fecha en la que se realizó la transacción

    // Constructor por defecto (sin parámetros)
    public Transacciones() {
    }

     // Constructor con dos parámetros, utilizado en caso de que solo se tenga el ID y la
fecha
    public Transacciones(int transaccionID, String fechaTransaccion) {
        this.transaccionID = transaccionID; // Asigna el ID de la transacción
        this.fechaTransaccion = fechaTransaccion; // Asigna la fecha de la transacción
    }

       // Constructor con todos los atributos para inicializar un objeto Transacciones
completo
        public Transacciones(int transaccionID, int cuentaID, int empleadoID, String
tipoTransaccion, double monto, String fechaTransaccion) {
        this.transaccionID = transaccionID; // Asigna el ID de la transacción
        this.cuentaID = cuentaID; // Asigna el ID de la cuenta involucrada
            this.empleadoID = empleadoID; // Asigna el ID del empleado que realizó la
transacción
        this.tipoTransaccion = tipoTransaccion; // Asigna el tipo de transacción
        this.monto = monto; // Asigna el monto de la transacción
        this.fechaTransaccion = fechaTransaccion; // Asigna la fecha de la transacción
```

```
    }

    // Métodos getter y setter para cada atributo
    // Permiten acceder y modificar los valores de los atributos de la clase

    public int getTransaccionID() {
        return transaccionID; // Devuelve el identificador de la transacción
    }

    public void setTransaccionID(int transaccionID) {
        this.transaccionID = transaccionID; // Asigna el identificador de la transacción
    }

    public int getCuentaID() {
        return cuentaID; // Devuelve el identificador de la cuenta involucrada
    }

    public void setCuentaID(int cuentaID) {
        this.cuentaID = cuentaID; // Asigna el identificador de la cuenta involucrada
    }

    public int getEmpleadoID() {
        return empleadoID; // Devuelve el identificador del empleado
    }

    public void setEmpleadoID(int empleadoID) {
        this.empleadoID = empleadoID; // Asigna el identificador del empleado
    }

    public String getTipoTransaccion() {
            return tipoTransaccion; // Devuelve el tipo de transacción ('Depósito', 'Retiro',
etc.)
    }

    public void setTipoTransaccion(String tipoTransaccion) {
        this.tipoTransaccion = tipoTransaccion; // Asigna el tipo de transacción
    }

    public double getMonto() {
        return monto; // Devuelve el monto de la transacción
    }

    public void setMonto(double monto) {
        this.monto = monto; // Asigna el monto de la transacción
```

```
    }

    public String getFechaTransaccion() {
        return fechaTransaccion; // Devuelve la fecha en la que se realizó la transacción
    }

    public void setFechaTransaccion(String fechaTransaccion) {
        this.fechaTransaccion = fechaTransaccion; // Asigna la fecha de la transacción
    }
}
```

CLASE USUARIODAO

```
package Model;
import java.math.BigDecimal;
import java.sql.Connection;
import java.sql.Date;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.Timestamp;
import java.text.SimpleDateFormat;
import java.time.ZoneId;
import java.time.ZonedDateTime;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Calendar;
import java.util.List;
import java.util.TimeZone;

public class UsuarioDAO {
    private String url = "jdbc:sqlserver://serverweb2024.database.windows.net;databaseName=probankDB";
    private String usuario = "Aquino9461";
    private String contraseña = "angelo.159632478";

    private Connection conectar() throws SQLException {
        return DriverManager.getConnection(url, usuario, contraseña);
    }
    //Para el ingreso del usuario al sistema
    public boolean verificarCredenciales(String nombreUsuario, String contraseña) {
     String sql = "SELECT Contraseña FROM usuariosistema WHERE NombreUsuario = ?"; //esta es la consulta
```

```
            try (Connection conexion = conectar(); PreparedStatement pstmt =
conexion.prepareStatement(sql)) { //La conexion y aplicamos PreparedStatement por
temas de seguridad
        pstmt.setString(1, nombreUsuario); //Asignamos el valor con el parametro
        ResultSet rs = pstmt.executeQuery(); //Ejecutamos la consulta
        if (rs.next()) { //toma el primer valor del campo y registro
            byte[] contraseñaEncriptadaBD = rs.getBytes("Contraseña");
                                                    String contraseñaDesencriptada =
desencriptarContraseña(contraseñaEncriptadaBD); //Desencriptamos para validad la
contra
            if (contraseñaDesencriptada != null) {
                return contraseñaDesencriptada.equals(contraseña); //Retornamos true si la
contra es correcta (si existe en ese usuario)
            } else {
                System.out.println("Error: La contraseña desencriptada es nula.");
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return false; // Retorna false si hubo un error o no se encontró
}
    //Desencriptar la onntraseña
    private String desencriptarContraseña(byte[] contraseñaEncriptada) {
        String contraseñaDesencriptada = null;
                        try (Connection conexion = conectar(); Statement stmt =
conexion.createStatement()) {
            // Abre la clave simétrica
                stmt.execute("OPEN SYMMETRIC KEY MiClaveSimetrica DECRYPTION BY
PASSWORD = 'angelo.159632478';");
            // Desencriptar la contraseña
                String sql = "SELECT CAST(DECRYPTBYKEY(?) AS NVARCHAR(255)) AS
Desencriptado";
            try (PreparedStatement pstmt = conexion.prepareStatement(sql)) {
                pstmt.setBytes(1, contraseñaEncriptada);
                ResultSet rs = pstmt.executeQuery();
                if (rs.next()) {
                    contraseñaDesencriptada = rs.getString("Desencriptado");
                }
            }
            // Cierra la clave
            stmt.execute("CLOSE SYMMETRIC KEY MiClaveSimetrica;");
        } catch (SQLException e) {
            e.printStackTrace();
```

```
        }
        return contraseñaDesencriptada;
    }
    //Encriptar la contraseña
    private byte[] encriptarContraseña(String contraseña) {
        byte[] contraseñaEncriptada = null;
                        try (Connection conexion = conectar(); Statement stmt =
conexion.createStatement()) {
            // Abrir la clave simétrica en SQL Server
                stmt.execute("OPEN SYMMETRIC KEY MiClaveSimetrica DECRYPTION BY
PASSWORD = 'angelo.159632478';");

            // Encriptar la contraseña
            String sql = "SELECT ENCRYPTBYKEY(KEY_GUID('MiClaveSimetrica'), ?)";
            try (PreparedStatement pstmt = conexion.prepareStatement(sql)) {
                        pstmt.setString(1, contraseña); // Insertar la contraseña en formato
NVARCHAR
                ResultSet rs = pstmt.executeQuery();
                if (rs.next()) {
                    contraseñaEncriptada = rs.getBytes(1); // Obtener el resultado en formato
VARBINARY (byte[])
                }
            }

            // Cerrar la clave simétrica
            stmt.execute("CLOSE SYMMETRIC KEY MiClaveSimetrica;");
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return contraseñaEncriptada;
    }
    //Retarnamos el rol de Usuario para Mostrarlo en la interface
    public String obtenerRol(String nombreUsuario) {
        String rol = null;
         String sql = "SELECT rol FROM usuariosistema WHERE NombreUsuario = ?"; //
Cambia "NombreUsuario" si tu columna tiene un nombre diferente

             try (Connection connection = conectar(); // Usa el método correcto para la
conexión
             PreparedStatement statement = connection.prepareStatement(sql)) {

                    statement.setString(1, nombreUsuario); // Asigna el valor del nombre de
usuario al primer parámetro de la consulta
            ResultSet resultSet = statement.executeQuery();
```

```
            if (resultSet.next()) {
                rol = resultSet.getString("rol"); // Cambia "rol" si tu columna tiene un nombre diferente
            }
        } catch (SQLException e) {
            e.printStackTrace(); // Manejo de excepciones
        }

        return rol; // Devuelve el rol obtenido o null si no se encuentra
    }
    //Retornamos el nombre y apellido para mostrarlo en la interface
    public String obtenerNombreCompleto(String nombreUsuario) {
        String sql = "SELECT e.Nombre, e.Apellido " +
                     "FROM empleado e " +
                     "INNER JOIN usuariosistema u ON e.EmpleadoID = u.EmpleadoID " +
                     "WHERE u.NombreUsuario = ?";
                      try (Connection conexion = conectar(); PreparedStatement pstmt = conexion.prepareStatement(sql)) {
            pstmt.setString(1, nombreUsuario);
            ResultSet rs = pstmt.executeQuery();
            if (rs.next()) {
                String nombre = rs.getString("Nombre");
                String apellido = rs.getString("Apellido");
                return nombre + " " + apellido;
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return null;
    }
    //Insertamos el Emp leado y Retornamos el EmpleadoID
     public int insertarEmpleado(String nombre, String apellido, String dni, String cargo, Date fechaContratacion, double salario, String numeroCelular) {
                    String sql = "INSERT INTO empleado (Nombre, Apellido, Dni, Cargo, FechaContratacion, Salario, NumeroCelular) " +
                     "VALUES (?, ?, ?, ?, ?, ?, ?)";

                      try (Connection conexion = conectar(); PreparedStatement stmt = conexion.prepareStatement(sql, PreparedStatement.RETURN_GENERATED_KEYS)) {
            stmt.setString(1, nombre);
            stmt.setString(2, apellido);
            stmt.setString(3, dni);
            stmt.setString(4, cargo);
```

```
            stmt.setDate(5, fechaContratacion);
            stmt.setDouble(6, salario);
            stmt.setString(7, numeroCelular);

             int filasAfectadas = stmt.executeUpdate(); //Cuantas filas fueron afectada debe
ser 1
            if (filasAfectadas > 0) {
                ResultSet rs = stmt.getGeneratedKeys();
                if (rs.next()) {
                    return rs.getInt(1);
                }
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }

        return 0;
        }
    //Insertamos el usuario Referente al ID del Empleado y retornamos true o false
         public boolean insertarUsuario(int empleadoID, String nombreUsuario, String
contraseña, String rol, Date fechaCreacion) {
                String sql = "INSERT INTO usuariosistema (EmpleadoID, NombreUsuario,
Contraseña, Rol, FechaCreacion) " +
                    "VALUES (?, ?, ?, ?, ?)";

                     try (Connection conexion = conectar(); PreparedStatement stmt =
conexion.prepareStatement(sql)) {
            stmt.setInt(1, empleadoID);
            stmt.setString(2, nombreUsuario);
                        stmt.setBytes(3, encriptarContraseña(contraseña)); // Aplicamos la
encriptacion de la contraseña
            stmt.setString(4, rol);
            stmt.setDate(5, fechaCreacion);

            int filasAfectadas = stmt.executeUpdate();
            return filasAfectadas > 0;
        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }
    }
     //Retornamos los empleados y su ID para la tabla Empleados
     public List<Empleado> obtenerEmpleados() {
        List<Empleado> empleados = new ArrayList<>();
```

```
        String sql = "SELECT * FROM empleado";

                try (Connection conexion = conectar(); PreparedStatement stmt =
conexion.prepareStatement(sql); ResultSet rs = stmt.executeQuery()) {
            while (rs.next()) {
                Empleado empleado = new Empleado();
                empleado.setEmpleadoID(rs.getInt("EmpleadoID"));
                empleado.setNombre(rs.getString("Nombre"));
                empleado.setApellido(rs.getString("Apellido"));
                empleado.setDni(rs.getString("DNI"));
                empleado.setCargo(rs.getString("Cargo"));
                empleado.setFechaContratacion(rs.getString("FechaContratacion"));
                empleado.setSalario(rs.getDouble("Salario"));
                empleado.setNumeroCelular(rs.getString("NumeroCelular"));
                empleados.add(empleado);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return empleados;
    }
    //Retornamos los usuarios y su ID para la tabla Empleados
    public List<UsuarioSistema> obtenerUsuarios() {
        List<UsuarioSistema> usuarios = new ArrayList<>();
    String sql = "SELECT EmpleadoID, NombreUsuario FROM usuariosistema";

            try (Connection conexion = conectar(); PreparedStatement stmt =
conexion.prepareStatement(sql); ResultSet rs = stmt.executeQuery()) {
        while (rs.next()) {
            UsuarioSistema usuario = new UsuarioSistema();
            usuario.setEmpleadoID(rs.getInt("EmpleadoID"));
            String nombreUsuario = rs.getString("NombreUsuario");
            usuario.setNombreUsuario(nombreUsuario);
            usuarios.add(usuario);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return usuarios;
    }
    //Obtener el EmpleadoID base al DNI para aplicar eliminar usuario por EmpleadoID
    public int obtenerEmpleadoIDPorDNI(String dni) {
        int empleadoID = -1; // Valor por defecto si no se encuentra el empleado
        String sql = "SELECT EmpleadoID FROM empleado WHERE DNI = ?";
```

```
            try (Connection conexion = conectar(); PreparedStatement stmt =
conexion.prepareStatement(sql)) {
                stmt.setString(1, dni);
                ResultSet rs = stmt.executeQuery();
                if (rs.next()) {
                    empleadoID = rs.getInt("EmpleadoID");
                }
            } catch (SQLException e) {
                e.printStackTrace();
            }
            return empleadoID;
        }
        //Eliminar el Empleado por DNI
        public void eliminarEmpleadoPorDNI(String dni) {
            String sql = "DELETE FROM empleado WHERE DNI = ?";

            try (Connection conexion = conectar(); PreparedStatement stmt =
conexion.prepareStatement(sql)) {
                stmt.setString(1, dni);
                //obtener el numero de filas afectas
                stmt.executeUpdate();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        //Eliminar usuario por EmpleadoID
        public void eliminarUsuarioPorEmpleadoID(int empleadoID) {
            String sql = "DELETE FROM usuariosistema WHERE EmpleadoID = ?";

            try (Connection conexion = conectar(); PreparedStatement stmt =
conexion.prepareStatement(sql)) {
                stmt.setInt(1, empleadoID);
                stmt.executeUpdate();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        //Obtenemos el empleado Para habilitar el modificar mediante el DNI
        public Empleado obtenerEmpleadoPorDNIModificar(String dni) {
        Empleado empleado = null;
        String sql = "SELECT Nombre, Apellido, Cargo, Salario, NumeroCelular " +
                     "FROM Empleado " +
                     "WHERE DNI = ?";
        try (Connection conexion = conectar();
```

```
         PreparedStatement stmt = conexion.prepareStatement(sql)) {
        stmt.setString(1, dni);
        ResultSet rs = stmt.executeQuery();

        if (rs.next()) {
            empleado = new Empleado();
            empleado.setNombre(rs.getString("Nombre"));
            empleado.setApellido(rs.getString("Apellido"));
            empleado.setCargo(rs.getString("Cargo"));
            empleado.setSalario(rs.getDouble("Salario"));
            empleado.setNumeroCelular(rs.getString("NumeroCelular"));
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return empleado;
}
    //Obtener el Usuario para habilitar el modificar Mediante el dni
    public UsuarioSistema obtenerUsuarioPorDNIModificar(String dni) {
        UsuarioSistema usuarioSistema = null;
        String sql = "SELECT us.NombreUsuario, us.Contraseña " +
                     "FROM usuariosistema us " +
                     "JOIN Empleado e ON us.EmpleadoID = e.EmpleadoID " +
                     "WHERE e.DNI = ?";

        try (Connection conexion = conectar();
             PreparedStatement stmt = conexion.prepareStatement(sql)) {
            stmt.setString(1, dni);
            ResultSet rs = stmt.executeQuery();

            if (rs.next()) {
                usuarioSistema = new UsuarioSistema();
                usuarioSistema.setNombreUsuario(rs.getString("NombreUsuario"));

usuarioSistema.setContraseña(desencriptarContraseña(rs.getBytes("Contraseña"))); //
Guarda la contraseña encriptada
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
            return usuarioSistema; // Retorna el objeto UsuarioSistema con la información
obtenida
    }
    //Actualizamos el Empleado mediante el DNI
```

```
    public boolean actualizarEmpleado(String dni, String nombre, String apellido, String
cargo, double salario, String numeroCelular) {
    String sql = "UPDATE empleado " +
                 "SET Nombre = ?, Apellido = ?, Cargo = ?, Salario = ?, NumeroCelular = ? " +
                 "WHERE DNI = ?";

             try (Connection conexion = conectar(); PreparedStatement stmt =
conexion.prepareStatement(sql)) {
        stmt.setString(1, nombre);
        stmt.setString(2, apellido);
        stmt.setString(3, cargo);
        stmt.setDouble(4, salario);
        stmt.setString(5, numeroCelular);
        stmt.setString(6, dni);

        int filasAfectadas = stmt.executeUpdate();
        return filasAfectadas > 0;
    } catch (SQLException e) {
        e.printStackTrace();
        return false;
    }
}
    //Actualizamos el Usuario mediante el DNI
          public boolean actualizarUsuario(String dni, String nombreUsuario, String
contraseña) {
    String sql = "UPDATE usuariosistema " +
                 "SET NombreUsuario = ?, Contraseña = ? " +
                  "WHERE EmpleadoID = (SELECT EmpleadoID FROM empleado WHERE DNI
= ?)";

             try (Connection conexion = conectar(); PreparedStatement stmt =
conexion.prepareStatement(sql)) {
        stmt.setString(1, nombreUsuario);
        stmt.setBytes(2, encriptarContraseña(contraseña)); //Encriptamos al contra
        stmt.setString(3, dni);

        int filasAfectadas = stmt.executeUpdate();
        return filasAfectadas > 0;

    } catch (SQLException e) {
        e.printStackTrace();
        return false;
    }
}
```

```
    //Insertamos el Cliente y Retornamos el ClienteID
    public int insertarCliente(String nombre, String apellido, String dni, String direccion,
String celular, String email ,Date fechaContratacion ) {
        String sql = "INSERT INTO cliente (Nombre, Apellido, DNI, Direccion, Telefono,
Email, FechaRegistro) " +
                "VALUES (?, ?, ?, ?, ?, ?, ?)";

                try (Connection conexion = conectar(); PreparedStatement stmt =
conexion.prepareStatement(sql, PreparedStatement.RETURN_GENERATED_KEYS)) {
        stmt.setString(1, nombre);
        stmt.setString(2, apellido);
        stmt.setString(3, dni);
        stmt.setString(4, direccion);
        stmt.setString(5, celular);
        stmt.setString(6, email);
        stmt.setDate(7, fechaContratacion);

        int filasAfectadas = stmt.executeUpdate();
        if (filasAfectadas > 0) {
            ResultSet rs = stmt.getGeneratedKeys();
            if (rs.next()) {
                return rs.getInt(1);
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }

    return 0;
    }
    //Insertamos la Cuenta Bancaria del cliente Referente al ID del Cliente y retornamos
true o false
    public boolean insertarCuentaBancaria(int ClienteID, String numCuenta, String
tipoCuenta, double Saldo, Date fechaApertura) {
            String sql = "INSERT INTO cuentaBancaria (ClienteID, NumeroCuenta,
TipoCuenta, Saldo, FechaApertura) " +
                "VALUES (?, ?, ?, ?, ?)";

                try (Connection conexion = conectar(); PreparedStatement stmt =
conexion.prepareStatement(sql)) {
        stmt.setInt(1, ClienteID);
        stmt.setString(2, numCuenta);
        stmt.setString(3, tipoCuenta);
        stmt.setDouble(4,Saldo );
```

```
            stmt.setDate(5, fechaApertura);

            int filasAfectadas = stmt.executeUpdate();
            return filasAfectadas > 0;
        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }
    }
    //Retornamos los Clientes y su ID para la tabla cuenta Bancaria
    public List<Cliente> obtenerClientes() {
        List<Cliente> clientes = new ArrayList<>();
        String sql = "SELECT * FROM cliente";

                try (Connection conexion = conectar(); PreparedStatement stmt =
conexion.prepareStatement(sql); ResultSet rs = stmt.executeQuery()) {
            while (rs.next()) {
                Cliente cliente = new Cliente();
                cliente.setClienteID(rs.getInt("ClienteID"));
                cliente.setNombre(rs.getString("Nombre"));
                cliente.setApellido(rs.getString("Apellido"));
                cliente.setDni(rs.getString("DNI"));
                cliente.setDireccion(rs.getString("Direccion"));
                cliente.setTelefono(rs.getString("Telefono"));
                cliente.setEmail(rs.getString("Email"));
                cliente.setFechaRegistro(rs.getString("FechaRegistro"));
                clientes.add(cliente);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return clientes;
    }
    //Retornamos las cuentas bancarias y su ID para la tabla Empleados
    public List<CuentaBancaria> obtenerCuentasBancarias() {
        List<CuentaBancaria> cuentas = new ArrayList<>();
       String sql = "SELECT ClienteID,NumeroCuenta,TipoCuenta,Saldo,FechaApertura
FROM cuentabancaria";

            try (Connection conexion = conectar(); PreparedStatement stmt =
conexion.prepareStatement(sql); ResultSet rs = stmt.executeQuery()) {
        while (rs.next()) {
            CuentaBancaria cuenta = new CuentaBancaria();
            cuenta.setClienteID(rs.getInt("ClienteID"));
```

```
            cuenta.setNumeroCuenta(rs.getString("NumeroCuenta"));
            cuenta.setTipoCuenta(rs.getString("TipoCuenta"));
            cuenta.setSaldo(rs.getDouble("Saldo"));
            cuentas.add(cuenta);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return cuentas;
    }
        //Obtener el ClienteID base al DNI para aplicar eliminar cuenta bancaria por EmpleadoID
    public int obtenerClienteIDPorDNI(String dni) {
        int ClienteID = -1;
        String sql = "SELECT ClienteID FROM cliente WHERE DNI = ?";
                    try (Connection conexion = conectar(); PreparedStatement stmt = conexion.prepareStatement(sql)) {
            stmt.setString(1, dni);
            ResultSet rs = stmt.executeQuery();
            if (rs.next()) {
                ClienteID = rs.getInt("ClienteID");
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return ClienteID;
    }
    //Eliminar el Cliente por DNI
    public void eliminarClientePorDNI(String dni) {
        String sql = "DELETE FROM cliente WHERE DNI = ?";

                    try (Connection conexion = conectar(); PreparedStatement stmt = conexion.prepareStatement(sql)) {
            stmt.setString(1, dni);
            //obtener el numero de filas afectas
            stmt.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    //Eliminar cuenta bancaria por ClienteID
    public void eliminarCuentaBancariaPorClienteID(int ClienteID) {
        String sql = "DELETE FROM cuentabancaria WHERE ClienteID = ?";
```

```
                try (Connection conexion = conectar(); PreparedStatement stmt =
conexion.prepareStatement(sql)) {
            stmt.setInt(1, ClienteID);
            stmt.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    //Obtenemos el cliente Para habilitar el modificar mediante el DNI
    public Cliente obtenerClientePorDNIModificar(String dni) {
    Cliente cliente = null;
     String sql = "SELECT Nombre, Apellido, Direccion, Telefono, Email, FechaRegistro "
+
                "FROM cliente " +
                "WHERE DNI = ?";
    try (Connection conexion = conectar();
        PreparedStatement stmt = conexion.prepareStatement(sql)) {
        stmt.setString(1, dni);
        ResultSet rs = stmt.executeQuery();

        if (rs.next()) {
            cliente = new Cliente();
            cliente.setNombre(rs.getString("Nombre"));
            cliente.setApellido(rs.getString("Apellido"));
            cliente.setDireccion(rs.getString("Direccion"));
            cliente.setTelefono(rs.getString("Telefono"));
            cliente.setEmail(rs.getString("Email"));
            cliente.setFechaRegistro(rs.getString("FechaRegistro"));
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return cliente;
}
    //Obtener la cuenta bancaria para habilitar el modificar Mediante el dni
    public CuentaBancaria obtenerCuentaBancariaPorDNIModificar(String dni) {
        CuentaBancaria cuentabancaria = null;
        String sql = "SELECT cb.NumeroCuenta " +
                    "FROM cuentabancaria cb " +
                    "JOIN cliente c ON cb.ClienteID = c.ClienteID " +
                    "WHERE c.DNI = ?";

        try (Connection conexion = conectar();
            PreparedStatement stmt = conexion.prepareStatement(sql)) {
```

```
            stmt.setString(1, dni);
            ResultSet rs = stmt.executeQuery();

            if (rs.next()) {
                cuentabancaria = new CuentaBancaria();
                cuentabancaria.setTipoCuenta(rs.getString("NumeroCuenta"));
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
            return cuentabancaria; // Retorna el objeto UsuarioSistema con la información obtenida
    }
    //Actualizamos el Cliente mediante el DNI
     public boolean actualizarCliente(String dni, String nombre, String apellido, String direccion, String celular, String email) {
    String sql = "UPDATE cliente " +
                 "SET Nombre = ?, Apellido = ?, Direccion = ?, Telefono = ?, Email = ?" +
                 "WHERE DNI = ?";

             try (Connection conexion = conectar(); PreparedStatement stmt = conexion.prepareStatement(sql)) {
        stmt.setString(1, nombre);
        stmt.setString(2, apellido);
        stmt.setString(3, direccion);
        stmt.setString(4, celular);
        stmt.setString(5, email);
        stmt.setString(6, dni);

        int filasAfectadas = stmt.executeUpdate();
        return filasAfectadas > 0;
    } catch (SQLException e) {
        e.printStackTrace();
        return false;
    }
}
    //Actualizamos el Usuario mediante el DNI
    public boolean actualizarCuentaBancaria(String dni, String tipoCuenta) {
    String sql = "UPDATE cuentabancaria " +
                 "SET TipoCuenta = ?" +
                 "WHERE ClienteID = (SELECT ClienteID FROM cliente WHERE DNI = ?)";

             try (Connection conexion = conectar(); PreparedStatement stmt = conexion.prepareStatement(sql)) {
```

```
            stmt.setString(1, tipoCuenta);
            stmt.setString(2, dni);

            int filasAfectadas = stmt.executeUpdate();
            return filasAfectadas > 0;

        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }
    }
    //obtener EmpleadoID por usuario
    public Integer obtenerEmpleadoIDporUsuario(String nombreUsuario) {
            String sql = "SELECT e.EmpleadoID " + // Agregué un espacio después de EmpleadoID
                    "FROM empleado e " +
                    "INNER JOIN usuariosistema u ON e.EmpleadoID = u.EmpleadoID " +
                    "WHERE u.NombreUsuario = ?";
                     try (Connection conexion = conectar(); PreparedStatement pstmt = conexion.prepareStatement(sql)) {
            pstmt.setString(1, nombreUsuario);
            ResultSet rs = pstmt.executeQuery();
            if (rs.next()) {
                return rs.getInt("EmpleadoID"); // Retorna el EmpleadoID como un Integer
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return null; // Retorna null si no se encuentra el empleado
    }
    //obtener CuentaID por numero de cuenta
    public Integer obtenerCuentaIDPorNumeroCuenta(String numeroCuenta) {
    String sql = "SELECT CuentaID FROM cuentabancaria WHERE NumeroCuenta = ?";
            try (Connection conexion = conectar(); PreparedStatement pstmt = conexion.prepareStatement(sql)) {
        pstmt.setString(1, numeroCuenta);
        ResultSet rs = pstmt.executeQuery();
        if (rs.next()) {
            return rs.getInt("CuentaID");
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return null; // Si no se encuentra la cuenta
```

```
    }
    //aplicar retiro
    public boolean restarMontoDeCuenta(int cuentaID, double monto) {
    String sql = "UPDATE cuentabancaria SET Saldo = Saldo - ? WHERE CuentaID = ?";
            try (Connection conexion = conectar(); PreparedStatement pstmt =
conexion.prepareStatement(sql)) {
        pstmt.setDouble(1, monto);
        pstmt.setInt(2, cuentaID);
        int filasAfectadas = pstmt.executeUpdate();
        return filasAfectadas > 0; // Retorna true si se actualizó el saldo
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return false; // Retorna false si hubo un error
}
    // Método para insertar la transacción
        public boolean insertarTransaccionRetiro(int cuentaID, int empleadoID, String
tipoTransaccion, double monto, String numeroCuenta) {
                String sql = "INSERT INTO transacciones (CuentaID, EmpleadoID,
TipoTransaccion, Monto, FechaTransaccion, NumeroCuenta) VALUES (?, ?, ?, ?, ?,
?)";

        // Obtener la fecha actual en la zona horaria de Perú
                                                    Calendar calendar =
Calendar.getInstance(TimeZone.getTimeZone("America/Lima"));
        java.util.Date fechaActual = calendar.getTime();
        java.sql.Timestamp timestamp = new java.sql.Timestamp(fechaActual.getTime());

                try (Connection conexion = conectar(); PreparedStatement pstmt =
conexion.prepareStatement(sql)) {
            pstmt.setInt(1, cuentaID);
            pstmt.setInt(2, empleadoID);
            pstmt.setString(3, tipoTransaccion);
            pstmt.setDouble(4, monto);
                pstmt.setTimestamp(5, timestamp); // Establecer la fecha actual como un
Timestamp
            pstmt.setString(6, numeroCuenta);

            int filasAfectadas = pstmt.executeUpdate();
            return filasAfectadas > 0; // Retorna true si se insertó la transacción
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return false; // Retorna false si hubo un error
```

```
    }
    // Método para sumar monto a la cuenta
    public boolean sumarMontoACuenta(int cuentaID, double monto) {
        String sql = "UPDATE cuentabancaria SET Saldo = Saldo + ? WHERE CuentaID = ?";
                try (Connection conexion = conectar(); PreparedStatement pstmt = conexion.prepareStatement(sql)) {
            pstmt.setDouble(1, monto);
            pstmt.setInt(2, cuentaID);
            int filasAfectadas = pstmt.executeUpdate();
            return filasAfectadas > 0; // Retorna true si se actualizó el saldo
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return false; // Retorna false si hubo un error
    }
    // Método para insertar la transacción de depósito
      public boolean insertarTransaccionDeposito(int cuentaID, int empleadoID, String tipoTransaccion, double monto, String numeroCuenta) {
                String sql = "INSERT INTO transacciones (CuentaID, EmpleadoID, TipoTransaccion, Monto, FechaTransaccion, NumeroCuenta) VALUES (?, ?, ?, ?, ?, ?)";

                                                    Calendar calendar = Calendar.getInstance(TimeZone.getTimeZone("America/Lima"));
        java.util.Date fechaActual = calendar.getTime();
        java.sql.Timestamp timestamp = new java.sql.Timestamp(fechaActual.getTime());

                try (Connection conexion = conectar(); PreparedStatement pstmt = conexion.prepareStatement(sql)) {
            pstmt.setInt(1, cuentaID);
            pstmt.setInt(2, empleadoID);
            pstmt.setString(3, tipoTransaccion);
            pstmt.setDouble(4, monto);
                  pstmt.setTimestamp(5, timestamp); // Establecer la fecha actual como un Timestamp
            pstmt.setString(6, numeroCuenta);

            int filasAfectadas = pstmt.executeUpdate();
            return filasAfectadas > 0; // Retorna true si se insertó la transacción
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return false; // Retorna false si hubo un error
```

```
    }
    public String obtenerNombrePorNumeroCuenta(String numeroCuenta) {
      String sql = "SELECT cliente.Nombre, cliente.Apellido FROM cliente INNER JOIN cuentabancaria ON cliente.ClienteID = cuentabancaria.ClienteID WHERE cuentabancaria.NumeroCuenta = ?";
    String nombreCompleto = null;

            try (Connection conexion = conectar(); PreparedStatement pstmt = conexion.prepareStatement(sql)) {
        pstmt.setString(1, numeroCuenta);
        ResultSet rs = pstmt.executeQuery();

        if (rs.next()) {
            String nombre = rs.getString("Nombre");
            String apellido = rs.getString("Apellido");
            // Concatenar nombre y apellido
              nombreCompleto = nombre + " " + apellido; // Puedes ajustar el formato si lo deseas
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }

    return nombreCompleto;
}
    public Transacciones obtenerUltimaOperacionConFecha() {
                String sql = "SELECT TOP 1 TransaccionID, FechaTransaccion FROM transacciones ORDER BY TransaccionID DESC";
        Transacciones transaccion = null;

                try (Connection conexion = conectar(); PreparedStatement pstmt = conexion.prepareStatement(sql)) {
            ResultSet rs = pstmt.executeQuery();

            if (rs.next()) {
                int transaccionID = rs.getInt("TransaccionID");
                    Timestamp fechaTransaccion = rs.getTimestamp("FechaTransaccion"); // Obtener Timestamp

                // Formatear la fecha y hora
                    SimpleDateFormat formatoFecha = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
                String fechaFormateada = formatoFecha.format(fechaTransaccion);
```

```
            // Crear una nueva instancia de Operacion
            transaccion = new Transacciones(transaccionID, fechaFormateada);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }

    return transaccion;
}
public Integer obtenerClienteIDPorNumeroCuenta(String numeroCuenta) {
    String sql = "SELECT ClienteID FROM cuentabancaria WHERE NumeroCuenta =
?";
    Integer clienteID = null;

            try (Connection conexion = conectar(); PreparedStatement pstmt =
conexion.prepareStatement(sql)) {
        pstmt.setString(1, numeroCuenta);
        ResultSet rs = pstmt.executeQuery();

        if (rs.next()) {
            clienteID = rs.getInt("ClienteID");
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }

    return clienteID;
}
public double calcularInteres(int plazoMeses, double monto) {
if (plazoMeses < 6) {
    return monto * 0.05 ; // 5% para menos de 6 meses
} else {
    return monto * 0.10; // 10% para 1 año o más
}
}
    public boolean insertarPrestamo(int clienteID, double montoPrestamo, double
interes, int plazo, java.sql.Date fechaDesembolso, int empleadoID, String
numeroCuenta) {
    String sql = "INSERT INTO prestamos (ClienteID, MontoPrestamo, Interes, Plazo,
FechaDesembolso, Estado, EmpleadoID, NumeroCuenta) VALUES (?, ?, ?, ?, ?, ?, ?,
?)";

// Estado se establece en "activo"
String estado = "activo";
```

```
        try (Connection conexion = conectar(); PreparedStatement pstmt =
conexion.prepareStatement(sql)) {
        pstmt.setInt(1, clienteID);
        pstmt.setDouble(2, montoPrestamo);
        pstmt.setDouble(3, interes);
        pstmt.setInt(4, plazo);
         pstmt.setDate(5, fechaDesembolso); // Asegúrate de que fechaDesembolso es un
objeto java.sql.Date
        pstmt.setString(6, estado);
        pstmt.setInt(7, empleadoID);
        pstmt.setString(8, numeroCuenta);

        int filasAfectadas = pstmt.executeUpdate();
        return filasAfectadas > 0; // Retorna true si se insertó el préstamo
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return false; // Retorna false si hubo un error
}
    public boolean existePrestamoPorNumeroCuenta(String numeroCuenta) {
        String sql = "SELECT COUNT(*) FROM prestamos WHERE NumeroCuenta = ?";
                try (Connection conexion = conectar(); PreparedStatement pstmt =
conexion.prepareStatement(sql)) {
            pstmt.setString(1, numeroCuenta);
            ResultSet rs = pstmt.executeQuery();

            if (rs.next()) {
                return rs.getInt(1) > 0; // Retorna true si hay préstamos asociados
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return false; // Retorna false si hubo un error
    }

    // Método para desactivar el préstamo
    public boolean desactivarPrestamo(String numeroCuenta) {
        String sql = "UPDATE prestamos SET Estado = 'No activo' WHERE NumeroCuenta
= ?";
                try (Connection conexion = conectar(); PreparedStatement pstmt =
conexion.prepareStatement(sql)) {
            pstmt.setString(1, numeroCuenta);
            int filasAfectadas = pstmt.executeUpdate();
```

```
            return filasAfectadas > 0; // Retorna true si se actualizó el estado
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return false; // Retorna false si hubo un error
    }
    //Listar los prestamos
    public List<Prestamos> obtenerTodosLosPrestamos() {
    List<Prestamos> prestamos = new ArrayList<>();
      String sql = "SELECT p.MontoPrestamo, p.Interes, p.Plazo, p.FechaDesembolso,
p.Estado, p.PrestamoID, cb.NumeroCuenta " +
                "FROM prestamos p " +
                "INNER JOIN cuentabancaria cb ON p.NumeroCuenta = cb.NumeroCuenta";

             try (Connection conexion = conectar(); PreparedStatement pstmt =
conexion.prepareStatement(sql);
         ResultSet rs = pstmt.executeQuery()) {

        while (rs.next()) {
            // Crear un nuevo objeto Prestamos
            Prestamos prestamo = new Prestamos();
            prestamo.setPrestamoID(rs.getInt("PrestamoID"));
            prestamo.setMontoPrestamo(rs.getDouble("MontoPrestamo"));
            prestamo.setInteres(rs.getDouble("Interes"));
            prestamo.setPlazo(rs.getInt("Plazo"));
            prestamo.setFechaDesembolso(rs.getDate("FechaDesembolso").toString());
            prestamo.setEstado(rs.getString("Estado"));
            prestamo.setNumeroCuenta(rs.getString("NumeroCuenta"));

            // Agregar a la lista
            prestamos.add(prestamo);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }

    return prestamos;
}

    public List<Cliente> obtenerTodosLosClientesNombreApellido() {
    List<Cliente> clientes = new ArrayList<>();
    String sql = "SELECT c.ClienteID, c.Nombre, c.Apellido " +
                "FROM cliente c " +
                "INNER JOIN cuentabancaria cb ON c.ClienteID = cb.ClienteID " +
```

```
                "INNER JOIN prestamos p ON cb.NumeroCuenta = p.NumeroCuenta"; // Aquí se unen las tablas para obtener clientes con préstamos

            try (Connection conexion = conectar(); PreparedStatement pstmt = conexion.prepareStatement(sql);
         ResultSet rs = pstmt.executeQuery()) {

        while (rs.next()) {
            int clienteID = rs.getInt("ClienteID");
            String nombre = rs.getString("Nombre");
            String apellido = rs.getString("Apellido");

            // Crear un objeto Cliente y agregarlo a la lista
            Cliente cliente = new Cliente(clienteID, nombre, apellido, "", "", "", "", "");
            clientes.add(cliente);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }

    return clientes;
}
```

CLASE USUARIO SISTEMA

```
package Model;

public class UsuarioSistema {

    // Atributos privados que almacenan la información del usuario del sistema
    private int usuarioID; // Identificador único del usuario
    private int empleadoID; // Identificador del empleado asociado a este usuario
    private String nombreUsuario; // Nombre de usuario del sistema
    private String contraseña; // Contraseña del usuario (debe ser cifrada)
    private String rol; // Rol del usuario ('Cajero' o 'Gerente')
    private String fechaCreacion; // Fecha en la que se creó la cuenta de usuario

    // Constructor por defecto (sin parámetros)
```

```
    public UsuarioSistema() {
    }

    // Constructor con parámetros para inicializar todos los atributos
     public UsuarioSistema(int usuarioID, int empleadoID, String nombreUsuario, String
contraseña, String rol, String fechaCreacion) {
        this.usuarioID = usuarioID; // Asigna el ID del usuario
        this.empleadoID = empleadoID; // Asigna el ID del empleado asociado
        this.nombreUsuario = nombreUsuario; // Asigna el nombre de usuario
         this.contraseña = contraseña; // Asigna la contraseña (debe ser cifrada antes de
almacenar)
        this.rol = rol; // Asigna el rol del usuario en el sistema
        this.fechaCreacion = fechaCreacion; // Asigna la fecha de creación de la cuenta
    }

    // Métodos getter y setter para acceder y modificar los atributos

    public int getUsuarioID() {
        return usuarioID; // Devuelve el identificador único del usuario
    }

    public void setUsuarioID(int usuarioID) {
        this.usuarioID = usuarioID; // Asigna el identificador único del usuario
    }

    public int getEmpleadoID() {
        return empleadoID; // Devuelve el identificador del empleado asociado
    }

    public void setEmpleadoID(int empleadoID) {
        this.empleadoID = empleadoID; // Asigna el identificador del empleado asociado
    }

    public String getNombreUsuario() {
        return nombreUsuario; // Devuelve el nombre de usuario
    }

    public void setNombreUsuario(String nombreUsuario) {
        this.nombreUsuario = nombreUsuario; // Asigna el nombre de usuario
    }

    public String getContraseña() {
        return contraseña; // Devuelve la contraseña del usuario
    }
```

```
    public void setContraseña(String contraseña) {
        this.contraseña = contraseña; // Asigna la contraseña del usuario
    }

    public String getRol() {
        return rol; // Devuelve el rol del usuario ('Cajero' o 'Gerente')
    }

    public void setRol(String rol) {
        this.rol = rol; // Asigna el rol del usuario ('Cajero' o 'Gerente')
    }

    public String getFechaCreacion() {
        return fechaCreacion; // Devuelve la fecha en que fue creada la cuenta del usuario
    }

    public void setFechaCreacion(String fechaCreacion) {
        this.fechaCreacion = fechaCreacion; // Asigna la fecha de creación de la cuenta
del usuario
    }
```

JFRAME

```
package View;
import Controller.ClienteController;
import Controller.CuentaBancariaController;
import Controller.EmpleadoController;
import Controller.PrestamosController;
import Controller.TransaccionesController;
import Controller.UsuarioSistemaController;
import Model.Cliente;
import Model.CuentaBancaria;
import Model.Empleado;
import Model.Prestamos;
import Model.Transacciones;
import Model.UsuarioDAO;
import Model.UsuarioSistema;
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.sql.Date;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.HashMap;
```

```
import java.util.List;
import java.util.Map;
import javax.swing.DefaultListCellRenderer;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.table.DefaultTableModel;
import java.text.SimpleDateFormat;


public class SistemaBancarioLogin extends javax.swing.JFrame {
   String usuario;
   String dniInsertadoModificarEmpleado;
   String dniInsertadoModificarCliente;

   public SistemaBancarioLogin() {
      initComponents();
      formulario();
      iniciarElementos();
   }
      //Elementos la interface del Jframe
      private void formulario(){
         this.setTitle("EMPRESA EMPLEADOS");
         this.setLocationRelativeTo(this);
         this.setResizable(false);
         this.setSize(new Dimension(1100,650));
      }
      //Elementos la interface del login
      public void formularioLogin(){
           SistemaBancarioLogin.setLocationRelativeTo(null); //Centrar la ventana en
la pantalla
            SistemaBancarioLogin.setResizable(false); //La ventana no se modifique
por el usuario
         SistemaBancarioLogin.setSize(new Dimension(550,300));
      }
      //Limpiar los campos de Empleado
      public void limpiarEmpleado(){
         txtNombreEmpleado.setText("");
         txtApellidoEmpleado.setText("");
         txtDniEmpleado.setText("");
         cboRolEmpleado.setSelectedIndex(0);
         txtFechaInicioEmpleado.setText("");
         txtSalarioEmpleado.setText("");
         txtCelularEmpleado.setText("");
         txtUsuarioEmpleado.setText("");
```

```
    txtContraseñaEmpleado.setText("");
    txtNombreEmpleado.requestFocus();
}
//Limpiar los campos Cliente
public void limpiarCliente(){
    txtNombreCliente.setText("");
    txtApellidoCliente.setText("");
    txtDniCliente.setText("");
    txtDireccionCliente.setText("");
    txtCelularCliente.setText("");
    txtEmailCliente.setText("");
    txtFechaRegistroCliente.setText("");
    txtNumeroCuentaCliente.setText("");
    cboTipoCuenta.setSelectedIndex(0);
    txtSaldoCliente.setText("");
    txtNombreCliente.requestFocus();
}
//Limpiar los campos de Vaucher
public void limpiarVaucher(){
    txtVaucherNombre.setText("Nombre :");
    txtVaucherNumeroCuenta.setText("Numero de cuenta:");
    txtVaucherMonto.setText("Monto ");
    txtVaucherNumeroCuenta.setText("Nº de Operacion");
    txtVaucherFecha.setText("Fecha: ");
    txtVaucherEmpleadoID.setText("Codigo Empleado");
}
//Limpiar los campo de Prestamo
public void limpiarPrestamo(){
    txtNumeroCuentaPrestamo.setText("");
    txtNumeroCuentaPrestamo.setText("");
    txtNumeroCuentaPrestamo.setText("");
    txtNumeroCuentaPrestamo.setText("");
}
//Funcion para habiliar todo del gerente
public void vistaGerente(){
    mostrarEmpleados();
}
//Funcion para habilitar todo del cajero
public void vistaCajero(){
    btnEmpleados.setVisible(false);
}
//Funcion para vista de Gerente y cajero
public void vistaGeneral(){
```

```
                UsuarioSistemaController usuarioController = new
UsuarioSistemaController();
        String rol =  usuarioController.verificarRol(usuario);
        String nombre = usuarioController.obtenerNombreCompleto(usuario);
        usuario = txtUsuario.getText();
        txtNombre2.setText("BIENVENIDO "+nombre);
        txtRol2.setText("ROL: "+rol);

    }
    //Damos valores a los Combos
    private void iniciarElementos(){
    DefaultListCellRenderer listRenderer = new DefaultListCellRenderer();
        listRenderer.setHorizontalAlignment(DefaultListCellRenderer.CENTER);
        cboRolEmpleado.setRenderer(listRenderer);
        this.cboRolEmpleado.addItem("Cajero");
        this.cboRolEmpleado.addItem("Gerente");

        listRenderer.setHorizontalAlignment(DefaultListCellRenderer.CENTER);
        cboTipoCuenta.setRenderer(listRenderer);
        this.cboTipoCuenta.addItem("Corriente");
        this.cboTipoCuenta.addItem("Ahorro");

    }
    //Funcion para mostrar los Empleados en la Tabla
    private void mostrarEmpleados() {
        DefaultTableModel model = (DefaultTableModel) tableEmpleados.getModel();
        model.setRowCount(0); // Elimina todas las filas actuales de la tabla

        EmpleadoController empleadoController = new EmpleadoController();
                UsuarioSistemaController usuarioController = new
UsuarioSistemaController();

                                    List<Empleado> empleados =
empleadoController.obtenerTodosLosEmpleados();
                            List<UsuarioSistema> usuariosSistema =
usuarioController.obtenerTodosLosUsuarios();

        Map<Integer, String> usuarioMap = new HashMap<>();
        for (UsuarioSistema usuario : usuariosSistema) {
            usuarioMap.put(usuario.getEmpleadoID(), usuario.getNombreUsuario());
        }
        for (Empleado empleado : empleados) {
                                        String nombreUsuario =
usuarioMap.getOrDefault(empleado.getEmpleadoID(), "No asignado");
```

```
            model.addRow(new Object[]{
                empleado.getNombre(),
                empleado.getApellido(),
                empleado.getDni(),
                empleado.getCargo(),
                empleado.getFechaContratacion(),
                empleado.getSalario(),
                empleado.getNumeroCelular(),
                nombreUsuario,
            });
        }
    }
    //Funcion para mostrar los Clientes en la tabla
    private void mostrarClientes(){
        DefaultTableModel model = (DefaultTableModel) tablaClientes.getModel();
        model.setRowCount(0);

        ClienteController clienteController = new ClienteController();
                    CuentaBancariaController cuentaBancariaController = new
CuentaBancariaController();

        List<Cliente> clientes = clienteController.obtenerTodosLosClientes();
                                            List<CuentaBancaria> cuentas =
cuentaBancariaController.obtenerTodasLasCuentas();


        Map<Integer, CuentaBancaria> cuentaMap = new HashMap<>();
        for (CuentaBancaria cuenta : cuentas) {
        cuentaMap.put(cuenta.getClienteID(), cuenta);
        }

        // Iterar sobre los clientes y buscar su cuenta bancaria en el Map
        for (Cliente cliente : clientes) {

            CuentaBancaria cuenta = cuentaMap.getOrDefault(cliente.getClienteID(),
null);
             String numeroCuenta = (cuenta != null) ? cuenta.getNumeroCuenta() :
"No asignada";
              String tipoCuenta = (cuenta != null) ? cuenta.getTipoCuenta() : "No
asignada";
            double saldo = (cuenta != null) ? cuenta.getSaldo() : 0.0;
```

```
                // Agregar la fila a la tabla con la información del cliente y la cuenta bancaria
                model.addRow(new Object[]{
                    cliente.getNombre(),
                    cliente.getApellido(),
                    cliente.getDni(),
                    cliente.getDireccion(),
                    cliente.getTelefono(),
                    cliente.getEmail(),
                    cliente.getFechaRegistro(),
                    numeroCuenta,
                    tipoCuenta,
                    saldo
                });
            }
        }
        //Funcion para mostrar los prestamos
        public void mostrarPrestamos(){
            PrestamosController prestamoController = new PrestamosController();

           DefaultTableModel model = (DefaultTableModel) tablePrestamo.getModel();
            model.setRowCount(0);

                                                    List<Prestamos> prestamosList = prestamoController.obtenerTodosLosPrestamos();
            List<Cliente> clientesList = prestamoController.obtenerTodosLosClientes();

            for (Prestamos prestamo : prestamosList) {
                // Encontrar el cliente correspondiente por el número de cuenta
                Cliente cliente = clientesList.stream()
                                                        .filter(c -> c.getClienteID() == prestamoController.obtenerClienteIDPorNumeroCuenta(prestamo.getNumeroCuenta()))
                        .findFirst()
                        .orElse(null);

                // Concatenar nombre y apellido del cliente
                    String nombreCompleto = (cliente != null) ? cliente.getNombre() + " " + cliente.getApellido() : "Desconocido";

                // Agregar la fila a la tabla con la información del préstamo
                model.addRow(new Object[]{
                    nombreCompleto,  // Mostrar nombre completo
                    prestamo.getMontoPrestamo(),
```

```
                prestamo.getInteres(),
                prestamo.getPlazo(),
                prestamo.getFechaDesembolso(),
                prestamo.getEstado(),
                prestamo.getPrestamoID() // PrestamoID si es necesario
            });
        }
    }
    //Convertir salario a double
    private double convertirSalario(String InsertSalario){
        double salario = 0;
        try {
            salario = Double.parseDouble(InsertSalario);
        } catch (NumberFormatException e) {
            JOptionPane.showMessageDialog(null, "El salario debe ser un número
válido.", "Error", JOptionPane.ERROR_MESSAGE);
        }
        return salario;
    }
private void formWindowOpened(java.awt.event.WindowEvent evt) {
    SistemaBancarioLogin.setVisible(true);
    formularioLogin();
    this.setVisible(false);
  }
  private void btnIngresar1ActionPerformed(java.awt.event.ActionEvent evt) {
    usuario = txtUsuario.getText();
    String contraseña = new String(txtContraseña.getPassword());
                        UsuarioSistemaController usuarioController = new
UsuarioSistemaController();
                                boolean credencialesValidas =
usuarioController.verificarCredenciales(usuario, contraseña);
    if (credencialesValidas) {
            JOptionPane.showMessageDialog(this, "Ingreso correcto", "Éxito",
JOptionPane.INFORMATION_MESSAGE);
        SistemaBancarioLogin.setVisible(false);
        this.setVisible(true);
        vistaGeneral();
        String rol =  usuarioController.verificarRol(usuario);
        if (rol.equals("Gerente")) {
            vistaGerente();
        }else if (rol.equals("Cajero")) {
            vistaCajero();
        }
    } else {
```

```
            JOptionPane.showMessageDialog(this, "Usuario o contraseña incorrectos",
"Error", JOptionPane.ERROR_MESSAGE);
        }
    }
    private void btnEmpleadosActionPerformed(java.awt.event.ActionEvent evt) {
        panelDatos.removeAll();
        panelEmpleados.setSize(870,557);
        panelEmpleados.setLocation(0, 0);
        panelDatos.add(panelEmpleados, BorderLayout.CENTER);
        panelDatos.revalidate();
        panelDatos.repaint();
        panelEmpleados.setVisible(true);
        btnAplicarCambiosEmpleados.setVisible(false);
    }
    private void btnClientesActionPerformed(java.awt.event.ActionEvent evt) {
        panelDatos.removeAll();
        panelClientes.setSize(870,557);
        panelClientes.setLocation(0, 0);
        panelDatos.add(panelClientes, BorderLayout.CENTER);
        panelDatos.revalidate();
        panelDatos.repaint();
        panelClientes.setVisible(true);
        mostrarClientes();
        btnAplicarCambiosCliente.setVisible(true);
    }
    private void btnTransaccionesActionPerformed(java.awt.event.ActionEvent evt) {
        panelDatos.removeAll();
        panelTransacciones.setSize(870,557);
        panelTransacciones.setLocation(0, 0);
        panelDatos.add(panelTransacciones, BorderLayout.CENTER);
        panelDatos.revalidate();
        panelDatos.repaint();
        panelTransacciones.setVisible(true);
    }
    private void btnPrestamosActionPerformed(java.awt.event.ActionEvent evt) {
        panelDatos.removeAll();
        panelPrestamo.setSize(870,557);
        panelPrestamo.setLocation(0, 0);
        panelDatos.add(panelPrestamo, BorderLayout.CENTER);
        panelDatos.revalidate();
        panelDatos.repaint();
        panelPrestamo.setVisible(true);
        mostrarPrestamos();
    }
```

```
    private void btnNuevoEmpleadoActionPerformed(java.awt.event.ActionEvent
evt) {
        limpiarEmpleado();
        txtFechaInicioEmpleado.setEditable(true);
        txtFechaInicioEmpleado.setFocusable(true);
        txtDniEmpleado.setEditable(true);
        txtDniEmpleado.setFocusable(true);
        btnAplicarCambiosEmpleados.setVisible(false);
    }
    private void btnCrearEmpleadoActionPerformed(java.awt.event.ActionEvent evt)
{
            String nombre = txtNombreEmpleado.getText().trim();
            String apellido = txtApellidoEmpleado.getText().trim();
            String dni = txtDniEmpleado.getText().trim();
            String cargo = cboRolEmpleado.getSelectedItem().toString();
            String fechaContratacionString = txtFechaInicioEmpleado.getText();
            String salarioString = txtSalarioEmpleado.getText().trim();
            String numeroCelular = txtCelularEmpleado.getText().trim();
            String nombreUsuario = txtUsuarioEmpleado.getText().trim();
            String contraseña = txtContraseñaEmpleado.getText().trim();
            String rol = cboRolEmpleado.getSelectedItem().toString();

             if (nombre.isEmpty() || apellido.isEmpty() || dni.isEmpty() || cargo.isEmpty()
||
              fechaContratacionString.isEmpty() || salarioString.isEmpty() ||
                                  numeroCelular.isEmpty()  ||  nombreUsuario.isEmpty()  ||
contraseña.isEmpty()) {
                 JOptionPane.showMessageDialog(null, "Todos los campos deben estar
completos.", "Error", JOptionPane.ERROR_MESSAGE);
            }else{
                double salario = Double.parseDouble(salarioString);
                // Convertir la fecha de String a java.sql.Date
                Date fechaContratacion = null;
                      SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd"); //
Formato de entrada

                try {
                    java.util.Date utilDate = sdf.parse(fechaContratacionString); // Parsear
la fecha
                          fechaContratacion = new Date(utilDate.getTime()); // Convertir a
java.sql.Date
                } catch (ParseException e) {
                    e.printStackTrace();
                    // Manejar el error de formato de fecha aquí
```

```
                return; // Salir del método si hay un error
            }

            // Instancias de los controladores
            EmpleadoController controllerEmple = new EmpleadoController();
                                UsuarioSistemaController controllerUsuSistem = new
UsuarioSistemaController();

            // 1. Agregar empleado y obtener el EmpleadoID generado
                int empleadoID = controllerEmple.agregarEmpleado(nombre, apellido,
dni, cargo, fechaContratacion, salario, numeroCelular);


            if (empleadoID != 0) { // Verificar si el empleado fue creado con éxito
                                                        boolean usuarioCreado =
controllerUsuSistem.agregarUsuario(empleadoID, nombreUsuario, contraseña, rol,
fechaContratacion);
                if (usuarioCreado) {
                    limpiarEmpleado();
                    mostrarEmpleados();
                } else {
                    System.out.println("Error al crear el usuario del sistema.");
                }
            } else {
                System.out.println("Error al crear el empleado.");
            }
        }
    }
    private void btnEliminarEmpleadoActionPerformed(java.awt.event.ActionEvent
evt) {

        String dniInsertado = JOptionPane.showInputDialog(null, "Por favor, ingresa el
DNI del Empleado:");

        if (dniInsertado != null && !dniInsertado.trim().isEmpty()) {
            EmpleadoController empleadoController = new EmpleadoController();
                            UsuarioSistemaController usuarioSistemaController = new
UsuarioSistemaController();

            try {
                                                        int empleadoID =
empleadoController.obtenerEmpleadoIDPorDni(dniInsertado);
            if (empleadoID != -1) { // Verifica que se encontró un empleado
```

```
usuarioSistemaController.eliminarUsuarioPorEmpleadoID(empleadoID);
                empleadoController.eliminarEmpleadoPorDNI(dniInsertado);

                                JOptionPane.showMessageDialog(null, "Empleado y usuario
eliminados con éxito.");
                mostrarEmpleados();
            } else {
                JOptionPane.showMessageDialog(null, "No se encontró un empleado
con el DNI proporcionado.");
            }
        } catch (Exception e) {
            e.printStackTrace();
            JOptionPane.showMessageDialog(null, "Error al eliminar el empleado: " +
e.getMessage());
        }
    } else {
        JOptionPane.showMessageDialog(null, "Entrada de DNI cancelada.");
    }

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

    txtFechaInicioEmpleado.setEditable(false);
    txtFechaInicioEmpleado.setFocusable(false);
    txtDniEmpleado.setEditable(false);
    txtDniEmpleado.setFocusable(false);
    btnAplicarCambiosEmpleados.setVisible(true);

        dniInsertadoModificarEmpleado = JOptionPane.showInputDialog(null, "Por
favor, ingresa el DNI del Empleado:");
                            if (dniInsertadoModificarEmpleado != null &&
!dniInsertadoModificarEmpleado.trim().isEmpty()) {

        EmpleadoController empleadoController = new EmpleadoController();
                        UsuarioSistemaController usuarioSistemaController = new
UsuarioSistemaController();


                                                    Empleado empleado =
empleadoController.obtenerEmpleadoPorDniModificar(dniInsertadoModificarEmple
ado);
    if (empleado != null) {
        txtNombreEmpleado.setText(empleado.getNombre());
```

```
            txtApellidoEmpleado.setText(empleado.getApellido());

            if (empleado.getCargo().equals("Cajero")) {
                cboRolEmpleado.setSelectedIndex(0);
            }else if(empleado.getCargo().equals("Gerente")){
                cboRolEmpleado.setSelectedIndex(1);
            }

            txtSalarioEmpleado.setText(String.valueOf(empleado.getSalario()));
            txtCelularEmpleado.setText(empleado.getNumeroCelular());

                                                    UsuarioSistema usuarioSistema =
usuarioSistemaController.obtenerUsuarioPorDniModificar(dniInsertadoModificarE
mpleado);
            if (usuarioSistema != null) {
                txtUsuarioEmpleado.setText(usuarioSistema.getNombreUsuario());
                txtContraseñaEmpleado.setText(usuarioSistema.getContraseña());


            } else {
                        JOptionPane.showMessageDialog(null, "No se encontró el usuario
asociado a este empleado.");
            }
        } else {
              JOptionPane.showMessageDialog(null, "No se encontró ningún empleado
con el DNI ingresado.");
        }



        }else{
            JOptionPane.showMessageDialog(null, "Entrada de DNI cancelada.");
        }

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

            String nombre = txtNombreEmpleado.getText().trim();
            String apellido = txtApellidoEmpleado.getText().trim();
            String cargo = cboRolEmpleado.getSelectedItem().toString();
            String salarioString = txtSalarioEmpleado.getText().trim();
            String numeroCelular = txtCelularEmpleado.getText().trim();
            String nombreUsuario = txtUsuarioEmpleado.getText().trim();
```

```
        String contraseña = txtContraseñaEmpleado.getText().trim();

                if (nombre.isEmpty() || apellido.isEmpty() || salarioString.isEmpty() ||
numeroCelular.isEmpty() ||
            nombreUsuario.isEmpty() || contraseña.isEmpty()) {
                JOptionPane.showMessageDialog(null, "Por favor, complete todos los
campos.", "Error", JOptionPane.ERROR_MESSAGE);
            return;
        }
        double salario = convertirSalario(salarioString);

        EmpleadoController empleadoController = new EmpleadoController();
                                UsuarioSistemaController usuarioController = new
UsuarioSistemaController();

                                                    boolean empleadoActualizado =
empleadoController.actualizarEmpleado(dniInsertadoModificarEmpleado, nombre,
apellido, cargo, salario, numeroCelular);
        if (empleadoActualizado) {
                        JOptionPane.showMessageDialog(null, "Empleado actualizado
exitosamente.");
        } else {
                         JOptionPane.showMessageDialog(null, "Error al actualizar el
empleado.");
        }

                                                    boolean usuarioActualizado =
usuarioController.actualizarUsuario(dniInsertadoModificarEmpleado,
nombreUsuario, contraseña);
        if (usuarioActualizado) {
                          JOptionPane.showMessageDialog(null, "Usuario actualizado
exitosamente.");
        } else {
            JOptionPane.showMessageDialog(null, "Error al actualizar el usuario.");
        }

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

        String nombre = txtNombreCliente.getText().trim();
        String apellido = txtApellidoCliente.getText().trim();
        String dni = txtDniCliente.getText().trim();
        String direccion = txtDireccionCliente.getText().trim();
        String numeroCelular = txtCelularCliente.getText().trim();
```

```
        String email = txtEmailCliente.getText().trim();
        String fechaRegistroString = txtFechaRegistroCliente.getText().trim();
        String numCuenta = txtNumeroCuentaCliente.getText().trim();
        String TipoCuenta = cboTipoCuenta.getSelectedItem().toString();
        String SaldoString = txtSaldoCliente.getText().trim();

                    if (nombre.isEmpty() || apellido.isEmpty() || dni.isEmpty() ||
direccion.isEmpty() || numeroCelular.isEmpty()
            || email.isEmpty() || numCuenta.isEmpty() || SaldoString.isEmpty()) {

                JOptionPane.showMessageDialog(null, "Todos los campos deben estar
completos.", "Error", JOptionPane.ERROR_MESSAGE);
            }else{
                double saldo = convertirSalario(SaldoString);

                Date fechaRegistro = null;
                    SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd"); //
Formato de entrada
                try {
                    java.util.Date utilDate = sdf.parse(fechaRegistroString);
                    fechaRegistro = new Date(utilDate.getTime());
                } catch (ParseException e) {
                    e.printStackTrace();
                    return;
                }
                ClienteController clienteController = new ClienteController();
                        CuentaBancariaController cuentaBancariaController = new
CuentaBancariaController();

                int clienteID = clienteController.agregarCliente(nombre, apellido, dni,
direccion, numeroCelular, email, fechaRegistro);

                if (clienteID != 0) {
                                                boolean cuentaBancariaCreada =
cuentaBancariaController.agregarCuentaBancaria(clienteID, numCuenta,
TipoCuenta, saldo, fechaRegistro);
                    if (cuentaBancariaCreada) {
                        JOptionPane.showMessageDialog(null, "Cliente y Cuenta Bancaria
creado con exito.");
                        limpiarCliente();
                    }else{
                        System.out.println("Error al crear el cuenta Bancaria.");
                    }
                }else {
```

```
                System.out.println("Error al crear el Cliente.");
            }


        }

    }
    private void btnEliminarClienteActionPerformed(java.awt.event.ActionEvent evt)
{
        String dniInsertado = JOptionPane.showInputDialog(null, "Por favor, ingresa
el DNI del Cliente:");

        if (dniInsertado != null && !dniInsertado.trim().isEmpty()) {
            ClienteController clienteController = new ClienteController();
                            CuentaBancariaController  cuentaBancariaController  =  new
CuentaBancariaController();
            try {
                int ClienteID = clienteController.obtenerClienteIDPorDni(dniInsertado);
                if (ClienteID != -1) { // Verifica que se encontró un empleado

cuentaBancariaController.eliminarCuentaBancariaPorEmpleadoID(ClienteID);
                clienteController.eliminarClientePorDNI(dniInsertado);
                      JOptionPane.showMessageDialog(null, "Cliente y cuenta bancaria
eliminados con éxito.");
                mostrarClientes();
                } else {
                 JOptionPane.showMessageDialog(null, "No se encontró un Cliente con
el DNI proporcionado.");
                }
            } catch (Exception e) {
                e.printStackTrace();
                JOptionPane.showMessageDialog(null, "Error al eliminar el empleado: " +
e.getMessage());
            }
        } else {
            JOptionPane.showMessageDialog(null, "Entrada de DNI cancelada.");
        }
    }
    private void btnNuevoClienteActionPerformed(java.awt.event.ActionEvent evt) {
        limpiarCliente();
        txtSaldoCliente.setEnabled(true);
        txtSaldoCliente.setEditable(true);
        txtNumeroCuentaCliente.setEnabled(true);
        txtNumeroCuentaCliente.setEditable(true);
```

```
        txtDniCliente.setEditable(true);
        txtDniCliente.setEnabled(true);
    }
        private void btnModificarClienteActionPerformed(java.awt.event.ActionEvent
evt) {
        txtSaldoCliente.setEnabled(false);
        txtSaldoCliente.setEditable(false);
        txtNumeroCuentaCliente.setEnabled(false);
        txtNumeroCuentaCliente.setEditable(false);
        txtDniCliente.setEditable(false);
        txtDniCliente.setEnabled(false);

        dniInsertadoModificarCliente = JOptionPane.showInputDialog(null, "Por favor,
ingresa el DNI del Cliente:");
                                    if (dniInsertadoModificarCliente != null &&
!dniInsertadoModificarCliente.trim().isEmpty()) {

            ClienteController clienteController = new ClienteController();
                        CuentaBancariaController cuentaBancariaController = new
CuentaBancariaController();


                                                        Cliente cliente =
clienteController.obtenerClientePorDniModificar(dniInsertadoModificarCliente);
          if (cliente != null) {
            txtNombreCliente.setText(cliente.getNombre());
            txtApellidoCliente.setText(cliente.getApellido());
            txtDireccionCliente.setText(cliente.getDireccion());
            txtCelularCliente.setText(cliente.getTelefono());
            txtEmailCliente.setText(cliente.getEmail());
            txtFechaRegistroCliente.setText(cliente.getFechaRegistro());

                                                    CuentaBancaria cuentabancaria =
cuentaBancariaController.obtenerCuentaBancariaPorDniModificar(dniInsertadoMo
dificarCliente);
            if (cuentabancaria != null) {
                if (cuentabancaria.getTipoCuenta().equals("Corriente")) {
                  cboTipoCuenta.setSelectedIndex(0);
                }else {
                  cboTipoCuenta.setSelectedIndex(1);
                }
                btnAplicarCambiosCliente.setVisible(true);
            } else {
```

```
                    JOptionPane.showMessageDialog(null, "No se encontró el usuario asociado a este empleado.");
                }
            } else {
                    JOptionPane.showMessageDialog(null, "No se encontró ningún empleado con el DNI ingresado.");
            }
        }else{
            JOptionPane.showMessageDialog(null, "Entrada de DNI cancelada.");
        }
private void btnAplicarCambiosClienteActionPerformed(java.awt.event.ActionEvent evt) {
        String nombre = txtNombreCliente.getText().trim();
        String apellido = txtApellidoCliente.getText().trim();
        String direccion = txtDireccionCliente.getText().trim();
        String celular = txtCelularCliente.getText().trim();
        String email = txtEmailCliente.getText().trim();
        String tipoCuenta = cboTipoCuenta.getSelectedItem().toString();

        // Validar que todos los campos estén completos
                if (nombre.isEmpty() || apellido.isEmpty() || direccion.isEmpty() || celular.isEmpty() ||
            email.isEmpty()) {
                JOptionPane.showMessageDialog(null, "Por favor, complete todos los campos.", "Error", JOptionPane.ERROR_MESSAGE);
            return;
        }

        ClienteController clienteController = new ClienteController();
                    CuentaBancariaController cuentaBancariaController = new CuentaBancariaController();

        // Actualizar información del cliente
                                            boolean clienteActualizado = clienteController.actualizarCliente(dniInsertadoModificarCliente, nombre, apellido, direccion, celular, email);
        if (clienteActualizado) {
                    JOptionPane.showMessageDialog(null, "Cliente actualizado exitosamente.");
        } else {
            JOptionPane.showMessageDialog(null, "Error al actualizar el cliente.");
        }
```

```
                                                    boolean cuentaActualizada =
cuentaBancariaController.actualizarCuentaBancaria(dniInsertadoModificarCliente,
tipoCuenta);
        if (cuentaActualizada) {
                JOptionPane.showMessageDialog(null, "Cuenta bancaria actualizada
exitosamente.");
            limpiarCliente();
        } else {
                JOptionPane.showMessageDialog(null, "Error al actualizar la cuenta
bancaria.");
        }
        limpiarCliente();
        mostrarClientes();
    }
    private void btnRetiroActionPerformed(java.awt.event.ActionEvent evt) {
        String numeroCuenta = txtNumeroCuentaTransaccion.getText().trim();
        String montoString = txtMontoTransaccion.getText().trim();
        String tipoTransaccion = "retiro"; // Tipo de transacción

                            TransaccionesController transaccionesController = new
TransaccionesController();

        // Validar campos vacíos
        if (numeroCuenta.isEmpty() || montoString.isEmpty()) {
                JOptionPane.showMessageDialog(null, "Por favor, complete todos los
campos.", "Error", JOptionPane.ERROR_MESSAGE);
            return;
        }

        // Validar que el monto sea un número válido
        double monto;
        try {
            monto = Double.parseDouble(montoString);
            if (monto <= 0) {
                JOptionPane.showMessageDialog(null, "El monto debe ser mayor que
cero.", "Error", JOptionPane.ERROR_MESSAGE);
                return;
            }
        } catch (NumberFormatException e) {
                JOptionPane.showMessageDialog(null, "El monto debe ser un número
válido.", "Error", JOptionPane.ERROR_MESSAGE);
            return;
        }
```

```
        // Verificar la existencia de la cuenta
                                                        Integer cuentaID =
transaccionesController.obtenerCuentaIdPorNumeroCuenta(numeroCuenta);
        if (cuentaID == null) {
            JOptionPane.showMessageDialog(null, "El número de cuenta no existe.",
"Error", JOptionPane.ERROR_MESSAGE);
            return;
        }

        // Obtener ID del empleado que realiza la transacción
                                                        int empleadoID =
transaccionesController.obtenerEmpleadoIdPorUsuario(usuario);

        // Restar monto de la cuenta
        if (transaccionesController.restarMontoDeCuenta(cuentaID, monto)) {
            // Insertar la transacción
                        if (transaccionesController.insertarTransaccionRetiro(cuentaID,
empleadoID, tipoTransaccion, monto, numeroCuenta)) {
            JOptionPane.showMessageDialog(null, "Retiro realizado exitosamente.");

            // Limpiar los campos de entrada
            txtNumeroCuentaTransaccion.setText("");
            txtMontoTransaccion.setText("");

            // Obtener información adicional para el voucher
                                                Transacciones ultimaTransaccion =
transaccionesController.obtenerUltimaOperacionConFecha(); // Método que
devuelve un objeto Transacciones
                                                        String nombreCliente =
transaccionesController.obtenerNombrePorNumeroCuenta(numeroCuenta);

            // Completar los campos del voucher
            txtVaucherNombre.setText("Nombre: " + nombreCliente);
                        txtVaucherNumeroCuenta.setText("Número de cuenta: " +
numeroCuenta);
            txtVaucherMonto.setText("Monto Retirado: " + monto);
                            txtVaucherOperacion.setText("Nº de Operación: " +
ultimaTransaccion.getTransaccionID());
                                                txtVaucherFecha.setText("Fecha: " +
ultimaTransaccion.getFechaTransaccion()); // Asegúrate de tener el método
getFechaTransaccion() en tu clase Transacciones
            txtVaucherEmpleadoID.setText("Código Empleado: " + empleadoID);
          } else {
```

```
                    JOptionPane.showMessageDialog(null, "Error al registrar la transacción.");
                }
            } else {
                JOptionPane.showMessageDialog(null, "Error al restar el monto de la cuenta.");
            }
        }
        private void btnDepositoActionPerformed(java.awt.event.ActionEvent evt) {
            String numeroCuenta = txtNumeroCuentaTransaccion.getText().trim();
            String montoString = txtMontoTransaccion.getText().trim();
            String tipoTransaccion = "deposito"; // Tipo de transacción

                    TransaccionesController transaccionesController = new TransaccionesController();

            // Validar campos vacíos
            if (numeroCuenta.isEmpty() || montoString.isEmpty()) {
                JOptionPane.showMessageDialog(null, "Por favor, complete todos los campos.", "Error", JOptionPane.ERROR_MESSAGE);
                return;
            }

            // Validar que el monto sea un número válido
            double monto;
            try {
                monto = Double.parseDouble(montoString);
                if (monto <= 0) {
                    JOptionPane.showMessageDialog(null, "El monto debe ser mayor que cero.", "Error", JOptionPane.ERROR_MESSAGE);
                    return;
                }
            } catch (NumberFormatException e) {
                JOptionPane.showMessageDialog(null, "El monto debe ser un número válido.", "Error", JOptionPane.ERROR_MESSAGE);
                return;
            }

            // Verificar la existencia de la cuenta
                                                Integer cuentaID = transaccionesController.obtenerCuentaIdPorNumeroCuenta(numeroCuenta);
            if (cuentaID == null) {
                JOptionPane.showMessageDialog(null, "El número de cuenta no existe.", "Error", JOptionPane.ERROR_MESSAGE);
```

```
            return;
        }

                                                                int empleadoID =
transaccionesController.obtenerEmpleadoIdPorUsuario(usuario);

        // Sumar monto a la cuenta
        if (transaccionesController.SumarMontoDeCuenta(cuentaID, monto)) {
            // Insertar la transacción
                        if (transaccionesController.insertarTransaccionDeposito(cuentaID,
empleadoID, tipoTransaccion, monto, numeroCuenta)) {
                            JOptionPane.showMessageDialog(null, "Depósito realizado
exitosamente.");

                // Limpiar los campos de entrada
                txtNumeroCuentaTransaccion.setText("");
                txtMontoTransaccion.setText("");

                // Obtener información adicional para el voucher
                                                    Transacciones ultimaTransaccion =
transaccionesController.obtenerUltimaOperacionConFecha(); // Método que
devuelve un objeto Transacciones
                                                            String nombreCliente =
transaccionesController.obtenerNombrePorNumeroCuenta(numeroCuenta);

                // Completar los campos del voucher
                txtVaucherNombre.setText("Nombre: " + nombreCliente);
                            txtVaucherNumeroCuenta.setText("Número de cuenta: " +
numeroCuenta);
                txtVaucherMonto.setText("Monto Depositado: " + monto);
                                    txtVaucherOperacion.setText("Nº de Operación: " +
ultimaTransaccion.getTransaccionID());
                                                    txtVaucherFecha.setText("Fecha: " +
ultimaTransaccion.getFechaTransaccion()); // Asegúrate de tener el método
getFechaTransaccion() en tu clase Transacciones
                txtVaucherEmpleadoID.setText("Código Empleado: " + empleadoID);
            } else {
                            JOptionPane.showMessageDialog(null, "Error al registrar la
transacción.");
            }
        } else {
                    JOptionPane.showMessageDialog(null, "Error al sumar el monto a la
cuenta.");
        }
```

```
    }
    private void btnRealizarPrestamoActionPerformed(java.awt.event.ActionEvent
evt) {
        String numeroCuenta = txtNumeroCuentaPrestamo.getText().trim();
        String montoString = txtMontoPrestamo.getText().trim();
        String plazoString = txtPlazoMesesPrestamo.getText().trim();
           String fechaString = txtFechaPrestamo.getText().trim(); // Asegúrate de que
sea un formato válido

        PrestamosController prestamoController= new PrestamosController();

        if (numeroCuenta.isEmpty() || montoString.isEmpty() || plazoString.isEmpty() ||
fechaString.isEmpty()) {
                JOptionPane.showMessageDialog(null, "Por favor, complete todos los
campos.", "Error", JOptionPane.ERROR_MESSAGE);
            return;
        }

                                                    Integer clienteID =
prestamoController.obtenerClienteIDPorNumeroCuenta(numeroCuenta);
        if (clienteID == null) {
              JOptionPane.showMessageDialog(null, "El número de cuenta no existe.",
"Error", JOptionPane.ERROR_MESSAGE);
            return;
        }

        double montoPrestamo;
        try {
            montoPrestamo = Double.parseDouble(montoString);
            if (montoPrestamo <= 0) {
                  JOptionPane.showMessageDialog(null, "El monto debe ser mayor que
cero.", "Error", JOptionPane.ERROR_MESSAGE);
                return;
            }
        } catch (NumberFormatException e) {
                JOptionPane.showMessageDialog(null, "El monto debe ser un número
válido.", "Error", JOptionPane.ERROR_MESSAGE);
            return;
        }

        int plazo;
        try {
            plazo = Integer.parseInt(plazoString);
            if (plazo <= 0) {
```

```
                JOptionPane.showMessageDialog(null, "El plazo debe ser mayor que cero.", "Error", JOptionPane.ERROR_MESSAGE);
                return;
            }
        } catch (NumberFormatException e) {
                JOptionPane.showMessageDialog(null, "El plazo debe ser un número válido.", "Error", JOptionPane.ERROR_MESSAGE);
            return;
        }


        int empleadoID = prestamoController.obtenerEmpleadoIdPorUsuario(usuario);

        double interes = prestamoController.calcularInteres(plazo, montoPrestamo);

        java.sql.Date fechaDesembolso = java.sql.Date.valueOf(fechaString);

        // Insertar el préstamo
            if (prestamoController.insertarPrestamo(clienteID, montoPrestamo, interes, plazo, fechaDesembolso, empleadoID, numeroCuenta)) {
                            JOptionPane.showMessageDialog(null, "Préstamo guardado exitosamente.");
            mostrarPrestamos();
        } else {
            JOptionPane.showMessageDialog(null, "Error al guardar el préstamo.");
        }
    }

    private void btnNuevoPrestamoActionPerformed(java.awt.event.ActionEvent evt) {
        limpiarPrestamo();
    }

        private void DesactivarPrestamoActionPerformed(java.awt.event.ActionEvent evt) {
                String numeroCuenta = JOptionPane.showInputDialog(null, "Ingrese el número de cuenta para desactivar el préstamo:");
        PrestamosController prestamoController= new PrestamosController();
        // Validar que se ingresó un número de cuenta
        if (numeroCuenta == null || numeroCuenta.trim().isEmpty()) {
                JOptionPane.showMessageDialog(null, "Por favor, ingrese un número de cuenta.", "Error", JOptionPane.ERROR_MESSAGE);
            return;
        }
```

```
        // Verificar si el préstamo existe para el número de cuenta
        if (!prestamoController.existePrestamoPorNumeroCuenta(numeroCuenta)) {
                JOptionPane.showMessageDialog(null, "No hay préstamos asociados a este número de cuenta.", "Error", JOptionPane.ERROR_MESSAGE);
            return;
        }

        // Actualizar el estado del préstamo a "No activo"
        if (prestamoController.desactivarPrestamo(numeroCuenta)) {
             JOptionPane.showMessageDialog(null, "El préstamo ha sido desactivado exitosamente.");
            mostrarPrestamos();
        } else {
            JOptionPane.showMessageDialog(null, "Error al desactivar el préstamo.");
        }
    }

    private void txtUsuarioActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
    }

    private void txtContraseñaActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
    }

    private void txtDniEmpleadoActionPerformed(java.awt.event.ActionEvent evt) {

    }

        private void txtCelularEmpleadoActionPerformed(java.awt.event.ActionEvent evt) {

    }

        private void txtSalarioEmpleadoActionPerformed(java.awt.event.ActionEvent evt) {

    }

    private void txtDniEmpleadoKeyTyped(java.awt.event.KeyEvent evt) {
   char c = evt.getKeyChar();
    String text = txtDniEmpleado.getText();
```

```
    // Permitir solo dígitos
    if (!Character.isDigit(c)) {
        evt.consume(); // Evitar caracteres que no sean dígitos
    }

    // Limitar la longitud total a 8 caracteres
    if (text.length() >= 8) {
        evt.consume(); // Evitar que se exceda la longitud total
    }
    }

    private void txtCelularEmpleadoKeyTyped(java.awt.event.KeyEvent evt) {
  char c = evt.getKeyChar();
    String text = txtCelularEmpleado.getText();

    // Permitir solo dígitos
    if (!Character.isDigit(c)) {
        evt.consume(); // Evitar caracteres que no sean dígitos
    }

    // Limitar la longitud total a 9 caracteres
    if (text.length() >= 8) {
        evt.consume(); // Evitar que se exceda la longitud total
    }
// TODO add your handling code here:
    }

    private void txtDniClienteActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
    }

    private void txtDniClienteKeyTyped(java.awt.event.KeyEvent evt) {

    char c = evt.getKeyChar();
    // Verificar si el carácter no es un número o si el campo ya tiene 8 números
    if (!Character.isDigit(c) ||  txtDniCliente.getText().length() >= 8) {
          evt.consume(); // Evitar que se escriban caracteres no numéricos o más de 8
dígitos
    }
            // TODO add your handling code here:
    }

    private void txtCelularClienteActionPerformed(java.awt.event.ActionEvent evt) {
```

```
    }

    private void txtCelularClienteKeyTyped(java.awt.event.KeyEvent evt) {
   char c = evt.getKeyChar();
    String text = txtCelularCliente.getText();

    // Permitir solo dígitos (0-9)
    if (!Character.isDigit(c)) {
        evt.consume(); // Evitar caracteres que no sean dígitos
    }

     // Limitar la longitud total a 10 caracteres (puedes ajustar este número según tus
necesidades)
    if (text.length() >= 8) {
        evt.consume(); // Evitar que se exceda la longitud total
    }    // TODO add your handling code here:
    }

                                                          private                  void
txtNumeroCuentaClienteActionPerformed(java.awt.event.ActionEvent evt) {

    }

    private void txtNumeroCuentaClienteKeyTyped(java.awt.event.KeyEvent evt) {
        char c = evt.getKeyChar();
    String text = txtNumeroCuentaCliente.getText();

    // Permitir solo dígitos (0-9)
    if (!Character.isDigit(c)) {
        evt.consume(); // Evitar caracteres que no sean dígitos
    }

    // Limitar la longitud total a 20 caracteres
    if (text.length() >= 20) {
        evt.consume(); // Evitar que se exceda la longitud total
    }
    }

    private void txtSaldoClienteActionPerformed(java.awt.event.ActionEvent evt) {

    }

    private void txtSaldoClienteKeyTyped(java.awt.event.KeyEvent evt) {
   char c = evt.getKeyChar();
```

```
    String text = txtSaldoCliente.getText();

    // Permitir solo dígitos y el punto decimal
    if (!Character.isDigit(c) && c != '.') {
        evt.consume(); // Evitar caracteres que no sean dígitos o el punto
    }

    // Limitar la longitud total a 20 caracteres
    if (text.length() >= 20) {
        evt.consume(); // Evitar que se exceda la longitud total
    }

    // Evitar más de un punto decimal
    if (c == '.') {
        if (text.contains(".")) {
            evt.consume(); // Evitar más de un punto decimal
        }
    }// TODO add y
    }

                                                    private                 void
txtNumeroCuentaTransaccionActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
    }

        private  void  txtNumeroCuentaTransaccionKeyTyped(java.awt.event.KeyEvent
evt) {
    char c = evt.getKeyChar();
    String text = txtNumeroCuentaTransaccion.getText();

    // Permitir solo dígitos (0-9)
    if (!Character.isDigit(c)) {
        evt.consume(); // Evitar caracteres que no sean dígitos
    }

    // Limitar la longitud total a 20 caracteres
    if (text.length() >= 20) {
        evt.consume(); // Evitar que se exceda la longitud total
    }
    }

      private  void  txtMontoTransaccionActionPerformed(java.awt.event.ActionEvent
evt) {
        // TODO add your handling code here:
```

```
    }

    private void txtMontoTransaccionKeyTyped(java.awt.event.KeyEvent evt) {
        char c = evt.getKeyChar();
    String text = txtMontoTransaccion.getText();

    // Permitir solo dígitos y el punto decimal
    if (!Character.isDigit(c) && c != '.') {
        evt.consume(); // Evitar caracteres que no sean dígitos o el punto
    }

    // Limitar la longitud total a 20 caracteres
    if (text.length() >= 20) {
        evt.consume(); // Evitar que se exceda la longitud total
    }

    // Evitar más de un punto decimal
    if (c == '.') {
        if (text.contains(".")) {
            evt.consume(); // Evitar más de un punto decimal
        }
    }
    }

    private void txtNumeroCuentaPrestamoKeyTyped(java.awt.event.KeyEvent evt) {
        char c = evt.getKeyChar();
    String text = txtNumeroCuentaPrestamo.getText();

    // Permitir solo dígitos (0-9)
    if (!Character.isDigit(c)) {
         evt.consume(); // Evitar caracteres que no sean dígitos (incluyendo espacios y
caracteres especiales)
    }

    // Limitar la longitud total a 20 caracteres
    if (text.length() >= 20) {
        evt.consume(); // Evitar que se exceda la longitud total
    }
    }

    private void txtMontoPrestamoKeyTyped(java.awt.event.KeyEvent evt) {
    char c = evt.getKeyChar();
    String text = txtMontoPrestamo.getText();
```

```
    // Permitir solo dígitos y el punto
    if (!Character.isDigit(c) && c != '.') {
        evt.consume(); // Evitar caracteres que no sean dígitos o el punto
    }

    // Limitar la longitud total a 20 caracteres
    if (text.length() >= 20) {
        evt.consume(); // Evitar que se exceda la longitud total
    }

    // Evitar más de un punto decimal
    if (c == '.') {
        if (text.contains(".")) {
            evt.consume(); // Evitar más de un punto
        }
    }
    }

    private void jTextField25KeyTyped(java.awt.event.KeyEvent evt) {
     char c = evt.getKeyChar();
    String text = jTextField25.getText();

    // Permitir solo dígitos y limitar la longitud a 20 caracteres
    if (!Character.isDigit(c) || text.length() >= 20) {
         evt.consume(); // Evitar que se escriban caracteres no numéricos o más de 20
dígitos
    }
    }

    private void jTextField28KeyTyped(java.awt.event.KeyEvent evt) {
    char c = evt.getKeyChar();
    String text = jTextField26.getText();

    // Permitir solo dígitos y el punto
    if (!Character.isDigit(c) && c != '.') {
        evt.consume(); // Evitar caracteres que no sean dígitos o el punto
    }

    // Limitar la longitud total a 20 caracteres
    if (text.length() >= 20) {
        evt.consume(); // Evitar que se exceda la longitud total
    }

    // Evitar más de un punto decimal
```

```
    if (c == '.') {
        if (text.contains(".")) {
            evt.consume(); // Evitar más de un punto
        }
    }
    }

    private void txtUsuarioKeyTyped(java.awt.event.KeyEvent evt) {
         char c = evt.getKeyChar();
     // Verificar si el carácter es una letra o número, y que no exceda la longitud de 8 caracteres
    if (!Character.isLetterOrDigit(c) || txtUsuario.getText().length() >= 20) {
                evt.consume(); // Evitar que se escriban caracteres no alfabéticos ni numéricos, o más de 8 caracteres
}
    }

    private void txtContraseñaKeyTyped(java.awt.event.KeyEvent evt) {
char c = evt.getKeyChar();
    // Verificar si el carácter es una letra, número o si es un espacio, y que no exceda la longitud de 8 caracteres
     if (!Character.isLetterOrDigit(c) || c == ' ' || txtContraseña.getText().length() >= 20)
{
         evt.consume(); // Evitar que se escriban caracteres no alfabéticos, numéricos, espacios, o más de 8 caracteres
    }       // TODO add your handling code here:
    }

    private void txtNombreEmpleadoKeyTyped(java.awt.event.KeyEvent evt) {
  char c = evt.getKeyChar();
    String text = txtNombreEmpleado.getText();

    // Permitir solo letras y espacios
    if (!Character.isLetter(c) && c != ' ') {
        evt.consume(); // Evitar caracteres que no sean letras o espacios
    }

     // Limitar la longitud total a 50 caracteres (puedes ajustar este número según tus necesidades)
    if (text.length() >= 50) {
        evt.consume(); // Evitar que se exceda la longitud total

}
    }
```

```
    private void txtApellidoEmpleadoKeyTyped(java.awt.event.KeyEvent evt) {
   char c = evt.getKeyChar();
    String text = txtApellidoEmpleado.getText();

    // Permitir solo letras y espacios
    if (!Character.isLetter(c) && c != ' ') {
        evt.consume(); // Evitar caracteres que no sean letras o espacios
    }

     // Limitar la longitud total a 50 caracteres (puedes ajustar este número según tus
necesidades)
    if (text.length() >= 50) {
        evt.consume(); // Evitar que se exceda la longitud total
    }
    }

    private void txtFechaInicioEmpleadoKeyTyped(java.awt.event.KeyEvent evt) {
         char c = evt.getKeyChar();
    String text = txtFechaInicioEmpleado.getText();

    // Permitir solo dígitos y el guion
    if (!Character.isDigit(c) && c != '-') {
        evt.consume(); // Evitar caracteres que no sean dígitos o guiones
    }

    // Limitar la longitud total a 10 caracteres (YYYY-MM-DD)
    if (text.length() >= 10) {
        evt.consume(); // Evitar que se exceda la longitud total
    }

    // Controlar la posición del guion
    if (c == '-') {
        // Evitar guiones al inicio o consecutivos
        if (text.length() == 0 || text.charAt(text.length() - 1) == '-') {
            evt.consume(); // Evitar guiones al inicio o consecutivos
        } else {
            // Permitir solo 2 guiones en las posiciones adecuadas
            int dashCount = 0;
            for (int i = 0; i < text.length(); i++) {
                if (text.charAt(i) == '-') {
                    dashCount++;
                }
            }
```

```
            // Verificar si hay menos de 2 guiones
            if (dashCount >= 2) {
                evt.consume(); // Evitar más de 2 guiones
            }
        }
    }
    }

    private void txtSalarioEmpleadoKeyTyped(java.awt.event.KeyEvent evt) {
   char c = evt.getKeyChar();
    String text = txtSalarioEmpleado.getText();

    // Permitir solo dígitos y el punto decimal
    if (!Character.isDigit(c) && c != '.') {
        evt.consume(); // Evitar caracteres que no sean dígitos o el punto
    }

    // Limitar la longitud total a 20 caracteres (ajusta según sea necesario)
    if (text.length() >= 20) {
        evt.consume(); // Evitar que se exceda la longitud total
    }

    // Evitar más de un punto decimal
    if (c == '.') {
        if (text.contains(".")) {
            evt.consume(); // Evitar más de un punto decimal
        }
    }

    // Evitar que el punto esté al inicio
    if (text.isEmpty() && c == '.') {
        evt.consume(); // Evitar que el punto esté al inicio
    }
    }

        private void txtNombreEmpleadoActionPerformed(java.awt.event.ActionEvent
evt) {
        // TODO add your handling code here:
    }

    private void txtUsuarioEmpleadoKeyTyped(java.awt.event.KeyEvent evt) {
   char c = evt.getKeyChar();
    String text = txtUsuarioEmpleado.getText();
```

```
    // Permitir solo letras y números
    if (!Character.isLetterOrDigit(c)) {
        evt.consume(); // Evitar caracteres que no sean letras o números
    }

    // Limitar la longitud total a 20 caracteres (ajusta según sea necesario)
    if (text.length() >= 20) {
        evt.consume(); // Evitar que se exceda la longitud total
    }
    }

    private void txtContraseñaEmpleadoKeyTyped(java.awt.event.KeyEvent evt) {
  char c = evt.getKeyChar();

    // Evitar espacios
    if (c == ' ') {
        evt.consume(); // Bloquear espacios
    }

    // Limitar la longitud total a 20 caracteres (ajusta según sea necesario)
    if (txtContraseñaEmpleado.getText().length() >= 20) {
        evt.consume(); // Evitar que se exceda la longitud total
    }
    }

    private void txtNombreClienteActionPerformed(java.awt.event.ActionEvent evt) {

    }

    private void txtNombreClienteKeyTyped(java.awt.event.KeyEvent evt) {
   char c = evt.getKeyChar();
    String text = txtNombreCliente.getText();

    // Permitir solo letras y espacios
    if (!Character.isLetter(c) && c != ' ') {
        evt.consume(); // Evitar caracteres que no sean letras o espacios
    }

     // Limitar la longitud total a 50 caracteres (puedes ajustar este número según tus
necesidades)
    if (text.length() >= 50) {
        evt.consume(); // Evitar que se exceda la longitud total
    }
    }
```

```
 private void jTextField26KeyTyped(java.awt.event.KeyEvent evt) {
char c = evt.getKeyChar();
 String text = jTextField26.getText();

 // Permitir solo dígitos y el punto
 if (!Character.isDigit(c) && c != '.') {
    evt.consume(); // Evitar caracteres que no sean dígitos o el punto
 }

 // Limitar la longitud total a 20 caracteres
 if (text.length() >= 20) {
    evt.consume(); // Evitar que se exceda la longitud total
 }

 // Evitar más de un punto decimal
 if (c == '.') {
    if (text.contains(".")) {
       evt.consume(); // Evitar más de un punto
    }
 }
 }

 private void jTextField27KeyTyped(java.awt.event.KeyEvent evt) {
  char c = evt.getKeyChar();
 String text = jTextField27.getText();

 // Permitir solo dígitos y el guion
 if (!Character.isDigit(c) && c != '-') {
    evt.consume(); // Evitar caracteres que no sean dígitos o guiones
 }

 // Limitar la longitud total a 10 caracteres (YYYY-MM-DD)
 if (text.length() >= 10) {
    evt.consume(); // Evitar que se exceda la longitud
 }

 // Evitar más de un guion consecutivo o en posiciones incorrectas
 if (c == '-') {
    // Evitar guiones al inicio o consecutivos
    if (text.length() == 0 || text.charAt(text.length() - 1) == '-') {
       evt.consume(); // Evitar guiones al inicio o consecutivos
    } else {
       // Permitir solo 2 guiones en las posiciones adecuadas
```

```
            int dashCount = 0;
            for (int i = 0; i < text.length(); i++) {
                if (text.charAt(i) == '-') {
                    dashCount++;
                }
            }
            // Evitar más de 2 guiones
            if (dashCount >= 2) {
                evt.consume(); // Evitar más de 2 guiones
            }
        }
    }
    }

    private void txtPlazoMesesPrestamoKeyTyped(java.awt.event.KeyEvent evt) {
char c = evt.getKeyChar();
    String text = txtPlazoMesesPrestamo.getText();

    // Permitir solo dígitos
    if (!Character.isDigit(c)) {
        evt.consume(); // Evitar caracteres que no sean dígitos
    }

    // Limitar la longitud total a 2 caracteres (para meses)
    if (text.length() >= 2) {
        evt.consume(); // Evitar que se exceda la longitud total
    }       // TODO add your handling code here:
    }

    private void txtFechaPrestamoKeyTyped(java.awt.event.KeyEvent evt) {
   char c = evt.getKeyChar();
    String text = txtFechaPrestamo.getText();

    // Permitir solo dígitos y el guion
    if (!Character.isDigit(c) && c != '-') {
        evt.consume(); // Evitar caracteres que no sean dígitos o guiones
    }

    // Limitar la longitud total a 10 caracteres (DD-MM-YYYY)
    if (text.length() >= 10) {
        evt.consume(); // Evitar que se exceda la longitud total
    }

    // Evitar más de un guion consecutivo o en posiciones incorrectas
```

```
    if (c == '-') {
        // Evitar guiones al inicio o consecutivos
        if (text.length() == 0 || text.charAt(text.length() - 1) == '-') {
            evt.consume(); // Evitar guiones al inicio o consecutivos
        } else {
            // Permitir solo 2 guiones en las posiciones adecuadas
            int dashCount = 0;
            for (int i = 0; i < text.length(); i++) {
                if (text.charAt(i) == '-') {
                    dashCount++;
                }
            }
            // Verificar si hay menos de 2 guiones
            if (dashCount >= 2) {
                evt.consume(); // Evitar más de 2 guiones
            }
        }
    }
    }

    private void txtApellidoClienteKeyTyped(java.awt.event.KeyEvent evt) {
        char c = evt.getKeyChar();
    String text = txtApellidoCliente.getText();

    // Permitir solo letras y espacios
    if (!Character.isLetter(c) && c != ' ') {
        evt.consume(); // Evitar caracteres que no sean letras o espacios
    }

     // Limitar la longitud total a 50 caracteres (puedes ajustar este número según tus
necesidades)
    if (text.length() >= 50) {
        evt.consume(); // Evitar que se exceda la longitud total
    }
    }

    private void txtDireccionClienteKeyTyped(java.awt.event.KeyEvent evt) {

    }

    private void txtEmailClienteKeyTyped(java.awt.event.KeyEvent evt) {
        char c = evt.getKeyChar();
    String text = txtEmailCliente.getText();
```

```
    // Permitir letras, números, puntos, guiones y arroba
     if (!Character.isLetter(c) && !Character.isDigit(c) && c != '.' && c != '_' && c != '-'
&& c != '@') {
        evt.consume(); // Evitar caracteres que no sean válidos
    }

    // Asegurarse de que el símbolo '@' y el dominio de Gmail estén en su lugar
    if (text.contains("@") && c != '\b') {
        // Evitar que se ingrese más de un '@'
        if (c == '@') {
            evt.consume(); // Evitar múltiples símbolos '@'
        }
        // Validar el dominio
        if (!text.endsWith("@gmail.com") && text.contains("@")) {
            if (text.length() > 25) { // Evitar que el texto supere el límite razonable
                evt.consume();
            }
        }
    }

     // Limitar la longitud total a 50 caracteres (puedes ajustar este número según tus
necesidades)
    if (text.length() >= 50) {
        evt.consume(); // Evitar que se exceda la longitud total
    }

    }

                                                    private                  void
txtFechaRegistroClienteActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
    }

    private void txtFechaRegistroClienteKeyTyped(java.awt.event.KeyEvent evt) {
    char c = evt.getKeyChar();
    String text = txtFechaRegistroCliente.getText();

    // Permitir solo dígitos y el guion
    if (!Character.isDigit(c) && c != '-') {
        evt.consume(); // Evitar caracteres que no sean dígitos o guiones
    }

    // Limitar la longitud total a 10 caracteres (YYYY-MM-DD)
    if (text.length() >= 10) {
```

```
            evt.consume(); // Evitar que se exceda la longitud total
        }

        // Controlar la posición del guion
        if (c == '-') {
            // Evitar guiones al inicio o consecutivos
            if (text.length() == 0 || text.charAt(text.length() - 1) == '-') {
                evt.consume(); // Evitar guiones al inicio o consecutivos
            } else {
                // Permitir solo 2 guiones en las posiciones adecuadas
                int dashCount = 0;
                for (int i = 0; i < text.length(); i++) {
                    if (text.charAt(i) == '-') {
                        dashCount++;
                    }
                }
                // Verificar si hay menos de 2 guiones
                if (dashCount >= 2) {
                    evt.consume(); // Evitar más de 2 guiones
                }
            }
        }
        }
        public static void main(String args[]) {
            /* Set the Nimbus look and feel */
                //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
            /* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
                                                    * For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
             */
            try {
                                        for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {
                    if ("Nimbus".equals(info.getName())) {
                        javax.swing.UIManager.setLookAndFeel(info.getClassName());
                        break;
                    }
                }
            } catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(SistemaBancarioLogin.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
```

```
        } catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(SistemaBancarioLogin.class.getName()).log(jav
a.util.logging.Level.SEVERE, null, ex);
        } catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(SistemaBancarioLogin.class.getName()).log(jav
a.util.logging.Level.SEVERE, null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(SistemaBancarioLogin.class.getName()).log(jav
a.util.logging.Level.SEVERE, null, ex);
        }
        //</editor-fold>
        //</editor-fold>

        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new SistemaBancarioLogin().setVisible(true);
            }
        });
    }

    // Variables declaration - do not modify
    private javax.swing.JButton DesactivarPrestamo;
    private javax.swing.JButton Modificar;
    private javax.swing.JFrame SistemaBancarioLogin;
    private javax.swing.JButton btnAplicarCambiosCliente;
    private javax.swing.JButton btnAplicarCambiosEmpleados;
    private javax.swing.JButton btnClientes;
    private javax.swing.JButton btnCrearCliente;
    private javax.swing.JButton btnCrearEmpleado;
    private javax.swing.JButton btnDeposito;
    private javax.swing.JButton btnEliminarCliente;
    private javax.swing.JButton btnEliminarEmpleado;
    private javax.swing.JButton btnEmpleados;
    private javax.swing.JButton btnIngresar1;
    private javax.swing.JButton btnModificarCliente;
    private javax.swing.JButton btnNuevaTransaccion;
    private javax.swing.JButton btnNuevoCliente;
    private javax.swing.JButton btnNuevoEmpleado;
    private javax.swing.JButton btnNuevoPrestamo;
    private javax.swing.JButton btnPrestamos;
```

```
private javax.swing.JButton btnRealizarPrestamo;
private javax.swing.JButton btnRetiro;
private javax.swing.JButton btnTransacciones;
private javax.swing.JComboBox<String> cboRolEmpleado;
private javax.swing.JComboBox<String> cboTipoCuenta;
private javax.swing.JButton jButton16;
private javax.swing.JButton jButton17;
private javax.swing.JComboBox<String> jComboBox2;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel10;
private javax.swing.JLabel jLabel12;
private javax.swing.JLabel jLabel2;
private javax.swing.JLabel jLabel3;
private javax.swing.JLabel jLabel4;
private javax.swing.JLabel jLabel5;
private javax.swing.JLabel jLabel6;
private javax.swing.JLabel jLabel7;
private javax.swing.JLabel jLabel8;
private javax.swing.JLabel jLabel9;
private javax.swing.JPanel jPanel1;
private javax.swing.JPanel jPanel2;
private javax.swing.JScrollPane jScrollPane1;
private javax.swing.JScrollPane jScrollPane2;
private javax.swing.JScrollPane jScrollPane4;
private javax.swing.JScrollPane jScrollPane5;
private javax.swing.JTextArea jTextArea2;
private javax.swing.JTextField jTextField25;
private javax.swing.JTextField jTextField26;
private javax.swing.JTextField jTextField27;
private javax.swing.JTextField jTextField28;
private javax.swing.JPanel panelClientes;
private javax.swing.JPanel panelDatos;
private javax.swing.JPanel panelEmpleados;
private javax.swing.JPanel panelEncabezado2;
private javax.swing.JPanel panelLogin;
private javax.swing.JPanel panelOpciones;
private javax.swing.JPanel panelPagosPrestamos;
private javax.swing.JPanel panelPrestamo;
private javax.swing.JPanel panelTransacciones;
private javax.swing.JTable tablaClientes;
private javax.swing.JTable tableEmpleados;
private javax.swing.JTable tablePrestamo;
private javax.swing.JTextField txtApellidoCliente;
private javax.swing.JTextField txtApellidoEmpleado;
```

```
    private javax.swing.JTextField txtCelularCliente;
    private javax.swing.JTextField txtCelularEmpleado;
    private javax.swing.JPasswordField txtContraseña;
    private javax.swing.JTextField txtContraseñaEmpleado;
    private javax.swing.JTextField txtDireccionCliente;
    private javax.swing.JTextField txtDniCliente;
    private javax.swing.JTextField txtDniEmpleado;
    private javax.swing.JTextField txtEmailCliente;
    private javax.swing.JTextField txtFechaInicioEmpleado;
    private javax.swing.JTextField txtFechaPrestamo;
    private javax.swing.JTextField txtFechaRegistroCliente;
    private javax.swing.JTextField txtMontoPrestamo;
    private javax.swing.JTextField txtMontoTransaccion;
    private javax.swing.JLabel txtNombre2;
    private javax.swing.JTextField txtNombreCliente;
    private javax.swing.JTextField txtNombreEmpleado;
    private javax.swing.JTextField txtNumeroCuentaCliente;
    private javax.swing.JTextField txtNumeroCuentaPrestamo;
    private javax.swing.JTextField txtNumeroCuentaTransaccion;
    private javax.swing.JTextField txtPlazoMesesPrestamo;
    private javax.swing.JLabel txtRol2;
    private javax.swing.JTextField txtSalarioEmpleado;
    private javax.swing.JTextField txtSaldoCliente;
    private javax.swing.JTextField txtUsuario;
    private javax.swing.JTextField txtUsuarioEmpleado;
    private javax.swing.JLabel txtVaucherEmpleadoID;
    private javax.swing.JLabel txtVaucherFecha;
    private javax.swing.JLabel txtVaucherMonto;
    private javax.swing.JLabel txtVaucherNombre;
    private javax.swing.JLabel txtVaucherNumeroCuenta;
    private javax.swing.JLabel txtVaucherOperacion;
    // End of variables declaration
}
```