# Memoria Practica 5 Refactorizar:

## Métricas del proyecto inicial:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | WMC | WMCn | CBO | DIT | NOC | Ccog |
| Cliente | 8 | 2 | 4 | 0 | 0 | 10 |
| Credito | 16 | 1,77778 | 2 | 1 | 2 | 7 |
| Cuenta | 2 | 1 | 3 | 0 | 0 | 0 |
| CuentaAhorro | 18 | 1,63636 | 6 | 1 | 0 | 7 |
| CuentaValores | 3 | 1 | 4 | 1 | 0 | 0 |
| Debito | 8 | 1,33333 | 4 | 1 | 0 | 2 |
| Movimiento | 6 | 1 | 2 | 0 | 0 | 0 |
| Tarjeta | 1 | 1 | 5 | 0 | 2 | 0 |
| Valor | 6 | 1 | 2 | 0 | 0 | 0 |

## Refactorizaciones aplicadas:

### General:

Observamos que dependiendo de la clase en la que nos encontramos, se seguía un criterio u otro para nombrar las variables, en algunas se empezaban por “m” y en otras no, para mantener la consistencia en los nombres de los atributos de las clases, hemos decidido que empezaran todos por m, por ejemplo: *mNombre*, refactorizamos las variables que no lo cumplían para que todas tengan el mismo estilo.

### Cliente:

1. Extraemos los datos de la dirección y creamos una clase *Direccion* para ellos.
2. Cambiamos los atributos de public a private, esto aumentara la complejidad, pero también aumenta la privacidad de los datos y dado que estamos tratando con información bancaria nos parecía algo muy importante.

Además, añadimos los getters y setters de las variables de la clase cliente.

1. Cambiamos el nombre de la variable *Cuenta* a *cuenta*, para seguir los estándares de Java.

### CuentaAhorro:

1. El limite de debito en el constructor estaba puesto el valor como inmediato, hemos creado una constante *LIMITE\_DEBITO\_INICIAL*=1000 y asignamos a *limiteDebito* el valor de la constante en el constructor. De esta forma será mas cómodo modificarlo de ser necesario en un futuro.
2. Observamos que hay repetición de código a la hora de crear un movimiento en la cuenta en 4 de los métodos: ingresar(String,double),ingresar(double), retirar(String,double) y retirar(double)

Por lo que hemos creado un método creaMovimiento(String,double) que engloba ese código repetido.

1. Cambiamos el nombre algunas variables para hacer el código mas auto explicativo.
   1. x -> cantidad
   2. r -> importe
   3. m -> movimiento

### Movimiento:

1. Refactorizacion de los nombres de los setters y getters para que sea mas fácil su comprensión a la hora de leer el código y sobre que variables hace referencia.

### Credito:

1. Observamos repetición en los códigos getGastosAcumulados() y liquidar() por lo que hemos creado un método llamado calculaImporte() que cumple esa función.
2. Extraído a un método el código referente a crear un movimiento. El código estaba repetido en los métodos retirar(double), pagoEnEstablecimiento(String,double) y ahora hacen referencia al método creaMovimiento(String, double)

Se añade gestión de errores en este método para reducir el WMC en uno, y en n-1 siendo n el número de métodos que tuviesen que hacer el control de errores.

1. Cambiamos el nombre algunas variables para hacer el código mas auto explicativo.
   1. x -> cantidad
   2. r -> importe
   3. m -> movimiento
2. Cambiamos el nombre de la clase a TarjetaDebito para entender que extiende de Tarjeta.

### Debito:

1. Cambiamos el nombre algunas variables para hacer el código mas auto explicativo.
   1. x -> cantidad
   2. c -> cuenta
2. Cambiamos el nombre de la clase a TarjetaDebito para entender que extiende de Tarjeta.

### Dirección:

1. Aplicados los cambios generales como por ejemplo el general de los nombres de variables.

### Tarjeta:

1. Cambiamos el nombre de la variable del constructor c por cuenta para que sea mas auto explicativo.

## Métricas tras la refactorización:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | WMC | WMCn | CBO | DIT | NOC | Ccog |
| Cliente | 15 | 1,36364 | 5 | 0 | 0 | 10 |
| Credito | 15 | 1,36364 | 2 | 1 | 2 | 5 |
| Cuenta | 2 | 1 | 3 | 0 | 0 | 0 |
| CuentaAhorro | 19 | 1,58333 | 6 | 1 | 0 | 7 |
| CuentaValores | 3 | 1 | 3 | 1 | 0 | 0 |
| Debito | 8 | 1,33333 | 3 | 1 | 0 | 2 |
| Movimiento | 6 | 1 | 2 | 0 | 0 | 0 |
| Tarjeta | 1 | 1 | 5 | 0 | 2 | 0 |
| Valor | 6 | 1 | 2 | 0 | 0 | 0 |
| Direccion | 7 | 1 | 1 | 0 | 0 | 0 |

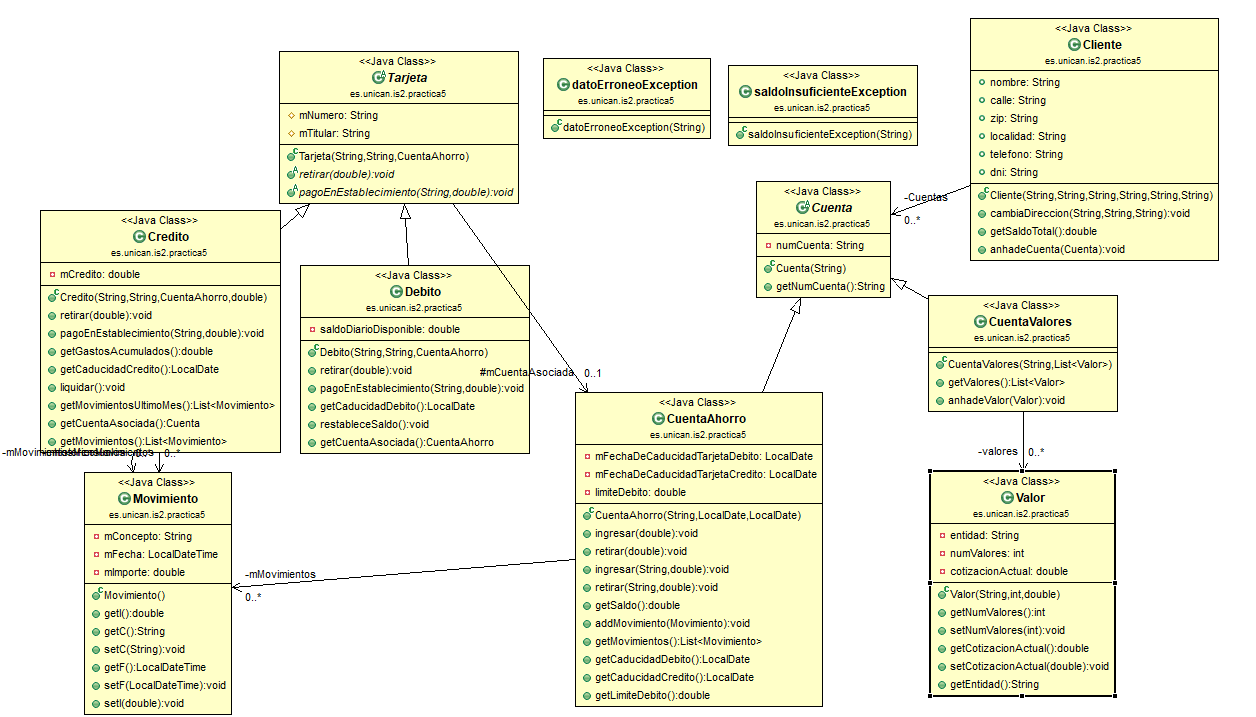
## Cálculo de las métricas:

El DIT se ha calculado teniendo en cuenta el número de clases que hay desde una clase hasta la raíz en una jerarquía de clases.

El NOC es el número total de clases que heredan de una clase dada.

El CBO es el numero de clases distintas que hay en AFF+EFF como conjuntos, es decir se quitan las repetidas.

## Diagrama de clases inicial:



## Diagrama de clases final:

