

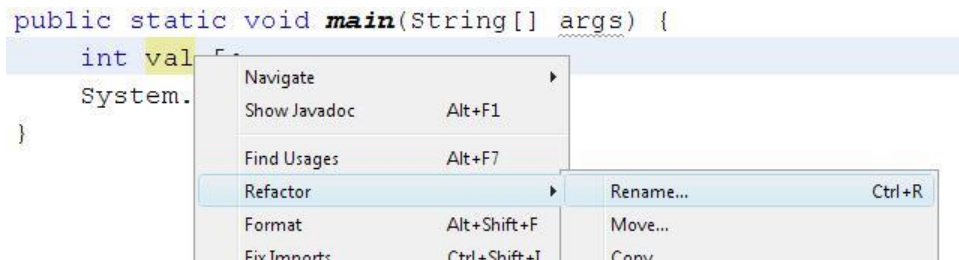
# UT5 – OPTIMIZACIÓN Y DOCUMENTACIÓN

## PARTE 2: REFACTORIZAR EN NETBEANS

# Refactoring en NetBeans

2

- Los IDEs ofrecen multitud de herramientas para hacer refactoring. Veamos algunos ejemplos con Netbeans.
- **Rename:** Te permite renombrar una variable, método o clase de forma que se renombra automáticamente en todo el proyecto.



- **Introduce Method:** Una parte de código la convertimos en un método.
  - ▣ Selecciona el código.
  - ▣ Haz clic con el botón derecho, despliega Refactor, Introduce, Method. En el menú debes poner nombre al método y el acceso.
  - ▣ Nota: observa que puedes introducir otras sentencias.

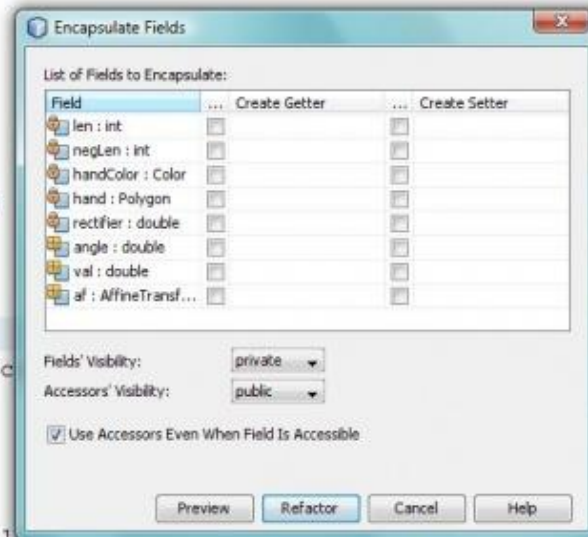
# Refactoring en NetBeans

3

- **Encapsulate Fields:** Sirve para crear métodos get o set de atributos. Además, opcionalmente puede actualizar el código con llamadas a estos métodos.
- ▣ Haz clic con el botón derecho en cualquier parte del código y escoge “Encapsulate Fields”.
- ▣ En el menú que aparece puedes escoger el campo que quieres encapsular, la accesibilidad y si quieres crear un método get o uno set.

```
class Hand{  
    private int len;  
    private int negLen;  
    private Color handColor;  
    private Polygon hand;  
    private double rectifier;  
    double angle;  
    double val;  
    AffineTransform af;  
}
```

```
Hand(double value, Color col, int rec  
    len = 100;  
    val = value;  
    negLen = 0;  
    handColor = col;  
    rectifier = Math.PI*rec/100;  
    initialize();
```



# Refactoring en NetBeans

4

- **Move Class:** Mueve una clase a otro paquete o clase. Actualiza todas las referencias del código del proyecto.
- **Safely Delete:** Busca las referencias a un elemento y lo borra solo si no hay referencias a él.
- **Change Method Parameters:** Permite añadir parámetros y modificar los accesos.
- **Extract Interface:** Seleccionas un conjunto de métodos y te crea una interfaz a partir de ellos.
- **Extract Superclass:** Crea una nueva clase abstracta. Cambia la clase actual para que extienda la nueva y mueve los métodos y atributos a la nueva clase.
- **Move Inner to Outer Level:** Mueve una clase interna un nivel hacia arriba en la jerarquía de clases.

# Refactoring en NetBeans

5

- **Copy Class:** Copia una clase a otro paquete o clase.
- **Pull Up:** Permite subir un método o campo a otra clase de la cual hereda la clase que contiene el método o campo que deseamos subir.
- **Pull Down:** Permite bajar una clase anidada, método o campo a otra clase la cual hereda de la clase que contiene a la clase anidada, método o campo que deseamos bajar.
- **Use Supertype Where Possible:** Despliega al programador todas las clases que extiende de la clase actual. El programador seleccionará una y NetBeans buscará en todo el proyecto referencias a la clase que se quiere refactorizar, si encuentra referencias, determinará si es posible utilizar la superclase seleccionada.

# Refactoring en NetBeans

6

- **Convert Anonymous to Member:** se usa para separar una clase interna anónima en una clase interna real. Hay varios tipos de clases anónimas:
  - ▣ Clase interna para definir e instanciar una instancia de una subclase sin nombre.
  - ▣ Clase interna para definir e instanciar una implementación anónima de una interfaz
  - ▣ Clase interna anónima definida como un argumento para un método.

# Refactoring en NetBeans -Ejemplo

7

## □ Recordamos:

- ▣ **La refactorización** es una disciplina técnica, que consiste en realizar pequeñas transformaciones en el código de un programa, para mejorar la estructura sin que cambie el comportamiento ni funcionalidad de este.
- ▣ **OBJETIVO:** mejorar la estructura interna del código
- Ejemplo: en la carpeta compartida descarga el proyecto de NetBeans refactor01.zip
- Abre el proyecto en el IDE NetBeans, comprueba las versiones y ejecuta el proyecto antes de realizar los cambios.

# Refactoring en NetBeans -Ejemplo

8

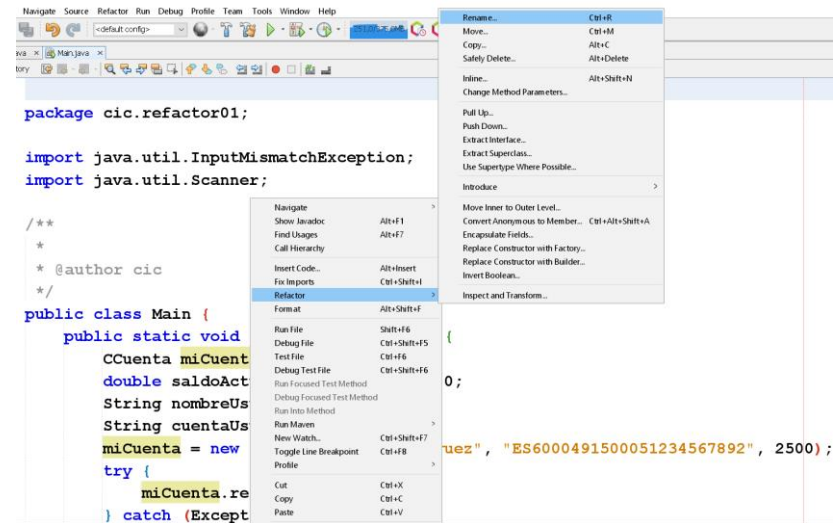
- hay definida una clase CCuenta, que cuenta con una serie de atributos y métodos. El proyecto cuenta asimismo con una clase Main, donde se hace uso de la clase descrita.
- Cambios a realizar:
  - ▣ 1.-Cambia el nombre de la variable “miCuenta” por “cuenta1”
  - ▣ 2.-Introduce el método “operativa\_cuenta”, que englobe las sentencias de la clase Main que operan con el objeto cuenta1.
  - ▣ 3.-Encapsula los cuatro atributos de la clase Ccuenta
  - ▣ 4.-Añadir un nuevo parámetro al método operativa\_cuenta, de nombre cantidad y de tipo float



# Refactoring en NetBeans -Ejemplo

9

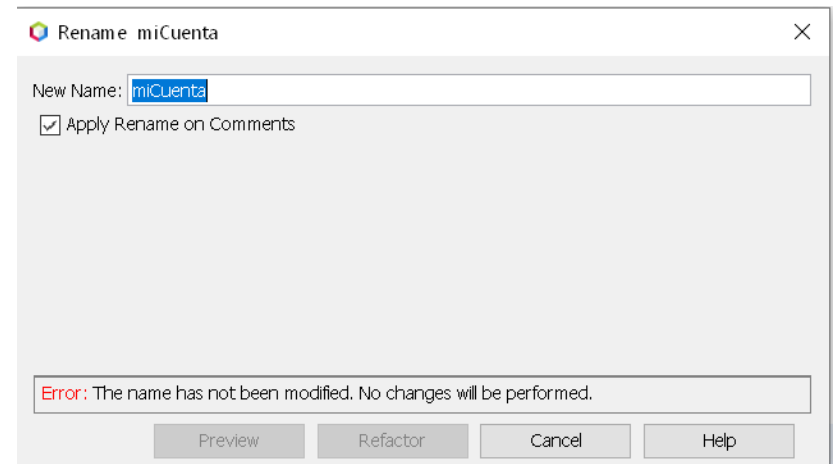
- 1.-Cambia el nombre de la variable “miCuenta” por “cuenta1”
  - Para cambiar el nombre de la variable “miCuenta” por “cuenta1” seleccionamos la variable a cambiar y pulsamos sobre ella con el botón derecho del ratón. Seleccionamos **Refactor** y en el nuevo submenú pulsamos en **Rename**, apareciendo la siguiente pantalla.



# Refactoring en NetBeans -Ejemplo

10

- Escribimos “cuenta1” como nuevo nombre y marcamos la casilla que nos permite incluir el renombrado incluso en los comentarios (por si acaso). Pulsamos sobre el botón **Preview** o el **Refactor**.



# Refactoring en NetBeans -Ejemplo

11

- Antes de realizar el cambio con el botón **Preview** podemos observar todos los cambios que se van a realizar y si estamos de acuerdo damos al botón **Do Refactoring**

The screenshot shows the NetBeans IDE with a refactoring operation in progress. On the left, the 'Refactoring' dialog is open, showing the 'Rename' tab. The operation is 'Rename **miCuenta** to **cuenta1** [5 occurrences]'. The 'Source Packages' list includes 'refactor01', 'cic.refactor01', 'Main.java', and 'Main'. The 'main' method is selected. The 'Rename variable miCuenta' checkbox is checked, and the 'Update reference to miCuenta' checkbox is also checked. At the bottom of the dialog are 'Do Refactoring' and 'Cancel' buttons.

The main editor displays a side-by-side comparison of the code before and after the refactoring. The left pane shows the original code in 'Main.java' and the right pane shows the 'Refactored Main.java'. The changes are as follows:

Line	Original Code (Main.java)	Refactored Code (Refactored Main.java)
15	<code>public class Main {</code>	<code>public class Main {</code>
16	<code>public static void main(Str</code>	<code>public static void main(Strin</code>
17	<code>CCuenta <b>miCuenta</b>, miCue</code>	<code>CCuenta <b>cuenta1</b>, miCuenta</code>
18	<code>double saldoActual, sal</code>	<code>double saldoActual, saldo</code>
19	<code>String nombreUsuario =</code>	<code>String nombreUsuario = "</code>
20	<code>String cuentaUsuario =</code>	<code>String cuentaUsuario = "</code>
21	<code><b>miCuenta</b> = new CCuenta(</code>	<code><b>cuenta1</b> = new CCuenta("Lu</code>
22	<code>try {</code>	<code>try {</code>
23	<code><b>miCuenta</b>.retirar(2300)</code>	<code><b>cuenta1</b>.retirar(2300)</code>
24	<code>} catch (Exception e) {</code>	<code>} catch (Exception e) {</code>
25	<code>System.out.print("F</code>	<code>System.out.print("Fal</code>
26	<code>}</code>	<code>}</code>
27	<code></code>	<code></code>
28	<code>try {</code>	<code>try {</code>
29	<code>System.out.println(</code>	<code>System.out.println("I</code>
30	<code><b>miCuenta</b>.ingresar(695)</code>	<code><b>cuenta1</b>.ingresar(695)</code>
31	<code>} catch (Exception e) {</code>	<code>} catch (Exception e) {</code>
32	<code></code>	<code></code>

# Refactoring en NetBeans -Ejemplo

12

```
public static void main(String[] args) {  
    CCuenta cuenta1, miCuenta2;  
    double saldoActual, saldoUsuario = 0;  
    String nombreUsuario = "";  
    String cuentaUsuario = "";  
    cuenta1 ← new CCuenta("Luis Rodriguez", "ES6000491500051234567892", 2500);  
    try {  
        cuenta1.retirar(2300);  
    } catch (Exception e) {  
        System.out.print("Fallo al retirar");  
    }  
  
    try {  
        System.out.println("Ingreso en cuenta");  
        cuenta1.ingresar(695);  
    } catch (Exception e) {  
        System.out.print("Fallo al ingresar");  
    }  
    saldoActual = cuenta1.estado();  
    System.out.println("El saldo actual es " + saldoActual);  
}
```

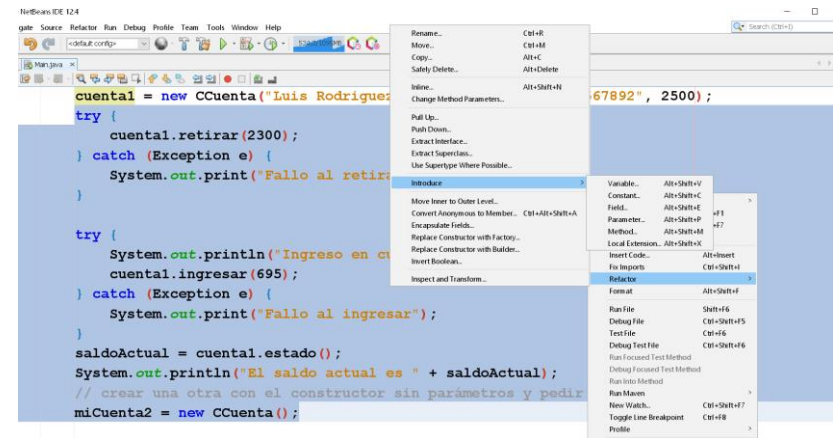
## Resultado del Refactor o Do Refactoring

Podemos apreciar como se ha modificado el nombre de la variable en todos los lugares donde antes estaba "miCuenta".

# Refactoring en NetBeans -Ejemplo

13

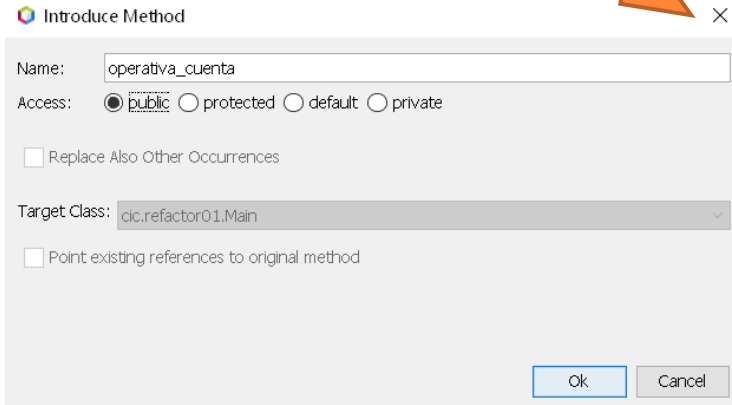
- **2.-Introduce el método “operativa\_cuenta”, que englobe las sentencias de la clase Main que operan con el objeto cuenta1**
- Para introducir el método “operativa\_cuenta”, seleccionamos los dos bloques try-catch de main y en el menú **Refactor** pulsamos sobre **Introduce** donde seleccionamos **Method**



# Refactoring en NetBeans -Ejemplo

14

- Escribimos  
“operativa\_cuenta”  
como nombre.  
Pulsamos sobre el  
botón **OK**.



Introduce Method

Name:

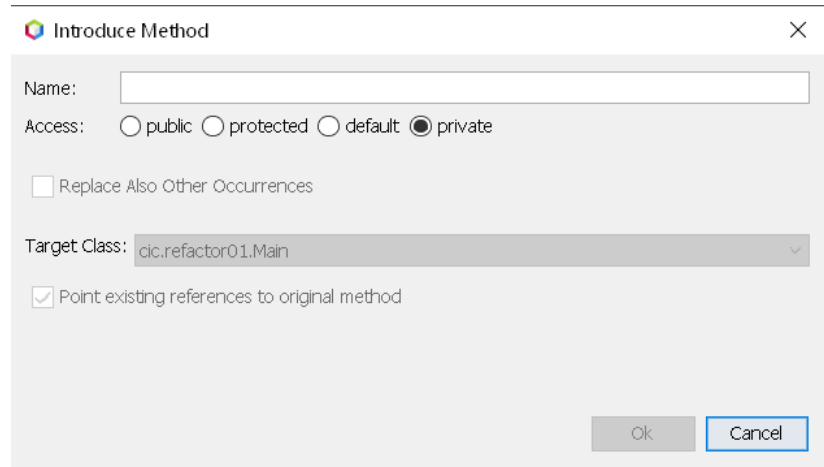
Access: ☒ public ☐ protected ☐ default ☐ private

☐ Replace Also Other Occurrences

Target Class:

☐ Point existing references to original method

Ok Cancel



Introduce Method

Name:

Access: ☐ public ☐ protected ☐ default ☒ private

☐ Replace Also Other Occurrences

Target Class:

☒ Point existing references to original method

Ok Cancel

# Refactoring en NetBeans -Ejemplo

15

```
public static CCuenta operativa_cuenta(CCuenta cuenta1) {  
    double saldoActual;  
    CCuenta miCuenta2;  
    try {  
        cuenta1.retirar(2300);  
    } catch (Exception e) {  
        System.out.print("Fallo al retirar");  
    }  
    try {  
        System.out.println("Ingreso en cuenta");  
        cuenta1.ingresar(695);  
    } catch (Exception e) {  
        System.out.print("Fallo al ingresar");  
    }  
    saldoActual = cuenta1.estado();  
    System.out.println("El saldo actual es " + saldoActual);  
    // crear una otra con el constructor sin parámetros y pedir los datos  
    miCuenta2 = new CCuenta();  
    return miCuenta2;  
}
```

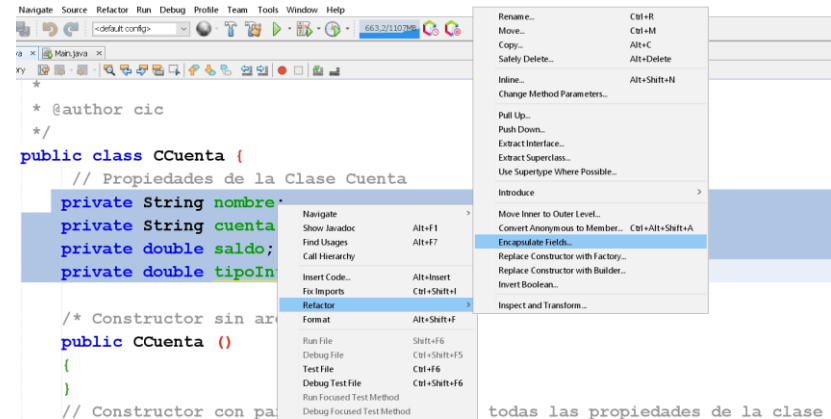


```
miCuenta2 = operativa_cuenta(cuenta1);
```

# Refactoring en NetBeans -Ejemplo

16

- **3.-Encapsula los cuatro atributos de la clase CCuenta**
  - ▣ Para encapsular los atributos de la clase **CCuenta** los pondremos todos de **tipo protected** con lo que sólo podrán ser accedidos desde la propia clase y desde aquellas que la hereden. También pondremos todos los **métodos getter/setter** de los atributos de clase, aunque ya existen algunos puestos (asignarNombre, obtenerNombre, etc).
  - ▣ Para hacer esto último procederemos a seleccionar la opción **Refactor** y escogeremos **encapsule fields**

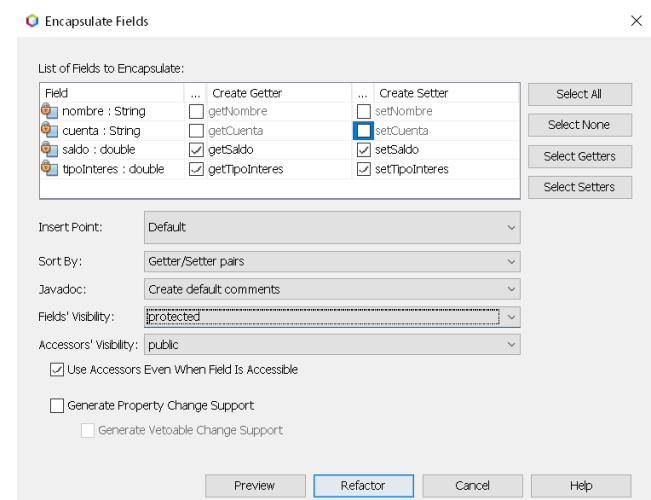
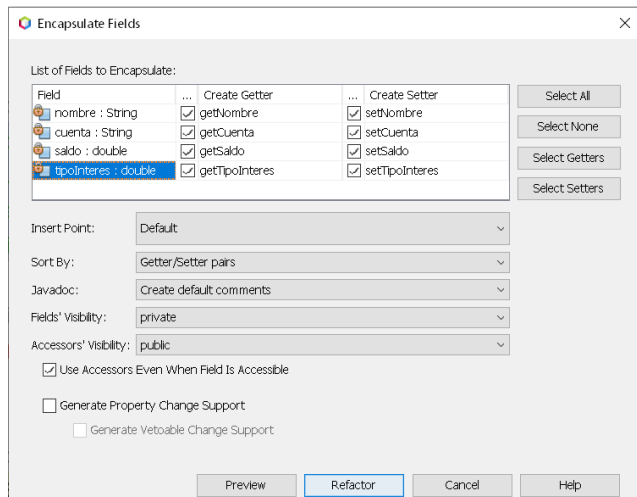




# Refactoring en NetBeans -Ejemplo

17

- En la nueva pantalla activamos las casillas de los métodos que nos faltan, es decir, el getter de tipoInteres y cuenta y el setter de saldo y tipoInteres y cambiamos la visibilidad de los campos a protected.
- El resto de opciones las dejamos por defecto y pulsamos sobre **Refactor**



# Refactoring en NetBeans -Ejemplo

18



```
/**
 * @return the saldo
 */
public double getSaldo() {
    return saldo;
}

/**
 * @param saldo the saldo to set
 */
public void setSaldo(double saldo) {
    this.saldo = saldo;
}

/**
 * @return the tipoInteres
 */
public double getTipoInteres() {
    return tipoInteres;
}

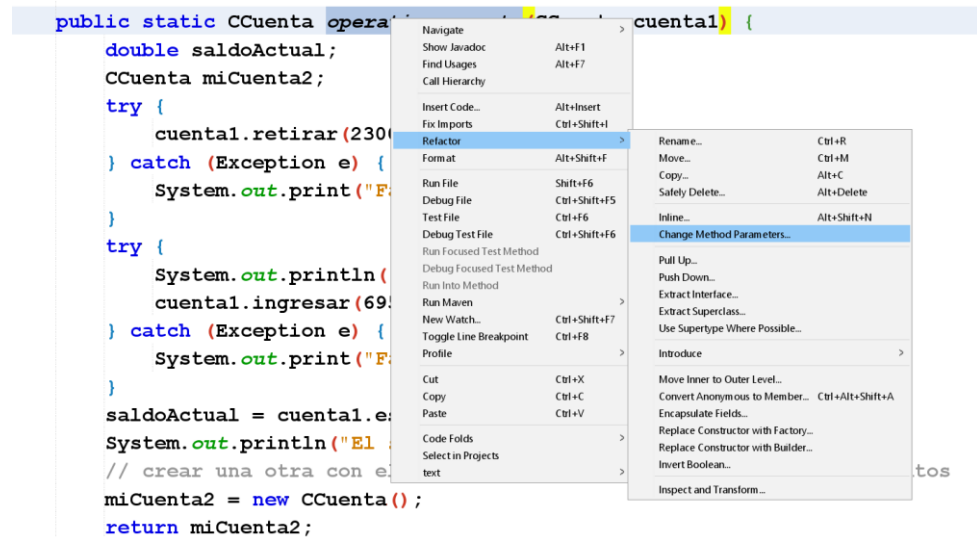
/**
 * @param tipoInteres the tipoInteres to set
 */
public void setTipoInteres(double tipoInteres) {
    this.tipoInteres = tipoInteres;
}
```

# Refactoring en NetBeans -Ejemplo

19

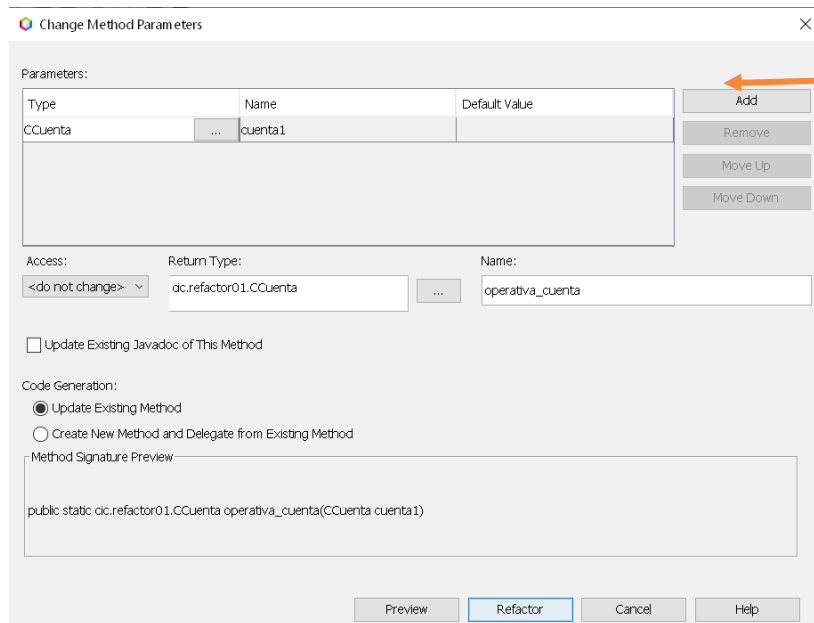
- 4.-Añadir un nuevo parámetro al método operativa\_cuenta, de nombre cantidad y de tipo float.

- Para añadir un nuevo parámetro nos colocamos sobre el método operativa\_cuenta y pulsamos en **Refactor** y seleccionamos **Change Method Parameters**



# Refactoring en NetBeans -Ejemplo

20



The dialog box 'Change Method Parameters' is shown. It has a 'Parameters' table with columns 'Type', 'Name', and 'Default Value'. The first row shows 'CCuenta' as the type and 'cuenta1' as the name. To the right of the table are buttons: 'Add', 'Remove', 'Move Up', and 'Move Down'. Below the table, there are fields for 'Access' (set to '<do not change>'), 'Return Type' (set to 'dc.refactor01.CCuenta'), and 'Name' (set to 'operativa\_cuenta'). There is a checkbox for 'Update Existing Javadoc of This Method'. Under 'Code Generation', the 'Update Existing Method' radio button is selected. A 'Method Signature Preview' box shows the code: 'public static dc.refactor01.CCuenta operativa\_cuenta(CCuenta cuenta1)'. At the bottom are buttons for 'Preview', 'Refactor', 'Cancel', and 'Help'.

Type	Name	Default Value
CCuenta	cuenta1	

Buttons: Add, Remove, Move Up, Move Down

Access: <do not change> Return Type: dc.refactor01.CCuenta Name: operativa\_cuenta

☐ Update Existing Javadoc of This Method

Code Generation:  
☒ Update Existing Method  
☐ Create New Method and Delegate from Existing Method

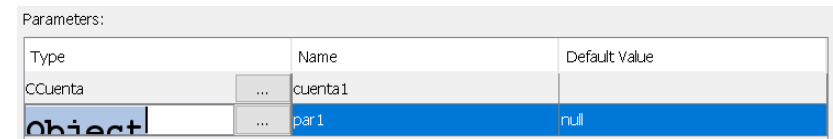
Method Signature Preview:  
public static dc.refactor01.CCuenta operativa\_cuenta(CCuenta cuenta1)

Buttons: Preview, Refactor, Cancel, Help

Pulsamos el botón **Add** y tecleamos **cantidad** sobre **par1**, **float** sobre **Object** y **0(cero)** sobre **null**.

Para cambiar estos valores hacemos doble clic sobre cada uno de los términos.

Pulsamos sobre **Refactor** y ya tendremos el nuevo parámetro añadido al método.



The dialog box is shown again, but now the 'Parameters' table has two rows. The first row is 'CCuenta' and 'cuenta1'. The second row, which is highlighted in blue, has 'Object' as the type and 'par1' as the name, with 'null' in the 'Default Value' column.

Type	Name	Default Value
CCuenta	cuenta1	
Object	par1	null

# Refactoring en NetBeans -Ejemplo

21

Change Method Parameters

Parameters:

Type	Name	Default Value
CCuenta	...	cuenta1
float	...	cantidad

Buttons: Add, Remove, Move Up, Move Down

Access: <do not change> Return Type: dc.refactor01.CCuenta Name: ...

operativa\_cuenta

☐ Update Existing Javadoc of This Method

Code Generation:

☒ Update Existing Method  
☐ Create New Method and Delegate from Existing Method

Method Signature Preview

```
public static dc.refactor01.CCuenta operativa_cuenta(CCuenta cuenta1, float cantidad)
```

Buttons: Preview, Refactor, Cancel, Help

```
public static CCuenta operativa_cuenta(CCuenta cuenta1, float cantidad) {  
    double saldoActual;  
    CCuenta miCuenta2;  
    try {  
        cuenta1.retirar(2300);  
    } catch (Exception e) {  
        System.out.print("Fallo al retirar");  
    }  
    try {  
        System.out.println("Ingreso en cuenta");  
        cuenta1.ingresar(695);  
    } catch (Exception e) {  
        System.out.print("Fallo al ingresar");  
    }  
    saldoActual = cuenta1.estado();  
    System.out.println("El saldo actual es " + saldoActual);  
    // crear una otra con el constructor sin parámetros y pedir los datos  
    miCuenta2 = new CCuenta();  
    return miCuenta2;  
}
```