Informe sobre el Sistema Bancario

Hecho por: Rodrigo González, Victor Hernández y Eduardo Estefanía

Link del repositorio: https://github.com/EduardoEO/Proyecto_Sist_Bancario

Este proyecto consiste en el desarrollo de un sistema bancario básico programado en C++, orientado a la gestión de usuarios, cuentas bancarias, tarjetas y transacciones. El objetivo principal es simular las operaciones esenciales que realiza un banco, como el registro e inicio de sesión de clientes, la consulta de saldos, movimientos entre cuentas, y la gestión de datos mediante archivos de texto estructurados.

El proyecto ha sido diseñado con un enfoque modular, utilizando programación orientada a objetos, lo que permite una clara separación de responsabilidades entre clases como Usuario, CuentaBancaria, Transaccion, Menu o GestorArchivos. Además, el código está estructurado de forma que facilita su ampliación y mantenimiento.

Se ha seguido una metodología de desarrollo ágil, trabajando de forma iterativa y en equipo, validando progresivamente cada funcionalidad. Esto ha permitido una mejora continua del producto, adaptándose a los requisitos definidos en cada fase del proyecto. El resultado es una aplicación sencilla, funcional y extensible, ideal como base para futuros desarrollos más complejos.

Funcionalidades:

A lo largo del proyecto hemos ido añadiendo funcionalidades necesarias para el correcto funcionamiento del código pero también hemos tenido que desechar otras funcionalidades secundarias.

Dividiremos las funcionalidades en los siguientes apartados principales marcado en negrita:

Gestión de Usuarios

- Registro de Usuarios:
  - Permite a un usuario no registrado registrarse en el sistema proporcionando su nombre, DNI y contraseña. Implementado en Menu::mostrarMenuRegistro y Banco::registrarUsuario.

- Inicio de Sesión:
  - Permite a un usuario registrado iniciar sesión con su DNI y contraseña. Implementado en Menu::iniciarSesion y Banco::iniciarSesion.

- Gestión de Contraseña:
  - Permite a un usuario cambiar su contraseña. Implementado en Menu::cambiarContrasena.

- Eliminación de Usuario:

    - Permite a un usuario eliminar su cuenta. Implementado en Banco::eliminarUsuario.

Gestión de Cuentas Bancarias

- Creación de Cuentas Bancarias:
    - Permite a un usuario crear nuevas cuentas bancarias. Implementado en Banco::crearCuentaBancaria.

- Fusión de Cuentas Bancarias:
    - Permite fusionar dos cuentas bancarias de un mismo usuario. Implementado en Menu::fusionarCuentasBancarias.

- Eliminación de Cuentas Bancarias:
    - Permite eliminar cuentas bancarias. Implementado en Banco::eliminarCuenta.

Gestión de Tarjetas Bancarias

- Creación de Tarjetas Bancarias:
    - Permite crear tarjetas bancarias asociadas a una cuenta. Implementado en Banco::crearTarjetaBancaria.

- Gestión de Tarjetas:
    - Permite activar, bloquear, cambiar el PIN, renovar, y mostrar detalles de las tarjetas. Implementado en Menu::mostrarMenuTarjetaBancaria.

- Renovación de Tarjetas:
    - Permite renovar tarjetas caducadas, se comprueba cada vez que se accede al menú de tarjetas. Implementado en Menu::renovarTarjetaBancaria.

Gestión de Transacciones

- Depósitos:
    - Permite depositar dinero en una cuenta bancaria. Implementado en Banco::depositar.

- Retiros:
    - Permite retirar dinero de una cuenta bancaria. Implementado en Banco::retirar.

- Transferencias:
    - Permite realizar transferencias entre cuentas bancarias. Implementado en Banco::transferencia.

- Historial de Transacciones:
    - Permite visualizar y guardar el historial de transacciones de una cuenta. Implementado en Menu::mostrarHistorialTransacciones.

Gestión de Datos

- Carga de Datos desde Archivos:

    - Carga usuarios, cuentas, transacciones y tarjetas desde archivos al iniciar el programa. Implementado en Banco::cargarDatosDesdeArchivos.

- Guardado de Datos en Archivos:
    - Guarda usuarios, cuentas, transacciones y tarjetas en archivos al salir del programa. Implementado en Banco::guardarDatos.

Interfaz de Usuario

- Menús Interactivos:
    - Menús para iniciar sesión, registrarse, gestionar cuentas, tarjetas y transacciones. Implementado en Menu.

Uso:

El programa es un sistema bancario que permite a los usuarios gestionar cuentas bancarias, tarjetas y realizar transacciones de manera interactiva. Está diseñado para ser ejecutado en la consola.

Al iniciar el programa, se cargan los datos almacenados previamente (usuarios, cuentas, transacciones, etc.) desde archivos. Esto se realiza mediante el método Banco::cargarDatosDesdeArchivos().

El programa muestra un menú inicial donde los usuarios pueden: Iniciar sesión si ya están registrados. Registrarse como nuevos usuarios. Esto se gestiona con el método Menu::mostrarMenuIniSesion()

Una vez iniciado sesión el usuario tendrá acceso a todas la funcionalidades mencionadas anteriormente según el tipo de cuenta bancaria.

Al salir del programa, todos los datos se guardan automáticamente en archivos mediante el método Banco::guardarDatos(). Esto asegura que los cambios realizados durante la sesión no se pierdan.

Si ocurre un error inesperado, el programa lo captura y muestra un mensaje en la consola: Por ejemplo: "Error inesperado: <detalle del error>". Esto se gestiona con bloques try-catch en el archivo main.cpp.

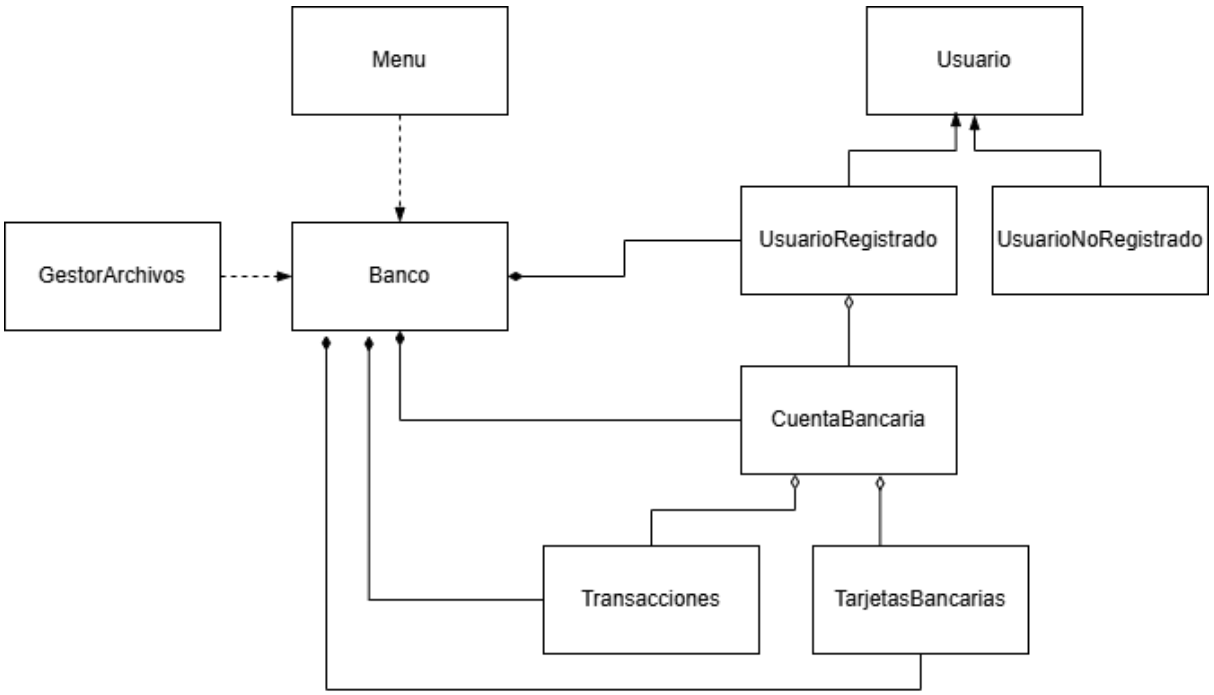
El programa debe ejecutarse en un entorno Windows debido al uso de funciones específicas como SetConsoleOutputCP(CP_UTF8) para manejar la codificación UTF-8 en la consola. Además, si quieres compilar el programa necesitarás la versión de C++ 17 o superior. Si no quieres compilarlo y solo ejecutarlo no te hará falta C++.

Estructura del repositorio:

Proyecto Sist Bancario/

```
│
├── include/        → Archivos .hpp (declaraciones)
├── src/            → Archivos .cpp (implementaciones)
├── build/          → Ficheros compilados (.exe)
├── Documentación/  → Informe, requisitos,  etc.
├── .vscode/        → Configuración del entorno
        └── *.json          → Configuración del proyecto
└── *                       → Resto de archivos (.txt, .obj y .pdb)
```

Diagrama de clases (simplificado):



Cómo la metodología agile ha sido usada:

El desarrollo del proyecto se ha llevado a cabo siguiendo un enfoque ágil, basado en la colaboración, la planificación por fases cortas y la mejora continua. Desde el inicio, se aplicaron los principios del desarrollo incremental, permitiendo construir el sistema por partes funcionales, testearlo progresivamente y adaptarlo conforme surgían nuevas necesidades.

Fase inicial – Planificación y diseño en equipo

Durante las primeras sesiones de trabajo, nos reunimos como equipo para definir la idea general del proyecto: un sistema bancario modular que permita gestionar usuarios, cuentas, movimientos y tarjetas. A partir de ahí, abordamos las siguientes tareas iniciales:

- Definición de la idea central del proyecto.
- Diseño del diagrama de clases, identificando entidades clave y sus relaciones.

- Planificación de la estructura de almacenamiento, optando por el uso de archivos .txt simulando una base de datos.
- Creación de la estructura del proyecto en carpetas, separando .hpp y .cpp, y configurando los archivos .json necesarios para la compilación.

Desarrollo iterativo e incremental
Después, iniciamos el desarrollo del código de manera incremental. Comenzamos por construir un menú base que nos permitió visualizar las funcionalidades que el sistema debía tener, actuando como punto de partida para implementar el resto de clases. Aunque la clase principal era Banco, trabajar desde el menú nos ayudó a definir la interacción entre módulos.

En esta fase, entregamos una primera versión funcional que incluía las operaciones básicas: gestión de usuarios, creación de cuentas y operaciones simples. Esta entrega sirvió como hito para evaluar el progreso y planificar la siguiente iteración.

Segunda iteración – Modularización y persistencia
Después de la entrega inicial, se identificó la necesidad de centralizar la lógica de entrada/salida. Por ello, se desarrolló la clase GestorDeArchivos, encargada de la lectura y escritura de datos en los archivos .txt. Esta clase se fue ampliando de forma progresiva, añadiendo métodos específicos conforme cada funcionalidad lo requería.

El programa se diseñó para operar en modo local: los datos se cargan al inicio desde los archivos, se modifican en memoria (RAM) y, tras completar las operaciones, se vuelven a guardar, asegurando una gestión eficiente y coherente de los datos.

Roadmap (Aproximado)
Aunque no lo hayamos hecho con esta estructura de semanas de manera exacta es una buena aproximación a la realidad.

Semana 1: Idea del proyecto, definición de clases, planificación.
Semana 2: Estructura del proyecto, organización de carpetas y archivos.
Semana 3: Desarrollo del menú y primeras funcionalidades.
Semana 4: Entrega de la primera versión funcional.
Semana 5: Implementación de GestionDeArchivos, refactorización y mejoras finales.

Comentarios de negocio:

Este sistema bancario puede representar la base funcional de una herramienta útil para pequeñas entidades financieras, cooperativas o startups fintech que necesiten gestionar cuentas, usuarios y transacciones sin recurrir a costosas soluciones comerciales.

Desde el punto de vista de negocio, el sistema podría ofrecerse como servicio SaaS (Software as a Service), donde las entidades usuarias pagarían una comisión por operación realizada (por ejemplo, transferencias, ingresos, retirada de efectivo, etc.). Este modelo permite monetizar el uso real del software, haciendo que sea accesible para organizaciones pequeñas y escalable en ingresos conforme aumente su volumen de actividad.

Además, el sistema es modular y fácilmente ampliable, lo que permite personalizarlo según las necesidades del cliente (por ejemplo, añadir soporte para tarjetas virtuales, criptomonedas o sistemas de verificación más avanzados), lo cual abre la puerta a nuevos modelos de negocio como licencias por módulos, tarifas premium o soporte técnico personalizado.

En resumen, este proyecto no solo cumple una función técnica y educativa, sino que también puede tener un enfoque comercial viable basado en uso por operación, con potencial para integrarse en servicios reales.