CLASE PROYECTO 1

Esta clase es la que tiene al método main por tanto esta es la clase principal, en esta clase se definen las variables que se ejecutaran al iniciar el programa y también están algunos de los métodos que se utilizaron en el proyecto.

Se definieron las variable estáticas que se utilizaron en el proyecto, seguido del método main y los demás métodos necesarios

Variables estáticas: Se definieron cuatro contadores que fueron utilizados durante el desarrollo del proyecto para llevar un conteo de distintas operaciones necesarias. En esta parte también se definió el arreglo de tipo **Clientes** (mas adelante se detallara esta Clase) con capacidad para guardar 5 objetos.

```
public class Proyecto1 {

    static int ContadorCuentas = 0; // Para  Cuentas
    static int ContadorOperaciones = 0; // Para Historial
    private static int contador; //para guardar clientes en distintas posiciones de array
    static int contador1 = 0; //tiene la cantidad de personas registradas

    private static Clientes clientes[] = new Clientes[5];
```

Método Main: Este es el método principal, el primero que se ejecutara al iniciar la ejecución del programa. Lo primero que realizara este método es instanciar todos los objetos del arreglo Clientes, luego inicializo uno de los contadores con el valor cero. Posterior a esto se procedió a crear un objeto en el cual se construyo una interfaz grafica la cual se definió como pantalla de inicio y se le pasa como parámetro el vector de clientes (Se detalla mas adelante).

```
public static void main(String[] args) {
    //creando arreglo de personas
    for (int i = 0; i < clientes.length; i++) {
        clientes[i] = new Clientes();
        //creara metodo para guardar
    }
    contador = 0;

       Crear y llamar la ventana de inicio
    HomePage ventanaInicio = new HomePage(clientes);
    ventanaInicio.setVisible(true);
}
```

Método Guardar: Este es el método que se utilizo para poder rellenar el vector Clientes con la información ingresada del usuario es por eso que se le pasaron tres parámetros y luego se pasa a almacenar esa información en cada posición gracias al contador definido en el método main. Se utilizo el manejo de excepciones al momento que el vector este lleno este mandara un mensaje por pantalla.

```
//Mettodo para guardar clientes
public static void guardar(String nombre, String apellido, int dpi) {
    try {
        //asignacion de datos a los clientes
        clientes[contador].setNombre(nombre);
        clientes[contador].setApellido(apellido);
        clientes[contador].setDPI(dpi);

        JOptionPane.showMessageDialog(null, "Cliente registrado exitosamente.");
        contador++;
        contador1++; //pendiente revision

    } catch (Exception ex) {
        //Saldra esto cuando ya existan 5 clientes y desea agregar otro
        JOptionPane.showMessageDialog(null, "NO SE PUEDE AGREGAR MAS CLIENTES");
    }
}
```

Método dpiDuplicado: Este es el método que se utiliza para verificar que el dpi que el usuario desea guardar en el vector no este ya registrado en cualquier posición del mismo. Se utilizo una bandera y un ciclo for para recorrer y todas las posiciones y si hay un valor idéntico cambia la bandera a true, en caso contrario el banderín sigue con el valor false y retorna este valor a otra formulario en el cual se va detallar mas adelante este se llama (CrearClientes).

```
//metodo para verificar si dpi es duplicado
public static boolean dpiDuplicado(int dpi) {
    boolean duplicado = false;
    for (int i = 0; i < 5; i++) {
        //si alguna de los datos guardados es igual al parametro EL DPI ES REPETIDO
        if (clientes[i].getDPI() == dpi) {
            duplicado = true;
        }
    }
    return duplicado;
}
```

CLASE CLIENTES

Esta clase fue creada con el fin de tener un molde para crear o registrar a los clientes que el usuario desee, primero se definieron los atributos generales para cualquier persona, también se definió como atributo un arreglo de tipo **Cuentas** (Esta clase se detalla mas adelante) con espacio para 5 objetos tipo Cuentas junto a un atributo para poder contar el indice del arreglo cuando se este llenando con los datos.

```
public class Clientes {

    //Atributos
    private int DPI;
    private String Apellido;
    private String nombre;
    Cuentas[] cuentaAsociada = {new Cuentas(), new Cuentas(), new Cuentas(), new Cuentas(), new Cuentas()};
    private int contadorCuentas;
```

Se crearon dos constructores, uno vació y otro recibe como parámetro el nombre, apellido y CUI de la persona

```java
    //constructor vacio
    public Clientes() {

    }

    //constructor
    public Clientes(int DPI, String Apellido, String nombre) {
        this.DPI = DPI;
        this.Apellido = Apellido;
        this.nombre = nombre;
        this.contadorCuentas = 0;
    }
```

Método AsociarCuenta: Después de esto se definió el método de la clase para poder asignarle una cuenta a un cliente en especifico, para esto se utilizo el contador que se definió como atributo entonces cada cliente tendra este contador igual a cero y cada que este cliente llame el método este contador se aumentara y cuando llegue a 5 el condiciona evitara que se sigan asociando cuentas a este cliente. Posterior a esto se crearon todos los métodos get y set.

```java
//METODOS PARA CLIENTES
public void AsociarCuenta(int id) {
    if (contadorCuentas < cuentaAsociada.length) {
        cuentaAsociada[contadorCuentas] = new Cuentas(id);
        JOptionPane.showMessageDialog(null, "Cuenta asociada exitosamente.");
        contadorCuentas++;
    } else {
        JOptionPane.showMessageDialog(null, "Este cliente llego al limite de cuentas asociadad.");
    }
}
```

CLASE CUENTAS

Esta clase fue creada para poder relacionarla con la clase Clientes estas tienen una relacion de uno a varios ya que un cliente puede tener varias cuentas. Cada cuenta tienen los mismo atributos los cuales son idCuenta y el saldo de la misma, también se definió un atributo de tipo **Historial** (clase detallada mas adelante) el cual tiene capacidad para 20 objetos de tipo Historial para poder tener un control de las transacciones que realiza cada cuenta, esto acompañado de un contador para poder guardar el numero de transacciones que lleva la cuenta.

```java
public class Cuentas {
    // atributos
    private int idCuenta;
    private double SaldoCuenta;
    Historial  operacion [] = new Historial[20];
    int ContadorTransacciones;
```

Se crearon dos constructores uno vació y otro que recibe como parámetros el id de la cuenta, en este constructor se aumentara en una unidad el valor para poder tener un id Único para cada cuenta. Y se inicializo el contador de transacciones igual a cero para cada cuenta creada.

```
public Cuentas() {

}

public Cuentas(int idCuenta) {
    this.idCuenta = (idCuenta+1);
    this.ContadorTransacciones = 0;
}
```

Método AsociarCuenta: Después de esto se definió el método de la clase para poder asignarle una operación o transacción a un cliente en especifico, para esto se utilizo el contador que se definió como atributo entonces cada cuenta tendra este contador igual a cero y cada que este cliente en esta cuenta llame el método este contador se aumentara y cuando llegue a 20 el condiciona evitara que se sigan guardando transacciones a la cuenta de este cliente. Posterior a esto se crearon todos los métodos get y set. Este método recibira varios parametros, como el id Transacción, la hora, descripción, MontoDebitado o Acreditado y el saldo actual de la cuenta.

```
public void AsociarOperacion(int idHistorial, LocalDateTime fechaHoraActuales, String Descripcion, double MontoDebitado, double M
    if(ContadorTransacciones < operacion.length){
        operacion[ContadorTransacciones] = new Historial(idHistorial, fechaHoraActuales, Descripcion, MontoDebitado, MontoAcredita
        ContadorTransacciones++;
    }
}
```

Método retirarSaldo: La función de este método es recibir una cantidad como parámetro y esta cantidad restarsela al saldo de la cuenta en donde se hace la transacción.

```
public void retirarSaldo(double SaldoCuenta){
    this.SaldoCuenta -= SaldoCuenta;
}
```

Método setSaldoCuenta: La función de este método es recibir una cantidad como parámetro y esta cantidad sumarsela al saldo de la cuenta en donde se hace la transacción.

```
public void setSaldoCuenta(double SaldoCuenta) {
    this.SaldoCuenta += SaldoCuenta;
}
```

CLASE HISTORIAL

Esta clase fue creada para poder tener un registro de las actividades relacinadas a las cuentas de igual manera esta tiene una relacion de una a varios ya que cada cuenta va tener muchas transacciones. Esta clase tendra varias atributos como id, fechaHora, Descripcion, MontoDebitado, MontoAcreditado y SaldoActual. El constructor recibe como parametros todos los atributos, luego están los respectivos get y set.

```
public class Historial {
    private int idHistorial;
    private LocalDateTime fechaHoraActuales;
    private String Descripcion;
    private double MontoDebitado;
    private double MontoAcreditado;
    private double saldoActual;


    //Atributos
    public Historial(int idHistorial, LocalDateTime fechaHoraActuales, String Descripcion, double MontoD
        this.idHistorial = (idHistorial+1);
        this.fechaHoraActuales = fechaHoraActuales;
        this.Descripcion = Descripcion;
        this.MontoDebitado = MontoDebitado;
        this.MontoAcreditado = MontoAcreditado;
        this.saldoActual = saldoActual;
    }
```

HOME PAGE

Lo primero que encontramos en este formulario es la librería importada y posteriormente el método constructor de la ventana. En este apartado se creo otro vector de tipo Clientes para poder almacenar los datos que se pasaron como parámetro al momento de llamar la ventana y también se creo un método vació para poder llamar a la ventana pero sin pasarle un arreglo de tipo Clientes.

```
public class HomePage extends javax.swing.JFrame {

    Clientes[] persona = new Clientes[5];

    public HomePage(Clientes[] pers) {
        initComponents();
        this.persona = pers;

    }

    public HomePage(){
        initComponents();
    }
```

Método btmAboutAction: Este método se ejecutara cuando el usuario presione el botón About que aparece en la interfaz grafica. Su funcionalidad es mostrar un mensaje por pantalla que contiene los datos del desarrollador de este programa.

```
private void btmAboutActionPerformed(java.awt.event.ActionEvent evt) {
    //mostrando datos en ventana emergente
    JOptionPane.showMessageDialog(null, "Mi nombre es: Jhonatan Alexander Aguilar Reyes\nMi carnet es: 202106003");
}
```

Método btmLoginAction: Este método se ejecutara cuando el usuario presione el botón login que aparece en la interfaz grafica. Su funcionalidad es dirigirnos a otra ventana. En este método unicamente se crea el objeto que contendrá la nueva ventana con el parámetro de tipo Clientes, esto para poder llevar los datos de los clientes hasta la ventana MenuPrincipal y también cierra la ventana actual.

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

        //redirigir a la proxima ventana
        Login ventanaRegistrar = new Login(persona);
        ventanaRegistrar.setVisible(true);
        this.dispose();
}
```

LOGIN

Lo primero que encontramos en este formulario es la librería importada y posteriormente el método constructor de la ventana. En este apartado se creo otro vector de tipo Clientes para poder almacenar los datos que se pasaron como parámetro al momento de llamar la ventana.

```
// se le pasa el arreglo de clientes para poder utilizarlos en esta ventana
Clientes[] persona = new Clientes[5];

public Login(Clientes[] pers) {
    initComponents();
    this.persona = pers;

}
```

Método btmRegresarAction: Este método simplemente limpia los campos en los que el usuario puede ingresar información y hace que regresemos a la ventana HOME PAGE cerrando la ventana actual.

```
private void btmRegresar1ActionPerformed(java.awt.event.ActionEvent evt) {
    // limpiar los textField
    try {
        txtUsuario.setText(null);
        txtPassword.setText(null);

        //volver a la ventana inicio
        HomePage ventanaInicio = new HomePage();
        ventanaInicio.setVisible(true);
        this.dispose();
```

Método btmInicioSecionAction: Este método se ejecuta cuando el usuario da clic en el botón "Iniciar sesión" en la interfaz grafica. Lo primero que hace es guardar la información que el usuario ingreso en dos variables de tipo String, Luego con un condicional valida si los campos de textos están vacíos si esto se cumple manda un menaje de advertencia por pantalla, en caso contrario valida con un segundo condicional si la información que ingreso el usuario es idéntica a los valores establecidos para dar acceso al programa, si esto se cumple se crea un objeto donde se construyo la ventana MENU PRINCIPAL y se le pasa como parámetro el vector clientes definido al principio y cierra la pagina actual. En caso contrario se procede a limpiar los campos de texto e indica al usuario por medio de un mensaje que el usuario o contraseña ingresados son incorrectos.

```
private void btmInicioSesionActionPerformed(java.awt.event.ActionEvent evt) {
    //condicionales para validar contrasena
    String user = txtUsuario.getText();
    String pass = txtPassword.getText();
    if (!txtUsuario.getText().isEmpty() && !txtPassword.getText().isEmpty()) {

        if (user.equals("administrador") && pass.equals("202106003")) {
            MenuPrincipal1 menu = new MenuPrincipal1(persona);
            menu.setVisible(true);
            this.dispose();

        } else {
            try {
                txtUsuario.setText("");
                txtPassword.setText("");
                JOptionPane.showMessageDialog(null, "Usuario o Contraseña Incorrectas");
            } catch (Exception ex) {

            }
        }
    } else {
        JOptionPane.showMessageDialog(null, "¡Debe rellenar todos los campos!");
    }
}
```

Menú Principal

En esta ventana se encuentran todas las funcionalidades que tiene el programa, la primer opción es "Registrar Clientes" en este apartado es donde se podrá registrar los clientes que estarán en el vector creado al principio de tipo Clientes. La segunda opción es "Crear Cuentas" en este apartado se podrá se podrá asignarle cuentas a los diferentes clientes creados anteriormente (5 cuentas por cliente máximo). La tercera opción es "Información Clientes" en este apartado se podrá visualizar la información de los clientes creados y se podrá realizar la búsqueda de cuantas cuentas tiene asociadas un cliente mediante su DPI. La cuarta opción es "Deposito" en este apartado se puede acreditar X cantidad de dinero a cualquiera de las cuentas creadas anteriormente. La quita opcion es "Transferencia" en este apartado se puede debitar X cantidad de dinero a una cuenta y acreditarsela a otra cuenta. La sexta opción es "Pago de Servicio" En este apartado se puede debitar X cantidad de dinero a una cuenta por algún servicio y la ultima opción es un "Historial de Transferencia" en el cual se realizara una búsqueda por id de la cuenta y aparecerán todas las transacciones que haya realizado.

REGISTRAR CLIENTES

Lo primero que encontramos en este formulario es la librería importada y posteriormente el método constructor de la ventana. En este apartado se creo otro vector de tipo Clientes para poder almacenar los datos que se pasaron como parámetro al momento de llamar la ventana.

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

    Clientes[] infoPersona = new Clientes[5];

    public CrearClientes(Clientes[] pers) {
        initComponents();
        this.infoPersona = pers;
    }
```

Método btmCrearAction: Este método se ejecuta cuendo el usuario presiona el botón Crear en la interfaz grafica. Primero valida que todos los campos de texto este llenos si se cumple realiza las siguientes operaciones: primero asigna el valor de los campos de texto a variables de tipo String y pasea el DPI para que sea un valor numero para posteriormente compararlo con los demás DPI. Posterior a eso entra a un condicional donde si el contador definido en la clase main es igual a cero entonces se llama el método **guardar** localizado en la clase principal (ya se detallo la funcionalidad del método), posterior a eso se limpian los campos de texto para que se pueda ingresar la información del siguiente cliente y en caso contrario el contador es diferente de cero entonces se llama al método **dpiDuplicado** y en el condicional si este método retorna "falso" se procederá a guardar al usuario en caso contrario mandara un mensaje de alerta donde se indica que no deben haber DPI duplicado.

```
    if (!txtapellido.getText().isEmpty() && !txtcui.getText().isEmpty() //verificar que los campos tengan informacion
            && !txtnombre.getText().isEmpty()) {
        try {

            String dpi = txtcui.getText();
            String name = txtnombre.getText();
            String lastName = txtapellido.getText();
            //pasar el valor DPI a int
            int sdpi = Integer.parseInt(dpi);

            if (Proyecto1.contador1 == 0) { //para guardar la primera posicion y no comparar con otros DPI
                Proyecto1.guardar(name, lastName, sdpi); //guardar

                //limpiar los txt
                txtcui.setText("");
                txtnombre.setText("");
                txtapellido.setText("");
            } else if (Proyecto1.dpiDuplicado(sdpi) == false) {  // comparar DPI
                //para poder utilizar info guardada en otros lugares
                //para poder notificar que el cliente se guardo exitosamente
                Proyecto1.guardar(name, lastName, sdpi);

                txtcui.setText("");
                txtnombre.setText("");
                txtapellido.setText("");
            } else {
                JOptionPane.showMessageDialog(null, "No es posible crear clientes con\nCUI duplicados. El CUI ingresa
                txtcui.setText("");
                txtnombre.setText("");
                txtapellido.setText("");
            }
        } catch (Exception ex) {
            JOptionPane.showMessageDialog(null, "¡Ingrese los datos como lo indica el manual de usuario!");
            txtcui.setText("");
            txtnombre.setText("");
            txtapellido.setText("");
        }
    } else {
        JOptionPane.showMessageDialog(null, "¡Debe rellenar todos los campos!");
        txtcui.setText("");
        txtnombre.setText("");
        txtapellido.setText("");
    }
}
```

CREAR CUENTAS

Lo primero que encontramos en este formulario es la librería importada y posteriormente el método constructor de la ventana. En este apartado se creo otro vector de tipo Clientes para poder almacenar los datos que se pasaron como parámetro al momento de llamar la ventana y también se definió una variable publica para poder guardar el indice elegido en el combobox.

```
public int indice;
Clientes[] infoPersona = new Clientes[5];

public CrearCuentas(Clientes[] personas) {
    initComponents();
    this.infoPersona = personas;
    //instanciar cada objeto del arreglo
    mostarInformacion();
}
```

Método mostrarInformacion: Este método es el encargado de eliminar los items que contiene el combobox por defecto y rellenarlos con un ciclo for, recorriendolo y utilizando los métodos del combobox agregarlos como lista de opciones.

```
public void mostarInformacion() {
    combo1.removeAllItems();
    for (int i = 0; i < infoPersona.length; i++) {
        if (infoPersona[i].getNombre() != null) {
            combo1.addItem(infoPersona[i].getDPI() + " - " + infoPersona[i].getNombre() + " " + infoPersona[i].getApellido());
        }
    }
}
```

Método mtbAsignarAction: Este método es llamado cuando el usuario presiona el botón “Asignar” en la interfaz grafica. La funcion de este método es tomar el indice de la opcion seleccionada en el combobox y con esa información pasar por un condicional switch y con esta información llamar a la persona del arreglo en esa posición (Indice) para luego llamar al método asociado a su clase **AsociarCuenta** y pasarle como parametro el contador definido en la clase principal main y después de realizar la operación del método llamado aumentar en uno el contador. Esto con cada uno de los casos respectivamente del caso que se seleccione por el usuario.

```
private void BTMAsignarActionPerformed(java.awt.event.ActionEvent evt) {
    indice = combo1.getSelectedIndex();

    switch (indice) {
        case 0: // cliente 1
            infoPersona[0].AsociarCuenta(Proyecto1.ContadorCuentas);
            Proyecto1.ContadorCuentas++;
            break;
        case 1: // cliente 2
            infoPersona[1].AsociarCuenta(Proyecto1.ContadorCuentas);
            Proyecto1.ContadorCuentas++;
            break;
        case 2: //cliente 3
            infoPersona[2].AsociarCuenta(Proyecto1.ContadorCuentas);
            Proyecto1.ContadorCuentas++;
            break;
        case 3: //cliente 4
            infoPersona[3].AsociarCuenta(Proyecto1.ContadorCuentas);
            Proyecto1.ContadorCuentas++;

            break;
        case 4: //cliente 5
            infoPersona[4].AsociarCuenta(Proyecto1.ContadorCuentas);
            Proyecto1.ContadorCuentas++;
            break;
    }
```

INFORMACION CLIENTES

Lo primero que encontramos en este formulario es las librería importada, posteriormente la inicializacion de dos tablas y el método constructor de la ventana. En este apartado se creo otro vector de tipo Clientes para poder almacenar los datos que se pasaron como parámetro al momento de llamar la ventana, dentro de este constructor se construyo la primera fila de ambas tablas.

```
package proyecto1;

import javax.swing.JOptionPane;
import javax.swing.table.DefaultTableModel;

public class InformacionClientes extends javax.swing.JInternalFrame {
    //tablas
    DefaultTableModel dtm = new DefaultTableModel();
    DefaultTableModel dtm1 = new DefaultTableModel();

    //arreglo de clientes
    Clientes[] infoPersona = new Clientes[5];

    //Constructor de la ventana
    public InformacionClientes(Clientes[] personas) {
        this.infoPersona = personas;
        initComponents();

        // para tabla de informacion cliente
        String[] titulo = new String[]{"CUI", "NOMBRE", "APELLIDO"};
        dtm.setColumnIdentifiers(titulo);
        tblDatos.setModel(dtm);
        //metodo ingresa informacion a la tabla
        agregarInfo();

        //para tabla de informacion cuenta asociada
        String[] title = new String[]{"Cuentas Asociadas"};
        dtm1.setColumnIdentifiers(title);
```

Método agregarInfo: Este método se utilizo para que el usuario pueda visualizar la información de los clientes ya creados al momento de ingresar a la ventana. Se utilizo un ciclo for y un condicional para

filtrar mas la búsqueda de los clientes ya registrados, posterior a esto con métodos propios de las tablas se agrego la información de los clientes en columnas diferentes.

```
//PARA AGREGAR DATOS A LA TABLA DE INFORMACION
public void agregarInfo() {
    for (int i = 0; i < infoPersona.length; i++) {
        if (infoPersona[i].getNombre() != null) {
            dtm.addRow(new Object[]{
                infoPersona[i].getDPI(), infoPersona[i].getNombre(), infoPersona[i].getApellido()
            });
        }
    } // finaliza el for

}
```

Método agregarInfoBusqueda: Este método es identico al anterior con la diferencia que recibe un parametro entero que es el indice de la persona que se desea buscar que cuentas tiene, por lo tanto se fija ese valor y la variable iteradora se le asigna al vector de cuentas para que este muestre todas las cuentas asociadas de esta persona.

```
//Metodo para mostrar informacion en la tabla de busqueda
public void AgregarInfoBusqueda(int per) { //parametro para tener fijada a la persona que veremos
    tb2Datos.setModel(dtm1);
    for (int i = 0; i < infoPersona.length; i++) {
        if (infoPersona[per].cuentaAsociada[i].getIdCuenta() != 0) { // valido que el id no sea 0
            dtm1.addRow(new Object[]{
                infoPersona[per].cuentaAsociada[i].getIdCuenta()
            });
        }
    }
}
```

Método btmBuscarAction: Este método inicia cuando el usuario ingresa el valor de un DPI en el campo y presiona el botón buscar. Primero valida que los campos esten llenos, posterior crea un contador igual a cero y con un condicional valida que si el valor ingresado es cero que el contador aumente para después poder mandar un mensaje de alerta por los datos erroneos, posterior a eso se limpian los valores de la segunda tabla para que puedan ingresar los valores de la siguiente busqueda. Se procede a guardar el valor ingresado en el campo en una variable y posteriormente con un for validar que alguno de los clientes tenga ese DPI, cuando encuentre coincidencias llamara el método mencionado anteriormente con el parámetro iterador.

```java
private void btmBuscarActionPerformed(java.awt.event.ActionEvent evt) {
    if (!txtIdBuscar.getText().isEmpty()) {
        //Para limpiar la tabla
        try {
            int ContadorCoincidencia = 0;
            int filas = dtm1.getRowCount();
            for (int i = 0; i < filas; i++) {
                dtm1.removeRow(0);
            }

            // para validar el valor a buscar
            String datoBuscar = txtIdBuscar.getText();
            int dpi = Integer.parseInt(datoBuscar);

            //buscar coincidencias
            for (int i = 0; i < infoPersona.length; i++) {
                if (dpi == 0) {
                    ContadorCoincidencia++;
                } else if (infoPersona[i].getDPI() == dpi) {
                    AgregarInfoBusqueda(i);
                    ContadorCoincidencia = (i + 1);
                }
            }
            // si la busqueda no encuentra coincidencias
            if (ContadorCoincidencia == 0) {
                JOptionPane.showMessageDialog(null, "No se encontraron Coincidencias");
```

DEPOSITO

Lo primero que encontramos en este formulario es la librerías importadas y posteriormente el método constructor de la ventana. En este apartado se creo otro vector de tipo Clientes para poder almacenar los datos que se pasaron como parámetro al momento de llamar la ventana y también se definió una variable publica para poder guardar el indice elegido en el combobox.

```java
package proyecto1;

import java.time.LocalDateTime;
import javax.swing.JOptionPane;


public class VDepositos extends javax.swing.JInternalFrame {

    public int indice;
    Clientes[] infoPersona = new Clientes[5];

    public VDepositos(Clientes[] persona) {
        this.infoPersona = persona;
        initComponents();
        mostrarInformacionCuentas();
    }
```

Método mostrarInformacion: Este método es el encargado de eliminar los items que contiene el combobox por defecto y rellenarlos con un ciclo for, recorriendolo y utilizando los métodos del combobox agregar las cuentas y la información de sus dueños.

```
public void mostrarInformacionCuentas() {
    comboCuentas.removeAllItems();
    for (int i = 0; i < infoPersona.length; i++) {
        if (infoPersona[i].getNombre() != null) {  //si el nombre esta vacio NO SE CUENTA
            for (int j = 0; j < 5; j++) {   // depues de elegir la posicion de la persona, se utilizara este for para recorrer
                if (infoPersona[i].cuentaAsociada[j].getIdCuenta() != 0) { //Cuentas con id = 0 NO SE CUENTAN
                    comboCuentas.addItem(infoPersona[i].cuentaAsociada[j].getIdCuenta() + "- Cuenta de " + infoPersona[i].getNo
                }
            }

        }
    }
}
```

Método btmAceptarAction: Este método es llamado cuando el cliente da clic en el botón aceptar de la interfaz grafica. Primero toma el indice de la opcion seleccionada por el usuario, luego guarda la información obtenida por el campo de texto. Posteriormente entre a una serie de condicionales y bucles para lograr obtener el id de la cuenta de cada persona y compararla con el indice seleccionado por el usuario. Si se cumple entonces se llama el método s**etSaldoCuenta** y se le pasa como parametro el monto ingresado. Despues se procede a enviar la información de la transacion a la clase Historial.

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

    if (!txtCantidad.getText().isEmpty()) {

        try {
            indice = comboCuentas.getSelectedIndex();
            String cantidad = txtCantidad.getText();
            //combierto a entero el monto ingresado
            double monto = Double.parseDouble(cantidad);
            if (monto > 0) {
                for (int i = 0; i < infoPersona.length; i++) {
                    if (infoPersona[i].getNombre() != null) {
                        for (int j = 0; j < 5; j++) {
                            if (infoPersona[i].cuentaAsociada[j].getIdCuenta() == (indice + 1)) { // validar si encontro al
                                infoPersona[i].cuentaAsociada[j].setSaldoCuenta(monto); //realizar la operacion

                                // Realizar asignacion al historial
                                LocalDateTime horaFecha = LocalDateTime.now(); //guardar la fecha y la hora
                                String info = "Deposito";   // guardar la descripcion de la operacion
                                double saldoActual = infoPersona[i].cuentaAsociada[j].getSaldoCuenta();
                                infoPersona[i].cuentaAsociada[j].AsociarOperacion(Proyecto1.ContadorOperaciones, horaFecha,
                                Proyecto1.ContadorOperaciones++; //aumenta la cantidad de operaciones

                                //Mensajes de confirmacion
                                JOptionPane.showMessageDialog(null, "Deposito realizado exitosamente.");
                            }
                        }
                    }
                }
            } else {
```

TRANSFERENCIA

Lo primero que encontramos en este formulario es la librerías importadas y posteriormente el método constructor de la ventana. En este apartado se creo otro vector de tipo Clientes para poder almacenar los datos que se pasaron como parámetro al momento de llamar la ventana y también se definió dos variables publicas para poder guardar el indice elegido en el combobox.

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

    //Indices para indicar las cuentas que realizaran las operaciones
    public int indiceTransferir;
    public int indiceAcreditar;
    //arreglo con la informacion de clientes
    Clientes[] infoPersona = new Clientes[5];

    public VTransferencias(Clientes[] persona) {
        this.infoPersona = persona;
        initComponents();
        mostrarInformacionCuentasTransferencia();
        mostrarInformacionCuentasAcreditar();
    }
```

Método mostrarInformacionCuentasTransferencia y Acreditar: Son dos métodos identicos pero que se muestran en diferentes combobox, primero se eliminan todos los items por defecto y luego con una serie de ciclos y condicionales se hace una busqueda para encontrar las cuentas y los datos de su dueño y mostrarlas como opciones en el combobox.

```
public void mostrarInformacionCuentasTransferencia() {
    comboTransferir.removeAllItems();
    for (int i = 0; i < infoPersona.length; i++) {
        if (infoPersona[i].getNombre() != null) {  //si el nombre esta vacio NO SE CUENTA
            for (int j = 0; j < 5; j++) {   // depues de elegir la posicion de la persona, se utilizara este for para recorre
                if (infoPersona[i].cuentaAsociada[j].getIdCuenta() != 0) { //Cuentas con id = 0 NO SE CUENTAN
                    comboTransferir.addItem(infoPersona[i].cuentaAsociada[j].getIdCuenta() + " - Cuenta de " + infoPersona[i]
                }
            }

        }
    }
}
```

Método btmTransferirAction: Este método es llamado cuando el cliente da clic en el botón transferir de la interfaz grafica. Primero toma el indice de la opcion seleccionada por el usuario, luego guarda la información obtenida por el campo de texto. Posteriormente entre a una serie de condicionales y bucles para lograr obtener el id de la cuenta de cada persona y compararla con el indice seleccionado por el usuario. Si se cumple entonces se llama el método **retirarSaldo** y se le pasa como parametro el monto ingresado. Despues se procede a enviar la información de la transacion a la clase Historial. Posterior a esto se procede a hacer la misma busqueda con los mismo condiciones con la diferencia que ahora se utilizara el segundo indice guardado para encontrar la cuenta a la que se tiene que acreditar la cantidad del monto y guardar los datos del Historial para tener el registro.

```
indiceTransferir = comboTransferir.getSelectedIndex();
indiceAcreditar = comboAcreditar.getSelectedIndex();

//validar que el valor ingresado sea correcto y no este vacio
if (!txtMonto.getText().isEmpty()) {

    //Para librar excepciones en el txt
    try {
        if (indiceTransferir == indiceAcreditar) {  // validar que las cuentas no sean iguales
            JOptionPane.showMessageDialog(null, "La cuenta destino no puede ser igual a la cuenta origen.");
        } else {
            //Validando el monto para las operaciones
            String cantidad = txtMonto.getText();
            double monto = Double.parseDouble(cantidad); //obtener la cantidad del monto

            if (monto > 0) { //validacion monto > 0
                for (int i = 0; i < infoPersona.length; i++) { // recorre arreglo de clientes
                    if (infoPersona[i].getNombre() != null) { // excepto los que no estan asignados = Null
                        for (int j = 0; j < 5; j++) {  // recorre arreglo de las cuentas del cliente al
                            if (infoPersona[i].cuentaAsociada[j].getIdCuenta() == (indiceTransferir + 1)) { //valida
                                //validar que el monto no supere el saldo de la cuenta
                                if (infoPersona[i].cuentaAsociada[j].getSaldoCuenta() >= monto) {   // validar que e
                                    infoPersona[i].cuentaAsociada[j].retirarSaldo(monto);   // de cumplirse, se reti

                                    // Realizar asignacion al historial
                                    LocalDateTime horaFecha = LocalDateTime.now(); //guardar la fecha y la hora
                                    String info = "Transferencia";   // guardar la descripcion de la operacion
                                    double saldoActual = infoPersona[i].cuentaAsociada[j].getSaldoCuenta();
                                    infoPersona[i].cuentaAsociada[j].AsociarOperacion(Proyecto1.ContadorOperaciones,


                                    Proyecto1.ContadorOperaciones++; //aumenta la cantidad de operaciones

                                } else {
                                    JOptionPane.showMessageDialog(null, "La cuenta origen no tiene suficientes fondos.");
                                    monto = 0;  // se hace el valor del monto = 0 Para que al seguir el codigo no pueda seg
                                    break;
                                }
                            }
                        }
                    }
                }
                // hacer el mismo procedimiento para la cuenta, pero esta ves acreditando el monto debitado a la anterior
                for (int i = 0; i < infoPersona.length; i++) {
                    if (infoPersona[i].getNombre() != null) {
                        for (int j = 0; j < 5; j++) {
                            if (infoPersona[i].cuentaAsociada[j].getIdCuenta() == (indiceAcreditar + 1) && monto != 0) {  /

                                infoPersona[i].cuentaAsociada[j].setSaldoCuenta(monto);     // acreditar saldo

                                // Realizar asignacion al historial
                                LocalDateTime horaFecha = LocalDateTime.now(); //guardar la fecha y la hora
                                String info = "Transferencia";   // guardar la descripcion de la operacion
                                double saldoActual = infoPersona[i].cuentaAsociada[j].getSaldoCuenta();
                                infoPersona[i].cuentaAsociada[j].AsociarOperacion(Proyecto1.ContadorOperaciones, horaFecha,
                                Proyecto1.ContadorOperaciones++; //aumenta la cantidad de operaciones

                                if (monto == 0) { // Para esto se hizo el valor del monto = 0 en la linea PENDIENTE

                                } else {
                                    JOptionPane.showMessageDialog(null, "Transferencia realizada exitosamente.");
```

PAGO SERVICIO

Lo primero que encontramos en este formulario es la librerías importadas y posteriormente el método constructor de la ventana. En este apartado se creo otro vector de tipo Clientes para poder almacenar los datos que se pasaron como parámetro al momento de llamar la ventana y también se definió dos variables publicas para poder guardar el indice elegido en el combobox.

```java
package proyecto1;

import java.time.LocalDateTime;
import javax.swing.JOptionPane;

public class VPagoServicio extends javax.swing.JInternalFrame {

    public int indiceCuenta;
    public int indiceServicio;
    Clientes[] infoPersona = new Clientes[5];

    public VPagoServicio(Clientes[] persona) {
        this.infoPersona = persona;
        initComponents();
        mostrarInformacionCuentas();
        mostrarInformacionPagos();
    }
```

Método mostrarInformacionPagos: Este método establece los items que tendra este combobox.

```java
//Elegir el tipo de pago a realizar
public void mostrarInformacionPagos() {
    comboServicio.removeAllItems();
    comboServicio.addItem("Luz Electrica");
    comboServicio.addItem("Agua");
    comboServicio.addItem("Servicio Telefonico");
}
```

Método mtbRealizarPagoAction: Este método es llamado cuando el usuario preciona el botón realizar Pago en la interfaz grafica, este método es identico a los demás, con la ayuda de ciclos y condicionales busca la cuenta seleccionada y valida que esta tenga saldo suficiente para hacer el pago y posteriormente guardar toda la información de la transacción para pasarla a la otra ventana.

```java
// validaciones y operaciones de la ventana
if (monto > 0) {
    for (int i = 0; i < infoPersona.length; i++) { // recorre arreglo de clientes
        if (infoPersona[i].getNombre() != null) { // excepto los que no estan asignados = Null
            for (int j = 0; j < 5; j++) {  // recorre arreglo de las cuentas del cliente al
                if (infoPersona[i].cuentaAsociada[j].getIdCuenta() == (indiceCuenta + 1)) { //valida que un cliente
                    //validar que el monto no supere el saldo de la cuenta
                    if (infoPersona[i].cuentaAsociada[j].getSaldoCuenta() >= monto) {   // validar que el monto a de
                        infoPersona[i].cuentaAsociada[j].retirarSaldo(monto);   // de cumplirse, se retira el fondo
                        JOptionPane.showMessageDialog(null, "Pago realizado exitosamente.");

                        // Realizar asignacion al historial
                        LocalDateTime horaFecha = LocalDateTime.now(); //guardar la fecha y la hora
                        String info = "Pago de servicio - " + n1;   // guardar la descripcion de la operacion
                        double saldoActual = infoPersona[i].cuentaAsociada[j].getSaldoCuenta();
                        infoPersona[i].cuentaAsociada[j].AsociarOperacion(Proyecto1.ContadorOperaciones, horaFecha,
                        Proyecto1.ContadorOperaciones++; //aumenta la cantidad de operaciones

                    } else {
                        JOptionPane.showMessageDialog(null, "La cuenta origen no tiene suficientes fondos.");
                    }
```

HISTORIAL TRANSFERENCIAS

Lo primero que encontramos en este formulario es las librerías importadas, posteriormente la inicializacion de una tabla y el método constructor de la ventana. En este apartado se creo otro vector de tipo Clientes para poder almacenar los datos que se pasaron como parámetro al momento de llamar la ventana, dentro de este constructor se construyo la primera fila de la tabla.

```
 */
public class VHistorial extends javax.swing.JInternalFrame {

    //tabla
    DefaultTableModel dtm = new DefaultTableModel();

    //arreglo clientes
    Clientes[] infoPersona = new Clientes[5];

    public VHistorial(Clientes[] persona) {
        this.infoPersona = persona;
        initComponents();

        // para tabla de informacion cliente
        String[] titulo = new String[]{"ID Transaccion", "Fecha", "Detalle", "Debito", "Credito", "Saldo Disponible"}
        dtm.setColumnIdentifiers(titulo);
        tblHistorial.setModel(dtm);
```

Método mtbHistorialAction: Este método primero limpia por completo la tabla, luego guarda el valor ingresado en el campo de texto en una variable y la transforma a entero. Posterior a eso entra a una serie de ciclos y condicionales para buscar y comparar el dato ingresado con el id guardado de las cuentas de los clientes y al encontrarlas muestra los datos del cliente y las transacciones realizadas por estas cuentas. Para esto hace uso de un método llamado **mostrarInformacion** que lleva como parametro las variable iteradoras de los ciclos con la finalidad de fijar a la persona y la cuenta en la que se desea buscar un historial.

```
    int filas = dtm.getRowCount();
    for (int i = 0; i < filas; i++) {
        dtm.removeRow(0);
    }

    String id = txtIdCuenta.getText(); //obtener dato ingresado
    int idCuenta = Integer.parseInt(id); //convertirlo a entero
    String nombre;
    int cui;
    String apellido;

    for (int i = 0; i < infoPersona.length; i++) { // recorre arreglo de clientes
        if (infoPersona[i].getNombre() != null) { // excepto los que no estan asignados = Null
            for (int j = 0; j < 5; j++) {  // recorre arreglo de las cuentas del cliente al
                if (infoPersona[i].cuentaAsociada[j].getIdCuenta() == idCuenta) { //valida que un cliente
                    //guardar los datos necesarios
                    apellido = infoPersona[i].getApellido();
                    cui = infoPersona[i].getDPI();
                    nombre = infoPersona[i].getNombre();
                    //hacer que sean visibles
                    txtApellido.setText(apellido);
                    txtCui.setText(Integer.toString(cui));
                    txtNombre.setText(nombre);

                    //Se pasan las variables de los for para realizar el proximo procedimiento
                    mostrarInformacion(i, j);
                }
            }
        }
    }
```

Método mostrarInformacion: Este método es llamado al momento de hacer una busqueda de transacciones este es el encargado de crear y mostrar los datos en la tabla y gracias a sus parametros ya solo se crea otro for (tercer for) el cual se va encargar de recorrer el arreglo Histotial que tiene como atributo la clase Cuentas

```java
//metodo para mostrar los datos de la tabla
public void mostrarInformacion(int fijoi, int fijoj) { //son las posiciones en las que esta la persona y la cuenta
    //Este procedimiento es el tercer for del algoritmo
    for (int j = 0; j < infoPersona[fijoi].cuentaAsociada[fijoj].ContadorTransacciones; j++) {
        dtm.addRow(new Object[]{
            infoPersona[fijoi].cuentaAsociada[fijoj].operacion[j].getIdHistorial(), //agrega primera columna, prir
            infoPersona[fijoi].cuentaAsociada[fijoj].operacion[j].getFechaHoraActuales(), //agrega segunda columna
            infoPersona[fijoi].cuentaAsociada[fijoj].operacion[j].getDescripcion(), //agrega Tercera columna, prir
            infoPersona[fijoi].cuentaAsociada[fijoj].operacion[j].getMontoDebitado(), //agrega cuarta columna, pri
            infoPersona[fijoi].cuentaAsociada[fijoj].operacion[j].getMontoAcreditado(), //agrega quinta columna, p
            infoPersona[fijoi].cuentaAsociada[fijoj].operacion[j].getSaldoActual() //agrega sexta columna, primera
        });
    }

}
```

VIDEO EXPLICATIVO DEL PROGRAMA:

https://drive.google.com/file/d/1xdSNtOd0mRIsqyCCPipASdOJ06aldjbg/view?usp=sharing