

Tarea Unidad 3

Diseño y Realización de Pruebas

Entornos de Desarrollo

Curso DAM 2024/25

Calificación obtenida: 10/10

Índice

Detalles de la tarea	3
1. Análisis de Caja Blanca del Método ingresar	5
1.1. Creación del Grafo de Nodos	5
1.2. Cálculo de Complejidad Ciclomática	6
1.3. Definición de Caminos de Prueba	7
1.4. Resultados Esperados de los Caminos	8
1.5. Casos de Prueba	9
2. Análisis de Caja Negra del Método retirar	10
2.1. Clases de Equivalencia	10
2.2. Análisis de Valores Límite	10
2.3. Resultados Esperados	11
3. Creación de Casos de Prueba con JUnit	12
3.1. Clase CCuentaTest	12
3.2. Capturas de pantalla	14
3.3. Análisis de resultados	15
4. Depuración del Método ingresar	15
4.1. Configuración de Puntos de Ruptura	15
4.2. Exportación de Puntos de Ruptura	17
4.3. Capturas y Análisis	19

Detalles de la tarea

Introducción

En esta tarea trabajarás con una clase Java llamada CCuenta, que contiene tres métodos principales: main, ingresar y retirar. El objetivo es analizar el comportamiento de estos métodos mediante técnicas de pruebas de software (caja blanca y caja negra), desarrollar casos de prueba con JUnit y aplicar depuración para identificar y resolver errores.

Esta actividad tiene como propósito fortalecer tus habilidades en programación y análisis de software. Si encuentras dificultades, revisa este enunciado cuidadosamente, sigue los pasos indicados y utiliza el foro para plantear tus dudas de forma concreta y específica.

Código Base

A continuación, se proporciona el código de base para desarrollar la tarea. Este código debe ser copiado e importado correctamente en tu entorno de desarrollo (NetBeans o Eclipse recomendado):

Método main

```
public static void main(String[] args) {
    CCuenta miCuenta = new CCuenta();
    System.out.println("Saldo inicial: " + miCuenta.dSaldo + "
    euros");
    miCuenta.ingresar(-100); // Depuración: entrada inválida
    System.out.println("Saldo inicial: " + miCuenta.dSaldo +
    "euros");
    miCuenta.ingresar(100);
    System.out.println("Saldo tras ingreso: " + miCuenta.dSaldo +
    " euros");
    miCuenta.ingresar(200);
    System.out.println("Saldo tras ingreso: " + miCuenta.dSaldo +
    " euros");
    miCuenta.ingresar(300); // Depuración: tercera actualización
    System.out.println("Saldo tras ingreso: " + miCuenta.dSaldo +
    " euros");
    miCuenta.retirar(50);
    System.out.println("Saldo tras retirada: " + miCuenta.dSaldo +
    " euros");
}
```

Método ingresar

```
public int ingresar(double cantidad) {
    int iCodErr;
```

```

        if (cantidad < 0) {
            System.out.println("No se puede ingresar una cantidad
negativa");
            iCodErr = 1;
        } else if (cantidad == -3) {
            System.out.println("Error detectable en pruebas de caja
blanca");
            iCodErr = 2;
        } else {
            dSaldo = dSaldo + cantidad;
            iCodErr = 0;
        }
        return iCodErr;
    }
}

```

Método retirar

```

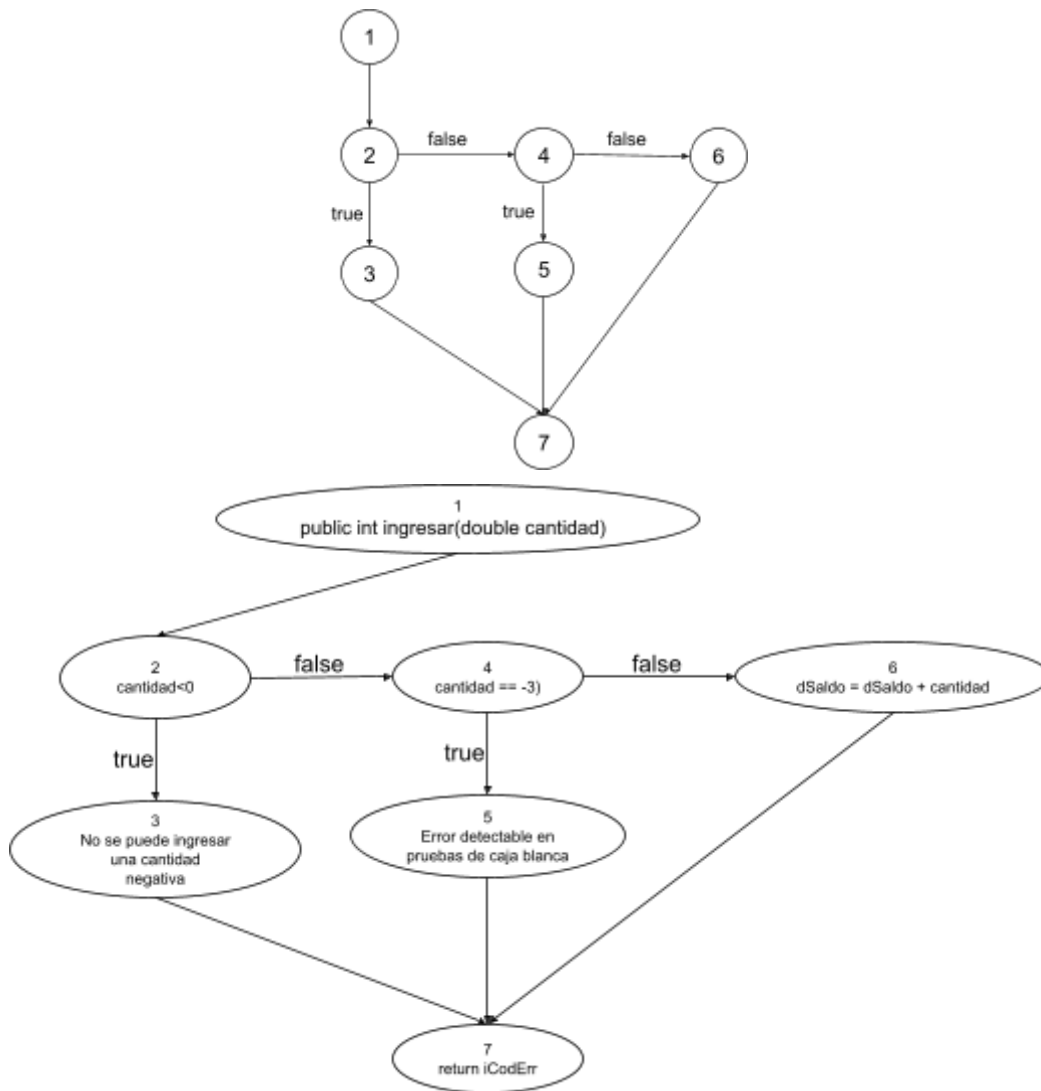
public void retirar(double cantidad) {
    if (cantidad <= 0) {
        System.out.println("No se puede retirar una cantidad
negativa");
    } else if (dSaldo < cantidad) {
        System.out.println("No hay suficiente saldo");
    }
}
}

```

1. Análisis de Caja Blanca del Método ingresar

1.1. Creación del Grafo de Nodos

Dibuja un grafo que represente las líneas-condición del método ingresar. Indica claramente los nodos, aristas y condiciones.



NODO		LÍNEAS	CÓDIGO
1	Inicio del método	1	public int ingresar(double cantidad) {
2	Evaluación de cantidad < 0	3	if (cantidad < 0) {
3	Acción cuando cantidad < 0	4-5	System.out.println("No se puede ingresar una cantidad negativa"); iCodErr = 1;
4	Evaluación de cantidad == -3	6	} else if (cantidad == -3) {

5	Acción cuando cantidad == -3	7-8	System.out.println("Error detectable en pruebas de caja blanca"); iCodErr = 2;
6	Acción para cantidad >= 0 && cantidad != -3	10-11	dSaldo = dSaldo + cantidad; iCodErr = 0;
7	Fin del método y retorno del valor.	13	return iCodErr;

1.2. Cálculo de Complejidad Ciclomática

Calcula la complejidad ciclomática utilizando:

- El número de regiones en el grafo.
- La fórmula: Número de aristas - Número de nodos + 2.
- La fórmula: Número de condiciones + 1.

A partir del grafo se determina su complejidad ciclomática. Es posible hacerlo por tres métodos diferentes, pero todos ellos han de dar el mismo resultado.

Método 1

$V(G) = a - n + 2$, siendo a el número de arcos o aristas del grafo y n el número de nodos.

aristas = 8

nodos = 7

$V(G) = 8 - 7 + 2$

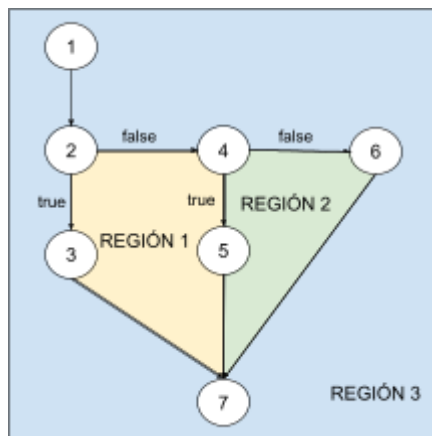
$V(G) = 3$

Método 2:

$V(G) = r$, siendo r el número de regiones cerradas del grafo (incluida la externa).

regiones = 3

$V(G) = 3$



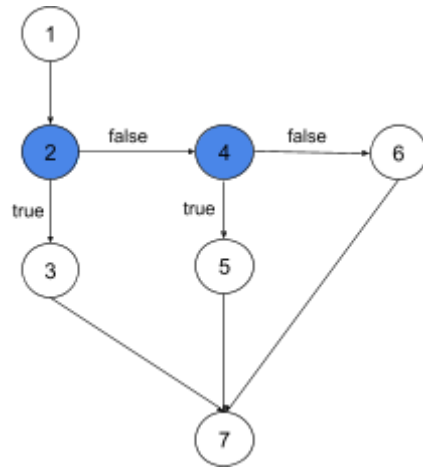
Método 3:

$V(G) = c + 1$, siendo c el número de nodos de condición.

nodos de condición = 2

$$V(G) = 2 + 1$$

$$V(G) = 3$$



1.3. Definición de Caminos de Prueba

Enumera los caminos posibles desde el grafo y detalla los casos de uso asociados.

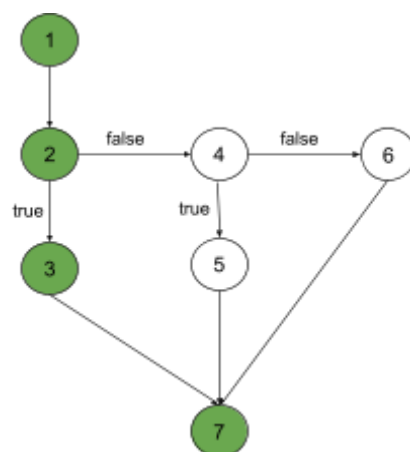
El número de caminos de prueba debe ser igual a la complejidad calculada. Consiste en hacer recorridos desde el inicio hasta el final del método con el que se esté trabajando. Se irán registrando los nodos por los que va pasando la ejecución del camino.

Cada nuevo camino deberá aportar el paso por nuevas aristas/nodos del grafo. La definición de caminos se hará desde los más sencillos a los más complicados.

$V(G) = 3$ por lo tanto **Número de caminos de prueba = 3**

CAMINO DE PRUEBA 1

Nodos 1 - 2 - 3 - 7



CAMINO DE PRUEBA 2	
Nodos 1 - 2 - 4 - 5 - 7	<pre> graph TD 1((1)) --> 2((2)) 2 -- false --> 4((4)) 4 -- true --> 5((5)) 5 --> 7((7)) 2 -- true --> 3((3)) 3 --> 7 4 -- false --> 6((6)) 6 --> 7 </pre>
CAMINO DE PRUEBA 3	
Nodos 1 - 2 - 4 - 6 - 7	<pre> graph TD 1((1)) --> 2((2)) 2 -- false --> 4((4)) 4 -- false --> 6((6)) 6 --> 7((7)) 2 -- true --> 3((3)) 3 --> 7 4 -- true --> 5((5)) 5 --> 7 </pre>

1.4. Resultados Esperados de los Caminos

Describe qué debería suceder en cada camino identificado, incluyendo mensajes y códigos de error.

El método ingresar funciona de la siguiente manera: Recibe como entrada un número decimal (cantidad), que es la cantidad de dinero que se quiere ingresar, y devuelve un número entero (int), que es un código de error para indicar si la operación fue exitosa o si hubo un problema. Declara una variable llamada iCodErr. Esta variable almacenará el código de error: 0 significa que todo está bien y 1 o 2 indican que hubo un error específico.

Camino 1: Revisa si la cantidad que se quiere ingresar es menor que 0. Si esto es cierto, significa que se intentó ingresar una cantidad negativa. Muestra un mensaje

en la consola diciendo que “No se puede ingresar una cantidad negativa”. Asigna el código de error 1 a iCodErr. Se muestra el código y finaliza el programa.

Camino 2: Revisa si la cantidad que se quiere ingresar es menor que 0. Si esto es falso, revisa si la cantidad que se quiere ingresar es exactamente -3. Si la cantidad es -3 muestra un mensaje en la consola diciendo “Error detectable en pruebas de caja blanca” y asigna el código de error 2 a iCodErr. Se muestra el código y finaliza el programa.

Camino 3: Revisa si la cantidad que se quiere ingresar es menor que 0. Si esto es falso, revisa si la cantidad que se quiere ingresar es exactamente -3. Si esto es falso significa que la cantidad que se quiere ingresar es mayor o igual a 0 y no es -3. Añade la cantidad a la variable dSaldo, que representa el saldo de la cuenta, y asigna el código de error 0 a iCodErr, indicando que todo salió bien. Se muestra el código y finaliza el programa.

1.5. Casos de Prueba

Define los datos de entrada y salida esperados para los caminos establecidos en 1.3 y 1.4.

CAMINO	DATOS DE ENTRADA	SALIDA ESPERADA	
		MENSAJE	CÓDIGO DE ERROR
1	cantidad < 0	No se puede ingresar una cantidad negativa	1
2	cantidad >= 0 && cantidad == -3	Error detectable en pruebas de caja blanca	2
3	cantidad >= 0 && cantidad != -3	dSaldo = dSaldo + cantidad (Se añade la cantidad a la variable dSaldo, que representa el saldo de la cuenta)	0

2. Análisis de Caja Negra del Método retirar

2.1. Clases de Equivalencia

Identifica las clases de equivalencia válidas e inválidas.

DOMINIO	CLASES DE EQUIVALENCIA		TIPO
	Nº DE CLASE	CLASE	
Números negativos distintos de -3	CL1	$(-\infty, 0)$	NO VÁLIDA
Número negativo igual a -3	CL2	$\{-3\}$	NO VÁLIDA
Cero	CL3	$[0]$	VÁLIDA
Números positivos	CL4	$(0, +\infty)$	VÁLIDA

2.2. Análisis de Valores Límite

Considera los valores extremos para cada clase de equivalencia.

DOMINIO	CLASE	TIPO	LÍMITE INFERIOR	LÍMITE SUPERIOR
Números negativos distintos de -3	CL1 $(-\infty, 0)$	NO VÁLIDA	$-\infty$	-0.01
Número negativo igual a -3	CL2 $\{-3\}$	NO VÁLIDA	-3	-3
Cero	CL3 $[0]$	VÁLIDA	0	0
Números positivos	CL4 $(0, +\infty)$	VÁLIDA	0.01	$+\infty$

2.3. Resultados Esperados

Especifica los resultados que se esperan para cada caso identificado en las clases de equivalencia y valores límite.

Nº CASO DE PRUEBA	CLASE	TIPO	LÍMITE	CANTIDAD	RESULTADOS ESPERADOS
CP1	CL1	NO VÁLIDA	Límite Inferior -0.01	-0.01	Error 1
CP2	CL1	NO VÁLIDA	Extremo $-\infty$	-1000	Error 1
CP3	CL2	NO VÁLIDA	Exacto -3	-3	Error 2
CP4	CL3	VÁLIDA	Exacto 0	0	Error 0
CP5	CL4	VÁLIDA	Extremo $+\infty$	1000	Error 0
CP6	CL4		Límite Inferior 0.01	0.01	Error 0

Clases de equivalencia no válidas (CL1 y CL2):

- Estas clases generan errores porque representan casos fuera del dominio permitido.
- Ejemplo: valores negativos diferentes de -3 o exactamente -3.

Clases de equivalencia válidas (CL3 y CL4):

- Estas clases cubren los valores aceptados por el método.
- Ejemplo: 0 (CL3) y valores positivos (CL4).

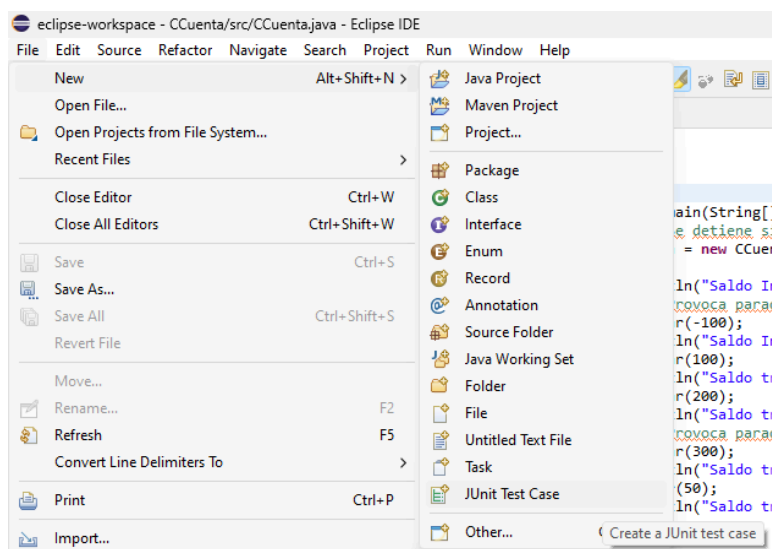
3. Creación de Casos de Prueba con JUnit

3.1. Clase CCuentaTest

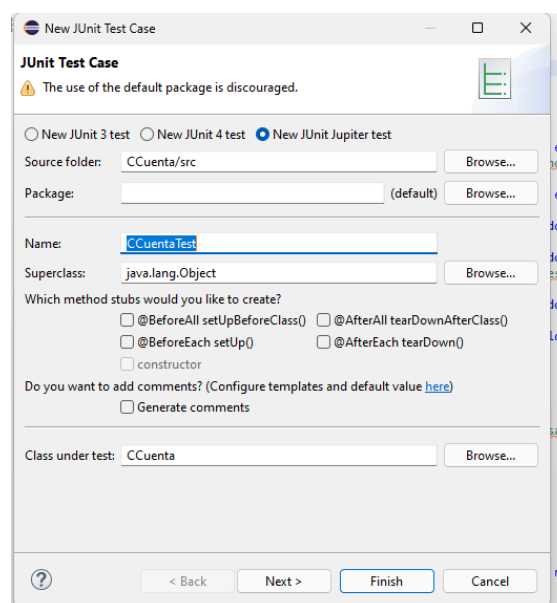
Crea una clase JUnit en Eclipse o NetBeans que implemente pruebas para los casos identificados en la sección de caja blanca.

CREAR UNA CLASE JUNIT EN ECLIPSE

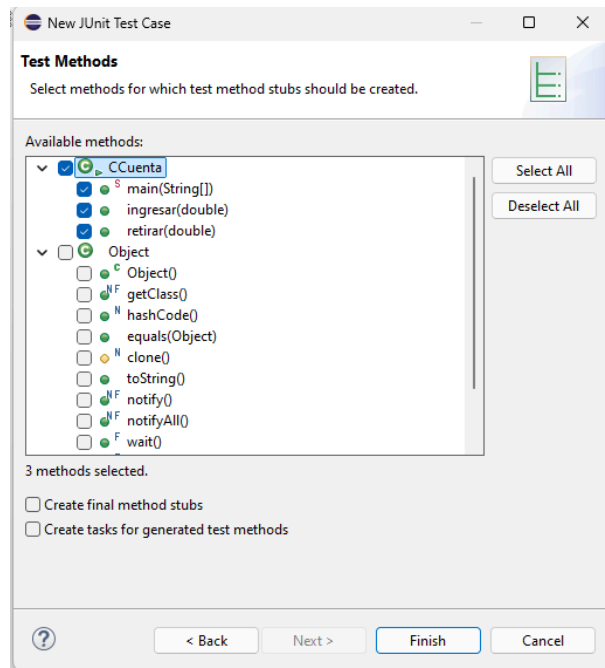
Sobre el nombre del proyecto elegir la opción new/JUnit Test



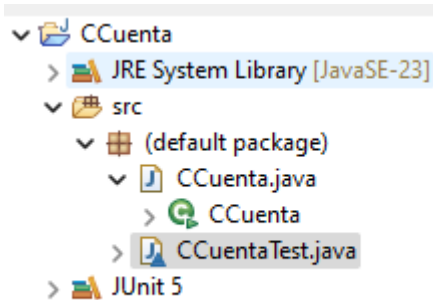
Poner el nombre de la clase acabado en Test



Seleccionar los métodos



Se crea un fichero java para hacer las pruebas



Pego el código proporcionado en la tarea en el archivo creado

```
CCuenta.java CCuentaTest.java x
1 import static org.junit.jupiter.api.Assertions.*;
2
3 import org.junit.jupiter.api.DisplayName;
4 import org.junit.jupiter.params.ParameterizedTest;
5 import org.junit.jupiter.params.provider.CsvSource;
6
7 class CCuentaTest {
8     CCuenta miCuenta = new CCuenta();
9
10    @ParameterizedTest
11    @CsvSource({"-10,1","-3,2","10,0"})
12    @DisplayName("Caja Blanca - Ingresar")
13    void testIngreso(double cant,int resul) {
14        assertEquals(resul,miCuenta.ingresar(cant));
15    }
16 }
17
```

3.2. Capturas de pantalla

Incluye capturas que muestren la creación y ejecución de los casos de prueba.

CREACIÓN Y EJECUCIÓN DE LOS CASOS DE PRUEBA

Creación de los casos de prueba

```
CCuenta.java  *CCuentaTest.java X
1 import static org.junit.jupiter.api.Assertions.*;
2
3 import org.junit.jupiter.api.DisplayName;
4 import org.junit.jupiter.params.ParameterizedTest;
5 import org.junit.jupiter.params.provider.CsvSource;
6
7 class CCuentaTest {
8     CCuenta miCuenta = new CCuenta();
9
10    @ParameterizedTest
11    @CsvSource({
12        "-0.01, 1", // CP1: CL1 - No válida, límite inferior
13        "-1000, 1", // CP2: CL1 - No válida, valor extremo (-∞)
14        "-3, 2", // CP3: CL2 - No válida, valor exacto (-3)
15        "0, 0", // CP4: CL3 - Válida, límite exacto (0)
16        "0.01, 0", // CP5: CL4 - Válida, límite inferior positivo
17        "1000, 0" // CP6: CL4 - Válida, valor extremo (+∞)
18    })
19    @DisplayName("Caja Blanca - Ingresar")
20    void testIngreso(double cant,int resul) {
21        assertEquals(resul,miCuenta.ingresar(cant));
22    }
23 }
24
```

Ejecución de los casos de prueba

Package Explorer JUnit X

Finished after 0,282 seconds

Runs: 6/6 Errors: 0 Failures: 1

CCuentaTest [Runner: JUnit 5] (0,005 s)

- ✓ Caja Blanca - Ingresar (0,005 s)
 - ✓ [1] -0.01, 1 (0,005 s)
 - ✓ [2] -1000, 1 (0,002 s)
 - ✗ [3] -3, 2 (0,018 s)
 - ✓ [4] 0, 0 (0,003 s)
 - ✓ [5] 0.01, 0 (0,003 s)
 - ✓ [6] 1000, 0 (0,001 s)

Failure Trace

org.opentest4j.AssertionFailedError: expected: <2> but was: <1>

3.3. Análisis de resultados

Explica qué indican los resultados de las pruebas y si los métodos funcionan según lo esperado.

La ejecución de los casos de prueba muestra que la mayoría de los escenarios cumplen con los resultados esperados, lo que sugiere que el método ingresar está implementado correctamente para los casos válidos y para la mayoría de los no válidos. Sin embargo, un caso (CP3) no ha pasado la prueba, lo que indica un posible problema en la lógica del método o en las expectativas definidas para este caso.

Casos de Prueba Exitosos:

- **CP1 y CP2 (Clase CL1 - No válida):** Los valores negativos distintos de -3 generan correctamente el resultado 1, indicando que la lógica para manejar estos casos funciona como se esperaba. Así pues, la validación de números negativos (no válidos) está correctamente implementada.
- **CP4 (Clase CL3 - Válida):** El valor 0 es aceptado y retorna el resultado esperado 0. Esto confirma que el método reconoce correctamente el límite inferior válido. Así pues, la lógica para manejar el valor 0 como un caso válido es correcta.
- **CP5 (Clase CL4 - Válida):** Los números positivos, incluyendo los valores límite (como 0.01 y valores muy grandes), devuelven el resultado esperado 0. Así pues, la lógica para manejar valores positivos está funcionando según lo esperado.

Caso de Prueba Fallido:

- **CP3 (Clase CL2 - No válida):** El valor específico -3 no ha devuelto el resultado esperado (2) y en su lugar ha devuelto 1. Así pues, el método ingresar no trata correctamente el caso específico de la clase de equivalencia CL2, donde -3 debería ser reconocido como un caso único y devolver el resultado 2.

4. Depuración del Método ingresar

4.1. Configuración de Puntos de Ruptura

Configura los siguientes puntos de ruptura en tu IDE:

- En la creación del objeto `miCuenta`.

- En la línea donde se verifica si `cantidad < 0`.
- En la tercera actualización de `dSaldo`.

CONFIGURAR PUNTOS DE RUPTURA

En la creación del objeto `miCuenta`.

```
5 public static void main(String[] args) {
6     // Depuracion. Se detiene siempre
7     CCuenta miCuenta = new CCuenta();
8 }
```

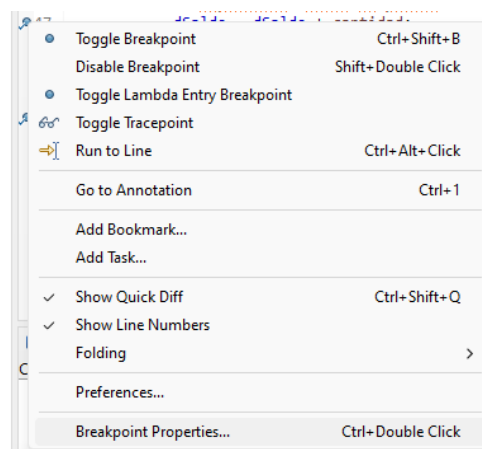
En la línea donde se verifica si `cantidad < 0`.

```
51 // Depuracion. Punto de parada cuando la cantidad es menor de 0
52 return iCodErr;
```

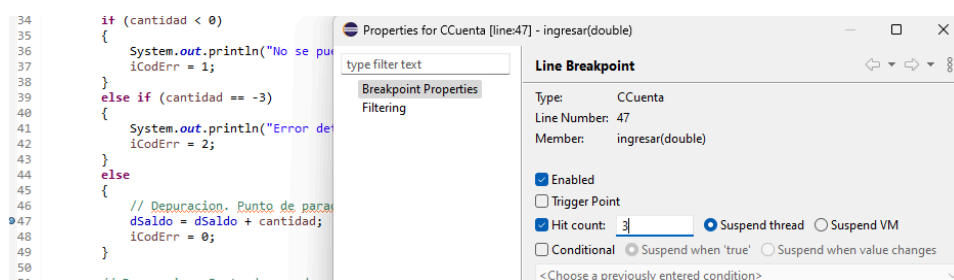
En la tercera actualización de `dSaldo`.

```
46 // Depuracion. Punto de parada. Solo en el 3 ingreso
47 dSaldo = dSaldo + cantidad;
48 iCodErr = 0;
```

Abrir ventana de propiedades haciendo clic derecho encima del punto de ruptura



Habilitar Hit Count y escribir 3 (porque quiero que se pare en la tercera actualización)

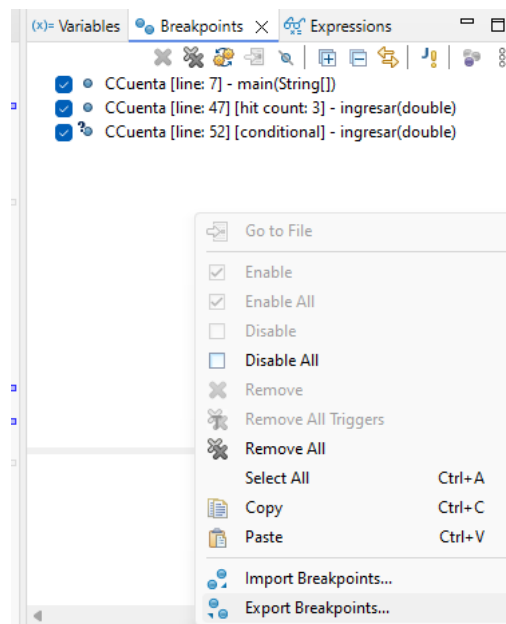


4.2. Exportación de Puntos de Ruptura

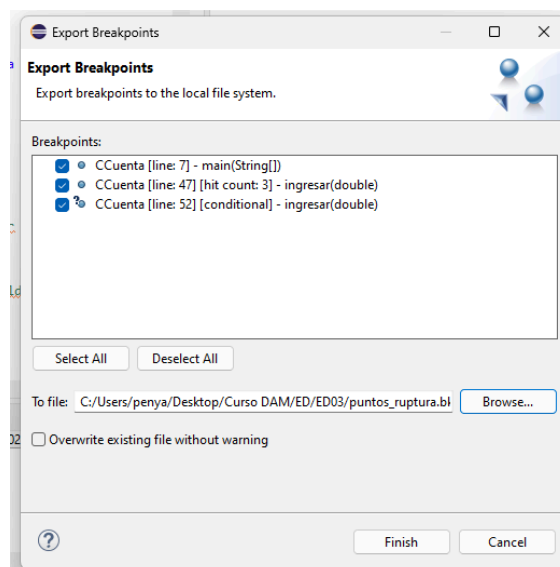
Exporta los puntos de ruptura desde tu IDE, conviértelos a formato .txt y adjúntalos en el documento copiando y pegando.

EXPORTAR PUNTOS DE RUPTURA

Botón derecho sobre la ventana



Seleccionar todos los puntos de ruptura y poner nombre al archivo



Abrir el archivo con el Bloc de notas, copiar el texto y pegarlo aquí

```
<?xml version="1.0" encoding="UTF-8"?>
<breakpoints>
<breakpoint enabled="true" persistent="true" registered="true">
<resource path="/CCuenta/src/CCuenta.java" type="1"/>
<marker charStart="117" lineNumber="7"
type="org.eclipse.jdt.debug.javaLineBreakpointMarker">
<attrib name="charStart" value="117"/>
<attrib name="org.eclipse.jdt.debug.core.suspendPolicy" value="2"/>
<attrib name="org.eclipse.jdt.debug.ui.JAVA_ELEMENT_HANDLE_ID"
value="/CCuenta/src/CCuenta.java[CCuenta]"/>
<attrib name="charEnd" value="155"/>
<attrib name="org.eclipse.debug.core.enabled" value="true"/>
<attrib name="message" value="Line breakpoint:CCuenta [line: 7] -
main(String[])"/>
<attrib name="org.eclipse.jdt.debug.core.installCount" value="0"/>
<attrib name="org.eclipse.debug.core.id" value="org.eclipse.jdt.debug"/>
<attrib name="org.eclipse.jdt.debug.core.typeName" value="CCuenta"/>
<attrib name="workingset_name" value=""/>
<attrib name="workingset_id"
value="org.eclipse.debug.ui.breakpointWorkingSet"/>
</marker>
</breakpoint>
<breakpoint enabled="true" persistent="true" registered="true">
<resource path="/CCuenta/src/CCuenta.java" type="1"/>
<marker charStart="1604" lineNumber="47"
type="org.eclipse.jdt.debug.javaLineBreakpointMarker">
<attrib name="charStart" value="1604"/>
<attrib name="org.eclipse.jdt.debug.core.suspendPolicy" value="2"/>
<attrib name="org.eclipse.jdt.debug.ui.JAVA_ELEMENT_HANDLE_ID"
value="/CCuenta/src/CCuenta.java[CCuenta]"/>
<attrib name="org.eclipse.jdt.debug.core.hitCount" value="3"/>
<attrib name="charEnd" value="1640"/>
<attrib name="org.eclipse.debug.core.enabled" value="true"/>
<attrib name="org.eclipse.jdt.debug.core.expired" value="false"/>
<attrib name="message" value="Line breakpoint:CCuenta [line: 47] [hit count: 3] -
ingresar(double)"/>
<attrib name="org.eclipse.jdt.debug.core.installCount" value="0"/>
<attrib name="org.eclipse.debug.core.id" value="org.eclipse.jdt.debug"/>
<attrib name="org.eclipse.jdt.debug.core.typeName" value="CCuenta"/>
<attrib name="workingset_name" value=""/>
<attrib name="workingset_id"
value="org.eclipse.debug.ui.breakpointWorkingSet"/>
</marker>
</breakpoint>
<breakpoint enabled="true" persistent="true" registered="true">
<resource path="/CCuenta/src/CCuenta.java" type="1"/>
<marker charStart="1765" lineNumber="52"
```

```

type="org.eclipse.jdt.debug.javaLineBreakpointMarker">
<attrib name="org.eclipse.jdt.debug.core.conditionEnabled" value="true"/>
<attrib name="charStart" value="1765"/>
<attrib name="org.eclipse.jdt.debug.core.suspendPolicy" value="2"/>
<attrib name="org.eclipse.jdt.debug.core.condition" value="cantidad &lt; 0"/>
<attrib name="org.eclipse.jdt.debug.ui.JAVA_ELEMENT_HANDLE_ID"
value="CCuenta/src&lt;{CCuenta.java[CCuenta]"/>
<attrib name="charEnd" value="1788"/>
<attrib name="org.eclipse.debug.core.enabled" value="true"/>
<attrib name="message" value="Line breakpoint:CCuenta [line: 52] [conditional] -
ingresar(double)"/>
<attrib name="org.eclipse.jdt.debug.core.installCount" value="0"/>
<attrib name="org.eclipse.debug.core.id" value="org.eclipse.jdt.debug"/>
<attrib name="org.eclipse.jdt.debug.core.typeName" value="CCuenta"/>
<attrib name="workingset_name" value=""/>
<attrib name="workingset_id"
value="org.eclipse.debug.ui.breakpointWorkingSet"/>
</marker>
</breakpoint>
</breakpoints>

```

4.3. Capturas y Análisis

Incluye capturas de pantalla de cada punto de ruptura en acción y describe qué verificaste en cada caso.

PUNTO DE RUPTURA 1

The screenshot shows the Eclipse IDE with the following components:

- Project Explorer:** Shows the project structure with 'CCuenta' and 'CCuentaTest.java'.
- Editor:** Displays the source code of 'CCuenta.java'. A breakpoint is set at line 9, which is highlighted in green. The code includes a static main method and several print statements for account operations.
- Console:** Shows the output of the program, including the initial balance and the results of deposit and withdrawal operations.
- Variables View:** Displays the state of the variables at the breakpoint. It shows 'args' as an array of strings, 'miCuenta' as an instance of 'CCuenta', and 'dSaldo' as a double value of 0.0.

Verificaciones punto de ruptura 1

Instanciación del objeto miCuenta: El objeto miCuenta se crea correctamente en la memoria.

(x)= Variables × Breakpoints Expressions	
Name	Value
<init>() returned	(No explicit return value)
args	String[0] (id=20)
miCuenta	CCuenta (id=23)
dSaldo	0.0

Valores iniciales de las propiedades: El valor inicial de las propiedades del objeto (en este caso, dSaldo) es 0.0 (porque es double).

(x)= Variables × Breakpoints Expressions	
Name	Value
println() returned	(No explicit return value)
args	String[0] (id=20)
miCuenta	CCuenta (id=23)
dSaldo	0.0

<Choose a previously entered expression>

Console × Problems Debug Shell

CCuenta [Java Application] C:\Program Files\Java\jdk-23\bin\javaw.exe

Saldo Inicial: 0.0 euros

PUNTO DE RUPTURA 2

Debug × Project Explorer

CCuenta [Java Application]

CCuenta at localhost:57629

Thread [main] (Suspended (breakpoint at line 52))

CCuenta.main(String[]) line: 11

CCuenta.java × CCuentaTest.java

```

27  /* Metodo para ingresar cantidades en la cuenta. Modifica el saldo.
28  * Este metodo va a ser probado con junit
29  */
30  public int ingresar(double cantidad)
31  {
32      int iCodErr;
33
34      if (cantidad < 0)
35      {
36          System.out.println("No se puede ingresar una cantidad negativa");
37          iCodErr = 1;
38      }
39      else if (cantidad == -3)
40      {
41          System.out.println("Error detectable en pruebas de caja blanca");
42          iCodErr = 2;
43      }
44      else
45      {
46          // Depuracion. Punto de parada. Solo en el 3 ingreso
47          dSaldo = dSaldo + cantidad;
48          iCodErr = 0;
49      }
50
51      // Depuracion. Punto de parada cuando la cantidad es menor de 0
52      return iCodErr;
53  }
54

```

(x)= Variables × Breakpoints Expressions

no method return value

this

CCuenta (id=23)

dSaldo

0.0

cantidad

-100.0

iCodErr

1

Console × Problems Debug Shell

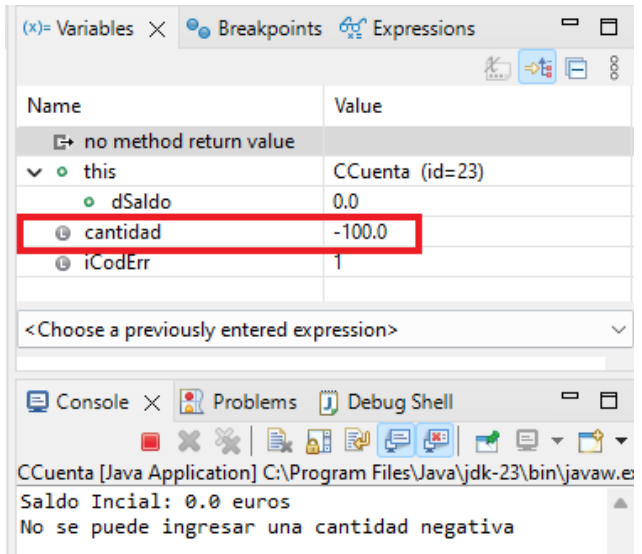
CCuenta [Java Application] C:\Program Files\Java\jdk-23\bin\javaw.exe

Saldo Inicial: 0.0 euros

No se puede ingresar una cantidad negativa

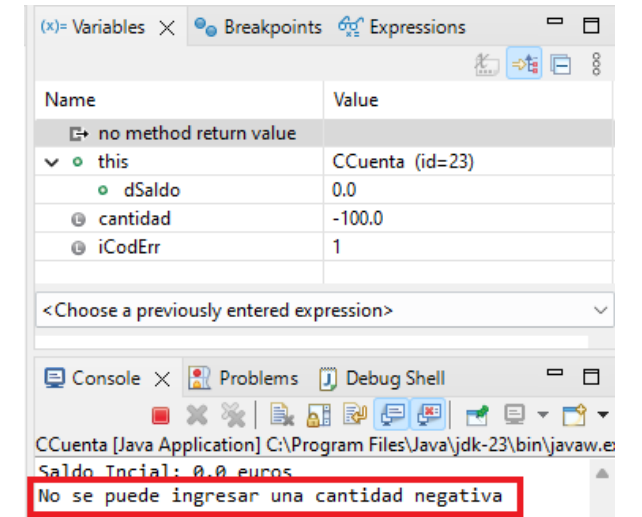
Verificaciones punto de ruptura 2

Valor de cantidad: Se muestra el valor de la variable cantidad que está siendo ingresado, en este caso -100.



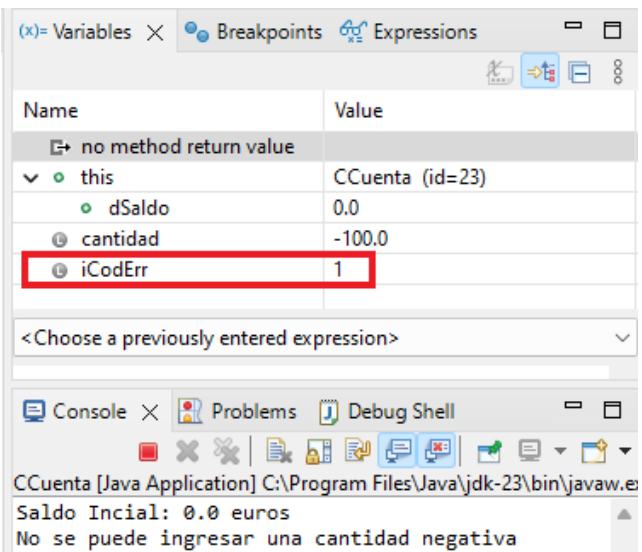
The screenshot shows the 'Variables' window in an IDE. The 'Name' column lists variables: 'no method return value', 'this', 'dSaldo', 'cantidad', and 'iCodErr'. The 'Value' column shows their respective values: 'no method return value' (no method return value), 'this' (CCuenta (id=23)), 'dSaldo' (0.0), 'cantidad' (-100.0), and 'iCodErr' (1). The 'cantidad' row is highlighted with a red box. Below the variables window, the 'Console' window shows the output: 'Saldo Inicial: 0.0 euros' and 'No se puede ingresar una cantidad negativa'.

Comprobación de la condición: Verifico que el flujo de ejecución entra correctamente. Como la condición es verdadera, se ejecuta el bloque de código que maneja las cantidades negativas, mostrando, en la consola, el mensaje “No se puede ingresar una cantidad negativa”.



The screenshot shows the 'Console' window in an IDE. The output text is 'Saldo Inicial: 0.0 euros' followed by 'No se puede ingresar una cantidad negativa'. The second line of output is highlighted with a red box.

Código de error: Verifico que el valor de retorno (iCodErr) es 1, ya que la cantidad introducida es menor que 0.



The screenshot shows the 'Variables' window in an IDE. The 'Name' column lists variables: 'no method return value', 'this', 'dSaldo', 'cantidad', and 'iCodErr'. The 'Value' column shows their respective values: 'no method return value' (no method return value), 'this' (CCuenta (id=23)), 'dSaldo' (0.0), 'cantidad' (-100.0), and 'iCodErr' (1). The 'iCodErr' row is highlighted with a red box. Below the variables window, the 'Console' window shows the output: 'Saldo Inicial: 0.0 euros' and 'No se puede ingresar una cantidad negativa'.

PUNTO DE RUPTURA 3

The screenshot shows an IDE with the following components:

- Project Explorer:** Shows the project structure with 'CCuenta [Java Application]' and 'CCuenta at localhost:57830'.
- Code Editor:** Displays the source code of 'CCuentaTest.java'. A breakpoint is set at line 47, which is highlighted. The code includes a method 'ingresar(double cantidad)' that updates the account balance.
- Variables View:** Shows the current state of variables:

Name	Value
no method return value	
this	CCuenta (id=23)
dSaldo	300.0
cantidad	300.0
- Console:** Displays the output of the application:


```
CCuenta [Java Application] C:\Program Files\Java\jdk-23\bin\javaw.exe
Saldo Inicial: 0.0 euros
No se puede ingresar una cantidad negativa
Saldo Inicial: 0.0 euros
Saldo tras ingreso: 100.0 euros
Saldo tras ingreso: 300.0 euros
```

Verificaciones punto de ruptura 3

Valor de cantidad: Verifico que el valor que se está ingresando es positivo (cantidad=300).

This block provides a closer look at the variables and console output:

- Variables View:** The 'cantidad' variable is highlighted with a red box, showing its value as 300.0.
- Console:** The output shows the sequence of operations:


```
CCuenta [Java Application] C:\Program Files\Java\jdk-23\bin\javaw.exe
Saldo Inicial: 0.0 euros
No se puede ingresar una cantidad negativa
Saldo Inicial: 0.0 euros
Saldo tras ingreso: 100.0 euros
Saldo tras ingreso: 300.0 euros
```

Valor de dSaldo antes de la actualización: Antes de realizar la suma, inspecciono el valor de dSaldo. En este punto, ya se han realizado los dos primeros ingresos, por lo que el saldo debe reflejar los ingresos previos. Verifico si la suma es correcta ($100+200=300$).

The screenshot shows the IDE's Variables window with the following data:

Name	Value
no method return value	
this	CCuenta (id=23)
dSaldo	300.0
cantidad	300.0

The Console output shows the following messages:

```

CCuenta [Java Application] C:\Program Files\Java\jdk-23\bin\javaw.e
Saldo Inicial: 0.0 euros
No se puede ingresar una cantidad negativa
Saldo Inicial: 0.0 euros
Saldo tras ingreso: 100.0 euros
Saldo tras ingreso: 300.0 euros
  
```

Comportamiento del programa con el tercer ingreso: Verifico que el tercer ingreso se realiza correctamente sin errores, sumando la cantidad al saldo ($300+300=600$) y que el código de error es 0 (ya que la cantidad ingresada es un valor positivo).

The screenshot shows the IDE's Variables window with the following data:

Name	Value
no method return value	
this	CCuenta (id=23)
dSaldo	600.0
cantidad	300.0
iCodErr	0

The Console output shows the following messages:

```

CCuenta [Java Application] C:\Program Files\Java\jdk-23\bin\javaw.e
Saldo Inicial: 0.0 euros
No se puede ingresar una cantidad negativa
Saldo Inicial: 0.0 euros
Saldo tras ingreso: 100.0 euros
Saldo tras ingreso: 300.0 euros
  
```

FIN DEL PROGRAMA

Verifico que todos los movimientos que ha habido en la cuenta sean correctos.

```

<terminated> CCuenta [Java Application] C:\Program Files\Java\jdk-2
Saldo Inicial: 0.0 euros
No se puede ingresar una cantidad negativa
Saldo Inicial: 0.0 euros
Saldo tras ingreso: 100.0 euros
Saldo tras ingreso: 300.0 euros
Saldo tras ingreso: 600.0 euros
Saldo tras retirada: 550.0 euros
  
```