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

El análisis de caja blanca consiste en analizar directamente el código de la aplicación, en nuestro ejemplo, el código analizar es el método ingresar que se expone a continuación:

```
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;

}
```

Como podemos ver, este método contiene un parámetro “cantidad” que es de tipo double. Dentro del método se incluye la variable “iCodErr” de tipo entero, que tomará distintos valores en función de qué caso se cumpla de entre los siguientes:

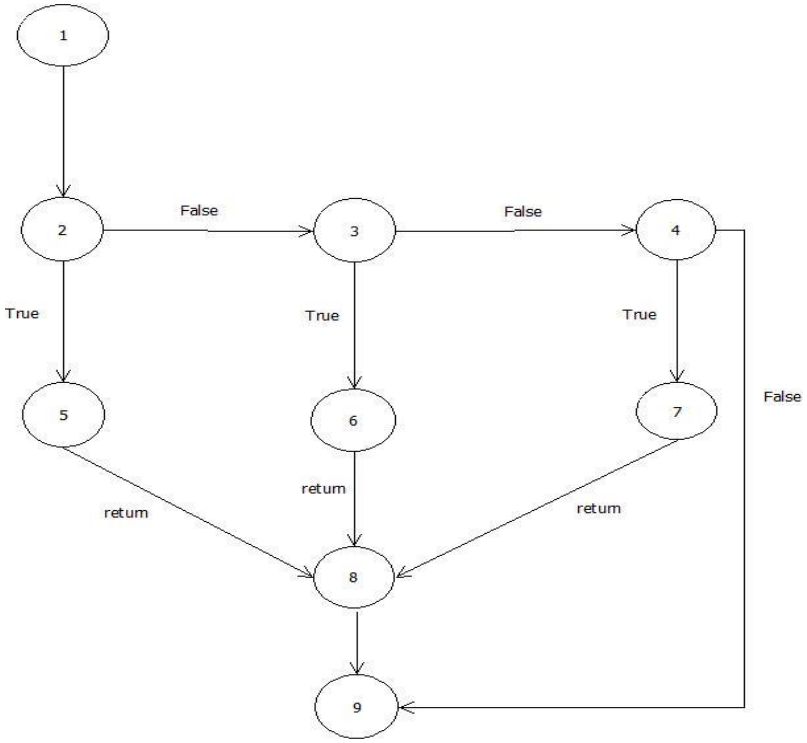
En caso de que la cantidad ingresada en cuenta sea menor que 0, se imprimirá un mensaje indicando que no se puede ingresar una cantidad negativa, y la variable iCodErr tomará el valor de 1.

En caso de que la cantidad ingresada en cuenta sea idéntica a 3, se imprimirá un mensaje indicando "Error detectable en pruebas de caja blanca", y la variable iCodErr tomará el valor de 2. En este caso, no se cumplirá nunca, ya que mientras el número sea menor que 0, se estará cumpliendo el primer caso, por lo que este código nunca llegará a ejecutarse.

Por último, en caso de que no se cumplan las condiciones anteriores, este código asignará a la variable dSaldo la cantidad introducida en el método ingresar, sumándolo al dSaldo existente, y la variable iCodErr tomará el valor 0.

Como conclusión, tras hacer un análisis de caja blanca, cabe decir que el siguiente código nunca llegará a ejecutarse, por lo que estaríamos ante un error de caja blanca, puesto que si la cantidad introducida es -3, será absorbida por la primera condición if(cantidad < 0).

GRAFO



Nodo	Línea - Condición
1	28-31
2	32 Cond cantidad < 0
3	37 Cond cantidad == -3
4	42 Cond cantidad > 0
5	34-35 iCodErr = 1, imprime mensaje.
6	39-40 iCodErr = 2, imprime mensaje.
7	45-46 iCodErr = 0, set dSaldo.
8	50 return iCodErr
9	51 Fin del método

COMPLEJIDAD CICLOMÁTICA

Método de cálculo	Complejidad	Comentarios
Nº de regiones	4	Hay que considerar la región exterior.
Nº de aristas - Nº nodos + 2	11 – 9 + 2 = 4	
Nº de condiciones + 1	3 + 1 = 4	Nodos 2, 3, 4.

CAMINOS DE PRUEBA

Camino 1: 1 - 2 - 5 - 8 - 9.

Camino 2: 1 - 2 - 3 - 6 - 8 - 9.

Camino 3: 1 - 2 - 3 - 4 - 8 - 9.

Camino 4: 1 - 2 - 3 - 4 - 9.

CASOS DE USO

Camino / Casos de uso	Datos			Salidas
	Cantidad	**iCodErr**	**dSaldo**	
1	<0	1	No actualizado	Sigue teniendo el mismo dSaldo. Retorna iCodErr = 1.
2	== -3	2	No actualizado	Sigue teniendo el mismo dSaldo. Retorna iCodErr = 2.
3	>0	0	dSaldo = dSaldo + Cantidad	Actualiza el saldo. Retorna iCodErr = 0.
4	Valor en formato no numérico.	No actualizado.	No actualizado.	Error. Fin del programa.

El camino 4 no viene reflejado en el programa.

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.**

Para realizar un análisis de caja negra del método retirar, consideraremos que valores nos arroja el método sin necesidad de observar el código. Es decir, probando su interfaz externa. En este caso los valores que habría que probar son los siguientes:

- Introduciendo un valor en cantidad que sea positivo y menor que el saldo actual ("dSaldo") debería dejar retirar la cantidad.
- Introduciendo un valor en cantidad que sea positivo y mayor que el saldo actual ("dSaldo") no debería dejar retirar la cantidad.
- Introduciendo un valor en cantidad que sea igual que el saldo actual ("dSaldo") debería dejar retirar la cantidad quedando el saldo a 0.

- Introduciendo un valor en cantidad que sea negativo no debería dejar retirar la cantidad.
- Introduciendo un valor en cantidad igual a 0 no debería dejar retirar la cantidad.

Clases de equivalencia y valores límites

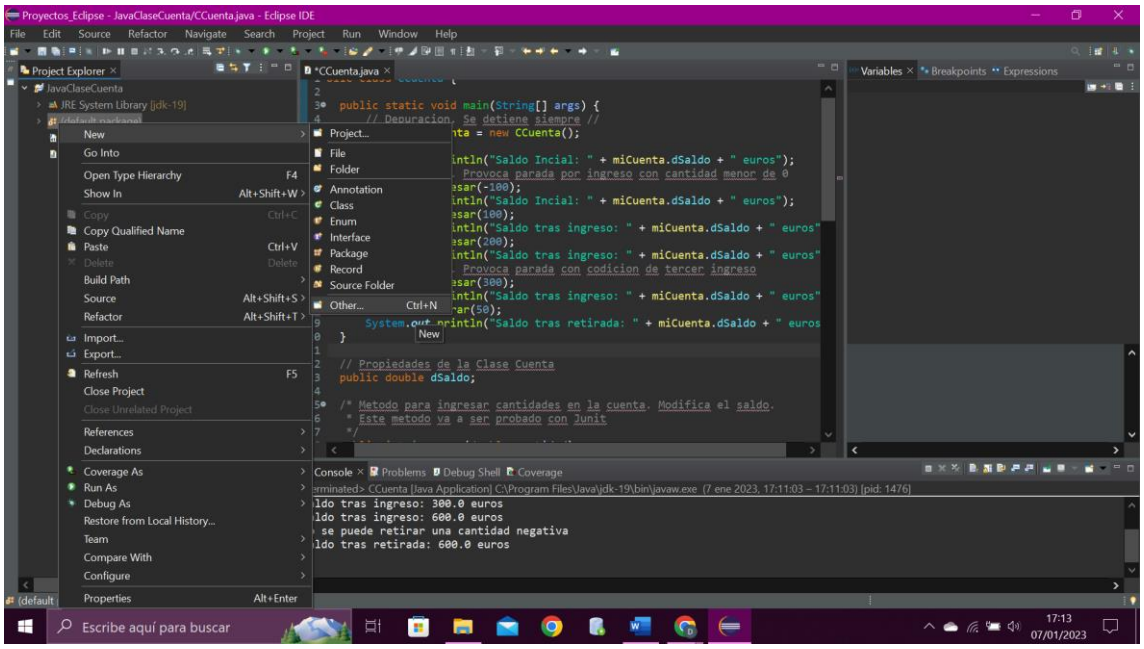
Condición de entrada	Clases de equivalencia	Clases válidas	COD	Clases no válidas	COD
Operación	Rango de valores de entrada.	Un dato entre 0 y dSaldo. Valor límite dSaldo (permitido)	C1B1	Un dato igual a 0.	C1E1
				Un dato menor que 0.	C1E2
			C1B2	Un dato mayor que dSaldo.	C1E3
				Inserta letras	C1E4

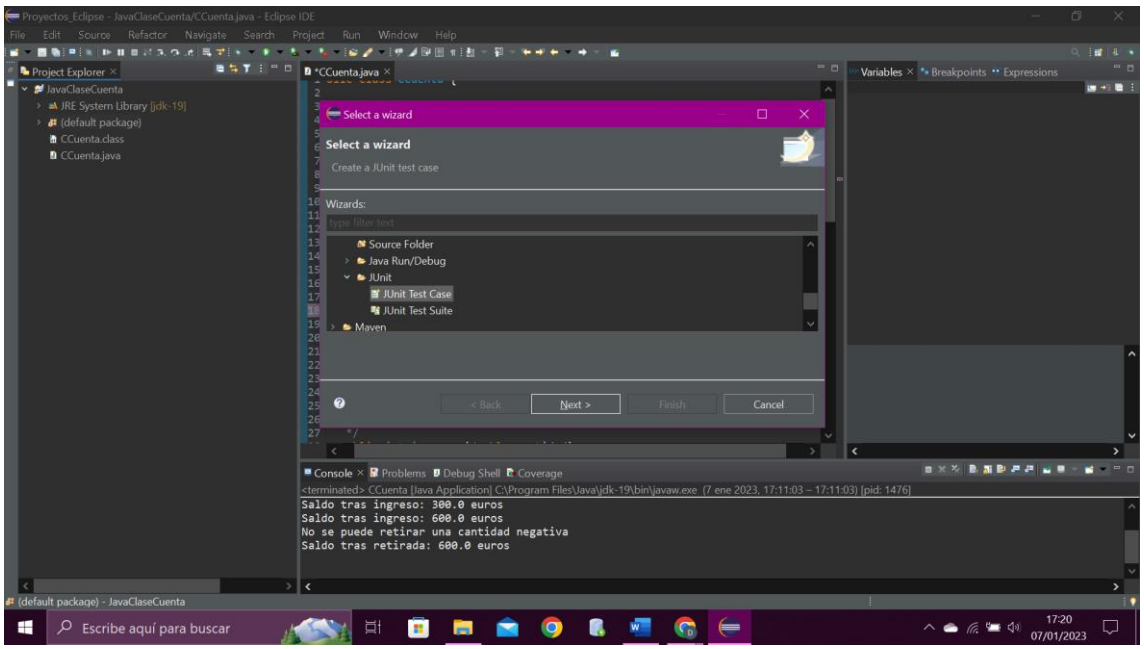
Casos de uso, resultados esperados y análisis

Casos de prueba	Clases de equivalencia	Condiciones de entrada	Resultado esperado
		Operación	
CP1	C1B1	0,01	Resta cantidad a dSaldo. dSaldo - 0,01.
CP2	C1B2	dSaldo	Resta cantidad a dSaldo dejando la cuenta a 0.
CP3	C1E1	0	No deja realizar operación.
CP4	C1E2	-0,01	No deja realizar operación.
CP5	C1E3	dSaldo + 0,01	No deja realizar operación.
CP6	C1E4	“ABCD”	No deja realizar la operación.

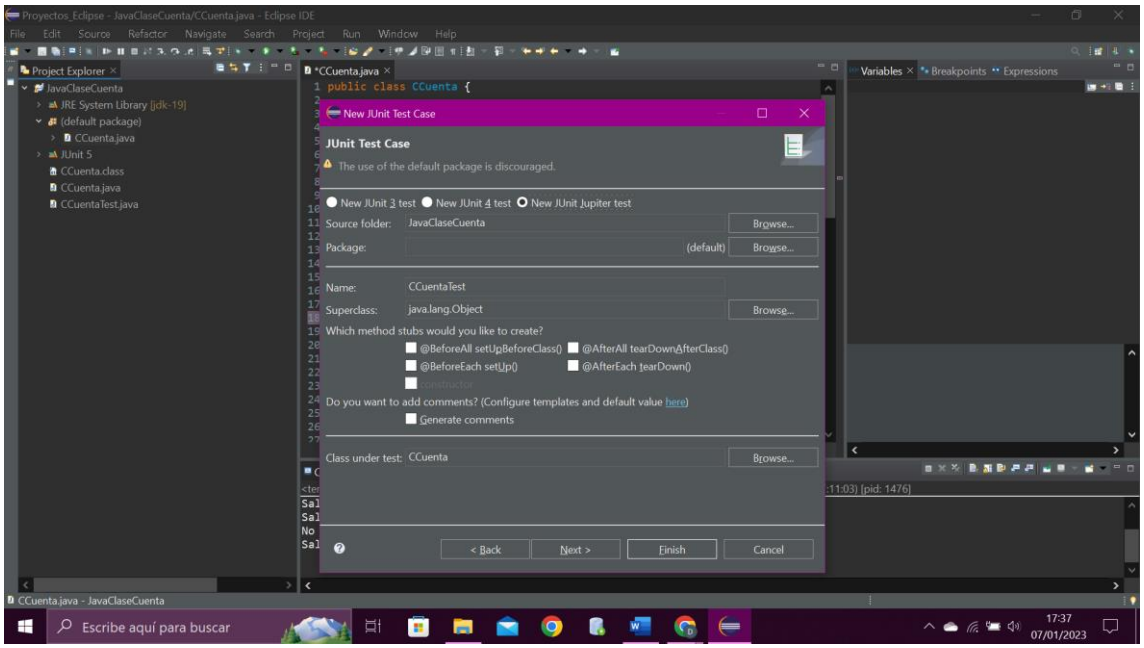
3. **Crea la clase CCuentaTest del tipo Caso de prueba JUnit en Eclipse que nos permita pasar las pruebas unitarias de caja blanca del método ingresar. Los casos de prueba ya los habrás obtenido en el primer apartado del ejercicio. Copia el código fuente de esta clase en el documento.**

Implementación de la clase JUnit5:

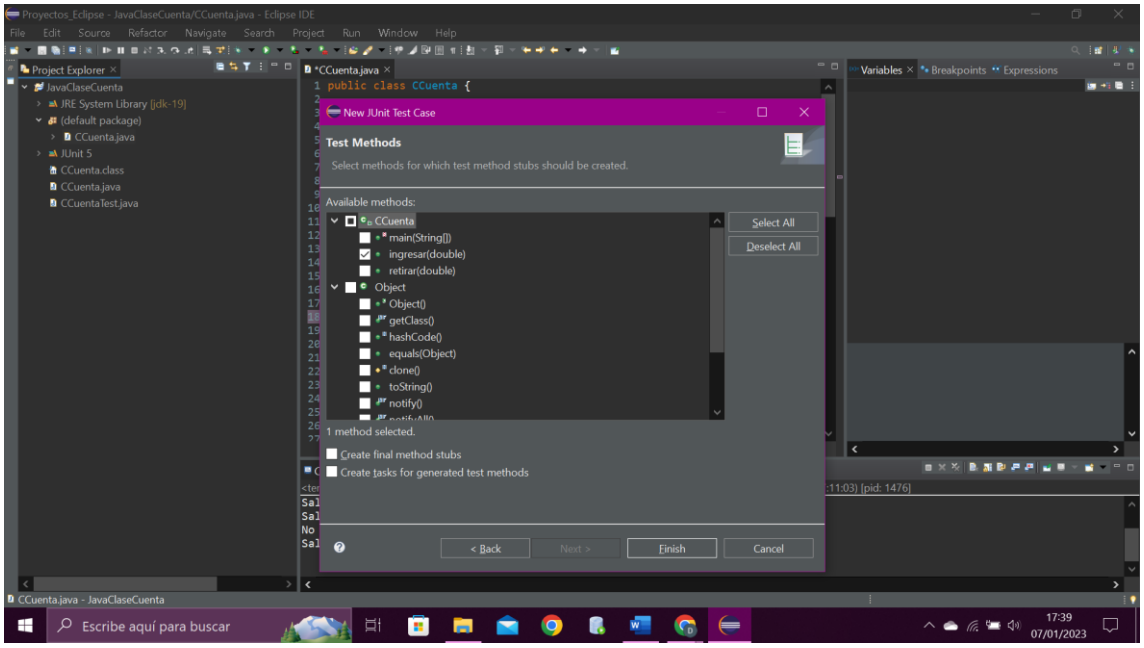


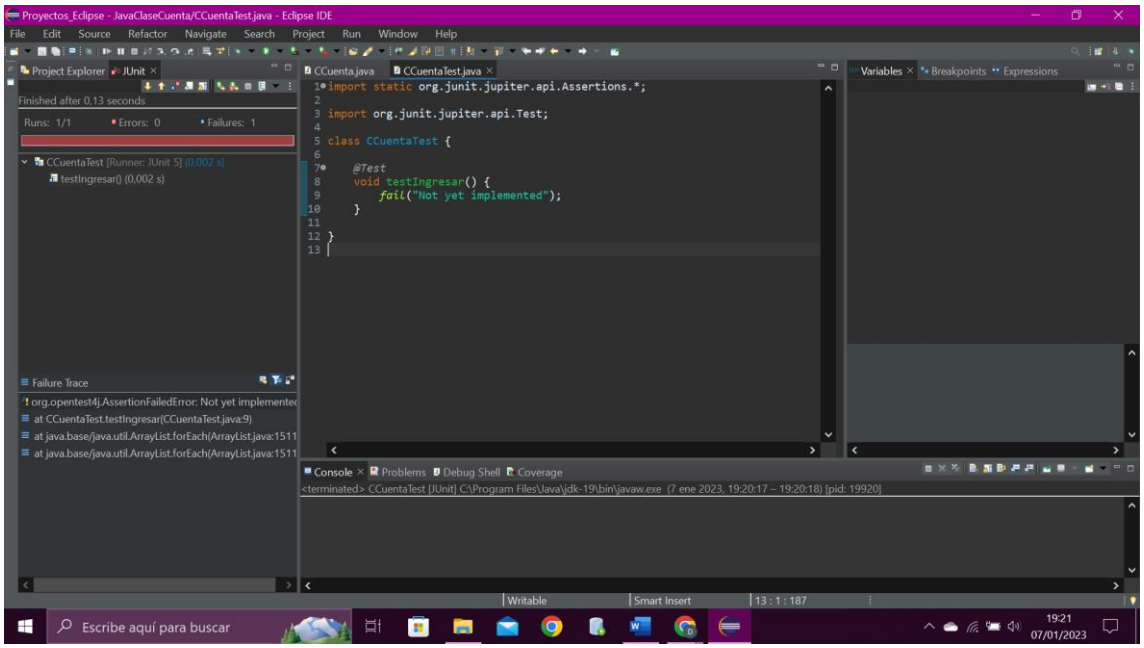


Indicamos nombre de la clase: CCuentaTest.

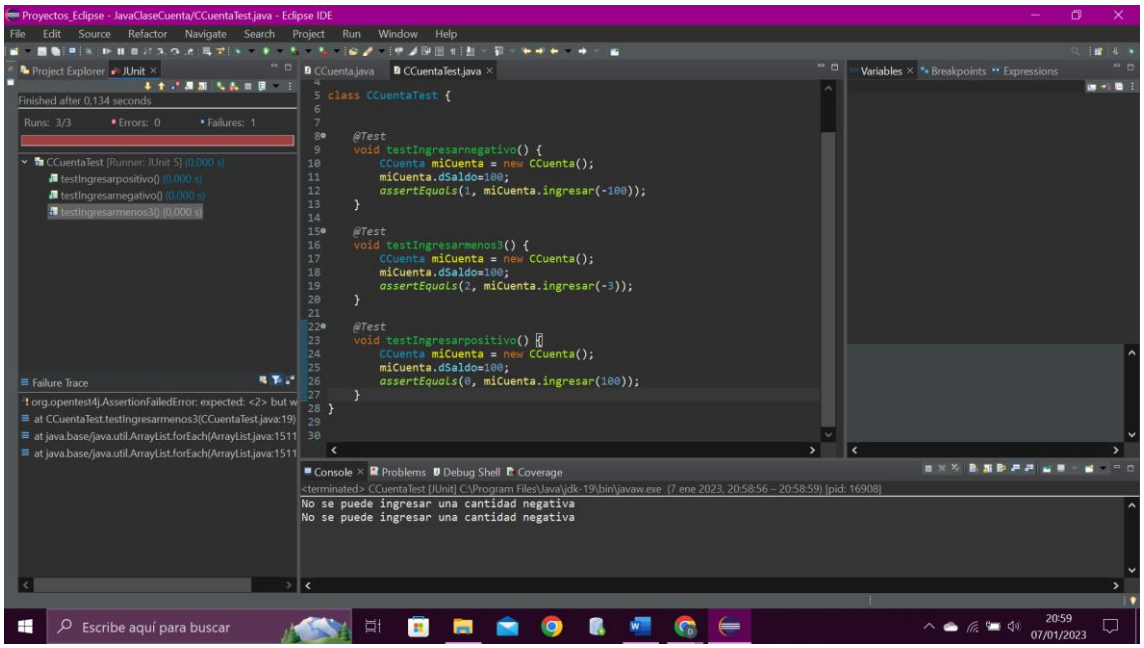


Seleccionamos el método que queremos testear, en este caso el método Ingresar(double).



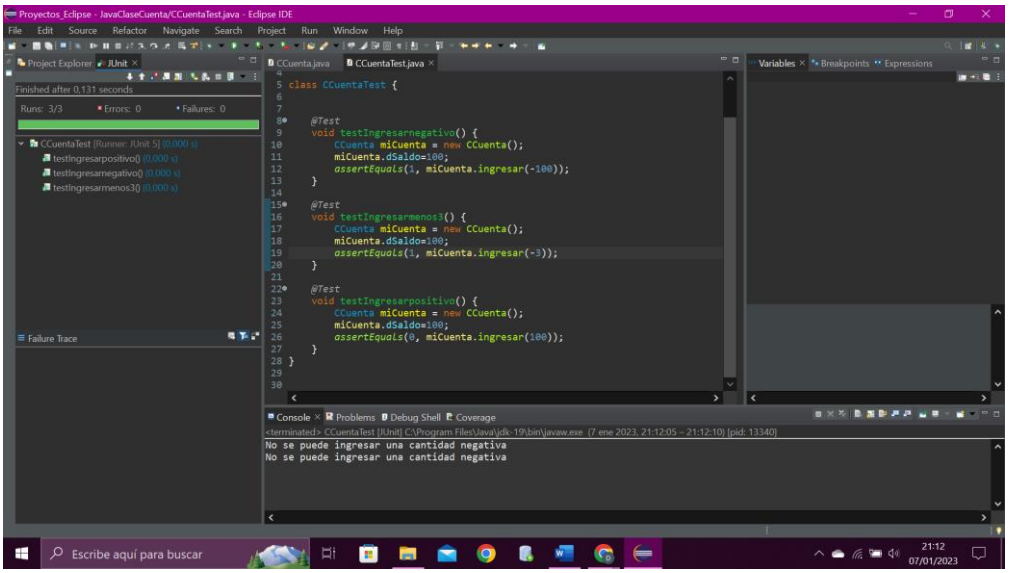


Para realizar la prueba del método, creamos una prueba unitaria para cada caso mencionado en el primer ejercicio. En cada una de las pruebas se inicializa un objeto de tipo cuenta con un valor de 100. Seguidamente se hace uso del método assertEquals de la clase Assert, que nos permite testear si el valor esperado, es realmente el valor arrojado por el método ingresar, es decir, qué valor de la variable iCodErr retorna el método.



Como se puede comprobar en la imagen anterior. Los test de numero positivo y número negativo no arrojan ningún error y funcionan según lo esperado. Pero el test que prueba el caso de que la cantidad introducida sea -3 arroja un fallo. Ya que

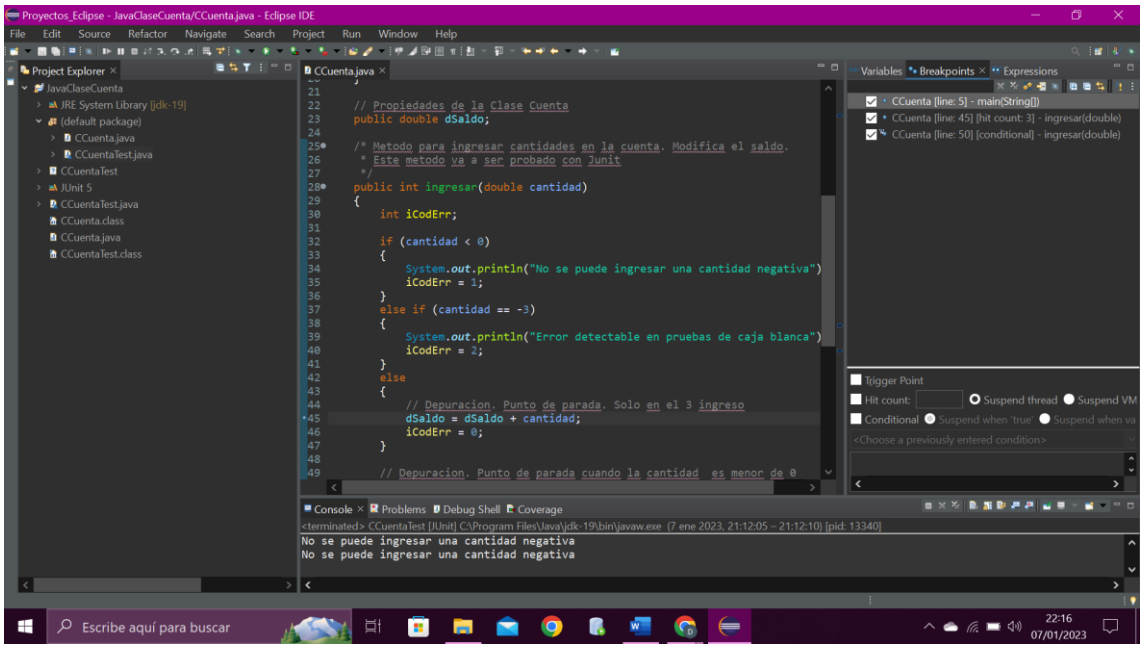
como se mencionaba en el ejercicio 1, este valor quedaría absorbido por el primer caso (condicional de si el número es negativo). Cosa que podemos comprobar, si cambiamos el valor esperado del método en el "testIngresamenos3" de ese caso por 1:



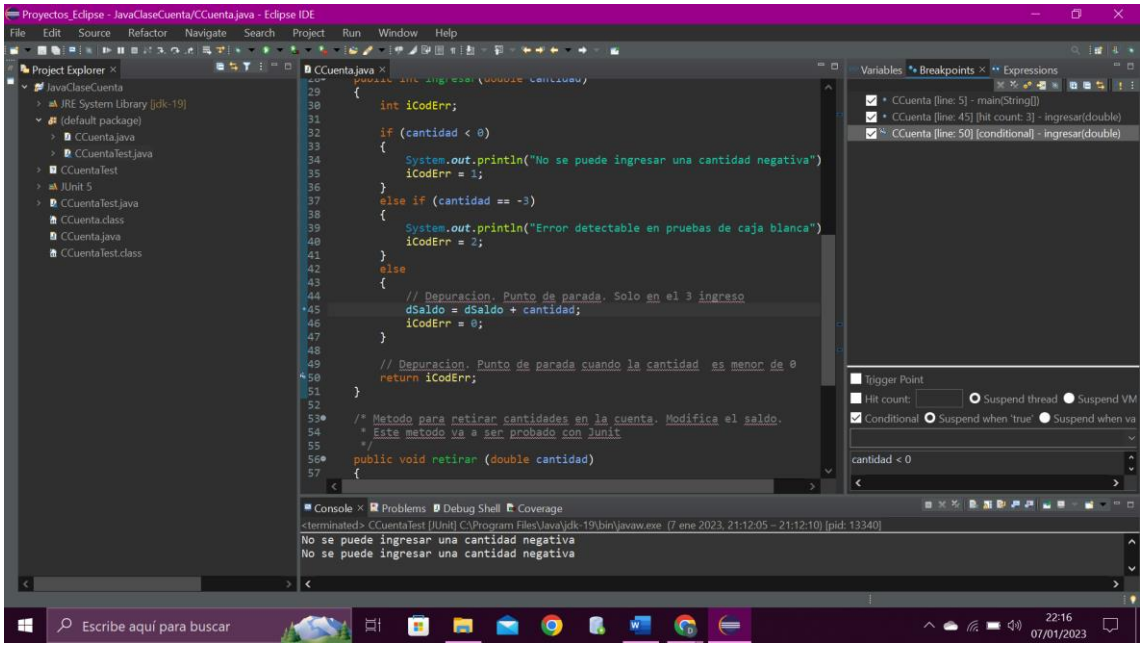
Como podemos ver, ahora ningún caso arroja fallos.

4. **Genera los siguientes puntos de ruptura para validar el comportamiento del método ingresar en modo depuración.**
    - **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.**
    - **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.**
    - **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.**

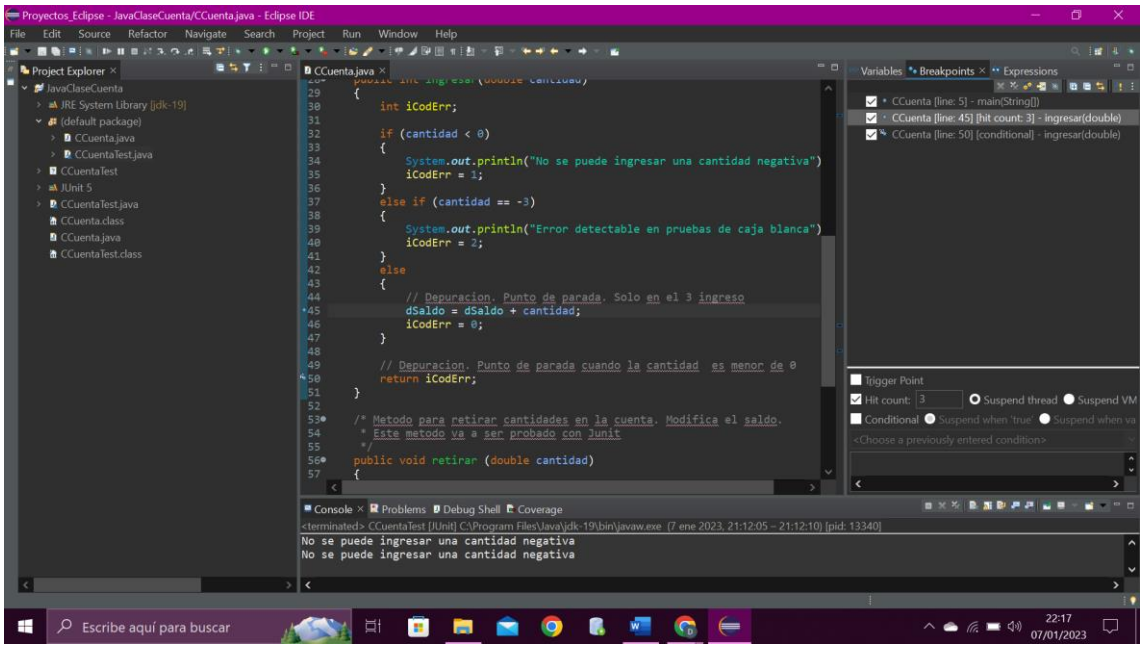
En primer lugar, he procedido a introducir los 3 puntos de ruptura en el método. En el primer punto de ruptura no se indica ninguna condición por lo que únicamente seleccionamos el lugar donde se detendrá la ejecución del programa por primera vez. Como podemos observar en la primera imagen, este punto de ruptura se encontraría en la línea 5:



Para el segundo punto de ruptura (línea 50), se pide que el programa se debe detener solo cuando la cantidad ingresada sea menor que 0. Por lo que, seleccionando este breakpoint en la parte derecha del IDE, marcamos la casilla Conditional y escribimos la condición para la que se detendrá (cantidad < 0).



Para el tercer punto de ruptura (línea 45) del programa, se pide que se detenga su ejecución cuando el saldo se actualice por tercera vez. En este caso, seleccionando el breakpoint correspondiente en el cuadro de la derecha del IDE, marcamos la casilla “Hit count” e introducimos el número 3.



Puntos de ruptura

<?xml version="1.0" encoding="UTF-8"?>

<breakpoints>

<breakpoint enabled="true" persistant="true" registered="true">

<resource path="/JavaClaseCuenta/CCuenta.java" type="1"/>

<marker charStart="116" lineNumber="5"
type="org.eclipse.jdt.debug.javaLineBreakpointMarker">

<attrib name="charStart" value="116"/>

<attrib name="org.eclipse.jdt.debug.core.suspendPolicy" value="2"/>

<attrib name="org.eclipse.jdt.debug.ui.JAVA_ELEMENT_HANDLE_ID"
value="=JavaClaseCuenta/<{CCuenta.java[CCuenta"/>

<attrib name="charEnd" value="154"/>

<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" persistant="true" registered="true">

<resource path="/JavaClaseCuenta/CCuenta.java" type="1"/>

<marker charStart="1604" lineNumber="45"
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="=JavaClaseCuenta/<{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: 45] [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" persistant="true" registered="true">

<resource path="/JavaClaseCuenta/CCuenta.java" type="1"/>

<marker charStart="1765" lineNumber="50"
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 < 0"/>

<attrib name="org.eclipse.jdt.debug.ui.JAVA_ELEMENT_HANDLE_ID"
value="=JavaClaseCuenta/<{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: 50] [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>