

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

Código fuente original

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

Haciendo las pruebas he comprobado que había un bloque de código que nunca que iba a poder ejecutar por lo tanto el código correcto para que todas las condiciones y partes del código se ejecuten queda así

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

Grafo	Nodo	Linea – Condición
<pre> graph TD     1((1)) --&gt; 2((2))     2 -- false --&gt; 3((3))     2 -- true --&gt; 4((4))     4 -- false --&gt; 5((5))     4 -- true --&gt; 6((6))     3 --&gt; 7((7))     5 --&gt; 7     6 --&gt; 7     7 -- return --&gt; 8((8))     8 -- fin </pre>	1	Linea: 29
	2	Linea: 31 Condición: “Cantidad < 0”
	3	Lineas: 41-45
	4	Linea 33 – Condición Cantidad == -3
	5	Lineas 36 - 39
	6	Lineas 34 - 35
	7	Linea: 48
	8	Linea 49

### Complejidad de McCabe o ciclomática

Método de cálculo	Complejidad	Comentarios
N.º de regiones	3	Incluida la exterior
N.º de aristas – n.º de Nodos + 2	$9 - 8 + 2 = 3$	
N.º de condiciones + 1	$2 + 1 = 3$	Nodos 2 y 4

### Caminos de prueba

- Camino 1 → 1, 2, 3, 7, 8.
- Camino 2 → 1, 2, 4, 6, 7, 8.
- Camino 3 → 1, 2, 4, 5, 7, 8.

## Casos de uso, resultados esperados y análisis

Caminos / Casos de uso	Datos	Salida
	cantidad	
1	-1	En pantalla: "No se puede ingresar una cantidad negativa" Retorna un 1
2	-3	En pantalla: "Error detectable en pruebas de caja blanca" Retorno un 2
3	100	Retorna un 0

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.

```

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
    {
        dSaldo = dSaldo - cantidad;
    }
}

```

## Clases de equivalencia

Rango de valores de entrada.	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.	Clase válida:
		Valor entre 1 y el saldo
		Clases no válidas
		<ul style="list-style-type: none"> <li>Menor que 0</li> <li>Mayor que Saldo</li> </ul>

## Análisis de Valores límite.

Caso	Clase de equivalencia	Valores límite
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.	Clase válida	Clase válida
	Valor entre 1 y el saldo Caso 1: 150	Caso 4: 0.01 Caso 5: dSaldo
	Clases no válidas	Clases no válidas
	<ul style="list-style-type: none"> <li>Menor que 0 <ul style="list-style-type: none"> <li>Caso 2: -100</li> </ul> </li> <li>Mayor que saldo <ul style="list-style-type: none"> <li>Caso 3: dSaldo + 5000</li> </ul> </li> </ul>	Caso 6 Menor que 0: -0.01 Caso 7 Mayor que saldo: dSaldo + 0.01

## Conjetura de Errores.

Una prueba típica que no hemos tenido en cuenta en los casos anteriores es el valor de 0 ya que para tipo de datos numéricos se puede dar el caso que se realiza alguna división y el valor de cero puede provocar una excepción y la finalización del programa en caso que no estuviera controlada.

- **Caso 8** valor 0

### Determinar las clase de equivalencia y analizar los valores límite

Condiciones de entrada	Clase de equivalencia	Clases válidas	COD	Clases no válidas	COD
Número decimales	Valores Importe que se quieren retirar	Caso 1	Cod1	Caso 2	Cod2
		Caso 4	Cod4	Caso 3	Cod3
		Caso 5	Cod5	Caso 6	Cod6
				Caso 7	Cod7
				Caso 8	Cod8

### Casos de uso, resultados esperados y análisis

Casos de prueba	Clase de equivalencia	Condiciones / Valor	Resultado Esperado
PRU-1	Cod1	Valor entre 1 y el saldo / 150	dSaldo = dSaldo - cantidad
PRU-2	Cod2	Menor que 0 / -100	No se puede retirar un cantidad negativa
PRU-3	Cod3	Mayor que saldo / saldo + 5000	No hay suficiente saldo
PRU-4	Cod4	0.01	dSaldo = dSaldo - cantidad
PRU-5	Cod5	sSaldo	dSaldo = dSaldo – cantidad / Saldo se queda a 0
PRU-6	Cod6	Menor que 0 / -0.01	No se puede retirar un cantidad negativa
PRU-7	Cod7	Mayor que saldo / dSaldo + 0.01	No hay suficiente saldo
PRU-8	Cod8	0	No se puede retirar un cantidad negativa
PRU-9	Cod9	0	El valor 0 no hace ninguna función

Observación: al introducir en las pruebas el cero observamos que salta el mensaje de “No se puede retirar un cantidad negativa” por lo tanto generamos un aviso para que sea modificado y controlado dicho valor, ya que bien es cierto que no altera el correcto funcionamiento del programa pero muestra un error que no se corresponde con el cero.

Adjunto Recomendación de código.

```
public void retirar (double cantidad)
{
    if (cantidad <= 0)
    {
        if (cantidad == 0) {
            System.out.println("El valor 0 no realiza ninguna función");
        } else {
            System.out.println("No se puede retirar una cantidad negativa");
        }
    }
    else if (dSaldo < cantidad)
    {
        System.out.println("No se hay suficiente saldo");
    }
    else
    {
        dSaldo = dSaldo - cantidad;
    }
}
```

3. Crea la clase CCuentaTest del tipo Caso de prueba JUnit 5 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 1 y tendrás que aplicarlo en el código de la prueba. Copia el código fuente correspondiente que te proporcionamos.

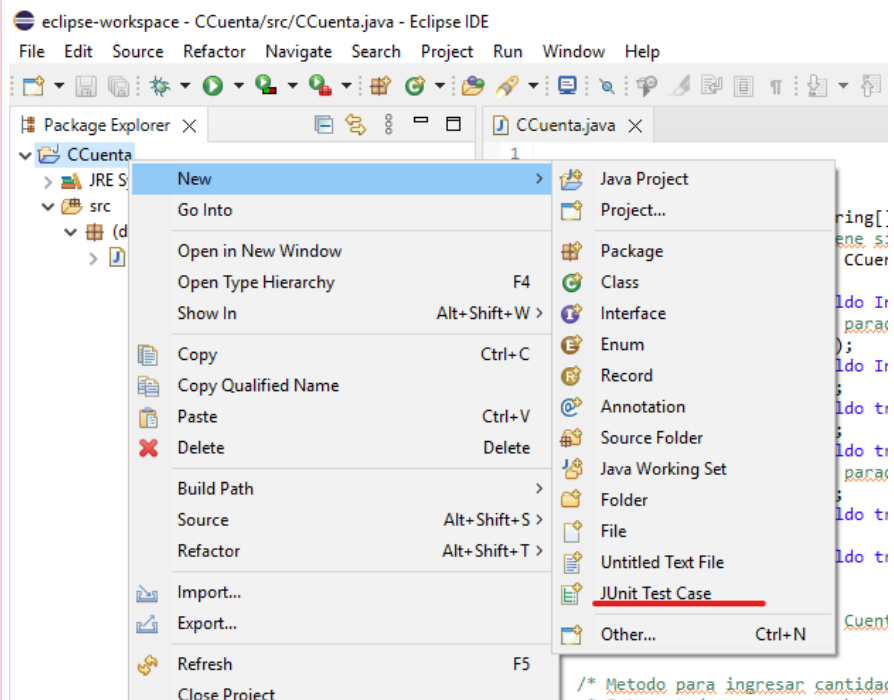
Creamos un nuevo proyecto llamado CCuenta y copiamos en el CCuenta.java el fichero que se aporta y con el que hemos realizado las pruebas de caja blanca y caja negra de los ejercicios anteriores.

```

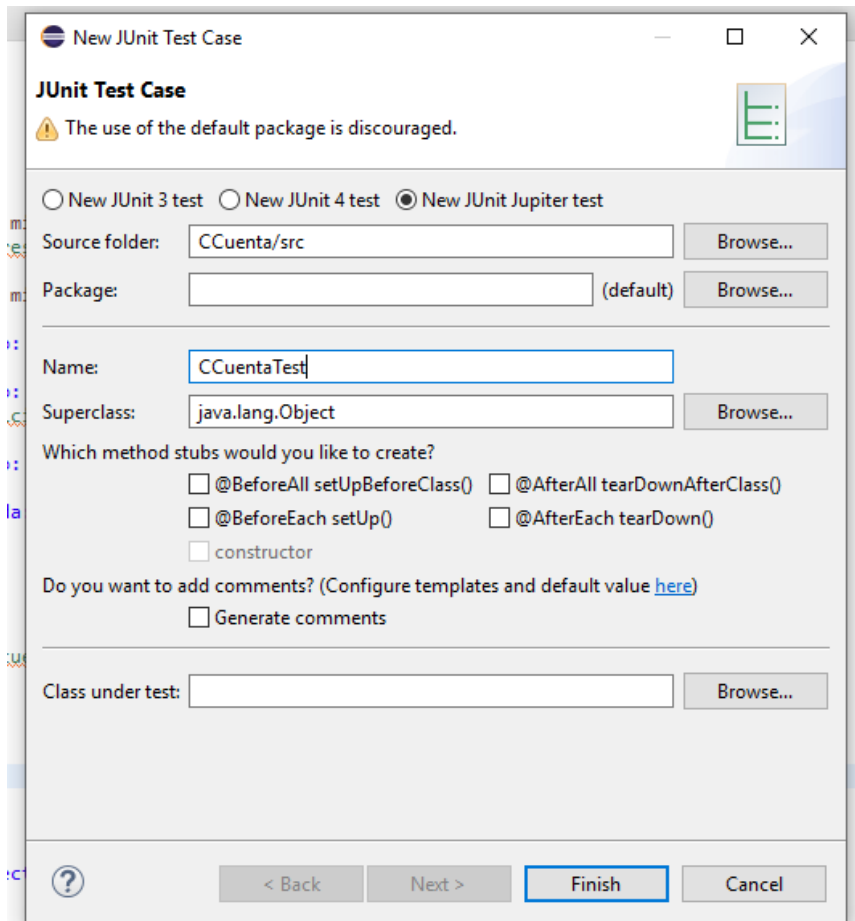
1 public class CCuenta {
2
3
4     public static void main(String[] args) {
5         // Depuracion. Se detiene siempre
6         CCuenta miCuenta = new CCuenta();
7
8         System.out.println("Saldo Inicial: " + miCuenta.dSaldo + " euros");
9         // Depuracion. Provoca parada por ingreso con cantidad menor de 0
10        miCuenta.ingresar(-100);
11        System.out.println("Saldo Inicial: " + miCuenta.dSaldo + " euros");
12        miCuenta.ingresar(100);
13        System.out.println("Saldo tras ingreso: " + miCuenta.dSaldo + " euros");
14        miCuenta.ingresar(200);
15        System.out.println("Saldo tras ingreso: " + miCuenta.dSaldo + " euros");
16        // Depuracion. Provoca parada con codicion de tercer ingreso
17        miCuenta.ingresar(300);
18        System.out.println("Saldo tras ingreso: " + miCuenta.dSaldo + " euros");
19        miCuenta.retirar(50);
20        System.out.println("Saldo tras retirada: " + miCuenta.dSaldo + " euros");
21    }
22
23    // Propiedades de la Clase Cuenta
24    public double dSaldo;
25
26    /* Metodo para ingresar cantidades en la cuenta. Modifica el saldo.
27     * Este metodo va a ser probado con Junit
28     */
29    public int ingresar(double cantidad) {
30        int iCodErr;
31
32        if (cantidad < 0) {
33
34            if (cantidad == -3) {

```

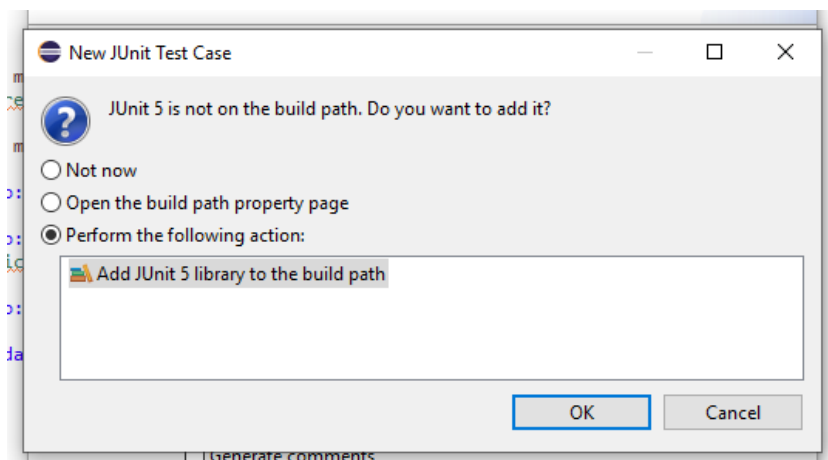
Con el botón derecho del ratón en el nombre del proyecto desplegamos el menú NEW y seleccionamos Junit Test Case



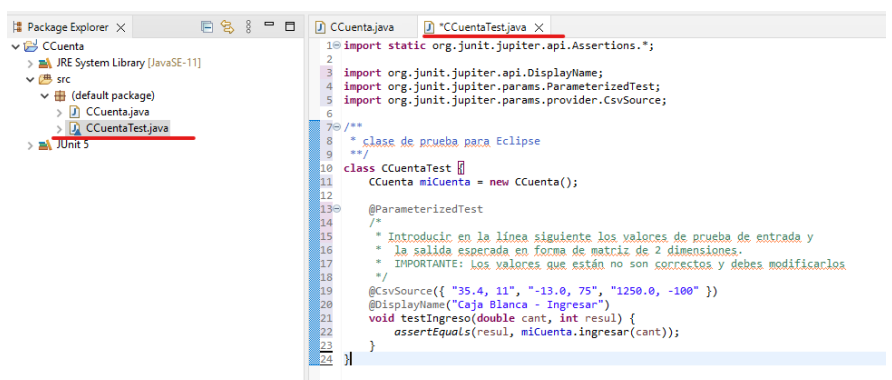
En la ventana que no muestra le datos un nombre a Junit Test → CcuentaTest.java



Le indicamos que queremos añadir la JUnit 5 para realizar las prueba unitarias.



Añadimos el código de test que se nos aporta al proyecto.



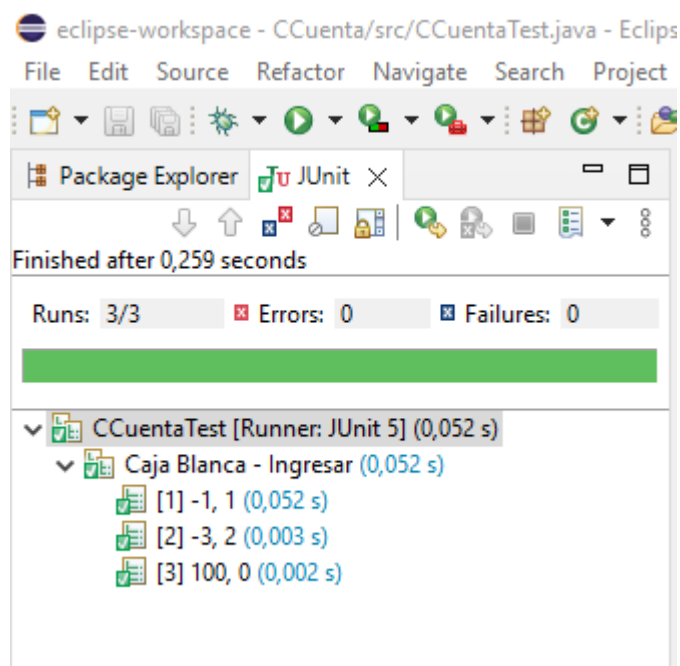
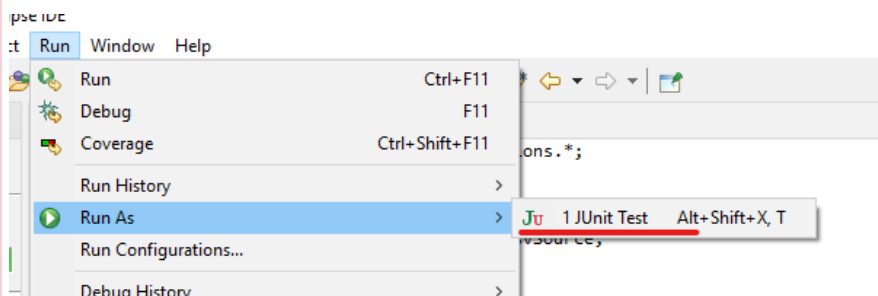
Cambiamos los datos con los que queremos hacer el test por los que obtuvimos en la prueba de caja blanca.

En el menú RUN ejecutamos el JUnit Test

Y observamos que las pruebas de TEST se han pasado con éxito.

```
* IMPORTANTE: Los valores que están no son correctos y debe:
*/
@CsvSource({ "35.4, 11", "-13.0, 75", "1250.0, -100" })
@DisplayName("Caja Blanca - Ingresar")
void testIngreso(double cant, int resul) {
    assertEquals(resul, miCuenta.ingresar(cant));
}

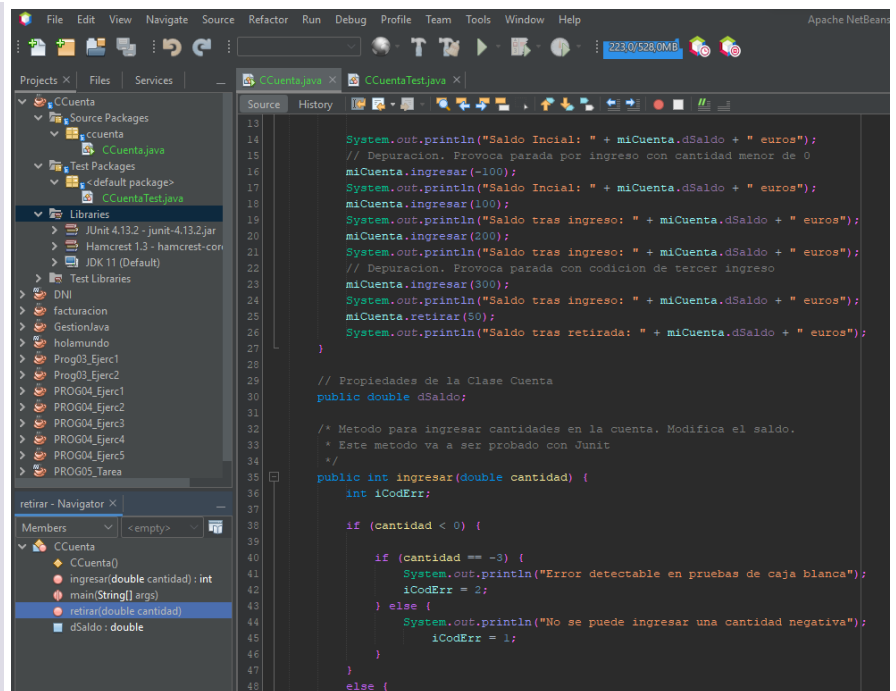
*/
@CsvSource({ "-1, 1", "-3, 2", "100, 0" })
@DisplayName("Caja Blanca - Ingresar")
void testIngreso(double cant, int resul) {
    assertEquals(resul, miCuenta.ingresar(cant));
}
```



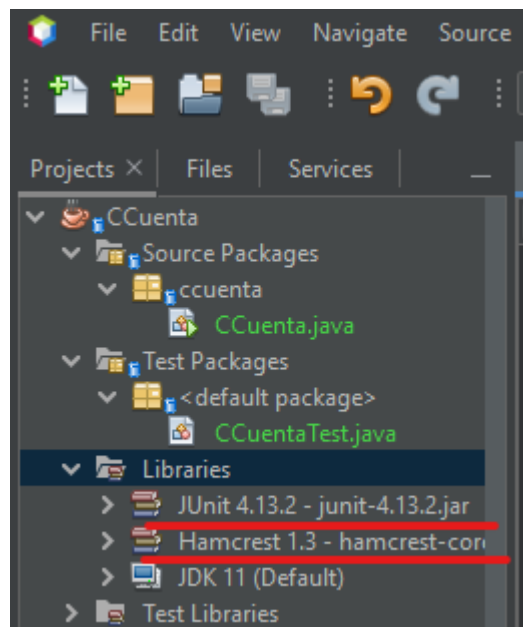


4. Crea la clase CCuentaTest del tipo Caso de prueba JUnit 4 en Netbeans que nos permita pasar las pruebas unitarias de caja blanca del método ingresar. Los casos de prueba ya los habrás obtenido en el primer apartado del ejercicio 1 y tendrás que aplicarlo en el código de la prueba. Copia el código fuente de esta clase que te proporcionamos. Será necesario quitar las librerías de Junit 5.6 y poner las de Junit 4.13.2 y Hamcrest 1.3

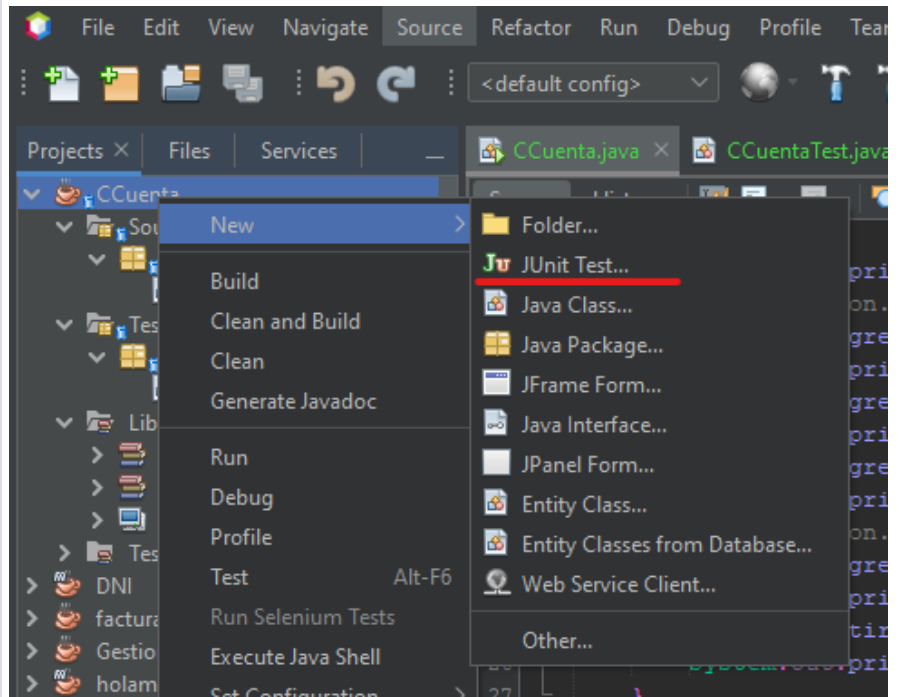
Creamos un nuevo proyecto llamado CCuenta y copiamos en el CCuenta.java el fichero que se aporta y con el que hemos realizado las pruebas de caja blanca y caja negra de los ejercicios anteriores.



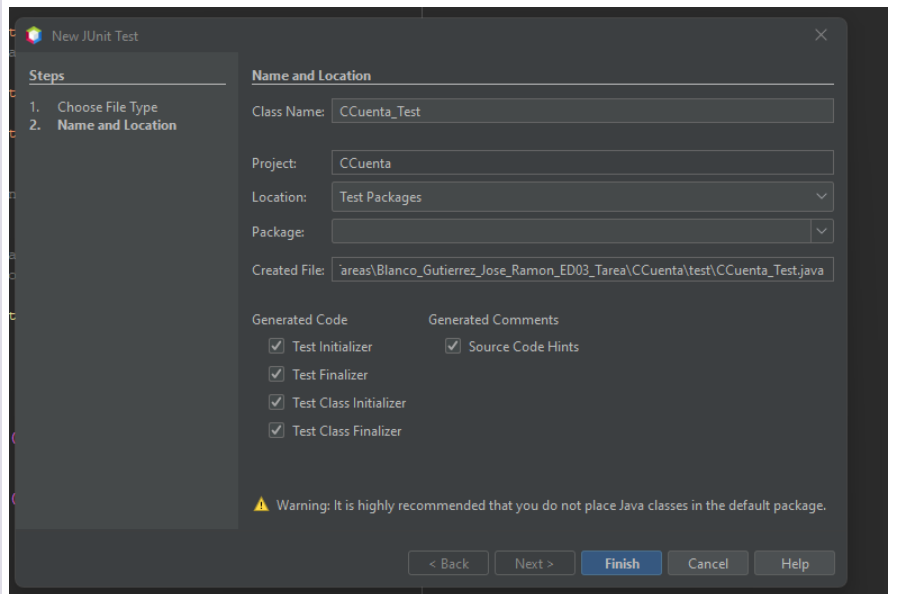
En librerías añadimos las dos librerías que necesita y que nos indica en el enunciado Junit 4.13.2 y Hamcrest 1.3



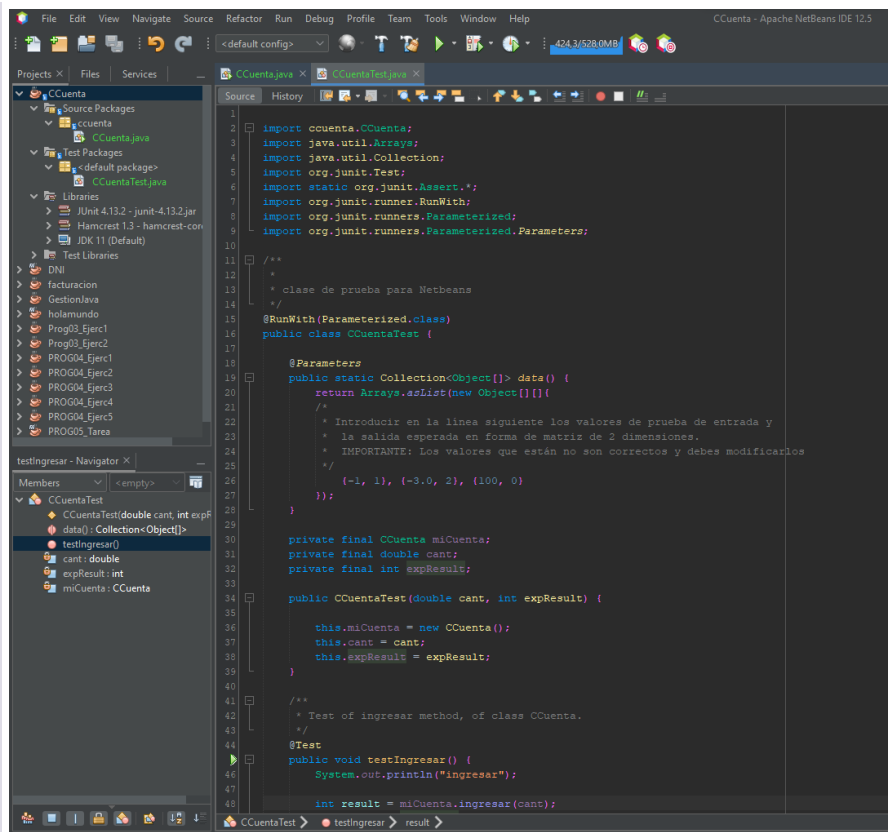
Con el botón derecho de ratón sobre el proyecto desplegamos el menú y vamos a New y seleccionamos JUnit Test



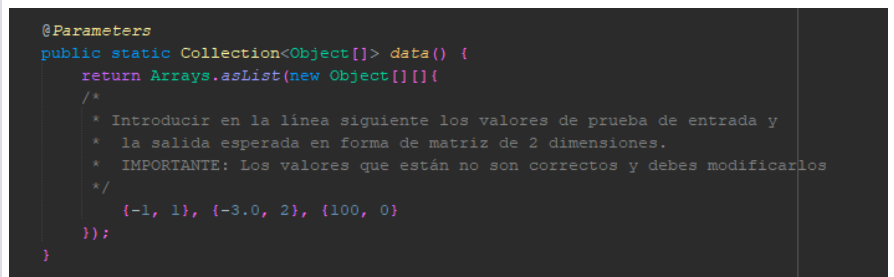
Indicamos el nombre para el test que sera el mismo de la clase que vamos a testear seguido de la palabra TEST



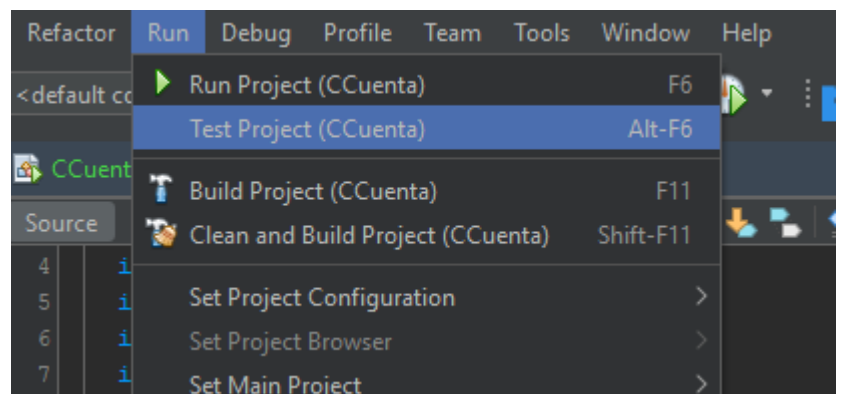
Copiamos el código que nos aporta la tarea sustituyendo todo el contenido de la clase CCuentaTest.java.



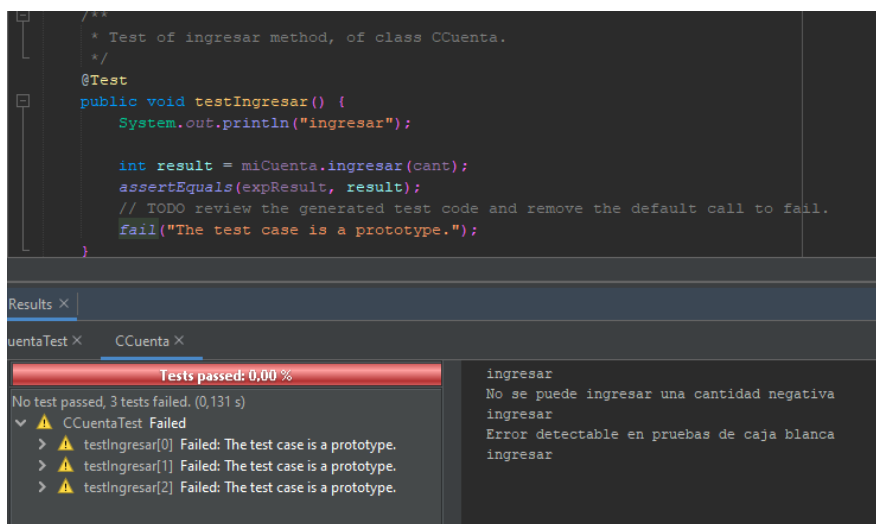
Como es una prueba de caja blanca ponemos los 3 valores de la Tabla de caja de usos y valores esperados en donde las llaves del fichero de Test, tantas llaves se paradas con comas que pongamos serán los test que ira realizando en nuestro caso solo obtuvimos 3.



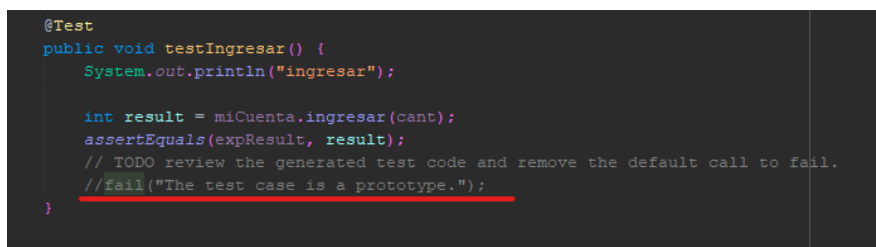
Y en el menú de RUN lanzamos el Test Project.



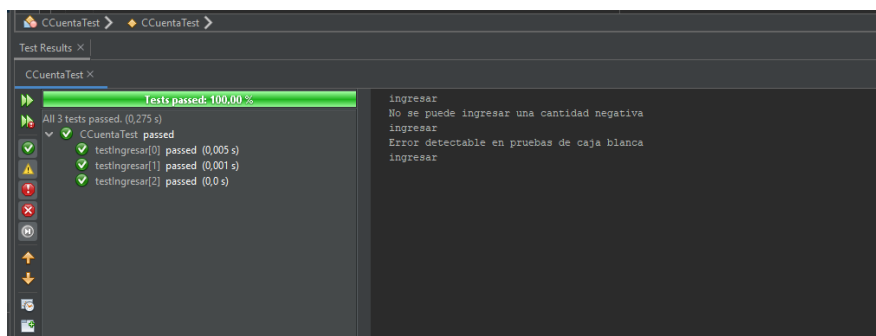
Nos aparece un error debido a que se está ejecutando un método que ocasiona que falle.



Para evitar que nos de error tenemos que comentar la linea del método fail



Y después volvemos a ejecutar el test y en la parte inferior observamos que la prueba de Caja Blanca se supera con éxito.

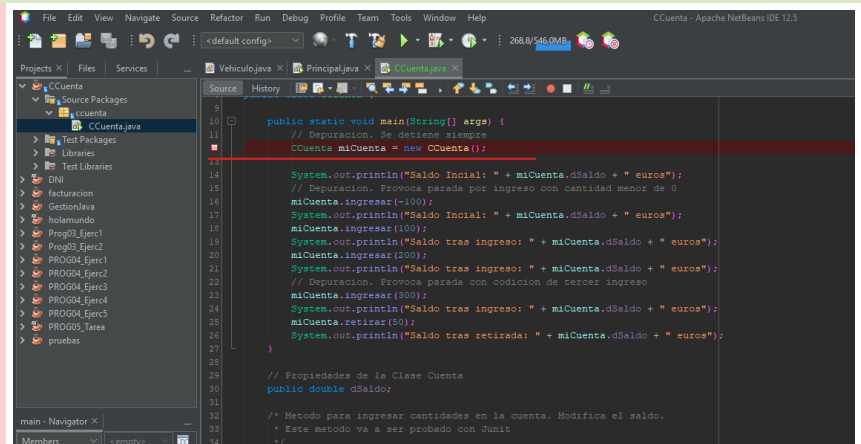


## 5. Genera los siguientes puntos de ruptura para validar el comportamiento del método ingresar en modo depuración.

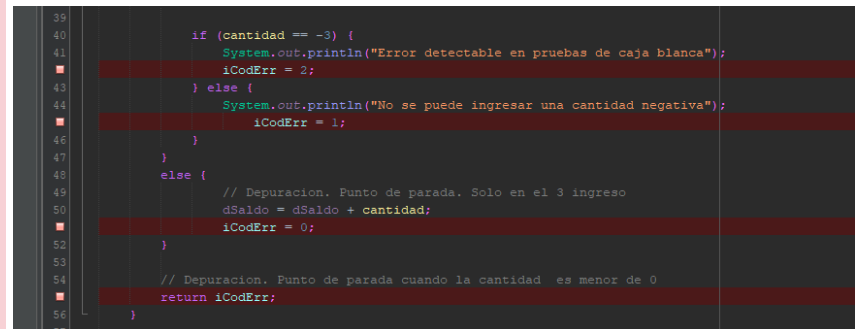
- Punto de parada sin condición al crear el objeto miCuenta en la función main.
- Punto de parada en cada instrucción del método ingresar que devuelva un código de error (Nota importante: Se debe corregir el código para que el flujo de programa pase por estos 3 puntos de parada)

### Netbeans (debug)

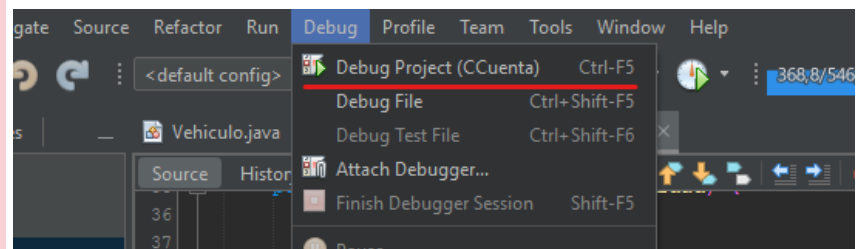
Se establece el Punto de parada en el Objeto miCuenta.



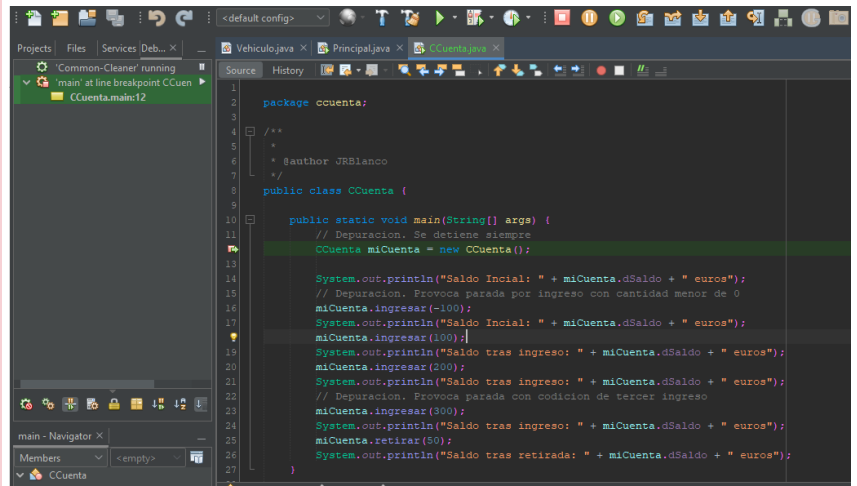
Se marcan los PUNTOS de parada en todos los sitios que el código introduce un dato en iCodErr y también establece uno en el return de la función.



Ejecutamos en proyecto en modo Debug y vamos viendo como se ejecuta y se va parando en los diferentes puntos...



## Punto parada de miCuenta



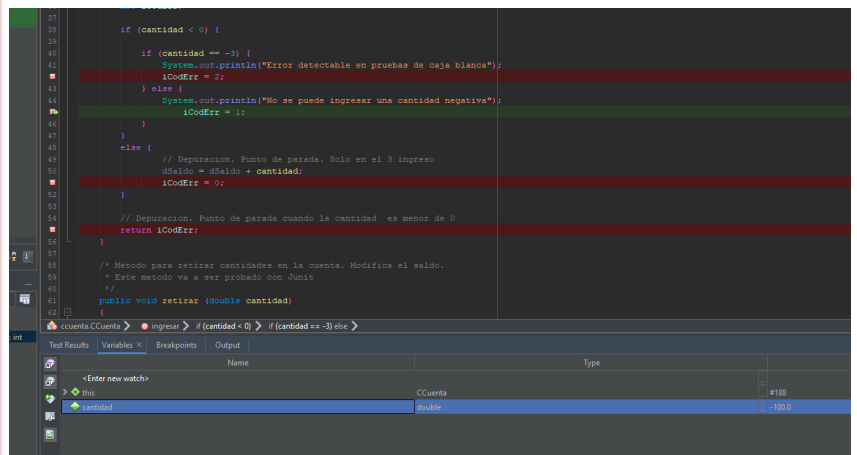
```
package ccuenta;

/**
 *
 * @author JRBlanco
 */
public class CCuenta {

    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");
    }
}
```

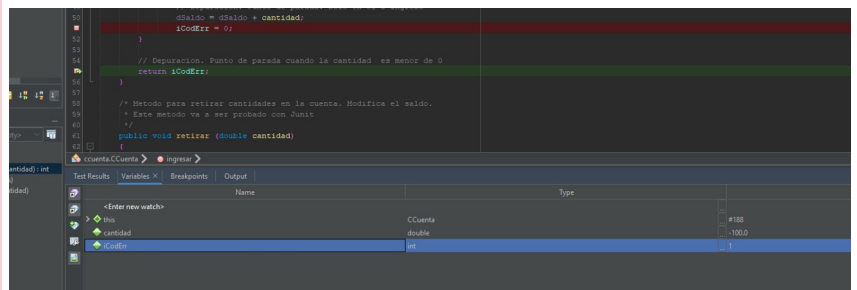
Como el dato es -100 mete el valor de 1 en iCodErr



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

Name	Type	Value
this	CCuenta	#188
cantidad	double	-100.0

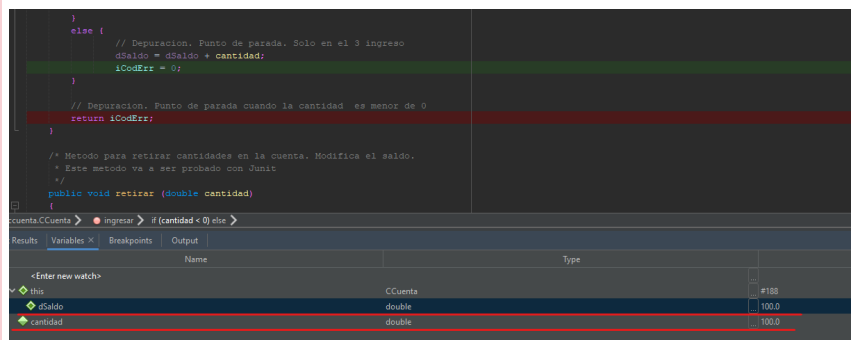
Y efectivamente retorna 1



```
dSaldo = dSaldo + cantidad;
iCodErr = 0;
}
// Depuracion. Punto de parada cuando la cantidad es menor de 0
return iCodErr;
}
```

Name	Type	Value
this	CCuenta	#188
cantidad	double	-100.0
iCodErr	int	1

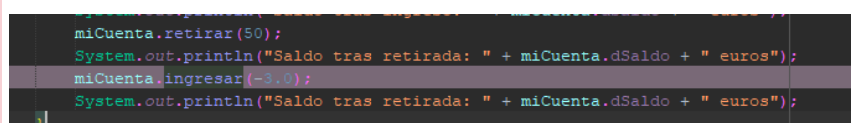
Con el valor 100 vemos que va correctamente a introducir un código 0 y también observamos que ha aumentado dSaldo...



```
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;
}
```

Name	Type	Value
this	CCuenta	#188
dSaldo	double	100.0
cantidad	double	100.0

Para comprobar que se para cuando se introduce -3 he creado dicha prueba



```
miCuenta.retirar(50);
System.out.println("Saldo tras retirada: " + miCuenta.dSaldo + " euros");
miCuenta.ingresar(-3.0);
System.out.println("Saldo tras retirada: " + miCuenta.dSaldo + " euros");
}
```

Y funciona correctamente.

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

/* Metodo para retirar cantidades en la cuenta. Modifica el saldo.
 * Este metodo va a ser probado con JUnit
 */
}
```

## Eclipse (Debug)

Se establece el primero punto de ruptura en la creación del objeto miCuenta.

```
eclipse-workspace - CCuenta/src/CCuenta.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer JUnit CCuentaTest.java
Runs: 0/0 Errors: 0 Failures: 0

1 public class CCuenta {
2
3
4
5 // Depuracion. Se detiene siempre
6 CCuenta miCuenta = new CCuenta();
7
8 System.out.println("Saldo Inicial: " + miCuenta.dSaldo + " euros");
9 // Depuracion. Provoca parada por ingreso con cantidad menor de 0
10 miCuenta.ingresar(-100);
11 System.out.println("Saldo Inicial: " + miCuenta.dSaldo + " euros");
12 miCuenta.ingresar(100);
13 System.out.println("Saldo tras ingreso: " + miCuenta.dSaldo + " euros");
14 miCuenta.ingresar(200);
15 System.out.println("Saldo tras ingreso: " + miCuenta.dSaldo + " euros");
16 // Depuracion. Provoca parada con codicion de tercer ingreso
17 miCuenta.ingresar(300);
18 System.out.println("Saldo tras ingreso: " + miCuenta.dSaldo + " euros");
19 miCuenta.retirar(50);
20 }
```

Y los siguientes puntos de ruptura en cada una de las opciones donde se modifica o retorna el valor a iCodErr

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

Lanzamos el programa en modo debug

```
Debug Console
Project Explorer
CCuentaTest.java
Thread (main) (Suspended Breakpoint at line 51 in CCuentaTest.java)
CCuentaTest.java (line 51)
C:\Program Files\Java\jdk-11.0.2\bin\java.exe (71 ms, 2022/08/02)

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

Adjunto el documento de puntos de ruptura que genera el Eclipse.

<?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="115" lineNumber="6" type="org.eclipse.jdt.debug.javaLineBreakpointMarker">
<attrib name="charStart" value="115"/>
<attrib name="org.eclipse.jdt.debug.core.suspendPolicy" value="2"/>
<attrib name="org.eclipse.jdt.debug.ui.JAVA_ELEMENT_HANDLE_ID" value="=CCuenta/src&lt;
{CCuenta.java[CCuenta"/>
<attrib name="charEnd" value="153"/>
<attrib name="org.eclipse.debug.core.enabled" value="true"/>
<attrib name="message" value="Line breakpoint:CCuenta [line: 6] - main(String[])" />
<attrib name="org.eclipse.jdt.debug.core.installCount" value="1"/>
<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="1460" lineNumber="38" type="org.eclipse.jdt.debug.javaLineBreakpointMarker">
<attrib name="charStart" value="1460"/>
<attrib name="org.eclipse.jdt.debug.core.suspendPolicy" value="2"/>
<attrib name="org.eclipse.jdt.debug.ui.JAVA_ELEMENT_HANDLE_ID" value="=CCuenta/src&lt;
{CCuenta.java[CCuenta"/>
<attrib name="charEnd" value="1488"/>
<attrib name="org.eclipse.debug.core.enabled" value="true"/>
<attrib name="message" value="Line breakpoint:CCuenta [line: 38] - ingresar(double)" />
<attrib name="org.eclipse.jdt.debug.core.installCount" value="1"/>
<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="1595" lineNumber="41" type="org.eclipse.jdt.debug.javaLineBreakpointMarker">
<attrib name="charStart" value="1595"/>
<attrib name="org.eclipse.jdt.debug.core.suspendPolicy" value="2"/>
<attrib name="org.eclipse.jdt.debug.ui.JAVA_ELEMENT_HANDLE_ID" value="=CCuenta/src&lt;
{CCuenta.java[CCuenta"/>
<attrib name="charEnd" value="1620"/>
<attrib name="org.eclipse.debug.core.enabled" value="true"/>
<attrib name="message" value="Line breakpoint:CCuenta [line: 41] - ingresar(double)" />
<attrib name="org.eclipse.jdt.debug.core.installCount" value="1"/>
<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="1773" lineNumber="47" type="org.eclipse.jdt.debug.javaLineBreakpointMarker">
<attrib name="charStart" value="1773"/>
<attrib name="org.eclipse.jdt.debug.core.suspendPolicy" value="2"/>
<attrib name="org.eclipse.jdt.debug.ui.JAVA_ELEMENT_HANDLE_ID" value="=CCuenta/src&lt;
{CCuenta.java[CCuenta"/>
<attrib name="charEnd" value="1794"/>
<attrib name="org.eclipse.debug.core.enabled" value="true"/>
<attrib name="message" value="Line breakpoint:CCuenta [line: 47] - ingresar(double)"/>
<attrib name="org.eclipse.jdt.debug.core.installCount" value="1"/>
<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="1896" lineNumber="51" type="org.eclipse.jdt.debug.javaLineBreakpointMarker">
<attrib name="charStart" value="1896"/>
<attrib name="org.eclipse.jdt.debug.core.suspendPolicy" value="2"/>
<attrib name="org.eclipse.jdt.debug.ui.JAVA_ELEMENT_HANDLE_ID" value="=CCuenta/src&lt;
{CCuenta.java[CCuenta"/>
<attrib name="charEnd" value="1919"/>
<attrib name="org.eclipse.debug.core.enabled" value="true"/>
<attrib name="message" value="Line breakpoint:CCuenta [line: 51] - ingresar(double)"/>
<attrib name="org.eclipse.jdt.debug.core.installCount" value="1"/>
<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>

```