



# TAREA TEMA 3

## Gemma Morais Villar

---

Entornos de Desarrollo

## FORMATO

- Archivo zip con el nombre `endes_t3_tarea_Apellido1_Apellido2_Nombre`, con los contenidos:
  - Proyecto en java que incluya la clase CCuenta y las pruebas unitarias.
  - Documento .pdf con las respuestas a las tareas del presente documento.

## INTRODUCCIÓN

Observa la clase CCuenta que puedes descargar en el siguiente enlace:

[https://github.com/guillermoroman/endes\\_recursos/blob/main/endes\\_t3/CCuenta.java](https://github.com/guillermoroman/endes_recursos/blob/main/endes_t3/CCuenta.java)

Lee los enunciados de todas las subtareas antes de comenzar a realizar la tarea.

## PRIMER VISTAZO (1 PT)

Haz un primer barrido del código y soluciona solo aquellos errores que evitan la compilación del programa y que vienen señalados por el IDE al tratar de compilar.

## MÉTODO INGRESAR (2 PT)

### CLASES DE EQUIVALENCIA

Diseña las clases de equivalencia para el método `ingresar`. Rellena la siguiente tabla (borra o añade filas si es necesario):

Criterio	#	Clase válida	#	Clase no válida
CantidadIngreso	1	Cantidad>0	2.1 2.2	<0 =0

## CASOS DE PRUEBA

Diseña los casos de prueba necesarios para verificar el método **ingresar**. Rellena la siguiente tabla (borra o añade filas si es necesario):

Clases de equivalencia	Entrada	Salida
1	Cantidad=1	"El ingreso ha sido correcto"
2.1	Cantidad=-1	"No se puede ingresar una cantidad negativa"
2.2	Cantidad=0	"No se puede ingresar una cantidad negativa"

## MÉTODO RETIRAR (2PT)

### CLASES DE EQUIVALENCIA

Diseña las clases de equivalencia para el método **retirar**. Rellena la siguiente tabla (borra o añade filas si es necesario):

Criterio	#	Clase válida	#	Clase no válida
SaldoInsuficiente	1	Saldo>cantidad	2	Cantidad>saldo
SaldoSuficiente	3	Saldo=cantidad	4	Cantidad>saldo
CantidadNegativa	5	Cantidad>0	6	Cantidad<=0

## CASOS DE PRUEBA

Diseña los casos de prueba necesarios para verificar el método **retirar**. Rellena la siguiente tabla (borra o añade filas si es necesario):

Clases de equivalencia	Entrada	Salida
1,5	Saldo=2 Cantidad=1	"Dinero retirado satisfactoriamente"
3,5	Saldo=1 Cantidad=1	"Dinero retirado satisfactoriamente"
2,4	Saldo=1 Cantidad=2	"No hay suficiente saldo"
1,6	Saldo=1 Cantidad=-1	"No se puede retirar una cantidad negativa"
1,6	Saldo=1 Cantidad=0	"No se puede retirar una cantidad negativa"

## TESTS CON JUNIT (2 PT)

Crea una clase de tipo test (en el lugar adecuado en la jerarquía del proyecto) llamada `CCuentaTest`.

Deberás crear métodos para cada uno de los casos de prueba propuestos.

## DOCUMENTAR LOS ERRORES ENCONTRADOS (3PT)

### DESCRIPCIÓN PORMENORIZADA

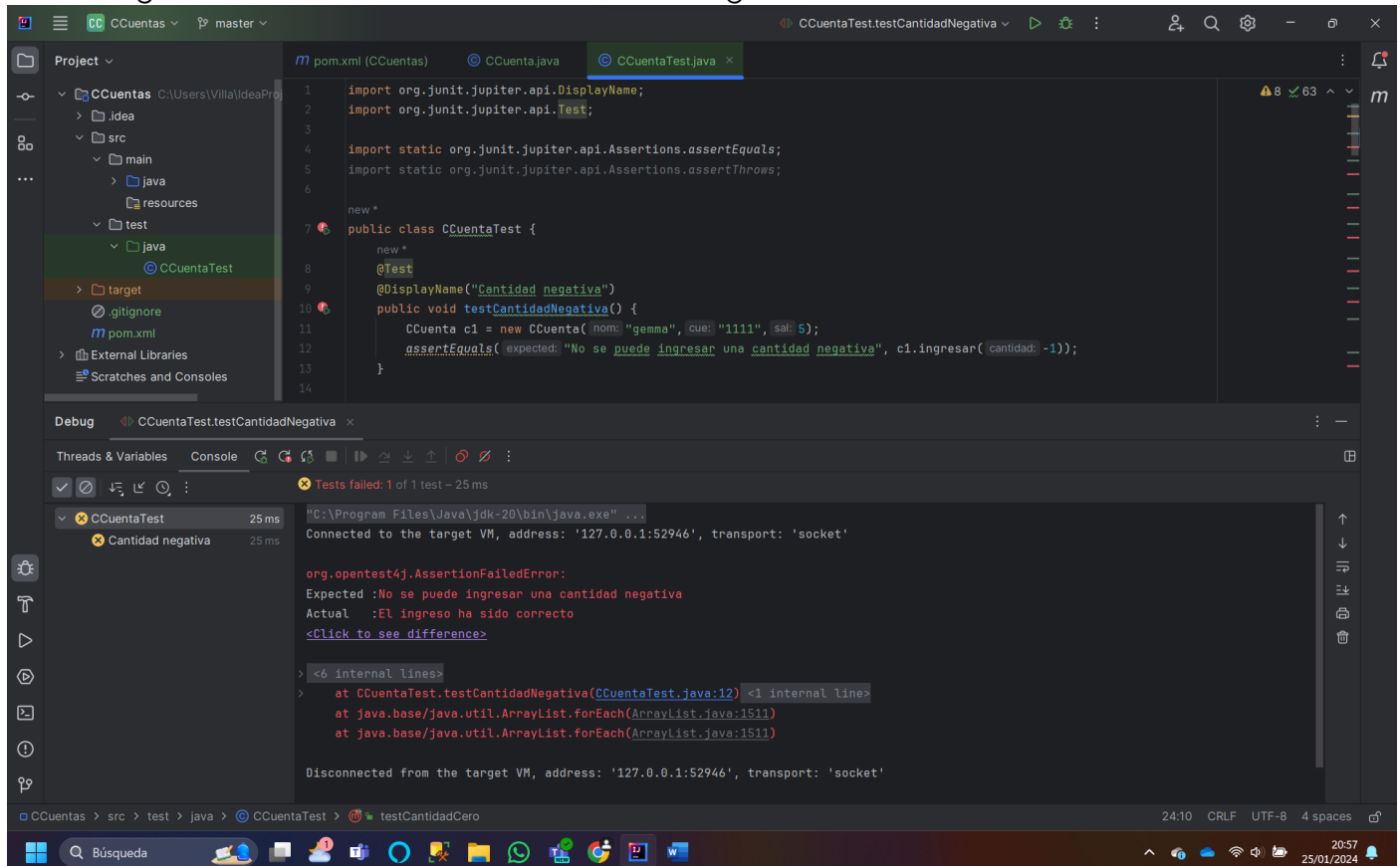
Se deben encontrar errores lógicos o bugs del sistema. Para **dos** de los errores encontrados, realiza una ejecución paso a paso y adjunta dos capturas de pantalla:

- Una captura de los valores que se ven en la inspección de variables justo antes de que aparezca un valor inesperado
- Una captura justo después de la ejecución de la sentencia que contiene el fallo y genera un valor inesperado.

Comenta por qué has colocado un punto de interrupción en el lugar elegido y comenta brevemente el error.

## 1ºERROR CANTIDAD NEGATIVA

Al ejecutar el test “Cantidad Negativa” con valor de entrada -1, se produce un error. Este test debería comprobar que el usuario no ingresara una cantidad negativa, y en caso de hacerlo devolviera el valor esperado “No se puede ingresar una cantidad negativa”. En su lugar recibimos el valor de salida “El ingreso ha sido correcto”.



```
1 import org.junit.jupiter.api.DisplayName;
2 import org.junit.jupiter.api.Test;
3
4 import static org.junit.jupiter.api.Assertions.assertEquals;
5 import static org.junit.jupiter.api.Assertions.assertThrows;
6
7 new *
8 public class CCuentaTest {
9     new *
10    @Test
11    @DisplayName("Cantidad negativa")
12    public void testCantidadNegativa() {
13        CCuenta c1 = new CCuenta("gemma", "cuel", "1111", "sal", 5);
14        assertEquals("expected: \"No se puede ingresar una cantidad negativa\", c1.ingresar( cantidad: -1));
15    }
16 }
```

Debug: CCuentaTest.testCantidadNegativa

Threads & Variables Console

Tests failed: 1 of 1 test - 25 ms

CCuentaTest 25 ms

Cantidad negativa 25 ms

org.opentest4j.AssertionFailedError:  
Expected :No se puede ingresar una cantidad negativa  
Actual :El ingreso ha sido correcto  
<Click to see difference>

<6 internal lines>

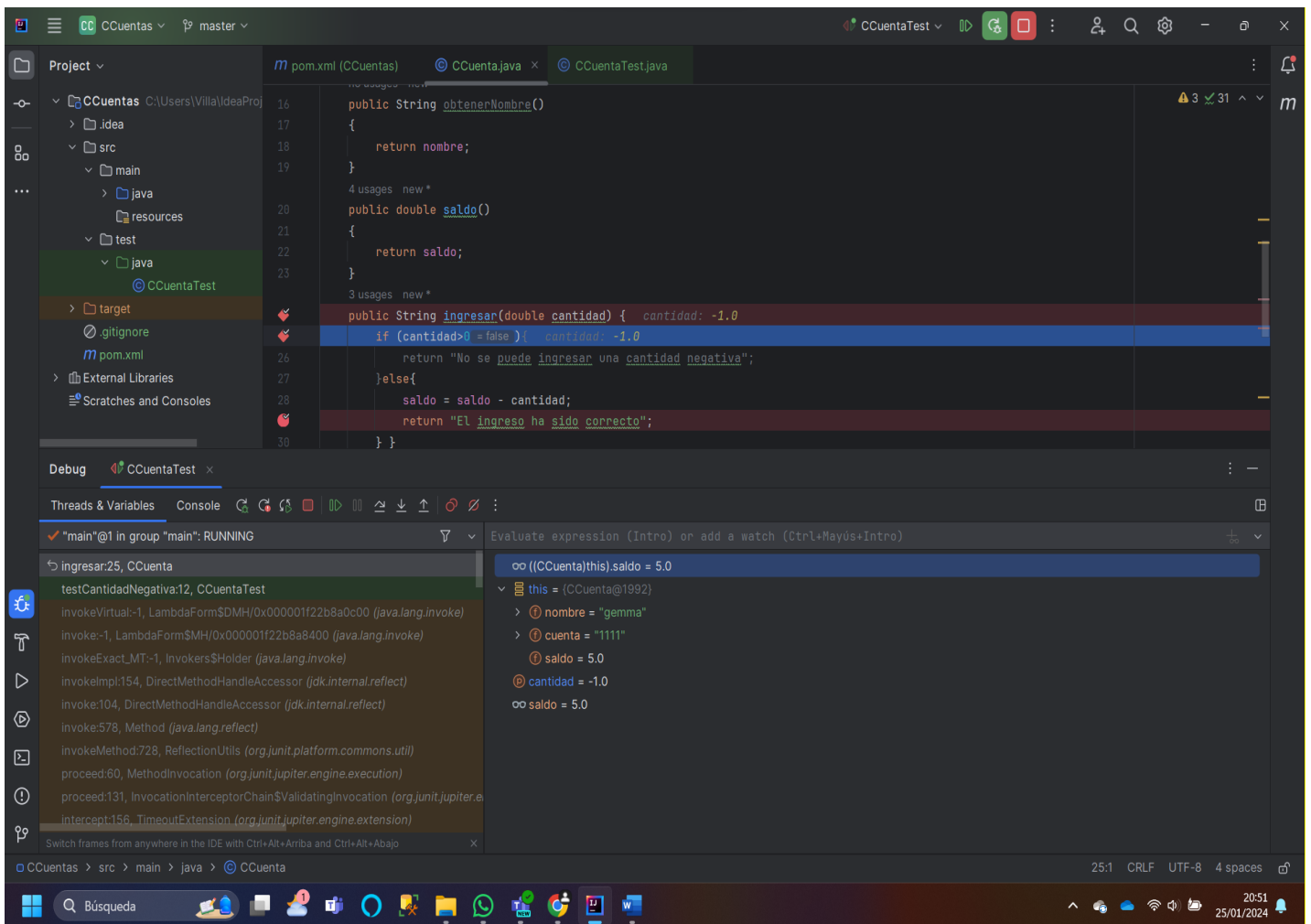
at CCuentaTest.testCantidadNegativa(CCuentaTest.java:12) <1 internal line>  
at java.base/java.util.ArrayList.forEach(ArrayList.java:1511)  
at java.base/java.util.ArrayList.forEach(ArrayList.java:1511)

Disconnected from the target VM, address: '127.0.0.1:52946', transport: 'socket'

Como este test se basa en el método “Ingresar” de la clase CCuenta, puse mis puntos de interrupción en este método para buscar la ejecución de la sentencia que contiene el fallo.

Cuando ejecutamos la sentencia paso a paso, el IDE nos indica con unos mensajes en gris claro que la condición `if(cantidad>0)` es false para la cantidad que introducimos de -1.

Se trata de un error lógico, ya que si el valor esperado es “No se puede ingresar una cantidad negativa” la condición debería ser `if(cantidad<0)` para que se ejecute cada vez que introduzcas un numero negativo (<0).



The screenshot displays the IntelliJ IDEA IDE interface. The top toolbar shows the 'Run' button (a green play icon). The left sidebar contains the 'Project' view, showing the project structure with folders like 'src', 'main', 'resources', 'test', and 'target'. The main editor window shows the 'CCuentaTest.java' file, which contains the following code:

```
16 public String obtenerNombre()
17 {
18     return nombre;
19 }
20 public double saldo()
21 {
22     return saldo;
23 }
24 public String ingresar(double cantidad) { cantidad: -1.0
25     if (cantidad > 0 = false) { cantidad: -1.0
26         return "No se puede ingresar una cantidad negativa";
27     } else {
28         saldo = saldo - cantidad;
29         return "El ingreso ha sido correcto";
30     } }
```

The bottom panel shows the 'Debug' console. The 'Threads & Variables' tab is active, displaying the test execution details. The test 'testCantidadNegativa:12, CCuentaTest' is running successfully. The console output shows the following steps:

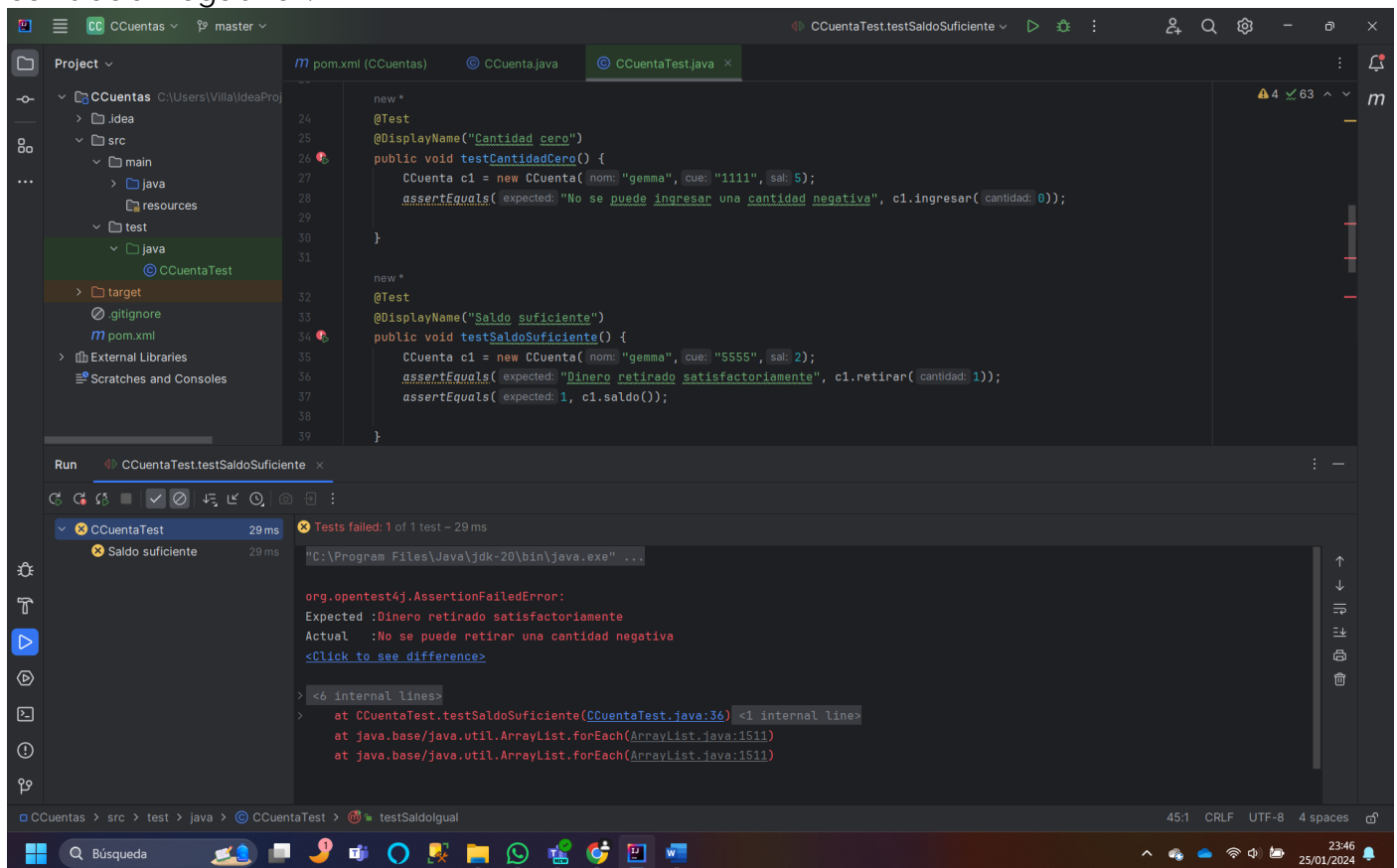
- invokeVirtual:-1, LambdaForm\$DMH/0x000001f22b8a0c00 (java.lang.invoke)
- invoke:-1, LambdaForm\$MH/0x000001f22b8a8400 (java.lang.invoke)
- invokeExact\_MT:-1, Invokers\$Holder (java.lang.invoke)
- invokeImpl:154, DirectMethodHandleAccessor (jdk.internal.reflect)
- invoke:104, DirectMethodHandleAccessor (jdk.internal.reflect)
- invoke:578, Method (java.lang.reflect)
- invokeMethod:728, ReflectionUtils (org.junit.platform.commons.util)
- proceed:60, MethodInvocation (org.junit.jupiter.engine.execution)
- proceed:131, InvocationInterceptorChain\$ValidatingInvocation (org.junit.jupiter.engine.intercept)
- intercept:156, TimeoutExtension (org.junit.jupiter.engine.extension)

The variables window shows the state of the test:

- this = (CCuenta@1992)
- nombre = "gemma"
- cuenta = "1111"
- saldo = 5.0
- cantidad = -1.0
- saldo = 5.0

## 2º ERROR SALDO SUFICIENTE

Al analizar este test encontramos varios errores. Empieza con unos valores iniciales de saldo=2 y con la cantidad=1 para retirar. El valor esperado es “Dinero retirado satisfactoriamente” sin embargo nos devuelve un valor actual “No se puede retirar una cantidad negativa”.



```
new *
@Test
@DisplayName("Cantidad cero")
public void testCantidadCero() {
    CCuenta c1 = new CCuenta( nom: "gemma", cue: "1111", sal: 5);
    assertEquals( expected: "No se puede ingresar una cantidad negativa", c1.ingresar( cantidad: 0));
}

new *
@Test
@DisplayName("Saldo suficiente")
public void testSaldoSuficiente() {
    CCuenta c1 = new CCuenta( nom: "gemma", cue: "5555", sal: 2);
    assertEquals( expected: "Dinero retirado satisfactoriamente", c1.retirar( cantidad: 1));
    assertEquals( expected: 1, c1.saldo());
}
```

Run CCuentaTest.testSaldoSuficiente

Tests failed: 1 of 1 test - 29 ms

org.opentest4j.AssertionFailedError:  
Expected :Dinero retirado satisfactoriamente  
Actual :No se puede retirar una cantidad negativa  
<Click to see difference>

<6 internal lines>

at CCuentaTest.testSaldoSuficiente(CCuentaTest.java:36) <1 internal line>  
at java.base/java.util.ArrayList.forEach(ArrayList.java:1511)  
at java.base/java.util.ArrayList.forEach(ArrayList.java:1511)

De nuevo coloco el punto de interrupción que corresponde al método que está analizando este test, método retirar, para ir comprobando las líneas de código.

```

25     if (cantidad > 0) {
26         return "No se puede ingresar una cantidad negativa";
27     } else {
28         saldo = saldo - cantidad;
29         return "El ingreso ha sido correcto";
30     } }
31
32     public String retirar (double cantidad) {
33         if (cantidad != 0 = true) {
34             return "No se puede retirar una cantidad negativa";
35         } else if (saldo() > cantidad) {
36             return "No hay suficiente saldo";
37         } else {
38             saldo = cantidad;
39             return "Dinero retirado satisfactoriamente";
40         } }
41
42     public String obtenerCuenta () { return cuenta;
43 }

```

Debug Console:

```

testSaldoSuficiente:36, CCuentaTest
invokeVirtual:-1, LambdaForm$DMH/0x00000228810ac000 (java.lang.invoke)
invoke:-1, LambdaForm$MH/0x00000228810ac800 (java.lang.invoke)
invokeExact_MT:-1, Invokers$Holder (java.lang.invoke)
invokeImpl:154, DirectMethodHandleAccessor (jdk.internal.reflect)
invoke:104, DirectMethodHandleAccessor (jdk.internal.reflect)
invoke:578, Method (java.lang.reflect)
invokeMethod:728, ReflectionUtils (org.junit.platform.commons.util)
proceed:60, MethodInvocation (org.junit.jupiter.engine.execution)
proceed:131, InvocationInterceptorChain$ValidatingInvocation (org.junit.jupiter.a
intercept:156, TimeoutExtension (org.junit.jupiter.engine.extension)

```

De primeras el ID nos indica que la condición del operador se cumple(true) ya que la cantidad 1 es diferente de 0 pero nos devuelve "No se puede retirar una cantidad negativa".

Cambiamos el operador  $\neq$  por  $\leq 0$  para que la condición se corresponda con el return.

Con el else if nos vuelve dar un error parecido al último, nos devuelve "No hay suficiente saldo". En este caso, para que haya saldo insuficiente la cantidad tendría que ser mayor que el saldo. Modificamos:

Else if(cantidad>saldo())

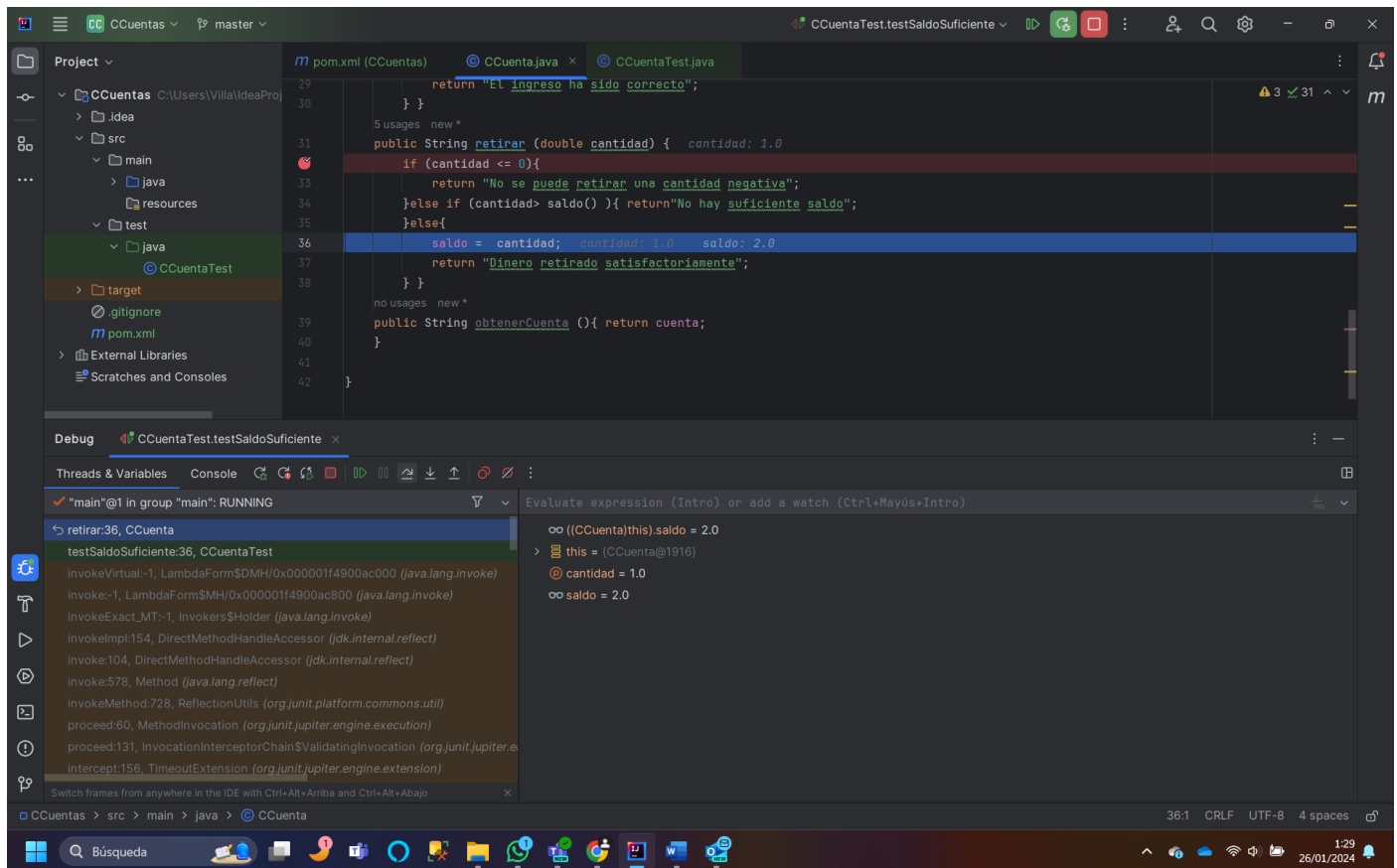
```

org.opentest4j.AssertionFailedError:
Expected :Dinero retirado satisfactoriamente
Actual   :No hay suficiente saldo
<Click to see difference>

```



Este test ya no daría fallos, sin embargo, la siguiente parte del código sí que daría fallo en otros test y también habría que modificarla.



```
29         return "El ingreso ha sido correcto";
30     } }
31     5 usages new *
32     public String retirar (double cantidad) { cantidad: 1.0
33         if (cantidad <= 0){
34             return "No se puede retirar una cantidad negativa";
35         }else if (cantidad> saldo() ){ return "No hay suficiente saldo";
36         }else{
37             saldo = cantidad; cantidad: 1.0 saldo: 2.0
38             return "Dinero retirado satisfactoriamente";
39         } }
40     no usages new *
41     public String obtenerCuenta () { return cuenta;
42     }
43 }
```

Debug Console:

Threads & Variables Console Evaluate expression (Intro) or add a watch (Ctrl+Mayús+Intro)

✓ "main" @1 in group "main": RUNNING

retirar:36, CCuenta

testSaldoSuficiente:36, CCuentaTest

invokeVirtual:-1, LambdaForm\$DMH/0x000001f4900ac000 (java.lang.invoke)

invoke:-1, LambdaForm\$MH/0x000001f4900ac800 (java.lang.invoke)

invokeExact\_MT:-1, Invokers\$Holder (java.lang.invoke)

invokeImpl:154, DirectMethodHandleAccessor (jdk.internal.reflect)

invoke:104, DirectMethodHandleAccessor (jdk.internal.reflect)

invoke:578, Method (java.lang.reflect)

invokeMethod:728, ReflectionUtils (org.junit.platform.commons.util)

proceed:60, MethodInvocation (org.junit.jupiter.engine.execution)

proceed:131, InvocationInterceptorChain\$ValidatingInvocation (org.junit.jupiter.engine.interception)

intercept:156, TimeoutExtension (org.junit.jupiter.engine.extension)

Switch frames from anywhere in the IDE with Ctrl+Alt+Arriba and Ctrl+Alt+Abajo

Watch:

- ((CCuenta)this).saldo = 2.0
- this = (CCuenta@1916)
- cantidad = 1.0
- saldo = 2.0

Saldo no es igual cantidad. Si lo que queremos obtener es el saldo tras la cantidad retirada tendría que ser saldo= saldo- cantidad.

## DOCUMENTACIÓN DE LOS ERRORES ENCONTRADOS

Debemos encontrar el resto de los errores del código.

Para cuatro errores, incluidos los dos elegidos en el apartado anterior, se deberá registrar:

- Test que encontró el fallo.
- Método donde se encuentra el error.
- Valores de entrada o configuración.
- Valor esperado.
- Valor obtenido.
- Fragmento de código problemático antes de solucionar el problema.
- Código tras solucionar el problema.

Test	Método	Valores entrada	Valores esperados	Valores salida	Código problema	Código arreglado
Cantidad Negativa	Test Cantidad Negativa Método Ingresar	Cantidad=-1	"No se puede ingresar una cantidad negativa"	"El ingreso ha sido correcto"	If (cantidad>0) {return "No se puede ingresar una cantidad negativa"}	If (cantidad<0) {return "No se puede ingresar una cantidad negativa"}
Cantidad Positiva	Test Cantidad Negativa Método Ingresar	Cantidad=1	"El ingreso ha sido correcto"	"No se puede ingresar una cantidad negativa"	If (cantidad>0) {return "No se puede ingresar una cantidad negativa"}	If (cantidad<0) {return "No se puede ingresar una cantidad negativa"}
Cantidad Cero	Test Cantidad Cero Método Ingresar	Cantidad=0	"No se puede ingresar una cantidad negativa"	"El ingreso ha sido correcto"	If (cantidad>0) {return "No se puede ingresar una cantidad negativa"}	If (cantidad<=0) {return "No se puede ingresar una cantidad negativa"}
Saldo Suficiente	Test Saldo Suficiente Método Retirar	Saldo=2 Cantidad=1	"Dinero retirado satisfactoriamente"	"No se puede retirar una cantidad negativa"	else {saldo=cantidad; return "Dinero retirado satisfactoriamente"}	else { saldo= saldo -cantidad; return "Dinero retirado satisfactoriamente" }
Saldo Igual Cantidad	Test Saldo Igual Método Retirar	Saldo=1 Cantidad=1	"Dinero retirado satisfactoriamente"	"No se puede retirar una cantidad negativa"	Else {saldo=cantidad; return "Dinero retirado satisfactoriamente"}	else { saldo= saldo -cantidad; return "Dinero retirado satisfactoriamente" }
Saldo Insuficiente	Test Saldo Insuficiente Método Retirar	Saldo=1 Cantidad=2	"No hay suficiente saldo"	"No se puede retirar una cantidad negativa"	} else if (saldo() > cantidad) { return "No hay suficiente saldo";}	} else if (cantidad>saldo() ) {return "No hay suficiente saldo"; }
Retirar Cantidad Negativa	test Retirar Cantidad Negativa Método Retirar	Cantidad=-1	"No se puede retirar una cantidad negativa"	"Dinero retirado satisfactoriamente"	If (cantidad!=0) {return "No se puede ingresar una cantidad negativa"}	If (cantidad<0) {return "No se puede ingresar una cantidad negativa"}
Retirar Cantidad Cero	test Retirar Cantidad Cero Método Retirar	Cantidad=0	"No se puede retirar una cantidad negativa"	"Dinero retirado satisfactoriamente"	If (cantidad!=0) {return "No se puede ingresar una cantidad negativa"}	If (cantidad<=0) {return "No se puede ingresar una cantidad negativa"}