

TAREA PARA ED03
Jesus David Leon Da Trindade

ÍNDICE

● Introducción-----	1.
● Objetivos-----	2.
● Material empleado-----	2.
● Desarrollo-----	2-13
● Conclusiones-----	14

Introducción.

En esta tarea vamos a realizar las siguientes acciones:

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

Objetivos.

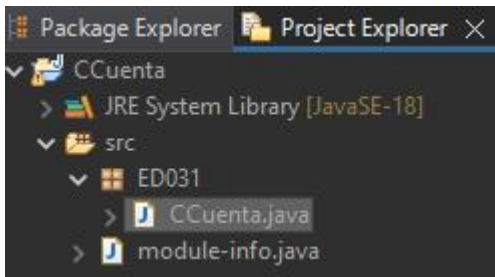
- Realizar varios análisis de un código de programación:
 - Análisis de caja blanca
 - Análisis de caja negra
 - Crear una clase para testear las pruebas de la caja blanca
 - Generar varios puntos de ruptura

Material.

- El IDE Eclipse, en su última versión
- El código de programación dado para dicha práctica (Método Main, Ingresar, Retirar
- Un Editor de texto (Google Docs)

Desarrollo.

Creamos el nuevo proyecto CCuenta.



Una vez creado procedemos a pegar todo el código proporcionado en la unidad.

```
1 package ED031;
2
3 public class CCuenta {
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 tencer 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 ya a ser probado con junit
28     */
29    public int ingresar(double cantidad)
30    {
31        int iCodErr;
32
33        if (cantidad < 0)
34        {
35            System.out.println("No se puede ingresar una cantidad negativa");
36            iCodErr = 1;
37        }
38        else if (cantidad == -3)
39        {
40            System.out.println("Error detectable en pruebas de caja blanca");
41            iCodErr = 2;
42        }
43        else
44        {
45            // Depuracion. Punto de parada. Solo en el 3 ingreso
46            dSaldo = dSaldo + cantidad;
47            iCodErr = 0;
48        }
49
50        // Depuracion. Punto de parada cuando la cantidad es menor de 0
51        return iCodErr;
52    }
53 }
```

Una vez tenemos el código en nuestro entorno de desarrollo que sería en este caso el Eclipse, procedemos a hacer un análisis de caja blanca que no es mas que un análisis o monitorización del código, estructura o diseño del software con la intención de mejorar la seguridad, eficiencia y la funcionalidad del mismo.

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

En el metodo ingresar podemos tener tres casos posibles a continuacion vamos a detallar en una prueba de caja blanca cada uno de ellos:

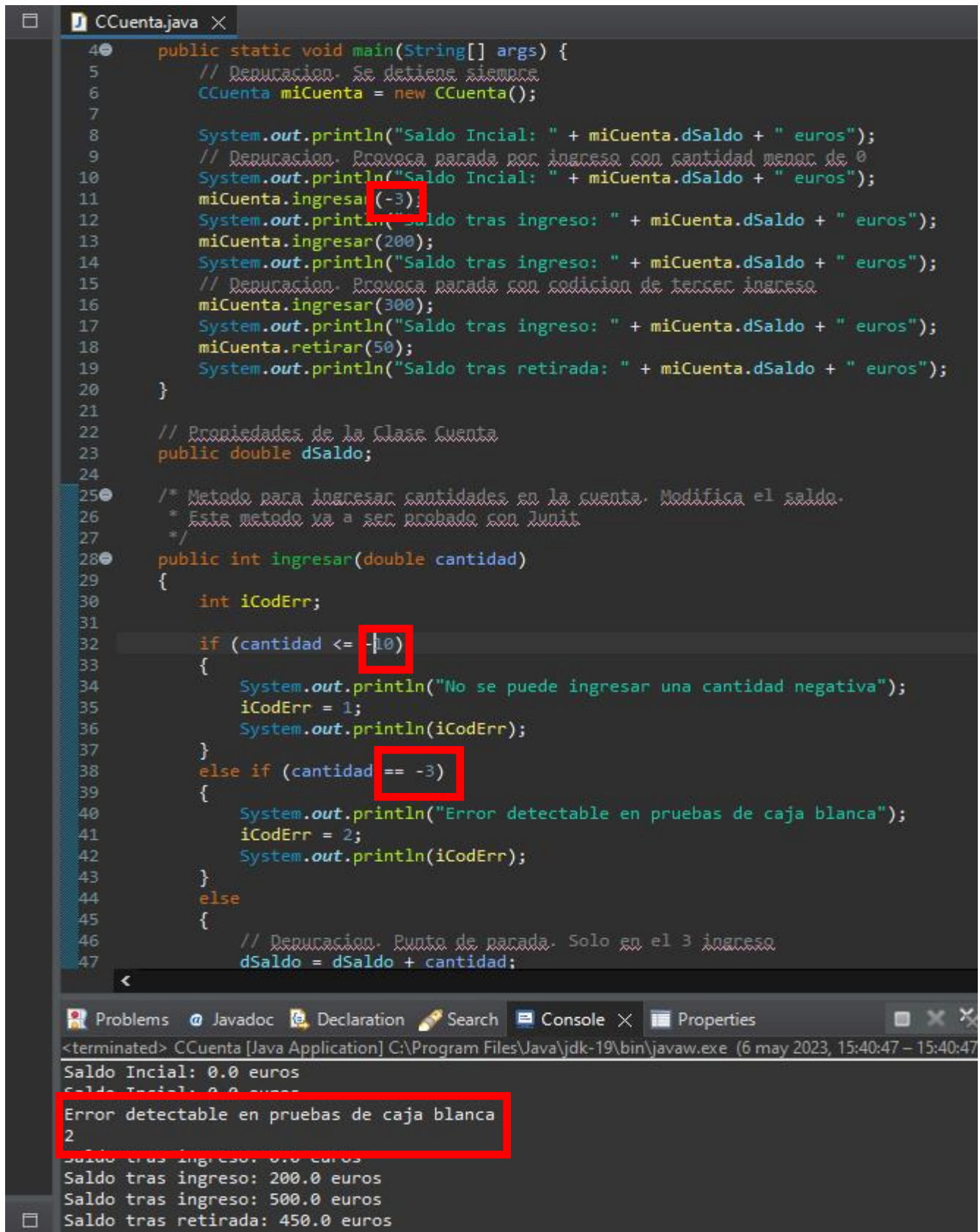
-En el caso que el valor que vamos a ingresar sea menor que 0, actuara el primer condicional del programa mostrando un mensaje por pantalla, indicando que no se puede ingresar un valor negativo y seteando a continuacion la variable iCodErr en 1.(Realice una pequeña modificacion al codigo para confirmar que la variable iCodErr establece el valor de 1.)

```
28 public int ingresar(double cantidad)
29 {
30     int iCodErr;
31
32     if (cantidad < 0)
33     {
34         System.out.println("No se pue
35         iCodErr = 1;
36         System.out.println(iCodErr);
37     }
38     else if (cantidad == -3)
39     {
40         System.out.println("Error det
41         iCodErr = 2;
42         System.out.println(iCodErr);
```

<terminated> CCuenta [Java Application] C:\Program Files\Ja

Saldo Inicial: 0.0 euros
Saldo Inicial: 0.0 euros
No se puede ingresar una cantidad negativa
1
Saldo tras ingreso: 0.0 euros
Saldo tras ingreso: 200.0 euros
Saldo tras ingreso: 500.0 euros
Saldo tras retirada: 450.0 euros

-En el siguiente caso -3 nos seguiría mostrando un mensaje en pantalla de no se puede ingresar una cantidad negativa. Para que esta condición se cumpla tendríamos que colocar un valor en la primera condición de -4 por ejemplo y en el valor a ingresar pondríamos -3 para que la condición se cumpla y nos muestre el mensaje por pantalla de (Error detectable en prueba de caja blanca).(Adjunto una foto probando la condición numero 2).



```
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    System.out.println("Saldo Inicial: " + miCuenta.dSaldo + " euros");
11    miCuenta.ingresar(-3);
12    System.out.println("Saldo tras ingreso: " + miCuenta.dSaldo + " euros");
13    miCuenta.ingresar(200);
14    System.out.println("Saldo tras ingreso: " + miCuenta.dSaldo + " euros");
15    // Depuracion. Provoca parada con codicion de tercer ingreso
16    miCuenta.ingresar(300);
17    System.out.println("Saldo tras ingreso: " + miCuenta.dSaldo + " euros");
18    miCuenta.retirar(50);
19    System.out.println("Saldo tras retirada: " + miCuenta.dSaldo + " euros");
20 }
21
22 // Propiedades de la Clase Cuenta
23 public double dSaldo;
24
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 <= -10)
33     {
34         System.out.println("No se puede ingresar una cantidad negativa");
35         iCodErr = 1;
36         System.out.println(iCodErr);
37     }
38     else if (cantidad == -3)
39     {
40         System.out.println("Error detectable en pruebas de caja blanca");
41         iCodErr = 2;
42         System.out.println(iCodErr);
43     }
44     else
45     {
46         // Depuracion. Punto de parada. Solo en el 3 ingreso
47         dSaldo = dSaldo + cantidad;
48     }
49 }
```

Problems Javadoc Declaration Search Console Properties

<terminated> CCuenta [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe (6 may 2023, 15:40:47 - 15:40:47)

Saldo Inicial: 0.0 euros
Saldo Inicial: 0.0 euros
Error detectable en pruebas de caja blanca
2
Saldo tras ingreso: 0.0 euros
Saldo tras ingreso: 200.0 euros
Saldo tras ingreso: 500.0 euros
Saldo tras retirada: 450.0 euros

Sin estas modificaciones el error no sería detectable. La variable En el caso 3 si el saldo es positivo se suma, y la variable iCodErr estable es valor en 0. (Adjunto una foto con la demostración).

```
27  */
28  public int ingresar(double cantidad)
29  {
30      int iCodErr;
31
32      if (cantidad <= -10)
33      {
34          System.out.println("No se puede ingresar una cantidad negativa");
35          iCodErr = 1;
36          System.out.println(iCodErr);
37      }
38      else if (cantidad == -3)
39      {
40          System.out.println("Error detectable en pruebas de caja blanca");
41          iCodErr = 2;
42          System.out.println(iCodErr);
43      }
44      else
45      {
46          // Depuracion. Punto de parada. Solo en el 3 ingreso
47          dSaldo = dSaldo + cantidad;
48          iCodErr = 0;
49      }
50  }
51  System.out.println(iCodErr);
52  // Depuracion. Punto de parada cuando la cantidad es menor de 0
53  return iCodErr;
54
```

Problems Javadoc Declaration Search Console Properties

<terminated> CCuenta [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe (6 may 2023, 17:47:01 - 1

Saldo Inicial: 0.0 euros
Saldo Inicial: 0.0 euros
0
Saldo tras ingreso: 10.0 euros
0
Saldo tras ingreso: 210.0 euros
0
Saldo tras ingreso: 510.0 euros
Saldo tras retirada: 460.0 euros

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.

Estableciendo un análisis de caja negra aplicando la lógica de programación vamos a establecer los siguientes puntos.

- Numero positivo, mayor al saldo actual: No podríamos retirar la cantidad indicada.

- Numero positivo, siendo menor que el saldo actual: Podríamos retirar el saldo que introducimos.

- Numero positivo igual al saldo actual:Podríamos retirar la cantidad.

- Numero 0 no haría nada por que le estaríamos indicando que vamos a retirar 0.

- Numero negativo:No podríamos retirar un numero negativo de una cuanta corriente.

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.

The screenshot shows the 'New JUnit Test Case' dialog box in Eclipse. The title bar says 'New JUnit Test Case'. Inside, there's a warning icon and text: 'The use of the default package is discouraged.' Below this, there are three radio buttons: 'New JUnit 3 test' (selected), 'New JUnit 4 test', and 'New JUnit Jupiter test'. There are two 'Browse...' buttons for 'Source folder' (set to 'CCuenta/src') and 'Package' (set to '(default)'). The 'Name' field is 'CCuentaTest' and the 'Superclass' is 'java.lang.Object'. Under 'Which method stubs would you like to create?', there are checkboxes for '@BeforeAll setUpBeforeClass()', '@AfterAll tearDownAfterClass()', '@BeforeEach setUp()', '@AfterEach tearDown()', and 'constructor'. The 'Do you want to add comments?' section has a 'Generate comments' checkbox. At the bottom, the 'Class under test' is 'ED031.CCuenta'. Navigation buttons at the bottom include '?', '< Back', 'Next >', 'Finish', and 'Cancel'.

```
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 import ED031.CCuenta;
8
9 class CCuentaTest {
10
11     CCuenta miCuenta = new CCuenta();
12
13     @ParameterizedTest
14     @CsvSource({"-10,1","-3,2","10,0"})
15     @DisplayName("Caja Blanca - Ingresar")
16     void testIngreso(double cant,int resul) {
17         assertEquals(resul,miCuenta.ingresar(cant));
18     }
19 }
```

<

Problems Javadoc Declaration Search Console X

<terminated> CCuentaTest [JUnit] C:\Program Files\Java\jdk-19\bin\javaw.exe

No se puede ingresar una cantidad negativa
1
1
Error detectable en pruebas de caja blanca
2
2
0

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

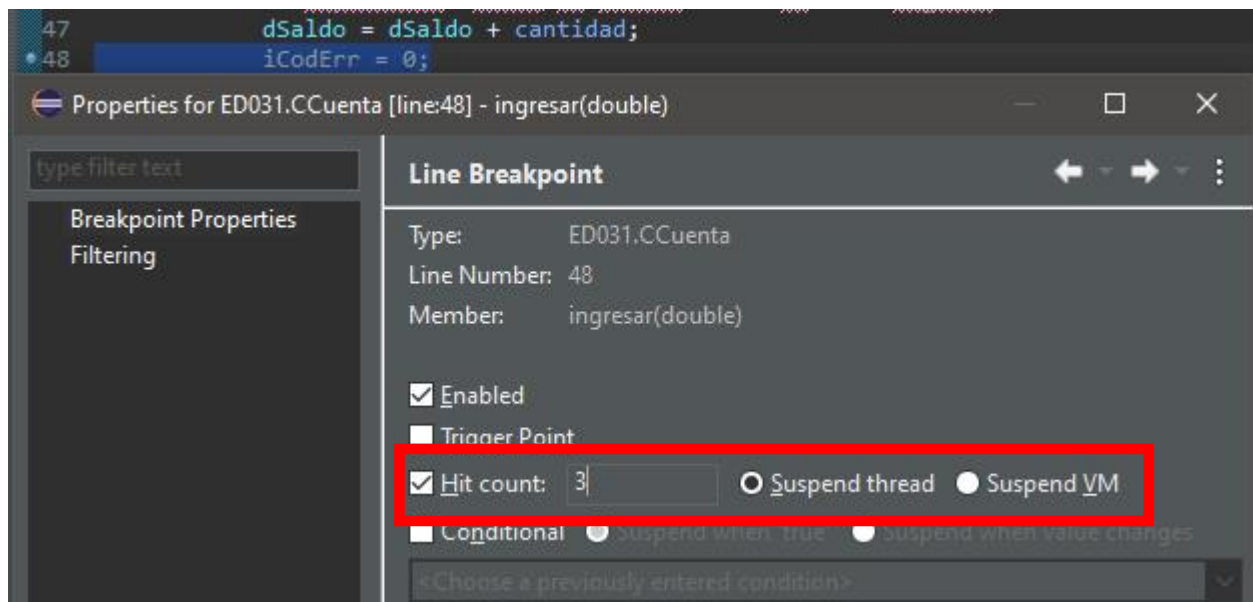
Punto de parada sin condición al crear el objeto miCuenta en la función main. Línea 3 del código del método main que se presenta en la siguiente página de este libro.

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

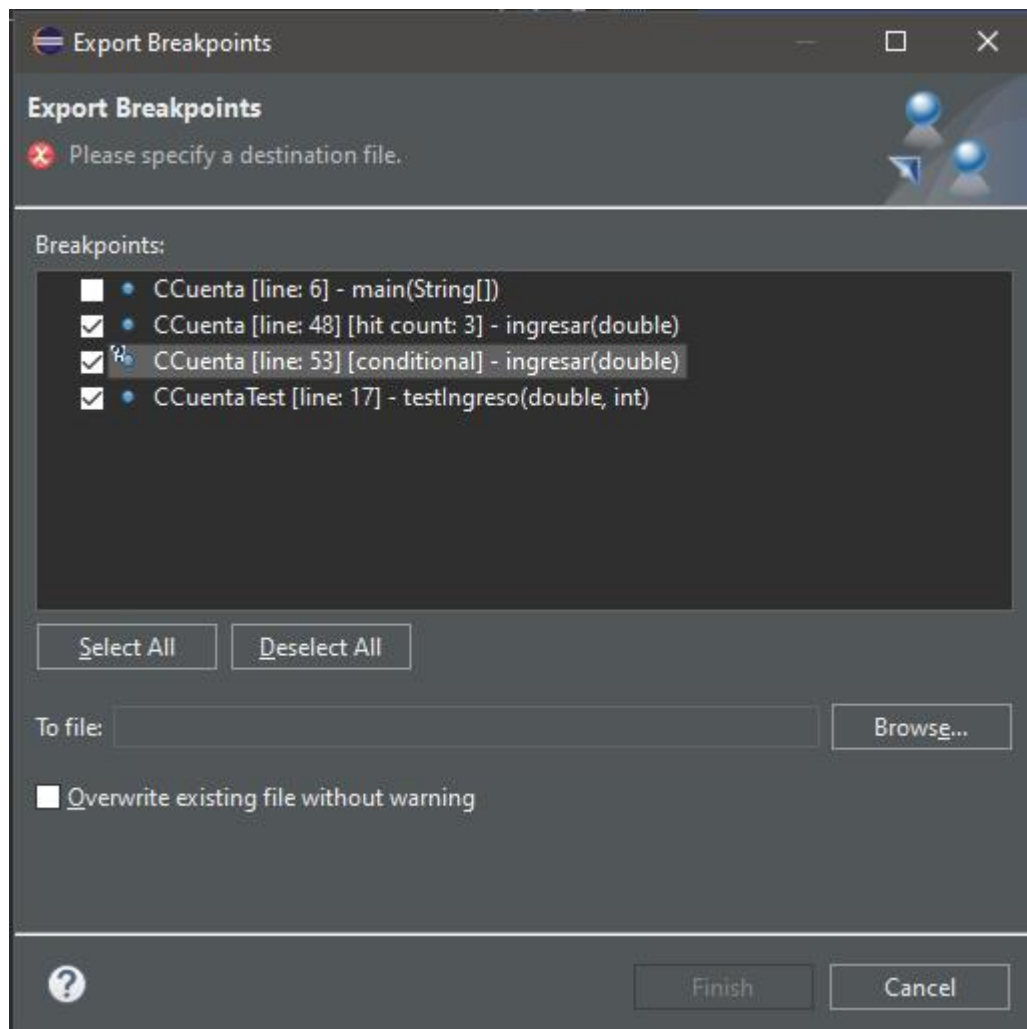
Punto de parada en la instrucción return del método ingresar sólo si la cantidad a ingresar es menor de 0. Línea 20 del código del método ingresar que se presenta más adelante.

```
43     }
44     else
45     {
46         // Depuración. Punto de parada. Solo en el 3 ingreso
47         dSaldo = dSaldo + cantidad;
48         iCodErr = 0;
49     }
50 }
```

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



El siguiente paso es exportar los breakpoints.




```
<?xml version="1.0" encoding="UTF-8"?>
<breakpoints>
<breakpoint enabled="true" persistent="true" registered="true">
<resource path="/CCuenta/src/ED031/CCuenta.java" type="1"/>
<marker charStart="125" lineNumber="6" type="org.eclipse.jdt.debug.javaLineBreakpointMarker">
<attrib name="charStart" value="125"/>
<attrib name="org.eclipse.jdt.debug.core.suspendPolicy" value="2"/>
<attrib name="org.eclipse.jdt.debug.ui.JAVA_ELEMENT_HANDLE_ID" value="/CCuenta/src/ED031/CCuenta.java"/>
<attrib name="charEnd" value="163"/>
<attrib name="org.eclipse.debug.core.enabled" value="true"/>
<attrib name="message" value="Line breakpoint:CCuenta [line: 6] - main(String[])" />
<attrib name="org.eclipse.debug.core.id" value="org.eclipse.jdt.debug"/>
<attrib name="org.eclipse.jdt.debug.core.typeName" value="ED031.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/ED031/CCuenta.java" type="1"/>
<marker charStart="1701" lineNumber="48" type="org.eclipse.jdt.debug.javaLineBreakpointMarker">
<attrib name="charStart" value="1701"/>
<attrib name="org.eclipse.jdt.debug.core.suspendPolicy" value="2"/>
<attrib name="org.eclipse.jdt.debug.ui.JAVA_ELEMENT_HANDLE_ID" value="/CCuenta/src/ED031/CCuenta.java"/>
<attrib name="org.eclipse.jdt.debug.core.hitCount" value="3"/>
<attrib name="charEnd" value="1722"/>
<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: 48] [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="ED031.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/ED031/CCuenta.java" type="1"/>
<marker charStart="1863" lineNumber="53" type="org.eclipse.jdt.debug.javaLineBreakpointMarker">
<attrib name="org.eclipse.jdt.debug.core.conditionEnabled" value="true"/>
<attrib name="charStart" value="1863"/>
<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="/CCuenta/src/ED031/CCuenta.java"/>
<attrib name="charEnd" value="1886"/>
</marker>
</breakpoint>
```

Conclusion.

La conclusión que saco personalmente de la tarea a sido interesante por un lado, con los análisis de caja blanca y negra, viendo como podríamos hacer para que nuestro código sea mas limpio, ordenado y eficiente.

Por otro lado me ha resultado muy interesante el tema de los breakpoint para ir ejecutando el código por partes, y de esta manera poder localizar los errores mucho mas fácil y rápido.