INFORME PRÁCTICA 5A: REFACTORIZACIONES

1. Introducción

Este documento tiene como objetivo presentar y justificar las refactorizaciones realizadas a un sistema software, con el fin de mejorar su calidad interna. Se detallan los cambios aplicados, fundamentados en principios de diseño y buenas prácticas, para optimizar la mantenibilidad, legibilidad y estructura del código sin modificar su comportamiento funcional.

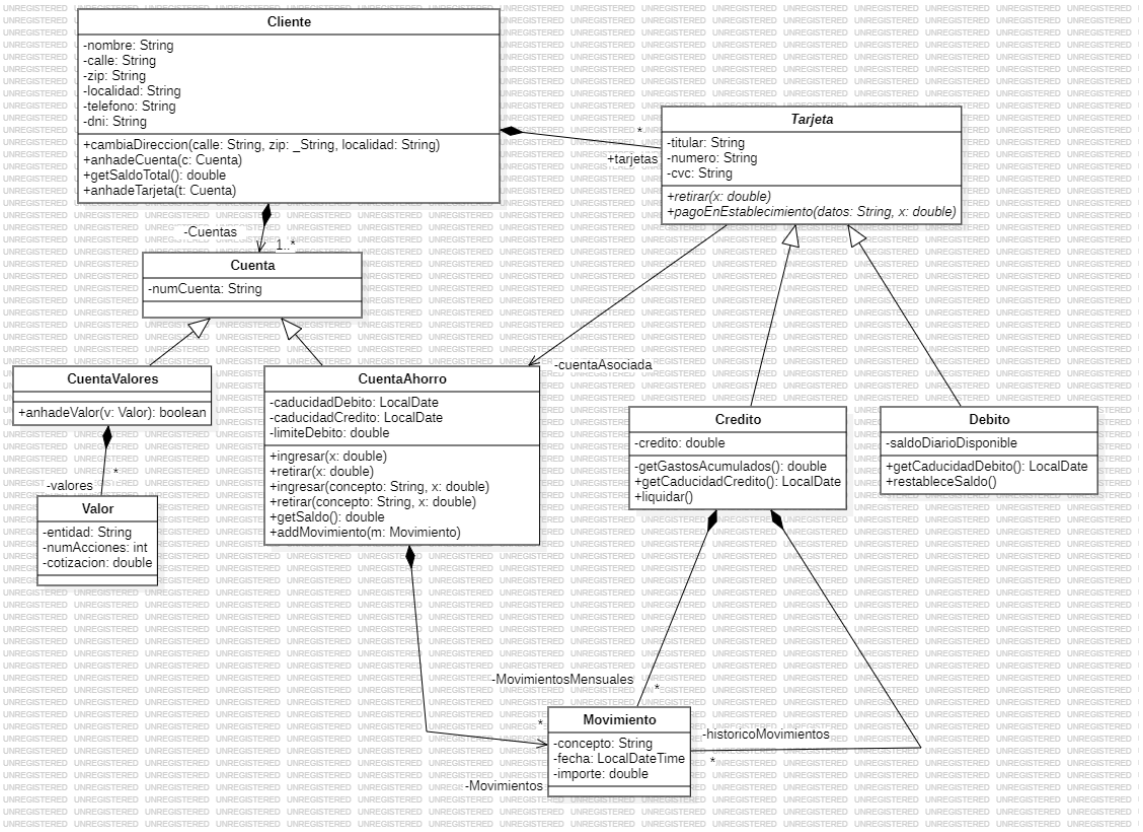
El proceso descrito en este informe consiste en analizar el código fuente para identificar malas prácticas, duplicación, o violaciones de principios de diseño, y aplicar refactorizaciones puntuales que mejoren la estructura y legibilidad sin alterar la funcionalidad del sistema.

2. Refactorizaciones aplicadas en el código

- Se estandarizaron el nombre de algunas variables en las clases para mejorar el entendimiento general del código.
- Se cambio el nombre del método getSaldo por calculaSaldo para que suene menos genérico.
- Añadimos un constructor en Movimiento para automáticamente crear la variable con los datos indicados, y de paso quitamos todas las llamadas a funciones de Movimiento que un constructor trivializará en otras clases, además de limpiar los métodos que hacian uso de estas llamadas. Hicimos esto para reducir la sobre complicación del código en varias clases, que llamaban a distintos métodos de Movimiento para cambiar el valor de sus atributos en vez de inicializarlos con ellos.
- Se agregaron los strings con los que se inicializaría Movimiento aparte en Credito y CuentaAhorro para facilitar la comprensión del código.
- Se añadieron variables en Credito y Débito de parte de CuentaAhorro, de las cuales se eliminó, además de métodos sets y gets con el fin de balancear el WMC total entre clases, y así reducir el de CuentaAhorro bajando el número de métodos que utiliza la clase.
- Añadimos el método actualizaCaducidad en la clase Tarjeta y el método calculaSaldo de la clase Cuenta para ser usado por sus subclases, y modificamos los métodos anhadeTarjeta y getSaldoTotal de la clase Cliente haciendo uso de los dos primeros métodos y aprovechando que las dos clases de las que provienen son superclases para que utilizará menos “instance of” con el fin de reducir el CBO, el CCog y el WMC.
- Creamos una clase nueva “ValidacionCantidades” que contendría los métodos de validación comunes que lanzan throws que utilizan las clases Crédito, Débito, y CuentaAhorro con el fin de reducir el CCog y WMC.

3. Comparación entre situación inicial y final del sistema

Inicial:



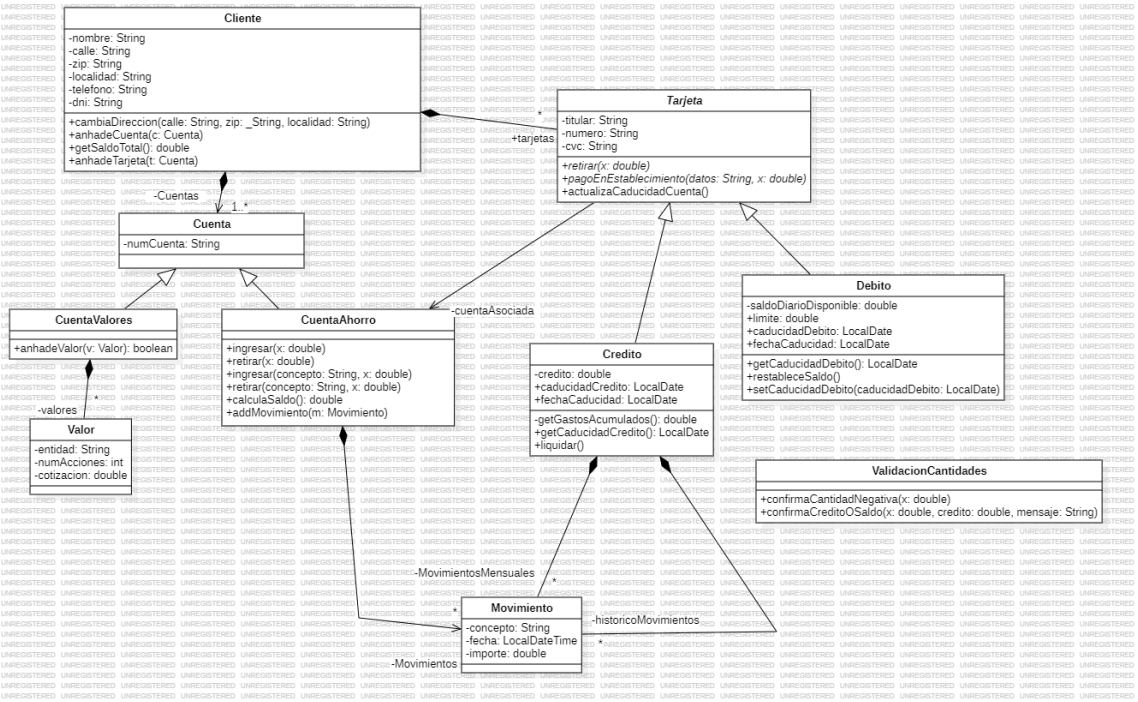
Clases	WMC	WMCn	CCog	CCogn	DIT	NOC	CBO
Cliente	15	1,36	9	0,81	0	0	9
Cuenta	2	1	0	0	0	2	3
CuentaValores	4	1,33	3	1	1	0	1
CuentaAhorro	19	1,46	7	0,53	1	0	10
Valor	7	1	1	0,14	0	0	1
Tarjeta	1	1	0	0	0	2	4
Credito	14	1,55	8	0,88	1	0	7
Debito	8	1,33	2	0,3	1	0	5
Movimiento	7	1	1	0,14	0	0	3
datoErroneoException	1	1	0	0	0	0	9
saldoInsuficienteException	1	1	0	0	0	0	6

Como plan de refactorización sacamos de estas métricas un número de prioridades:

- Hay que bajar el WMC (recomendado que sea menor de 10) y la CCog (cuanto más se baje, mejor) de **Cliente**.
- Hay que bajar el WMC (recomendado que sea menor de 10) y la CCog (cuanto más se baje, mejor) de **CuentaAhorro**.

- Hay que bajar el WMC (recomendado que sea menor de 10) y la CCog (cuanto más se baje, mejor) de **Credito**.
- Si hay que aumentar algunas de las otras métricas para eso, no importa.
- Si es posible, bajar el CBO de clases muy dependientes entre si (si compromete la bajada de otras métricas, no es necesario).
- No prestar atención a los Exception, no hace falta cambiar nada de esos.

Final:



Clases	WMC	WMCn	CCog	CCogn	DIT	NOC	CBO
Cliente	11	1	1	0,09	0	0	2
Cuenta	2	1	0	0	0	2	3
CuentaValores	5	1,25	4	1	1	0	1
CuentaAhorro	8	1	1	0,125	1	0	11
Valor	7	1	1	0,14	0	0	1
Tarjeta	1	1	0	0	0	2	4
Credito	12	1,09	3	0,27	1	0	10
Debito	9	1	0	0	1	0	4
Movimiento	8	1	1	0,12	0	0	3
datoErroneoException	1	1	0	0	0	0	10
saldoInsuficienteException	1	1	0	0	0	0	7
ValidacionCantidades	4	2	2	1	0	0	16

Analizando las nuevas métricas, podemos notar varios factores:

- Aunque Cliente y Crédito tengan WMCs superiores a 10, observando sus WMCns correspondientes, podemos observar que están o en su valor mínimo posible (1) o muy cerca de él. Mirando el código, efectivamente podemos ver que en Cliente no hay

ningún método con un valor mayor que 1, y en Credito, solo uno, el cual vale 2. Considerando los métodos dentro, también nos damos cuenta de que no tendría sentido delegar funcionalidades, ya que estas clases ya hacen lo que deben y poco más. Por tanto, su refactorización es correcta.

- El código en general es muy facil y limpio de entender, siendo la mayor métrica de CCog 4.
- Por otro lado, sí que ha habido un mayor aumento que decremento de dependencias en todas las clases. Estas son justificables, como especificadas en la lista de refactorizaciones, y se ha hecho lo posible por bajar el CBO siempre que se pudiera.
- Haciendo uso de las clases de prueba modificadas para adecuarse al código nuevo, y creando una adicional para la nueva clase añadida, hemos notado que se mantiene la misma cobertura que previamente, por lo que el código sigue funcionando como debería.

Teniendo en cuenta todos estos datos, podemos afirmar que la diferencia de métricas refleja correctamente las mejoras de calidad introducidas en el código.