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 retirar 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 Incial: " + miCuenta.dSaldo + " euros");
        // Depuracion. Provoca parada por ingreso con cantidad menor de 0
        miCuenta.ingresar(-100);
        System.out.println("Saldo Incial: " + 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
        {

        }
    }
```

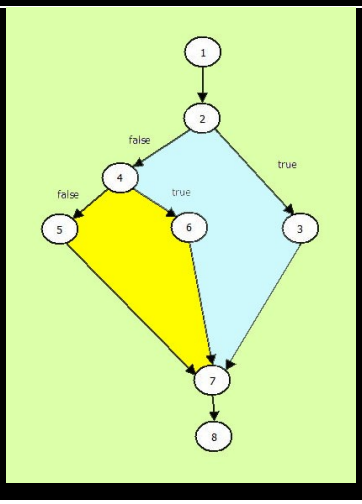
Deberás realizar un documento donde dar respuesta a los siguientes apartados:

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

```
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      }
```

Creación del Grafo

Ya en la creación del grafo se puede observar que la condición contiene un error en uno de sus elementos, pues nunca llegará a entrar en las líneas 39-40.

Grafo	Nodos	Líneas de código
	1	30. int iCoderr;.
	2	32. Cond. cantidad < 0.
	3	34 - 35.
	4	42. Cond. cantidad ==-3.
	5	45-46.
	6	39-40.
	7	50. Fin de toda condición.
	8	52 – 53.

Complejidad de McCabe o ciclomática

Método de cálculo	Complejidad	Comentarios
Nº de regiones	3	Región amarilla y región verde (externa).
Nº de aristas – Nº de nodos + 2	9 – 8 + 2 = 3	
Nº de condiciones + 1	2 + 1 = 3	Nodo 2 y nodo 4.

Caminos de prueba

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

Casos de uso, resultados esperados y análisis

Caminos / Casos de uso	Datos			Salidas
	cantidad	dSaldo	iCodErr	
1	-100	0	1	1
2	200	dSaldo+200	0	0
3	*-3*	*0*	*2*	*2*

Conclusión

Realizando el análisis de Caja Blanca, aunque a simple vista es fácil de determinar, en los casos de uso se puede observar que el camino 3 nunca se llega a realizar. Se trata de las instrucciones integradas en: "*else if (cantidad == -3)".* El elemento de la condición anterior (cantidad < 0) abarca el caso en el que se ingresa -3 en la variable cantidad.

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.

Clases de equivalencia

Condición de entrada	**Clases de equivalencia**	**Clases válidas**	**COD**	**Clases no válidas**	**COD**
Cantidad de saldo a retirar	Número finito y consecutivo de valores	Un dato mayor que 0.	C1B1	Un dato menor que 0	C1E1
		Un dato menor que o igual a Saldo	C1B2	Un dato mayor que Saldo	C1E2

Conjetura de errores

- **Valores no numéricos:** habría que considerar la posibilidad de implementar en el programa la posibilidad de evitar que el valor de Cantidad, que será un dato introducido por el usuario, sea un dato no numérico.

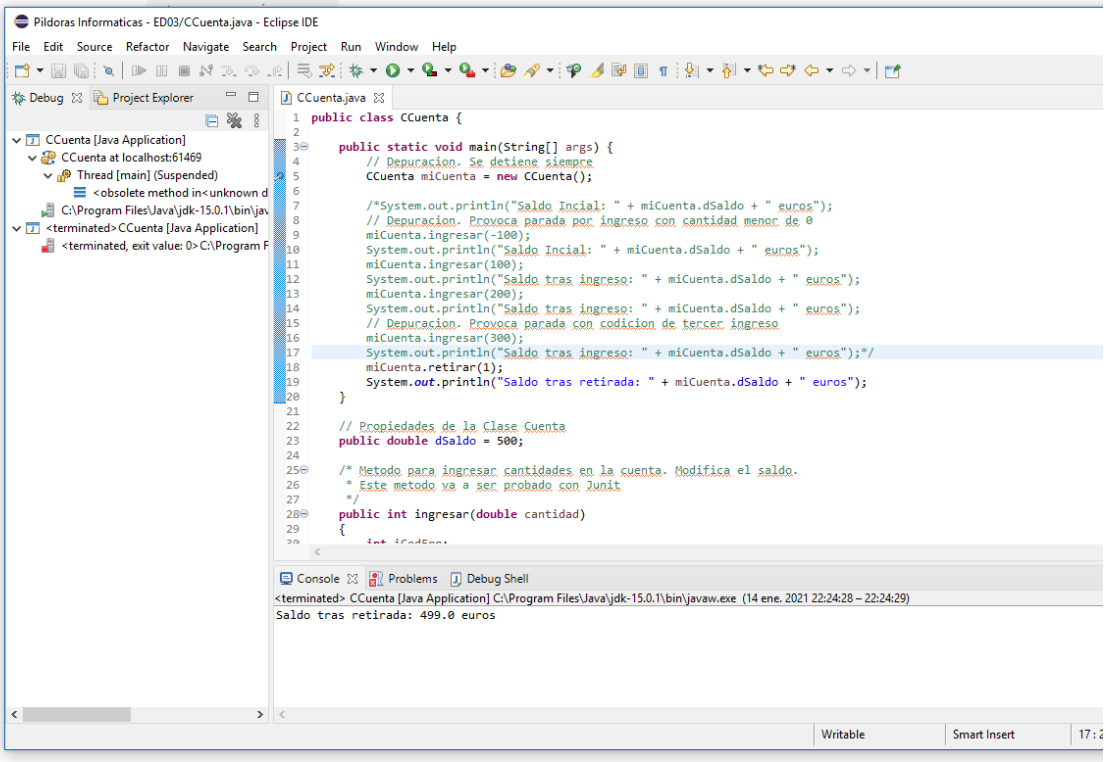
Valores límite

Casos de prueba	**Clases de equivalencia**	**Condiciones de entrada**	**Límite de aprobación máxima**	**Resultado esperado**
		Cantidad	**Saldo**	
CP1	C1B1	1	500	Restar la cantidad a retirar al saldo actual
CP2	C1B2	500	500	Restar la cantidad al saldo actual
CP3	C1E1	0	500	Mensaje de error
CP4	C1E1	-1	500	Mensaje de error
CP5	C1E2	501	500	Mensaje de error
CP5	**C1E3**	"Hola"	N (indiferente)	Mensaje de error
CP6	C1B1, C1B2	0	0	Mensaje de error

He considerado seis casos de prueba: CP1, CP2… CP6., donde la aprobación límite de datos viene dada por la variable Saldo (saldo disponible del cliente que va a retirar dinero), a la cual le he dado el valor de 500. Para comprobar que los resultados esperados coinciden con la ejecución del programa, asigno a la variable saldo el valor de 500, comento las líneas de código que no necesito y dejo solo las que ejecutan el método retirar.

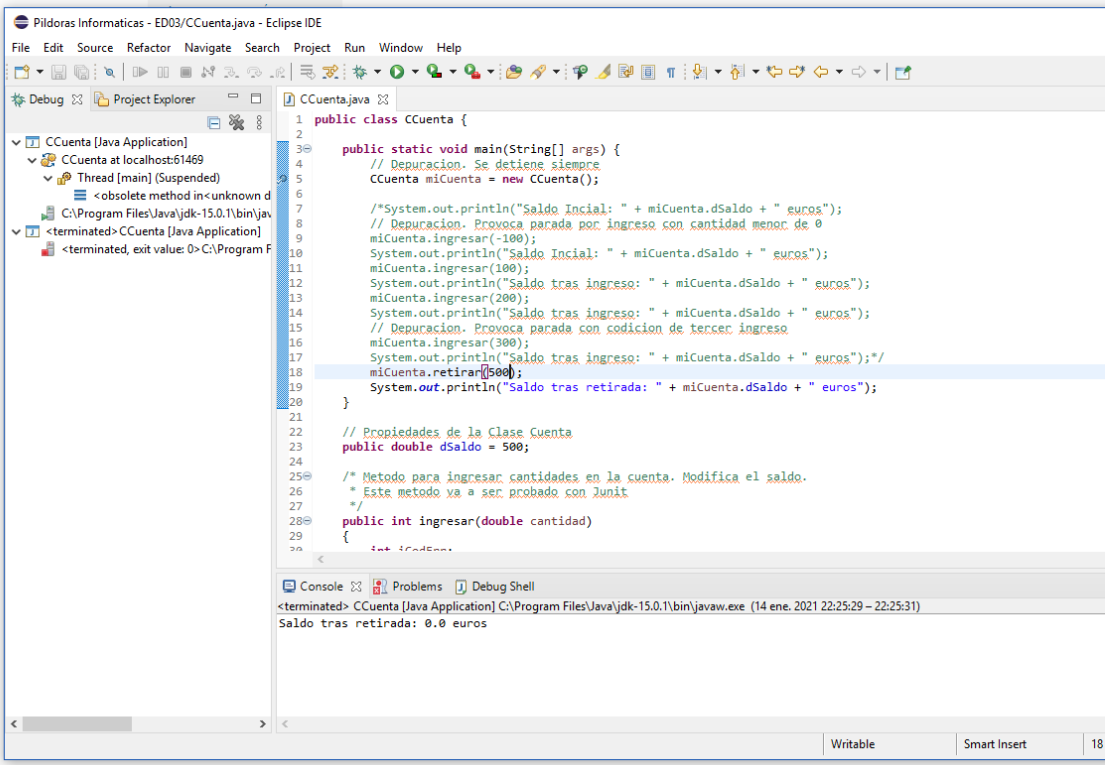
- CP1. Cantidad es igual a 1 y el saldo disponible es igual a 500. Resultado esperado: realizar la operación de retirada de dinero, restando el dinero al saldo disponible. Saldo = 499.

XERACH ERNESTO CASANOVA CABRERA

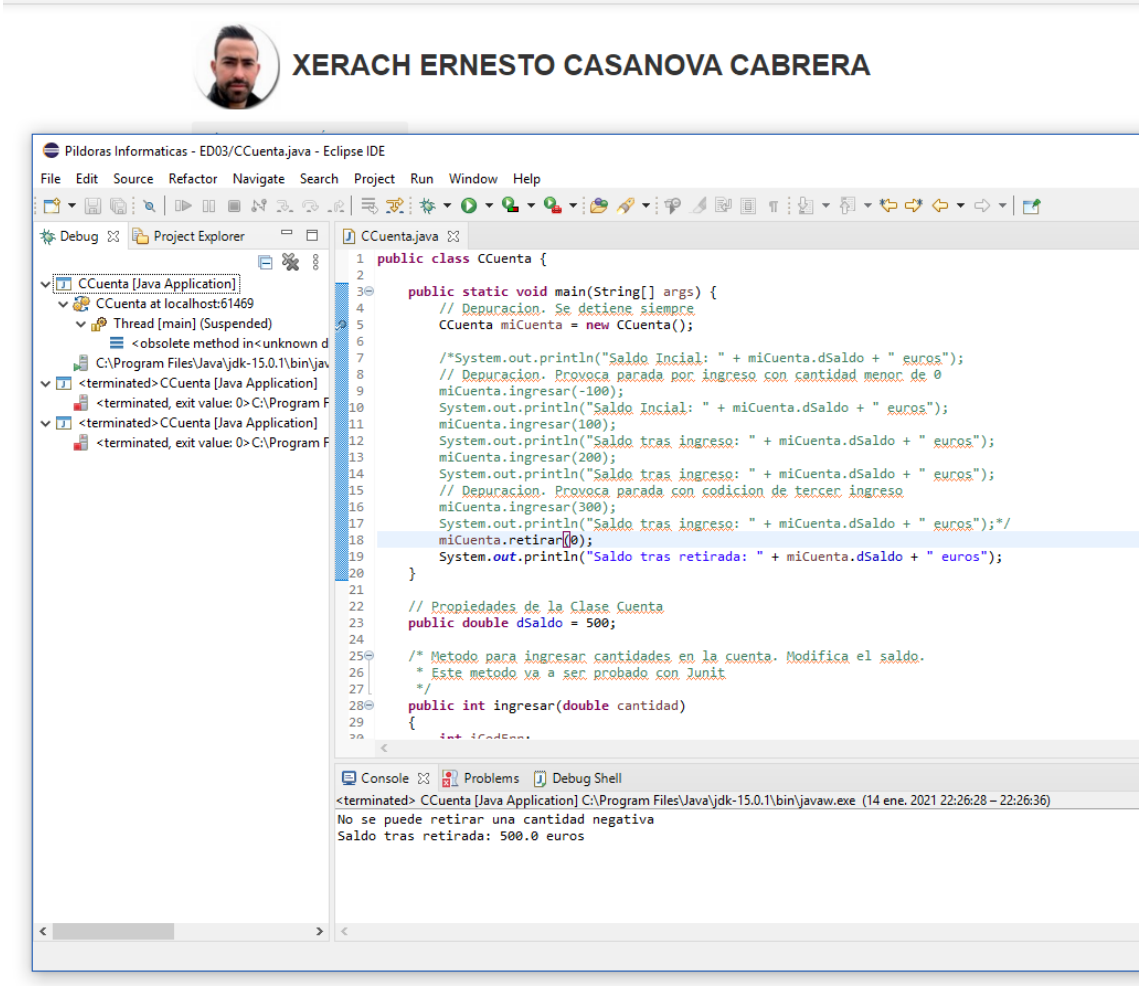


- CP2. Cantidad es igual a 500 y el saldo disponible es igual a 500. Resultado esperado: realizar la operación de retirada de dinero, restando el dinero al saldo disponible. Saldo = 0.

XERACH ERNESTO CASANOVA CABRERA

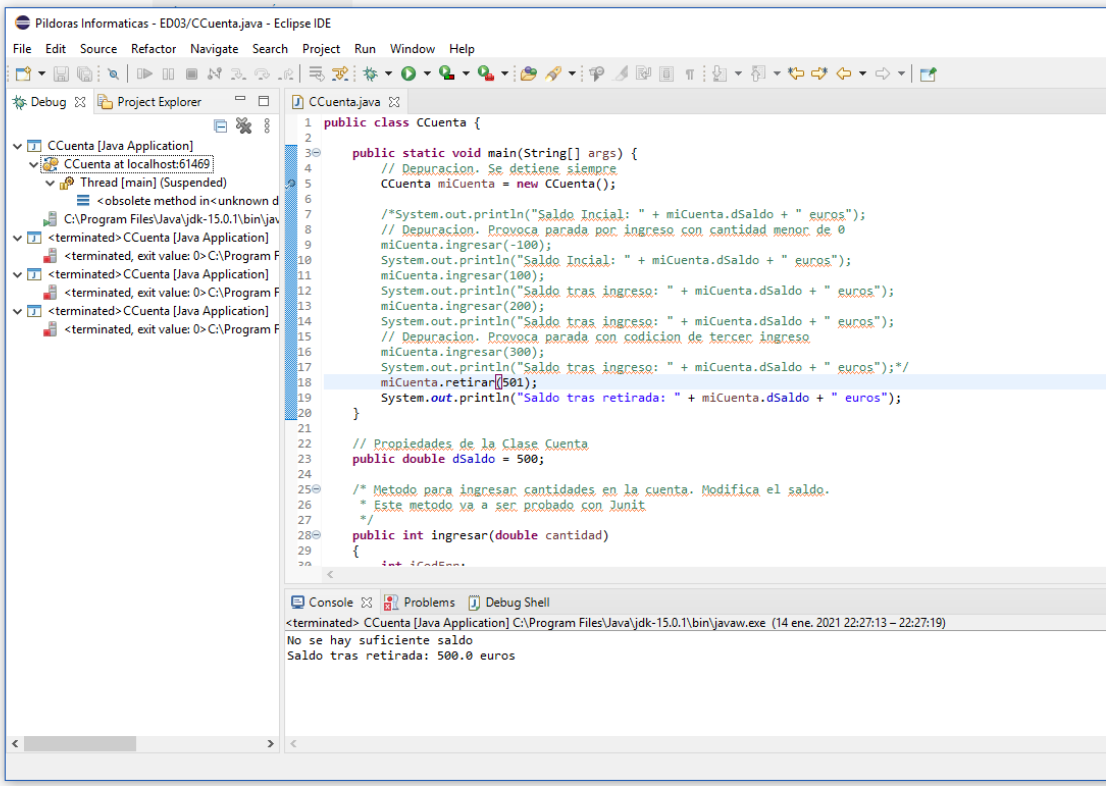


- CP3. Cantidad es igual a cero y el saldo disponible es igual a 500. Resultado esperado: mensaje de error. No se pueden retirar cantidades iguales o menores de cero.



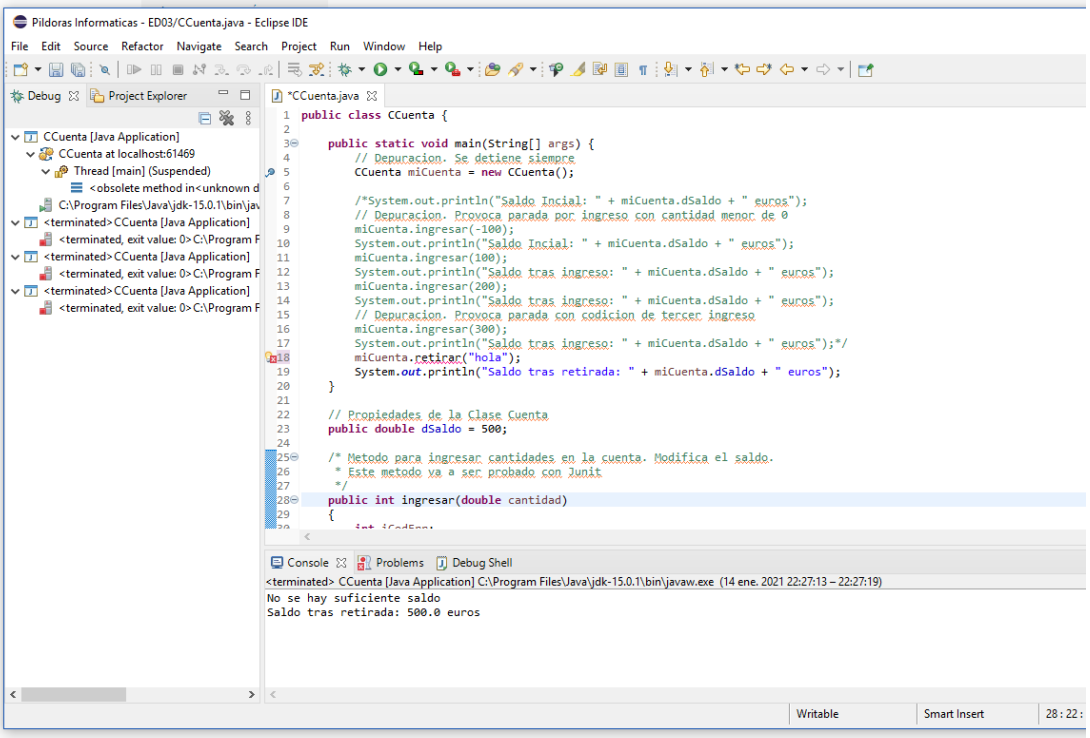
- CP4. Cantidad es igual a 501 y el saldo disponible es igual a 500. Resultado esperado: mensaje de error. No se pueden retirar cantidades mayores del saldo disponible.

XERACH ERNESTO CASANOVA CABRERA



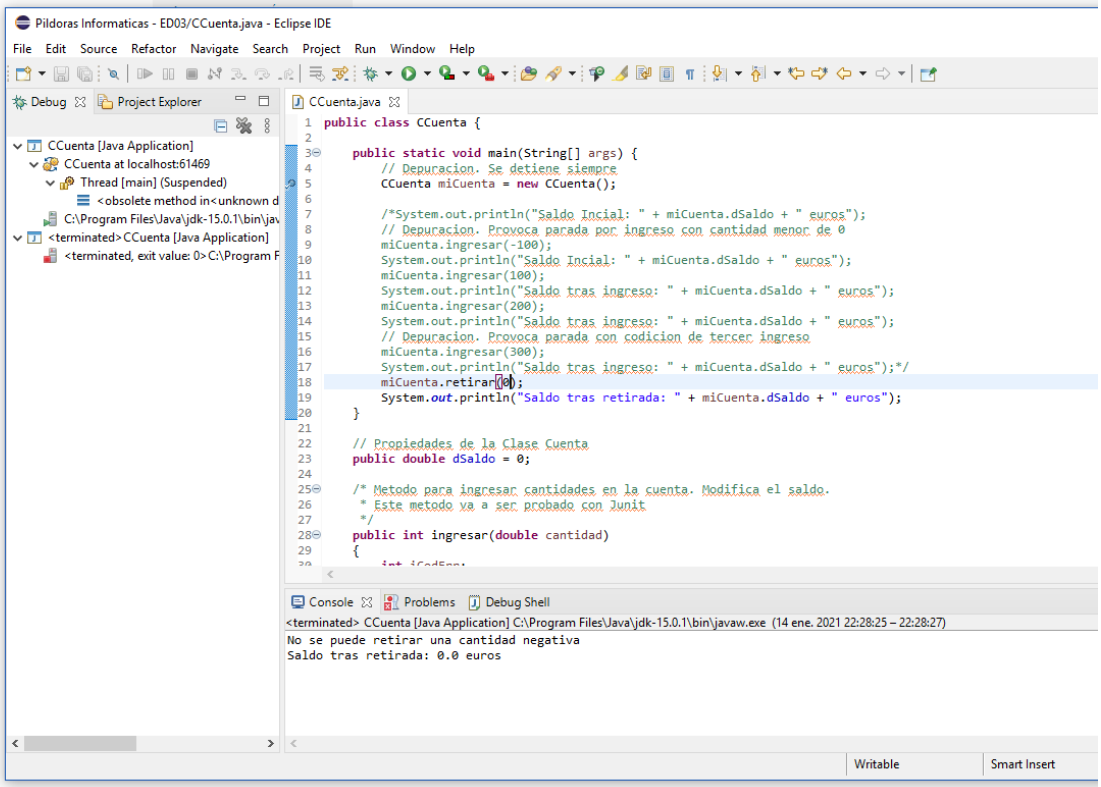
- CP5. Cantidad es igual a la cadena de caracteres “Hola”: mensaje de error. Valor introducido incorrecto. En este caso el programa no deja compilar, pero podría darse el caso, según la implementación del programa, que el usuario pudiese insertar caracteres en la cantidad a retirar y en este caso se diera un error en tiempo de ejecución del programa.

XERACH ERNESTO CASANOVA CABRERA



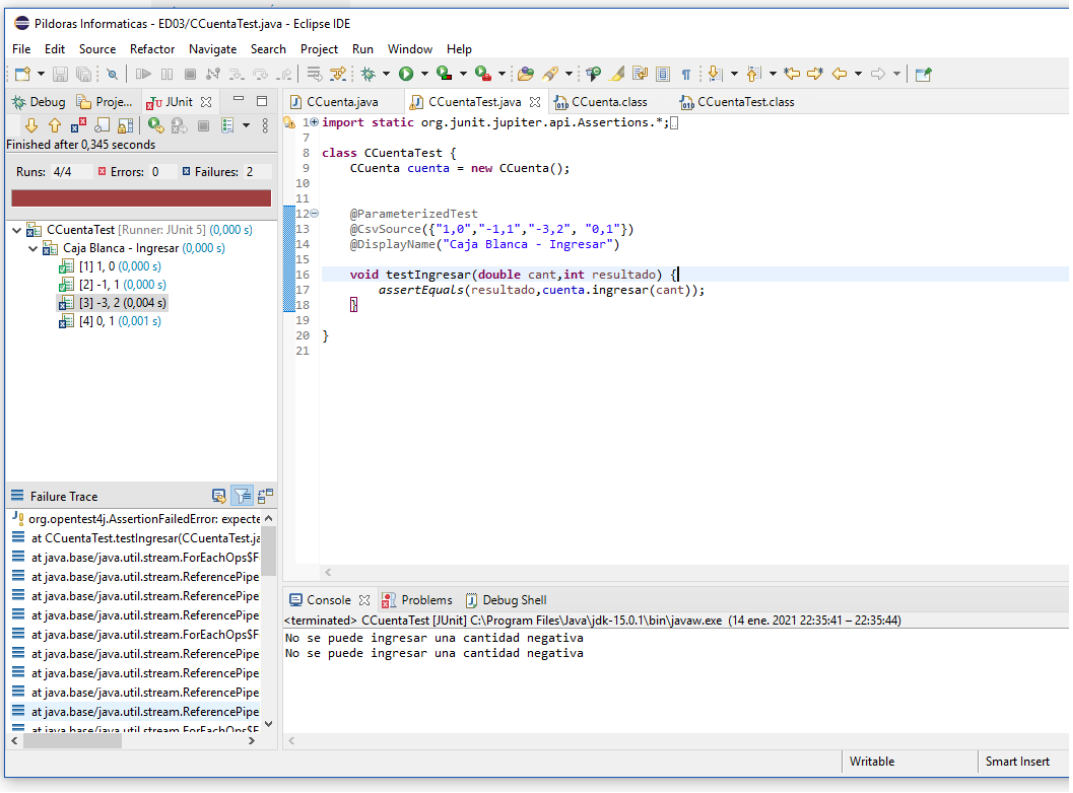
- CP6. Cantidad es igual a 0 y saldo es igual a 0. Mensaje de error. No se pueden retirar cantidades menores o iguales de cero (se obvia, por tanto, el hecho de que en la cuenta no haya saldo disponible).

XERACH ERNESTO CASANOVA CABRERA



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.

XERACH ERNESTO CASANOVA CABRERA

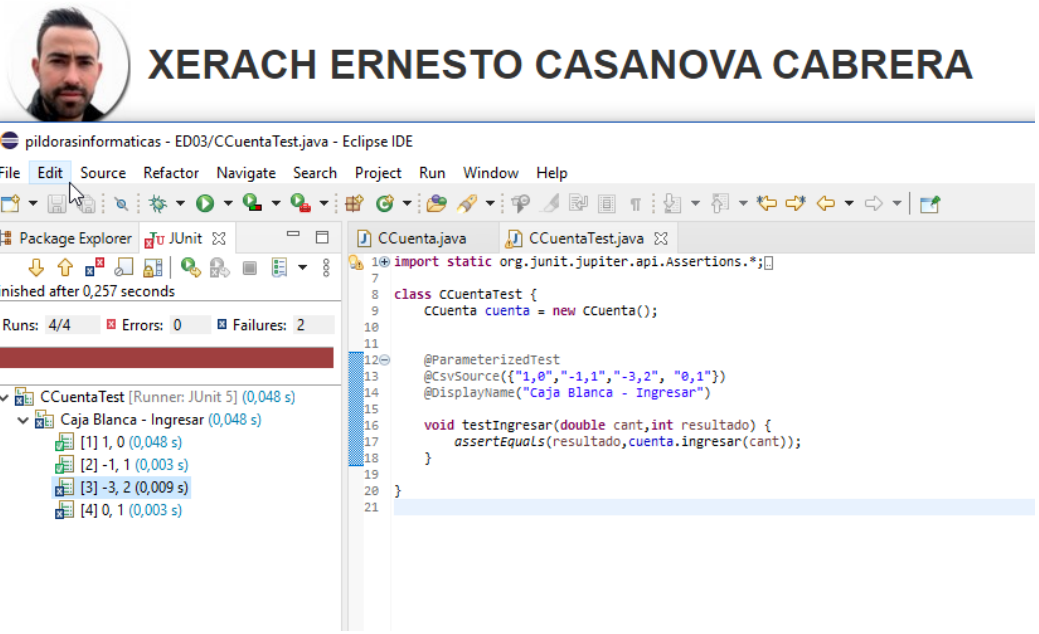


Código:

```
import static org.junit.jupiter.api.Assertions.*;


class CCuentaTest {
	CCuenta cuenta = new CCuenta();


	@ParameterizedTest
	@CsvSource({"1,0","-1,1","-3,2", "0,1"})
	@DisplayName("Caja Blanca - Ingresar")

	void testIngresar(double cant,int resultado) {
		assertEquals(resultado,cuenta.ingresar(cant));
	}

}
```

He creado la clase CCuentaTest en Junit con el código facilitado.

El código crea una nueva clase CCuenta llamada cuenta para hacer pruebas con parámetros, esos parámetros se definen con #CsvSource y pertenecen, en cada prueba, al valor que se devuelve y el valor que se pasa por parámetro. El método testIngresar es el encargado de usar esos parámetros para realizar las pruebas.



He generado 4 casos distintos en los que 2 de ellos dan resultados erróneos que habría que corregir:

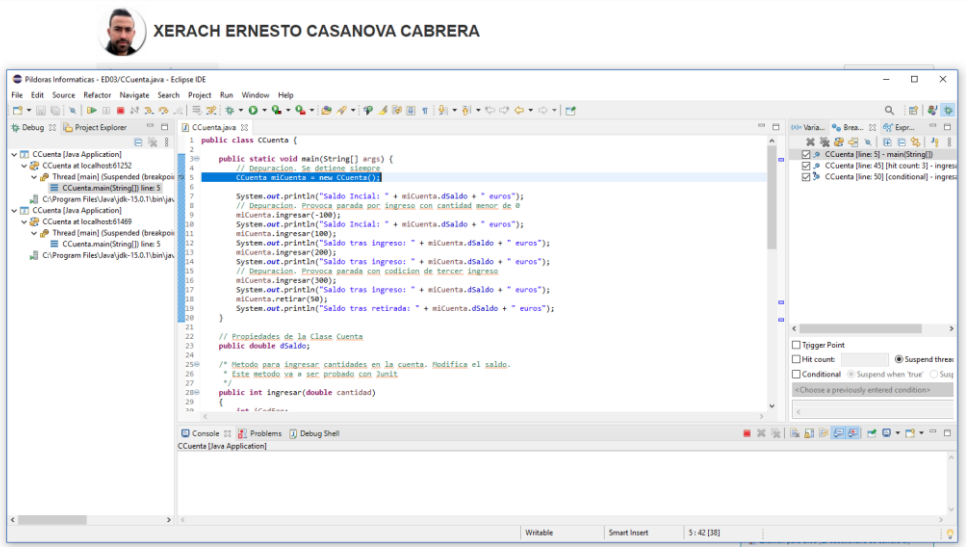
1. Ingresando la cantidad de -3, no devuelve un 2 en la variable iCodErr. Esto es debido a que nunca entra en el elemento de la condición. Siempre que el valor cantidad introducido sea menor de cero, iCodErr será igual a 1. Por tanto, todo el elemento de la condición en la que nunca se llega a entrar, debe eliminarse.

2. El programa no debería dejar ingresar una cantidad de 0, entonces iCodErr debería devolver 1 sin embargo. iCodErr devuelve 0.

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

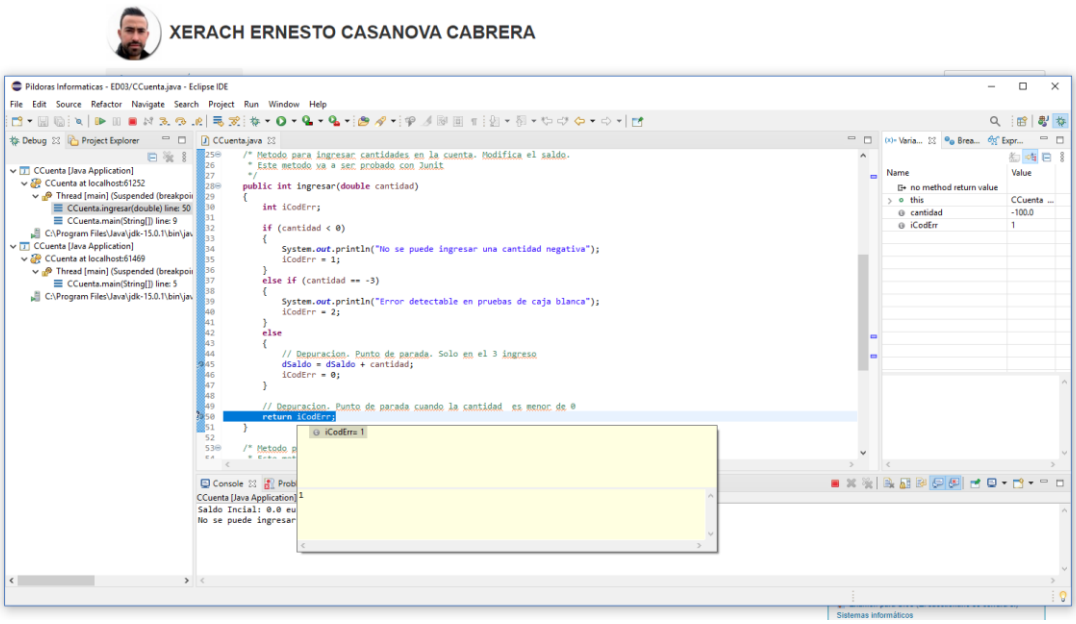
Punto de parada sin condición al crear el objeto miCuenta en la función main. Línea 3 del código del método main que se presenta en la siguiente página de este libro.

Al ejecutar la aplicación en modo depuración, el programa se detiene en el primer punto de ruptura, el cual, al no llevar ninguna condición, detiene la aplicación justo en ese punto cada vez que pase por esa línea de código.



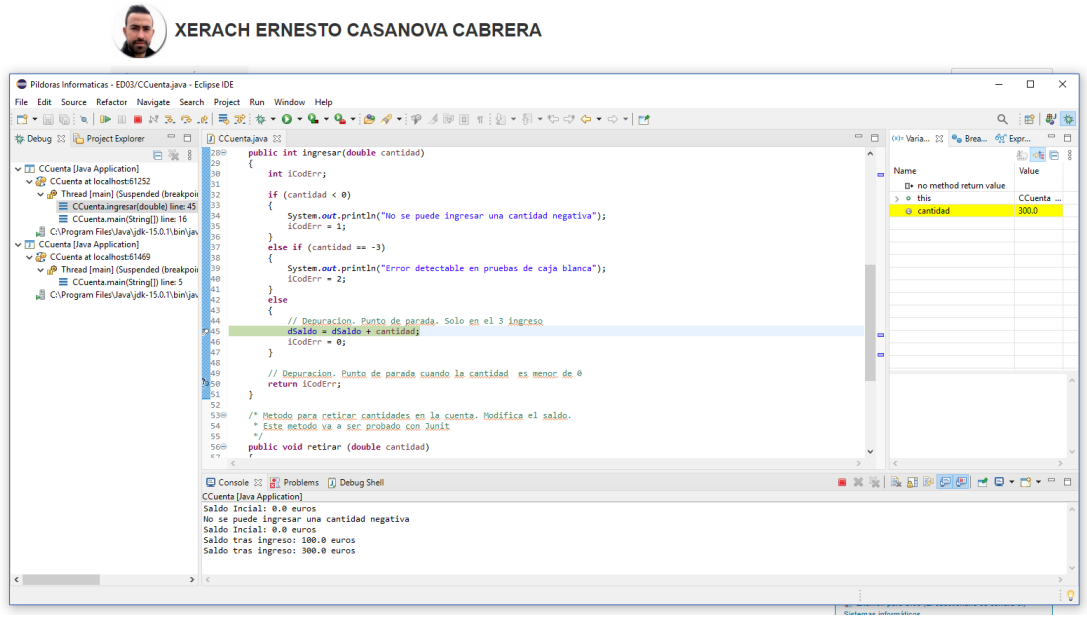
Punto de parada en la instrucción return del método ingresar sólo si la cantidad a ingresar es menor de 0. Línea 20 del código del método ingresar que se presenta más adelante.

En esta ocasión, el punto de ruptura se detiene solo si la variable cantidad es menor que cero, en la ventana de las variales observamos que cantidad tiene un valor de -100 y por tando, iCodErr devuelve 1.



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.

Por último, podemos observar como en el tercer punto de ruptura, se detiene el programa cuando ha pasado por esa línea de código 3 veces. Como vemos en la consola, se ha impreso por pantalla el valor de saldo dos veces y en el momento en que se ha detenido el programa, se va a actualizar dsaldo con 300 euros más.



Pulsando el botón derecho sobre la ventana de puntos de ruptura selecciona la opción "Exportar puntos de ruptura". Seleccionas los tres puntos de ruptura generados y guardas el fichero. El fichero tendrá la extensión bkpt, la cambias por txt. Ahora abres el fichero y copias el contenido íntegramente al documento.

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

<breakpoint enabled="true" persistant="true" registered="true">
<resource path="/ED03/CCuenta.java" type="1"/>
<marker charStart="113" lineNumber="5" type="org.eclipse.jdt.debug.javaLineBreakpointMarker">
<attrib name="charStart" value="113"/>
<attrib name="org.eclipse.jdt.debug.core.suspendPolicy" value="2"/>
<attrib name="org.eclipse.jdt.debug.ui.JAVA_ELEMENT_HANDLE_ID"
value="=ED03/<{CCuenta.java[CCuenta"/>
<attrib name="charEnd" value="151"/>
<attrib name="org.eclipse.debug.core.enabled" value="true"/>
<attrib name="message" value="Line breakpoint:CCuenta [line: 5] - 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" persistant="true" registered="true">
<resource path="/ED03/CCuenta.java" type="1"/>
<marker charStart="1600" lineNumber="45" type="org.eclipse.jdt.debug.javaLineBreakpointMarker">
<attrib name="charStart" value="1600"/>
<attrib name="org.eclipse.jdt.debug.core.suspendPolicy" value="2"/>
<attrib name="org.eclipse.jdt.debug.ui.JAVA_ELEMENT_HANDLE_ID"
value="=ED03/<{CCuenta.java[CCuenta"/>
<attrib name="org.eclipse.jdt.debug.core.hitCount" value="3"/>
<attrib name="charEnd" value="1636"/>
<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.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" persistant="true" registered="true">
<resource path="/ED03/CCuenta.java" type="1"/>
<marker charStart="1761" lineNumber="50" type="org.eclipse.jdt.debug.javaLineBreakpointMarker">
<attrib name="org.eclipse.jdt.debug.core.conditionEnabled" value="true"/>
<attrib name="charStart" value="1761"/>
<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="=ED03/<{CCuenta.java[CCuenta"/>
<attrib name="charEnd" value="1784"/>
<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.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>