



Carátula para entrega de prácticas

Facultad de Ingeniería

Laboratorio de docencia

Ingeniería en computación

Laboratorios de computación salas A y B

Profesor: _____ Ing. Rene Adrian Davila Perez

Asignatura: _____ Programación Orientada a Objetos

Grupo: _____ 1

No. de práctica: _____ Proyecto 2

Integrante: _____ 424121462

Semestre: _____ 2025-1

Fecha de entrega: _____ 18 de octubre de 2024

Observaciones: _____

CALIFICACIÓN: _____

Índice

| | |
|------------------|-------------|
| 1. Introducción | (2) |
| <hr/> | |
| 2. Marco Teórico | (2-3) |
| <hr/> | |
| 3. Desarrollo | (3-4) |
| <hr/> | |
| 4. Resultados | (4) |
| <hr/> | |
| 5. Conclusiones | (4-5) |
| <hr/> | |
| 6. Referencias | (5) |
| <hr/> | |

Introducción:

El proyecto se centrará en la creación de un sistema de cuentas bancarias utilizando los principios de programación orientada a objetos en Java. Se plantea la siguiente estructura:

1. Crear una clase base llamada CuentaBanco que incluya un atributo privado `saldo`, asegurando el uso de encapsulamiento, y que implemente los métodos necesarios como depositar y retirar.
2. Desarrollar dos subclases: CuentaAhorro y RevisionCuenta.
 - La clase CuentaAhorro tendrá un atributo privado limiteRetiro, y sobrescribirá el método `retirar`, aplicando restricciones de retiro basadas en el límite definido.
 - La clase RevisionCuenta incluirá un atributo privado cuotaConsulta, y también sobrescribirá el método retirar, añadiendo una cuota por cada operación de consulta o retiro.
3. Implementar una clase Principal con un método estático que permita realizar retiros según el tipo de cuenta, demostrando polimorfismo al manejar objetos de diferentes subclases.

Este proyecto tiene como objetivo aplicar los conceptos de herencia, encapsulamiento y polimorfismo aprendidos hasta ahora. La herencia permitirá reutilizar el código base de CuentaBanco en las subclases, mientras que el polimorfismo facilitará la gestión de distintos tipos de cuentas de manera flexible y eficiente.

Se espera que el programa nos permita un diseño claro y robusto, asegurando que las operaciones bancarias se ejecuten correctamente bajo diferentes reglas de negocio según el tipo de cuenta. Además, el uso de encapsulamiento garantizará la protección adecuada de los datos, promoviendo buenas prácticas de programación orientada a objetos.

Marco Teórico:

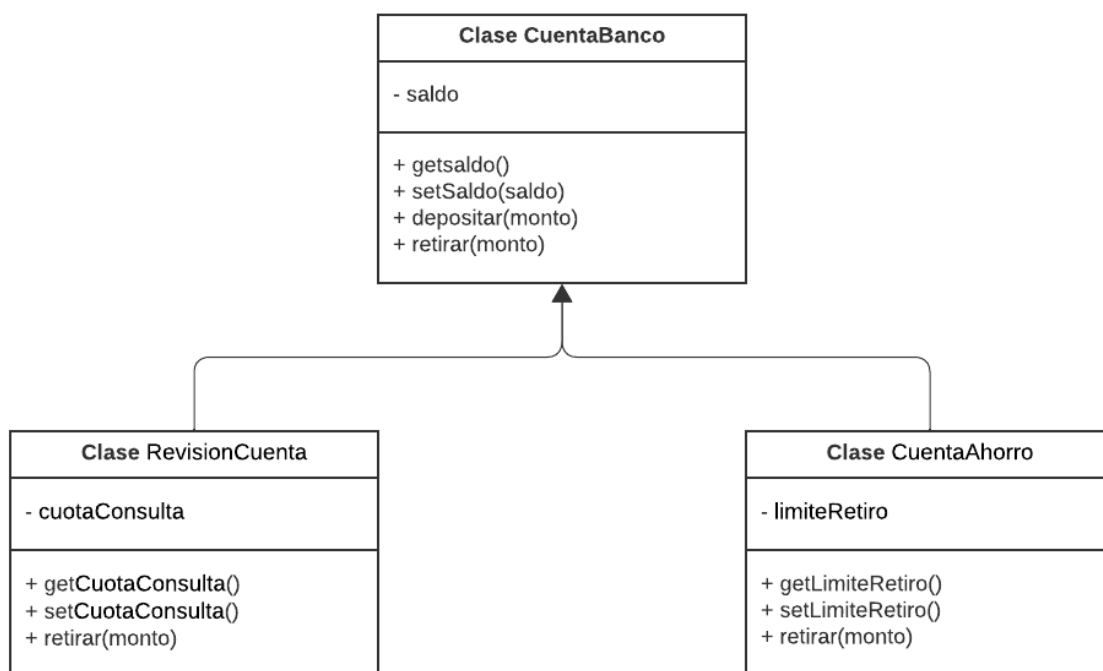
El polimorfismo permite que un mismo objeto adopte diferentes comportamientos según el contexto, lo que resulta esencial para que el código se adapte a las necesidades del programador. Facilita el uso intercambiable de clases, un esencial para crear software que pueda evolucionar.

Al emplear interfaces, se facilita la incorporación de nuevas implementaciones sin necesidad de alterar el código existente. Esto respeta el principio de abierto para extensión y cerrado para modificación, un pilar fundamental del diseño orientado a objetos. Las interfaces y el polimorfismo posibilitan trabajar con referencias de clases más generales, promoviendo la reutilización del código y evitando la duplicación de lógica. [1]

Las interfaces actúan como una capa de abstracción al establecer qué acciones debe realizar una clase, sin especificar cómo llevarlas a cabo. Esto simplifica la comprensión del sistema, permitiendo a los desarrolladores concentrarse en los métodos que debe cumplir una clase, sin complicar los detalles de su implementación.

Integrar polimorfismo e interfaces contribuye a un diseño más limpio y organizado. Las responsabilidades y funciones de las clases se vuelven claras, lo que mejora su lectura, beneficiando a los desarrolladores. [2]

Desarrollo:

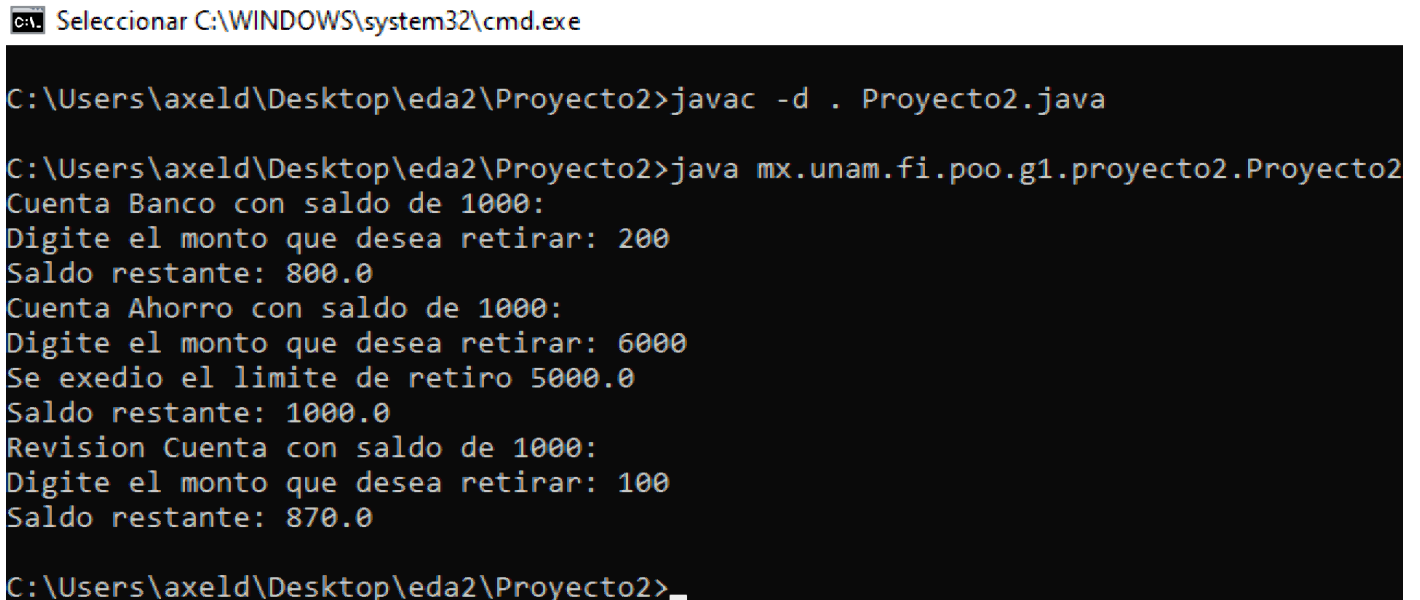


Se realizaron modificaciones a los códigos entregados en la sesión práctica debido a la falta de métodos y encapsulamientos, además de agregar de mejor manera las funcionalidades de los códigos, donde originalmente a la hora de ejecutarse el programa presentaba fallas por no hacer correcta referencia a estos gracias al atributo `this`, ya una vez actualizado todos los códigos, se procedió a realizar las pruebas correspondientes, donde:

- La Cuenta Banco se inicializa con un saldo de 1000 y se le restaran 200, esperando como resultado que quede un saldo final de 800.
- La Cuenta Ahorro se inicializa al igual con 1000 y se le intentará retirar 6000, esperando un error de ejecución debido a la restricción establecida en la función sobreescrita de la misma clase, donde se establece que no se puede retirar más de 5000.

- Para Revision Cuenta igual se inicializa con 1000 y se le retirara 100, esperando como resultado 870 debido a la sobrescritura de la función retirar, en la cual se establece que la cuota por movimiento es de 30, dando 130 como el monto que se retirara.

Resultados:



```
C:\Users\axeld\Desktop\eda2\Proyecto2>javac -d . Proyecto2.java
C:\Users\axeld\Desktop\eda2\Proyecto2>java mx.unam.fi.poo.g1.proyecto2.Proyecto2
Cuenta Banco con saldo de 1000:
Digite el monto que desea retirar: 200
Saldo restante: 800.0
Cuenta Ahorro con saldo de 1000:
Digite el monto que desea retirar: 6000
Se exedio el limite de retiro 5000.0
Saldo restante: 1000.0
Revision Cuenta con saldo de 1000:
Digite el monto que desea retirar: 100
Saldo restante: 870.0
C:\Users\axeld\Desktop\eda2\Proyecto2>_
```

En la imagen anterior se puede observar la ejecución del programa bajo la prueba que se realizó donde se inicializan las 3 cuentas con saldo inicial de 1000.

Conclusiones:

En conclusión, comprender la teoría es fundamental para el éxito del proyecto. La aplicación de conceptos como la herencia, el encapsulamiento y el polimorfismo no solo permite una estructura de código más eficiente y reutilizable, sino que también facilita la implementación de diversas reglas para cada tipo de cuenta. Sin un sólido entendimiento teórico, la creación de este sistema podría resultar en un diseño confuso y poco mantenible, lo que afectaría la calidad y funcionalidad del software. Por lo tanto, dominar estos principios no solo optimiza el desarrollo, sino que asegura que el sistema sea robusto, claro y capaz de adaptarse a futuras modificaciones, promoviendo buenas prácticas en la programación orientada a objetos.

Asimismo, comprender la herencia y la sobrescritura en Java es crucial para lograr una práctica eficiente. Estos principios ofrecen un enfoque para organizar el código de manera efectiva, facilitando la reutilización. Sin un entendimiento teórico adecuado, aplicar correctamente estas

herramientas en proyectos reales se vuelve desafiante, lo que puede derivar en un código desorganizado y difícil de mantener.

Referencias:

[1] "III - Herencia," UNAM. [En línea]. Disponible:

http://profesores.fi-b.unam.mx/carlos/java/java_basico3_4.html. [Accedido: Oct. 4, 2024].

[2] ChatGPT. [En línea]. Disponible: <https://chatgpt.com>. [Accedido: Sep. 29, 2024]. Pregunta:

“Platícame más acerca de la sobrescritura en Java”