**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 Cociente

José ignacio Gutiérrez Cerrato

Antes verifico la rama de trabajo, para eso primero realizo merge entre la rama local main y la rama local de desarrollo JIG\_ED :

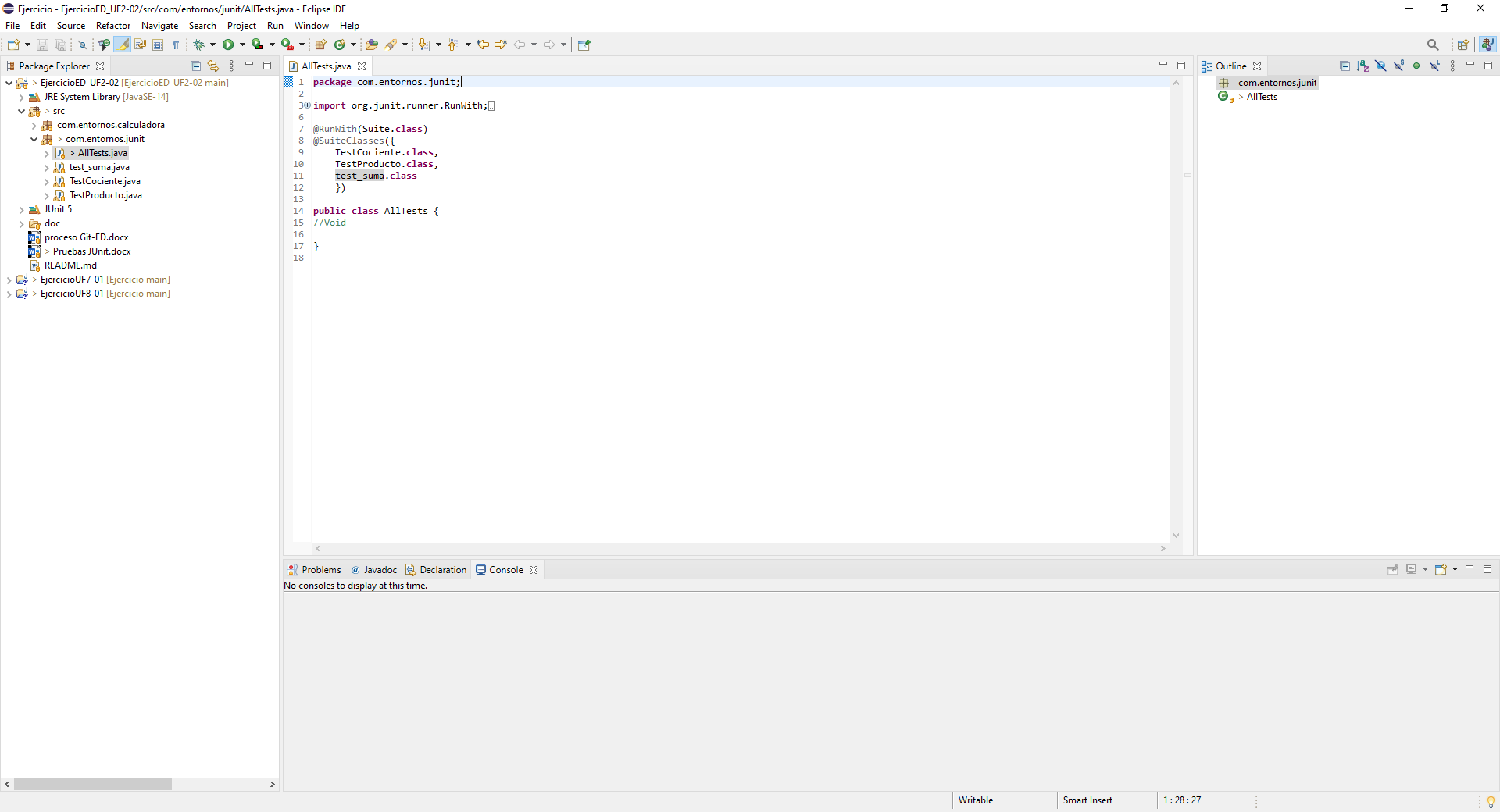
🡪git checkout JIG\_ED

🡪git merge main

Ya Podemos comenzar con la actividad tal y como hemos comentado anteriormente.

Creamos un nuevo paquete (**com.entornos.junit**) y a continuación creamos una nueva clase de tipo **Junit Test Case.**

También podemos crear una nueva clase Junit Test Suite para aglutinar desde esa clase la ejecución de las distintas pruebas case que realicemos.



**Ejemplo de Junit Test Suite**

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

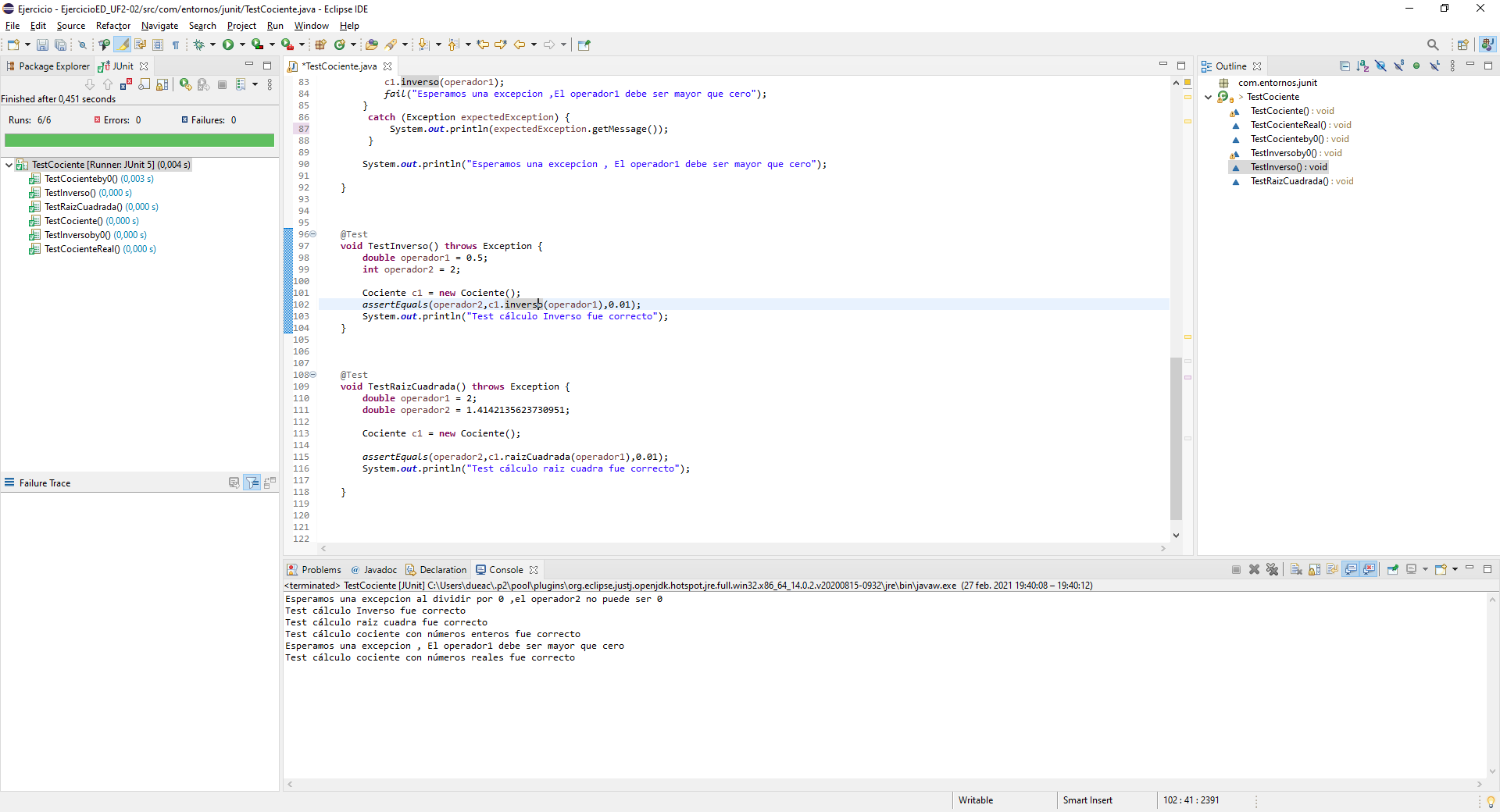
En este quinto test implementamos el test “TestInverso” para verificar que el método c1.inverso(operador1) funciona correctamente, comparando con el valor esperado , en este caso el operador2 .

*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.

Una vez finalizadas las modificaciones continuamos con git:

🡪git add .

🡪git commit -m “TestCociente + Documento de word.”

🡪git checkout main

🡪git merge JIG\_ED

🡪git push origin main.