**Pruebas unitarias Junit**

URL del repositorio GITHUB compartido: <https://github.com/Vasi81/Calculadora_Entornos>

Nos ponemos de acuerdo para crear el nuevo paquete con el nombre com.entornos.junit , cada uno en su proyecto “Calculadora\_Entornos”.

Una vez creado el paquete en el proyecto cada miembro del equipo realiza las pruebas con una clase creada por otro miembro del equipo.

Se hacen varias reuniones para repartir tareas, las clases que testar cada uno tal y como indican en los requerimientos de la actividad, aclarar dudas y tomar decisiones.

Mediante git vamos completando las distintas pruebas en el repositorio y actualizando este mismo documento por cada uno de los integrantes.

**Lo primero que hacemos todos los integrantes de la actividad antes de nada es actualizar nuestro repositorios locales con el repositorio remoto, de esa manera todas comenzaremos con la misma imagen del proyecto “git pull origin main”.**

Pruebas métodos clase Producto

Vesselin Stanev

Primer @Test

En el primer test implemento la prueba testProductoEntero que realiza un test multiplicando 2 números enteros. Para esta prueba utilizo el assertTrue que devuelve TRUE si el resultado del producto de los dos números es igual a 35

Segundo @Test

En este caso implemento la prueba testProductoReales que realiza un test multiplicando 2 números reales. Para esta prueba utilizo el **assertEquals** que comprueba si el resultado esperado (35.0) coincide con el resultado real que nos daría el método metProductoReal (), (metProductoReal(operador3, operador4). Si el resultado esperado coincide con el resultado real nos sacará por consola el siguiente mensaje:"El test del cálculo del producto de dos números reales es correcto"

}

Tercer @Test

En este caso implemento la prueba del testProductoTresReales que realiza un test multiplicando 3 números reales. Para esta prueba utilizo de nuevo el método **assertEquals** que compruebasi el resultado esperado (315) coincide con el resultado real que nos daría el método metProductoReal3(), (metProductoReal3(operador3, operador4, operador5). Si el resultado esperado coincide con el resultado real nos sacará por consola el siguiente mensaje: "El test del cálculo del producto de tres números reales es correcto"

Cuarto @Test

En este caso hago la prueba implementando testPotencia utilizando el método **fail** de Junit que hace que salte la excepción cuando el valor del operador 2 es mayor que 15. Llamamos al método metProductoPotencia dentro del bloque try-catch que captura la excepción esperada.

En caso contrario (si el valor es <=15) la operación se ejecutaría correctamente.

Pruebas métodos clase Suma

José María Marqués González

Primer @Test

En el primer test “TestSumaEnteros”, realiza una suma de dos números enteros utilizando “AsserEquals”. Y esperando que el resultado se igual al esperado, en este caso sería 11, sumando operador1 = 5 y operador2 = 6. Siendo los operadores mayores a cero.

Segundo @Test

En este segundo test “testSumaReal”, realiza una suma de tres números reales utilizando el método “asserEquals”. Y esperando que el resultado se igual al esperado, en este caso sería 10, sumando operador3 = 3.0, operador4 = 6.0 y operador5 = 1.0. Siendo los operadores mayores a cero.

Tercer @Test

En este tercer test “testSumaReal”, realiza una suma de dos números reales utilizando el método “asserEquals”. Y esperando que el resultado se igual al esperado, en este caso sería 8, sumando operador6 = 6.5 y operador7 = 1.5. Siendo los operadores mayores a cero.

Cuarto @Test

En este cuarto test “testUnicoentero”, realiza la suma de todos los enteros acumulados más el valor que le doy a mi variable num = 5. El resultado es el esperado 16. Los operadores mayores a cero también.

Quinto @Test

En este quinto test “testUnicoReal”, realizo la suma de todos los números acumulados reales, y les sumo al valor num = 1.0 que le doy a mi variable. El resulta do es 11 que es el esperado. Los operadores mayor a cero.

Pruebas métodos clase Cociente

José ignacio Gutiérrez Cerrato

Primer @Test

En este primer test implementamos el test “TestCociente”realizando la división de dos números enteros , operador1 con valor 12 y operador2 con valor 2.

**assertEquals(6,c1.cociente(operador1, operador2));**

Devuelve el valor true por lo tanto el test es correcto, ya que 12/2 es igual a 6 que es el valor que usamos como comparación con el resultado esperado.

Segundo @Test

En este segundo test implementamos el test “TestCocienteReal”realizando la división de dos números reales , operador1 con valor 12.0 y operador2 con valor 2.0.

**assertEquals(6,c1.** TestCocienteReal **(operador1, operador2),0.01);**

Devuelve el valor true por lo tanto el test es correcto, ya que 12/2 es igual a 6 que es el valor que usamos como comparación con el resultado esperado.

Tercer @Test

En este tercer test implementamos el test “TestCocienteby0” gestionamos la excepción esperada en caso de tener una división por 0, operador2=0.

**try** {

c1.cociente(operador1, operador2);

*fail*("Esperamos una excepcion al dividir por 0 ,el operador2 no puede ser 0");

}

**catch** (Exception expectedException) {

System.***out***.println(expectedException.getMessage());

}

System.***out***.println("Esperamos una excepcion al dividir por 0 ,el operador2 no puede ser 0");

Cuarto @Test

En este cuarta prueba implementamos el test “TestInversoby0” gestionamos la excepción esperada en caso de tener un valor 0 el operador1.

**try** {

c1.inverso(operador1);

*fail*("Esperamos una excepcion ,El operador1 debe ser mayor que cero");

}

**catch** (Exception expectedException) {

System.***out***.println(expectedException.getMessage());

}

Quinto @Test

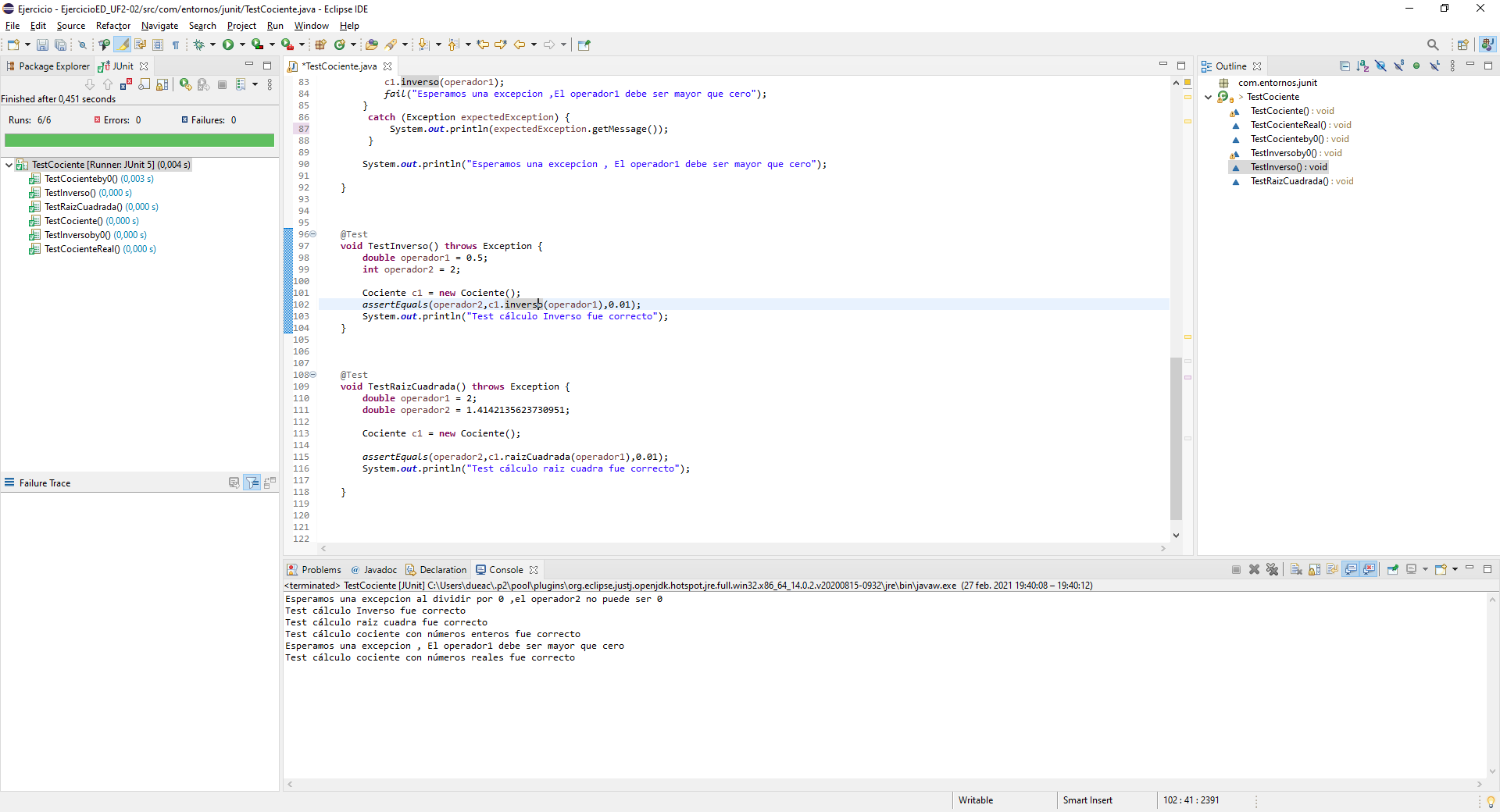
En este quinto test implementamos el test “TestInverso” para verificar que el método c1.inverso(operador1) funciona correctamente, comparando con el valor esperado , en este caso el operador2 .

*assertEquals*(operador2,c1.inverso(operador1),0.01);

Sexto @Test

En este sexto test implementamos el test “TestRaizCuadrada” para verificar que la raíz cuadrada funciona correctamente.

*assertEquals*(operador2,c1.raizCuadrada(operador1),0.01);



Resultado de las pruebas unitarias.

Pruebas métodos clase Resta

Daniel Pavón Gómez

Primer @Test

En el primero Test voy a implementar el caso de la resta de dos números enteros. Utilizo el **assertEquals.** Este comprueba el valor del resultado esperado con el que debería de devolver la función *restaDosEnteros(operador1, operador2)*. Si coincide el valor esperado con el que devuelve la función, nos muestra un mensaje por consola diciendo que todo ha salido correctamente. Sino, junit nos muestra error en este test.

Segundo @Test

En el segundo Test voy a implementar el caso de la resta de dos números reales. Utilizo el assertEquals. Este comprueba el valor del resultado esperado con el que debería devolver la función *restaDosReales(operador 1, operador 2). Ambos operadores son doubles. Si el resultado esperado coincide con el que muestra la función, nos muestra por consola la frase “Test cálculo resta de dos números reales completado correctamente”*

}

Tercer @Test

En el tercer Test voy a implementar el caso de la resta de tres números reales. Utilizo el assertEquals para comprobar el valor del resultado esperado de la funcion *restaTresEnteros(operador1, operador2, operador3).* Si el valor esperado es el mismo que el que devuelve la función, muestra por consola el mensaje: *"La resta de Tres reales ha funcionado correctamente"*

Cuarto @Test

Este es el último test, en este voy a comprobar que funcione correctamente la resta acumulada. Hago una resta del operador1 y del operador2 y el valor lo meto con la variable “laResta” en el setter de *“memoria”*. Luego hago un **assertEquals** donde voy a comprobar el valor del resultado esperado de la funcion restaAcumulada(operador3) y si recibe el valor esperado, devuelve: *"La resta acumulada ha funcionado correctamente.".*