TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE TIJUANA

SUBDIRECCIÓN ACADÉMICA
DEPARTAMENTO DE SISTEMAS Y COMPUTACIÓN

SEMESTRE:
Agosto - Diciembre 2025

CARRERA:
Ingeniería en Sistemas Computacionales

MATERIA:
Patrones de diseño

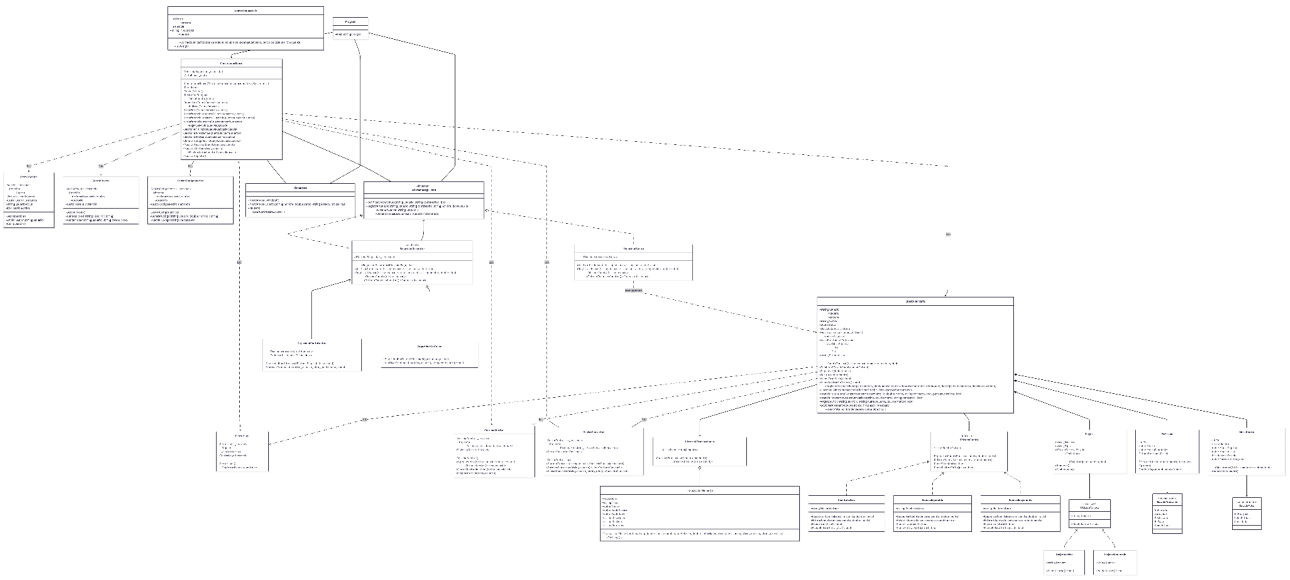
TÍTULO ACTIVIDAD:
Examen unidad 4 y 5

UNIDAD A EVALUAR:
Unidad 4 y 5

NOMBRE Y NÚMERO DE CONTROL DEL ALUMNO:
Ceballos Resendiz Veronica Sarahi 20211759

NOMBRE DEL MAESTRO (A):
Maribel Guerrero Luis

DIAGRAMA UML



CÓDIGO

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading;
// ================================================
// SISTEMA BANCARIO BBVA MEJORADO
// Patrones implementados:
// - DECORADOR (Estructural): Sistema de seguridad en capas
// - SINGLETON (Creacional): Gestor único de sesión y logs
// - MEMENTO (Comportamiento): Historial de transacciones
// - ESTADO (Comportamiento): Estados de cuenta bancaria
// - MVC (Arquitectura): Separación de responsabilidades
// ================================================
#region PATRÓN SINGLETON - Gestores Únicos del Sistema
public sealed class GestorSesion
{
```

```
    private static GestorSesion _instancia;
    private static readonly object _lock = new object();
    private string _usuarioActual;
    private DateTime _inicioSesion;
    private GestorSesion() { }
    public static GestorSesion Instancia
    {
        get
        {
            if (_instancia == null)
            {
                        lock (_lock) { if (_instancia == null)
_instancia = new GestorSesion(); }
            }
            return _instancia;
        }
    }
     public void IniciarSesion(string usuario) { _usuarioActual
= usuario; _inicioSesion = DateTime.Now; }
    public void CerrarSesion() { _usuarioActual = null; }
    public string UsuarioActual => _usuarioActual;
                      public     bool     SesionActiva     =>
!string.IsNullOrEmpty(_usuarioActual);
}


public sealed class GestorLogs
{
    private static GestorLogs _instancia;
```

```
    private static readonly object _lock = new object();
    private List<string> _logs = new List<string>();
    private GestorLogs() { }
    public static GestorLogs Instancia
    {
        get
        {
            if (_instancia == null)
            {
                        lock (_lock) { if (_instancia == null) _instancia = new GestorLogs(); }
            }
            return _instancia;
        }
    }
           public void RegistrarEvento(string evento) { _logs.Add($"[{DateTime.Now:yyyy-MM-dd HH:mm:ss}] {evento}"); }
}

public sealed class GestorTokens
{
    private static GestorTokens _instancia;
    private static readonly object _lock = new object();
        private Dictionary<string, (string Token, DateTime Expiracion)> _tokens = new Dictionary<string, (string, DateTime)>();
    private Random _random = new Random();
    private GestorTokens() { }
    public static GestorTokens Instancia
```

```
    {
        get
        {
            if (_instancia == null)
            {
                        lock (_lock) { if (_instancia == null)
_instancia = new GestorTokens(); }
            }
            return _instancia;
        }
    }
    public string GenerarToken(string usuario)
    {
                    string token = _random.Next(100000,
999999).ToString();
                                _tokens[usuario] = (token,
DateTime.Now.AddMinutes(5));
              GestorLogs.Instancia.RegistrarEvento($"Token 2FA
generado: {usuario}");
        return token;
    }
    public bool ValidarToken(string usuario, string token)
    {
        if (_tokens.ContainsKey(usuario))
        {
            var datos = _tokens[usuario];
                      if (DateTime.Now <= datos.Expiracion &&
datos.Token == token)
            {
```

```
                _tokens.Remove(usuario);
                return true;
            }
        }
        return false;
    }
}

public sealed class SistemaAlertas
{
    private static SistemaAlertas _instancia;
    private static readonly object _lock = new object();
                private    Dictionary<string,    List<string>>
_alertasPorUsuario = new Dictionary<string, List<string>>();
    private SistemaAlertas() { }
    public static SistemaAlertas Instancia
    {
        get
        {
            if (_instancia == null)
            {
                        lock (_lock) { if (_instancia == null)
_instancia = new SistemaAlertas(); }
            }
            return _instancia;
        }
    }
    public void AgregarAlerta(string usuario, string mensaje)
```

```
    {
                if (!_alertasPorUsuario.ContainsKey(usuario))
_alertasPorUsuario[usuario] = new List<string>();

_alertasPorUsuario[usuario].Add($"[{DateTime.Now:HH:mm}]
{mensaje}");
                if (_alertasPorUsuario[usuario].Count > 20)
_alertasPorUsuario[usuario].RemoveAt(0);
    }
    public List<string> ObtenerAlertas(string usuario)
    {
          return _alertasPorUsuario.ContainsKey(usuario) ? new
List<string>(_alertasPorUsuario[usuario])        :        new
List<string>();
    }
    public int ContarAlertasNoLeidas(string usuario)
    {
            return _alertasPorUsuario.ContainsKey(usuario) ?
_alertasPorUsuario[usuario].Count : 0;
    }
    public void LimpiarAlertas(string usuario)
    {
                if (_alertasPorUsuario.ContainsKey(usuario))
_alertasPorUsuario[usuario].Clear();
    }
}

public sealed class GestorContactos
{
    private static GestorContactos _instancia;
```

```
    private static readonly object _lock = new object();

          private Dictionary<string, List<ContactoBancario>>
_contactosPorUsuario = new Dictionary<string,
List<ContactoBancario>>();

    private GestorContactos() { }

    public static GestorContactos Instancia

    {

        get

        {

            if (_instancia == null)

            {

                        lock (_lock) { if (_instancia == null)
_instancia = new GestorContactos(); }

            }

            return _instancia;

        }

    }

             public void GuardarContacto(string usuario,
ContactoBancario contacto)

    {

               if (!_contactosPorUsuario.ContainsKey(usuario))
_contactosPorUsuario[usuario] = new List<ContactoBancario>();

          var existente = _contactosPorUsuario[usuario].Find(c
=> c.Alias.ToLower() == contacto.Alias.ToLower());

        if (existente == null)

        {

            _contactosPorUsuario[usuario].Add(contacto);

                GestorLogs.Instancia.RegistrarEvento($"Contacto
guardado: {contacto.Alias} para {usuario}");

        }
```

```
    }

        public List<ContactoBancario> ObtenerContactos(string
usuario)

    {

         return _contactosPorUsuario.ContainsKey(usuario) ? new
List<ContactoBancario>(_contactosPorUsuario[usuario]) : new
List<ContactoBancario>();

    }

       public ContactoBancario BuscarContacto(string usuario,
string alias)

    {

        if (_contactosPorUsuario.ContainsKey(usuario))

                return _contactosPorUsuario[usuario].Find(c =>
c.Alias.ToLower() == alias.ToLower());

        return null;

    }

}


public sealed class GestorCodigosRetiro

{

    private static GestorCodigosRetiro _instancia;

    private static readonly object _lock = new object();

     private Dictionary<string, (string Usuario, double Monto,
DateTime Expiracion)> _codigos = new Dictionary<string,
(string, double, DateTime)>();

    private Random _random = new Random();

    private GestorCodigosRetiro() { }

    public static GestorCodigosRetiro Instancia

    {

        get
```

```
        {
            if (_instancia == null)
            {
                        lock (_lock) { if (_instancia == null)
_instancia = new GestorCodigosRetiro(); }
            }
            return _instancia;
        }
    }
    public string GenerarCodigo(string usuario, double monto)
    {
                    string codigo = _random.Next(100000,
999999).ToString();
                     _codigos[codigo] = (usuario, monto,
DateTime.Now.AddHours(24));
          GestorLogs.Instancia.RegistrarEvento($"Código retiro
generado: {codigo} para {usuario}");
        return codigo;
    }
       public (bool Valido, string Usuario, double Monto)
ValidarCodigo(string codigo)
    {
        if (_codigos.ContainsKey(codigo))
        {
            var datos = _codigos[codigo];
            if (DateTime.Now <= datos.Expiracion)
            {
                _codigos.Remove(codigo);
                return (true, datos.Usuario, datos.Monto);
```

```
            }
            _codigos.Remove(codigo);
        }
        return (false, null, 0);
    }
}

#endregion

#region PATRÓN MEMENTO - Historial y Recuperación de Estados

public class TransaccionMemento
{
    public DateTime Fecha { get; }
    public string Tipo { get; }
    public double Monto { get; }
    public double SaldoAnterior { get; }
    public double SaldoNuevo { get; }
    public string Descripcion { get; }
    public string Destino { get; }
    public string Referencia { get; }

        public TransaccionMemento(string tipo, double monto,
double saldoAnterior, double saldoNuevo,
                                  string descripcion, string
destino = "", string referencia = "")
    {
        Fecha = DateTime.Now;
```

```
        Tipo = tipo;
        Monto = monto;
        SaldoAnterior = saldoAnterior;
        SaldoNuevo = saldoNuevo;
        Descripcion = descripcion;
        Destino = destino;
        Referencia = referencia;
    }

    public override string ToString()
    {
            string info = $"[{Fecha:dd/MM HH:mm}] {Tipo}: ${Monto:N2}";
            if (!string.IsNullOrEmpty(Destino)) info += $" → {Destino}";
            if (!string.IsNullOrEmpty(Referencia)) info += $" | Ref: {Referencia}";
        info += $" | ${SaldoAnterior:N2} → ${SaldoNuevo:N2}";
        return info;
    }
}

public class HistorialTransacciones
{
        private List<TransaccionMemento> _historial = new List<TransaccionMemento>();

    public void Guardar(TransaccionMemento memento)
    {
```

```
        _historial.Add(memento);
        if (_historial.Count > 100) _historial.RemoveAt(0);
    }

          public  List<TransaccionMemento>  ObtenerUltimas(int
cantidad)
    {
                    return  _historial.OrderByDescending(t  =>
t.Fecha).Take(cantidad).ToList();
    }
}

#endregion

#region PATRÓN ESTADO - Comportamiento Dinámico

public interface IEstadoCuenta
{
    void Depositar(CuentaBancaria cuenta, double monto);
    void Retirar(CuentaBancaria cuenta, double monto);
    bool PuedeTransferir();
    bool PuedeRetirarSinTarjeta();
    string NombreEstado { get; }
}

public class CuentaActiva : IEstadoCuenta
{
    public string NombreEstado => "ACTIVA";
```

```
    public void Depositar(CuentaBancaria cuenta, double monto)
    {
            if (monto <= 0) { Console.WriteLine(" Monto inválido."); return; }
        double saldoAnterior = cuenta.Saldo;
        cuenta.Saldo += monto;
             cuenta.RegistrarTransaccion("DEPÓSITO", monto, saldoAnterior, cuenta.Saldo, "Depósito exitoso");
         SistemaAlertas.Instancia.AgregarAlerta(cuenta.Usuario, $"Depósito: ${monto:N2}");
            Console.WriteLine($" Depósito exitoso. Saldo: ${cuenta.Saldo:N2}");
    }

    public void Retirar(CuentaBancaria cuenta, double monto)
    {
            if (monto <= 0 || cuenta.Saldo < monto) { Console.WriteLine(" Monto inválido o saldo insuficiente."); return; }
        double saldoAnterior = cuenta.Saldo;
        cuenta.Saldo -= monto;
              cuenta.RegistrarTransaccion("RETIRO", monto, saldoAnterior, cuenta.Saldo, "Retiro exitoso");
         SistemaAlertas.Instancia.AgregarAlerta(cuenta.Usuario, $"Retiro: ${monto:N2}");
              Console.WriteLine($" Retiro exitoso. Saldo: ${cuenta.Saldo:N2}");
            if (cuenta.Saldo < 100) cuenta.CambiarEstado(new CuentaBajaSaldo());
    }
```

```
    public bool PuedeTransferir() => true;
    public bool PuedeRetirarSinTarjeta() => true;
}

public class CuentaBajaSaldo : IEstadoCuenta
{
    public string NombreEstado => "SALDO BAJO";

    public void Depositar(CuentaBancaria cuenta, double monto)
    {
        if (monto <= 0) return;
        double saldoAnterior = cuenta.Saldo;
        cuenta.Saldo += monto;
                cuenta.RegistrarTransaccion("DEPÓSITO", monto,
saldoAnterior, cuenta.Saldo, "Recuperación");
               Console.WriteLine($" Depósito exitoso. Saldo:
${cuenta.Saldo:N2}");
        if (cuenta.Saldo >= 100)
        {
            cuenta.CambiarEstado(new CuentaActiva());
            Console.WriteLine(" Cuenta reactivada.");
        }
    }

    public void Retirar(CuentaBancaria cuenta, double monto)
    {
        Console.WriteLine(" Saldo bajo. No puedes retirar.");
```

```
    }

    public bool PuedeTransferir() => false;
    public bool PuedeRetirarSinTarjeta() => false;
}

public class CuentaSuspendida : IEstadoCuenta
{
    public string NombreEstado => "SUSPENDIDA";
     public void Depositar(CuentaBancaria cuenta, double monto)
=> Console.WriteLine(" Cuenta suspendida.");
      public void Retirar(CuentaBancaria cuenta, double monto)
=> Console.WriteLine(" Cuenta suspendida.");
    public bool PuedeTransferir() => false;
    public bool PuedeRetirarSinTarjeta() => false;
}

public interface IEstadoTarjeta
{
    bool PuedeUsarse();
    string Nombre { get; }
}

public class TarjetaActiva : IEstadoTarjeta
{
    public bool PuedeUsarse() => true;
    public string Nombre => "ACTIVA";
}
```

```
public class TarjetaBloqueada : IEstadoTarjeta
{
    public bool PuedeUsarse() => false;
    public string Nombre => "BLOQUEADA";
}

public enum EstadoPrestamo { Solicitado, Aprobado, Rechazado, EnPago, Completado }
public enum EstadoMeta { EnProgreso, Completada, Cancelada }

#endregion

#region MODELO - Entidades del Sistema

public class ContactoBancario
{
    public string Alias { get; set; }
    public string NombreCompleto { get; set; }
    public string Banco { get; set; }
    public string TipoCuenta { get; set; }
    public string NumeroCuenta { get; set; }

            public ContactoBancario(string alias, string nombreCompleto, string banco, string tipoCuenta, string numeroCuenta)
    {
        Alias = alias;
```

```
        NombreCompleto = nombreCompleto;
        Banco = banco;
        TipoCuenta = tipoCuenta;
        NumeroCuenta = numeroCuenta;
    }

    public override string ToString()
    {
            return $"{Alias} - {NombreCompleto} ({Banco}) -
{TipoCuenta}: {NumeroCuenta}";
    }
}

public class Tarjeta
{
    public string Numero { get; set; }
    public string Tipo { get; set; }
    public IEstadoTarjeta Estado { get; set; }
    public DateTime FechaExpiracion { get; set; }

     public Tarjeta(string tipo, bool generarNumeroCompleto =
false)
    {
        Random rnd = new Random();
        Thread.Sleep(10);

             if (generarNumeroCompleto && tipo.ToUpper() ==
"CRÉDITO")
```

```
        {
             Numero = $"{rnd.Next(4000, 4999)} {rnd.Next(1000, 9999)} {rnd.Next(1000, 9999)} {rnd.Next(1000, 9999)}";
        }
        else
        {
            Numero = $"**** **** **** {rnd.Next(1000, 9999)}";
        }

        Tipo = tipo.ToUpper();
        Estado = new TarjetaActiva();
         FechaExpiracion = DateTime.Now.AddYears(tipo.ToUpper() == "CRÉDITO" ? 5 : 3);
    }

    public void Bloquear()
    {
        Estado = new TarjetaBloqueada();
        Console.WriteLine($"🔒 Tarjeta {Numero} bloqueada.");
    }

    public void Desbloquear()
    {
        Estado = new TarjetaActiva();
                    Console.WriteLine($"🔓 Tarjeta {Numero} desbloqueada.");
    }
}
```

```
public class Prestamo
{
    public int Id { get; set; }
    public double Monto { get; set; }
    public double MontoPendiente { get; set; }
    public EstadoPrestamo Estado { get; set; }

    public Prestamo(int id, double monto, int plazo)
    {
        Id = id;
        Monto = monto;
        MontoPendiente = monto * 1.15;
        Estado = EstadoPrestamo.Solicitado;
    }

     public void Aprobar() { Estado = EstadoPrestamo.Aprobado;
}

    public bool RealizarPago(double monto)
    {
            if (monto <= 0 || monto > MontoPendiente) return
false;
        MontoPendiente -= monto;
                        Estado = MontoPendiente <= 0 ?
EstadoPrestamo.Completado : EstadoPrestamo.EnPago;
        return true;
    }
```

```
}

public class MetaAhorro
{
    public int Id { get; set; }
    public string Nombre { get; set; }
    public double MontoObjetivo { get; set; }
    public double MontoActual { get; set; }
    public EstadoMeta Estado { get; set; }

    public MetaAhorro(int id, string nombre, double objetivo)
    {
        Id = id;
        Nombre = nombre;
        MontoObjetivo = objetivo;
        MontoActual = 0;
        Estado = EstadoMeta.EnProgreso;
    }

    public void Abonar(double monto)
    {
        MontoActual += monto;
        if (MontoActual >= MontoObjetivo)
        {
            Estado = EstadoMeta.Completada;
                    Console.WriteLine($"  ¡Meta  completada:
{Nombre}!");
        }
```

```
    }

        public double PorcentajeProgreso => (MontoActual / MontoObjetivo) * 100;
}

public class CuentaBancaria
{
    public string Usuario { get; set; }
    public string NombreTitular { get; set; }
    public string NumeroCuenta { get; set; }
    public string CLABE { get; set; }
    public double Saldo { get; set; }
    private IEstadoCuenta _estado;
    private HistorialTransacciones _historial;
    public List<Tarjeta> Tarjetas { get; set; }
    public List<Prestamo> Prestamos { get; set; }
    public List<MetaAhorro> Metas { get; set; }
    private int _siguienteIdPrestamo = 1;
    private int _siguienteIdMeta = 1;

        public CuentaBancaria(string usuario, string nombre, double saldoInicial)
    {
        Usuario = usuario;
        NombreTitular = nombre;
        Saldo = saldoInicial;
```

```
        Random rnd = new Random();
                        NumeroCuenta = rnd.Next(1000000000,
1999999999).ToString();
                    CLABE = "012180" + rnd.Next(100000000,
999999999).ToString() + rnd.Next(10, 99).ToString();


        _historial = new HistorialTransacciones();
           _estado = saldoInicial >= 100 ? (IEstadoCuenta)new
CuentaActiva() : (IEstadoCuenta)new CuentaBajaSaldo();
          Tarjetas = new List<Tarjeta> { new Tarjeta("DÉBITO")
};
        Prestamos = new List<Prestamo>();
        Metas = new List<MetaAhorro>();
    }


    public void CambiarEstado(IEstadoCuenta nuevoEstado)
    {
        _estado = nuevoEstado;
          GestorLogs.Instancia.RegistrarEvento($"Cambio estado
{NombreTitular}: {nuevoEstado.NombreEstado}");
    }


              public void Depositar(double monto) =>
_estado.Depositar(this, monto);
    public void Retirar(double monto) => _estado.Retirar(this,
monto);
    public string EstadoActual => _estado.NombreEstado;
                  public bool PuedeTransferir() =>
_estado.PuedeTransferir();
               public bool PuedeRetirarSinTarjeta() =>
_estado.PuedeRetirarSinTarjeta();
```

```
        public void RegistrarTransaccion(string tipo, double
monto, double saldoAnterior, double saldoNuevo,
                                         string descripcion, string
destino = "", string referencia = "")
    {
            var memento = new TransaccionMemento(tipo, monto,
saldoAnterior, saldoNuevo, descripcion, destino, referencia);
        _historial.Guardar(memento);
    }

                            public       List<TransaccionMemento>
ObtenerUltimasTransacciones(int          cantidad)          =>
_historial.ObtenerUltimas(cantidad);

            public   bool   TransferirConDatos(ContactoBancario
destinatario,   double   monto,   string   concepto,   bool
guardarContacto)
    {
        if (!PuedeTransferir() || monto <= 0 || Saldo < monto)
        {
            Console.WriteLine(" Transferencia no permitida.");
            return false;
        }

        double saldoAnterior = Saldo;
        Saldo -= monto;

                        string   refTransferencia   =   "TRF"   +
DateTime.Now.ToString("yyMMddHHmmss");
```

```
                                    string    detalleDestino    =
$"{destinatario.NombreCompleto} - {destinatario.Banco}";


                  RegistrarTransaccion("TRANSFERENCIA",  monto,
saldoAnterior,      Saldo,      concepto,      detalleDestino,
refTransferencia);

                SistemaAlertas.Instancia.AgregarAlerta(Usuario,
$"Transferencia ${monto:N2} a {destinatario.NombreCompleto}");


        if (guardarContacto)

        {

             GestorContactos.Instancia.GuardarContacto(Usuario,
destinatario);

        }


        Console.WriteLine($"\n Transferencia exitosa");

        Console.WriteLine($" Referencia: {refTransferencia}");

                                        Console.WriteLine($"    A:
{destinatario.NombreCompleto}");

        Console.WriteLine($" Banco: {destinatario.Banco}");

        Console.WriteLine($" Monto: ${monto:N2}");


        return true;

    }


       public  bool  TransferirInterno(CuentaBancaria  destino,
double monto, string concepto)

    {

        if (!PuedeTransferir() || monto <= 0 || Saldo < monto)

        {
```

```
            Console.WriteLine(" Transferencia no permitida.");
            return false;
        }

        double saldoAnterior = Saldo;
        Saldo -= monto;
                    string refTransferencia = "INT" +
DateTime.Now.ToString("yyMMddHHmmss");

          RegistrarTransaccion("TRANSFERENCIA INTERNA", monto,
saldoAnterior, Saldo, concepto, destino.NombreTitular,
refTransferencia);
                SistemaAlertas.Instancia.AgregarAlerta(Usuario,
$"Enviaste ${monto:N2} a {destino.NombreTitular}");

        destino.Saldo += monto;
         destino.RegistrarTransaccion("TRANSFERENCIA RECIBIDA",
monto, destino.Saldo - monto, destino.Saldo, concepto,
NombreTitular, refTransferencia);

SistemaAlertas.Instancia.AgregarAlerta(destino.Usuario,
$"Recibiste ${monto:N2} de {NombreTitular}");

            Console.WriteLine($" Transferencia exitosa. Ref:
{refTransferencia}");
        return true;
    }

         public bool PagarServicio(string servicio, string
numeroCuenta, double monto)
    {
```

```
        if (Saldo < monto)
        {
            Console.WriteLine(" Saldo insuficiente.");
            return false;
        }

        double saldoAnterior = Saldo;
        Saldo -= monto;
                            string refPago = "PAG" +
DateTime.Now.ToString("yyMMddHHmmss");

                 RegistrarTransaccion("PAGO SERVICIO", monto,
saldoAnterior, Saldo, $"{servicio} - Cuenta: {numeroCuenta}",
"", refPago);
         SistemaAlertas.Instancia.AgregarAlerta(Usuario, $"Pago
${monto:N2} a {servicio}");

        Console.WriteLine($"Pago realizado a {servicio}");
        Console.WriteLine($"Referencia: {refPago}");
        Console.WriteLine($"Cuenta: {numeroCuenta}");
        return true;
    }

    public Prestamo SolicitarPrestamo(double monto, int plazo)
    {
          var prestamo = new Prestamo(_siguienteIdPrestamo++,
monto, plazo);
        prestamo.Aprobar();
        Prestamos.Add(prestamo);
```

```
        Saldo += monto;

                SistemaAlertas.Instancia.AgregarAlerta(Usuario,
$"Préstamo ${monto:N2} aprobado");

        return prestamo;

    }


         public MetaAhorro CrearMeta(string nombre, double
objetivo)

    {

          var meta = new MetaAhorro(_siguienteIdMeta++, nombre,
objetivo);

        Metas.Add(meta);

         SistemaAlertas.Instancia.AgregarAlerta(Usuario, $"Meta
'{nombre}' creada");

        return meta;

    }

}


#endregion


#region PATRÓN DECORADOR - Sistema de Seguridad en Capas


public interface ISistemaSeguridad

{

          bool VerificarCredenciales(string usuario, string
contraseña);

     void RegistrarUsuario(string usuario, string contraseña,
string nombre, double saldo);

    CuentaBancaria ObtenerCuenta(string usuario);
```

```
                    Dictionary<string,      CuentaBancaria>
ObtenerTodasLasCuentas();
}

public class SeguridadBasica : ISistemaSeguridad
{
          private  Dictionary<string,  (string  Contraseña,
CuentaBancaria  Cuenta)>  _usuarios  =  new  Dictionary<string,
(string, CuentaBancaria)>();

     public bool VerificarCredenciales(string usuario, string
contraseña)
    {
                  return  _usuarios.ContainsKey(usuario)  &&
_usuarios[usuario].Contraseña == contraseña;
    }

        public  void  RegistrarUsuario(string  usuario,  string
contraseña, string nombre, double saldo)
    {
        if (!_usuarios.ContainsKey(usuario))
        {
              var cuenta = new CuentaBancaria(usuario, nombre,
saldo);
            _usuarios.Add(usuario, (contraseña, cuenta));
            Console.WriteLine(" Registro exitoso.");
                GestorLogs.Instancia.RegistrarEvento($"Usuario
registrado: {usuario}");
        }
        else
```

```
        {
            Console.WriteLine(" Usuario ya existe.");
        }
    }

    public CuentaBancaria ObtenerCuenta(string usuario)
    {
                    return _usuarios.ContainsKey(usuario) ?
_usuarios[usuario].Cuenta : null;
    }

                public Dictionary<string, CuentaBancaria>
ObtenerTodasLasCuentas()
    {
                    var cuentas = new Dictionary<string,
CuentaBancaria>();
        foreach (var kvp in _usuarios)
            cuentas[kvp.Key] = kvp.Value.Cuenta;
        return cuentas;
    }
}

public abstract class SeguridadDecorator : ISistemaSeguridad
{
    protected ISistemaSeguridad _seguridad;

    public SeguridadDecorator(ISistemaSeguridad seguridad)
    {
```

```
        _seguridad = seguridad;
    }

     public virtual bool VerificarCredenciales(string usuario,
string contraseña)
    {
              return _seguridad.VerificarCredenciales(usuario,
contraseña);
    }

       public virtual void RegistrarUsuario(string usuario,
string contraseña, string nombre, double saldo)
    {
              _seguridad.RegistrarUsuario(usuario, contraseña,
nombre, saldo);
    }

         public virtual CuentaBancaria ObtenerCuenta(string
usuario)
    {
        return _seguridad.ObtenerCuenta(usuario);
    }

         public virtual Dictionary<string, CuentaBancaria>
ObtenerTodasLasCuentas()
    {
        return _seguridad.ObtenerTodasLasCuentas();
    }
}
```

```
public class SeguridadConIntentos : SeguridadDecorator
{
     private Dictionary<string, int> _intentosFallidos = new Dictionary<string, int>();
      private Dictionary<string, DateTime> _bloqueos = new Dictionary<string, DateTime>();

    public SeguridadConIntentos(ISistemaSeguridad seguridad) : base(seguridad) { }

    public override bool VerificarCredenciales(string usuario, string contraseña)
    {
        if (_bloqueos.ContainsKey(usuario))
        {
                    TimeSpan diferencia = DateTime.Now - _bloqueos[usuario];
            if (diferencia.TotalMinutes < 3)
            {
                     Console.WriteLine($" Bloqueado. Espera {Math.Ceiling(3 - diferencia.TotalMinutes)} min.");
                return false;
            }
            else
            {
                _bloqueos.Remove(usuario);
                _intentosFallidos[usuario] = 0;
            }
        }
```

```
                                        bool valido =
_seguridad.VerificarCredenciales(usuario, contraseña);


        if (valido)

        {

            _intentosFallidos[usuario] = 0;

                 GestorLogs.Instancia.RegistrarEvento($"Inicio
exitoso: {usuario}");

            return true;

        }

        else

        {

                 if (!_intentosFallidos.ContainsKey(usuario))
_intentosFallidos[usuario] = 0;

            _intentosFallidos[usuario]++;


            if (_intentosFallidos[usuario] >= 3)

            {

                    Console.WriteLine(" Cuenta bloqueada por 3
minutos.");

                _bloqueos[usuario] = DateTime.Now;

            }

            else

            {

                   Console.WriteLine($" Contraseña incorrecta.
Restantes: {3 - _intentosFallidos[usuario]}");

            }

            return false;

        }
```

```
    }
}

public class SeguridadConToken : SeguridadDecorator
{
      public SeguridadConToken(ISistemaSeguridad seguridad) :
base(seguridad) { }

    public override bool VerificarCredenciales(string usuario,
string contraseña)
    {
                                bool    credencialesValidas    =
_seguridad.VerificarCredenciales(usuario, contraseña);
        if (!credencialesValidas) return false;

                                          string     token     =
GestorTokens.Instancia.GenerarToken(usuario);
        Console.WriteLine($"\n Token de seguridad: {token}");
        Console.WriteLine(" Expira en 5 minutos.");
        Console.Write("Ingresa el token: ");
        string tokenIngresado = Console.ReadLine();

              if  (GestorTokens.Instancia.ValidarToken(usuario,
tokenIngresado))
        {
            Console.WriteLine(" Token válido.");
            return true;
        }
        else
```

```
        {
            Console.WriteLine(" Token inválido o expirado.");
            return false;
        }
    }
}

#endregion

#region VISTA - Interfaz de Usuario (MVC)

public class VistaBanco
{
    public void MostrarMenuPrincipal()
    {
        Console.Clear();
        Console.ForegroundColor = ConsoleColor.Cyan;

Console.WriteLine("╔═══════════════════════════════════════════╗");
            Console.WriteLine("║           BANCO BBVA MÉXICO
║");

Console.WriteLine("╚═══════════════════════════════════════════╝");
        Console.ResetColor();
        Console.WriteLine("\n1. Registrar Cliente");
        Console.WriteLine("2.  Iniciar Sesión");
             Console.WriteLine("3.   Retiro sin Tarjeta (con
código)");
        Console.WriteLine("4.  Salir");
```

```
        Console.Write("\nOpción: ");
    }

    public void MostrarMenuCuenta(string nombre, double saldo,
string estado, int alertas)
    {
        Console.Clear();
        Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine("╔═══════════════════════════════════════════╗");
        Console.WriteLine($"║  {nombre,-34}║");

Console.WriteLine("╚═══════════════════════════════════════════╝");
        Console.ResetColor();
          Console.WriteLine($"\n Saldo: ${saldo:N2} | Estado:
{estado}");
            if (alertas > 0) Console.WriteLine($" {alertas}
alertas nuevas");

        Console.WriteLine("\n═══ OPERACIONES ═══");
        Console.WriteLine("1. Depositar");
        Console.WriteLine("2. Retirar");
        Console.WriteLine("3. Transferir");
        Console.WriteLine("4. Pagar Servicios");

        Console.WriteLine("\n═══ TARJETAS ═══");
        Console.WriteLine("5. Gestionar Tarjetas");

        Console.WriteLine("\n═══ CRÉDITOS E INVERSIONES ═══");
```

```
        Console.WriteLine("6. Préstamos");
        Console.WriteLine("7. Metas de Ahorro");

        Console.WriteLine("\n═══ UTILIDADES ═══");
             Console.WriteLine("8. Generar Código Retiro sin
Tarjeta");
        Console.WriteLine("9. Historial");
        Console.WriteLine("10. Ver Alertas");
        Console.WriteLine("11. Mis Datos");

        Console.WriteLine("\n═══ SOPORTE ═══");
        Console.WriteLine("12. Chat/Soporte");
        Console.WriteLine("13. Cerrar Sesión");
        Console.Write("\nOpción: ");
    }

    public void Pausar()
    {
        Console.WriteLine("\nPresiona Enter...");
        Console.ReadLine();
    }

    public string LeerContraseñaOculta()
    {
        string contraseña = "";
        ConsoleKeyInfo tecla;

        do
```

```
        {
            tecla = Console.ReadKey(true);
             if (tecla.Key != ConsoleKey.Backspace && tecla.Key
!= ConsoleKey.Enter)
            {
                contraseña += tecla.KeyChar;
                Console.Write("*");
            }
                else if (tecla.Key == ConsoleKey.Backspace &&
contraseña.Length > 0)
            {
                        contraseña = contraseña.Substring(0,
contraseña.Length - 1);
                Console.Write("\b \b");
            }
        } while (tecla.Key != ConsoleKey.Enter);

        Console.WriteLine();
        return contraseña;
    }
}

#endregion

#region CONTROLADOR - Lógica de Negocio (MVC)

public class ControladorBanco
{
```

```
    private ISistemaSeguridad _seguridad;
    private VistaBanco _vista;

        public ControladorBanco(ISistemaSeguridad seguridad,
VistaBanco vista)
    {
        _seguridad = seguridad;
        _vista = vista;
    }

    public void Registrar()
    {
        Console.Clear();

Console.WriteLine("╔═══════════════════════════════════════════╗");
            Console.WriteLine("║          Registro de Cliente
║");

Console.WriteLine("╚═══════════════════════════════════════════╝");

        Console.Write("\n Nombre completo: ");
        string nombre = Console.ReadLine();
        Console.Write(" Usuario: ");
        string usuario = Console.ReadLine();
        Console.Write(" Contraseña: ");
        string contraseña = _vista.LeerContraseñaOculta();
        Console.Write(" Saldo inicial: $");
```

```
            if (!double.TryParse(Console.ReadLine(), out double
saldo) || saldo < 0)
        {
            Console.WriteLine(" Saldo inválido.");
            _vista.Pausar();
            return;
        }

              _seguridad.RegistrarUsuario(usuario, contraseña,
nombre, saldo);
        _vista.Pausar();
    }

    public void IniciarSesion()
    {
        Console.Clear();

Console.WriteLine("╔═══════════════════════════════════════════╗");
             Console.WriteLine("║              Iniciar Sesión
║");

Console.WriteLine("╚═══════════════════════════════════════════╝");

        Console.Write("\n Usuario: ");
        string usuario = Console.ReadLine();

        int intentos = 0;
        while (intentos < 4)
        {
```

```
            Console.Write(" Contraseña: ");
            string contraseña = _vista.LeerContraseñaOculta();

                 if (_seguridad.VerificarCredenciales(usuario,
contraseña))
            {
                GestorSesion.Instancia.IniciarSesion(usuario);
                MenuCuenta(usuario);
                return;
            }
            else
            {
                intentos++;
                if (intentos < 4)
                {
                                Console.WriteLine($"\n  Intento
{intentos}/4");
                    _vista.Pausar();
                }
            }
        }
        Console.WriteLine("\n Límite alcanzado.");
        Thread.Sleep(2000);
    }

    public void RetiroConCodigo()
    {
        Console.Clear();
```

```
Console.WriteLine("╔═══════════════════════════════════════════╗");
            Console.WriteLine("║         Retiro sin Tarjeta ║");
Console.WriteLine("╚═══════════════════════════════════════════╝");

        Console.Write("\nIngresa tu código de 6 dígitos: ");
        string codigo = Console.ReadLine();

                                        var resultado =
GestorCodigosRetiro.Instancia.ValidarCodigo(codigo);

        if (resultado.Valido)
        {
                                            var cuenta =
_seguridad.ObtenerCuenta(resultado.Usuario);
                    if (cuenta != null && cuenta.Saldo >=
resultado.Monto)
            {
                cuenta.Retirar(resultado.Monto);
                     Console.WriteLine($"\nRetiro exitoso de
${resultado.Monto:N2}");
                                    Console.WriteLine($"Usuario:
{cuenta.NombreTitular}");
            }
            else
            {
                Console.WriteLine("Saldo insuficiente.");
            }
```

```
        }
        else
        {
            Console.WriteLine("Código inválido o expirado.");
        }
        _vista.Pausar();
    }

    private void MenuCuenta(string usuario)
    {
        var cuenta = _seguridad.ObtenerCuenta(usuario);
        if (cuenta == null) return;

        int opcion;
        do
        {
                                                int    alertas    =
SistemaAlertas.Instancia.ContarAlertasNoLeidas(usuario);
                    _vista.MostrarMenuCuenta(cuenta.NombreTitular,
cuenta.Saldo, cuenta.EstadoActual, alertas);

             if (!int.TryParse(Console.ReadLine(), out opcion))
opcion = 0;

            switch (opcion)
            {
                case 1: Depositar(cuenta); break;
                case 2: Retirar(cuenta); break;
```

```
                case 3: Transferir(cuenta); break;
                case 4: PagarServicio(cuenta); break;
                case 5: GestionarTarjetas(cuenta); break;
                case 6: GestionarPrestamos(cuenta); break;
                case 7: GestionarMetas(cuenta); break;
                case 8: GenerarCodigoRetiro(cuenta); break;
                case 9: MostrarHistorial(cuenta); break;
                case 10: MostrarAlertas(usuario); break;
                case 11: MostrarDatosCuenta(cuenta); break;
                case 12: MostrarSoporte(); break;
                case 13:
                    GestorSesion.Instancia.CerrarSesion();
                    Console.WriteLine("\n Sesión cerrada.");
                    Thread.Sleep(1500);
                    break;
                default:
                    Console.WriteLine(" Opción inválida.");
                    Thread.Sleep(1000);
                    break;
            }
        } while (opcion != 13);
    }

    private void Depositar(CuentaBancaria cuenta)
    {
        Console.Write("\n Monto a depositar: $");
            if (double.TryParse(Console.ReadLine(), out double
monto))
```

```
            cuenta.Depositar(monto);
        else
            Console.WriteLine(" Monto inválido.");
        _vista.Pausar();
    }

    private void Retirar(CuentaBancaria cuenta)
    {
        Console.Write("\n Monto a retirar: $");
           if (double.TryParse(Console.ReadLine(), out double
monto))
            cuenta.Retirar(monto);
        else
            Console.WriteLine(" Monto inválido.");
        _vista.Pausar();
    }

    private void Transferir(CuentaBancaria cuenta)
    {
        Console.Clear();
        Console.WriteLine("═══ TRANSFERIR ═══\n");
        Console.WriteLine("1. Transferencia a otra cuenta");
              Console.WriteLine("2. Transferencia a contacto
guardado");
        Console.WriteLine("3. Transferencia interna BBVA");
        Console.Write("\nOpción: ");
```

```
            if (!int.TryParse(Console.ReadLine(), out int
tipoTrans)) return;

        if (tipoTrans == 1)
        {
            TransferenciaConDatos(cuenta);
        }
        else if (tipoTrans == 2)
        {
            TransferenciaContactoGuardado(cuenta);
        }
        else if (tipoTrans == 3)
        {
            TransferenciaInterna(cuenta);
        }
    }

    private void TransferenciaConDatos(CuentaBancaria cuenta)
    {
        Console.Clear();
              Console.WriteLine("═══ TRANSFERENCIA CON DATOS
═══\n");

        Console.Write(" Nombre completo del beneficiario: ");
        string nombreBeneficiario = Console.ReadLine();

        Console.Write(" Banco destino: ");
        string banco = Console.ReadLine();
```

```
        Console.WriteLine("\nTipo de cuenta:");
        Console.WriteLine("1. CLABE (18 dígitos)");
        Console.WriteLine("2. Tarjeta (16 dígitos)");
        Console.WriteLine("3. Número de cuenta (10 dígitos)");
        Console.Write("Opción: ");

        string tipoCuenta = "";
        string numeroCuenta = "";

        if (int.TryParse(Console.ReadLine(), out int tipo))
        {
            switch (tipo)
            {
                case 1:
                    tipoCuenta = "CLABE";
                    Console.Write(" CLABE (18 dígitos): ");
                    numeroCuenta = Console.ReadLine();
                    break;
                case 2:
                    tipoCuenta = "TARJETA";
                        Console.Write(" Número de tarjeta (16
dígitos): ");
                    numeroCuenta = Console.ReadLine();
                    break;
                case 3:
                    tipoCuenta = "CUENTA";
```

```
                    Console.Write(" Número de cuenta (10
dígitos): ");
                    numeroCuenta = Console.ReadLine();
                    break;
                }
            }

            Console.Write("Monto: $");
              if (!double.TryParse(Console.ReadLine(), out double
monto)) return;

            Console.Write("Concepto: ");
            string concepto = Console.ReadLine();

            Console.Write("\n¿Guardar este contacto? (S/N): ");
            bool guardar = Console.ReadLine().ToUpper() == "S";

            if (guardar)
            {
                Console.Write("Alias para el contacto: ");
                string alias = Console.ReadLine();
                  var contacto = new ContactoBancario(alias,
nombreBeneficiario, banco, tipoCuenta, numeroCuenta);
                  cuenta.TransferirConDatos(contacto, monto,
concepto, true);
            }
            else
            {
```

```
                    var contacto = new ContactoBancario("",
nombreBeneficiario, banco, tipoCuenta, numeroCuenta);
                    cuenta.TransferirConDatos(contacto, monto,
concepto, false);
        }

        _vista.Pausar();
    }

     private void TransferenciaContactoGuardado(CuentaBancaria
cuenta)
    {
        Console.Clear();
        Console.WriteLine("═══ CONTACTOS GUARDADOS ═══\n");

                                        var contactos =
GestorContactos.Instancia.ObtenerContactos(cuenta.Usuario);

        if (contactos.Count == 0)
        {
                    Console.WriteLine("No tienes contactos
guardados.");
            _vista.Pausar();
            return;
        }

        for (int i = 0; i < contactos.Count; i++)
        {
            Console.WriteLine($"{i + 1}. {contactos[i]}");
```

```
        }

        Console.Write("\nSelecciona contacto: ");
          if (int.TryParse(Console.ReadLine(), out int sel) &&
sel > 0 && sel <= contactos.Count)
        {
            Console.Write("Monto: $");
             if (double.TryParse(Console.ReadLine(), out double
monto))
            {
                Console.Write("Concepto: ");
                string concepto = Console.ReadLine();
                  cuenta.TransferirConDatos(contactos[sel - 1],
monto, concepto, false);
            }
        }
        _vista.Pausar();
    }

    private void TransferenciaInterna(CuentaBancaria cuenta)
    {
        Console.Clear();
            Console.WriteLine("═══ TRANSFERENCIA INTERNA BBVA
═══\n");

        var cuentas = _seguridad.ObtenerTodasLasCuentas();
        int i = 1;
        var lista = new List<CuentaBancaria>();
```

```
        foreach (var kvp in cuentas)
        {
            if (kvp.Key != cuenta.Usuario)
            {
                                        Console.WriteLine($"{i}.
{kvp.Value.NombreTitular}                 -                Cuenta:
{kvp.Value.NumeroCuenta}");
                lista.Add(kvp.Value);
                i++;
            }
        }

        if (lista.Count == 0)
        {
            Console.WriteLine("No hay otros usuarios.");
            _vista.Pausar();
            return;
        }

        Console.Write("\nSelecciona destinatario: ");
          if (int.TryParse(Console.ReadLine(), out int sel) &&
sel > 0 && sel <= lista.Count)
        {
            Console.Write("Monto: $");
             if (double.TryParse(Console.ReadLine(), out double
monto))
            {
                Console.Write("Concepto: ");
                string concepto = Console.ReadLine();
```

```
                        cuenta.TransferirInterno(lista[sel - 1],
monto, concepto);
            }
        }
        _vista.Pausar();
    }

    private void PagarServicio(CuentaBancaria cuenta)
    {
        Console.Clear();
        Console.WriteLine("═══ PAGAR SERVICIOS ═══\n");
        Console.WriteLine("1. Luz (CFE)");
        Console.WriteLine("2. Agua");
        Console.WriteLine("3. Teléfono");
        Console.WriteLine("4. Internet");
        Console.WriteLine("5. Gas");
        Console.Write("\nServicio: ");

         string[] servicios = { "", "CFE", "Agua", "Teléfono",
"Internet", "Gas" };
          if (int.TryParse(Console.ReadLine(), out int srv) &&
srv > 0 && srv < servicios.Length)
        {
            Console.Write("Número de cuenta del servicio: ");
            string numeroCuenta = Console.ReadLine();

            Console.Write("Monto: $");
             if (double.TryParse(Console.ReadLine(), out double
monto))
```

```
            {
                cuenta.PagarServicio(servicios[srv], numeroCuenta, monto);
            }
        }
        _vista.Pausar();
    }

    private void GestionarTarjetas(CuentaBancaria cuenta)
    {
        Console.Clear();
        Console.WriteLine("═══ GESTIÓN DE TARJETAS ═══\n");

        for (int i = 0; i < cuenta.Tarjetas.Count; i++)
        {
            var t = cuenta.Tarjetas[i];
             Console.WriteLine($"{i + 1}. {t.Tipo} {t.Numero} - {t.Estado.Nombre} - Exp: {t.FechaExpiracion:MM/yy}");
        }

        Console.WriteLine("\n1. Bloquear tarjeta");
        Console.WriteLine("2. Desbloquear tarjeta");
        Console.WriteLine("3. Agregar tarjeta de crédito");
        Console.Write("\nOpción: ");

        if (int.TryParse(Console.ReadLine(), out int op))
        {
            if (op == 1 || op == 2)
```

```
            {
                        Console.Write("Número de tarjeta (1-" +
cuenta.Tarjetas.Count + "): ");
                    if (int.TryParse(Console.ReadLine(), out int
num) && num > 0 && num <= cuenta.Tarjetas.Count)
                {
                    if (op == 1)
                        cuenta.Tarjetas[num - 1].Bloquear();
                    else
                                        cuenta.Tarjetas[num -
1].Desbloquear();
                }
            }
            else if (op == 3)
            {
                var tarjeta = new Tarjeta("CRÉDITO", true);
                cuenta.Tarjetas.Add(tarjeta);
                     Console.WriteLine($"\n Tarjeta de crédito
agregada");
                                    Console.WriteLine($"  Número:
{tarjeta.Numero}");
                                    Console.WriteLine($"  Expira:
{tarjeta.FechaExpiracion:MM/yyyy}");
            }
        }
        _vista.Pausar();
    }

    private void GestionarPrestamos(CuentaBancaria cuenta)
    {
```

```
        Console.Clear();
        Console.WriteLine("═══ PRÉSTAMOS ═══\n");

        if (cuenta.Prestamos.Count == 0)
        {
            Console.WriteLine("No tienes préstamos.");
        }
        else
        {
            foreach (var p in cuenta.Prestamos)
            {
                  Console.WriteLine($"#{p.Id} - ${p.Monto:N2} |
Pendiente: ${p.MontoPendiente:N2} | {p.Estado}");
            }
        }

        Console.WriteLine("\n1. Solicitar préstamo");
        Console.WriteLine("2. Pagar préstamo");
        Console.Write("\nOpción: ");

        if (int.TryParse(Console.ReadLine(), out int op))
        {
            if (op == 1)
            {
                Console.Write("Monto: $");
                    if (double.TryParse(Console.ReadLine(), out
double monto))
                {
```

```
                    Console.Write(" Plazo (meses): ");
                      if (int.TryParse(Console.ReadLine(), out
int plazo))
                    {
                                                var prestamo =
cuenta.SolicitarPrestamo(monto, plazo);
                                Console.WriteLine($"\n Préstamo
#{prestamo.Id} aprobado");
                            Console.WriteLine($"Total a pagar:
${prestamo.MontoPendiente:N2}");
                    }
                }
            }
            else if (op == 2)
            {
                Console.Write("ID del préstamo: ");
                  if (int.TryParse(Console.ReadLine(), out int
id))
                {
                    var prestamo = cuenta.Prestamos.Find(p =>
p.Id == id);
                    if (prestamo != null)
                    {
                                Console.Write($"Monto a pagar
(Pendiente: ${prestamo.MontoPendiente:N2}): $");
                                                        if
(double.TryParse(Console.ReadLine(), out double pago))
                        {
                                    if (cuenta.Saldo >= pago &&
prestamo.RealizarPago(pago))
                            {
```

```
                                cuenta.Saldo -= pago;
                                Console.WriteLine($" Pago realizado. Restante: ${prestamo.MontoPendiente:N2}");
                            }
                            else
                            {
                                Console.WriteLine(" Pago inválido o saldo insuficiente.");
                            }
                        }
                    }
                }
            }
            _vista.Pausar();
        }

        private void GestionarMetas(CuentaBancaria cuenta)
        {
            Console.Clear();
            Console.WriteLine("═══ METAS DE AHORRO ═══\n");

            if (cuenta.Metas.Count == 0)
            {
                Console.WriteLine("No tienes metas.");
            }
            else
            {
```

```
            foreach (var m in cuenta.Metas)
            {
                    Console.WriteLine($"#{m.Id} {m.Nombre} -
${m.MontoActual:N2}/${m.MontoObjetivo:N2}
({m.PorcentajeProgreso:F1}%) - {m.Estado}");
            }
        }

        Console.WriteLine("\n1. Crear meta");
        Console.WriteLine("2. Abonar a meta");
        Console.Write("\nOpción: ");

            if (int.TryParse(Console.ReadLine(), out int
opcionMeta))
        {
            if (opcionMeta == 1)
            {
                Console.Write("Nombre: ");
                string nombreMeta = Console.ReadLine();
                Console.Write("Objetivo: $");
                   if (double.TryParse(Console.ReadLine(), out
double objetivo))
                {
                        var meta = cuenta.CrearMeta(nombreMeta,
objetivo);
                            Console.WriteLine($"Meta #{meta.Id}
'{meta.Nombre}' creada.");
                }
            }
            else if (opcionMeta == 2)
```

```
            {
                Console.Write("ID de la meta: ");
                  if (int.TryParse(Console.ReadLine(), out int
idMeta))
                {
                      var meta = cuenta.Metas.Find(m => m.Id ==
idMeta);
                    if (meta != null)
                    {
                        Console.Write($"Monto a abonar: $");
                                                                if
(double.TryParse(Console.ReadLine(), out double abono))
                        {
                            if (cuenta.Saldo >= abono)
                            {
                                cuenta.Saldo -= abono;
                                meta.Abonar(abono);
                                     Console.WriteLine($" Abono
realizado. Progreso: {meta.PorcentajeProgreso:F1}%");
                            }
                            else
                            {
                                       Console.WriteLine("Saldo
insuficiente.");
                            }
                        }
                    }
                    else
                    {
```

```
                                    Console.WriteLine("Meta no
encontrada.");
                    }
                }
            }
        }
        _vista.Pausar();
    }

    private void GenerarCodigoRetiro(CuentaBancaria cuenta)
    {
        Console.Clear();
            Console.WriteLine("═══ GENERAR CÓDIGO RETIRO SIN
TARJETA ═══\n");

        Console.Write("Monto a retirar: $");
           if (double.TryParse(Console.ReadLine(), out double
monto))
        {
                            if (cuenta.Saldo >= monto &&
cuenta.PuedeRetirarSinTarjeta())
            {
                                        string codigo =
GestorCodigosRetiro.Instancia.GenerarCodigo(cuenta.Usuario,
monto);

                        Console.WriteLine("\nCódigo generado
exitosamente");

Console.WriteLine("╔═══════════════════════════════════════╗");
```

```
                    Console.WriteLine($"║    CÓDIGO: {codigo}
║");

Console.WriteLine("╚═══════════════════════════════════════╝");

                Console.WriteLine($"\nMonto: ${monto:N2}");

                Console.WriteLine("Válido por 24 horas");

                     Console.WriteLine("\nUsa este código en
cualquier cajero BBVA");

                   Console.WriteLine("   sin necesidad de tu
tarjeta.");



SistemaAlertas.Instancia.AgregarAlerta(cuenta.Usuario,
$"Código retiro generado: ${monto:N2}");

            }

            else

            {

                 Console.WriteLine("Saldo insuficiente o cuenta
no permite retiros sin tarjeta.");

            }

        }

        else

        {

            Console.WriteLine("Monto inválido.");

        }

        _vista.Pausar();

    }


    private void MostrarHistorial(CuentaBancaria cuenta)

    {
```

```
        Console.Clear();
            Console.WriteLine("═══ HISTORIAL DE TRANSACCIONES ═══\n");

                                        var transacciones = cuenta.ObtenerUltimasTransacciones(15);

        if (transacciones.Count == 0)
        {
            Console.WriteLine("No hay transacciones.");
        }
        else
        {
            foreach (var t in transacciones)
            {
                Console.WriteLine(t.ToString());
            }
        }
        _vista.Pausar();
    }

    private void MostrarAlertas(string usuario)
    {
        Console.Clear();
              Console.WriteLine("═══ ALERTAS Y NOTIFICACIONES ═══\n");

                                          var alertas = SistemaAlertas.Instancia.ObtenerAlertas(usuario);
```

```
        if (alertas.Count == 0)
        {
            Console.WriteLine("No tienes alertas.");
        }
        else
        {
            foreach (var alerta in alertas)
            {
                Console.WriteLine($" {alerta}");
            }

            Console.Write("\n¿Limpiar alertas? (S/N): ");
            if (Console.ReadLine().ToUpper() == "S")
            {
SistemaAlertas.Instancia.LimpiarAlertas(usuario);
                Console.WriteLine(" Alertas eliminadas.");
            }
        }
        _vista.Pausar();
    }

    private void MostrarDatosCuenta(CuentaBancaria cuenta)
    {
        Console.Clear();
        Console.WriteLine("═══ MIS DATOS BANCARIOS ═══\n");
        Console.WriteLine($"titular: {cuenta.NombreTitular}");
```

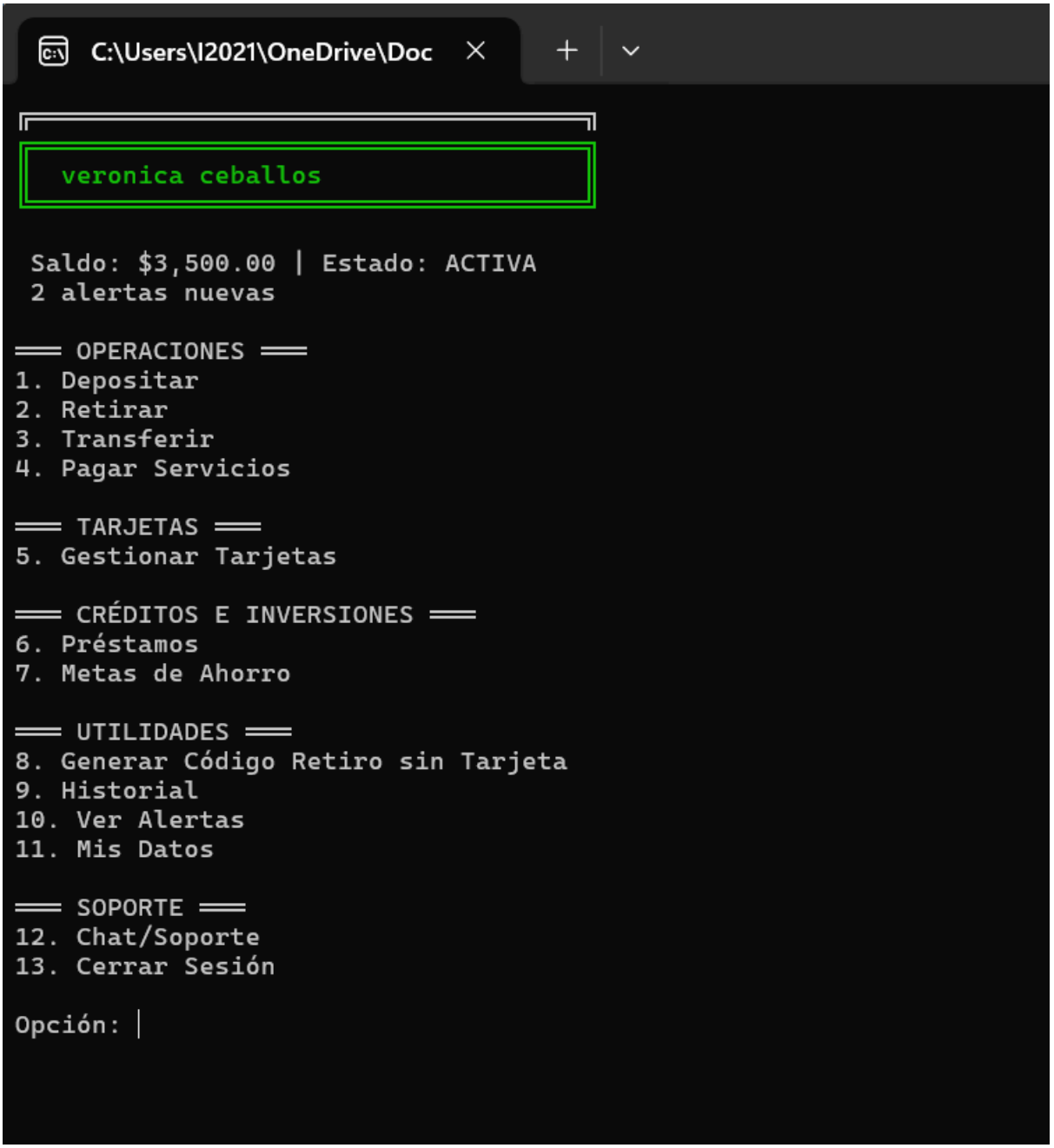
```
                    Console.WriteLine($"Número de cuenta:
{cuenta.NumeroCuenta}");
        Console.WriteLine($"CLABE: {cuenta.CLABE}");
        Console.WriteLine($"Saldo: ${cuenta.Saldo:N2}");
        Console.WriteLine($"Estado: {cuenta.EstadoActual}");
                                Console.WriteLine($"\nTarjetas:
{cuenta.Tarjetas.Count}");
                    Console.WriteLine($"Préstamos activos:
{cuenta.Prestamos.Count(p => p.Estado !=
EstadoPrestamo.Completado)}");
                    Console.WriteLine($"Metas de ahorro:
{cuenta.Metas.Count(m => m.Estado ==
EstadoMeta.EnProgreso)}");
        _vista.Pausar();
    }

    private void MostrarSoporte()
    {
        Console.Clear();
        Console.WriteLine("═══ CHAT DE SOPORTE ═══\n");
         Console.WriteLine(" ¡Hola! Soy el asistente virtual de
BBVA.");
        Console.WriteLine("\n También puedes contactarnos:");
        Console.WriteLine("   Teléfono: 55-2262-6391");
        Console.WriteLine("   WhatsApp: 55-2262-6391");
        Console.WriteLine("   Chat en línea: www.bbva.mx");
        Console.WriteLine("\nHorario de atención:");
         Console.WriteLine("   Lunes a Viernes: 8:00 AM - 8:00
PM");
        Console.WriteLine("   Sábados: 9:00 AM - 3:00 PM");
```

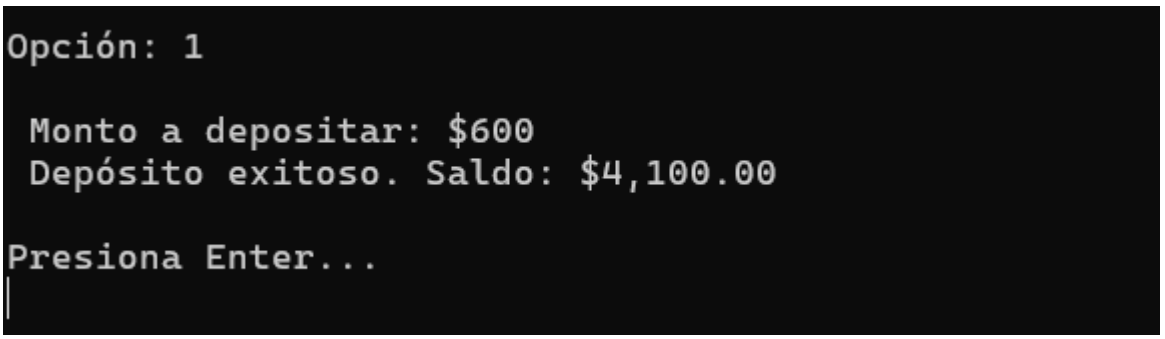
```
        _vista.Pausar();
    }
}
#endregion
#region PROGRAMA PRINCIPAL
class Program
{
    static void Main(string[] args)
    {
        Console.OutputEncoding = System.Text.Encoding.UTF8;
        ISistemaSeguridad seguridad = new SeguridadBasica();
        seguridad = new SeguridadConIntentos(seguridad);
        seguridad = new SeguridadConToken(seguridad);

        var vista = new VistaBanco();
            var controlador = new ControladorBanco(seguridad,
vista);

             seguridad.RegistrarUsuario("juan", "1234", "Juan
Pérez", 5000);
            seguridad.RegistrarUsuario("maria", "5678", "María
González", 3000);
          seguridad.RegistrarUsuario("carlos", "9012", "Carlos
Ramírez", 7500);

        Console.Clear();
         Console.WriteLine("Sistema inicializado con 3 usuarios
de prueba:");
        Console.WriteLine("   • juan / 1234");
```

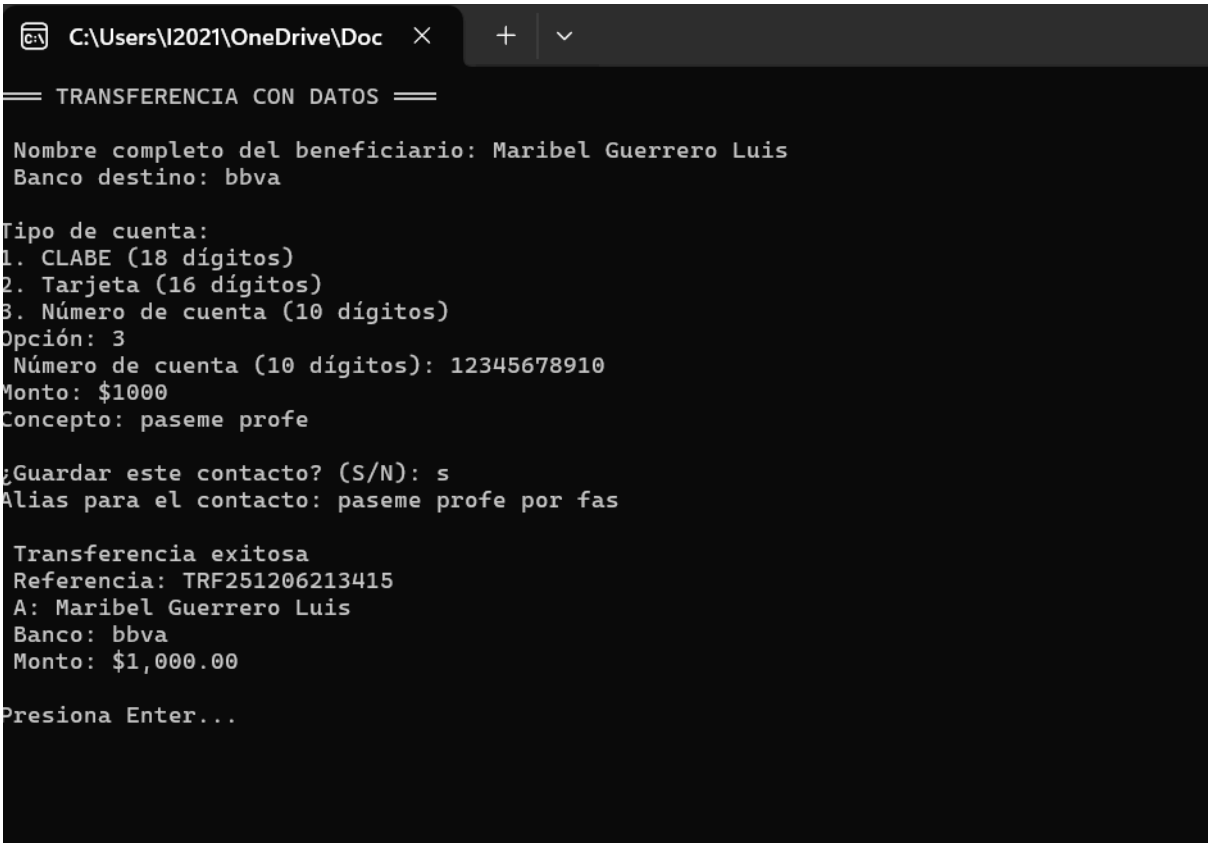
```
        Console.WriteLine("   • maria / 5678");
        Console.WriteLine("   • carlos / 9012");
                    Console.WriteLine("\nPresiona  Enter  para
continuar...");
        Console.ReadLine();

        int opcion;
        do
        {
            vista.MostrarMenuPrincipal();
             if (!int.TryParse(Console.ReadLine(), out opcion))
opcion = 0;

            switch (opcion)
            {
                case 1:
                    controlador.Registrar();
                    break;
                case 2:
                    controlador.IniciarSesion();
                    break;
                case 3:
                    controlador.RetiroConCodigo();
                    break;
                case 4:
                    Console.WriteLine("\n ¡Hasta pronto!");
                    Thread.Sleep(1500);
                    break;
```

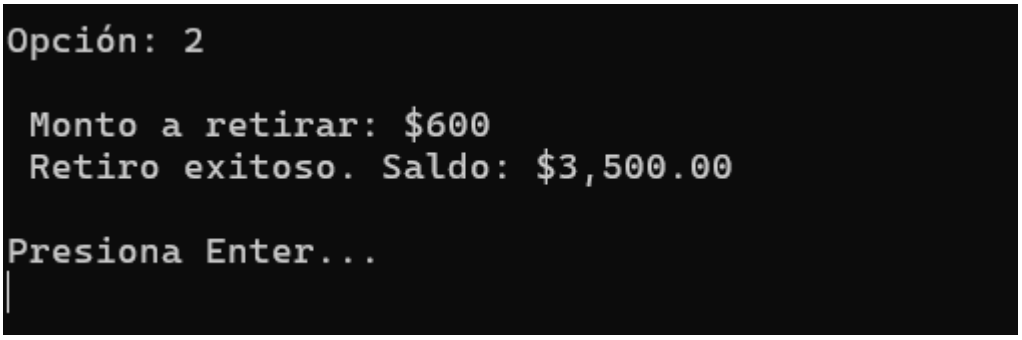
```
                default:
                    Console.WriteLine("Opción inválida.");
                    Thread.Sleep(1000);
                    break;
            }
        } while (opcion !=4);
    }
}
#endregion
```

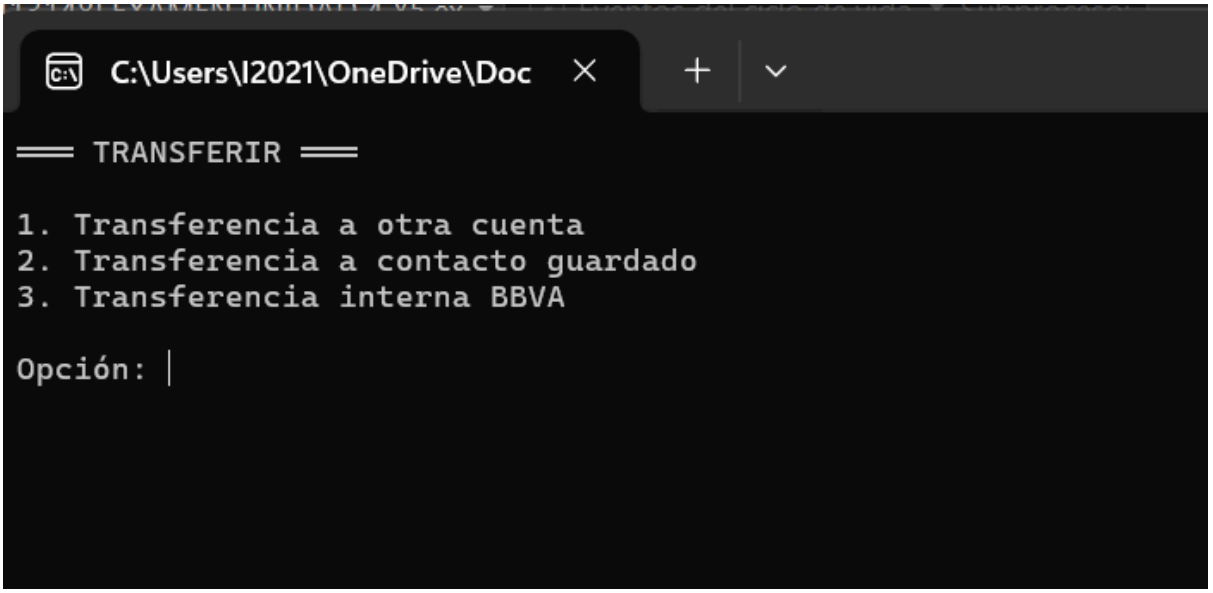
CAPTURAS DEL PROGRAMA

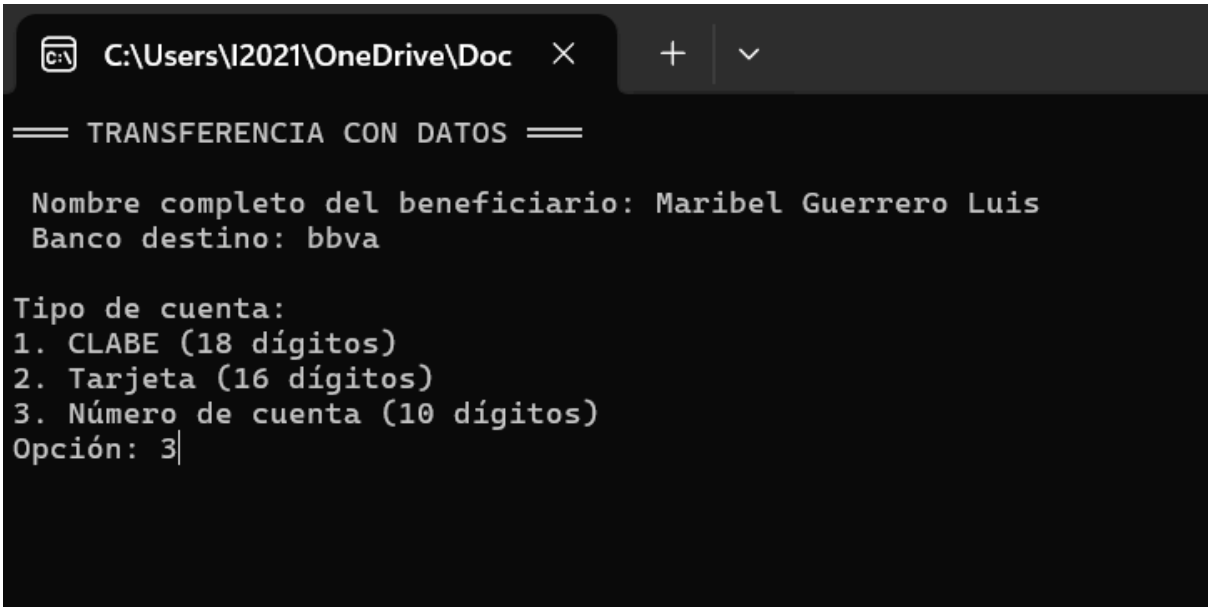

C:\Users\I2021\OneDrive\Doc
veronica ceballos
Saldo: $3,500.00 | Estado: ACTIVA
2 alertas nuevas
OPERACIONES
1. Depositar
2. Retirar
3. Transferir
4. Pagar Servicios
TARJETAS
5. Gestionar Tarjetas
CRÉDITOS E INVERSIONES
6. Préstamos
7. Metas de Ahorro
UTILIDADES
8. Generar Código Retiro sin Tarjeta
9. Historial
10. Ver Alertas
11. Mis Datos
SOPORTE
12. Chat/Soporte
13. Cerrar Sesión
Opción:

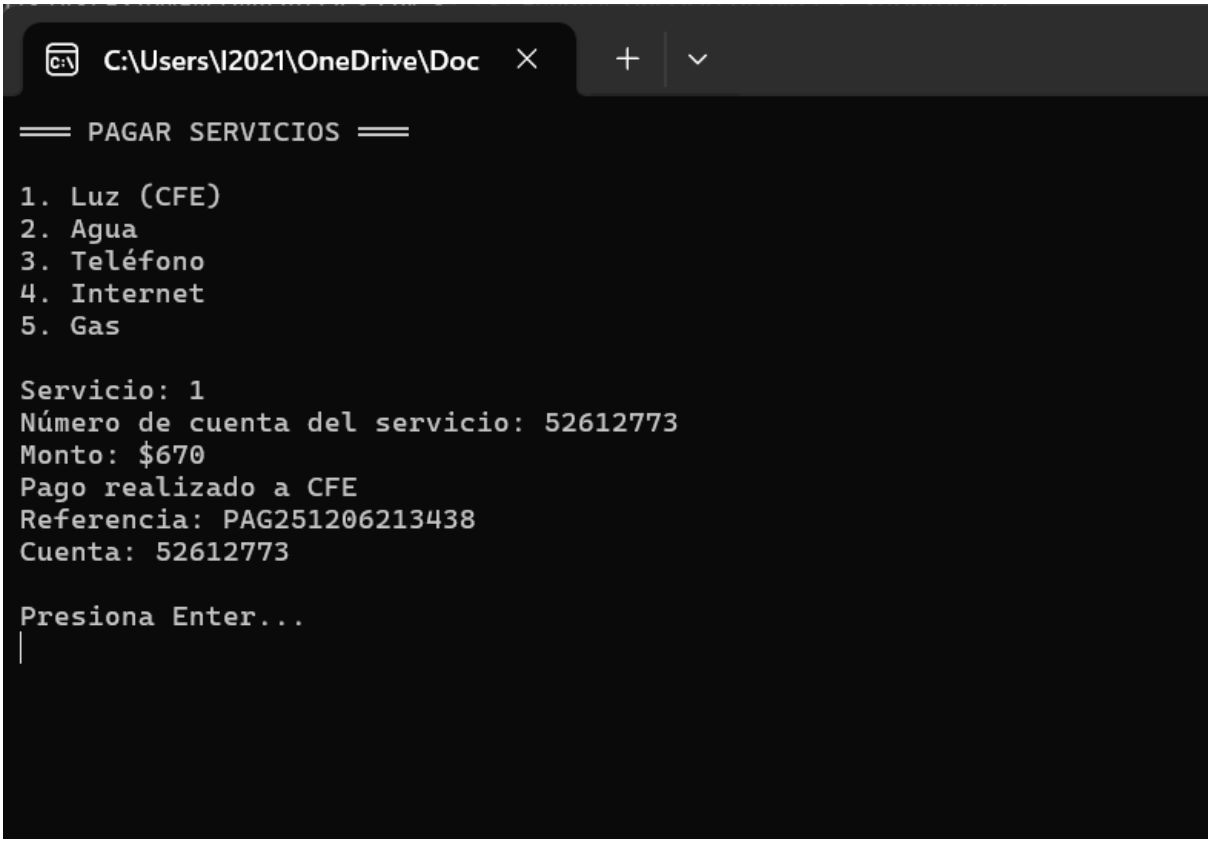

Opción: 1
Monto a depositar: $600
Depósito exitoso. Saldo: $4,100.00
Presiona Enter...

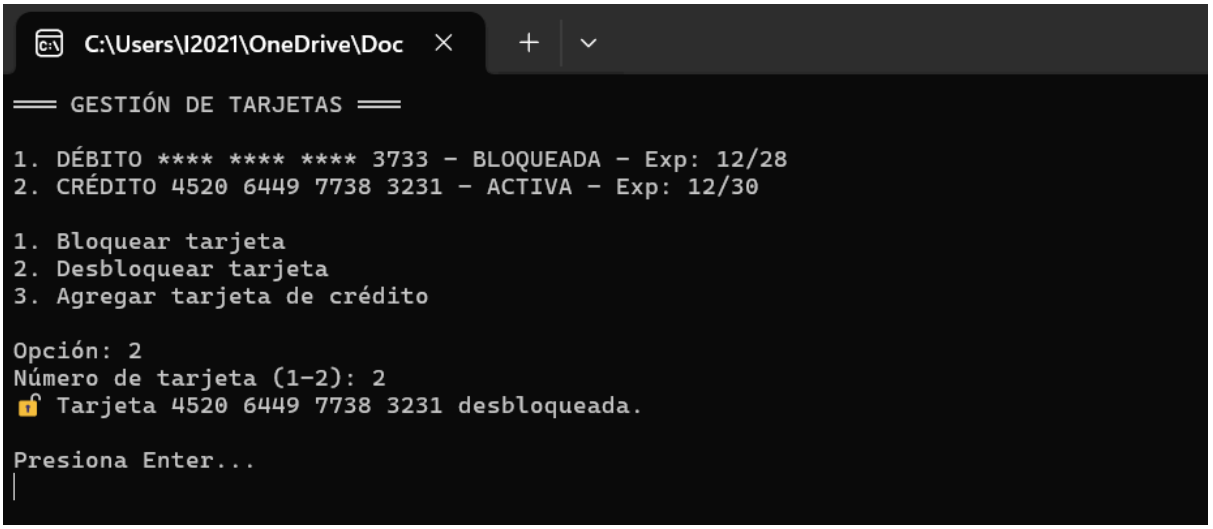

C:\Users\I2021\OneDrive\Doc
TRANSFERENCIA CON DATOS
Nombre completo del beneficiario: Maribel Guerrero Luis
Banco destino: bbva
Tipo de cuenta:
1. CLABE (18 dígitos)
2. Tarjeta (16 dígitos)
3. Número de cuenta (10 dígitos)
Opción: 3
Número de cuenta (10 dígitos): 12345678910
Monto: $1000
Concepto: paseme profe
¿Guardar este contacto? (S/N): s
Alias para el contacto: paseme profe por fas
Transferencia exitosa
Referencia: TRF251206213415
A: Maribel Guerrero Luis
Banco: bbva
Monto: $1,000.00
Presiona Enter...

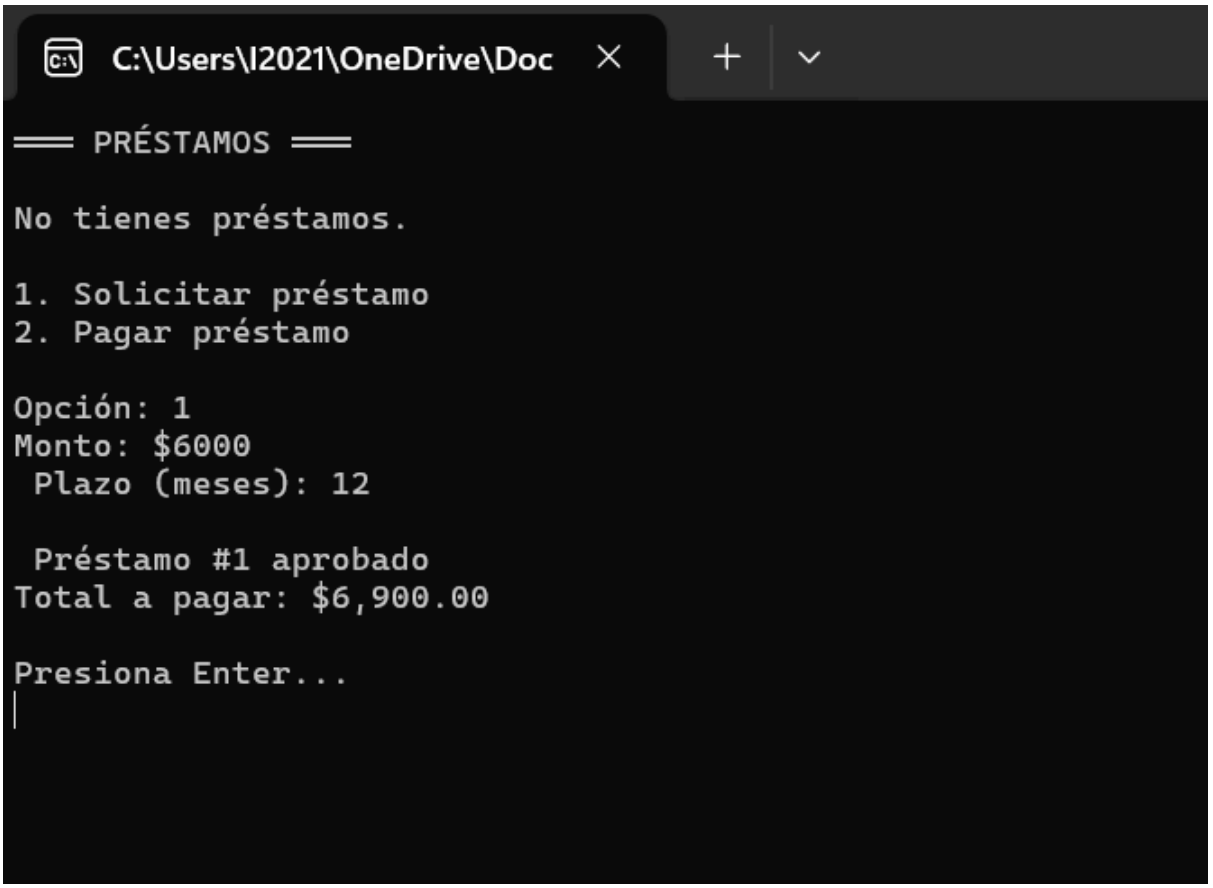

Opción: 2
Monto a retirar: $600
Retiro exitoso. Saldo: $3,500.00
Presiona Enter...

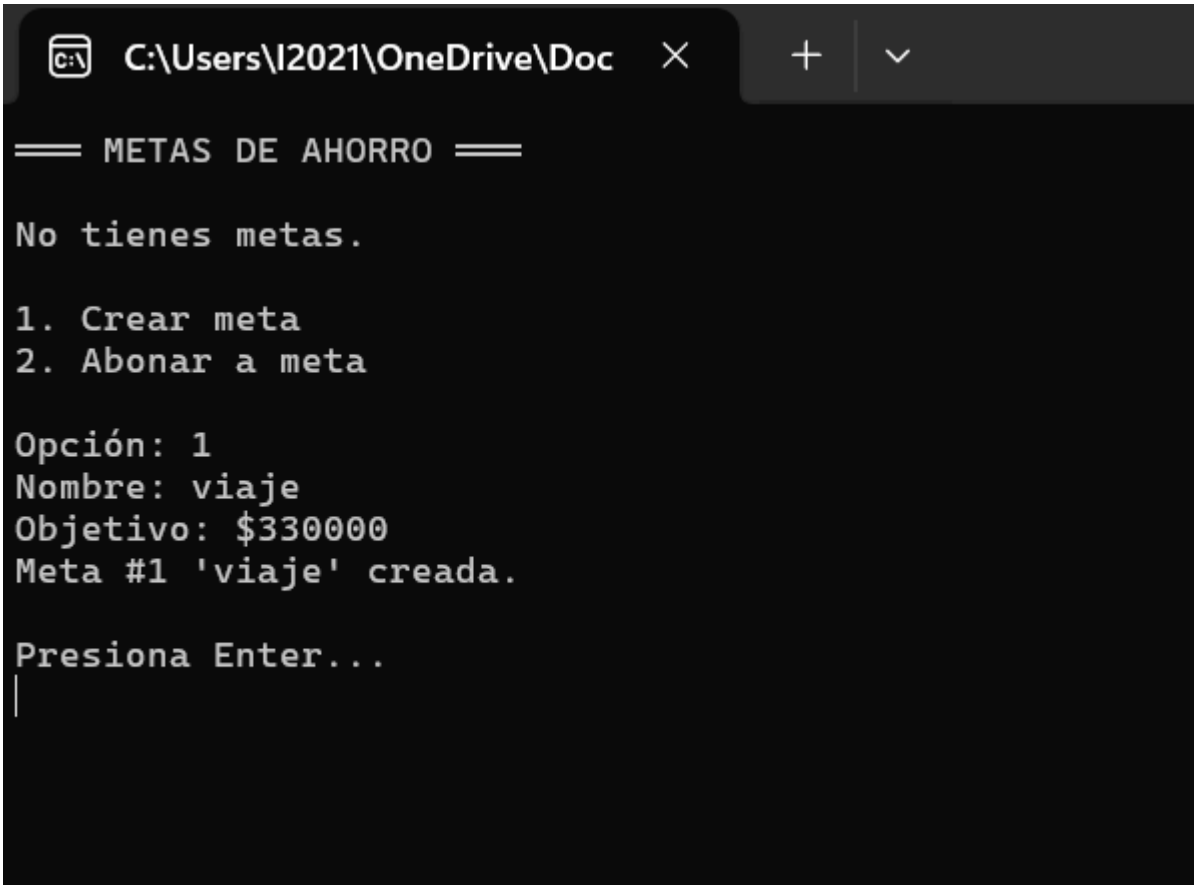

C:\Users\I2021\OneDrive\Doc
TRANSFERIR
1. Transferencia a otra cuenta
2. Transferencia a contacto guardado
3. Transferencia interna BBVA
Opción:


C:\Users\I2021\OneDrive\Doc
=== TRANSFERENCIA CON DATOS ===
Nombre completo del beneficiario: Maribel Guerrero Luis
Banco destino: bbva
Tipo de cuenta:
1. CLABE (18 dígitos)
2. Tarjeta (16 dígitos)
3. Número de cuenta (10 dígitos)
Opción: 3

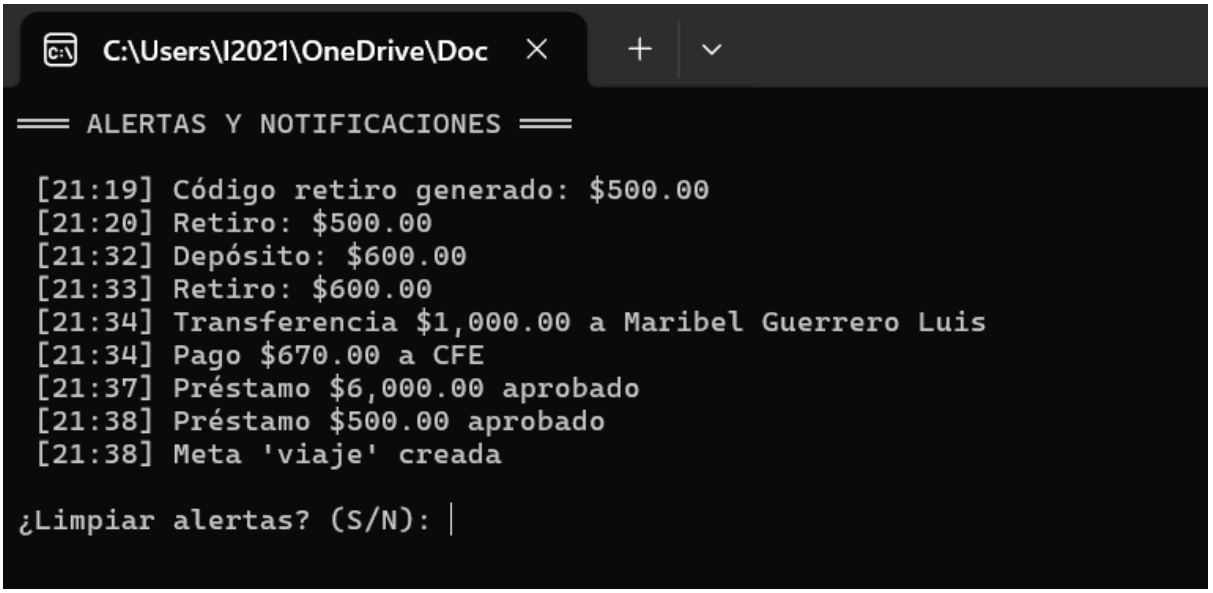

C:\Users\I2021\OneDrive\Doc
=== PAGAR SERVICIOS ===
1. Luz (CFE)
2. Agua
3. Teléfono
4. Internet
5. Gas
Servicio: 1
Número de cuenta del servicio: 52612773
Monto: $670
Pago realizado a CFE
Referencia: PAG251206213438
Cuenta: 52612773
Presiona Enter...

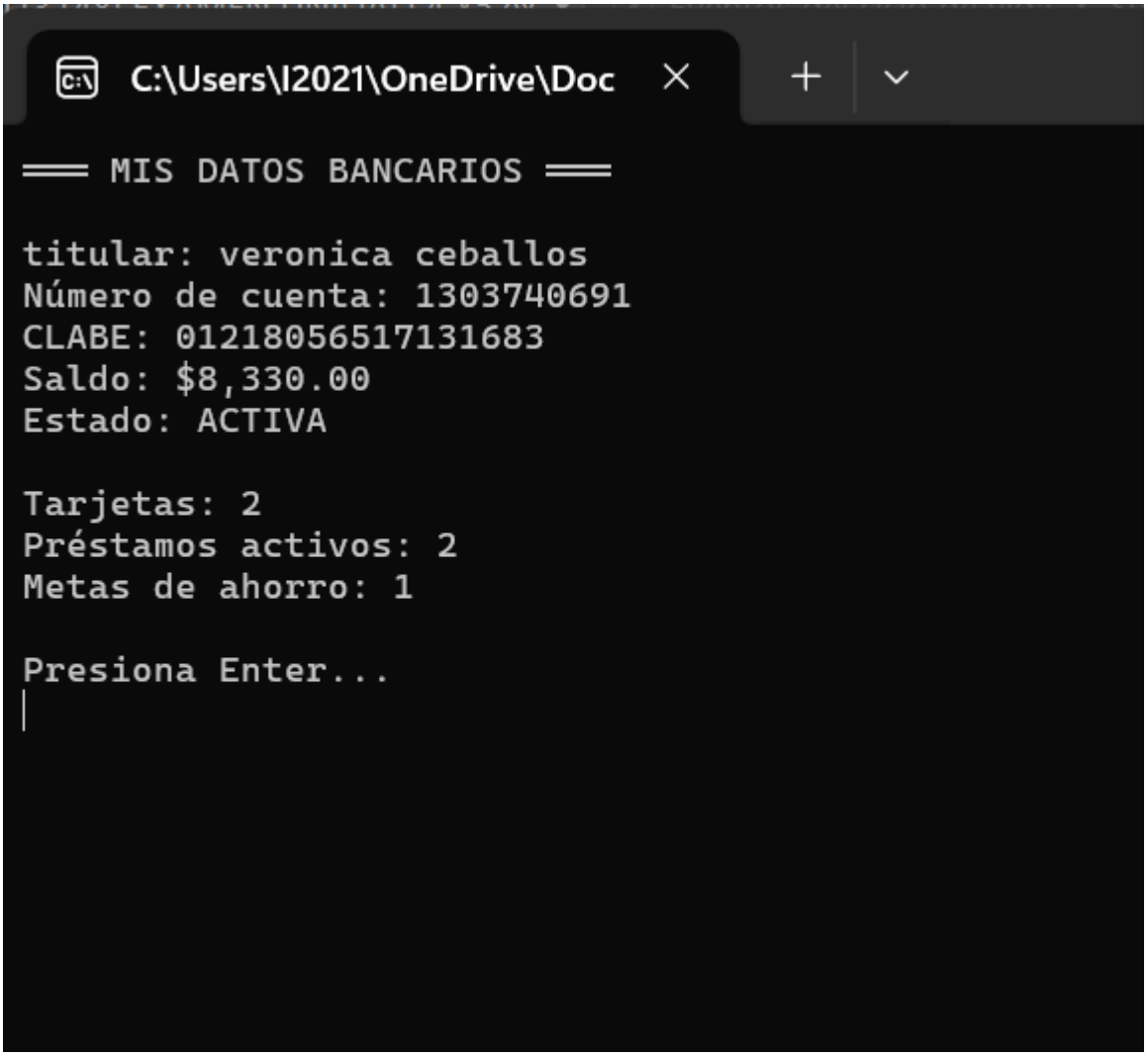

C:\Users\I2021\OneDrive\Doc
GESTIÓN DE TARJETAS
1. DÉBITO **** **** **** 3733 - BLOQUEADA - Exp: 12/28
2. CRÉDITO 4520 6449 7738 3231 - ACTIVA - Exp: 12/30
1. Bloquear tarjeta
2. Desbloquear tarjeta
3. Agregar tarjeta de crédito
Opción: 2
Número de tarjeta (1-2): 2
Tarjeta 4520 6449 7738 3231 desbloqueada.
Presiona Enter...

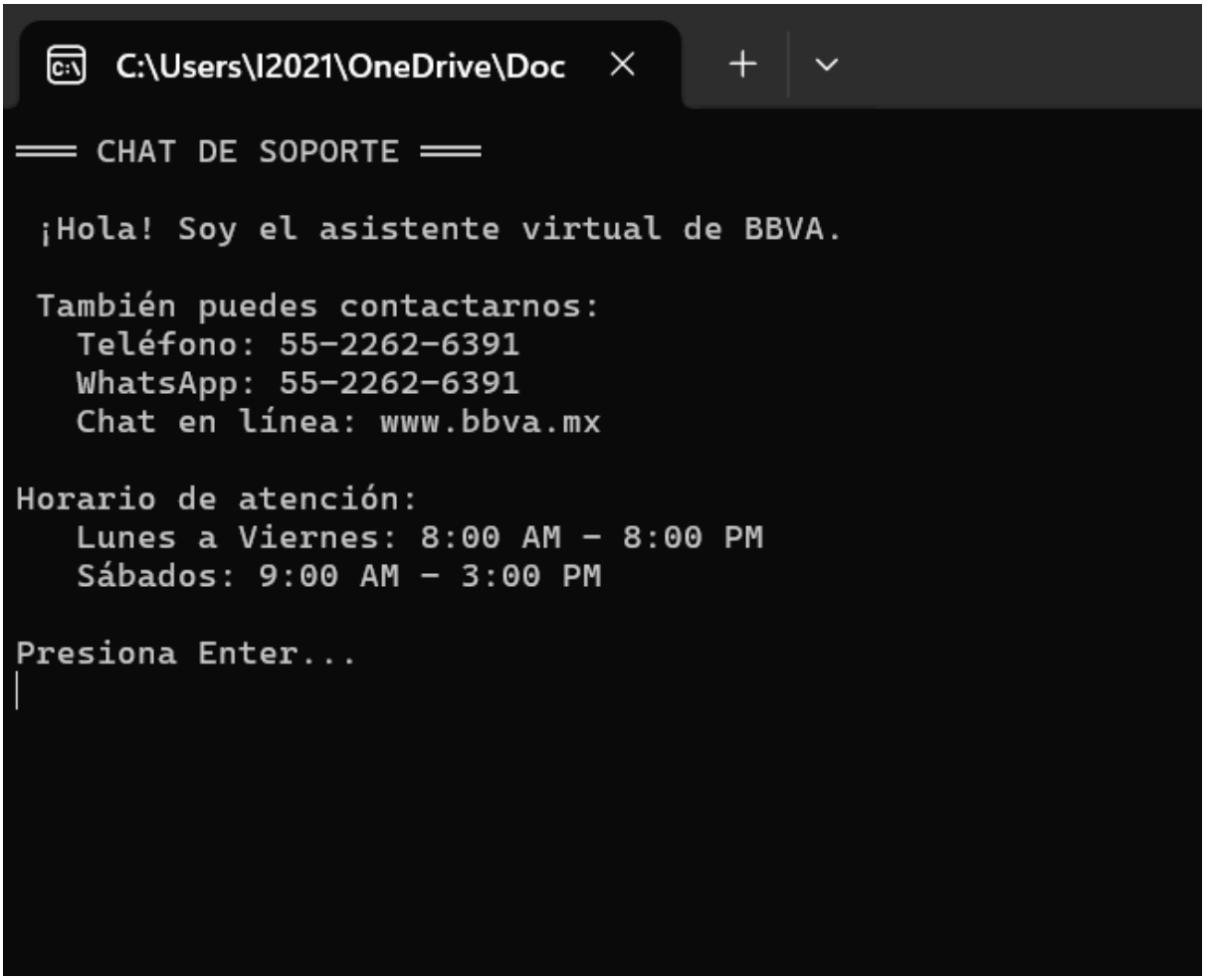

C:\Users\I2021\OneDrive\Doc
PRÉSTAMOS
No tienes préstamos.
1. Solicitar préstamo
2. Pagar préstamo
Opción: 1
Monto: $6000
Plazo (meses): 12
Préstamo #1 aprobado
Total a pagar: $6,900.00
Presiona Enter...

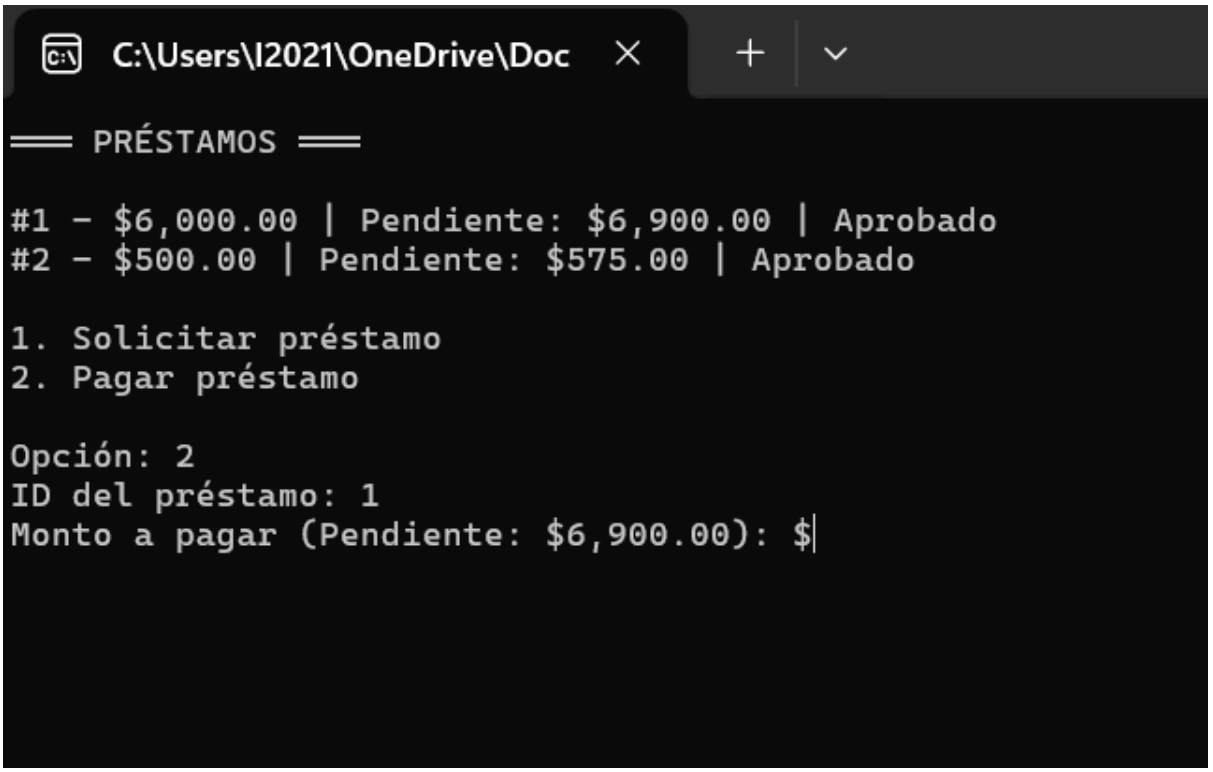

C:\Users\I2021\OneDrive\Doc
═══ METAS DE AHORRO ═══
No tienes metas.
1. Crear meta
2. Abonar a meta
Opción: 1
Nombre: viaje
Objetivo: $330000
Meta #1 'viaje' creada.
Presiona Enter...

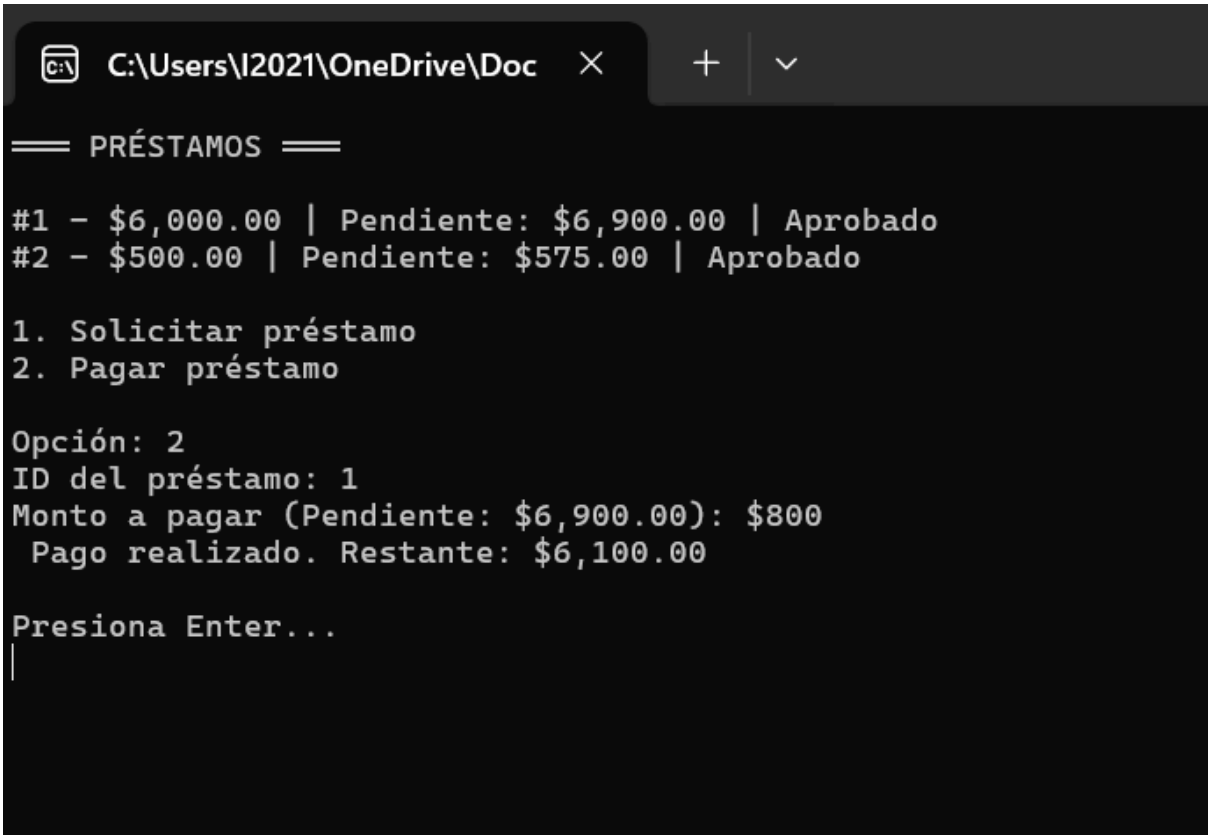

C:\Users\I2021\OneDrive\Doc
═══ HISTORIAL DE TRANSACCIONES ═══
[06/12 21:34] PAGO SERVICIO: $670.00 | Ref: PAG251206213438 | $2,500.00 → $1,830.00
[06/12 21:34] TRANSFERENCIA: $1,000.00 → Maribel Guerrero Luis – bbva | Ref: TRF251206213415 | $3,500.00 → $2,500.00
[06/12 21:33] RETIRO: $600.00 | $4,100.00 → $3,500.00
[06/12 21:32] DEPÓSITO: $600.00 | $3,500.00 → $4,100.00
[06/12 21:20] RETIRO: $500.00 | $4,000.00 → $3,500.00
Presiona Enter...

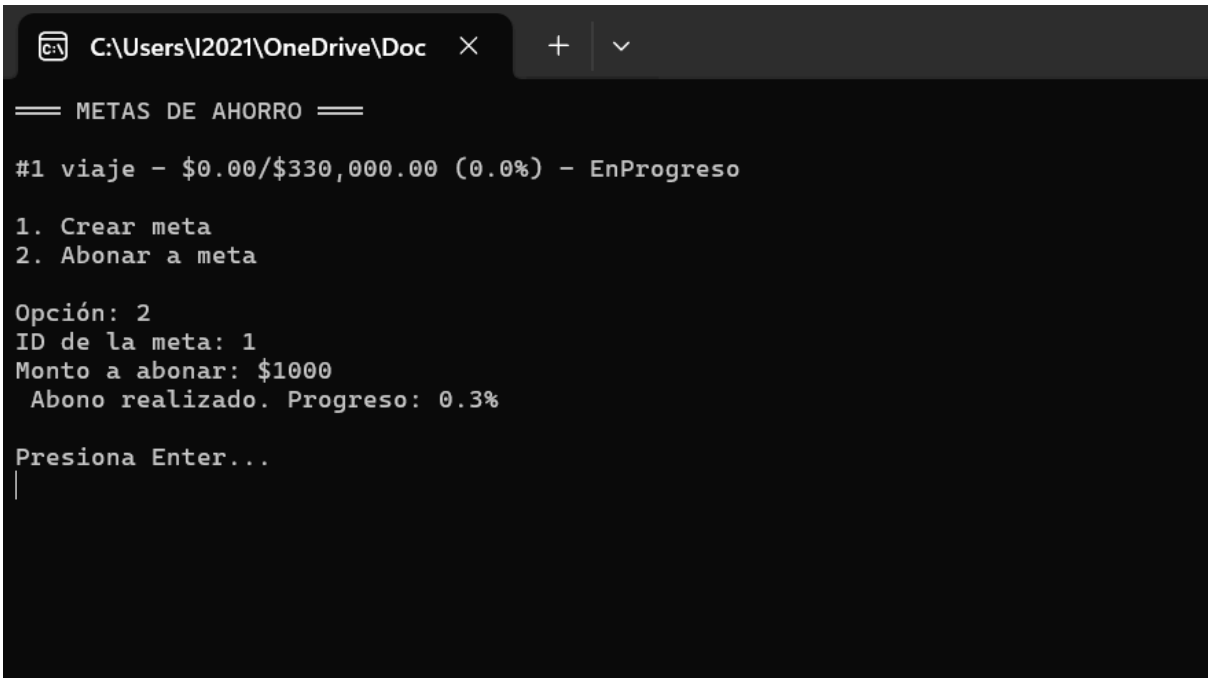

C:\Users\I2021\OneDrive\Doc
═══ ALERTAS Y NOTIFICACIONES ═══
[21:19] Código retiro generado: $500.00
[21:20] Retiro: $500.00
[21:32] Depósito: $600.00
[21:33] Retiro: $600.00
[21:34] Transferencia $1,000.00 a Maribel Guerrero Luis
[21:34] Pago $670.00 a CFE
[21:37] Préstamo $6,000.00 aprobado
[21:38] Préstamo $500.00 aprobado
[21:38] Meta 'viaje' creada
¿Limpiar alertas? (S/N):

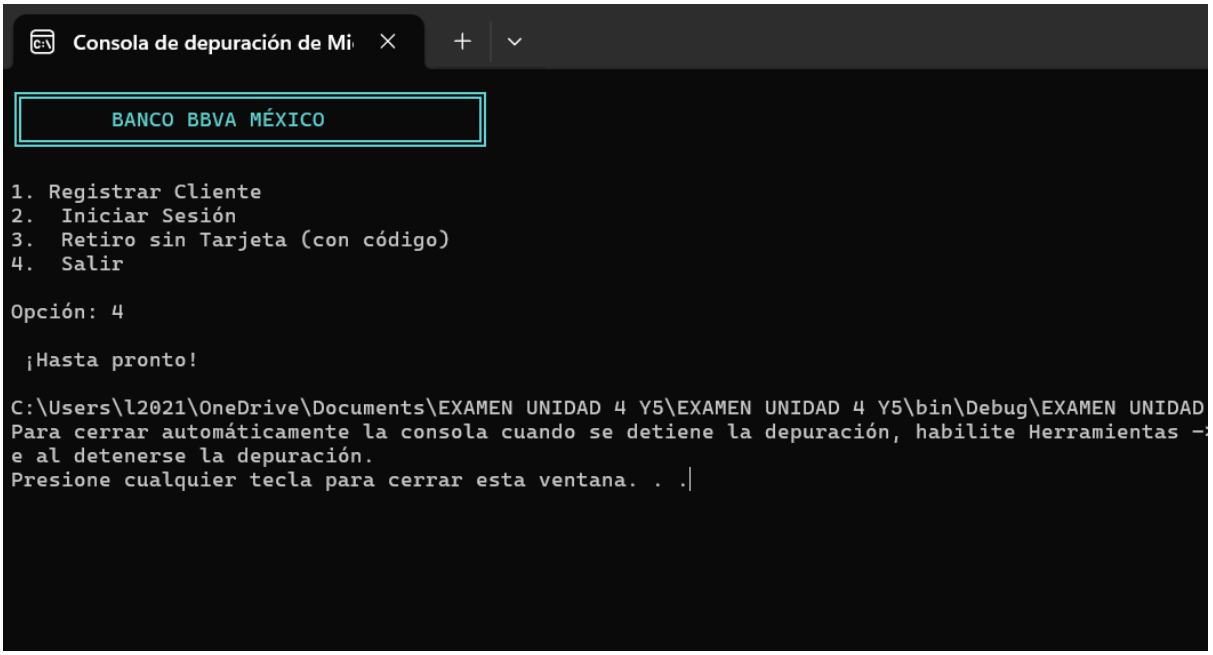

C:\Users\I2021\OneDrive\Doc
═══ MIS DATOS BANCARIOS ═══
titular: veronica ceballos
Número de cuenta: 1303740691
CLABE: 012180565171316 83
Saldo: $8,330.00
Estado: ACTIVA
Tarjetas: 2
Préstamos activos: 2
Metas de ahorro: 1
Presiona Enter...


C:\Users\I2021\OneDrive\Doc
=== CHAT DE SOPORTE ===
¡Hola! Soy el asistente virtual de BBVA.
También puedes contactarnos:
Teléfono: 55-2262-6391
WhatsApp: 55-2262-6391
Chat en línea: www.bbva.mx
Horario de atención:
Lunes a Viernes: 8:00 AM - 8:00 PM
Sábados: 9:00 AM - 3:00 PM
Presiona Enter...


C:\Users\I2021\OneDrive\Doc
=== PRÉSTAMOS ===
#1 - $6,000.00 | Pendiente: $6,900.00 | Aprobado
#2 - $500.00 | Pendiente: $575.00 | Aprobado
1. Solicitar préstamo
2. Pagar préstamo
Opción: 2
ID del préstamo: 1
Monto a pagar (Pendiente: $6,900.00): $


C:\Users\I2021\OneDrive\Doc
═══ PRÉSTAMOS ═══
#1 - $6,000.00 | Pendiente: $6,900.00 | Aprobado
#2 - $500.00 | Pendiente: $575.00 | Aprobado
1. Solicitar préstamo
2. Pagar préstamo
Opción: 2
ID del préstamo: 1
Monto a pagar (Pendiente: $6,900.00): $800
Pago realizado. Restante: $6,100.00
Presiona Enter...


C:\Users\I2021\OneDrive\Doc
═══ METAS DE AHORRO ═══
#1 viaje - $0.00/$330,000.00 (0.0%) - EnProgreso
1. Crear meta
2. Abonar a meta
Opción: 2
ID de la meta: 1
Monto a abonar: $1000
Abono realizado. Progreso: 0.3%
Presiona Enter...



CONCLUSIÓN

El sistema es una aplicación bancaria que permite gestionar cuentas de usuario, tarjetas, transferencias, préstamos, metas de ahorro y operaciones administrativas. logs, alertas, generación/validación de tokens y códigos de retiro. Está organizada con arquitectura MVC, separa responsabilidades en componentes reutilizables y probados. El sistema prioriza seguridad que es: autenticación, control de intentos, tokens, trazabilidad (logs y mementos), y consistencia en entornos concurrentes.

Usuarios típicos: clientes bancarios que realizan depósitos, retiros, transferencias y administran tarjetas; administradores que revisan logs, alertas y gestionan cuentas.

En el siguiente programa se utilizan los siguientes patrones de diseño que son:

- Patron singleton que se encarga de los gestores únicos como lo que es el inicio de sesión, Logs, Tokens, Alertas, Contactos y Códigos
- Patron memento que se encarga de el historial de transacciones
- Patrón estado que usa Estados de cuenta y tarjeta
- Patron decorador haciéndose cargo de el sistema de seguridad de capas
- MVC qué es el se encarga de la vista del usuario, controladores y modelo