

Laboratorio 8

Enlace de mi repositorio:

https://github.com/DanielEscobar19/C0126_22B_Laboratorios_DanielEscobar

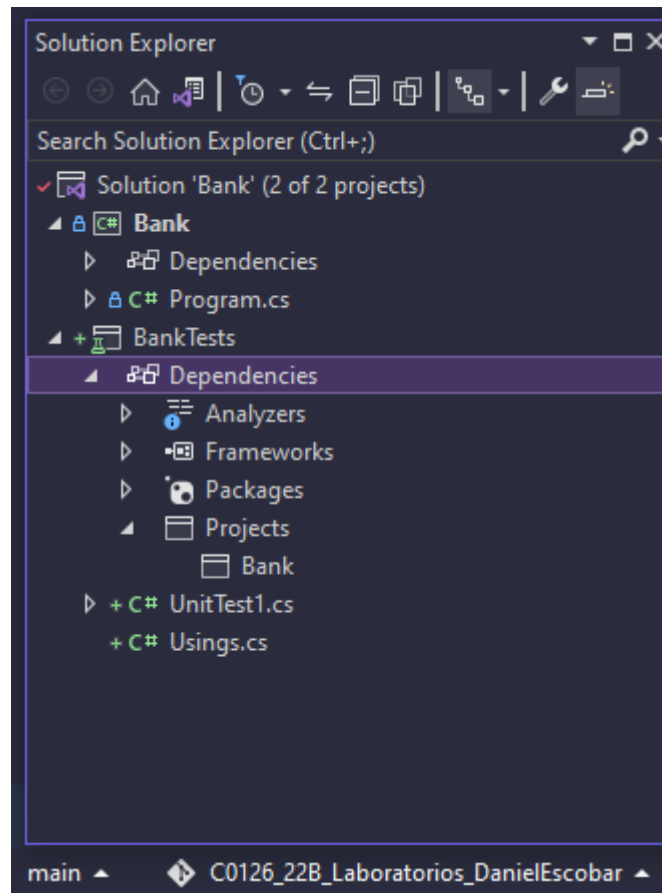
Código de la sección "Create a project to test"

Program.cs de proyecto Bank

```
1  using System;
2
3  namespace BankAccountNS
4  {
5      /// <summary>
6      /// Bank account demo class.
7      /// </summary>
8      4 references
9      public class BankAccount
10     {
11         private readonly string _customerName;
12         private double _balance;
13
14         0 references
15         private BankAccount() { }
16
17         1 reference
18         public BankAccount(string customerName, double balance)
19         {
20             _customerName = customerName;
21             _balance = balance;
22         }
23
24         0 references
25         public string CustomerName
26         {
27             get { return _customerName; }
28         }
29
30         1 reference
31         public double Balance
32         {
33             get { return _balance; }
34         }
35
36         1 reference
37         public void Debit(double amount)
38         {
39             if (amount > _balance)
40             {
41                 throw new ArgumentOutOfRangeException("amount");
42             }
43
44             if (amount < 0)
45             {
46                 throw new ArgumentOutOfRangeException("amount");
47             }
48
49             _balance -= amount; // intentionally incorrect code
50         }
51
52         1 reference
53         public void Credit(double amount)
54         {
55             if (amount < 0)
56             {
57                 throw new ArgumentOutOfRangeException("amount");
58             }
59
60             _balance += amount;
61         }
62
63         0 references
64         public static void Main()
65         {
66             BankAccount ba = new BankAccount("Mr. Bryan Walton", 11.99);
67
68             ba.Credit(5.77);
69             ba.Debit(11.22);
70             Console.WriteLine("Current balance is ${0}", ba.Balance);
71         }
72     }
73 } // Daniel Escobar Giraldo | C02748
```

Sección “Create a unit test project”

Proyecto de testing BankTests con referencia a Bank



Sección “Create the test class” y “Create the first test method”

Archivo BankAccountTests.cs

```

1  using BankAccountNS;
2
3  namespace BankTests
4  {
5      [TestClass]
6      0 references
7      public class BankAccountTests
8      {
9          [TestMethod]
10         0 references
11         public void Debit_WithValidAmount_UpdatesBalance()
12         {
13             // Arrange
14             double beginningBalance = 11.99;
15             double debitAmount = 4.55;
16             double expected = 7.44;
17             BankAccount account = new BankAccount("Mr. Bryan Walton", beginningBalance);
18
19             // Act
20             account.Debit(debitAmount);
21
22             // Assert
23             double actual = account.Balance;
24             Assert.AreEqual(expected, actual, 0.001, "Account not debited correctly");
25         }
26     }
27 } // Daniel Escobar Giraldo | C02748

```

Resultado de ejecutar los casos de prueba

24 }
25 } // Daniel Escobar Giraldo | C02748

94 % No issues found

Test Explorer

1 0 1

Test run finished: 1 Tests (0 Passed, 1 Failed, 0 Skipped) run in 105 ms

Test	Duration	Traits	Error Message
BankTests (1)	38 ms		
BankTests (1)	38 ms		
BankAccountTests (1)	38 ms		

Sección "Fix your code and rerun your tests"

Archivo BankAccount.cs

```

30
31 2 references | 1/1 passing
32 public void Debit(double amount)
33 {
34     if (amount > m_balance)
35     {
36         throw new ArgumentOutOfRangeException("amount");
37     }
38     if (amount < 0)
39     {
40         throw new ArgumentOutOfRangeException("amount");
41     }
42
43     m_balance -= amount; // linea corregida
44 } // Daniel Escobar Giraldo | C02748
45

```

Resultado de ejecutar los casos de prueba

Test Explorer Test run finished: 1 Tests (1 Passed, 0 Failed, 0 Skipped) run in 85 ms			
Test	Duration	Traits	Error
BankTests (1)	18 ms		
BankTests (1)	18 ms		
BankAccountTests (1)	18 ms		

Sección “Use unit tests to improve your code”

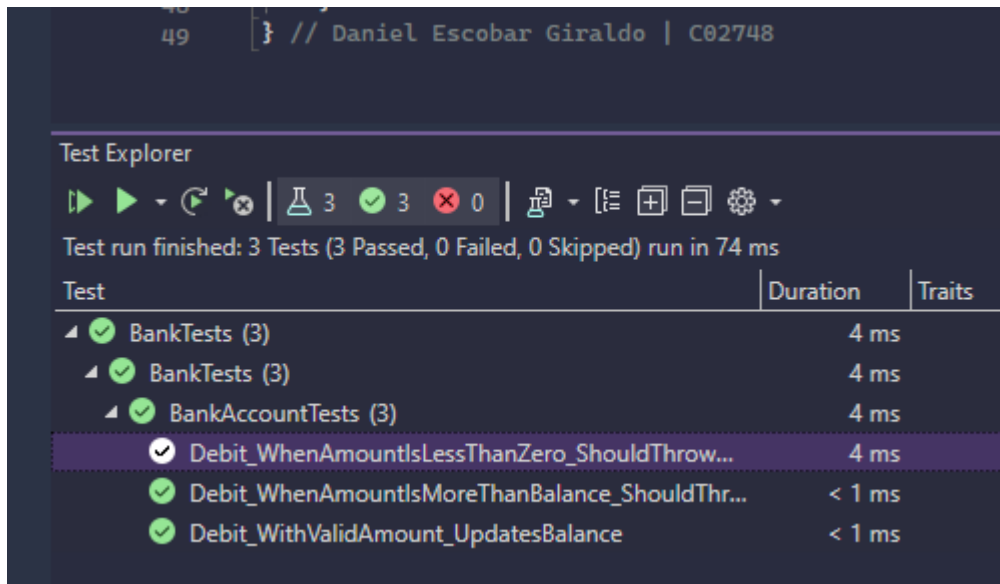
Archivo BankAccountTests.cs

```

24
25 [TestMethod]
26 0 references
27 public void Debit_WhenAmountIsLessThanZero_ShouldThrowArgumentOutOfRangeException()
28 {
29     // Arrange
30     double beginningBalance = 11.99;
31     double debitAmount = -100.00;
32     BankAccount account = new BankAccount("Mr. Bryan Walton", beginningBalance);
33
34     // Act and assert
35     Assert.ThrowsException<System.ArgumentOutOfRangeException>(() => account.Debit(debitAmount));
36 }
37
38 [TestMethod]
39 0 references
40 public void Debit_WhenAmountIsMoreThanBalance_ShouldThrowArgumentOutOfRangeException()
41 {
42     // Arrange
43     double beginningBalance = 11.99;
44     double debitAmount = 50.5;
45     BankAccount account = new BankAccount("Mr. Bryan Walton", beginningBalance);
46
47     // Act and assert
48     Assert.ThrowsException<System.ArgumentOutOfRangeException>(() => account.Debit(debitAmount));
49 }
50 } // Daniel Escobar Giraldo | C02748

```

Resultado de ejecutar los casos de prueba



Note que las pruebas pasan porque el Assert verifica que si se activa la excepción `ArgumentOutOfRangeException()` y el método `Debit()` efectivamente tira esta excepción, por lo que la prueba pasa correctamente.

Sección “Refactor the code under test”

Archivo `BankAccount.cs`

```

1  using System;
2
3  namespace BankAccountNS
4  {
5      /// <summary>
6      /// Bank account demo class.
7      /// </summary>
8      public class BankAccount
9      {
10         private readonly string m_customerName;
11         private double m_balance;
12         public const string DebitAmountExceedsBalanceMessage = "Debit amount exceeds balance";
13         public const string DebitAmountLessThanZeroMessage = "Debit amount is less than zero";
14
15         0 references
16         private BankAccount() { }
17
18         4 references | 3/3 passing
19         public BankAccount(string customerName, double balance)
20         {
21             m_customerName = customerName;
22             m_balance = balance;
23         }
24
25         0 references
26         public string CustomerName
27         {
28             get { return m_customerName; }
29         }
30
31         2 references | 1/1 passing
32         public double Balance
33         {
34             get { return m_balance; }
35         }
36
37         4 references | 3/3 passing
38         public void Debit(double amount)
39         {
40             if (amount > m_balance)
41             {
42                 throw new System.ArgumentOutOfRangeException("amount", amount, DebitAmountExceedsBalanceMessage);
43             }
44
45             if (amount < 0)
46             {
47                 throw new System.ArgumentOutOfRangeException("amount", amount, DebitAmountLessThanZeroMessage);
48             }
49
50             m_balance -= amount; // linea corregida
51         }
52     } // Daniel Escobar Giraldo | C02748
53 }

```

Sección "Refactor the test methods"

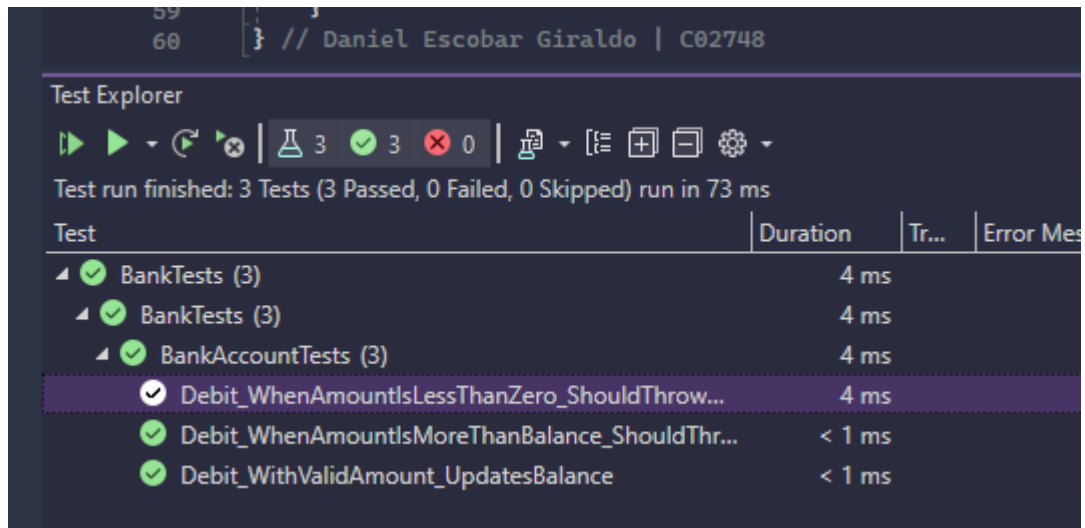
Archivo BankAccountTests.cs

```

37 [TestMethod]
38 // 0 references
39 public void Debit_WhenAmountIsMoreThanBalance_ShouldThrowArgumentOutOfRangeException()
40 {
41     // Arrange
42     double beginningBalance = 11.99;
43     double debitAmount = 20.0;
44     BankAccount account = new BankAccount("Mr. Bryan Walton", beginningBalance);
45
46     // Act
47     try
48     {
49         account.Debit(debitAmount);
50     }
51     catch (System.ArgumentOutOfRangeException e)
52     {
53         // Assert
54         StringAssert.Contains(e.Message, BankAccount.DebitAmountExceedsBalanceMessage);
55         return;
56     }
57
58     Assert.Fail("The expected exception was not thrown.");
59 }
60 // Daniel Escobar Giraldo | C02748

```

Resulta de las pruebas BankAccountTests.cs



Test Explorer

Test run finished: 3 Tests (3 Passed, 0 Failed, 0 Skipped) run in 73 ms

Test	Duration	Tr...	Error Mes
BankTests (3)	4 ms		
BankTests (3)	4 ms		
BankAccountTests (3)	4 ms		
Debit_WhenAmountIsLessThanZero_ShouldThrow...	4 ms		
Debit_WhenAmountIsMoreThanBalance_ShouldThr...	< 1 ms		
Debit_WithValidAmount_UpdatesBalance	< 1 ms		

En este último caso se espera que se tire la excepción de que el debitAmount es mayor al beginningBalance. Como si se está dando esta excepción la prueba pasa correctamente.



Preguntas:

A) Basado en el tutorial del ejercicio de pruebas unitarias, en la última sección, Retest, rewrite, and reanalyze, explique muy brevemente ¿Por qué se necesita agregar la instrucción Assert.Fail a este caso de prueba?

En el tutorial se explica que sin el Assert.Fail la prueba no fallaría en el caso de que se de una excepción diferente la ArgumentOutOfRangeException. Por esto se agrega el Assert.Fail para que la prueba falle en que se de una excepción diferente a la esperada ya que el punto de esta prueba es detectar la excepción mencionada.

B) Basado en la lectura de conceptos sobre pruebas unitarias, responda muy brevemente ¿cuál es el valor de las pruebas unitarias en el flujo de desarrollo de software?

Las pruebas unitarias permiten verificar suposiciones explícitas e implícitas que se hayan realizado en el código y además permite verificar que el código trabaja dentro de los límites que requiere y puede responder a las entradas de datos que se salen de estos límites. En general las pruebas unitarias mejoran la calidad del código y su mantenibilidad.

C) Basado en la lectura de conceptos sobre pruebas unitarias, responda muy brevemente ¿cuáles son las principales partes que componen una prueba unitaria?

Una prueba unitaria está compuesta de tres partes:

- **Arrange section:** es la parte donde se inicializan los objetos necesarios y se asignan los datos que luego se envían al método que se desea probar..
- **Act section:** es la parte que invoca el método que se está probando con los parámetros definidos en la sección de Arrange.
- **Assert section:** parte que se encarga de revisar que el método bajo prueba se comporte como se espera.

D) Basado en la lectura de conceptos sobre pruebas unitarias, responda muy brevemente ¿Para qué sirve establecer un timeout a un caso de prueba?

Los timeouts permiten asignar un tiempo límite de ejecución de los métodos de prueba. Esto sirve en casos donde la velocidad de ejecución es lo que se está probando y se requiere llegar a un tiempo meta para considerar que la prueba es exitosa. Por otro lado, son útiles para evitar que los métodos de prueba se queden en un bucle infinito consumiendo poder de procesamiento. Puede ser que el desarrollador deo un bucle infinito por error y el timeout detiene la ejecución y muestra que se falló la prueba después de cierta cantidad de tiempo.