

TAREA PARA ED03.

Detalles de la tarea de esta unidad.

Enunciado.

En esta tarea se considera una clase **Java** CCuenta que dispone de los métodos main, ingresar y retirar. Este es el código de los métodos main, ingresar y retir que deberás tener en cuenta para resolver la tarea:

Método main

```
public static void main(String[] args) {
    // Depuracion. Se detiene siempre
    CCuenta miCuenta = new CCuenta();
    System.out.println("Saldo Inicial: " + miCuenta.dSaldo + " euros");
    // Depuracion. Provoca parada por ingreso con cantidad menor de 0
    miCuenta.ingresar(-100);
    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");
    // Depuracion. Provoca parada con codicion de tercer ingreso
    miCuenta.ingresar(300);
    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
    {
        // Depuracion. Punto de parada. Solo en el 3 ingreso
        dSaldo = dSaldo + cantidad;
        iCodErr = 0;
    }
}
```

```
// Depuracion. Punto de parada cuando la cantidad es menor de 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 se hay suficiente saldo");
    }
    else
    {
    }
}
}
```

Deberás realizar un documento donde dar respuesta a los siguientes apartados:

1. Realiza un **análisis de caja blanca** completo del método ingresar.

Las pruebas de **caja blanca** son un tipo de pruebas de software que se centran en el código fuente de un programa. El objetivo de estas pruebas es asegurar que el código funciona correctamente y cumple con los requisitos especificados.

El método ingresar tiene un parámetro **cantidad** tipo **double** con tres casos posibles:

- El caso donde **cantidad** es menor que 0, donde nos imprimirá por pantalla "No se puede ingresar una cantidad negativa" y asigna el valor de la variable **iCodErr** de tipo **int** al valor -1.
- El caso donde **cantidad** sea igual a -3, donde nos imprimirá por pantalla "Error detectable en pruebas de caja blanca" y asigna el valor de la variable **iCodErr** a 2. Tal y como indica el error, este error solo podría detectada en pruebas de caja blanca, es decir, viendo el código fuente.
- Si no se cumple ninguno de los anteriores, no imprimirá ningún mensaje y asigna el valor de la variable **iCodErr** a 0.

El segundo caso no llegaría a ejecutarse nunca ya que -3 es < 0 y se ejecuta antes el primer caso. Por eso el segundo caso es indetectable sin ver el código.

2. Realiza un **análisis de caja negra**, incluyendo valores límite y conjetura de errores del método retirar. Debes considerar que este método recibe como parámetro la cantidad a retirar, que no podrá ser menor a 0. Además en ningún caso esta cantidad podrá ser mayor al saldo actual. Al tratarse de

pruebas funcionales no es necesario conocer los detalles del código pero te lo pasamos para que lo tengas.

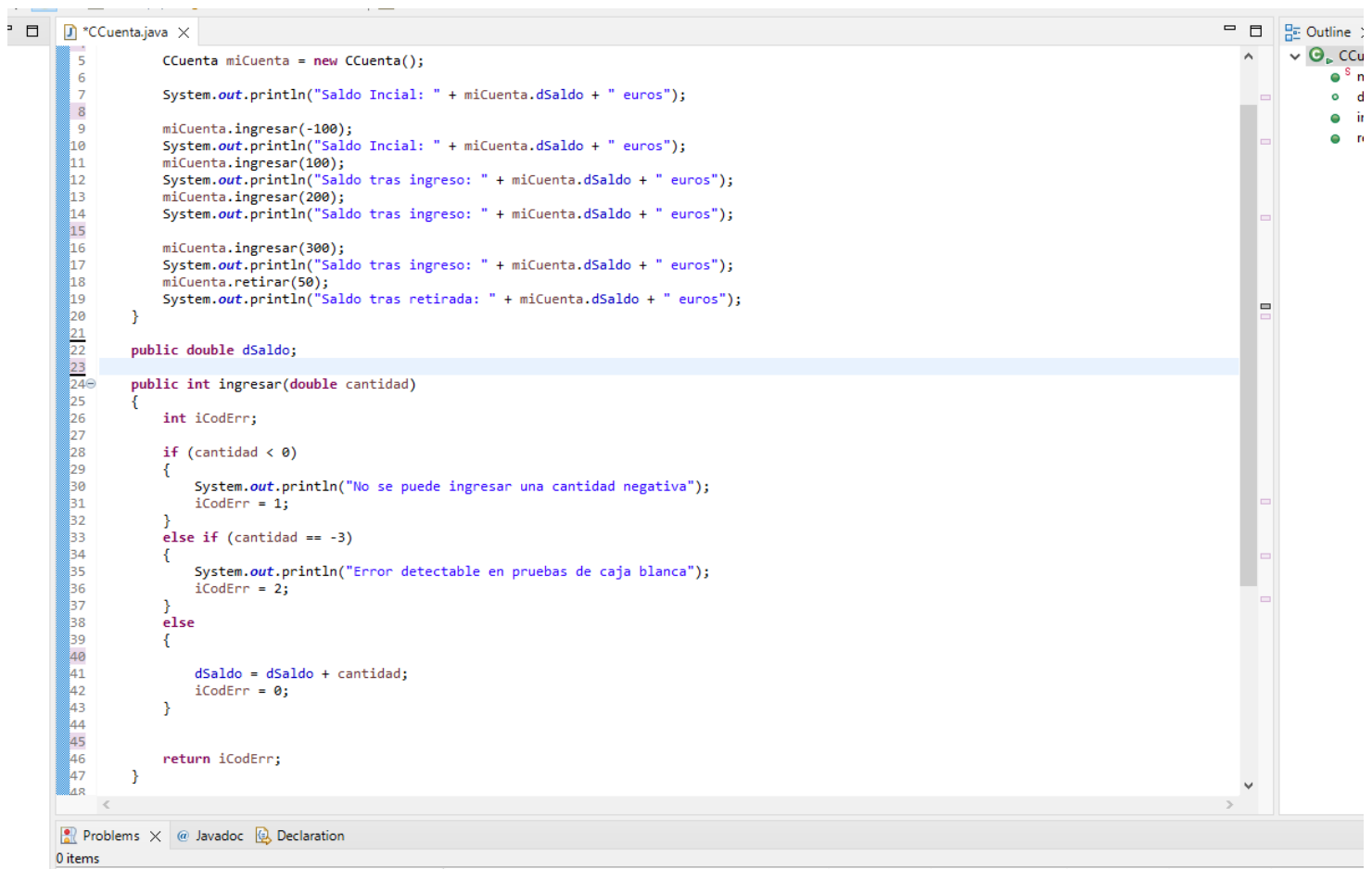
El análisis de **caja negra** es un enfoque de pruebas de software que se centra en el comportamiento externo, es decir, sin ver el código.

- Numero positivo mayor que el saldo actual, no debería dejarnos retirar la cantidad indicada.
- Numero positivo menor que el saldo actual, debería dejarnos retirar la cantidad indicada.
- Numero positivo, siendo igual al saldo actual, debería dejarnos retirar la cantidad indicada.
- Valor 0, debería dejarnos retirar la cantidad indicada, al ser 0 no retiraría nada o indicarlo como un valor nulo y anular la extracción.
- Numero negativo, nos debería dar error ya que no podemos retirar cantidades negativas.

En cuanto a pruebas de seguridad, deberíamos tener en cuenta también un análisis de caja negra. Por ejemplo, prevenir el ataque de inyecciones SQL mediante el uso de consultas parametrizadas, uso de procedimientos... Todo esto suponiendo que el método **retirar** estuviera desplegado en una web, como una aplicación bancaria con una base de datos detrás.

3. Crea la clase CCuentaTest del tipo **Caso de prueba JUnit** en **Eclipse** que nos permita pasar las pruebas unitarias de caja blanca del método ingresar. Los casos de prueba ya los habrás obtenido en el primer apartado del ejercicio. Copia el código fuente de esta clase en el documento.

El código quedaría así.



```
1 CCuenta miCuenta = new CCuenta();
2
3 System.out.println("Saldo Inicial: " + miCuenta.dSaldo + " euros");
4
5 miCuenta.ingresar(-100);
6 System.out.println("Saldo Inicial: " + miCuenta.dSaldo + " euros");
7 miCuenta.ingresar(100);
8 System.out.println("Saldo tras ingreso: " + miCuenta.dSaldo + " euros");
9 miCuenta.ingresar(200);
10 System.out.println("Saldo tras ingreso: " + miCuenta.dSaldo + " euros");
11
12 miCuenta.ingresar(300);
13 System.out.println("Saldo tras ingreso: " + miCuenta.dSaldo + " euros");
14 miCuenta.retirar(50);
15 System.out.println("Saldo tras retirada: " + miCuenta.dSaldo + " euros");
16
17 }
18
19 public double dSaldo;
20
21 public int ingresar(double cantidad)
22 {
23     int iCodErr;
24
25     if (cantidad < 0)
26     {
27         System.out.println("No se puede ingresar una cantidad negativa");
28         iCodErr = 1;
29     }
30     else if (cantidad == -3)
31     {
32         System.out.println("Error detectable en pruebas de caja blanca");
33         iCodErr = 2;
34     }
35     else
36     {
37         dSaldo = dSaldo + cantidad;
38         iCodErr = 0;
39     }
40
41     return iCodErr;
42 }
```

```

45     return iCodErr;
46 }
47
48
49
50 public void retirar (double cantidad)
51 {
52     if (cantidad <= 0)
53     {
54         System.out.println("No se puede retirar una cantidad negativa");
55     }
56     else if (dSaldo < cantidad)
57     {
58         System.out.println("No se hay suficiente saldo");
59     }
60     else
61     {
62         dSaldo = dSaldo - cantidad;
63     }
64 }
65 }

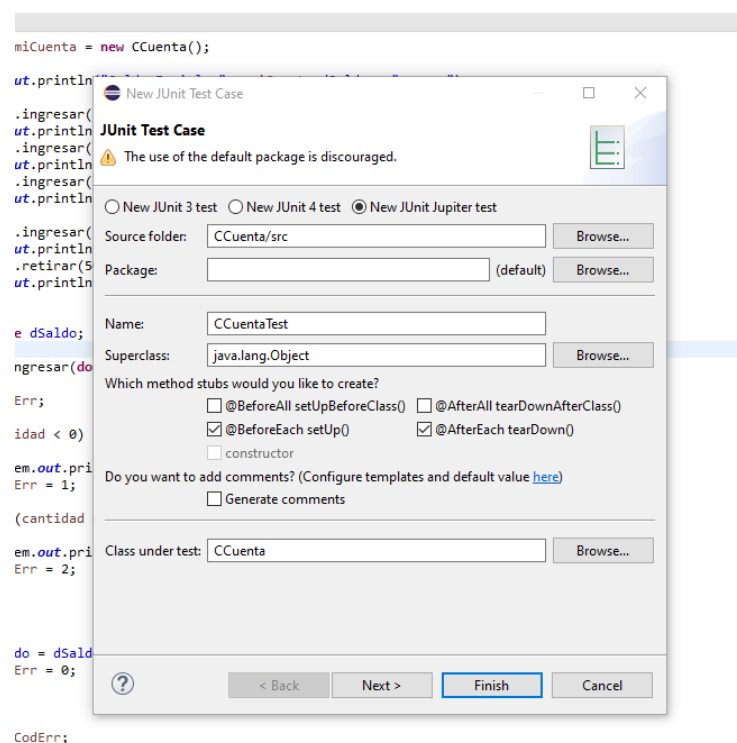
```

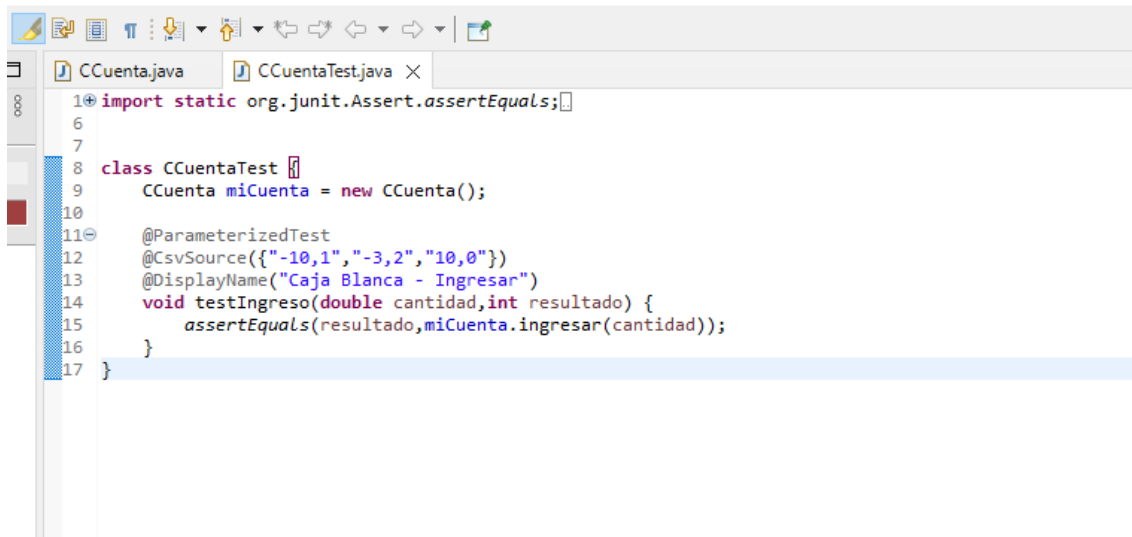
Problems | @ Javadoc | Declaration

0 items

Description	Resource	Path	Location	Type
entornosdeDesarrollo - Word				
entornosdeDesarrollo - ...				

Añado una nueva clase llamada CCuentaTest de tipo Junit Test Case





```
1 import static org.junit.Assert.assertEquals;
2
3
4
5
6
7
8 class CCuentaTest {
9     CCuenta miCuenta = new CCuenta();
10
11     @ParameterizedTest
12     @CsvSource({"-10,1","-3,2","10,0"})
13     @DisplayName("Caja Blanca - Ingresar")
14     void testIngreso(double cantidad,int resultado) {
15         assertEquals(resultado,miCuenta.ingresar(cantidad));
16     }
17 }
```

El método de prueba está anotado con **@ParameterizedTest**, lo que significa que se ejecutará varias veces con diferentes conjuntos de datos de entrada. Los conjuntos de datos se proporcionan mediante la anotación **@CsvSource**, que proporciona tres conjuntos de datos en forma de pares de valores (cantidad, resultado).

```
@CsvSource({"-10,1","-3,2","10,0"})
```

Representa el resultado esperado mientras que

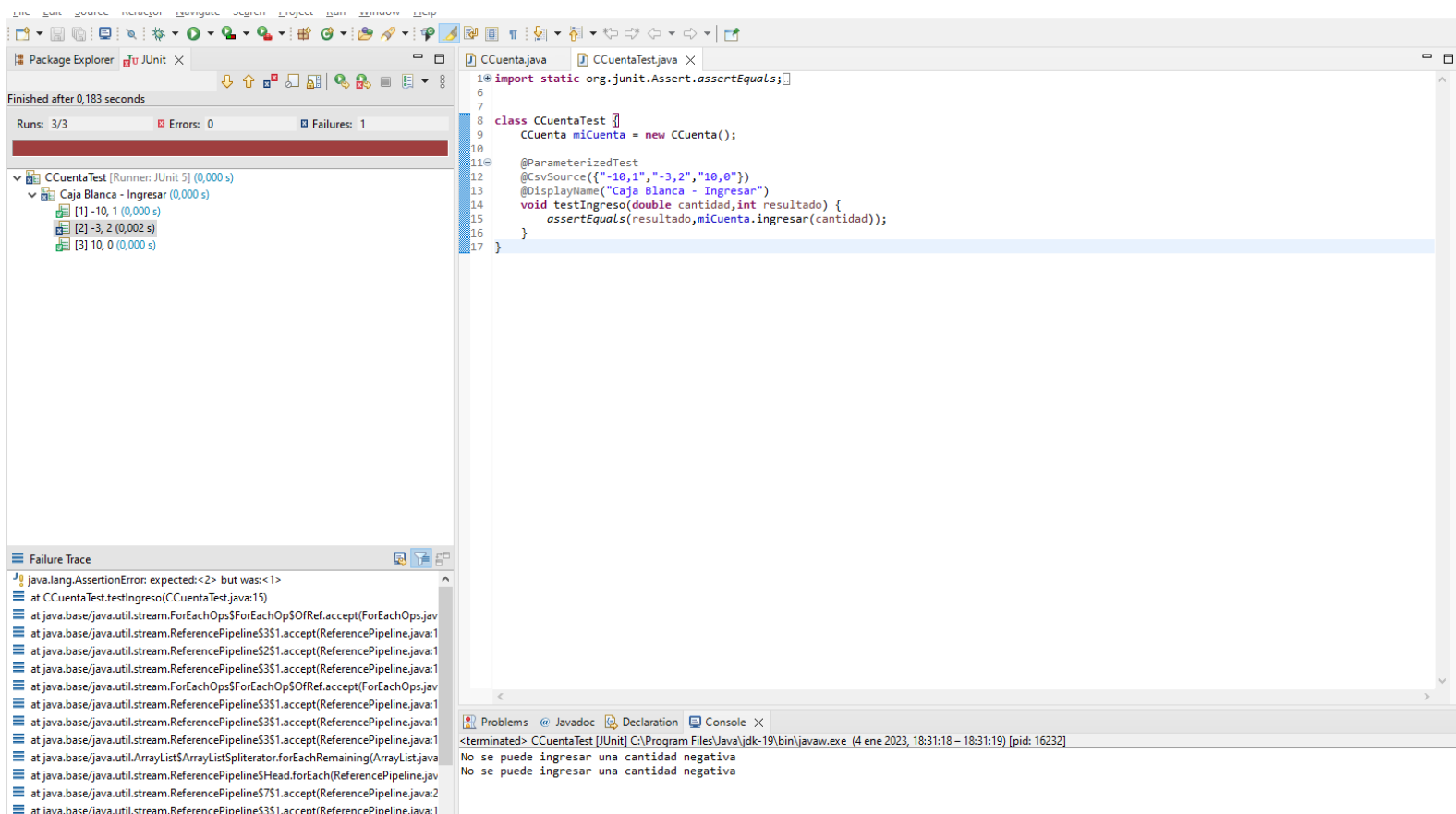
```
miCuenta.ingresar(cantidad)
```

representa el resultado obtenido.

El método de prueba llama al método **"ingresar"** de **"miCuenta"** y luego utiliza el método **"assertEquals"** para comparar el resultado devuelto por el método **"ingresar"** con el resultado esperado proporcionado en el conjunto de datos. Si el resultado esperado y el resultado devuelto son iguales, la prueba se considera exitosa. Si no lo son, se generará un error de prueba. Y adaptamos el código al test.

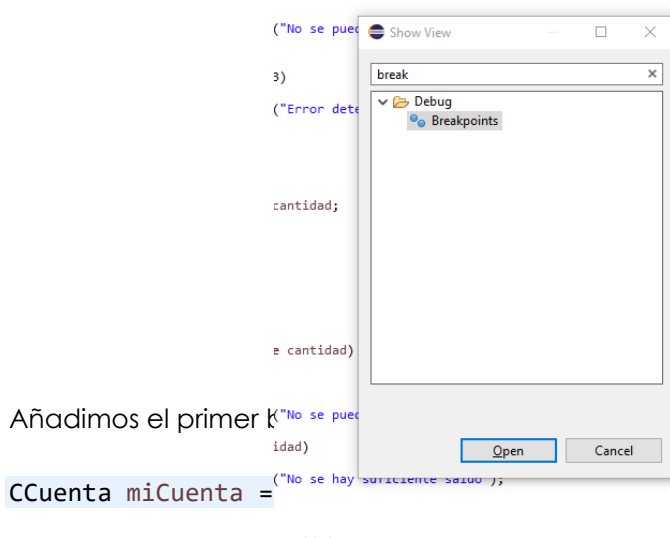
He tenido problemas al ejecutar Junit test, residía en que había añadido la librería Junit a modulepath en lugar de a classpath.

Tras ejecutarlo correctamente obtenemos lo siguiente:



4. Genera los siguientes **puntos de ruptura** para validar el comportamiento del método ingresar en modo depuración.
 - o Punto de parada sin condición al crear el objeto miCuenta en la función main. Línea 3 del código del método main que se presenta en la siguiente página de este libro.

Para poder ver el breakpoint en consola iremos a show view y filtraremos por breakpoint.

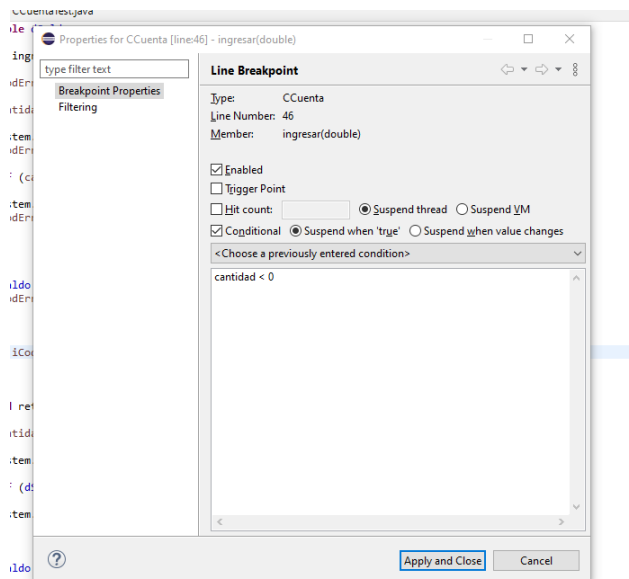


- Punto de parada en la instrucción return del método ingresar sólo si la cantidad a ingresar es menor de 0. Línea 20 del código del método ingresar que se presenta más adelante.

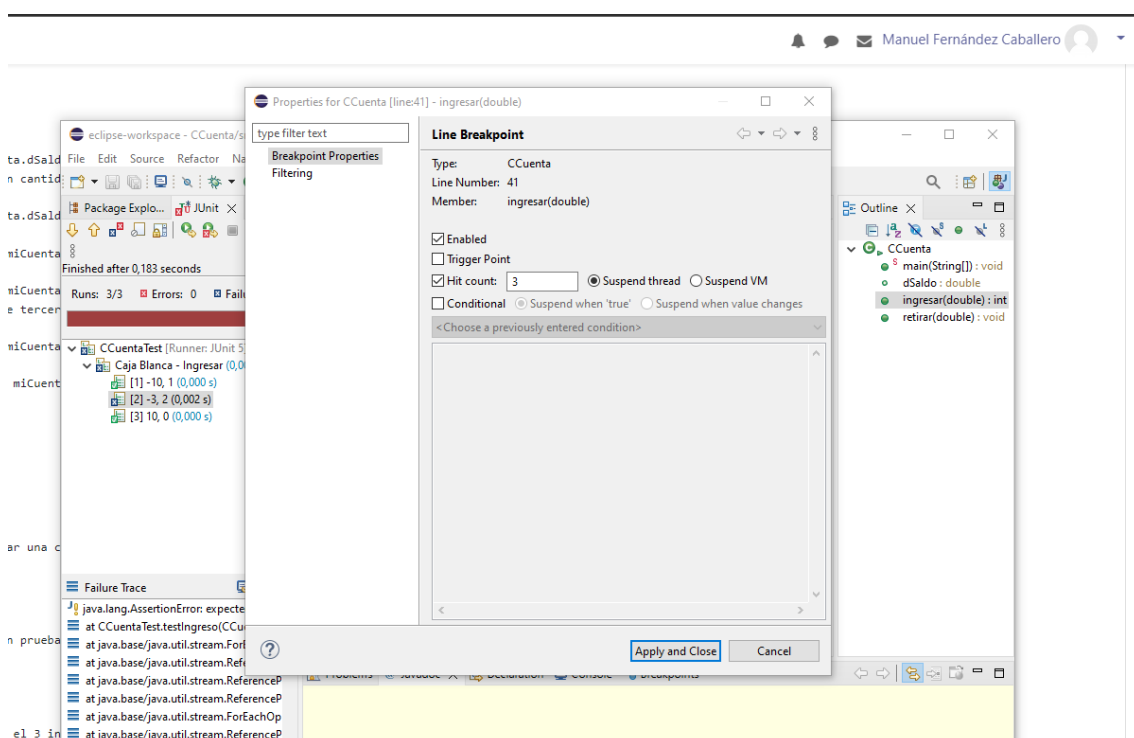
Añadimos el segundo breakpoint con condición en

```
return iCodErr;
```

Condición añadida en click derecho sobre breakpoint, breakpoint properties.

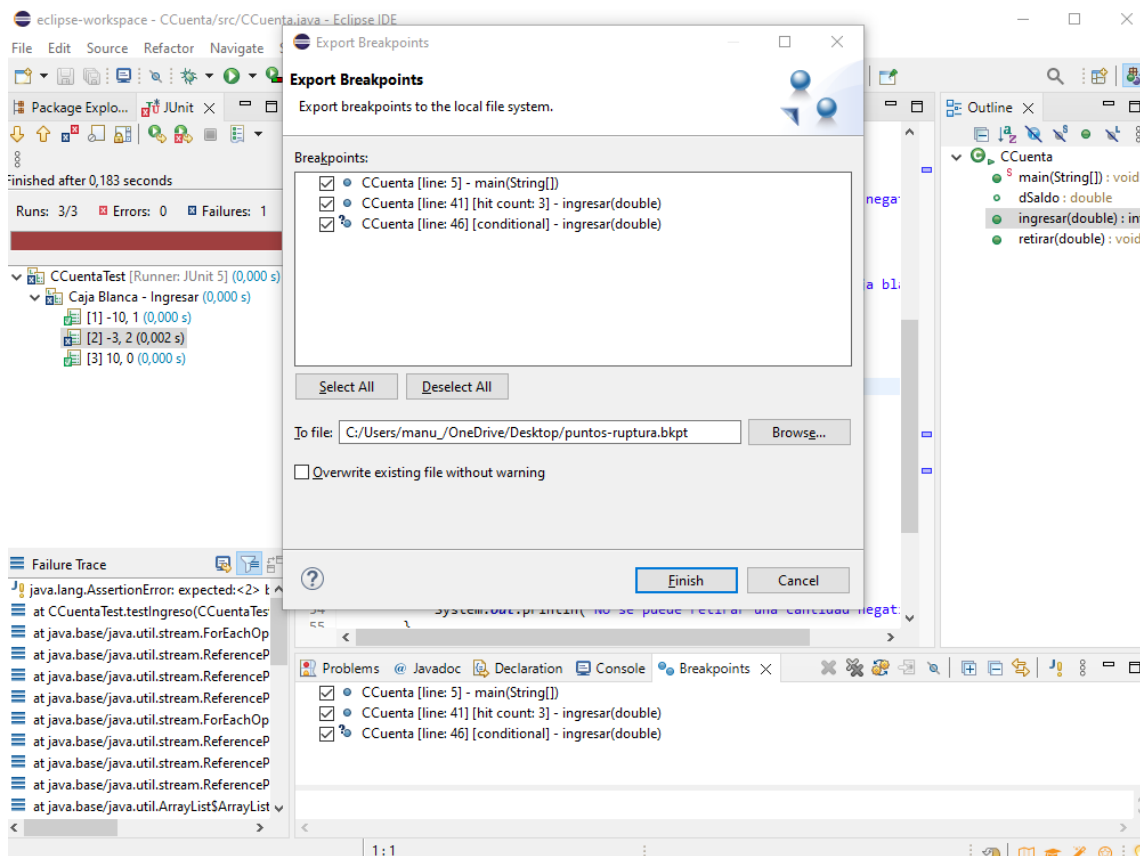


- Punto de parada en la instrucción donde se actualiza el saldo, sólo deberá parar la tercera vez que sea actualizado. Línea 16 del código del método ingresar que se presenta más adelante.



Como dice que solo deberá parar la tercera vez que sea actualizado, utilizamos Hit count y seteamos el contador a 3.

Exportamos los breakpoints creados



Si lo abrimos con un editor de texto, vemos el siguiente XML:

```
<?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="80" lineNumber="5"
type="org.eclipse.jdt.debug.javaLineBreakpointMarker">
```

```
<attrib name="charStart" value="80"/>
```

```
<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="118"/>
```

```
<attrib name="org.eclipse.debug.core.enabled" value="true"/>
```



```
<attrib name="message" value="Line breakpoint:CCuenta [line: 5] - main(String[])" />
<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="1216" lineNumber="41"
type="org.eclipse.jdt.debug.javaLineBreakpointMarker">
<attrib name="charStart" value="1216" />
<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="1252" />
<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: 41] [hit count: 3] -
ingresar(double)" />
<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="1309" lineNumber="46"
type="org.eclipse.jdt.debug.javaLineBreakpointMarker">
```

```

<attrib name="org.eclipse.jdt.debug.core.conditionEnabled" value="true"/>

<attrib name="charStart" value="1309"/>

<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="1332"/>

<attrib name="org.eclipse.debug.core.enabled" value="true"/>

<attrib name="message" value="Line breakpoint:CCuenta [line: 46] [conditional] -
ingresar(double)"/>

<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"/>

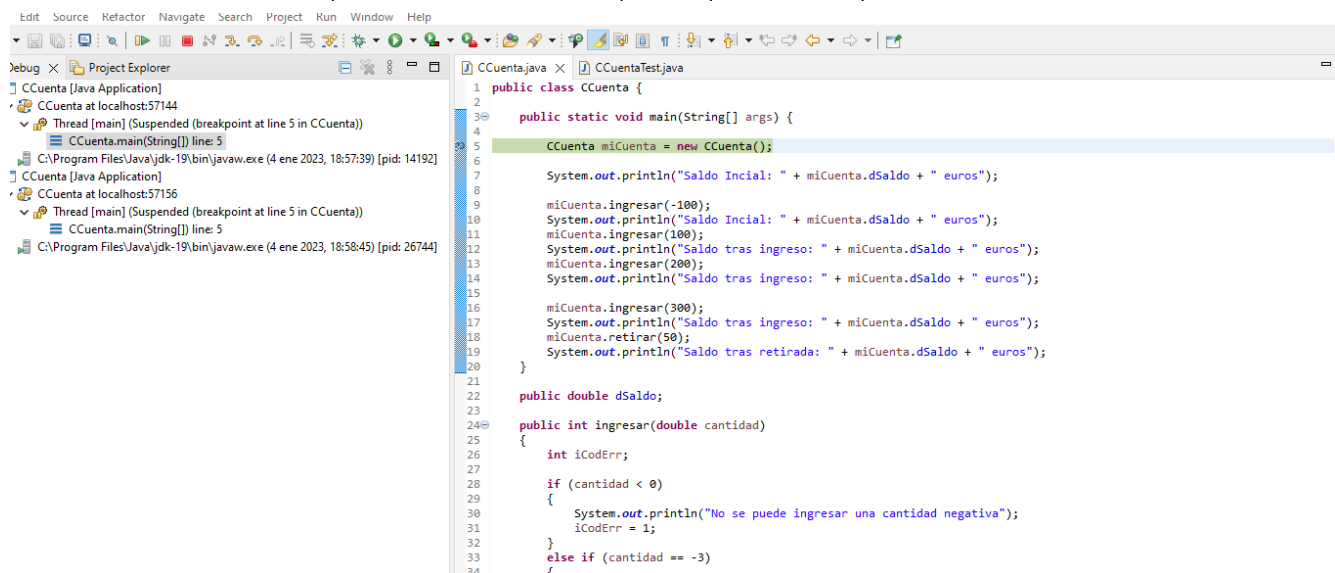
</marker>

</breakpoint>

</breakpoints>

```

Para depurar el programa haríamos click derecho, debug as Java application y observaríamos que se detendría en el primer punto de ruptura:



Pudiendo continuar con las siguientes instrucciones con la tecla F6 (Step Over).