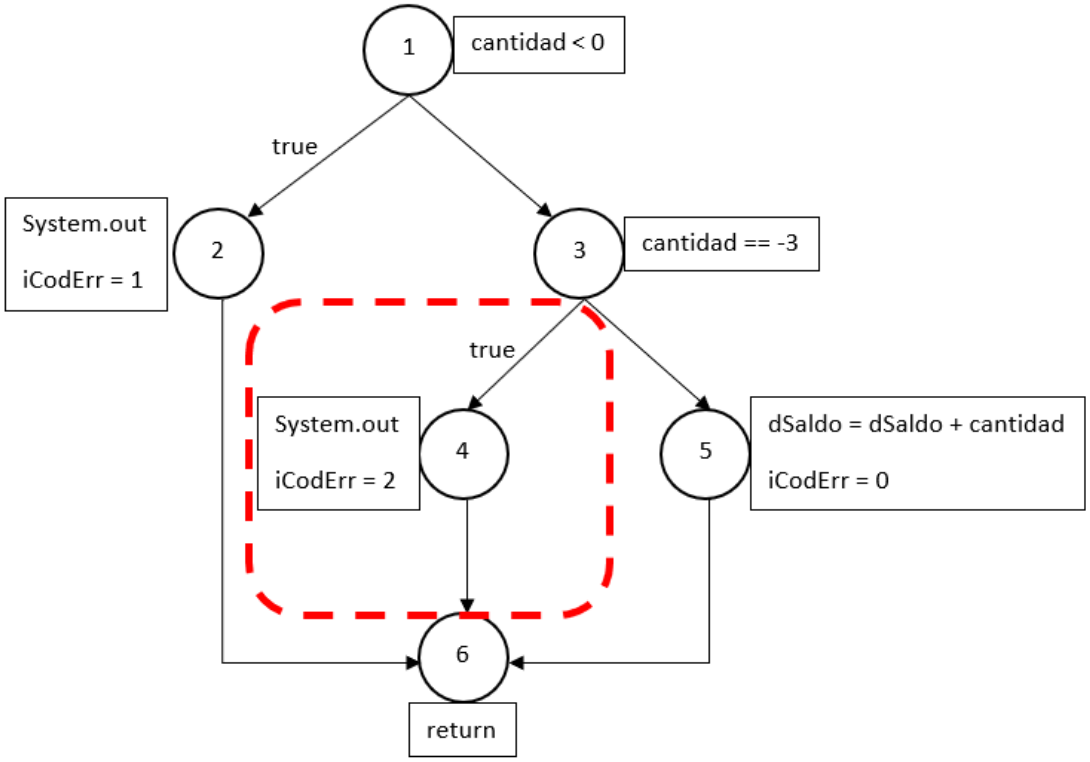
# Solución de la tarea para ED03. Diseño y realización de pruebas

## Caja blanca. Método ingresar.

### Creación del grafo.




**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 reestructurar el código para que tenga el comportamiento previsto. Es la zona rodeada por una línea roja discontinua en el grafo

## 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<br>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 controlando 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 3. |

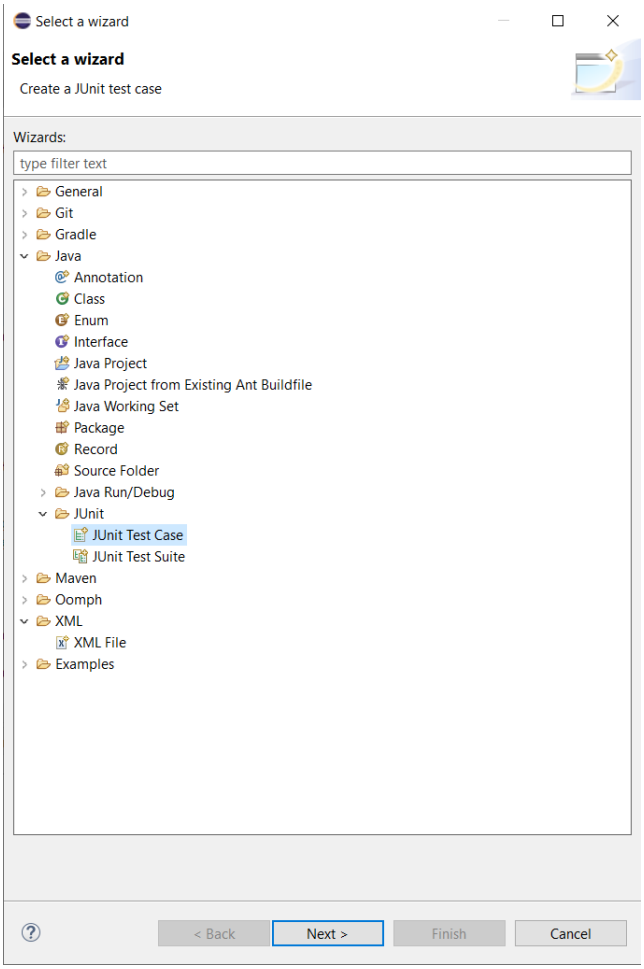
**Notas**:

- 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.

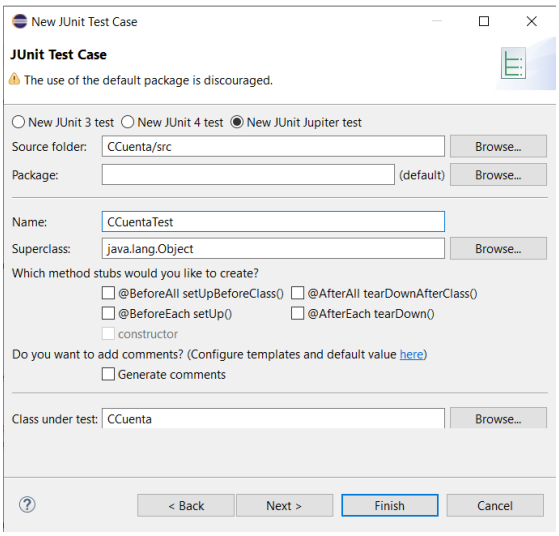
# JUnit. Método ingresar.

## ECLIPSE

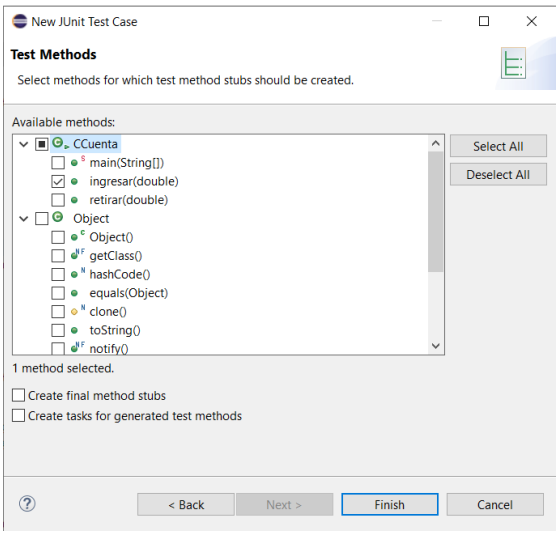
- Crear la clase CCuentaTest del tipo **Caso de prueba JUnit** en Eclipse. Opción de menú: "Archivo/Nuevo/Otras/Junit/ Caso de prueba JUnit".



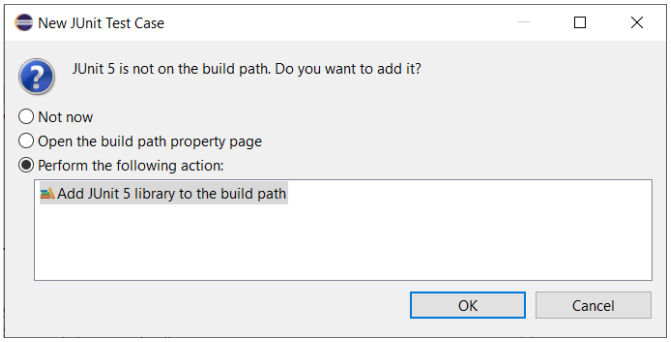

- Aparece el asistente donde indicaremos que el nombre de la clase de prueba sea CCuentaTest y que la versión de JUnit a utilizar sea "New JUnit jupiter test". También se debe indicar la clase que se va a probar



- Seleccionar el método ingresar para ser probado.




- Aceptar que la librería JUnit sea incluida.

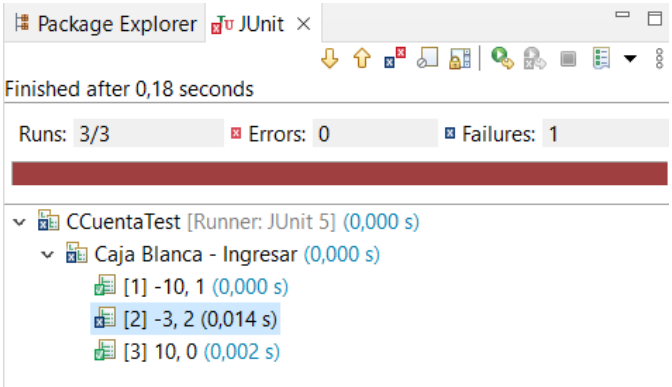



Una vez aceptado, **Eclipse** crea la estructura de la clase de prueba. Sustituimos el código con el proporcionado en el enunciado, que actualizaremos la línea de CsvSource como sigue:

@CsvSource({"-10,1","-3,2","10,0"})

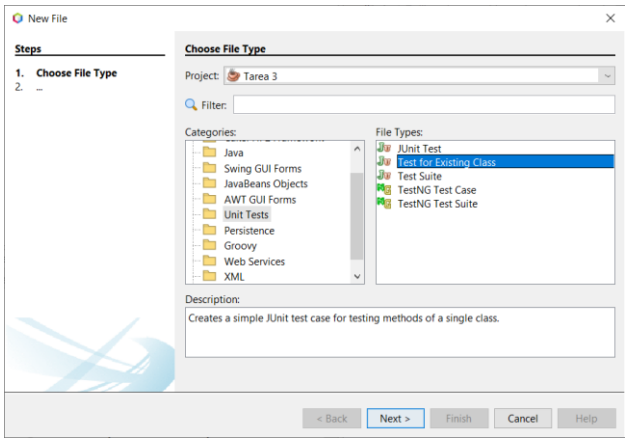
Observa las tres tuplas de datos de entrada y resultados esperados que habíamos obtenido en el estudio de caja blanca para el método ingresar.

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.

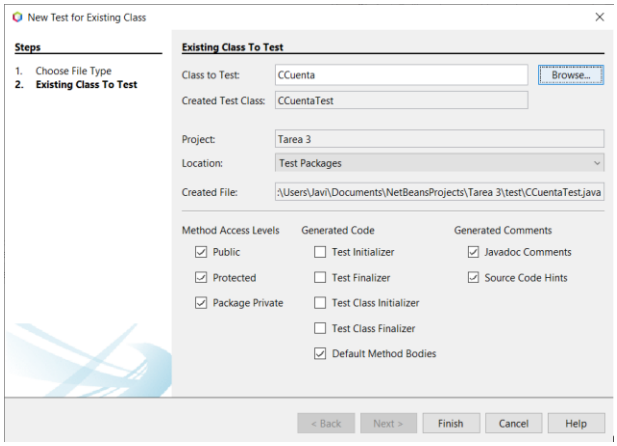


## NETBEANS

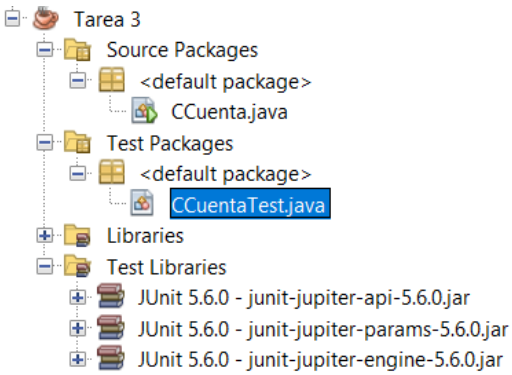
- Crear un nuevo archivo de la categoría **Unit Test** del tipo **Test para clase existente**



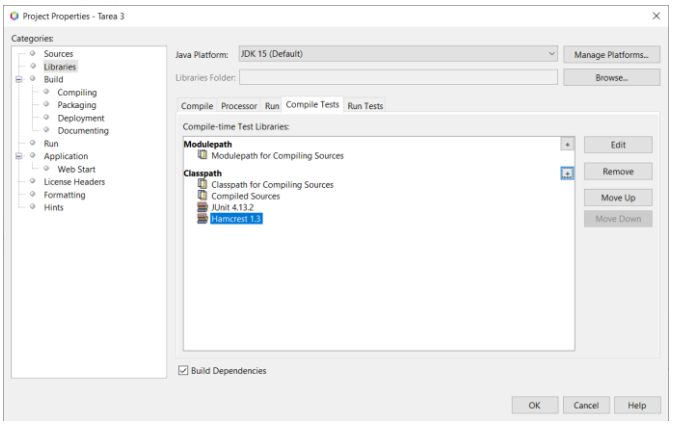

- Seleccionamos la clase CCuenta




- Nos queda una estructura de proyecto como la de la imagen




- Como nos piden realizar pruebas con Junit 4 tenemos que sustituir las librerías de prueba. Eliminamos las librerías de Junit 5 y ponemos las indicadas en el enunciado:




- Sustituimos el código generado por el proporcionado:
- Corregimos los datos de entrada del método etiquetado como @Parameters y, como dice el propio código, corregimos los valores:

{-10, 1}, {-3.0, 2}, {10.0, 0}

- Y eliminamos la llamada autogenerada a fail

```
// TODO review the generated test code and remove the default call to fail.
fail("The test case is a prototype.");
```

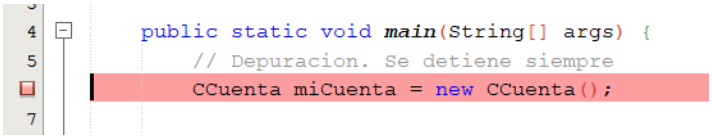
Observa las tres tuplas de datos de entrada y resultados esperados que habíamos obtenido en el estudio de caja blanca para el método ingresar.

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.

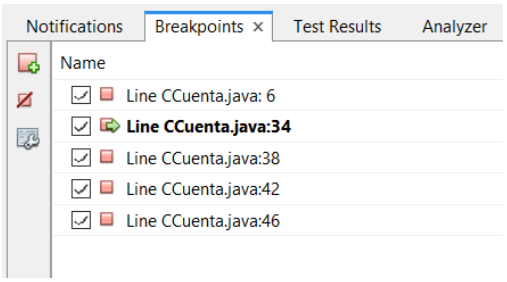
Se definen tres puntos de parada:

- **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 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)** En este apartado el enunciado crea dudas porque por un lado indica donde "devuelva un código de error" y por otro lado indica "3 puntos de parada". Si queremos analizar el código de forma adecuada tenemos que hacer caso a ambas cosas, dependiendo de cómo modifiquemos el código. Para analizarlo de forma exhaustiva la forma más adecuada es la siguiente:

```
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          System.out.println("No se puede ingresar una cantidad negativa");
34          iCodErr = 1;
35      }
36      if (cantidad == -3) {
37          System.out.println("Error detectable en pruebas de caja blanca");
38          iCodErr = 2;
39      } else {
40          // Depuracion. Punto de parada. Solo en el 3 ingreso
41          dSaldo = dSaldo + cantidad;
42          iCodErr = 0;
43      }
44
45      // Depuracion. Punto de parada cuando la cantidad  es menor de 0
46      return iCodErr;
47  }
48
```




En eclipse, con otra posible variación de código:

```java
/* Metodo para ingresar cantidades en la cuenta. Modifica el saldo.
 * Este metodo va a ser probado con Junit
 */
public int ingresar(double cantidad) {
    int 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;
}

```