SANDRA ALEGRÍA Y SEBASTIÁN RAMÍREZ

MÉTRICAS:

WMC: Mide la complejidad de una clase.

WMCn : WMC normalizado.

CBO: Mide el acoplamiento de una clase con el resto de clases del sistema.

DIT: Longitud desde una subclase hasta la clase raíz.

NOC : Número de subclases inmediatas de una clase dada.

	WMC		WMCn		CCog		CBO		DIT		NOC	
	INI	FIN	INI	FIN	INI	FIN	INI	FIN	INI	FIN	INI	FIN
Cliente	14	8	1,4	1,14	7	1	4	3	0	0	0	0
Credito	17	14	1,89	1,55	8	6	6	6	1	1	0	0
Cuenta	2	2	1	1	0	0	4	4	0	0	2	2
CuentaAhorro	18	16	1,64	1,23	7	3	8	7	1	1	0	0
CuentaValores	5	7	1,66	1,75	3	4	3	2	1	1	0	0
Debito	8	8	1,33	1,14	2	1	4	4	1	1	0	0
Movimiento	6	8	1	1	0	0	2	2	0	0	0	0
Tarjeta	1	1	1	1	0	0	5	5	0	0	2	2
Valor	7	7	1	1	0	0	2	2	0	0	0	0
Dirección	-	5	-	1	-	0	-	1	-	0	-	0

CONLUSIÓN: Aunque las métricas no se han balanceado del todo, hemos conseguido alivianar las clases más complejas y hemos conseguido simplificar el código en gran medida en cuanto a facilidad de lectura y comprensión aunque no se refleje mucho en las métricas.

CBO: (Antes de la refactorización)

NOTA: Interno (dentro del código de la clase, la clase depende de ellos).

Externo (fuera del código de la clase en cuestión, otras clases depende de ello).

CLASE	ENUMERACIÓN
Cliente	Interno: Cuenta, Valor, CuentaAhorro, CuentaValores Externo: -
Credito	Interno: Movimiento, saldoInsuficienteException, datoErroneoException, Cuenta, CuentaAhorro, Tarjeta Externo: -
Cuenta	Interno: - Externo: Cliente, Credito, CuentaAhorro, CuentaValores
CuentaAhorro	Interno: Cuenta, Movimiento, datoErroneoException, saldoInsuficienteException Externo: Cliente, Credito, Debito, Tarjeta
CuentaValores	Interno: Valor, Cuenta

	Externo: Cliente
Debito	Interno: Tarjeta, CuentaAhorro, saldoInsuficienteException, datoErroneoException Externo: -
Movimiento	Interno: - Externo: Credito, CuentaAhorro
Tarjeta	Interno: CuentaAhorro, datoErroneoException, saldoInsuficienteException Externo: Credito, Debito
Valor	Interno: - Externo: Cliente, CuentaValores

CBO: (Despúes de la refactorización):

CLASE	ENUMERACIÓN
Cliente	Interno: Cuenta, Valor, Dirección. Externo: -
Credito	Interno: Movimiento, saldoInsuficienteException, datoErroneoException, Cuenta, CuentaAhorro, Tarjeta Externo: -
Cuenta	Interno: - Externo: Cliente, Credito, CuentaAhorro, CuentaValores
CuentaAhorro	Interno: Cuenta, Movimiento, datoErroneoException, saldoInsuficienteException Externo: Credito, Debito, Tarjeta
CuentaValores	Interno: Valor, Cuenta Externo: -
Debito	Interno: Tarjeta, CuentaAhorro, saldoInsuficienteException, datoErroneoException Externo: -
Movimiento	Interno: - Externo: Credito, CuentaAhorro
Tarjeta	Interno: CuentaAhorro, datoErroneoException, saldoInsuficienteException Externo: Credito, Debito
Valor	Interno: - Externo: Cliente, CuentaValores
Dirección	Interno: - Externo: Cliente

REFACTORIZACIONES:

CuentaAhorro:

En la clase CuentaAhorro hemos extraído una sección del código, de los métodos “retirar” a un nuevo método gestionExcepcionesRetirar, consiguiendo así que el CCog disminuta y además el WMC también. También hemos extraído una sección del código, de los métodos “ingresar” a un nuevo método gestionExcepcionesIngresar, de esta manera conseguimos que el CCog disminuya.

En la clase Movimiento hemos creado un constructor para que en la clase CuentaAhorro los métodos sean más legibles y breves, aunque de esta manera aumentemos la complejidad de la clase Movimiento por agregarle un método más.

En la clase CuentaAhorro hemos añadido un nuevo atributo “saldo” de tipo double, además hemos modificado los dos métodos “ingresar” y los dos “retirar” de manera que se actualice este saldo por cada movimiento. Además en el método “addMoviemiento” hemos hecho que se actualice el saldo también para mantener la funcionalidad del programa original. Con esto conseguimos que en el método “getSaldo” no sea necesario recorrer cada vez los movimientos y simplificamos (es un simple return) el método tanto por parte del CCog como en el WMC.

En esta clase hemos decidido agrupar el código de los métodos “ingresar” en uno sólo y que el método ingresar que no recibe como parámetro un concepto, le pase el string correspondiente al otro método, de esta manera eliminamos duplicidades de código. Hemos empleado esta misma estrategia en los métodos “retirar”.

Credito:

En esta clase hemos usado el constructor creado en Movimiento anteriormente, para conseguir de nuevo que los métodos sean más legibles y breves.

En el método “retirar” hemos eliminado el else después de la excepción ya que si la excepción salta, el programa no va a realizar la línea de código siguiente y esto en esencia es lo mismo que tener un if-else pero con menor CCog y WMC.

En el método “liquidar” hemos omitido el for y hemos llamado al método ”getGastosAcumulados” ya que el código es idéntico, a excepción del return que en “getGastosAcumulados” retorna -r y en “liquidar” necesitamos el valor en positivo, pero hemos solventado esto añadiendo un – antes de la llamada al método. Gracias a esto hemos reducido el WMC y el CCog.

En los métodos “retirar” y “pagoEnEstablecimiento” nos planteamos la idea de agrupar el código para evitar duplicidad (al igual que en Debito) sin embargo, en este caso habría que pasar como parámetro dos Strings y dos doubles. Hemos decidido dejarlo como está porque hemos considerado que esto hubiera complicado la comprensión y lectura del código.

Cliente:

En esta clase hay sólo un método significativo en cuento a cómo afecta a las métricas. Para suavizar su efecto hemos extraído la parte del for que afecta a las cuentas de tipo CuentaValores en el que se calcula su saldo, y lo hemos llevado a la clase CuentaValores a un nuevo método llamado “getSaldo”, de esta manera disminuimos el WMC y CCog en cliente. Y aunque en CuentaValores estas métricas aumenten por este cambio, termina estando más equilibrado.

Además, nos hemos dado cuenta de que en “gestSaldoTotal” tanto CuentaAhorro y CuentaValores hay un método “getSaldo” por lo que hemos “trasladado” el método a la clase Cuenta como método abstracto. Y hemos mantenido las implementaciones en cada una de las dos cuentas. De esta manera, no es necesario comprobar mediante ifs de qué tipo de cuenta se trata, consiguiendo de esta forma que el método esté mucho más simplificado. Además las métricas se ven muy afectadas positivamente.

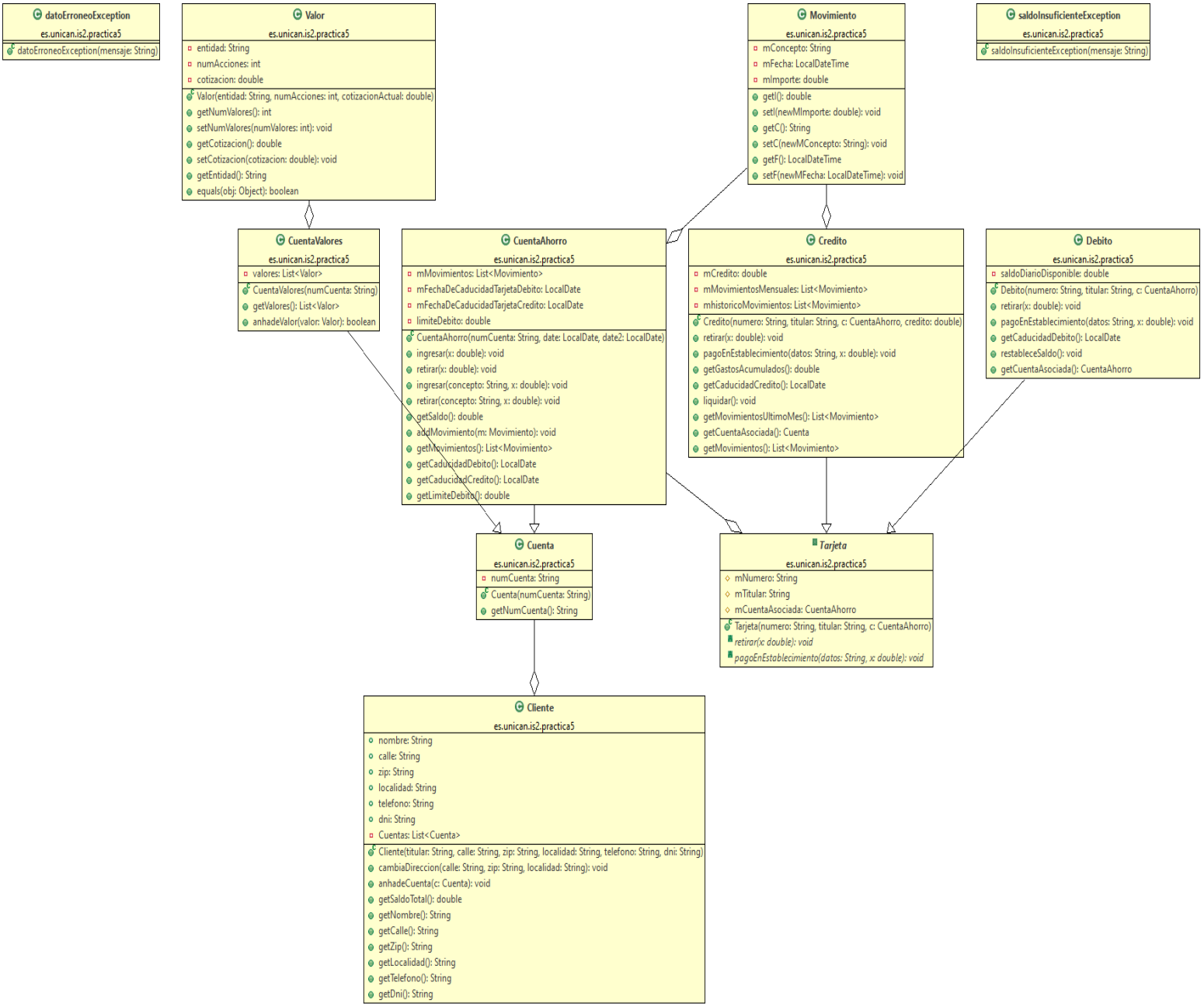
Por último, en cliente, hemos extraído la responsabilidad de la dirección del cliente a una clase externa llamada “Dirección” que se ocupa de realizar todo lo que está relacionado con la dirección del cliente.

Debito:

En este método hemos agrupado el código de los métodos “retirar” y “pagoEnEstablecimiento” en un nuevo método “retirarDinero”, manteniendo en todo momento la funcionalidad del código original. Gracias a esto eliminamos duplicidades en el código, simplificando su lectura.

DIAGRÁMAS DE CLASES: (a continuación)

ANTES DE LA REFACTORIZACIÓN:



DESPUÉS DE LA REFACTORIZACIÓN: