Universidad de Costa Rica
Daniel Escobar Giraldo C02748
Ingeniería de Software
Grupo 01 - II-2022

Laboratorio 8 - TDD

Enlace de mi repositorio:

https://github.com/DanielEscobar19/C0126_22B_Laboratorios_DanielEscobar

Código de la sección “Create a test and generate code”

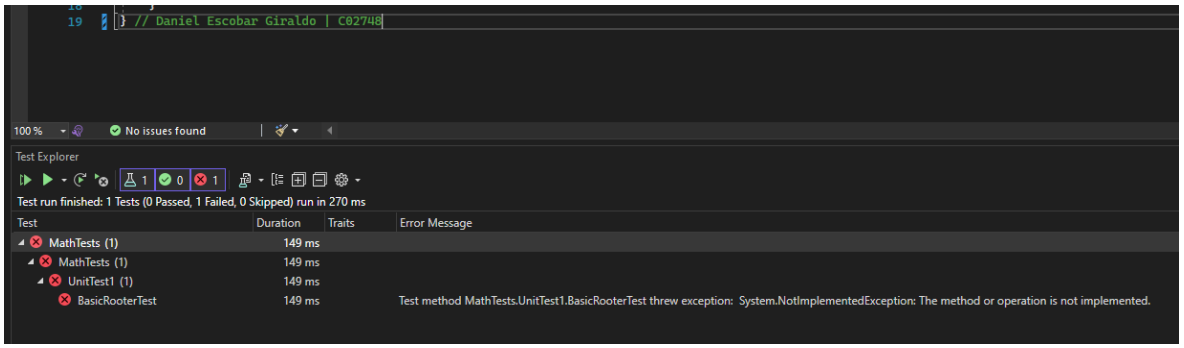
UnitTest1.cs

```
using MyMath;

namespace MathTests
{
    [TestClass]
    0 references
    public class UnitTest1
    {
        [TestMethod]
        0 references
        public void BasicRooterTest()
        {
            // Create an instance to test:
            Rooter rooter = new Rooter();
            // Define a test input and output value:
            double expectedResult = 2.0;
            double input = expectedResult * expectedResult;
            // Run the method under test:
            double actualResult = rooter.SquareRoot(input);
            // Verify the result:
            Assert.AreEqual(expectedResult, actualResult, delta: expectedResult / 100);
        }
    }
} // Daniel Escobar Giraldo | C02748
```

Class1.cs

```
1   namespace MyMath
2   {
        0 references
3       public class Class1
4       {
5
6       }
7
        3 references
8       public class Rooter
9       {
            1 reference | 1/1 passing
10          public Rooter()
11          {
12          }
13
            1 reference | 1/1 passing
14          public double SquareRoot(double input)
15          {
16              throw new NotImplementedException();
17          }
18      }
19  } // Daniel Escobar Giraldo | C02748
```

Resultado de pruebas



Código de la sección “Verify a code change”

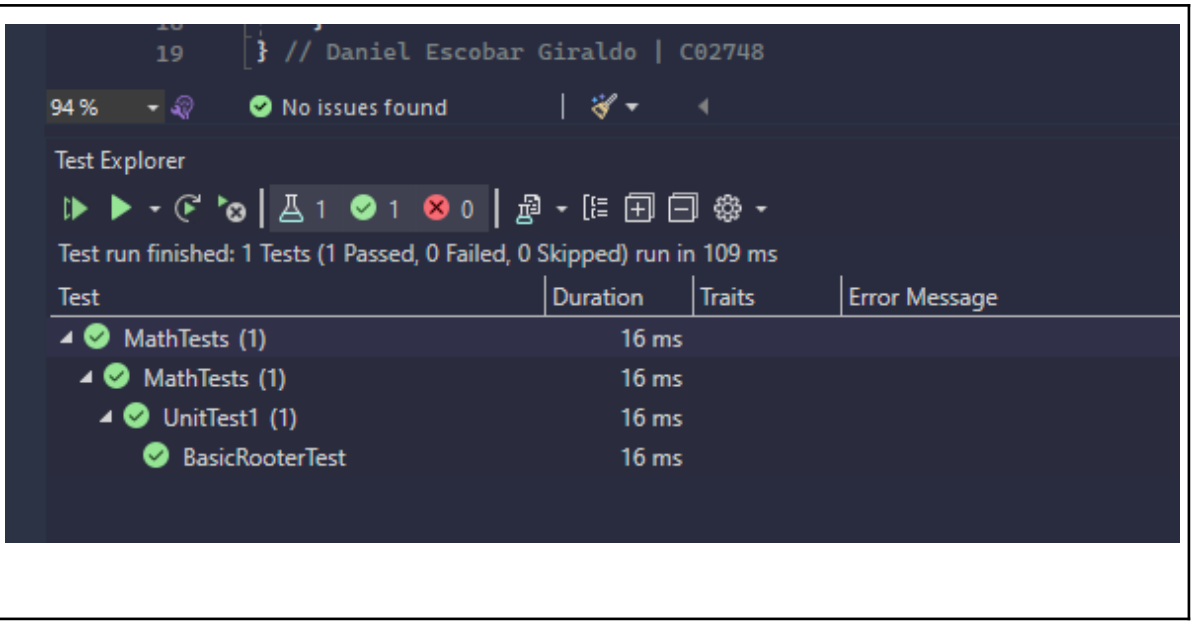
UnitTest1.cs

```
using MyMath;

namespace MathTests
{
    [TestClass]
    0 references
    public class UnitTest1
    {
        [TestMethod]
        0 references
        public void BasicRooterTest()
        {
            // Create an instance to test:
            Rooter rooter = new Rooter();
            // Define a test input and output value:
            double expectedResult = 2.0;
            double input = expectedResult * expectedResult;
            // Run the method under test:
            double actualResult = rooter.SquareRoot(input);
            // Verify the result:
            Assert.AreEqual(expectedResult, actualResult, delta: expectedResult / 100);
        }
    }
} // Daniel Escobar Giraldo | C02748
```

Class1.cs

```
namespace MyMath
{
    0 references
    public class Class1
    {

    }

    3 references
    public class Rooter
    {
        1 reference | 1/1 passing
        public Rooter()
        {
        }

        1 reference | 1/1 passing
        public double SquareRoot(double input)
        {
            return input / 2;
        }
    }
} // Daniel Escobar Giraldo | C02748
```

Resultado de pruebas

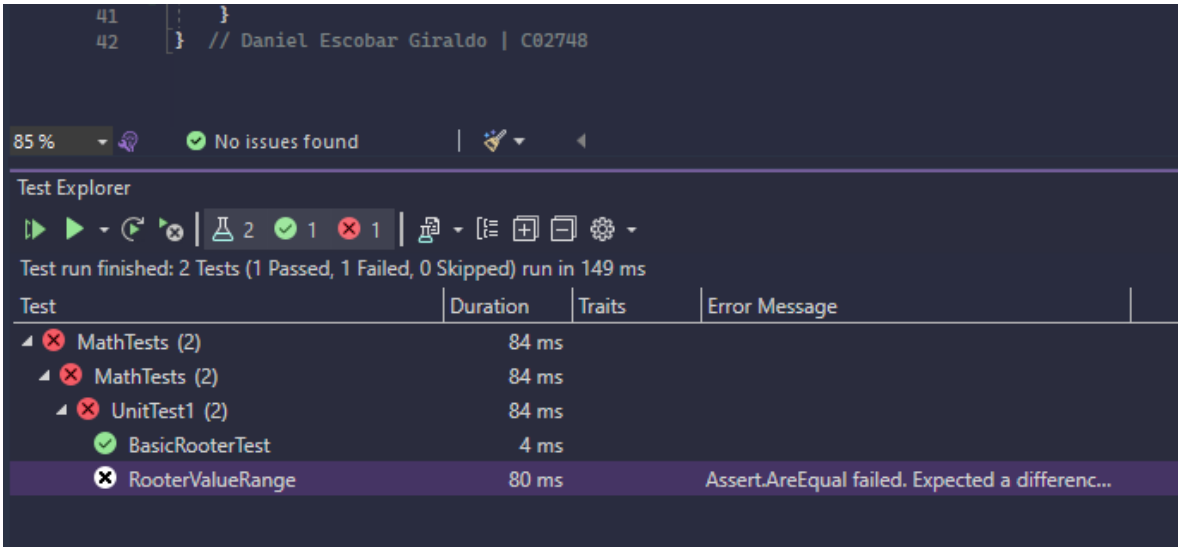

19 } // Daniel Escobar Giraldo | C02748
94 %
No issues found
Test Explorer
1 1 0
Test run finished: 1 Tests (1 Passed, 0 Failed, 0 Skipped) run in 109 ms
Test
Duration
Traits
Error Message
MathTests (1) 16 ms
MathTests (1) 16 ms
UnitTest1 (1) 16 ms
BasicRooterTest 16 ms

Código de la sección “Extend the range of inputs”

UnitTest1.cs

```
using MyMath;

namespace MathTests
{
    [TestClass]
    0 references
    public class UnitTest1
    {
        [TestMethod]
        | 0 references
        public void BasicRooterTest()
        {
            // Create an instance to test:
            Rooter rooter = new Rooter();
            // Define a test input and output value:
            double expectedResult = 2.0;
            double input = expectedResult * expectedResult;
            // Run the method under test:
            double actualResult = rooter.SquareRoot(input);
            // Verify the result:
            Assert.AreEqual(expectedResult, actualResult, delta: expectedResult / 100);
        }

        [TestMethod]
        | 0 references
        public void RooterValueRange()
        {
            // Create an instance to test.
            Rooter rooter = new Rooter();

            // Try a range of values.
            for (double expected = 1e-8; expected < 1e+8; expected *= 3.2)
            {
                RooterOneValue(rooter, expected);
            }
        }

        1 reference | 0/1 passing
        private void RooterOneValue(Rooter rooter, double expectedResult)
        {
            double input = expectedResult * expectedResult;
            double actualResult = rooter.SquareRoot(input);
            Assert.AreEqual(expectedResult, actualResult, delta: expectedResult / 1000);
        }
    }
} // Daniel Escobar Giraldo | C0278
```

Ejecución luego de cambiar UniTest1.cs



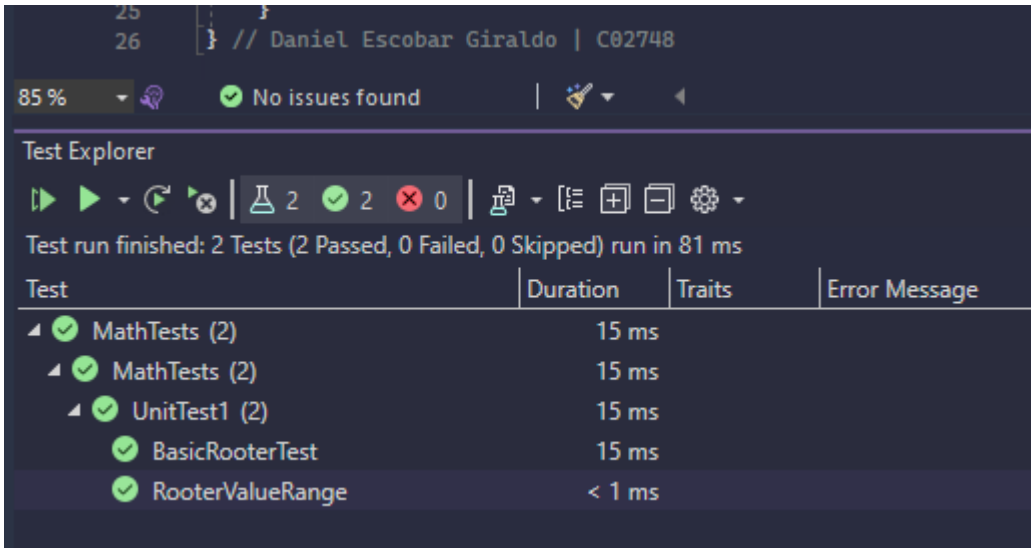
Note que el método BasicRooterTest pasa las pruebas pero el nuevo método no.

Para esto vamos a modificar la clase Class1.cs

Modificación a la clase Class1.cs

```
1  namespace MyMath
2  {
       0 references
3      public class Class1
4      {
5
6      }
7
       6 references
8      public class Rooter
9      {
           2 references | 1/2 passing
10         public Rooter()
11         {
12         }
13
           2 references | 1/1 passing
14         public double SquareRoot(double input)
15         {
16             double result = input;
17             double previousResult = -input;
18             while (Math.Abs(previousResult - result) > result / 1000)
19             {
20                 previousResult = result;
21                 result = result - (result * result - input) / (2 * result);
22             }
23             return result;A
24         }
25     }
26 } // Daniel Escobar Giraldo | C02748
```

Ejecución luego de cambiar Class1.cs



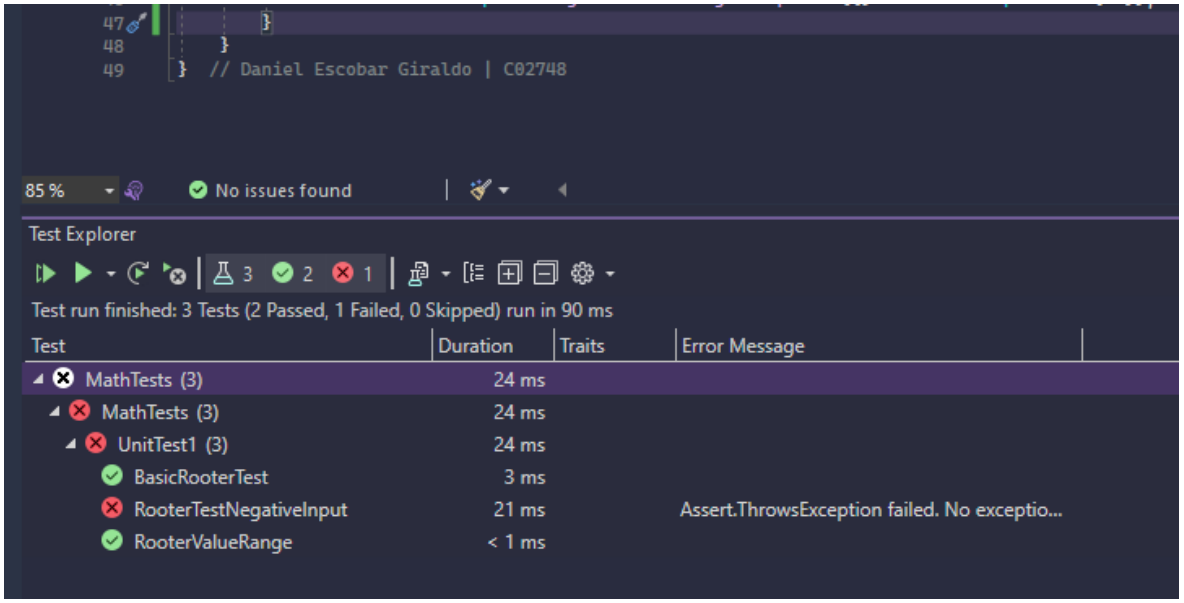
Note que el método SquareRoot() ahora es más robusto y pasa las dos pruebas.

Código de la sección “Add tests for exceptional cases”

Se agrega un caso de prueba a UnitTest1.cs

```
41
42        [TestMethod]
          0 references
43        public void RooterTestNegativeInput()
44        {
45            Rooter rooter = new Rooter();
46            Assert.ThrowsException<ArgumentOutOfRangeException>(() => rooter.SquareRoot(-1));
47        }
48    }
49  } // Daniel Escobar Giraldo | C02748
```

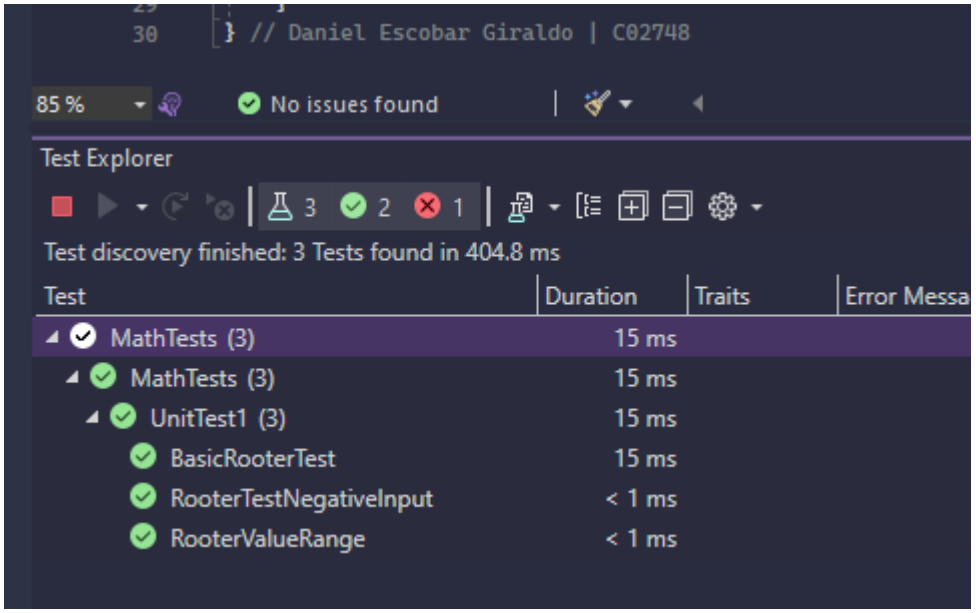
Ejecución luego de cambiar UniTest1.cs



Modificación a la clase Class1.cs

```
1   namespace MyMath
2   {
        0 references
3       public class Class1
4       {
5
6       }
7
        8 references
8       public class Rooter
9       {
            3 references | 2/3 passing
10          public Rooter()
11          {
12          }
13
            3 references | 1/2 passing
14          public double SquareRoot(double input)
15          {
16              if (input <= 0.0)
17              {
18                  throw new ArgumentOutOfRangeException();
19              }
20              double result = input;
21              double previousResult = -input;
22              while (Math.Abs(previousResult - result) > result / 1000)
23              {
24                  previousResult = result;
25                  result = result - (result * result - input) / (2 * result);
26              }
27              return result;
28          }
29      }
30  } // Daniel Escobar Giraldo | C02748
```

Ejecución luego de cambiar Class1.cs



Preguntas:

A) De la sección, Create a test and generate code, explique muy brevemente ¿Por qué la prueba que ejecutó en el paso #6 falló?

Esta prueba falló porque el método Square Root tira una excepción. Luego de esto se modificó el método para que devuelva el valor que recibe dividido entre 2. Con esta modificación pasa al caso de prueba porque recibe el número 4 el cual la raíz cuadrada es efectivamente 2. No obstante con otros valores la prueban se pasa.

B) De la sección, Extend the range of inputs, explique muy brevemente ¿Por qué la prueba que ejecutó en el paso #2 falló?

El caso de prueba RooterValueRange() falla debido a que el método SquareRoot() retorna la mitad del valor que recibe por parámetro en lugar de la raíz cuadrada. Por lo tanto RooterValueRange() falla porque este envía varios números y espera recibir la raíz cuadrada de tales números y no la mitad.

Mientras tanto el caso de prueba BasicRooterTest() pasa el test porque este método le envía un 4 al método SquareRoot() el cual retorna la mitad de 4 y en el caso del 4 el resultado esperado es un 2 porque esta es la raíz y la mitad de 4.