Solución de la tarea para ED03.

Diseño y realización de pruebas

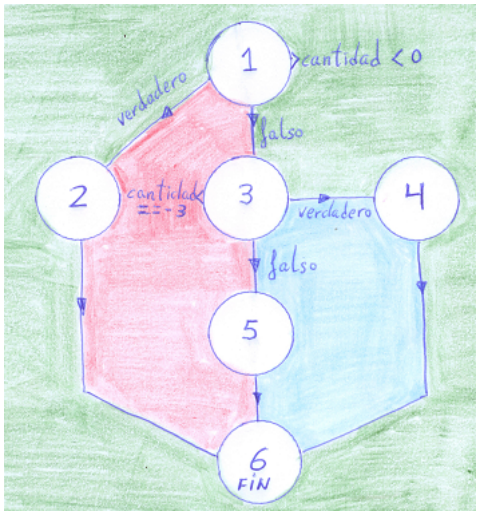
Para dar solución a la tarea, se ha incluido el fichero comprimido, Tarea-sw.rar, donde aparece el código fuente de la aplicación que tienes que probar. Pulsa aquí para que te puedas descargar dicho código fuente.

Para ejecutar la solución procederemos de la siguiente forma:

1. Se descomprime el fichero Tarea-sw.rar en la carpeta de proyectos de NetBeans "NetBeansProjects".
2. Ejecutamos NetBeans, y accedemos al menú Archivo - Abrir Proyecto - PrácticaUnidad3.
3. Ejecutamos la aplicación, comprobando que se genera sin ningún error.

Caja blanca. Método ingresar.

Creación del grafo.

Grafo	Nodo	Línea - Condición
	1	Lin: 3. Cond. < 0
	2	Lin: 5-6
	3	Lin: 8. Cond. == - 3
	4	Lin: 10-11
	5	Lin: 16-17
	6	Lin: 20

Nota: podría haberse considerado un nodo más correspondiente a la definición de la variable iCodErr encima del nodo 1. Este cambio no introduce modificación alguna en las pruebas ya que la complejidad ciclomática es la misma y no introduce posibles bifurcaciones en la ejecución del código (caminos alternativos).

Complejidad ciclomática.

Calculada por los tres métodos posibles. Todos ellos deben dar el mismo resultado.

Método de cálculo	Complejidad	Comentarios
Nº de regiones	3	Hay que considerar la región exterior.
Nº de aristas - Nº nodos + 2	7 - 6 + 2 = 3	
Nº de condiciones + 1	2 + 1 = 3	Nodos 1, 3

Caminos de prueba.

Su número debe ser igual a la complejidad calculada. Cada nuevo camino deberá aportar el paso por nuevas aristas/nodos del grafo. La definición de caminos se hará desde los más sencillos a los más complicados.

- **Camino 1**: **1 - 2 - 6**
- **Camino 2**: 1 - **3 - 4** - 6
- **Camino 3:** 1 - 3 - **5** - 6

Casos de uso.

El único dato de entrada en el método es la cantidad a ingresar:

Camino / Caso uso	Cantidad
1	- 10
2	- 3
3	10

Resultados esperados.

Camino / Caso uso	Resultado esperado
1	Mensaje: "No se puede ingresar una cantidad negativa". Devuelve código de error 1.
2	Mensaje: "Error detectable en pruebas de caja blanca". Devuelve código de error 2.
3	Incrementa el valor de la variable saldo la cantidad indicada. Devuelve código 0, sin error.

Nota: en las pruebas reales del código, se comprobará que para el caso de prueba 2, el resultado obtenido es el mensaje: "No se puede ingresar una cantidad negativa" y con código de error 1. Este resultado difiere del esperado, por lo que habrá que reestructura el código para que tenga el comportamiento previsto.

Caja negra. Método retirar.

El enunciado del programa indica que el método retirar 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.

Clases de equivalencia.

Condición de entrada	Clases válidas		Clases no válidas	
Cantidad < saldo	(1)	Cantidad <= 100 Se prueba con un saldo de partida de 100 €.	(2)	Cantidad > 100
			(3)	El parámetro recibe un carácter en lugar de un número.
Cantidad > 0	(4)	Cantidad >=0	(5)	Cantidad < 0
			(6)	Mismo caso que 3

Análisis de valores límite (AVL).

Los valores límite en este ejercicio son:

En el caso de prueba 1: cantidad = 100.

- En el caso de prueba 2: cantidad = 101.
- En el caso de prueba 4: cantidad = 0.
- En el caso de prueba 5: cantidad = -1.

Conjetura de errores.

El valor 0 para cantidad ya está considerado en el caso de prueba 4. No se detecta ninguna otra casuística para considerar casos de prueba adicionales.

Datos de entrada y resultados esperados.

Casos	Cantidad	Resultado esperado

1	100	El valor del saldo se reduce en 100 €.
2	101	El programa genera un mensaje advirtiendo de saldo insuficiente.
3	'H'	El programa generará una excepción controlado el error provocado al introducir una letra en lugar de un número.
4	0	El valor del saldo se reduce en 0 €.
5	-1	El programa genera un mensaje advirtiendo de que la cantidad solicitada para retirar es menor de 0.
6	'H'	Mismo caso de uso que el 2.

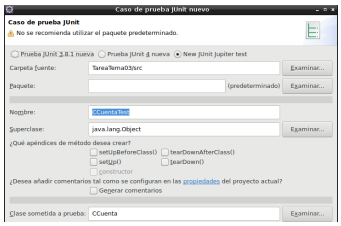
Notas:

- Posteriormente habrá que lanzar la ejecución del programa con los datos de entrada indicados en la tabla y comprobar si los resultados obtenidos coinciden con los esperados.
- Observa que para las pruebas de caja negra no hemos tenido en cuenta el código implementado.

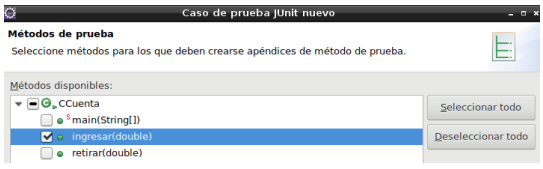
JUnit. Método ingresar.

Los pasos a seguir son (con Eclipse):

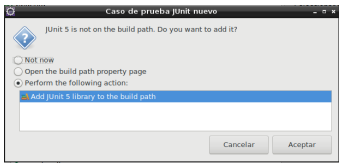
- Crear la clase CCuentaTest del tipo **Caso de prueba JUnit** en Eclipse. Opción de menú: "Archivo/Nuevo/Otras/Junit/ Caso de prueba JUnit".
- Aparece el asistente donde indicaremos que el nombre de la clase de prueba sea CCuentaTest y que la versión de JUnit a utilizar sea "New JUnit jupiter test".



- Seleccionar el método ingresar para ser probado.



- Aceptar que la librería JUnit sea incluida.

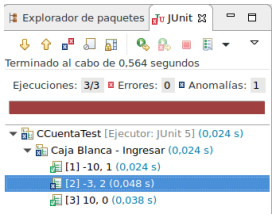


Una vez aceptado, **Eclipse** crea la estructura de la clase de prueba, que actualizaremos como sigue:

```
1  import static org.junit.jupiter.api.Assertions.*;
2  import org.junit.jupiter.api.DisplayName;
3  import org.junit.jupiter.params.ParameterizedTest;
4  import org.junit.jupiter.params.provider.CsvSource;
5  class CCuentaTest {
6          CCuenta miCuenta = new CCuenta();
7          @ParameterizedTest
8          @CsvSource({"-10,1","-3,2","10,0"})
9          @DisplayName("Caja Blanca - Ingresar")
10          void testIngeso(double cant,int resul) {
11                    assertEquals(resul,miCuenta.ingresar(cant));
12          }
13 }
```

Observa en la línea 8 las tres tuplas de datos de entrada y resultados esperados que habíamos obtenido en el estudio de caja blanca para el método ingresar.

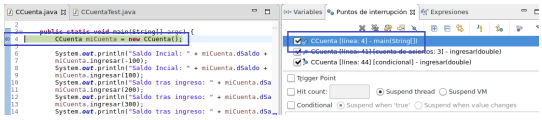
El resultado de las pruebas **JUnit** es el esperado, los casos de prueba 1 y 3 son satisfactorios y el caso 2 advierte de fallo. Se pueden ver los resultados en la siguiente figura.



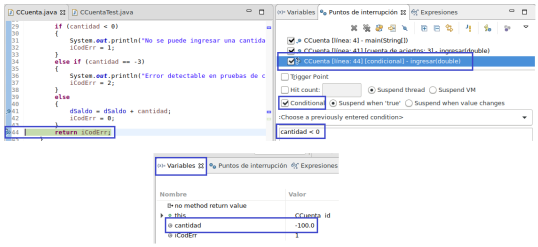
Depuración.

Se definen tres puntos de parada:

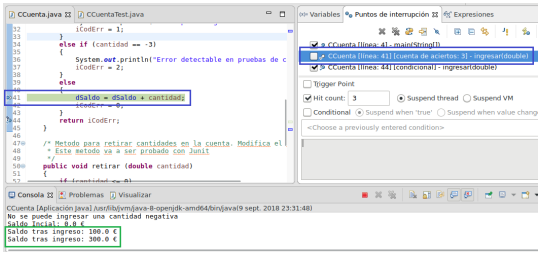
- **Punto de parada sin condición al crear el objeto miCuenta en la función main**. Pulsar con el ratón en el margen izquierdo de la línea de código donde queremos activar el punto de ruptura. No se incluye condición alguna en la vista "Puntos de interrupción".



- **Punto de parada en la instrucción return del método ingresar sólo si la cantidad a ingresar es menor de 0**. Activar el punto de ruptura e incluir la condición cantidad < 0 (ver vista puntos de interrupción). Observa que durante la depuración, el programa ha parado y en la vista variables el valor de cantidad es menor de 0 (-100 en la imagen).



- **Punto de parada en la instrucción donde se actualiza el saldo, sólo deberá parar la tercera vez que sea actualizado**. Activar el punto de ruptura y activar el valor del campo "Hit count" a 3. Observa que durante la depuración, el programa nos informa en la consola de dos ingresos anteriores. Por lo tanto ha parado al ser el tercera actualización de la variable dSaldo mediante ingreso.



La depuración del código se ha realizado con el siguiente código en el 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");
}
```