UNAH
UNIVERSIDAD NACIONAL
AUTÓNOMA DE HONDURAS

Campus
Comayagua

Ingeniería en Sistemas

IS-410 Programación Orientada a Objetos

Catedrático: Oscar Guillermo Hernández Ramírez

Presentado por: Elmer Eduardo Valenzuela Mejía

Cuenta: 20231001440

Viernes 11 de abril del 2025

Repositorios:

https://github.com/EduardoVal3/front-gtbancarias

https://github.com/EduardoVal3/API-GT-Bancarias

Índice

INTRODUCCIÓN

El presente proyecto tiene como objetivo desarrollar un sistema de Gestión de Transacciones Bancarias utilizando principios de la Programación Orientada a Objetos (POO). Este sistema permite gestionar diferentes áreas clave de una entidad bancaria, como préstamos, recursos humanos, tarjetas de crédito y transacciones bancarias, además de ofrecer funcionalidades generales para la administración de cuentas de usuarios.

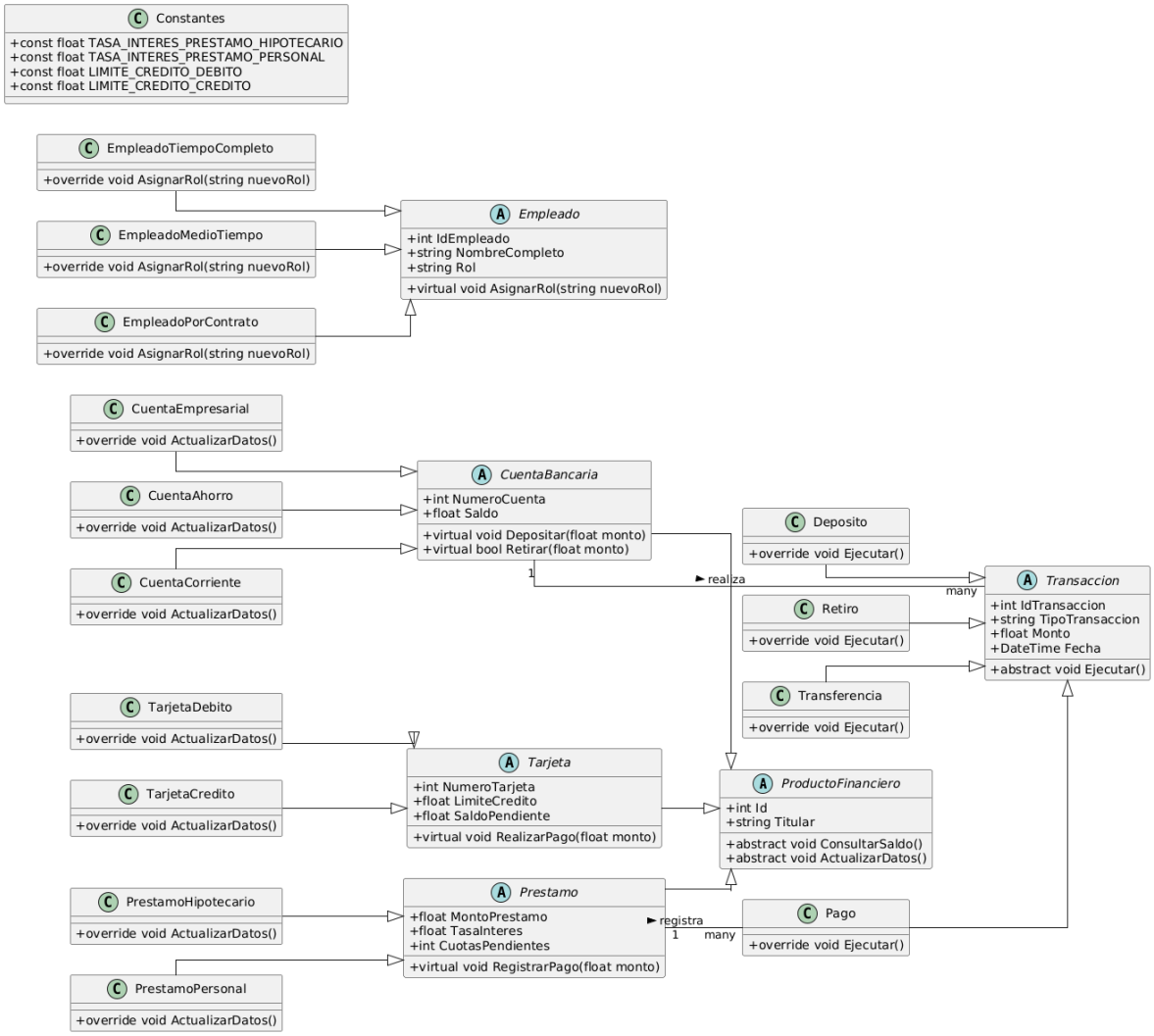
El proyecto se desarrolla en C# y utiliza conceptos de POO, como herencia, clases abstractas, interfaces, polimorfismo y constantes. Además, se implementa un menú interactivo en consola que permite al usuario realizar operaciones como crear cuentas, consultar saldos, depositar y retirar dinero, transferir fondos, y más.

Este documento describe los requerimientos funcionales y técnicos del sistema, así como el diseño de las clases y su interacción mediante un diagrama de clases.

REQUERIMIENTOS FUNCIONALES

- Gestionar Préstamos:
    1) Registrar nuevos préstamos (hipotecarios y personales).
    2) Consultar el saldo pendiente de un préstamo.
    3) Registrar pagos para reducir el saldo pendiente.
    4) Listar todos los préstamos activos.
- Gestionar Recursos Humanos:
    5) Registrar empleados (tiempo completo, medio tiempo y por contrato).
    6) Asignar o modificar roles de empleados.
    7) Listar todos los empleados registrados.
- Gestionar Tarjetas:
    8) Emitir tarjetas de crédito y débito.
    9) Realizar pagos con tarjetas de crédito.
    10) Consultar el saldo pendiente de una tarjeta.
    11) Listar todas las tarjetas emitidas.
- Gestionar Transacciones Bancarias:
    12) Registrar depósitos en cuentas bancarias.
    13) Registrar retiros de cuentas bancarias.
    14) Registrar transferencias entre cuentas.
    15) Listar el historial de transacciones realizadas.
- Gestión General de Cuentas de Usuarios:
    16) Crear cuentas bancarias (ahorro, corriente, empresarial).
    17) Consultar el saldo de una cuenta.
    18) Depositar dinero en una cuenta.
    19) Retirar dinero de una cuenta.
    20) Transferir fondos entre cuentas.
    21) Ver el historial de transacciones de una cuenta.
- Salir del Sistema:
    22) Opción para cerrar el programa de manera segura.

DIAGRAMA DE CLASES



REQUERIMIENTOS TÉCNICOS

Nombre Clase		ProductoFinanciero (Clase Abstracta)
Descripción		Clase base abstracta para representar productos financieros como cuentas bancarias, préstamos y tarjetas.
Atributos		
Nombre	***Tipo***	***Descripción***
Id	int	Identificador único del producto financiero.
Titular	string	Nombre del titular del producto financiero.

Métodos :

- **ConsultarSaldo()** (**abstract**): Método abstracto para consultar el saldo del producto financiero.
- **ActualizarDatos()** (**abstract**): Método abstracto para actualizar los datos del producto financiero.

Nombre Clase		CuentaBancaria (Clase Abstracta)
Descripción		Clase abstracta que hereda de ProductoFinanciero y representa una cuenta bancaria genérica.
Atributos		
Nombre	***Tipo***	***Descripción***
NumeroCuenta	int	Número de cuenta bancaria.
Saldo	float	Saldo disponible en la cuenta.

Métodos :

- **Depositar(float monto)** (**virtual**): Método para depositar dinero en la cuenta.
- **Retirar(float monto)** (**virtual**): Método para retirar dinero de la cuenta.
- **ConsultarSaldo()** (**override**): Implementación específica para consultar el saldo de la cuenta.
- **ActualizarDatos()** (**abstract**): Método abstracto para actualizar los datos de la cuenta.

Subclases de CuentaBancaria:

a. Clase: CuentaAhorro

- **Descripción** : Representa una cuenta de ahorro.
- **Métodos** :
    - **ActualizarDatos()** (**override**): Implementación específica para actualizar los datos de una cuenta de ahorro.

b. Clase: CuentaCorriente

- **Descripción** : Representa una cuenta corriente.
- **Métodos** :
    - **ActualizarDatos()** (**override**): Implementación específica para actualizar los datos de una cuenta corriente.

c. Clase: CuentaEmpresarial

- **Descripción** : Representa una cuenta empresarial.
- **Métodos** :
    - **ActualizarDatos()** (**override**): Implementación específica para actualizar los datos de una cuenta empresarial.

Nombre Clase		Prestamo (Clase Abstracta)
Descripción		Clase abstracta que hereda de ProductoFinanciero y representa un préstamo genérico.
Atributos		
Nombre	***Tipo***	***Descripción***
MontoPrestamo	float	Monto total del préstamo.
TasaInteres	Float	Tasa de interés aplicada al préstamo.
CuaotasPendientes	Int	Número de cuotas pendientes.

Métodos :

- **RegistrarPago(float monto)** (**virtual**): Método para registrar pagos del préstamo.
- **ConsultarSaldo()** (**override**): Implementación específica para consultar el saldo pendiente del préstamo.

Subclases de Prestamo

a. Clase: PrestamoHipotecario

- **Descripción** : Representa un préstamo hipotecario.
- **Métodos** :
    - **ActualizarDatos()** (**override**): Implementación específica para actualizar los datos de un préstamo hipotecario.

b. Clase: PrestamoPersonal

- **Descripción** : Representa un préstamo personal.
- **Métodos** :
    - **ActualizarDatos()** (**override**): Implementación específica para actualizar los datos de un préstamo personal.

Nombre Clase		Empleado (Clase Abstracta)
Descripción		Clase abstracta que representa a un empleado del banco.
Atributos		
Nombre	***Tipo***	***Descripción***
IdEmpleado	int	Identificador único del empleado.
NombreCompleto	string	Nombre del empleado.
Rol	string	Rol asignado al empleado.

- **Métodos** :
    - **AsignarRol(string nuevoRol)** (**virtual**): Método para asignar o modificar el rol del empleado.
    - **ActualizarDatos()** (**abstract**): Método abstracto para actualizar los datos del empleado.

Subclases de Empleado

a. Clase: EmpleadoTiempoCompleto

- **Descripción** : Representa a un empleado de tiempo completo.

- **Métodos** :

- **AsignarRol(string nuevoRol)** (**override**): Implementación específica para asignar roles a empleados de tiempo completo.

b. Clase: EmpleadoMedioTiempo

- **Descripción** : Representa a un empleado de medio tiempo.
- **Métodos** :
    - **AsignarRol(string nuevoRol)** (**override**): Implementación específica para asignar roles a empleados de medio tiempo.

c. Clase: EmpleadoPorContrato

- **Descripción** : Representa a un empleado por contrato.
- **Métodos** :
    - **AsignarRol(string nuevoRol)** (**override**): Implementación específica para asignar roles a empleados por contrato.

Nombre Clase		Tarjeta (Clase Abstracta)
Descripción		Clase abstracta que hereda de ProductoFinanciero y representa una tarjeta genérica.
Atributos		
Nombre	***Tipo***	***Descripción***
NumeroTarjeta	float	Número de la tarjeta.
LimiteCredito	float	Límite de crédito de la tarjeta.
SaldoPendiente	float	Saldo pendiente de la tarjeta.

Métodos :

- **RealizarPago(float monto)** (**virtual**): Método para realizar pagos con la tarjeta.
- **ConsultarSaldo()** (**override**): Implementación específica para consultar el saldo pendiente de la tarjeta.

10. Subclases de Tarjeta

a. Clase: TarjetaCredito

- **Descripción** : Representa una tarjeta de crédito.
- **Métodos** :
    - **ActualizarDatos()** (**override**): Implementación específica para actualizar los datos de una tarjeta de crédito.

b. Clase: TarjetaDebito

- **Descripción** : Representa una tarjeta de débito.
- **Métodos** :
    - **ActualizarDatos()** (**override**): Implementación específica para actualizar los datos de una tarjeta de débito.

Nombre Clase		Transaccion (Clase Abstracta)
Descripción		Clase abstracta que representa una transacción genérica.
Atributos		
Nombre	***Tipo***	***Descripción***
IdTransaccion	int	Identificador único de la transacción.
TipoTransaccion	String	Tipo de transacción (depósito, retiro, transferencia, pago).
Monto	Float	Monto de la transacción.
Fecha	DateTime	Fecha de la transacción.

- **Métodos** :
    - **Ejecutar()** (**abstract**): Método abstracto para ejecutar la transacción.

Subclases de Transaccion

a. Clase: Deposito

- **Descripción** : Representa un depósito en una cuenta bancaria.
- **Métodos** :
    - **Ejecutar()** (**override**): Implementación específica para ejecutar un depósito.

b. Clase: Retiro

- **Descripción** : Representa un retiro de una cuenta bancaria.
- **Métodos** :
    - **Ejecutar()** (**override**): Implementación específica para ejecutar un retiro.

c. Clase: Transferencia

- **Descripción** : Representa una transferencia entre cuentas bancarias.
- **Métodos** :
    - **Ejecutar()** (**override**): Implementación específica para ejecutar una transferencia.

d. Clase: Pago

- **Descripción** : Representa un pago de un préstamo.
- **Métodos** :
    - **Ejecutar()** (**override**): Implementación específica para ejecutar un pago.

Nombre Clase		Constantes
Descripción		Clase estática que contiene valores constantes utilizados en el sistema.
Atributos		
Nombre	***Tipo***	***Descripción***
TASA_INTERES_PRESTAMO_HIPOTECARIO	const float	Tasa de interés para préstamos hipotecarios.
TASA_INTERES_PRESTAMO_PERSONAL	const float	Tasa de interés para préstamos personales.
LIMITE_CREDITO_DEBITO	const float	Límite máximo de crédito para tarjetas de débito.
LIMITE_CREDITO_CREDITO	const float	Límite máximo de crédito para tarjetas de crédito.

-Incorporación de Herencia

- La herencia se utiliza para organizar las clases en una jerarquía lógica:
    - **CuentaBancaria**, **Prestamo** y **Tarjeta** heredan de **ProductoFinanciero**.
    - **CuentaAhorro**, **CuentaCorriente** y **CuentaEmpresarial** heredan de **CuentaBancaria**.
    - **PrestamoHipotecario** y **PrestamoPersonal** heredan de **Prestamo**.
    - **EmpleadoTiempoCompleto**, **EmpleadoMedioTiempo** y **EmpleadoPorContrato** heredan de **Empleado**.
    - **TarjetaCredito** y **TarjetaDebito** heredan de **Tarjeta**.

-Incorporación de Polimorfismo

- El polimorfismo se aplica mediante métodos abstractos y virtuales:
    - Los métodos **ConsultarSaldo()** y **ActualizarDatos()** son abstractos en las clases base y se sobrescriben en las subclases.
    - Los métodos **Ejecutar()** en las subclases de **Transaccion** permiten comportamientos específicos según el tipo de transacción.

-Uso de Listas

- Se utilizan listas (**List<T>**) para administrar objetos:
    - **List<CuentaBancaria>** para gestionar cuentas bancarias.
    - **List<Prestamo>** para gestionar préstamos.
    - **List<Empleado>** para gestionar empleados.
    - **List<Tarjeta>** para gestionar tarjetas.
    - **List<Transaccion>** para gestionar transacciones.