Informe Trabajo Final Programación II
Banco UCC 2024

Universidad Católica de Córdoba
Cátedra B1

Integrantes:
- Atias, Benjamin
- Navarro, Lorenzo Manuel
- Zago, Francisco Antonio

Profesor:
De Lima Tovar, Eduardo Luis

Clases, entidades, atributos y métodos:

Clase Persona
- Atributo Nombre
- Atributo Apellido
- Atributo Dni

Clase Transacción
- Atributo Número_Transacción
- Atributo Número_Cliente_Transacción
- Atributo Monto
- Atributo Tipo_Moneda
- Atributo Tipo_Transacción
- Atributo Dia
- Atributo Mes
- Atributo Año

Clase Cliente (Hereda de Persona)
- Atributo Tipo_Cliente
- Atributo Año_Ingreso
- Atributo Estado
- Atributo Tarjeta

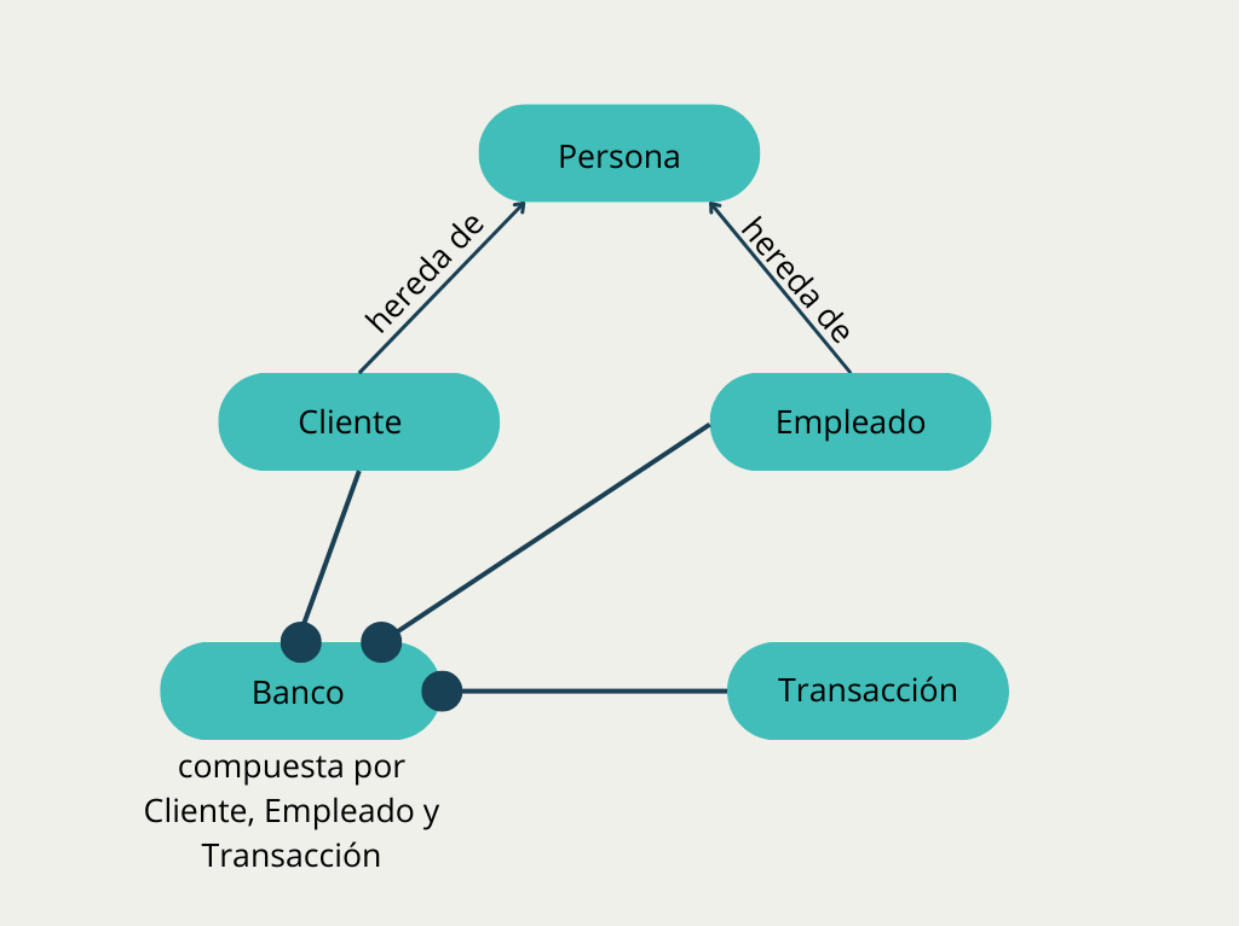
- Método Cambiar_Estado

Clase Empleado (Hereda de Persona)

Clase Banco
- Atributo Clientes[100] (Composición)
- Atributo Empleados[100] (Composición)
- Atributo Transacciones[100] (Composición)
- Atributo Número_De_Clientes
- Atributo Número_De_Transacciones

- Método Cambiar_Estado_Cliente
- Método Agregar_Cliente
- Método Listar_Clientes
- Método Detallar_Clientes
- Método Realizar_Transacción
- Método Listar_Transacciones
- Método Informes_Transacciones_Por_Mes
- Método Dni_Existe

Diagrama de relación entre clases, con herencia y composición:



Librerías usadas y funciones que dependen de esas librerías:

Libreria <cctype>
Se utiliza para comprobar si un carácter pertenece a una categoría específica (como sí es una letra, un dígito, un carácter en blanco, etc.) o para convertir caracteres entre mayúsculas y minúsculas.
La utilizamos en las funciones:
bool esnumero()
bool esNombreValido()
cliente::string get_tipo_clente();
cliente::int get_anio_ingreso();
cliente::bool get_estado();
cliente::string get_tarjeta();

Librería <fstream>
La utilizamos para leer y guardar datos en los archivos .txt
La utilizamos en las funciones:
void banco::listar_clientes()

void banco::realizar_transaccion()

Librería <string>
La utilizamos para ver cuantos caracteres tienen algunas variables string y así hacer comprobaciones.
bool esnumero
cliente::string get_tipo_clente();
cliente::int get_anio_ingreso();
cliente::bool get_estado();
cliente::string get_tarjeta();

Librería <limits>
La usamos para saber datos como el valor máximo y mínimo que puede contener una variable , así como otras propiedades específicas de los tipos de datos.
bool esnumero()
cliente::string get_tipo_clente();
cliente::int get_anio_ingreso();
cliente::bool get_estado();
cliente::string get_tarjeta();

Métodos que usa el programa y descripción:

Metodos set y get:

void set_nombre(string);
    void set_apellido(string);
    void set_dni(int);
    string get_nombre();
    string get_apellido();
    int get_dni();

void set_numero_transaccion(float);
    void set_monto(float);
    void set_numero_cliente_transaccion(int);
    void set_tipo_moneda(string);
    void set_tipo_transaccion(string);
    void set_dia(int);
    void set_mes(int);
    void set_anio(int);

    float get_numero_transaccion();
    float get_monto();
    int get_numero_cliente_transaccion();
    string get_tipo_moneda();

```
    string get_tipo_transaccion();
    int get_dia();
    int get_mes();
    int get_anio();

void set_tipo_cliente(string);
    void set_anio_ingreso(int);
    void set_estado(bool);
    void set_tarjeta(string);

    string get_tipo_clente();
    int get_anio_ingreso();
    bool get_estado();
    string get_tarjeta();

  void set_num_clientes(int);
    int get_num_clientes();
    void set_num_transacciones(int);
    int get_num_transacciones();
```

Estos métodos encapsulan los datos, proporcionando una interfaz controlada para acceder y modificar los atributos de un objeto.

Clase cliente

void cambiar_estado();
Le pedimos al usuario que ingrese un DNI, comprobar que ese DNI corresponda a un cliente y luego le preguntamos si quiere darlo de baja o de alta según el estado en el que esté el cliente solicitado.

Clase banco

void agregar_cliente();
Agregamos a los clientes usando arreglos y los sets y gets pasando con todas las comprobaciones necesarias.

void detallar_clientes();
Mostramos los datos de un cliente solicitado en pantalla.

void listar_clientes();
Guardamos todos los datos de los clientes en un archivo .txt llamado clientes.txt

void realizar_transaccion();

Le pedimos al usuario que proporcione todos los datos necesarios para hacer una transacción, de pasar todas las comprobaciones estos datos se guardarán en un archivo .txt llamado transacciones.txt .

void listar_transacciones();
Solicitamos al usuario un número de cliente, y si este es válido se mostrarán todas las transacciones que este hizo por pantalla.

void informes_transacciones_mes();
Solicitamos un mes y año, luego se mostrarán por pantalla todas las transacciones de ese mes y año por pantalla.

bool dni_existe(int);
Comprobacion para verificar que dos clientes distintos puedan registrarse con el mismo DNI

Código programa:

```
#include <iostream>
#include <cctype>
#include <fstream>
#include <string>
#include <limits>

using namespace std;

// clase persona//
class persona
{

private:
   string nombre;
   string apellido;
   int dni;

public:
   persona();
   persona(string, string, int);
   void set_nombre(string);
   void set_apellido(string);
   void set_dni(int);
   string get_nombre();
   string get_apellido();
   int get_dni();
```

```
};
// clase transaccion//
class transaccion
{
private:
    float numero_transaccion;
    float monto;
    int numero_cliente_transaccion;
    string tipo_moneda;
    string tipo_transaccion;
    int dia;
    int mes;
    int anio;

public:
    transaccion();
    transaccion(float, float, int, string, string, int, int, int);

    void set_numero_transaccion(float);
    void set_monto(float);
    void set_numero_cliente_transaccion(int);
    void set_tipo_moneda(string);
    void set_tipo_transaccion(string);
    void set_dia(int);
    void set_mes(int);
    void set_anio(int);

    float get_numero_transaccion();
    float get_monto();
    int get_numero_cliente_transaccion();
    string get_tipo_moneda();
    string get_tipo_transaccion();
    int get_dia();
    int get_mes();
    int get_anio();
};

// clase cliente, hereda de persona//
class cliente : public persona
{

private:
    string tipo_cliente;
    int anio_ingreso;
```

```
    bool estado;
    string tarjeta;

public:
    cliente();
    cliente(string, string, int, string, int, bool, string);
    void realizar_transaccion();
    void cambiar_estado();

    void set_tipo_cliente(string);
    void set_anio_ingreso(int);
    void set_estado(bool);
    void set_tarjeta(string);

    string get_tipo_clente();
    int get_anio_ingreso();
    bool get_estado();
    string get_tarjeta();
};

class empleado : public persona
{
private:
public:
    empleado();
};

class banco
{
private:
    cliente clientes[100];
    empleado empleados[100];
    transaccion transacciones[100];
    int num_clientes;
    int num_transacciones;

public:
    banco();
    banco(cliente[], empleado[], transaccion[], int, int);
    void set_num_clientes(int);
    int get_num_clientes();
    void set_num_transacciones(int);
    int get_num_transacciones();
    void cambiar_estado_cliente();
```

```
    void agregar_cliente();
    void listar_clientes();
    void detallar_clientes();
    void realizar_transaccion();
    void listar_transacciones();
    void informes_transacciones_mes();
    bool dni_existe(int);
};

#include <iostream>
#include <cctype>
#include <fstream>
#include <string>
#include <limits>
#include "banco.h"
using namespace std;

persona::persona(){

};

persona::persona(string _nombre, string _apellido, int _dni)
{

    nombre = _nombre;
    apellido = _apellido;
    dni = _dni;
};

void persona::set_nombre(string _nombre)
{
    nombre = _nombre;
}
void persona::set_apellido(string _apellido)
{
    apellido = _apellido;
}
void persona::set_dni(int _dni)
{
    dni = _dni;
}
string persona::get_nombre()
{
```

```
    return nombre;
}
string persona::get_apellido()
{
    return apellido;
}
int persona::get_dni()
{
    return dni;
}
// constructores y metodos transacciones//

transaccion::transaccion(){

};
transaccion::transaccion(float _numero_transaccion, float _monto, int _numero_cliente_transaccion, string _tipo_moneda, string _tipo_transaccion, int _dia, int _mes, int _anio)
{
    numero_transaccion = _numero_transaccion;
    monto = _monto;
    numero_cliente_transaccion = _numero_cliente_transaccion;
    tipo_moneda = _tipo_moneda;
    tipo_transaccion = _tipo_transaccion;
    dia = _dia;
    mes = _mes;
    anio = _anio;
};

void transaccion::set_numero_transaccion(float _numero_transaccion)
{
    numero_transaccion = _numero_transaccion;
}
void transaccion::set_monto(float _monto)
{
    monto = _monto;
}
void transaccion::set_numero_cliente_transaccion(int _numero_cliente_transaccion)
{
    numero_cliente_transaccion = _numero_cliente_transaccion;
}
void transaccion::set_tipo_moneda(string _tipo_moneda)
{
    tipo_moneda = _tipo_moneda;
```

```
}
void transaccion::set_tipo_transaccion(string _tipo_transaccion)
{
    tipo_transaccion = _tipo_transaccion;
}
void transaccion::set_dia(int _dia)
{
    dia = _dia;
}
void transaccion::set_mes(int _mes)
{
    mes = _mes;
}
void transaccion::set_anio(int _anio)
{
    anio = _anio;
}
float transaccion::get_numero_transaccion()
{
    return numero_transaccion;
}
float transaccion::get_monto()
{
    return monto;
}
int transaccion::get_numero_cliente_transaccion()
{
    return numero_cliente_transaccion;
}
string transaccion::get_tipo_moneda()
{
    return tipo_moneda;
}
string transaccion::get_tipo_transaccion()
{
    return tipo_transaccion;
}
int transaccion::get_dia()
{
    return dia;
}
int transaccion::get_mes()
{
    return mes;
```

```
}

int transaccion::get_anio()
{
    return anio;
}

cliente::cliente(){};
cliente::cliente(string _nombre, string _apellido, int _dni, string _tipo_cliente, int _anio_ingreso, bool _estado, string _tarjeta) : persona(_nombre, _apellido, _dni)
{
    tipo_cliente = _tipo_cliente;
    anio_ingreso = _anio_ingreso;
    estado = _estado;
    tarjeta = _tarjeta;
};

void cliente::set_tipo_cliente(string _tipo_cliente)
{
    tipo_cliente = _tipo_cliente;
}
void cliente::set_anio_ingreso(int _anio_ingreso)
{
    anio_ingreso = _anio_ingreso;
}
void cliente::set_estado(bool _estado)
{
    estado = _estado;
}
void cliente::set_tarjeta(string _tarjeta)
{
    tarjeta = _tarjeta;
}
string cliente::get_tipo_clente()
{
    return tipo_cliente;
}
int cliente::get_anio_ingreso()
{
    return anio_ingreso;
}
bool cliente::get_estado()
{
    return estado;
```

```
}
string cliente::get_tarjeta()
{
   return tarjeta;
}
empleado::empleado(){};

banco::banco()
{
   num_clientes = 0;
   num_transacciones = 0;
}

banco::banco(cliente _clientes[], empleado _empleados[], transaccion
_transacciones[], int _num_clientes, int _num_transacciones)
{
   num_clientes = _num_clientes;
   num_transacciones = _num_transacciones;

        // Only copy as many elements as specified by _num_clientes and
_num_transacciones
   for (int i = 0; i < num_clientes; i++)
   {
      clientes[i] = _clientes[i];
   }
   for (int i = 0; i < num_transacciones; i++)
   {
      transacciones[i] = _transacciones[i];
   }
   for (int i = 0; i < num_clientes; i++)
   {
      empleados[i] = _empleados[i];
   }
}

void banco::set_num_clientes(int _num_clientes)
{
   num_clientes = _num_clientes;
}
int banco::get_num_clientes()
{
   return num_clientes;
}
void banco::set_num_transacciones(int _num_transacciones)
```

```
{
    num_transacciones = _num_transacciones;
}
int banco::get_num_transacciones()
{
    return num_transacciones;
}

void menu()
{
    cout << "*****************************************************" << endl;
    cout << "1. Agregar cliente" << endl;
    cout << "2. Cambiar estado de cliente" << endl;
    cout << "3. Listar clientes" << endl;
    cout << "4. Detallar cliente" << endl;
    cout << "5. Realizar transaccion" << endl;
    cout << "6. Listar transacciones" << endl;
    cout << "7. Informe de extracciones y depositos por mes y anio" << endl;
    cout << "*****************************************************" << endl;
    cout << "0. Terminar programa" << endl;
}

bool esnumero(string numero)
{
    int i;
    for (i = 0; i < numero.length(); i++)
    {
        if (isdigit(numero[i]) == false)
        {
            return false;
        }
    }
    return true;
}

bool banco::dni_existe(int dni)
{
    for (int i = 0; i < num_clientes; i++)
    {
        if (clientes[i].get_dni() == dni)
        {
            return true;
        }
    }
```

```
    return false;
}

bool esNombreValido(const string &nombre1)
{
    for (char c : nombre1)
    {
        if (!isalpha(c) && !isspace(c))
        {
            return false;
        }
    }
    return true;
}

string obtenerNombreCliente()
{
    string nombre1;
    bool nombreValido = false;

    while (!nombreValido)
    {
        cout << "Ingrese el nombre del cliente: ";
        cin >> nombre1;

        nombreValido = esNombreValido(nombre1);

        if (!nombreValido)
        {
              cout << "Nombre no válido. Ingrese solo caracteres letras y espacios." <<
endl;
        }
    }

    return nombre1;
}

string obtenerApellidoCliente()
{
    string apellido1;
    bool nombreValido = false;

    while (!nombreValido)
    {
```

```
        cout << "Ingrese el Apellido del cliente: ";
        cin >> apellido1;

        nombreValido = esNombreValido(apellido1);

        if (!nombreValido)
        {
              cout << "Apellido no válido. Ingrese solo caracteres letras y espacios." <<
endl;
        }
    }

    return apellido1;
}

bool esDniValido(int dni1)
{
    // Verificar que el número esté dentro del rango y tenga máximo 8 dígitos, tambien
que sea coherente
    return (dni1 >= 9999999 && dni1 <= 99999999);
};

int obtenerDniCliente()
{
    int dni1;
    bool numeroValido = false;

    while (!numeroValido)
    {
        cout << "Ingrese el DNI del cliente: ";
        if (!(cin >> dni1))
        {
            cin.clear();                                    // Limpiar el estado de error
                cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Descartar la entrada
inválida
            cout << "Entrada inválida. Por favor, ingrese un DNI valido" << endl;
        }
        else
        {
            if (esDniValido(dni1))
            {
                numeroValido = true;
            }
            else
```

```
            {
                cout << "DNI no válido. Ingrese un número entero con máximo 8 dígitos."
<< endl;
            }
        }
    }

    return dni1;
}

bool esAnioValido(int anio)
{
    return (anio >= 1990 && anio <= 2025);
}

string obtenerTipoCliente(){
    string tipo_cliente1;
    bool tipoValido = false;

    while (!tipoValido)
    {
        cout << "Ingrese el tipo de cliente (Plata, Oro, Platino): ";
        cin >> tipo_cliente1;

        if (tipo_cliente1 == "Plata" || tipo_cliente1 == "Oro" || tipo_cliente1 == "Platino")
        {
            tipoValido = true;
        }
        else
        {
            cout << "Tipo de cliente no válido" << endl;
        }
    }

    return tipo_cliente1;

}

int obtenerAnioCliente()
{
    string anioStr;
    int anioingreso1;
    bool anioValido = false;
```

```
    while (!anioValido)
    {
        cout << "Ingrese el año del cliente (entre 1990 y 2025): ";
        cin >> anioStr;

        if (esnumero(anioStr))
        {
            anioingreso1 = stoi(anioStr);
            if (esAnioValido(anioingreso1))
            {
                anioValido = true;
            }
            else
            {
                cout << "Año no válido. Ingrese un año entre 1990 y 2025." << endl;
            }
        }
        else
        {
            cout << "Entrada inválida. Por favor, ingrese un número válido." << endl;
        }
    }

    return anioingreso1;
}


bool esEstadoValido(int estado1)
{
    return (estado1 == 0 || estado1 == 1);
}
int obtenerEstadoCliente()
{
    int estado1;
    bool estadoValido = false;

    while (!estadoValido)
    {
        cout << "Ingrese el estado del cliente (0 si es baja, 1 si es alta): "<<endl;
        if (!(cin >> estado1))
        {
            cin.clear();                                    // Limpiar el estado de error
                cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Descartar la entrada
inválida
```

```
            cout << "Entrada inválida. Por favor, ingrese un estado valido" << endl;
        }
        else
        {
            if (esEstadoValido(estado1))
            {
                estadoValido = true;
            }
            else
            {
                cout << "Estado no válido,(0 si es baja, 1 si es alta) " << endl;
            }
        }
    }

    return estado1;
}

bool esNumeroTarjetaValido(const string &tarjeta)
{
    // Verificar que todos los caracteres sean dígitos y la longitud esté entre 13 y 18
    if (tarjeta.length() >= 13 && tarjeta.length() <= 18)
    {
        for (char c : tarjeta)
        {
            if (!isdigit(c))
            {
                return false;
            }
        }
        return true;
    }
    return false;
}

string obtenerNumeroTarjeta(string tipo_cliente)
{
    bool band = 1;
    string tarjeta;


        if(tipo_cliente == "Plata"){
            cout << "No tiene acceso a una tarjeta de credito"<<endl;;
            tarjeta = "No tiene tarjeta";
```

```
        }
        else if(tipo_cliente == "Oro"){
            while(band == 1){
            cout << "Ingrese el numero de tarjeta (13-18 digitos): "<<endl;
            cin >> tarjeta;
            if(esNumeroTarjetaValido(tarjeta)){
                break;
            }
            }
        }
        else if(tipo_cliente == "Platino"){
            while(band == 1){
            cout << "Ingrese el numero de tarjeta (13-18 digitos): "<<endl;
            cin>>tarjeta;
            if(esNumeroTarjetaValido(tarjeta)){
                break;
            }
            }
        }else {
            tarjeta = "Error en la tarjeta";
        }

    return tarjeta;
}

void banco::agregar_cliente()
{
    if (num_clientes >= 100)
    {
        cout << "No se pueden agregar más clientes." << endl;
        return;
    }

    string nombre1, apellido1, tipo_cliente1, tarjeta1;
    int dni1, anioingreso1, estado1;
    nombre1 = obtenerNombreCliente();
    apellido1 = obtenerApellidoCliente();
    dni1 = obtenerDniCliente();

    if (dni_existe(dni1))
    {
        cout << "El cliente ya existe." << endl;
        return;
    }
```

```
    tipo_cliente1 = obtenerTipoCliente();

    anioingreso1 = obtenerAnioCliente();

    estado1 = obtenerEstadoCliente();

    tarjeta1 = obtenerNumeroTarjeta(tipo_cliente1);

        cliente nuevo_cliente(nombre1, apellido1, dni1, tipo_cliente1, anioingreso1,
estado1, tarjeta1);
    clientes[num_clientes] = nuevo_cliente;
    num_clientes++;

    cout << "Cliente agregado correctamente." << endl;
}

void banco::cambiar_estado_cliente(){
    int dni1;
    bool band = 1;
    cout << "Ingrese el DNI del cliente a cambiar el estado: " << endl;
    cin >> dni1;
    if(num_clientes == 0){
        cout << "No hay clientes registrados" << endl;
        return;
    }

    for (int i = 0; i < num_clientes; i++)
    {
        if (clientes[i].get_dni() == dni1)
        {
            band = 0;
            if (clientes[i].get_estado() == 1)
            {
                clientes[i].set_estado(0);
                cout << "El cliente ha sido dado de baja" << endl;
            }
            else
            {
                clientes[i].set_estado(1);
                cout << "El cliente ha sido dado de alta" << endl;
            }
        }
    }
```

```
    if (band == 1)
    {
        cout << "No existe el cliente" << endl;
    }

}

void banco::listar_clientes()
{

    if (num_clientes == 0)
    {
        cout << "No hay clientes registrados" << endl;
        return;
    }
    cout << "-----------------------------------" << endl;
    for (int i = 0; i < num_clientes; i++)
    {
          cout << "Cliente Numero " << i + 1 << " con DNI: " << clientes[i].get_dni() <<
endl;
        cout << "-----------------------------------" << endl;
    }
    ofstream archivo("clientes.txt"); // Abre el archivo clientes.txt para escritura
    if (archivo.is_open())
    {
        for (int i = 0; i < num_clientes; ++i)
        {
            // Escribir los datos del cliente en el archivo en el formato deseado
            archivo << "Nombre: " << clientes[i].get_nombre() << endl;
            archivo << "Apellido: " << clientes[i].get_apellido() << endl;
            archivo << "DNI: " << clientes[i].get_dni() << endl;
            archivo << "Tipo de Cliente: " << clientes[i].get_tipo_clente() << endl;
            archivo << "Año de Ingreso: " << clientes[i].get_anio_ingreso() << endl;
            archivo << "Estado: " << (clientes[i].get_estado() ? "Activo" : "Baja") << endl;
            archivo << "Tarjeta: " << clientes[i].get_tarjeta() << endl;
            archivo << "-----------------------------------" << endl;
        }
        archivo.close(); // Cierra el archivo después de escribir todos los clientes
        cout << "Clientes guardados en clientes.txt correctamente." << endl;
    }
    else
    {
        cout << "No se pudo abrir el archivo clientes.txt." << endl;
    }
```

```
};

void banco::detallar_clientes()
{
   int numdetallar;
   cout << "Ingrese el numero del cliente a detallar: " << endl;
   cin >> numdetallar;
   numdetallar--;

   if (numdetallar > num_clientes)
   {
      cout << "No existe el cliente" << endl;
      return;
   }
   cout << "Nombre: " << clientes[numdetallar].get_nombre() << endl;
   cout << "Apellido: " << clientes[numdetallar].get_apellido() << endl;
   cout << "DNI: " << clientes[numdetallar].get_dni() << endl;
   cout << "Tipo de Cliente: " << clientes[numdetallar].get_tipo_clente() << endl;
   cout << "Año de Ingreso: " << clientes[numdetallar].get_anio_ingreso() << endl;
    cout << "Estado: " << (clientes[numdetallar].get_estado() ? "Activo" : "Baja") <<
endl;
   cout << "Tarjeta: " << clientes[numdetallar].get_tarjeta() << endl;
   cout << "-------------------------------------------------" << endl;
}

void banco::realizar_transaccion()
{
   int numero_cliente;
   bool band = 1;
   string tipo1, moneda1;
   string monto1, dia1, mes1, anio1;
   float monto2;
   int dia2, mes2, anio2;
   cout << "Ingrese el numero del cliente de la transaccion: " << endl;
   cin >> numero_cliente;
   if (numero_cliente > num_clientes)
   {
      cout << "No existe el cliente" << endl;
      return;
   }
   do
   {
      cout << "Tipo de transaccion (Deposito o Extraccion): " << endl;
      cin >> tipo1;
```

```
    if (tipo1 == "Deposito" || tipo1 == "Extraccion")
    {
        band = 0;
    }
} while (band == 1);
band = 1;
do
{
    cout << "Moneda de transaccion (Pesos o Dolares): " << endl;
    cin >> moneda1;
    if (moneda1 == "Pesos" || moneda1 == "Dolares")
    {
        band = 0;
    }
} while (band == 1);
band = 1;
do
{
    cout << "Monto de la transaccion: " << endl;
    cin >> monto1;
    if (esnumero(monto1))
    {
        monto2 = stof(monto1);
        band = 0;
    }
} while (band == 1);
band = 1;
do
{
    cout << "Dia de la transaccion: " << endl;
    cin >> dia1;
    if (esnumero(dia1))
    {
        if (stoi(dia1) > 0 && stoi(dia1) < 32)
        {
            dia2 = stoi(dia1);
            band = 0;
        }
    }
} while (band == 1);
band = 1;
do
{
    cout << "Mes de la transaccion: " << endl;
```

```
        cin >> mes1;
        if (esnumero(mes1))
        {
            if (stoi(mes1) > 0 && stoi(mes1) < 13)
            {
                mes2 = stoi(mes1);
                band = 0;
            }
        }
    } while (band == 1);
    band = 1;
    do
    {
        cout << "Año de la transaccion: " << endl;
        cin >> anio1;
        if (esnumero(anio1))
        {
            if (stoi(anio1) > 1990 && stoi(anio1) < 2025)
            {
                anio2 = stoi(anio1);
                band = 0;
            }
        }
    } while (band == 1);
        transaccion  nueva_transaccion(num_transacciones,  monto2,  numero_cliente,
moneda1, tipo1, dia2, mes2, anio2);
    transacciones[num_transacciones] = nueva_transaccion;
    cout << "Transaccion realizada correctamente" << endl;
    num_transacciones++;

    ofstream archivo("transacciones.txt");
    if (archivo.is_open())
    {
        archivo << "Numero de transaccion: " << num_transacciones << "\n";
          archivo << "Nombre del cliente: " << clientes[numero_cliente - 1].get_nombre()
<< "\n";
        archivo << "Numero de cliente: " << numero_cliente << "\n";
        archivo << "Tipo de transaccion: " << tipo1 << "\n";
        archivo << "Moneda: " << moneda1 << "\n";
        archivo << "Monto: " << monto2 << "\n";
        archivo << "Fecha: " << dia2 << "/" << mes2 << "/" << anio2 << "\n";
        archivo << "------------------------------\n";
        archivo.close();
    }
```

```
    else
    {
        cout << "No se pudo abrir el archivo para guardar la transaccion." << endl;
    }
}

void banco::listar_transacciones()
{
    int numero_cliente, verificador = num_transacciones;

    // Solicitar número de cliente
    cout << "Ingrese el número del cliente para listar transacciones: ";
    cin >> numero_cliente;

    // Validar si el número de cliente es válido
    if (numero_cliente > num_clientes || numero_cliente <= 0)
    {
        cout << "Número de cliente inválido." << endl;
        return;
    }

    // Mostrar las transacciones del cliente
     cout << "Transacciones del cliente: " << clientes[numero_cliente - 1].get_nombre()
<< endl;
    cout << "----------------------------------------" << endl;
    for (int i = 0; i < num_transacciones; i++)
    {
        if (transacciones[i].get_numero_cliente_transaccion() == numero_cliente)
        {
            verificador--;
                                        cout << "Número de transacción: " <<
transacciones[i].get_numero_transaccion() + 1 << endl;
             cout << "Tipo de transacción: " << transacciones[i].get_tipo_transaccion() <<
endl;
            cout << "Tipo de moneda: " << transacciones[i].get_tipo_moneda() << endl;
            cout << "Monto: " << transacciones[i].get_monto() << endl;
                            cout << "Fecha: " << transacciones[i].get_dia() << "/" <<
transacciones[i].get_mes() << "/" << transacciones[i].get_anio() << endl;
            cout << "----------------------------------------" << endl;
        }
    }
    if(verificador == num_transacciones){
        cout << "No hay transacciones para el cliente seleccionado" << endl;
    }
```

```
}

void banco::informes_transacciones_mes()
{
    if (num_transacciones == 0)
    {
        cout << "No hay transacciones registradas" << endl;
        return;
    }
    string mes1, anio1;
    int mes, anio, verificador = num_transacciones;
    bool band = false;

    cout << "Ingrese el mes a informar: "<<endl;
    cin >> mes1;
    cout << "Ingrese el año a informar: "<<endl;
    cin >> anio1;
    do
    {
        if (esnumero(mes1))
        {
            mes = stoi(mes1);
            if (mes < 1 || mes > 12)
            {
                cout << "Mes inválido. Ingrese un número entre 1 y 12." << endl;
                band = true;
            }
            else
            {
                band = false;
            }
        }
        else
        {
            cout << "Mes inválido. Ingrese un número entre 1 y 12." << endl;
            band = true;
        }
    } while (band);
    band = false;
        do
    {
        if (esnumero(anio1))
        {
            anio = stoi(anio1);
```

```
            if (anio < 1990 || anio > 2025)
            {
                cout << "Anio inválido. Ingrese un número entre 1990 y 2025." << endl;
                band = true;
            }
            else
            {
                band = false;
            }
        }
        else
        {
            cout << "Anio inválido. Ingrese un número entre 1990 y 2025." << endl;
            band = true;
        }
    } while (band);

    for (int i = 0; i < num_transacciones; i++)
    {
        if (transacciones[i].get_mes() == mes || transacciones[i].get_anio() == anio)
        {
            verificador --;
                                        cout << "Número de transacción: " <<
transacciones[i].get_numero_transaccion() + 1 << endl;
             cout << "Tipo de transacción: " << transacciones[i].get_tipo_transaccion() <<
endl;
            cout << "Tipo de moneda: " << transacciones[i].get_tipo_moneda() << endl;
            cout << "Monto: " << transacciones[i].get_monto() << endl;
                            cout << "Fecha: " << transacciones[i].get_dia() << "/" <<
transacciones[i].get_mes() << "/" << transacciones[i].get_anio() << endl;
            cout << "----------------------------------------" << endl;
        }
    }
    if(verificador == num_transacciones){
        cout << "No hay transacciones en el mes y año seleccionado" << endl;
    }
}

#include <iostream>
#include <cctype>
#include <fstream>
#include <string>
#include <limits>
#include "banco.cpp"
```

```
using namespace std;

int main()
{
    bool bandmenu = 0;
    string eleccion;
    int eleccionentera;
    string nombre1, apellido1, tipo_cliente1, tarjeta1;

    banco banco1;

    do
    {
        menu();
        cin >> eleccion;
        if (!esnumero(eleccion))
        {
            cout << "No Ingreso un numero valido" << endl;
        }
        else
        {
            eleccionentera = stoi(eleccion);
            if (eleccionentera == 0)
            {
                cout << "Gracias por usar el programa" << endl;
                bandmenu = 1;
            }
            else
            {
                switch (eleccionentera)
                {
                case 1:
                    banco1.agregar_cliente();
                    break;
                case 2:
                    banco1.cambiar_estado_cliente();
                    break;
                case 3:
                    banco1.listar_clientes();
                    break;
                case 4:
                    banco1.detallar_clientes();
                    break;
                case 5:
```

```
                banco1.realizar_transaccion();
                break;
            case 6:
                banco1.listar_transacciones();
                break;
            case 7:
                banco1.informes_transacciones_mes();
                break;
            default:
                cout << "Opcion no valida" << endl;
                break;
            }
        }
    }
  } while (!bandmenu);
}
```