PROYECTO 1
MANUAL TÉCNICO

ENTIDAD BANCARIA

DATOS

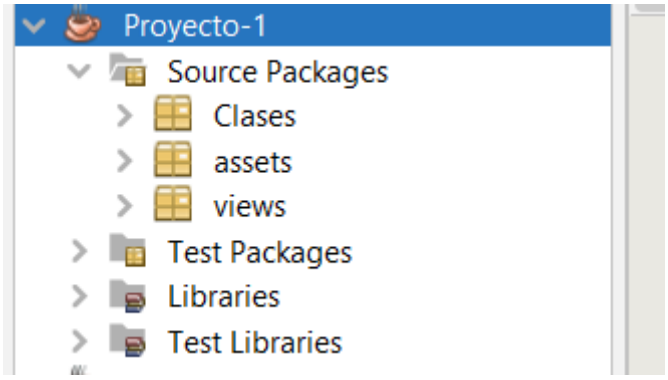
Nombre:Alvaro Gabriel Ramirez Alvarez
Carnet: 202112674

Proyecto 1
Lab.IPC

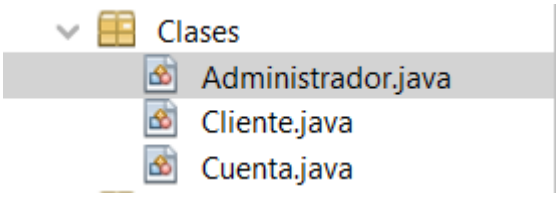
MANUAL TÉCNICO

En este manual presenta el desarrollo y la lógica del siguiente código desde el punto de vista del programador.

Lo primero que se encuentra es la distribución de los paquetes del proyecto.



Ubicados en "src"



En el paquete "Clases" se encuentran las clases del proyecto que ayudan a identificar las entidades sobre las que se van a trabajar.

Lo primero que se ve en la clase es el paquete que se está usando que en este caso es "Clases", Luego la creación de la clase Cliente con sus respectivos parámetros,
A partir de la línea no. 13 se encuentra el constructor.

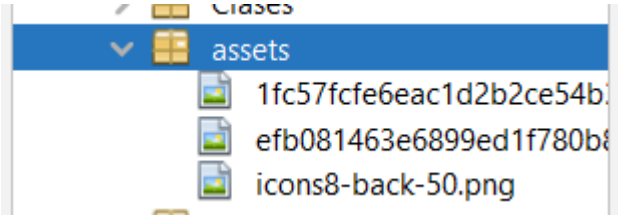
```
5   package Clases;
6
7   public class Cliente {
8       public String cui;
9       public String nombre;
10      public String apellido;
11      public Cuenta[] cuentasAsociadas;
12
13      public Cliente(String cui, String nombre, String apellido) {
14          this.cui = cui;
15          this.nombre = nombre;
16          this.apellido = apellido;
17           this.cuentasAsociadas = new Cuenta[5];
18      }
```

Luego sus Getter y Setter que me permiten almacenar y mostrar datos

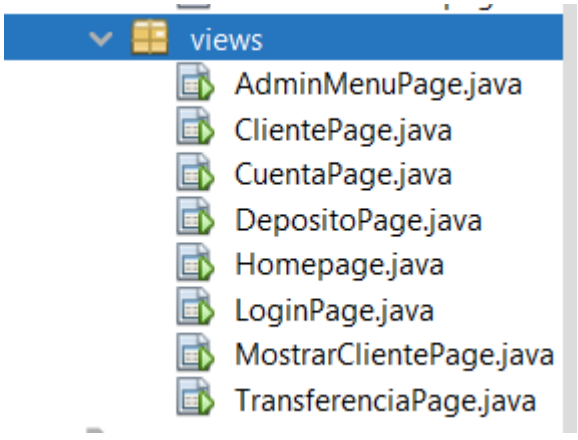
```
20      public String getCui() {
21          return cui;
22      }
23      public void setCui(String cui) {
24          this.cui = cui;
25      }
26      public String getNombre() {
27          return nombre;
28      }
29      public void setNombre(String nombre) {
30          this.nombre = nombre;
31      }
32      public String getApellido() {
33          return apellido;
34      }
35      public void setApellido(String apellido) {
36          this.apellido = apellido;
37      }
38      public Cuenta[] getCuentasAsociadas() {
39          return cuentasAsociadas;
40      }
```

Y así en todas las clases.

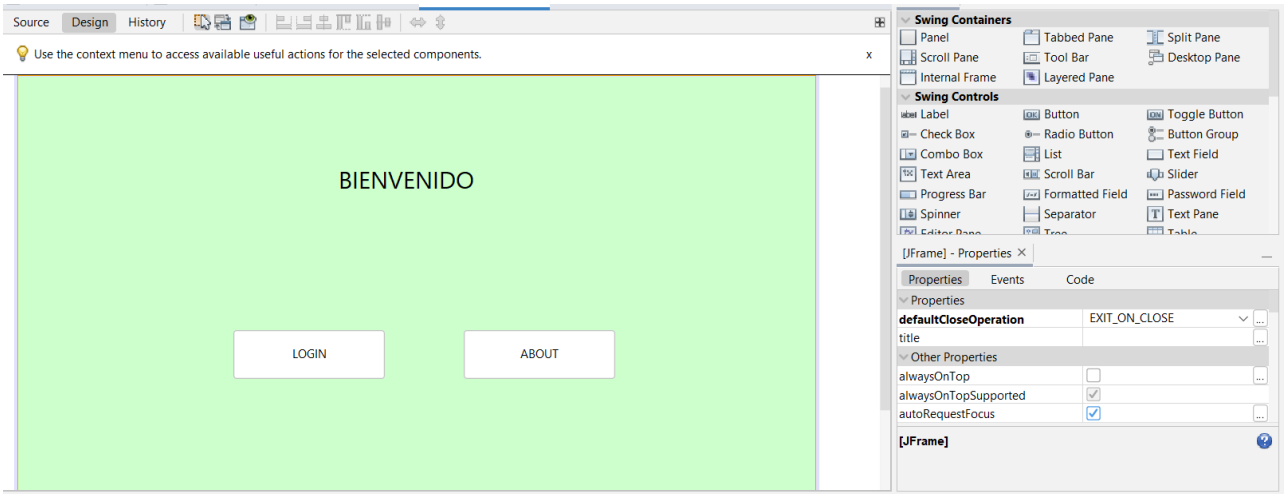
Luego está el paquete de “assets” en donde almacené imágenes para organizar mejor los archivos.



Se encuentra también el paquete de “views” en donde se almacenan todas las vistas del proyecto.



Las vistas son creadas por medio de los containers que importamos de la librería **Swing.** que es toda la parte gráfica del proyecto.



en la parte de  “source” se maneja los controladores de la parte gráfica creando las funciones.


```
10      *
11      * @author ACER
12      */
13     public class Homepage extends javax.swing.JFrame {
14
15         /**
16          * Creates new form Homepage
17          */
18         public Homepage() {
19             initComponents();
20              setLocationRelativeTo(null);
21         }
22
```

también las declaraciones de las variables de cada container, para la manipulación de ellos.

```
146         // Variables declaration - do not modify
147         private javax.swing.JButton btnAbout;
148         private javax.swing.JButton btnLogin;
149         private javax.swing.JLabel jLabel2;
150         private javax.swing.JPanel jPanel1;
151         // End of variables declaration
```

El botón “ABOUT” envía un mensaje de los datos: “ALVARO GABRIEL RAMIREZ ALVAREZ 202112674”

```
96      private void btnAboutActionPerformed(java.awt.event.ActionEvent evt) {
97          // TODO add your handling code here:
98          JOptionPane.showMessageDialog(this,"ALVARO GABRIEL RAMIREZ ALVAREZ \n 202112674");
99
100
101     }
```

En la parte de login se utilizó una verificación de que estuvieran llenos los contenedores de texto para poder comparar los datos.

```
147     if(txtUsuario.getText().isEmpty()&&txtPassword.getText().isEmpty()){
148         JOptionPane.showMessageDialog(this,"Ingrese Usuario y contraseña");
149     }else{
150
```

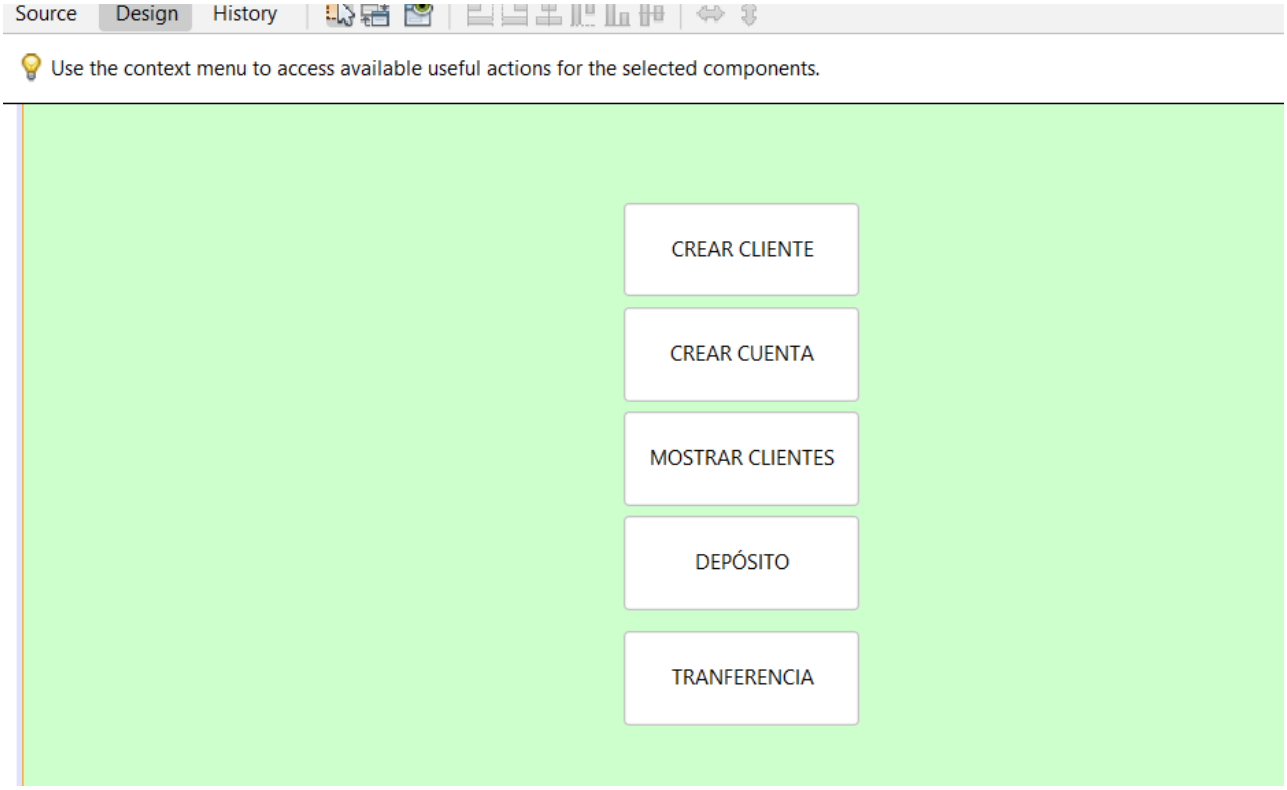
si se llenan los campos se procede a verificar la función de logearse, de lo contrario envía un *JOptionPane.showMessageDialog* para notificar que los datos ingresados son erróneos.

```
152     if(Administradores[i] != null){
153         if(Administradores[i].getUserAdminstrador().equals(txtUsuario.getText()) && Administradores[i].getPassword().equals(txtPassword.getText())){
154           // JOptionPane.showMessageDialog(this,"HOLA!");
155           AdminMenuPage adminMenu =new AdminMenuPage();
156           adminMenu.setVisible(true);
157           this.setVisible(false);
158         }else{
159             JOptionPane.showMessageDialog(this,"Datos Erroneos","Alerta", JOptionPane.OK_OPTION);
160         }
161     }
162 }
```

al botón de “REGRESAR” se instancia la vista de “HomePage” para que regrese de ventana.

```
167     private void btnBackActionPerformed(java.awt.event.ActionEvent evt) {
168         // TODO add your handling code here:
169         Homepage home = new Homepage();
170         home.setVisible(true);
171         this.setVisible(false);
172     }
```

Si los datos del Administrador son correctos se dirige a la vista “AdminMenuPage”, que es el menú del administrador y sus opciones. PRACTICAMENTE SOLO SON BOTONES QUE REDIRIGEN A OTRAS VISTAS



En “ClientePage” se empieza con el uso de un **for** para la creación de clientes que puede hacer el administrador.

Extrayendo los datos con un *Txt.getText()*;

El for hace la condición de que crea un cliente si este no ha llegado al límite que se pueden crear que en este caso son 5 clientes.

```
168         for (int i = 0; i < listaClientes.length; i++) {
169             if (listaClientes[i] == null) {
170                 listaClientes[i] = nuevoCliente;
171                 op = true;
172                 //System.out.println(nuevoCliente.toString());
173                 JOptionPane.showMessageDialog(null, "Cliente creado existosamente", "Información", JOptionPane.INFORMATION_MESSAGE);
174                 break;
175             } else {
176                 if (listaClientes[i].cui.equals(txtCUI.getText())) {
                        JOptionPane.showMessageDialog(null, "No se pueden crear clientes con CUI duplicados. "
178                             + "\n El CUI ingresado ya existe en el sistema.", "Información", JOptionPane.WARNING_MESSAGE);
179                 op = true;
180                 break;
181                 }
182             }
183         }
```

Si se pasa de los 5 clientes saldrá una alerta que no permita la creación de más clientes

```
if (op == false) {
    JOptionPane.showMessageDialog(null, "No es posible crear más clientes", "Alerta", JOptionPane.WARNING_MESSAGE);
}
```

Al igual si se crean 2 clientes con el mismo CUI, este no los dejará.

```
if (listaClientes[i].cui.equals(txtCUI.getText())) {
    JOptionPane.showMessageDialog(null, "No se pueden crear clientes con CUI duplicados. "
            + "\n El CUI ingresado ya existe en el sistema.", "Información", JOptionPane.WARNING_MESSAGE);
    op = true;
    break;
}
```

En la vista “CuentaPage” se cuenta con la creación y asignación de cuentas para cada cliente.

Como se encuentra escrita en la clase “Cliente”

```
11          public Cuenta[] cuentasAsociadas;
```

Primero se asginan los valores de cada cliente en un *ComboBox* .

```
141     public void ImprimirInfo(){
142       ClientePage clientePage = new ClientePage();
143
144         Cliente[] data = clientePage.listaClientes;
145
146
147         for (int i = 0; i < data.length; i++) {
148             if (data[i] != null) {
149                 cbCliente.addItem(data[i].cui+" - "+data[i].nombre +" - "+data[i].apellido);
150             }
151         }
152     }
```

En el botón de agregar se escribe la función de asignar una cuenta al cliente que se selecciona en el *comboBox* .

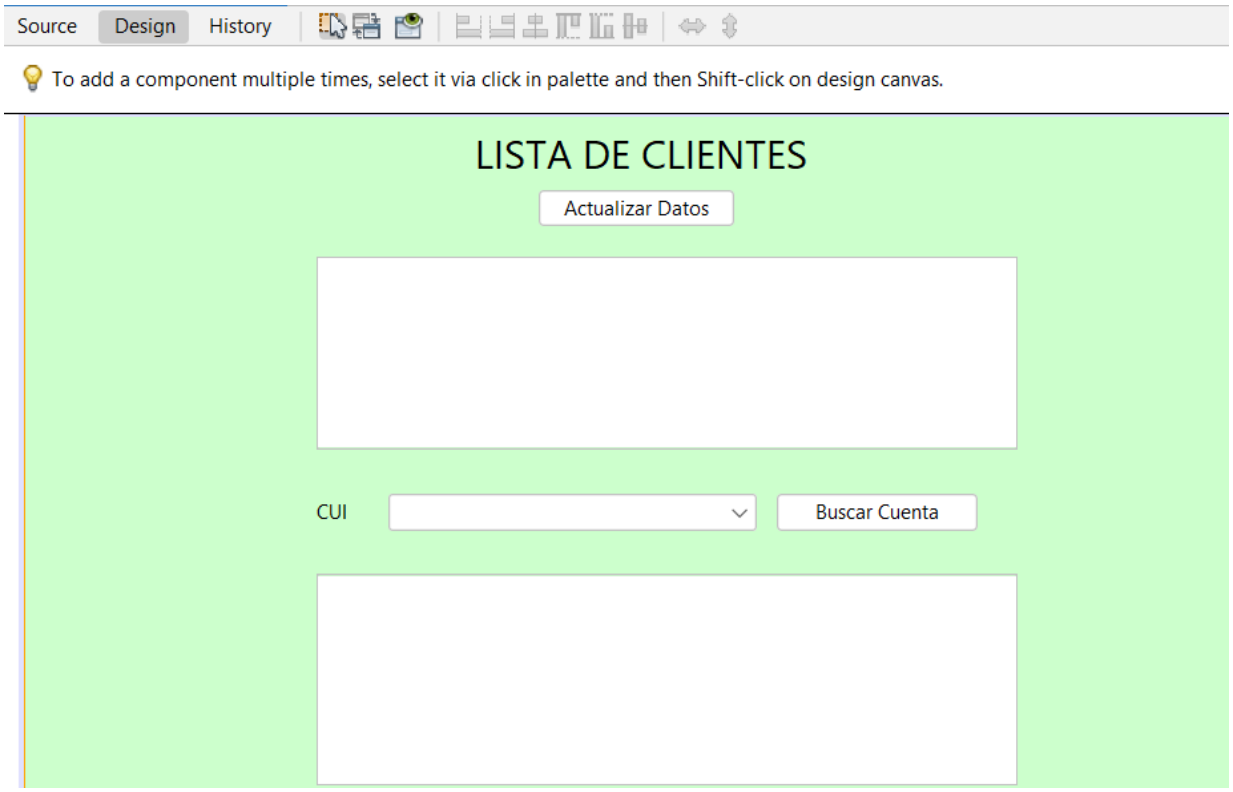
Inicializándola con saldo: 0.0 y con un ID autoincrementable.

```
    private void btnAgregarCuentaActionPerformed(java.awt.event.ActionEvent evt) {
170         ClientePage clientePage = new ClientePage();
171         Cliente[] data = clientePage.listaClientes;
172         boolean op = false;
173         int valorElegido = cbCliente.getSelectedIndex();
174         System.out.println(data[valorElegido].cuentasAsociadas.length + "++++++++++++++++++++++++++++++");
175
176         for (int i = 0; i < data[valorElegido].cuentasAsociadas.length; i++) {
177
178             if (data[valorElegido].cuentasAsociadas[i] == null) {
179
180                 for (int j = 0; j < data[valorElegido].cuentasAsociadas.length; j++) {
181
182                     if (data[valorElegido].cuentasAsociadas[j] == null) {
183
184                         data[valorElegido].cuentasAsociadas[j] = new Cuenta(j, data[valorElegido].cui, 0.0);
185                         op = true;
186                         JOptionPane.showMessageDialog(null, "Cuenta creado existosamente a "
187                 + data[valorElegido].nombre.toUpperCase(), "Información", JOptionPane.INFORMATION_MESSAGE);
188                         break;
189                     }
190                 }
191                 break;
```

este tampoco permite la creación de más de 5 cuentas por cliente.

```
if (op == false) {
    JOptionPane.showMessageDialog(null, "No es posible crear más Cuentas", "Alerta", JOptionPane.WARNING_MESSAGE);
}
```

En la vista de Listado de Clientes podemos ver dos tablas y una búsqueda por medio de un *comboBox*.



Se hace uso de un for para que se llenen los datos de la primera tabla. Con la instancia de los datos llamados desde la clase de “ClientePage”.

```
164     private void btnDatosActionPerformed(java.awt.event.ActionEvent evt) {
165
166         ClientePage clientePage = new ClientePage();
167         Cliente[] data = clientePage.listaClientes;
168         boolean op=false;
169         String matriz[][] = new String[data.length][3];
170         for(int i=0; i<data.length;i++){
171             if(data[i] != null){
172                 matriz[i][0] = data[i].getCui();
173                 matriz[i][1] = data[i].getNombre();
174                 matriz[i][2] = data[i].getApellido();
175                 op=true;
176              }
177         }
178         if(op==false){
179         JOptionPane.showMessageDialog(null, "No hay clientes Creados", "Información", JOptionPane.INFORMATION_MESSAGE);
180         }
```

En la siguiente vista se crean 1 comboBox para la asignación un monto de dinero a una cuenta.

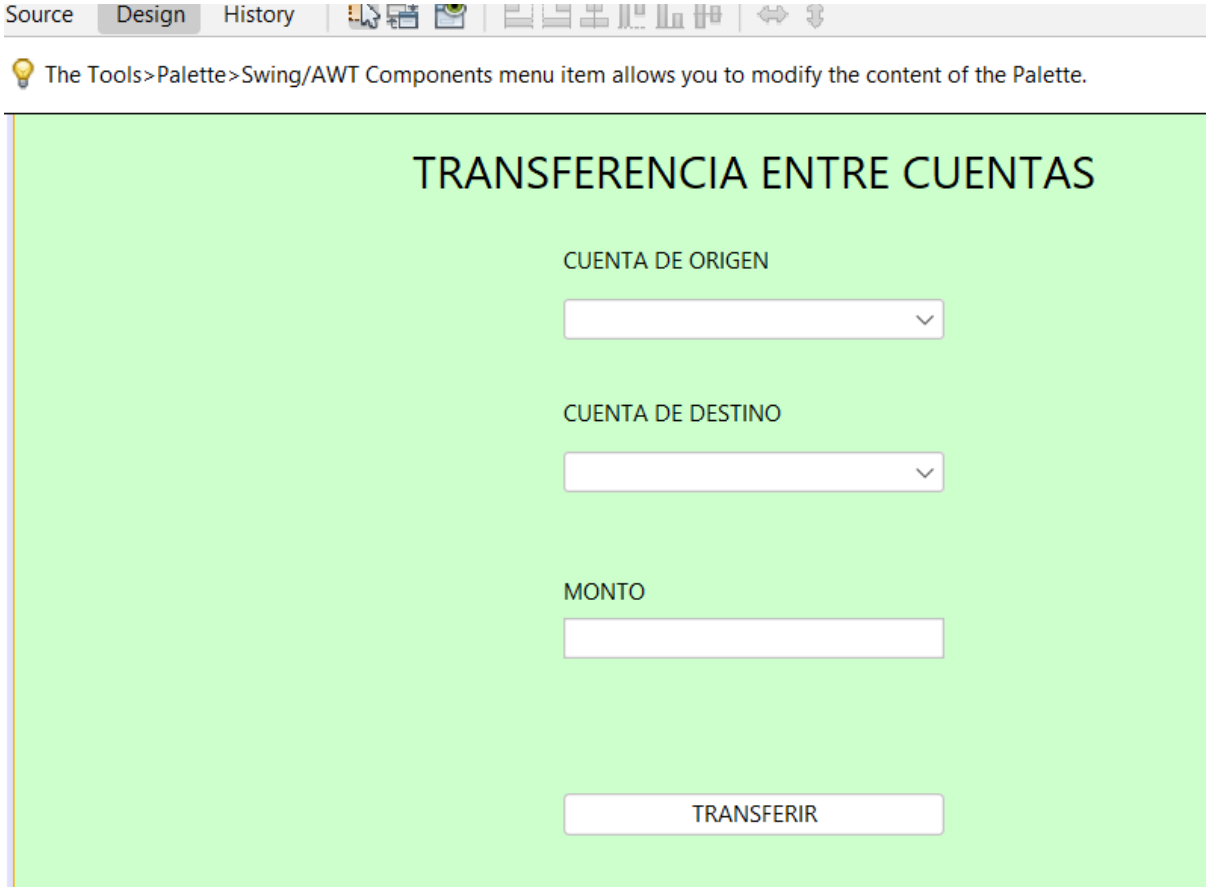
Primero se busca los valores que por medio del combobox luego se sabe que cuenta se le debe aumentar el saldo.
luego la condición de que si menor o igual a cero no dejará realizar ningún depósito.

```
    private void btnDepositarActionPerformed(java.awt.event.ActionEvent evt) {
134     ClientePage clientePage = new ClientePage();
        Cliente[] data = clientePage.listaClientes;
136
137     int valorCombo = cbCuenta.getSelectedIndex();
138     String cuiCliente = data[valorCombo].cuentasAsociadas[valorCombo].dpiCliente;
139     int idCuenta = data[valorCombo].cuentasAsociadas[valorCombo].idCuenta;
140     double monto = Double.parseDouble(txtMonto.getText());
141     if (monto<=0) {
142         JOptionPane.showMessageDialog(null, "El monto debe ser mayor a 0", "Información", JOptionPane.WARNING_MESSAGE);
143
```

Con los siguiente for se hace una doble verificación la creación del depósito por medio de una alerta.

```
            for (int i = 0; i < data.length; i++) {
146             if (data[i] != null) {
                    if (data[i].cui == cuiCliente) {
                        for (int j = 0; j < data[i].cuentasAsociadas.length; j++) {
149                         if (data[i].cuentasAsociadas[j] != null) {
150                             if (data[i].cuentasAsociadas[j].idCuenta == idCuenta) {
151                                 data[i].cuentasAsociadas[j].saldo += monto;
152                                 JOptionPane.showMessageDialog(null, "Deposito Realizado Exitosamente",
153                                         "Información", JOptionPane.INFORMATION_MESSAGE);
154                             }
155                         }
156                     }
157                 }
158             }
```

En la transferencia entre cuentas se hace uso de dos *comboBox* para especificar la cuenta de destino y la de origen.



Se cargan los datos a los *comboBox*

```
193  public void ImprimirInfo() {
194          ClientePage clientePage = new ClientePage();
             Cliente[] data = clientePage.listaClientes;
196
             for (int i = 0; i < data.length; i++) {
198              if (data[i] != null) {
                     for (int j = 0; j < data[i].cuentasAsociadas.length; j++) {
200                      if (data[i].cuentasAsociadas[j] != null) {
201                          data[i].cuentasAsociadas[j].ImprimirCuenta();
202                          cbOrigen.addItem(data[i].cuentasAsociadas[j].idCuenta + " -  Cuenta de " + data[i].nombre);
203                          cbDestino.addItem(data[i].cuentasAsociadas[j].idCuenta + " -  Cuenta de " + data[i].nombre);
204                      }
205                  }
206              }
207          }
208
209      }
```

Se utilizan dos for para el uso de crédito y débito de las cuentas

```
             for (int i = 0; i < data.length; i++) {
155              if (data[i] != null) {
                     if (data[i].cui == cuiClienteCredito) {
                         for (int j = 0; j < data[i].cuentasAsociadas.length; j++) {
158                          if (data[i].cuentasAsociadas[j] != null) {
159                              if (data[i].cuentasAsociadas[j].idCuenta == idCuentaOrigen) {
160                                  data[i].cuentasAsociadas[j].saldo += monto;
161                              }
162                          }
163                      }
164                  }
165              }
166          }
             for (int i = 0; i < data.length; i++) {
168              if (data[i] != null) {
                     if (data[i].cui == cuiDebito) {
                         for (int j = 0; j < data[i].cuentasAsociadas.length; j++) {
171                          if (data[i].cuentasAsociadas[j] != null) {
172                              if (data[i].cuentasAsociadas[j].idCuenta == idCuentaDestino) {
173                                  data[i].cuentasAsociadas[j].saldo -= monto;
174                                  JOptionPane.showMessageDialog(null, "Transferencia Realizada Exitosamente", "Información", JOptionPane.INFORMATION_
175                              }
```