Deber 12
Nombres: Alex Travez, Mateo Oviedo
Fecha de realización: 04 /01/2025
Fecha de entrega: 07/01/2025

**RESULTADOS:**

***Genere el diagrama de casos de uso:***

**Caso de Uso: Registrar Información en Banco**

**i. Nombre único:** Registrar Información en Banco

**ii. Actores:** Usuario

**iii. Condiciones de entrada:**

- El programa está en ejecución.
- El usuario selecciona la opción "Registrar información en banco" en el menú principal.

**iv. Flujo de eventos:**

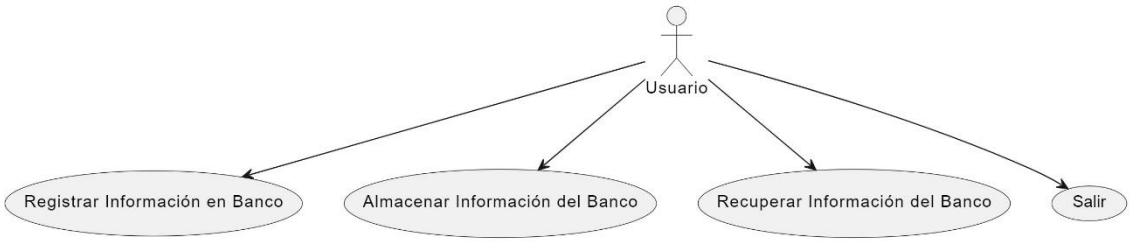
1. El sistema solicita al usuario que ingrese el nombre del propietario.
2. El usuario ingresa el nombre del propietario.
3. El sistema solicita al usuario que ingrese el balance inicial.
4. El usuario ingresa el balance inicial.
5. El sistema solicita al usuario que seleccione el banco (1 para Banco del Pichincha, 2 para Banco del Austro).
6. El usuario selecciona el banco.
7. El sistema crea una nueva cuenta bancaria con los datos proporcionados y la agrega a la lista de cuentas.
8. El sistema muestra un mensaje confirmando la creación de la cuenta.
9. El sistema repite los pasos 1-8 hasta que se hayan registrado 10 cuentas bancarias.

**v. Condiciones de salida:**

- La información de las cuentas bancarias se ha registrado correctamente en la lista de cuentas.
- El sistema vuelve al menú principal.

**vi. Requerimientos especiales:**

- La información debe ser validada para asegurar que los datos ingresados sean correctos (e.g., balance positivo, selección de banco válida).




---

**Caso de Uso: Almacenar Información del Banco**

**i. Nombre único:** Almacenar Información del Banco

**ii. Actores:** Usuario

**iii. Condiciones de entrada:**

- El programa está en ejecución.
- El usuario selecciona la opción "Almacenar información del banco" en el menú principal.
- Existen cuentas bancarias registradas en la lista de cuentas.

**iv. Flujo de eventos:**

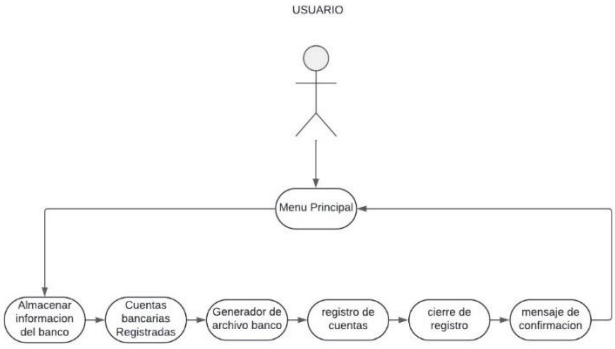
1. El sistema abre un flujo de salida de objetos hacia el archivo banco.asj.
2. El sistema escribe la lista de cuentas bancarias en el archivo.
3. El sistema cierra el flujo de salida.
4. El sistema muestra un mensaje confirmando que la información ha sido almacenada exitosamente.

**v. Condiciones de salida:**

- La información de las cuentas bancarias se ha guardado correctamente en el archivo banco.asj.
- El sistema vuelve al menú principal.

**vi. Requerimientos especiales:**

- El archivo banco.asj debe estar accesible para escritura.



**Caso de Uso: Recuperar Información del Banco**

**i. Nombre único:** Recuperar Información del Banco

**ii. Actores:** Usuario

**iii. Condiciones de entrada:**

- El programa está en ejecución.
- El usuario selecciona la opción "Recuperar información del banco" en el menú principal.
- El archivo banco.asj existe y contiene datos válidos.

**iv. Flujo de eventos:**

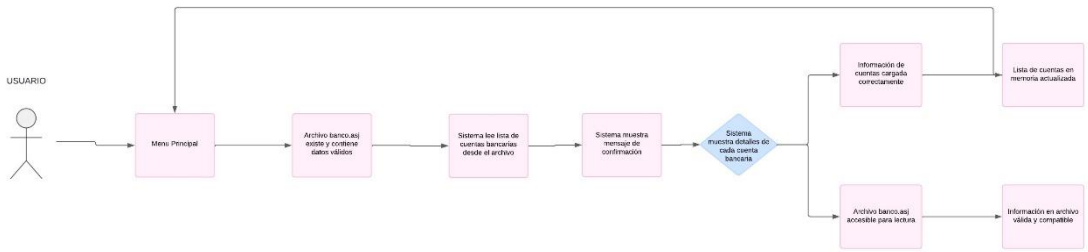
1. El sistema abre un flujo de entrada de objetos desde el archivo banco.asj.
2. El sistema lee la lista de cuentas bancarias desde el archivo.
3. El sistema cierra el flujo de entrada.
4. El sistema muestra un mensaje confirmando que la información ha sido recuperada exitosamente.
5. El sistema muestra los detalles de cada cuenta bancaria recuperada.

**v. Condiciones de salida:**

- La información de las cuentas bancarias se ha cargado correctamente desde el archivo banco.asj.
- La lista de cuentas en memoria se ha actualizado con la información recuperada.
- El sistema vuelve al menú principal.

**vi. Requerimientos especiales:**

- El archivo banco.asj debe estar accesible para lectura.
- La información en el archivo debe ser válida y compatible con la estructura esperada.



**Caso de Uso: Salir**

**i. Nombre único:** Salir

**ii. Actores:** Usuario

**iii. Condiciones de entrada:**

- El programa está en ejecución.
- El usuario selecciona la opción "Salir" en el menú principal.

**iv. Flujo de eventos:**

1. El sistema confirma la selección del usuario.
2. El sistema realiza cualquier operación de cierre necesaria (e.g., guardar información no almacenada, cerrar flujos abiertos).
3. El sistema termina la ejecución del programa.

**v. Condiciones de salida:**

- El programa se ha cerrado correctamente.

**vi. Requerimientos especiales:**

- N/A

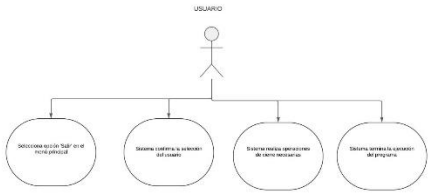
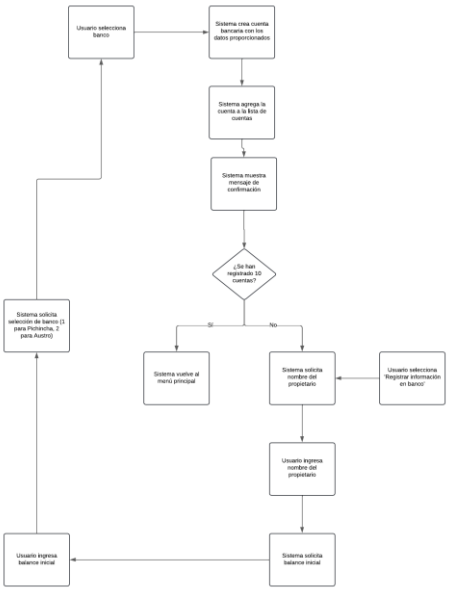
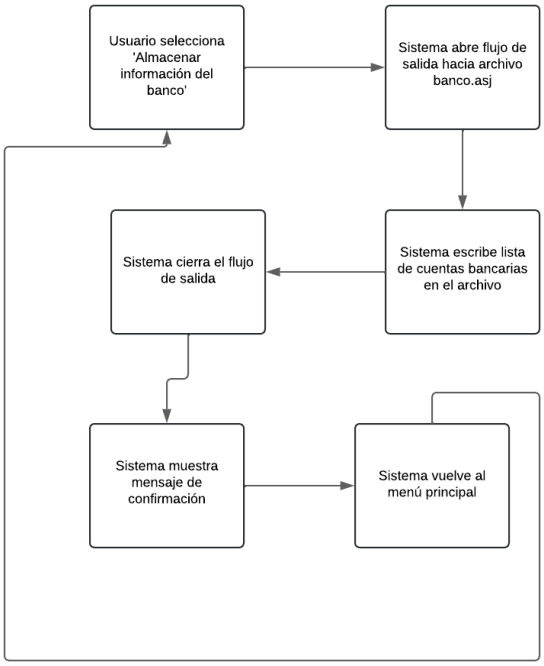



***Diagrama de actividad para cada caso de uso***

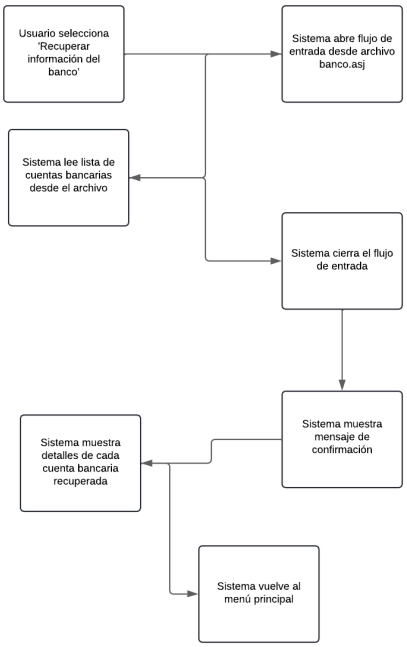
**Proporcionar Datos**



## Almacenar información del banco:




## Recuperación del banco:



**Opción Salir:**

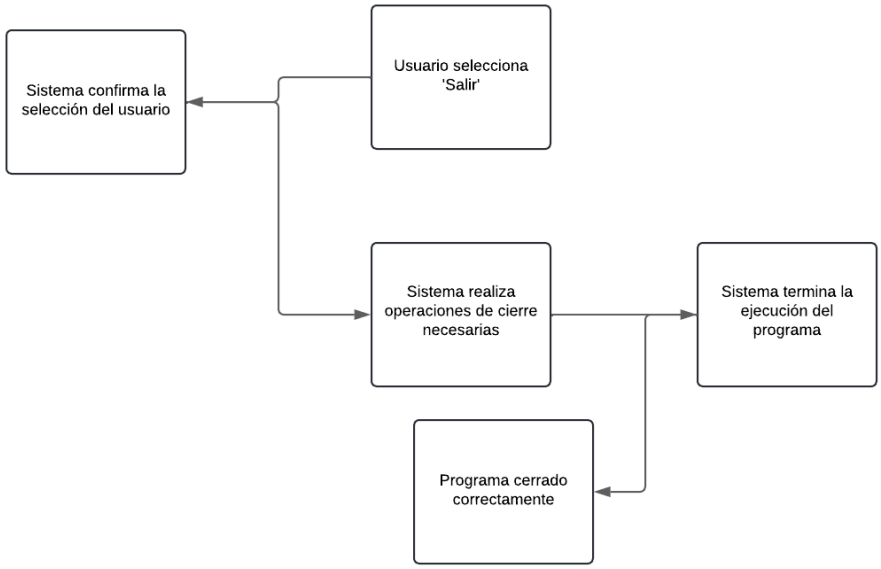
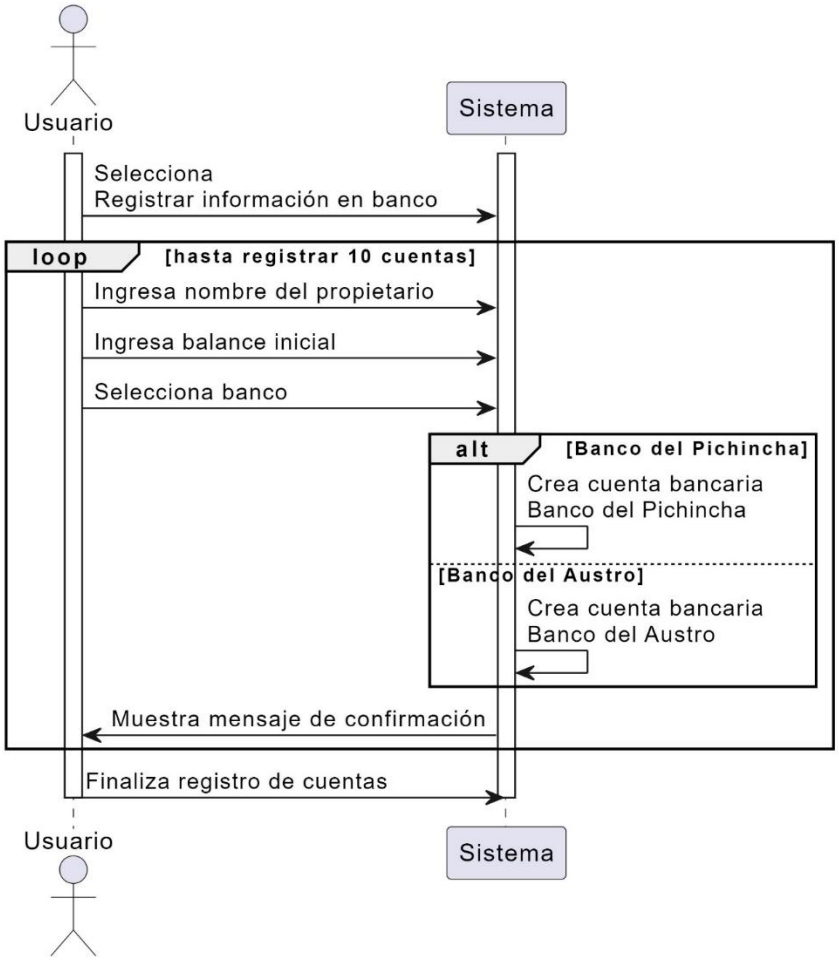
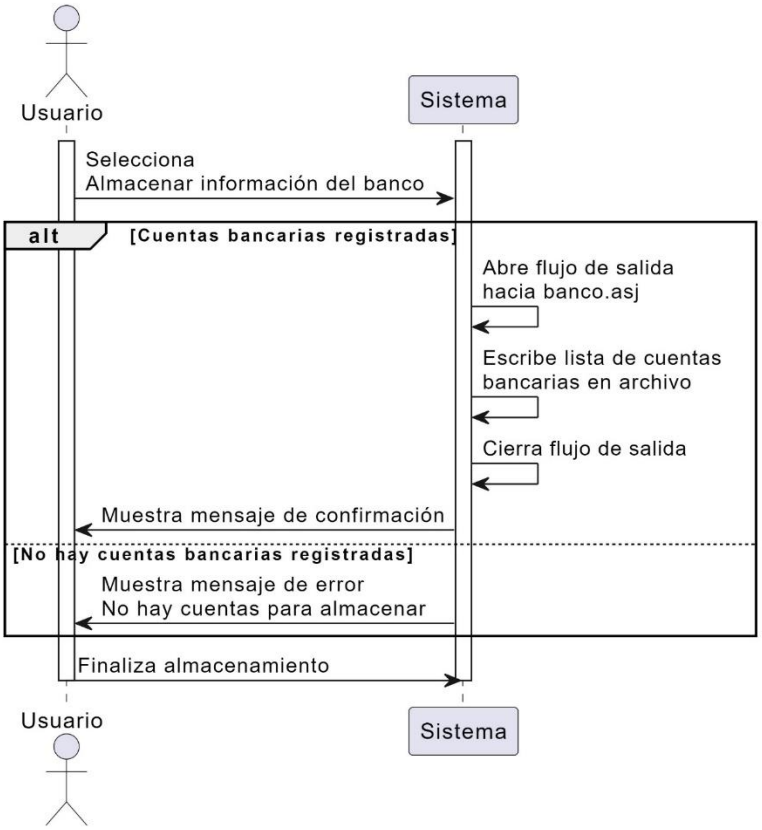
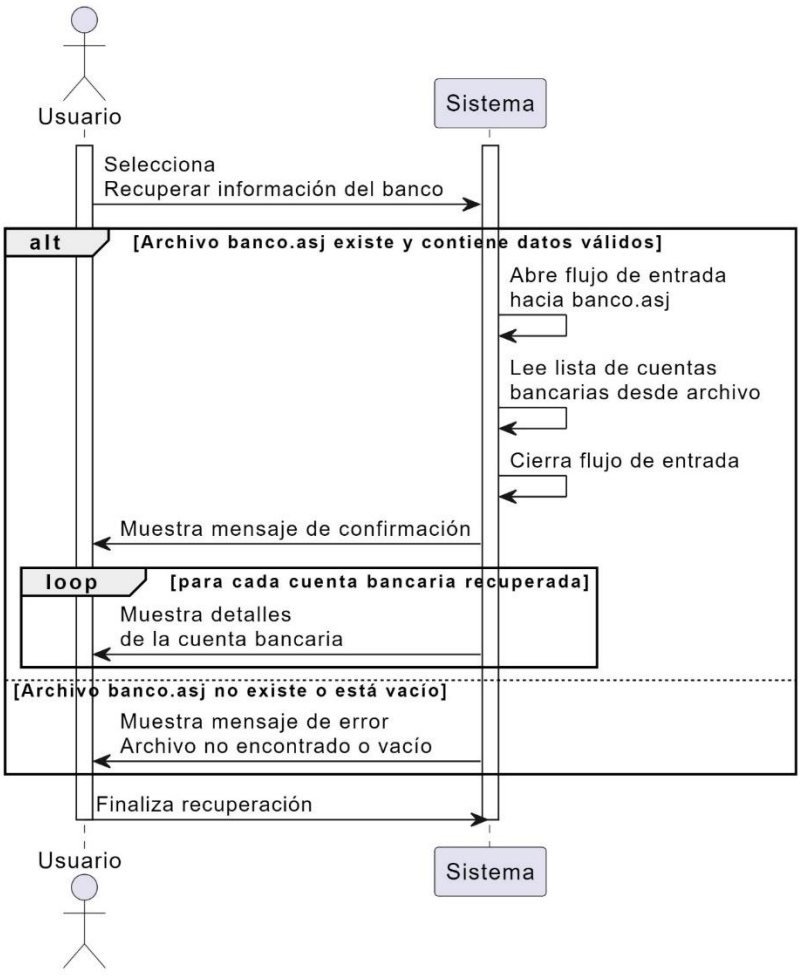



***diagrama de secuencia***




Usuario
Sistema
Selecciona
Almacenar información del banco
alt
[Cuentas bancarias registradas]
Abre flujo de salida
hacia banco.asj
Escribe lista de cuentas
bancarias en archivo
Cierra flujo de salida
Muestra mensaje de confirmación
[No hay cuentas bancarias registradas]
Muestra mensaje de error
No hay cuentas para almacenar
Finaliza almacenamiento
Usuario
Sistema


Usuario
Sistema
Selecciona
Recuperar información del banco
alt
[Archivo banco.asj existe y contiene datos válidos]
Abre flujo de entrada
hacia banco.asj
Lee lista de cuentas
bancarias desde archivo
Cierra flujo de entrada
Muestra mensaje de confirmación
loop
[para cada cuenta bancaria recuperada]
Muestra detalles
de la cuenta bancaria
[Archivo banco.asj no existe o está vacío]
Muestra mensaje de error
Archivo no encontrado o vacío
Finaliza recuperación
Usuario
Sistema

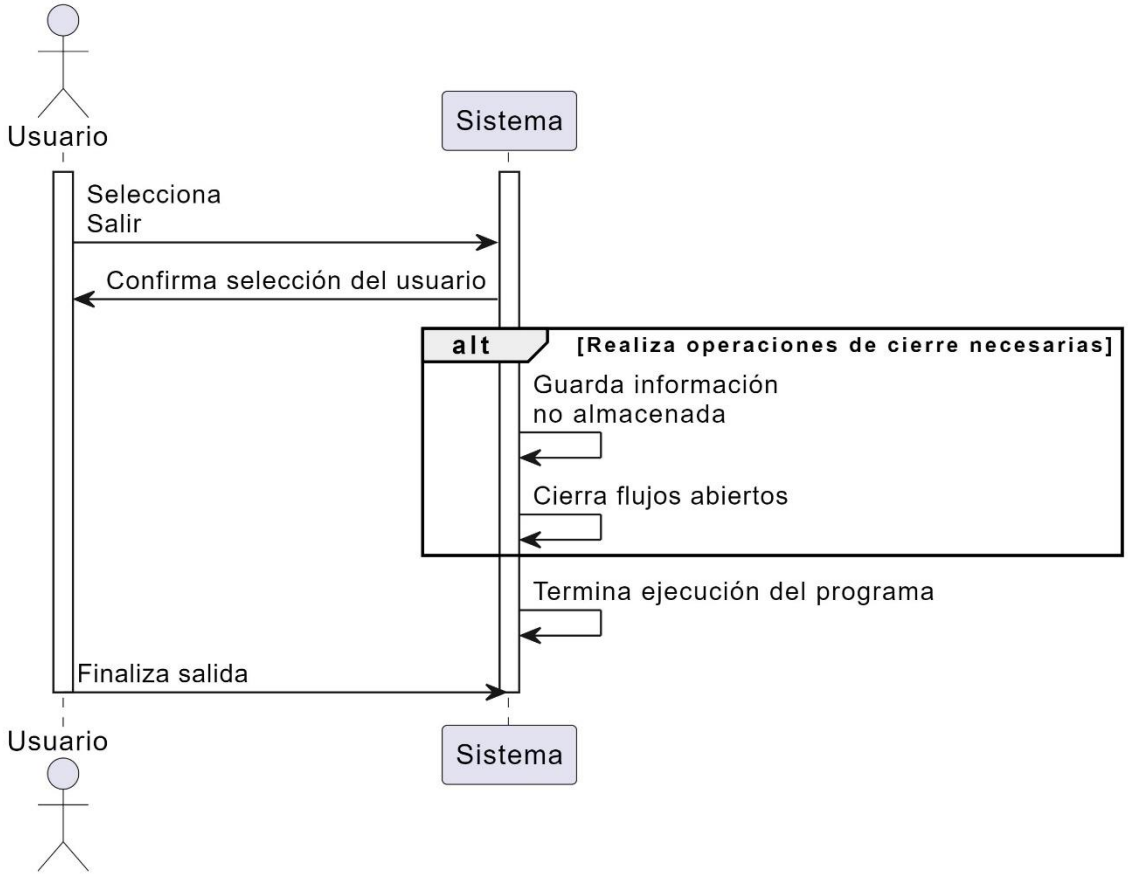
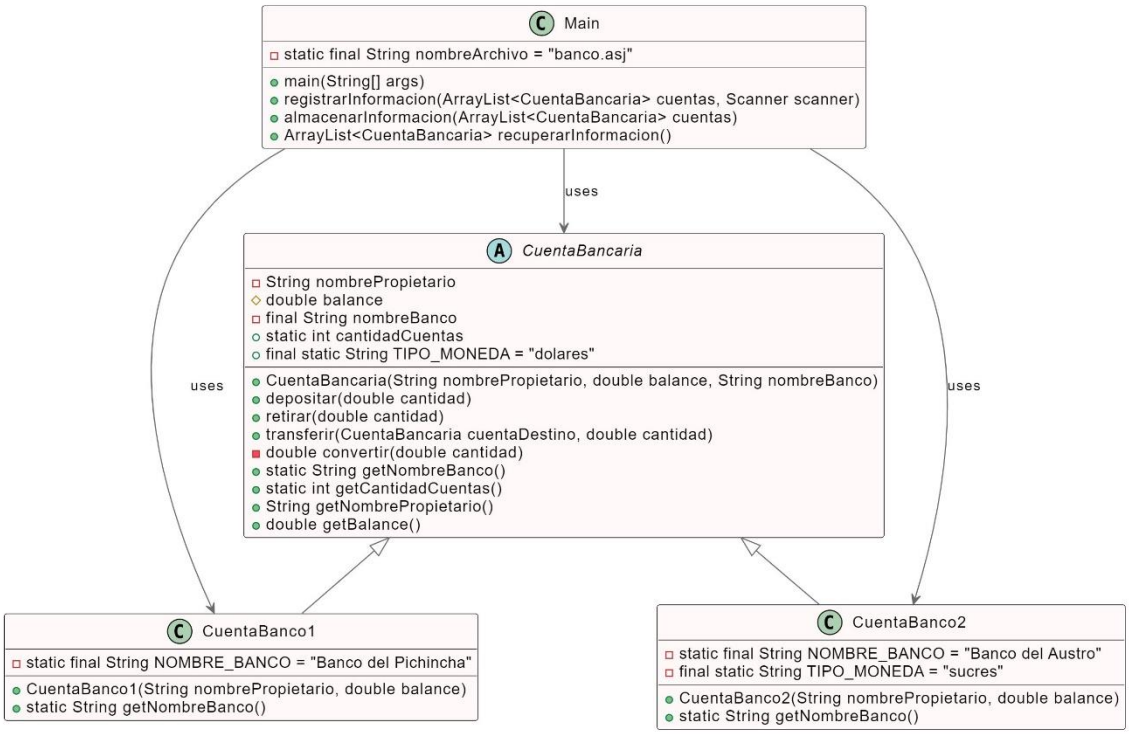



***Diagrama de clases:***



**Conclusiones:**

Alex Travez

- El uso de clases y métodos bien definidos permite una clara comprensión de las responsabilidades de cada componente del sistema. La separación de preocupaciones entre la lógica del programa, la gestión de datos y la presentación es evidente y bien manejada.
- Las clases CuentaBancaria, CuentaBanco1 y CuentaBanco2 encapsulan de manera adecuada los atributos y métodos relacionados con las cuentas bancarias. Esto permite una gestión más segura y controlada de los datos de cada cuenta.

Mateo Oviedo

- Los métodos almacenarInformacion y recuperarInformacion demuestran una correcta implementación de la serialización y deserialización de objetos, asegurando que la información de las cuentas bancarias pueda ser guardada y recuperada de manera persistente.
- El menú principal y las interacciones con el usuario están claramente delineadas, facilitando la navegación y la realización de operaciones básicas. La estructura del menú y la implementación del bucle principal permiten un flujo de trabajo intuitivo para el usuario.

**Recomendaciones:**

Alex Travez

- Implementar una validación más robusta para los datos ingresados por el usuario. Esto incluye verificar que el nombre del propietario no esté vacío, que el balance inicial sea un número positivo, y que la selección del banco sea válida. Esto evitará errores y mejorará la robustez del sistema.
- Utilizar diagramas para planificar y diseñar modificaciones o extensiones del sistema. Los diagramas de clases y actividades permiten visualizar cómo nuevas funcionalidades o cambios impactarán la arquitectura existente, ayudando a planificar mejor las implementaciones y reducir el riesgo de introducir errores.

Mateo Oviedo

- Incorporar diagramas en la documentación del proyecto para proporcionar una vista estructurada del sistema. Esto ayuda a los futuros desarrolladores y mantenedores a entender rápidamente cómo está organizado el código, las dependencias entre clases, y los flujos de trabajo principales.
- Emplear los diagramas para revisar y validar el diseño antes de la implementación. Los diagramas permiten detectar errores o omisiones en la lógica del negocio, relaciones entre clases, y flujos de actividades, evitando problemas durante el desarrollo y mantenimiento del software.

**Bibliografia:** Analisis y clases de diseño, Lenguaje UML (Material de clase, presentado por David Mejia)[Consultado: 4-JANUARY-2025].