**AE-3. Pruebas de software**  **con JUnit**

**Grupo 10:**

Lidia Díaz Mendoza

Sergio Martínez Rivera

Carlos Rábago Torcates

**URL del repositorio:** <https://github.com/Cambixx/WS_Actividad_ED.git>

**ÍNDICE**

1. Resta (realizado por Sergio Martínez Rivera )

2. Suma (realizado por Carlos Rábago Torcates)

3. Producto (realizado por Lidia Díaz Mendoza )

4. Cociente (realizado por Sergio Martínez Rivera )

**1. RESTA**

* He realizado las pruebas de los métodos de la clase resta, para ello lo primero que he tenido que hacer es crear una nueva carpeta que he llamado test donde van a ir los JUnit test case que utilizaremos para realizar el ejercicio.
* He creado el test case y lo he llamado testResta.
* Para poder trabajar desde el test case con un objeto resta, he tenido que importarme el paquete donde se encuentra dicha clase. Como hasta ahora habíamos trabajado con el paquete por defecto he tenido que crear un nuevo paquete al que he nombrado como principal y he movido alli todas las clases. Una vez realizado lo anterior ya he podido hacer el import principal.resta; y ponerme a trabajar en las pruebas.
* Me he declarado un objeto resta que he llamado r1 para trabajar con todas las pruebas.
* He creado un método llamado limpiar() que es @BeforeEach y se va a encargar de crear vacío el objeto r1 siempre antes de realizar cada @test.
* El primer método con @test será el de la resta de dos números reales. He creado una variable ResultadoEsperado y le he puesto el valor que debe darme la operación y la variable ResultadoObtenido va a tener el valor resultante de la resta de la llamada al método restaDosReales. Compruebo que el método hace bien su función comparando el resultadoEsperado con el resultadoObtenido con AssertEquals.
* Con los dos siguientes métodos hago prácticamente lo mismo, pero trabajando con enteros en vez de con reales y cos sus respectivos métodos.
* Para el método de acumulado he llamado con el objeto al propio método y le he introducido varios valores.
* He creado de nuevo la variable ResultadoEsperado a la que he dado el valor resultante de sumar los valores que he ido metiendo al método. Y por otro lado la variable ResultadoObtenido tiene el valor de Acumulado, es decir de getAcumulado(). Con AssertEquals de nuevo comparo el ResultadoObtenido con el ResultadoEsperado.

\*Al trabajar Carlos y yo de manera simultanea nos hemos dado cuenta antes de subir mi parte a github que el también había creado la carpeta para trabajar los test y habia movido las clases a un nuevo paquete que no era el de por defecto y por tanto el proyecto me salía con conflictos. Nos hemos puesto de acuerdo y he borrado y trabajado a partir de sus modificaciones. Una vez solucionado lo he subido al repositorio local.

**2. SUMA**

* Para empezar he creado la carpeta test y he configurado el buid path con el src, seguidamente he creado el paquete junit donde se van a realizar las pruebas de los métodos de las diferentes clases. En esta oportunidad a mi me ha tocado probar los métodos de la clase suma.
* Para comenzar a hacer pruebas he creado un test case que he llamado SumaTest en el cual he declarado una variable de clase llamada s1 que va a ser de tipo Suma.
* Luego he hecho la primera anotación que ha sido un @BeforeEach el cual se va a encargar de crear un nuevo objeto s1 y cada vez que se haga un nuevo test sed volverá a limpiar s1 (por decirlo de alguna manera) inicializándolo como un nuevo objeto.
* A continuación se realiza la segunda anotación que en este caso es el primer @test en el cual pruebo el método sumaDosReales(), aquí declaro una variable ‘resultado’ de tipo double la cual va a almacenar el resultado obtenido al utilizar el método y que se va a utilizar para comparar con el resultado esperado. He realizado 4 pruebas: una con dos números reales positivos, otra con un número positivo y otro negativo, con dos números negativos y colocando 0 en ambos parámetros, en todos los casos las pruebas se han dado por válidas.
* En el segundo @test se hizo la prueba del método sumaDosEnteros() se hicieron las mismas pruebas que en el test anterior pero en este caso pasando nímeros enteros, de igual manera en este caso las pruebas fueron válidas.
* El tercer @test se hizo la prueba al método sumaTresReales() en este caso se hicieron 4 pruebas donde se le indicó por parámetros al método 3 números reales positivos, otra dónde un número era positivo, un 0 y otro negativo, luego otra donde el resultado fuese negativo y otra donde los tres parámetros fuesen 0. En todos los casos las pruebas fueron válidas.
* Para el cuarto @test dónde se hizo la prueba al método guardarValorAcumulado() se hizo una primera prueba dónde se llamaba al método para acumular un primer número positivo y luego se indicaba otro número positivo y en este caso la prueba se validó sin problemas pero al indicarle un tercer valor, en este caso negativo, la prueba falló como se puede ver a continuación.





* Al intentar acumular un número negativo el resultado obtiene una diferencia de 0.000000000000001 lo cual hace que, al intentar comparar lo esperado con lo obtenido el resultado de fallo ya que el assertEquals arroja false.
* Por lo anteriormente mencionado he decidido modificar el código de el método guardarValorAcumulado() de tal manera que la precisión del valor acumulado sea de dos decimales. Ya en este caso he vuelto a realizar la prueba y ha dado por válida. También he decidido hacer la prueba he introducir otro número negativo mayor al que ya estaba acumulado para forzar un valor acumulado negativo y también acumulando un 0 y ambas pruebas resultaron válidas.

**3. PRODUCTO**

* Lo primero ha sido hacer un pull para bajarme los cambios de mis compañeros.
* Después he creado **JUnit test case** y lo he llamado **TestProducto**. Aquí he añadido "*import activdadED.Producto;*" para acceder al método.
* He declarado una variable de tipo **Producto** llamada **p1** y he creado un método con ***@BeforeEach*** para limpiar y crear un nuevo objeto cada vez que se realice un test.
* He creado el primer test con la anotación **@Test** y lo he llamado ***testProductoReales***. Aquí he declarado una variable **resultado** de tipo double para almacenar el resultado que pasaré con el método y poder comapararlo con el **resultado esperado**. He hecho 3 pruebas: con números positivos, con 0 y con números negativos. Todas las pruebas salieron correctas.
* He repetido los mismos pasos para el segundo **@Test** para probar el método ***ProductoEnteros***, pero en este caso utilizando números enteros.
* En el tercer **@Test** he probado el método ***ProductoTresReales***, donde he realizado pruebas con números reales positivos, negativos y con algún 0. Aquí también fueron válidas todas las pruebas.
* Repito las mismas pruebas con el cuarto **@Test** para probar el método ***Potencia***, utilizando números reales positivos, negativos y 0. Todos los resultados fueron los esperados y las pruebas fueron válidas.

**4. COCIENTE**

* Para poder trabajar con el cociente he realizado lo mismo que con la clase Resta al principio.
* He creado un test case de Cociente y en él me he importado el paquete con import principal.Cociente;
* También me he creado un objeto tipo Cociente para poder hacer las pruebas, y he creado un método limpiar con la misma función de crear un objeto nuevo vacío de Cociente antes de cada @test con el @BeforeEach.
* Los @test de los métodos de dividir enteros y reales no difieren demasiado de los que he realizado con los métodos de Resta. Son en esencia lo mismo, pero en vez de darle a la variable ResultadoObtenido el valor resultante de una resta lo será de una división.
* Al método de número inverso, además de comprobar que el Resultadoesperado sea igual que el obtenido elevando el numero introducido por –1, le hago otra prueba que consiste en preguntar a través de una sentencia if si el resultado de multiplicar el número introducido por el ResultadoObtenido da igual a la unidad, es decir le pregunto si el número introducido por su inverso es igual a 1.
* Con el método de la raiz cuadrada, llamo con el objeto c1 al método de Raiz y le paso un valor por parámetro y compruebo si la raiz cuadrada de ese número indroducido es igual al ResultadoEsperado. Por otro lado, compruebo si realiza bien la operación multiplicando el resultado obtenido por sí mismo y preguntando si es igual al número introducido previamente.

\*Respecto a las pruebas con el número 0, con la división de dos números reales el resultado me daba 9.223372036854776E16 cuando el divisor es igual a 0, por tanto he realizado las pruebas comparando con ese número. Lo mismo sucede con el inverso si el número introducido es 0. Sin embargo si al return del método le quitas el MathRound el resultado es infinity y no he sabido trabajar con el. Por tanto la prueba la he realizado con 9.223372036854776E16 .

 

Con la raiz cuadrada si le metes un un número negativo te da 0 al utilizar el MathRound, si lo quitas el resultado es NaN , y no he sabido trabajar con dicho valor. Por tanto he decidido en la clase Cociente mantener en el return del método el MathRound y que en el test el resultado esperado sea 0.0.









Lo mismo ocurre con el método de dividir enteros, si le quitas al método de la clase cociente el MathRound del return el valor es infinity, y tampoco se como trabajarlo.