1ºDAW - Entornos de desarrollo

# Práctica 5: Pruebas en Java

En esta práctica vamos a reforzar lo aprendido en la unidad “Diseño y realización de pruebas”.

Para llevar a cabo las pruebas vamos a utilizar tres herramientas:

* JUnit: <https://junit.org/junit5/>
* EclEmma (cobertura de código): <https://www.eclemma.org/>
* SonarLint: <https://www.sonarsource.com/>

Con estas herramientas debemos analizar y realizar pruebas al siguiente proyecto software:

* <https://github.com/JarFP/CalculadoraED>

El profesor introducirá en clase brevemente la funcionalidad del proyecto, que es una calculadora.

El objetivo de esta práctica no es modificar nada del código del proyecto original, tan solo hacer los test del software empleando las herramientas mencionadas, para ello cuentas con el código y con los requisitos listados en el archivo *markdown* del proyecto.

Para ilustrar el resultado deberás clonar el repositorio en tu propio usuario de GitHub y hacer una batería de pruebas empleando JUnit, como has aprendido en clase. Debes probar lo que dicten los requisitos y lo que tu intuición te diga que es interesante probar. Cuando finalices las pruebas debes subir los cambios a tu repositorio.

Deberás detectar cinco tipos diferentes de *code smells* con SonarLint, explicar por que disminuyen la calidad del código y proponer soluciones a los mismos.

También debes usar la extensión EclEmma para valorar la cobertura de pruebas que has alcanzado en el código del proyecto, debes explicar porqué no se llega al 100% de cobertura (en caso de no llegar) y justificar si son necesarias o no dichas pruebas.

Evaluación de la práctica

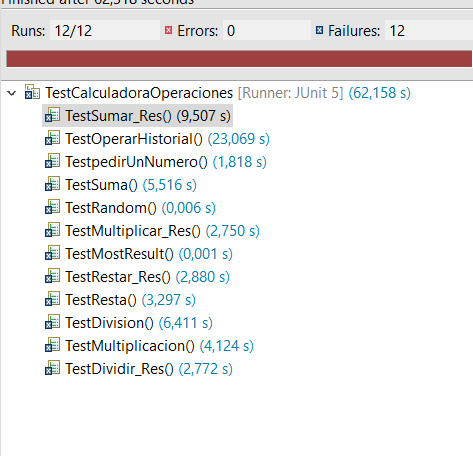
* Uso correcto de JUnit para hacer test unitarios: 60 puntos.
* *Code smells* detectados, explicados y soluciones propuestas: 20 puntos.
* Análisis de cobertura del código y explicación sobre el porcentaje de código no cubierto: 20 puntos.

Entregable

Debes entregar una pequeña memoria en formato PDF en la que figuren:

* La URL de GitHub con tu versión de la solución propuesta con JUnit.
* Capturas de pantalla para ilustrar los resultados de los test unitarios.
* Capturas de pantalla para ilustrar los cinco tipos *code smells* detectados con SonarLint y la redacción de la solución que propones para mejorar la calidad del código.
* Capturas de pantalla de la cobertura alcanzada y la redacción que justifique dicho porcentaje.

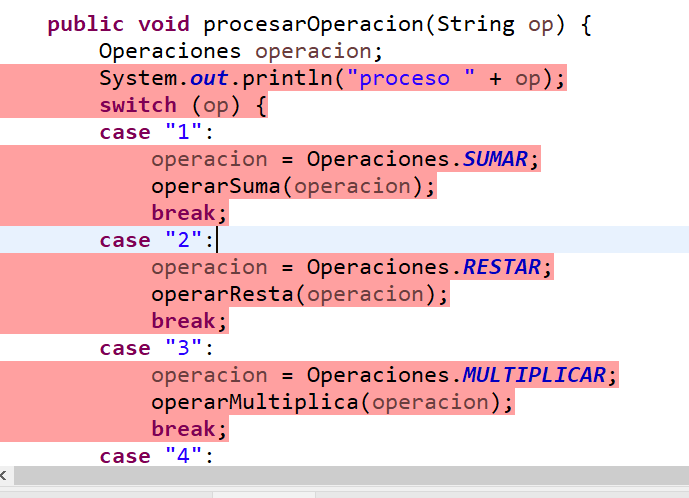
El porcentaje de cobertura alcanzado en esta prueba unitaria es, aunque relativamente aceptable, insuficiente. En primer lugar, no se cubren todas las operaciones que cumple la calculadora, ya que en secciones como “Añadir resultado aleatorio”, falta plantear una prueba que pruebe el funcionamiento de la misma.



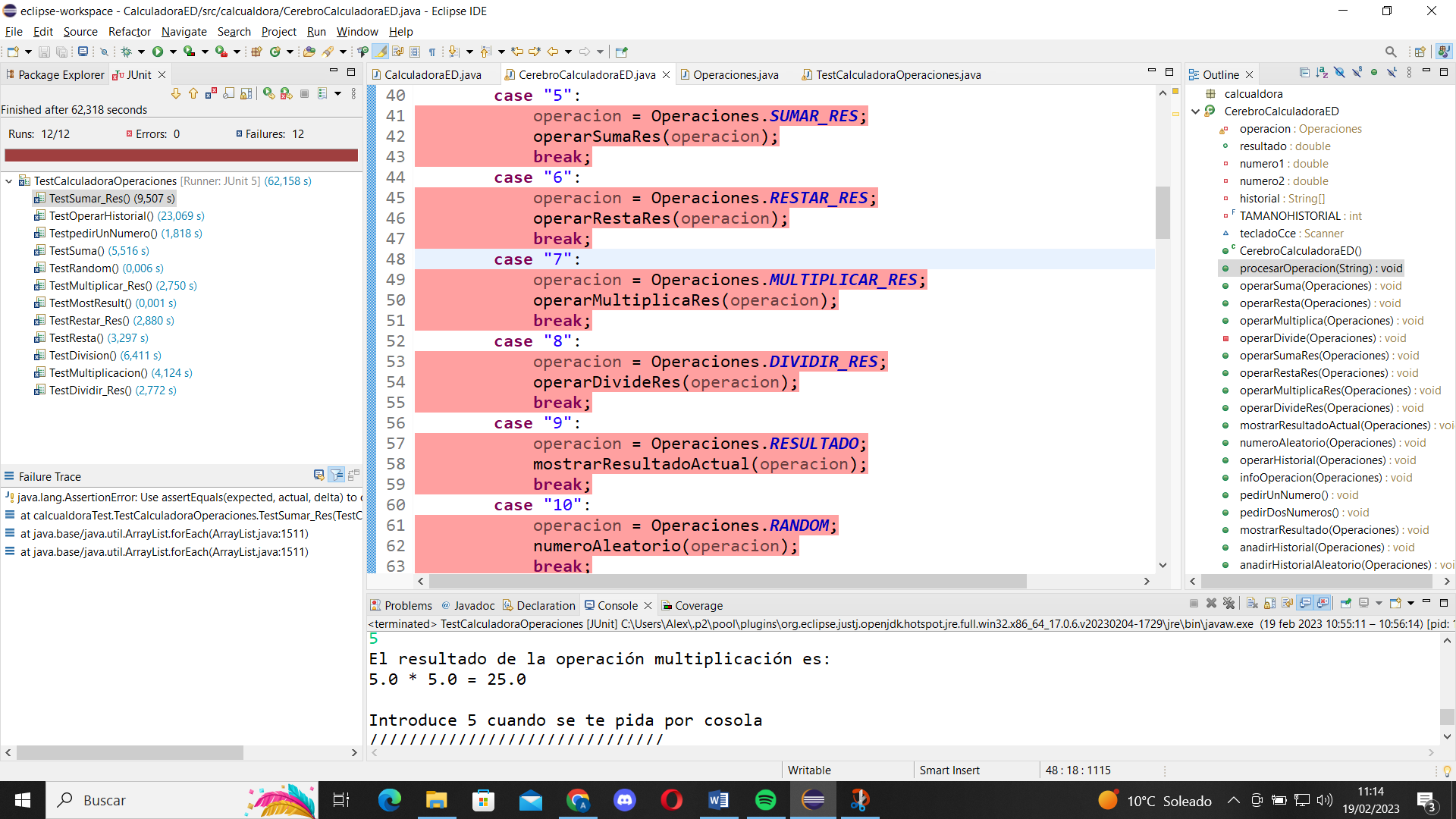
Por otro lado, algunas de las constantes que se han llegado a probar son: “procesarOperacion”, siendo el principal integrante de las pruebas unitarias que se han hecho, ya que es una pieza clave para poder realizar llamamiento a los métodos que se han probado, entre estos métodos encontramos operaciones aritméticas (sumar, restar, multiplicar, dividir) que reciben la operación correspondiente como parámetro y otras operaciones como: Añadir historial aleatorio, que muestra un número aleatorio y Operar historial, que muestra por pantalla las últimas operaciones realizadas y su resultado.

Algunos de los “code smells” que he detectado son:

1.La estructura usada en “procesarOperacion”, esta se puede considerar un code smell ya que la estructura del mismo puede hacer que el código sea difícil de mantener si se desean agregar nuevas funciones, ya que objigaría a los programadores a repetir la estructura de “procesarOperacion” una gran cantidad de veces en el switch case.



2. En los métodos operarSumaRes, operarResraRes, operarMultiplicaRes y operarDivideRes se puede detectar un code smell, ya que la única diferencia entre ellos son los operadores aritméticos, lo que provoca una baja capacidad de mantener el código, pues, cuando se deba actualizar el mismo, se deberá hacer en varios lugares.



3. La clase Operaciones es un dato que no tiene ningún comportamiento y sólo almacena la constante de cambio de operación, esto podría considerarse un code smell ya que no agrega valor añadido al código y parece que se ha planteado así para ahorrar tiempo pero complicar la manutención del código a lo largo del tiempo.

4. El CerebroCalculadoraED tiene muchas responsabilidades y código, por ello, sería buena idea dividir la funcionalidad en otras clases como “sumar” o “restar” por separado en vez de sobrecargar el código de una única clase.

5. El mét.odo getId nunca se llega a utilizar, además de que está mal planteado, pues, en vez de poner cada uno de los datos por separado en diferentes métodos, se podría hacer en un único método

