

**Universidad Nacional de Ingeniería**  
**Área de conocimiento de la Tecnología de la información y comunicación**  
**Ingeniería de sistemas**



**Mejoras del proyecto elaborado en clase.**

**Integrantes:** Katherine Nahemi Acevedo Umaña  
Melany Victoria Téllez Rodríguez  
Lilliam Maria Tapia Pérez  
Bianca Danelia Hernandez Molina  
Jacqueline Dayana Carcache Payan  
Heylin de los Angeles Pineda Siu.

**Asignatura:** Programación 2.

**URL de git:** <https://github.com/bianca-hernandez05/Banco2t1-Mejorado.git>

**Fecha:** Sábado 24 de mayo del 2025.

**Docente:** Ing. Abel Edmundo Marin Reyes.

## Programa después de algunas mejoras

Katherine

### ▢ Carpeta Data

#### ▢ Código

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;
using BancoSimple2T1.Models;

namespace BancoSimple2T1.Data
{
    6 referencias
    public class BancoSimpleContext : DbContext
    {
        //Aqui se llaman a cada una de las clases que estan dentro de la carpeta Models
        //las cuales representan cada una de las tablas que estan en la base de datos.

        3 referencias
        public DbSet <Cliente> Cliente { get; set; }
        7 referencias
        public DbSet <Cuenta> Cuenta { get; set;}
        2 referencias
        public DbSet <Transaccion> Transacciones { get; set;}

        0 referencias
        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            optionsBuilder.UseSqlServer(@"Server = DESKTOP-QHPU8CM\SQLEXPRESS; database =BancoSimple2T1; trusted_Connection = true; trustse

        }

        //Definicion de filtro global
        0 referencias
        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<Cuenta>().HasQueryFilter ( c => c.Activa);
        }
    }
}
```

## ▢ Carpeta Models

### Clase Cliente

#### ▢ Código

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace BancoSimple2T1.Models
{
    4 referencias
    public class Cliente
    {
        //Se agregan cada uno de los atributos de la clase Cliente
        //los cuales son los mismos atributos de la tabla Cliente de la base de datos.

        0 referencias
        public int ClienteId { get; set; }
        5 referencias
        public string Nombre { get; set; }
        1 referencia
        public string Identificacion { get; set; }
        0 referencias
        public List<Cuenta> Cuentas { get; set; } = new List<Cuenta>();
    }
}
```

## Clase Cuenta

□ código

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace BancoSimple2T1.Models
{
    6 referencias
    public class Cuenta
    {
        //Se agregan cada uno de los atributos de la clase Cuenta
        //los cuales son los mismos atributos de la tabla Cuenta de la base de datos.

        5 referencias
        public int CuentaId { get; set; }
        4 referencias
        public string NumeroCuenta { get; set; }
        6 referencias
        public decimal Saldo { get; set; }
        7 referencias
        public bool Activa { get; set; } = true;
        2 referencias
        public int ClienteId { get; set; }
        6 referencias
        public Cliente cliente { get; set; }
    }
}
```

## Clase Transaccion

## ▣ código

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace BancoSimple2T1.Models
{
    2 referencias
    public class Transaccion
    {
        //Se agregan cada uno de los atributos de la clase Transaccion
        //los cuales son los mismos atributos de la tabla Transacciones de la base de datos.

        0 referencias
        public int TransaccionId { get; set; }
        1 referencia
        public decimal Monto { get; set; }
        1 referencia
        public DateTime Fecha { get; set; } = DateTime.Now;
        1 referencia
        public string Descripcion { get; set; }
        1 referencia
        public int ? CuentaOrigenId { get; set; }
        1 referencia
        public int ? CuentaDestinoId { get; set; }
    }
}
```

## ▣ Formulario Form1

## ▣ Diseño

The screenshot displays a Windows application window titled "Form1". The interface is divided into two main sections: "Cliente" on the left and "Cuenta" on the right. In the "Cliente" section, there is a "Buscar Cliente" button followed by a text input field, and a large empty rectangular area below it. In the "Cuenta" section, there is a large empty rectangular area. Below the "Cuenta" area is an "AgregarCuenta" button. At the bottom of the window, there are three buttons: "AgregarCliente" on the left, and "Transferencia" (green), "Ver Transferencia" (purple), and "Desactivar Cuenta" (red) on the right. The window has standard Windows window controls (minimize, maximize, close) in the top right corner.

## ▣ Código Form1.cs

```

using BancoSimple2T1.Data;
using BancoSimple2T1.Models;
using Microsoft.EntityFrameworkCore;
namespace BancoSimple2T1
{
    3 referencias
    public partial class Form1 : Form
    {
        //Creamos una instancia de la clase BancoSimpleContext
        //para tener acceso a cada una de las clases (Tablas)

        private BancoSimpleContext _db = new BancoSimpleContext();
        1 referencia
        public Form1()
        {
            InitializeComponent();
            CargarInformacion();
        }
    }
}

```

```

//Metodo para cargar la informacion
5 referencias
private void CargarInformacion()
{
    dgvClientes.DataSource = _db.Cliente.ToList();
    var cuenta = _db.Cuenta.
        Include(c => c.cliente).Where(c => c.Activa).
        Select(c => new
        {
            c.CuentaId,
            c.NumeroCuenta,
            c.Saldo,
            NombreCliente = c.cliente.Nombre,
            c.Activa,
            c.ClienteId
        }).ToList();
    dgvCuentas.DataSource = cuenta;
}
}

```



```
//Este boton sirve para agregar a cada uno de los clientes
1 referencia
private void btnAgregarCliente_Click(object sender, EventArgs e)
{
    var form = new AgregarClienteForm();
    if (form.ShowDialog() == DialogResult.OK)
    {
        _db.Cliente.Add(form.NuevoCliente);
        _db.SaveChanges();
        CargarInformacion();
    }
}
```

```
//Este boton sirve para agregar las cuentas de cada cliente
1 referencia
private void btnAgregarCuenta_Click(object sender, EventArgs e)
{
    if (dgvClientes.SelectedRows.Count == 0)
    {
        MessageBox.Show("Seleccione un cliente primero");
        return;
    }
    var clienteId = (int)dgvClientes.SelectedRows[0].Cells["ClienteId"].Value;
    var form = new AgregarCuentaForm(clienteId);
    if (form.ShowDialog() == DialogResult.OK)
    {
        _db.Cuenta.Add(form.NuevaCuenta);
        _db.SaveChanges();
        CargarInformacion();
    }
}
```



```

//Este metodo sirve para realizar las transacciones (de dinero)
//de la cuenta de un cliente a otra cuenta de otro cliente
1 referencia
private void RealizarTransaccion(int cuentaOrigenId, int cuentaDestinoId, decimal monto)
{
    using var transferencia = _db.Database.BeginTransaction(System.Data.IsolationLevel.Serializable)
    try
    {
        //Filtro y ordenacion
        var cuentaOrigen = _db.Cuenta.FirstOrDefault(c => c.CuentaId == cuentaOrigenId);
        var cuentaDestino = _db.Cuenta.FirstOrDefault(c => c.CuentaId == cuentaDestinoId);

        //Aqui se verifica si la cantidad del monnto de x cuenta es la adecuada
        //para poder transferir a otra cuenta (monto suficiente)

        if (cuentaOrigen == null || cuentaDestino == null)
            throw new Exception("Una o ambas cuentas no existen");

        if (!cuentaOrigen.Activa || !cuentaDestino.Activa)
            throw new Exception("Una o ambas cuentas están inactivas");
        if (cuentaOrigen.Saldo < monto)
            throw new Exception("Saldo Insuficiente en la cuenta Origen");

        cuentaOrigen.Saldo -= monto;
        cuentaDestino.Saldo += monto;
    }
}

```

```

//Aqui se registra la transaccion
_db.Transacciones.Add(new Transaccion
{
    Monto = monto,
    Fecha = DateTime.Now,
    Descripcion = "Transferencia entre cuentas",
    CuentaOrigenId = cuentaOrigenId,
    CuentaDestinoId = cuentaDestinoId
});

_db.SaveChanges();

//Aqui esta completada la Transaccion
transferencia.Commit();
MessageBox.Show("Transferencia realizada");
CargarInformacion();
}
catch (Exception ex)
{
    transferencia.Rollback();

    var inner = ex.InnerException?.Message ?? "No hay InnerException";
    MessageBox.Show($"Error al guardar:\n{ex.Message}\n\nDetalle:\n{inner}");
}
}

```

```

//Este boton sirve para transferir el dinero de una cuenta a otra
//teniendo en cuenta las dos cuentas seleccionadas
1 referencia
private void btnTransferencia_Click(object sender, EventArgs e)
{
    if (dgvCuentas.SelectedRows.Count != 2)
    {
        MessageBox.Show("Seleccione exactamente 2 cuentas");
        return;
    }

    var cuentaOrigenId = (int)dgvCuentas.SelectedRows[1].Cells["CuentaId"].Value;
    var cuentaDestinoId = (int)dgvCuentas.SelectedRows[0].Cells["CuentaId"].Value;

    var form = new TransaccionesForms(cuentaOrigenId, cuentaDestinoId);
    if (form.ShowDialog() == DialogResult.OK)
    {
        RealizarTransaccion(cuentaOrigenId, cuentaDestinoId, form.Monto);
    }
}

```

```

//Este boton ayuda a desactivar la cuenta de un cliente que esta activa
//y que talvez ya no la usara
1 referencia
private void btnDesactivar_Click(object sender, EventArgs e)
{
    if (dgvCuentas.SelectedRows.Count != 1)
    {
        MessageBox.Show("Selecciones solo una cuenta exactamente");
        return;
    }
    else
    {
        var cuentaId = (int)dgvCuentas.SelectedRows[0].Cells["CuentaId"].Value;
        var cuenta = _db.Cuenta.Find(cuentaId);
        cuenta.Activa = false;
        ~~~~~
        _db.SaveChanges();
        CargarInformacion();
    }
}

```

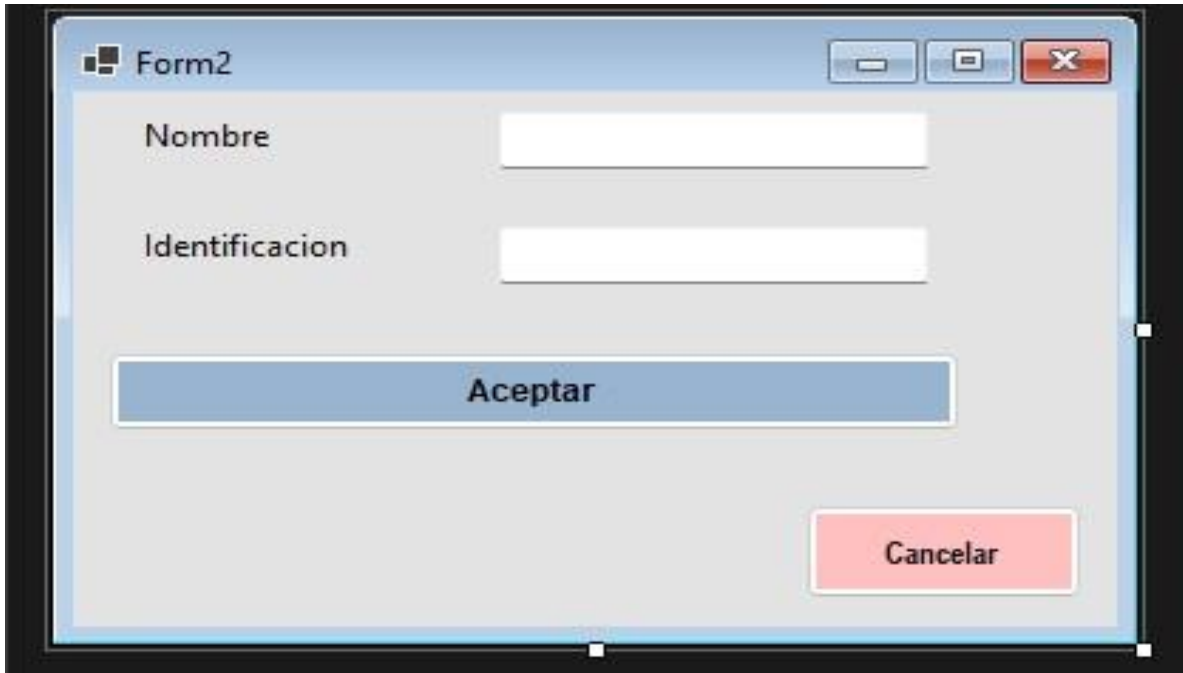
```
//Este boton sirve para buscar el nombre del cliente que se requiere encontrar
//dentro de la tabla de clientes del form1
1 referencia
private void btnBuscarCliente_Click_1(object sender, EventArgs e)
{
    //Busqueda de patrones con like
    var patron = txtBuscarCliente.Text;
    var cliente = _db.Cliente.Where(c => EF.Functions.Like(c.Nombre, $"%{patron}%")).ToList();

    dgvClientes.DataSource = cliente;
}

//Este boton nos ayuda a poder identificar las tranferencias que se han hecho
//durante la ejecucion del programa
1 referencia
private void btnVerTransferencia_Click_1(object sender, EventArgs e)
{
    var form = new VerTransferenciaForms();
    form.ShowDialog();
}
```

## Formulario AgregarClienteForm

### ▮ Diseño



The screenshot shows a Windows Forms application window titled "Form2". Inside the window, there are two text boxes. The first text box is labeled "Nombre" and the second is labeled "Identificacion". Below these text boxes is a large blue button labeled "Aceptar". In the bottom right corner of the window is a red button labeled "Cancelar". The window has standard Windows OS controls (minimize, maximize, close) in the top right corner.

### ▮ código AgregarClienteForm.cs

```
using BancoSimple2T1.Models;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace BancoSimple2T1
{
    3 referencias
    public partial class AgregarClienteForm : Form
    {
        //Aqui llamamos a la clase Cliente para crear un nuevo cliente
        2 referencias
        public Cliente NuevoCliente { get; private set; }

        1 referencia
        public AgregarClienteForm()
        {
            InitializeComponent();
        }
    }
}
```



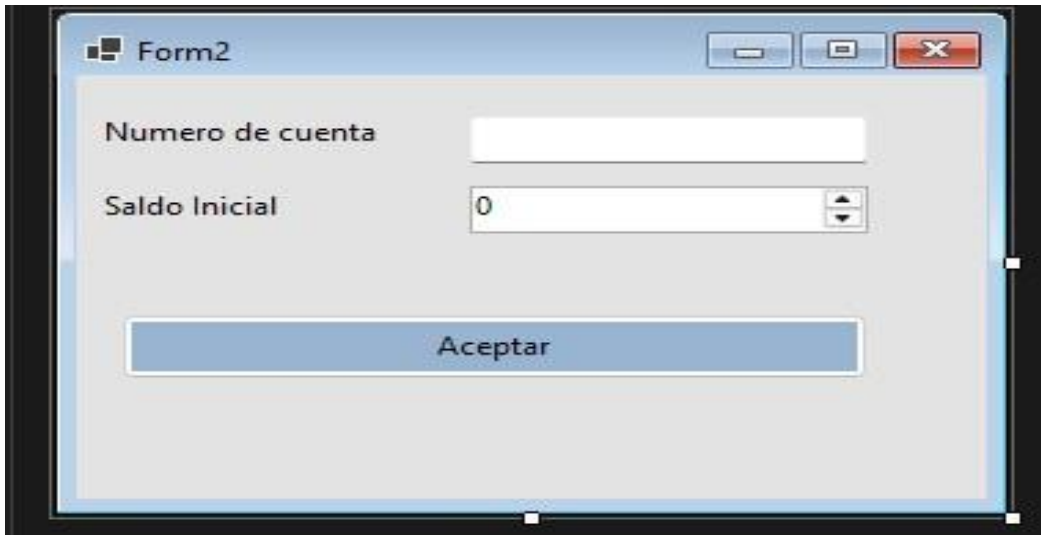
□

```
//Este boton se utiliza para aceptar de que se esta agregando un nuevo cliente
//tanto su nombre y su identificacion
1 referencia
private void btnAceptar_Click(object sender, EventArgs e)
{
    if (string.IsNullOrEmpty(txtNombre.Text) || string.IsNullOrEmpty(txtIdentificacion.Text))
    {
        MessageBox.Show("Todos los campos son necesarios");
        return;
    }
    NuevoCliente = new Cliente
    {
        Nombre = txtNombre.Text,
        Identificacion = txtIdentificacion.Text
    };
    DialogResult = DialogResult.OK;
    Close();
}
```

```
//Este boton se utiliza para cancelar o eliminar
//es decir cuando ya no queremos agregar a x cliente
1 referencia
private void btnCancelar_Click(object sender, EventArgs e)
{
    DialogResult = DialogResult.Cancel;
    Close();
}
```

## Formulario AgregarCuentaForm

### ▢ Diseño



### ▢ código AgregarCuentaForm.cs

```
using Bancosimple2T1.Models;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace BancoSimple2T1
{
    3 referencias
    public partial class AgregarCuentaForm : Form
    {
        //Aqui mandamos a llamar a la clase cuenta para crear una cuenta para un cliente
        2 referencias
        public Cuenta NuevaCuenta { get; private set; }
        private int _clienteId;

        //en este metodo se necesita el id del cliente al que queremos agregarle
        //una nueva cuenta
        1 referencia
        public AgregarCuentaForm(int clienteId)
        {
            InitializeComponent();
            _clienteId = clienteId;
        }
    }
}
```



□

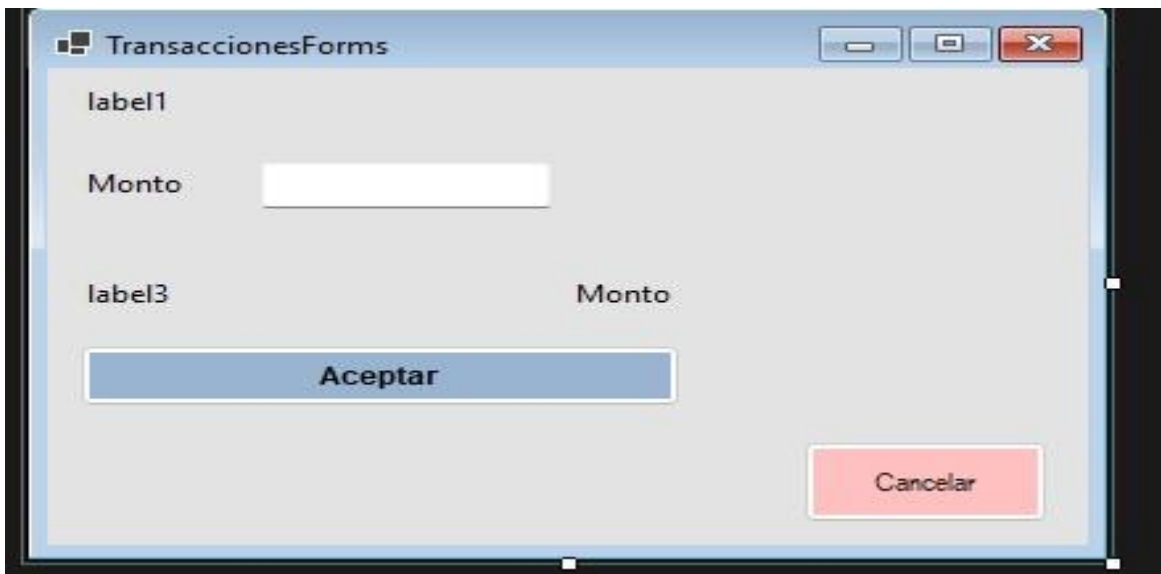
```
//Este boton se utiliza para aceptar de que estamos agregando
//una nueva cuenta a un cliente en especifico
1 referencia
private void btnAceptar_Click(object sender, EventArgs e)
{
    if (string.IsNullOrEmpty(txtNumeroCuenta.Text))
    {
        MessageBox.Show("El numero de cuenta es requerido");
        return;
    }

    NuevaCuenta = new Cuenta
    {
        NumeroCuenta = txtNumeroCuenta.Text,
        Saldo = numSaldoInicial.Value,
        ClienteId = _clienteId,
        Activa = true
    };

    DialogResult = DialogResult.OK;
    Close();
}
```

## □ Formulario TransaccionesForms

### □ Diseño



The screenshot shows a Windows application window titled "TransaccionesForms". The window contains a form with the following elements:

- A label "label1" at the top left.
- A label "Monto" followed by a text input field.
- A label "label3" and another label "Monto" below the input field.
- A blue button labeled "Aceptar" (Accept) in the center.
- A red button labeled "Cancelar" (Cancel) at the bottom right.

## Código TransaccionesForms.cs

```
using BancoSimple2T1.Data;
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace BancoSimple2T1
{
    3 referencias
    public partial class TransaccionesForms : Form
    {
        2 referencias
        public decimal Monto { get; private set; }

        //Se necesita 2 variables la cuenta que va a transferir (dinero) y la
        // cuenta que va a obtener el dinero

        private int _cuentaOrigenId;
        private int _cuentaDestinoId;

        //Llamamos a la clase BancoSimpleContext y creamos un objeto para comunicacion (db)
        private BancoSimpleContext db;

        //Llamamos a la clase BancoSimpleContext y creamos un objeto para comunicacion (db)
        private BancoSimpleContext db;

        //Para la ejecucion de este formulario se necesita la cuenta a transferir y la que
        //obtendra la transferencia
        1 referencia
        public TransaccionesForms(int cuentaOrigenId, int cuentaDestinoId)
        {
            InitializeComponent();
            _cuentaOrigenId = cuentaOrigenId;
            _cuentaDestinoId = cuentaDestinoId;

            db = new BancoSimpleContext();
            CargarInformacionCuenta();
        }
    }
}
```

```

//Este metodo nos ayuda a cargar la informacion de la transaccion
//teniendo en cuenta los nombres de las personas y el monto a transferir y a recibir
1 referencia
private void CargarInformacionCuenta()
{
    var cuentaOrigen = db.Cuenta.
        Include(c => c.cliente).
        First(c => c.CuentaId == _cuentaOrigenId);

    var cuentaDestino = db.Cuenta.
        Include(c => c.cliente).
        First(c => c.CuentaId == _cuentaDestinoId);

    lblOrigen.Text = $"Nombre: {cuentaOrigen.cliente.Nombre} cuenta {cuentaOrigen.NumeroCuenta}";
    lblDestino.Text = $"Nombre: {cuentaDestino.cliente.Nombre} cuenta {cuentaDestino.NumeroCuenta}";
    lblDisponible.Text = $"Saldo Disponible : {cuentaOrigen.Saldo:c}";
}

```

```

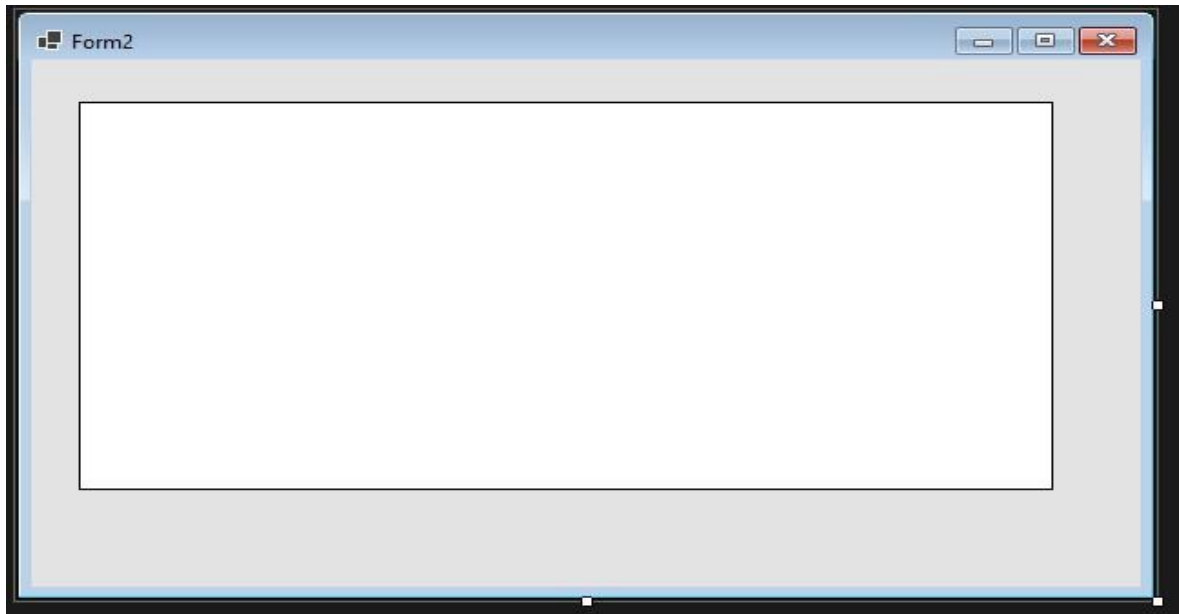
//Este boton nos ayuda para aceptar la transaccion que se quiere realizar
1 referencia
private void btnAceptar_Click(object sender, EventArgs e)
{
    if (decimal.TryParse(txtSaldo.Text, out decimal monto) && monto > 0)
    {
        Monto = monto;
        DialogResult = DialogResult.OK;
        Close();
    }
    else
    {
        MessageBox.Show("Ingrese un monto mayor a 0");
    }
}

//Este boton nos ayuda a cancelar la transaccion que ya no se quiere realizar
1 referencia
private void btnCancelar_Click(object sender, EventArgs e)
{
    DialogResult = DialogResult.Cancel;
    Close();
}

```

## Formulario VerTransferenciaForms

### ▮ Diseño



### ▮ Código VerTransferenciaForms.cs

```
using BancoSimple2T1.Data;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace BancoSimple2T1
{
    3 referencias
    public partial class VerTransferenciaForms : Form
    {
        //Aqui mandamos a llamar a la clase BancoSimpleContext mediante
        //una instancia para tener conexion mediante el objeto (con)
        private BancoSimpleContext con = new BancoSimpleContext();
        1 referencia
        public VerTransferenciaForms()
        {
            InitializeComponent();
            CargarTransferencias();
        }
    }
}
```

```
//Este metodo nos ayuda a ver las transferencias que se hicieron durante la ejecucion
//del programa
1 referencia
private void CargarTransferencias()
{
    dgvTransferencias.DataSource = con.Transacciones.ToList();
}
```

Durante la revisión del programa sBancoSimple2T1, se realizaron las primeras mejoras importantes para optimizar el funcionamiento y facilitar la comprensión del usuario. En primer lugar, se corrigió y se cambió los nombres de las variables y métodos que estaban mal escritos o no eran representativos, utilizando nombres más claros.

También se realizó comentarios para uno de los métodos y funciones del código para que su propósito fuera más entendible. Además, en los formularios con diseño, se les mejoro la estructura visual, ajustando nombres, colores y tipos de letra para ofrecer una mejor experiencia al usuario; como también, se realizaron mejoras en las clases que están dentro de las carpetas Data y Models, organizando mejor el código y asegurando una mayor coherencia en todo el proyecto.

Durante cada mejora se estará subiendo un documento con los cambios realizados a partir de este programa.

Jacqueline

Bueno primero observamos que en el código del Form1 hay varios duplicados, para evitar esto vamos a crear 4 métodos nuevos en esta clase. Simplificamos la instancia.

```
// Instancia de BancoSimpleContext para acceder a la base de datos con _db.  
private readonly BancoSimpleContext _db = new();  
1 referencia
```

```
// Método para validar la selección en un DataGridView  
3 referencias  
private bool ValidarSeleccion(DataGridView dgv, int cantidadEsperada, string mensajeError)  
{  
    if (dgv.SelectedRows.Count != cantidadEsperada)  
    {  
        MessageBox.Show(mensajeError);  
        return false;  
    }  
    return true;  
}
```

```
// >>>> CAMBIO: Sobrecarga método ObtenerIdSeleccionado para fila específica  
2 referencias  
private int ObtenerIdSeleccionado(DataGridView dgv, string nombreColumna, int indexFila)  
{  
    return (int)dgv.SelectedRows[indexFila].Cells[nombreColumna].Value;  
}
```

```
// Método para guardar cambios y recargar datos  
2 referencias  
private void GuardarYCargar()  
{  
    _db.SaveChanges();  
    CargarInformacion();  
}
```

```
// Método para obtener el Id seleccionado en un DataGridView, columna dada  
1 referencia  
private int ObtenerIdSeleccionado(DataGridView dgv, string nombreColumna)  
{  
    return (int)dgv.SelectedRows[0].Cells[nombreColumna].Value;  
}
```



□

Ya aplicados los cambios:

```
//Este boton ayuda a desactivar la cuenta de un cliente que esta activa
//y que talvez ya no la usara
1 referencia
private void btnDesactivar_Click(object sender, EventArgs e)
{
    // CAMBIO: validar selección con método
    if (!ValidarSeleccion(dgvCuentas, 1, "Seleccione solo una cuenta exactamente"))
        return;

    else
    {
        var cuentaId = (int)dgvCuentas.SelectedRows[0].Cells["CuentaId"].Value;
        var cuenta = _db.Cuenta.Find(cuentaId);
        cuenta.Activa = false;
        // CAMBIO: guardar y recargar con método
        GuardarYCargar();
    }
}
```

```
//Este boton sirve para transferir el dinero de una cuenta a otra
//teniendo en cuenta las dos cuentas seleccionadas
1 referencia
private void btnTransferencia_Click(object sender, EventArgs e)
{
    // CAMBIO: validar selección con método
    if (!ValidarSeleccion(dgvCuentas, 2, "Seleccione exactamente 2 cuentas"))
        return;

    // CAMBIO: uso método sobrecargado para fila 1 y fila 0
    var cuentaOrigenId = ObtenerIdSeleccionado(dgvCuentas, "CuentaId", indexFila: 1);
    var cuentaDestinoId = ObtenerIdSeleccionado(dgvCuentas, "CuentaId", indexFila: 0);

    var form = new TransaccionesForms(cuentaOrigenId, cuentaDestinoId);
    if (form.ShowDialog() == DialogResult.OK)
    {
        RealizarTransaccion(cuentaOrigenId, cuentaDestinoId, form.Monto);
    }
}
```

```
//Este boton sirve para agregar las cuentas de cada cliente
1 referencia
private void btnAgregarCuenta_Click(object sender, EventArgs e)
{
    // CAMBIO: validación usando método nuevo
    if (!ValidarSeleccion(dgvClientes, 1, "Seleccione un cliente primero"))
        return;

    // CAMBIO: obtener id usando método nuevo
    var clienteId = ObtenerIdSeleccionado(dgvClientes, "ClienteId");
    var form = new AgregarCuentaForm(clienteId);
    if (form.ShowDialog() == DialogResult.OK)
    {
        _db.Cuenta.Add(form.NuevaCuenta);
        // CAMBIO: reemplazo
        GuardarYCargar();
    }
}
```

- En el código de TransaccionesForms creamos un nuevo método.

```
// CAMBIO: metodo para obtener la cuenta con su cliente
2 referencias
private Cuenta ObtenerCuentaConCliente(int cuentaId)
{
    return db.Cuenta.Include(c => c.cliente).First(c => c.CuentaId == cuentaId);
}
```

Lo implementamos de la siguiente forma:

```
//Este metodo nos ayuda a cargar la informacion de la transaccion
//teniendo en cuenta los nombres de las personas y el mosto a transferir y a resivir
1 referencia
private void CargarInformacionCuenta()
{
    // CAMBIO: con el nuevo metodo
    var cuentaOrigen = ObtenerCuentaConCliente(_cuentaOrigenId);
    var cuentaDestino = ObtenerCuentaConCliente(_cuentaDestinoId);

    lblOrigen.Text = $"Nombre: {cuentaOrigen.cliente.Nombre} cuenta {cuentaOrigen.NumeroCuenta}";
    lblDestino.Text = $"Nombre: {cuentaDestino.cliente.Nombre} cuenta {cuentaDestino.NumeroCuenta}";
    lblDisponible.Text = $"Saldo Disponible : {cuentaOrigen.Saldo:c}";
}
```

- En el VerTransferenciaForms simplificamos la instancia, que es la misma que utilizamos en el Form1.

```
// Instancia de BancoSimpleContext para acceder a la base de datos con _db.
private readonly BancoSimpleContext _db = new();
1 referencia
```

```
private void CargarTransferencias()
{
    dgvTransferencias.DataSource = _db.Transacciones.ToList();
}
```

Los cambios realizados, como la creación de métodos reutilizables y la sobrecarga de funciones, no solo eliminaron el código repetido, sino que también mejoraron significativamente la organización, legibilidad y mantenibilidad del proyecto. Al centralizar funciones comunes como la validación de selección o la obtención de IDs desde el DataGridView, se reduce la posibilidad de errores y se facilita el mantenimiento del código. Estos ajustes también permiten una estructura más clara y coherente para futuras mejoras del sistema.