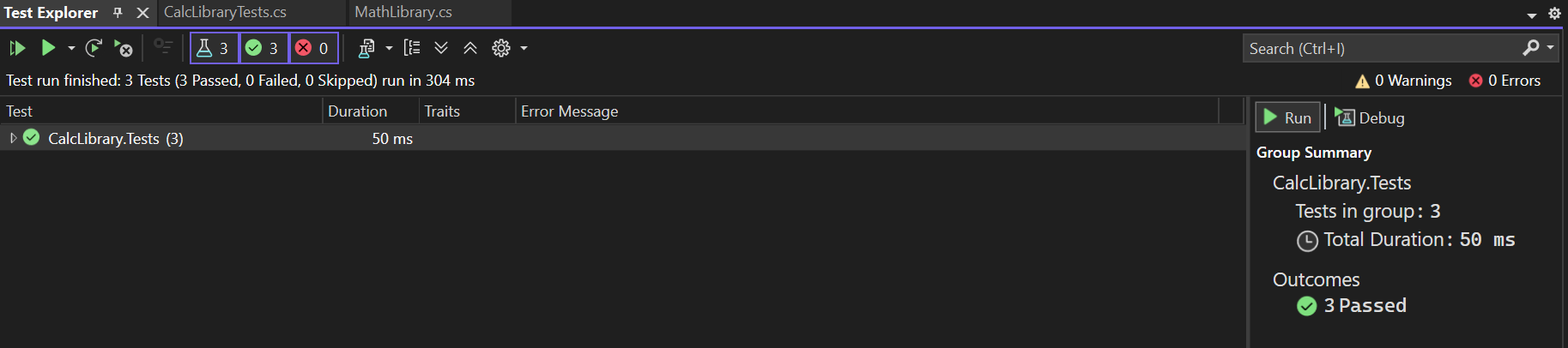
# NUnit and Moq Exercises Solutions

## 1.NUnit - TestFixture & Test

*Code*

|  |
| --- |
| using CalcLibrary; using NUnit.Framework; using NUnit.Framework.Internal.Execution; using System; namespace CalcLibraryTests { [TestFixture] public class Tests { private SimpleCalculator \_calculator;  [SetUp]  public void Setup()  {  \_calculator = new SimpleCalculator();  }   [TearDown]  public void TearDown()  {  \_calculator.AllClear();  }   [TestCase(10, 5, 15)]  [TestCase(-2, -3, -5)]  [TestCase(0, 0, 0)]  public void Addition\_Should\_Return\_Expected\_Result(double a, double b, double expected)  {  var result = \_calculator.Addition(a, b);  Assert.That(result, Is.EqualTo(expected));  } } |

*Output*

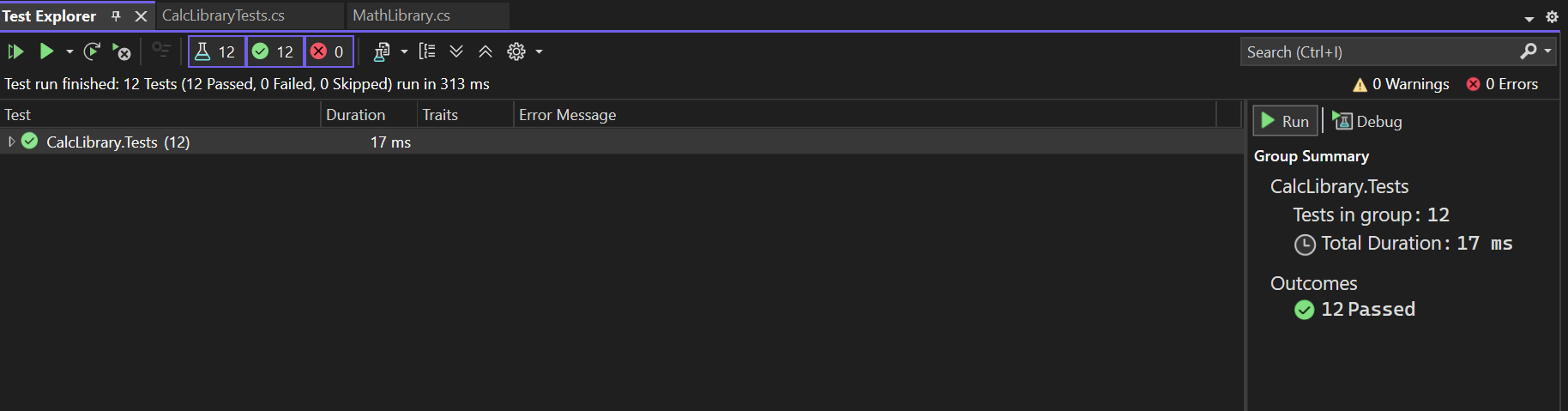


## 2.NUnit – Parameterized Test Cases

*Code*

|  |
| --- |
| using CalcLibrary;  using NUnit.Framework;  using NUnit.Framework.Internal.Execution;  using System;  namespace CalcLibraryTests {  [TestFixture]  public class Tests {  private SimpleCalculator \_calculator;  [SetUp]  public void Setup()  {  \_calculator = new SimpleCalculator();  }   [TearDown]  public void TearDown()  {  \_calculator.AllClear();  }   [TestCase(10, 5, 5)]  [TestCase(-2, -3, 1)]  [TestCase(0, 0, 0)]  public void Subtraction\_ShouldReturnExpectedResult(double a, double b, double expected)  {  var result = \_calculator.Subtraction(a, b);  Assert.AreEqual(expected, result);  }   [TestCase(10, 5, 50)]  [TestCase(-2, -3, 6)]  [TestCase(0, 10, 0)]  public void Multiplication\_ShouldReturnExpectedResult(double a, double b, double expected)  {  var result = \_calculator.Multiplication(a, b);  Assert.AreEqual(expected, result);  }   [TestCase(10, 2, 5)]  [TestCase(-9, -3, 3)]  [TestCase(0, 5, 0)]  public void Division\_ShouldReturnExpectedResult(double a, double b, double expected)  {  var result = \_calculator.Division(a, b);  Assert.AreEqual(expected, result);  }   [Test]  public void Division\_ByZero\_ShouldThrowArgumentException()  {  try  {  \_calculator.Division(5, 0);  Assert.Fail("Division by Zero");  }  catch (ArgumentException e)  {  Assert.AreEqual("Second Parameter Can't be Zero", ex.Message);  }  }   [Test]  public void AllClear\_ShouldResetResultToZero()  {  \_calculator.Addition(10, 5);  \_calculator.AllClear();  Assert.AreEqual(0, \_calculator.GetResult);  }   [Test]  public void TestandClear()  {  double sum = \_calculator.Addition(7, 3);  Assert.AreEqual(10, sum);   \_calculator.AllClear();  Assert.AreEqual(0, \_calculator.GetResult);  }  } } |

*Output*

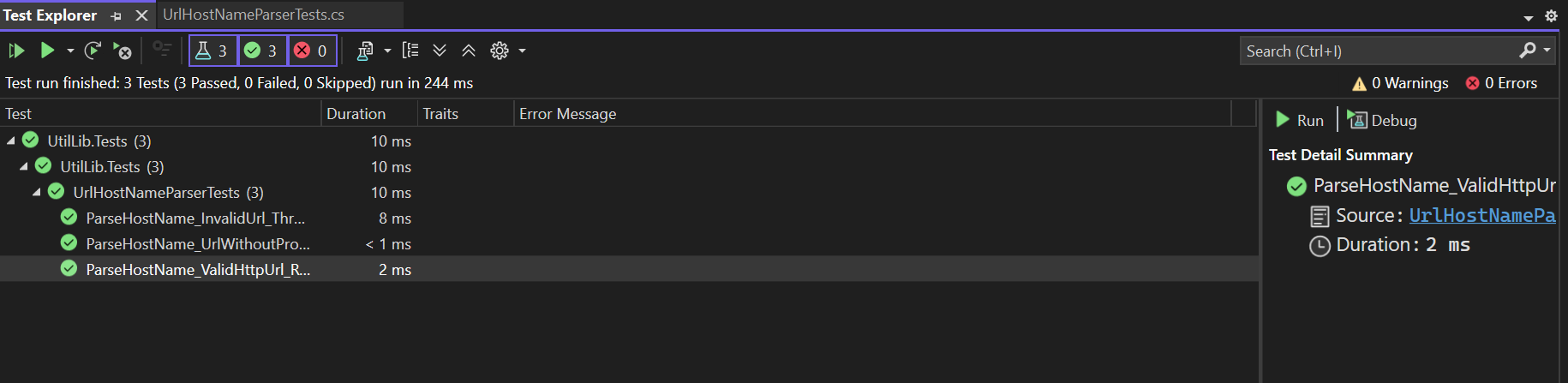


## 3. NUnit – UtilLib Project UrlHostNameParser Class

*Code*

|  |
| --- |
| using NUnit.Framework; using System; using UtilLib;  namespace UtilLib.Tests { [TestFixture] public class UrlHostNameParserTests { private UrlHostNameParser \_parser;  [SetUp]  public void Setup()  {  \_parser = new UrlHostNameParser();  }   [Test]  public void ParseHostName\_ValidHttpUrl\_ReturnsHostName()  {  string url = "<http://www.example.com/api/health>";  string expected = "[www.example.com](https://www.example.com)";   string actual = \_parser.ParseHostName(url);   Assert.That(actual, Is.EqualTo(expected));  }   [Test]  public void ParseHostName\_InvalidUrl\_ThrowsFormatException()  {  string url = "<ftp://example.com>";   var ex = Assert.Throws<FormatException>(() => \_parser.ParseHostName(url));  Assert.That(ex.Message, Is.EqualTo("Url is not in correct format"));  }   [Test]  public void ParseHostName\_UrlWithoutProtocol\_ThrowsFormatException()  {  string url = "[www.example.com](https://www.example.com)";   var ex = Assert.Throws<FormatException>(() => \_parser.ParseHostName(url));  Assert.That(ex.Message, Is.EqualTo("Url is not in correct format"));  } }  } |

*Output*

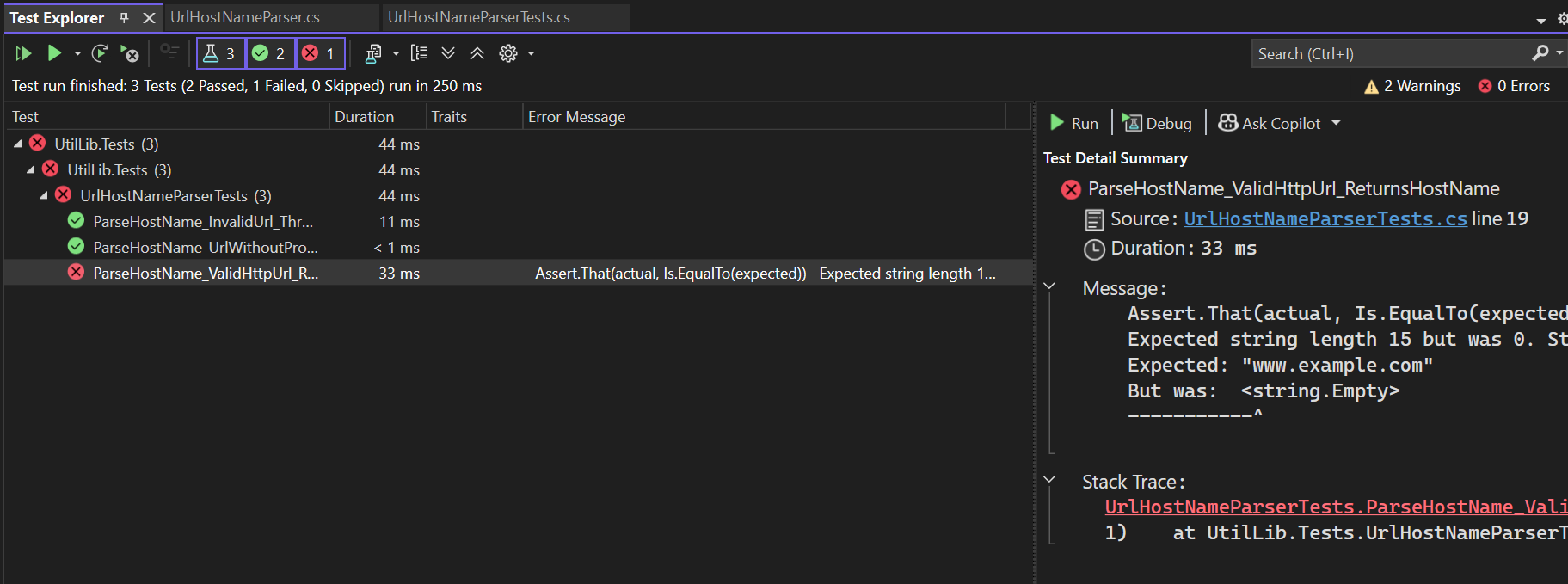


### Breaking the test by modifying the source code and observing the test results

*Code (UrlHostNameParserTests class is same, only change made is in UrlHostNameParser)*

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;    namespace UtilLib  {  public class UrlHostNameParser  {  public string ParseHostName(string url)  {  string protocol = url.Split(':')[0];    if (protocol.Equals("http")|| protocol.Equals("https"))  {  string hostName = url.Split(':')[1].Substring(1).Split('/')[0]; //Changed Substring(2) to Substring(1)    return hostName;    }  else  {  throw new FormatException("Url is not in correct format");  }  }  }  } |

*Output*

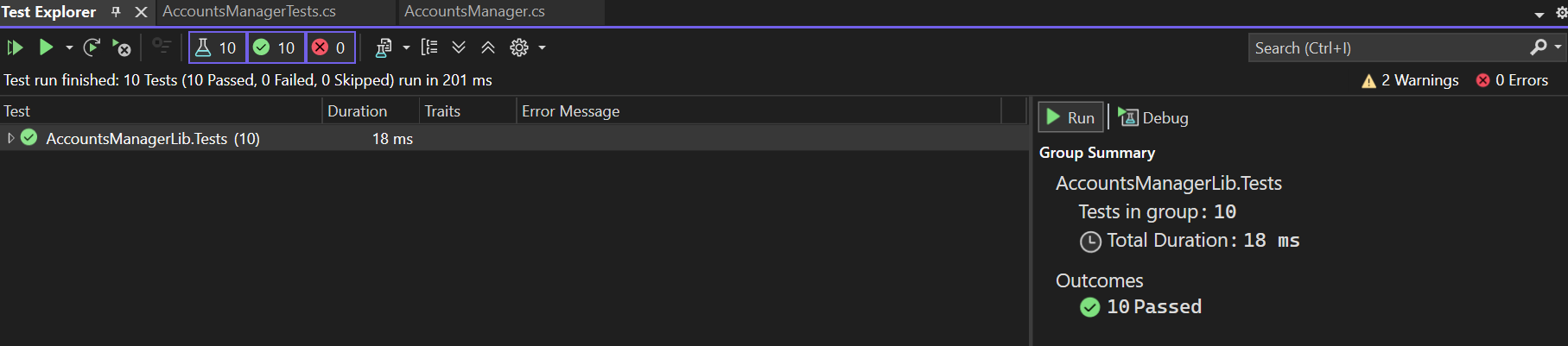


## 4.NUnit – AccountManagerLib Project AccountManager Class

*Code*

|  |
| --- |
| using NUnit.Framework; using System; using AccountsManagerLib;  namespace AccountsManagerLib.Tests { [TestFixture] public class AccountsManagerTests { private AccountsManager \_accountsManager;  [SetUp]   public void Setup()  {  \_accountsManager = new AccountsManager();  }   // Test Cases if provided null credentials  [Test]  [Description("Verifies that ValidateUser throws FormatException when userId is empty.")]  public void ValidateUser\_EmptyUserId\_ThrowsFormatException()  {  string userId = "";  string password = "anypassword";  string expectedErrorMessage = "Both user id and password are mandatory";   var ex = Assert.Throws<FormatException>(() => \_accountsManager.ValidateUser(userId, password));  Assert.That(ex.Message, Is.EqualTo(expectedErrorMessage),  "Expected FormatException with specific message for empty userId.");  }   [Test]  [Description("Verifies that ValidateUser throws FormatException when userId is null.")]  public void ValidateUser\_NullUserId\_ThrowsFormatException()  {  string userId = null;  string password = "anypassword";  string expectedErrorMessage = "Both user id and password are mandatory";   var ex = Assert.Throws<FormatException>(() => \_accountsManager.ValidateUser(userId, password));  Assert.That(ex.Message, Is.EqualTo(expectedErrorMessage),  "Expected FormatException with specific message for null userId.");  }   [Test]  [Description("Verifies that ValidateUser throws FormatException when password is empty.")]  public void ValidateUser\_EmptyPassword\_ThrowsFormatException()  {  string userId = "anyuser";  string password = "";  string expectedErrorMessage = "Both user id and password are mandatory";   var ex = Assert.Throws<FormatException>(() => \_accountsManager.ValidateUser(userId, password));  Assert.That(ex.Message, Is.EqualTo(expectedErrorMessage),  "Expected FormatException with specific message for empty password.");  }   [Test]  [Description("Verifies that ValidateUser throws FormatException when password is null.")]  public void ValidateUser\_NullPassword\_ThrowsFormatException()  {  string userId = "anyuser";  string password = null;  string expectedErrorMessage = "Both user id and password are mandatory";   var ex = Assert.Throws<FormatException>(() => \_accountsManager.ValidateUser(userId, password));  Assert.That(ex.Message, Is.EqualTo(expectedErrorMessage),  "Expected FormatException with specific message for null password.");  }   // Test Cases for Valid Credentials  [Test]  [Description("Verifies successful login for user\_11 with correct password.")]  public void ValidateUser\_ValidUser11Credentials\_ReturnsWelcomeMessage()  {  string userId = "user\_11";  string password = "secret@user11";  string expectedOutput = "Welcome user\_11!!!";   string actualOutput = \_accountsManager.ValidateUser(userId, password);   Assert.That(actualOutput, Is.EqualTo(expectedOutput),  $"Expected welcome message for {userId}.");  }   [Test]  [Description("Verifies successful login for user\_22 with correct password.")]  public void ValidateUser\_ValidUser22Credentials\_ReturnsWelcomeMessage()  {  string userId = "user\_22";  string password = "secret@user22";  string expectedOutput = "Welcome user\_22!!!";   string actualOutput = \_accountsManager.ValidateUser(userId, password);   Assert.That(actualOutput, Is.EqualTo(expectedOutput),  $"Expected welcome message for {userId}.");  }   // Test Cases for Invalid Credentials  [Test]  [Description("Verifies that ValidateUser returns 'Invalid user id/password' for incorrect user\_11 password.")]  public void ValidateUser\_IncorrectUser11Password\_ReturnsInvalidMessage()  {  string userId = "user\_11";  string password = "wrongpassword";  string expectedOutput = "Invalid user id/password";   string actualOutput = \_accountsManager.ValidateUser(userId, password);   Assert.That(actualOutput, Is.EqualTo(expectedOutput),  "Expected 'Invalid user id/password' for incorrect password.");  }   [Test]  [Description("Verifies that ValidateUser returns 'Invalid user id/password' for incorrect user\_22 password.")]  public void ValidateUser\_IncorrectUser22Password\_ReturnsInvalidMessage()  {  string userId = "user\_22";  string password = "badpassword";  string expectedOutput = "Invalid user id/password";   string actualOutput = \_accountsManager.ValidateUser(userId, password);   Assert.That(actualOutput, Is.EqualTo(expectedOutput),  "Expected 'Invalid user id/password' for incorrect password.");  }   [Test]  [Description("Verifies that ValidateUser returns 'Invalid user id/password' for an unknown user.")]  public void ValidateUser\_UnknownUser\_ReturnsInvalidMessage()  {  string userId = "user\_99";   string password = "somepassword";  string expectedOutput = "Invalid user id/password";   string actualOutput = \_accountsManager.ValidateUser(userId, password);   Assert.That(actualOutput, Is.EqualTo(expectedOutput),  "Expected 'Invalid user id/password' for an unknown user.");  }   // Edge Case: Checks the case senstivity  [Test]  [Description("Verifies that ValidateUser fails when userId is correct but case is different.")]  public void ValidateUser\_ValidUser11Credentials\_CaseMismatch\_ReturnsInvalidMessage()  {  string userId = "User\_11";  string password = "secret@user11";  string expectedOutput = "Invalid user id/password";   string actualOutput = \_accountsManager.ValidateUser(userId, password);   Assert.That(actualOutput, Is.EqualTo(expectedOutput),  "Expected 'Invalid user id/password' for userId with case mismatch.");  } }  } |

*Output*

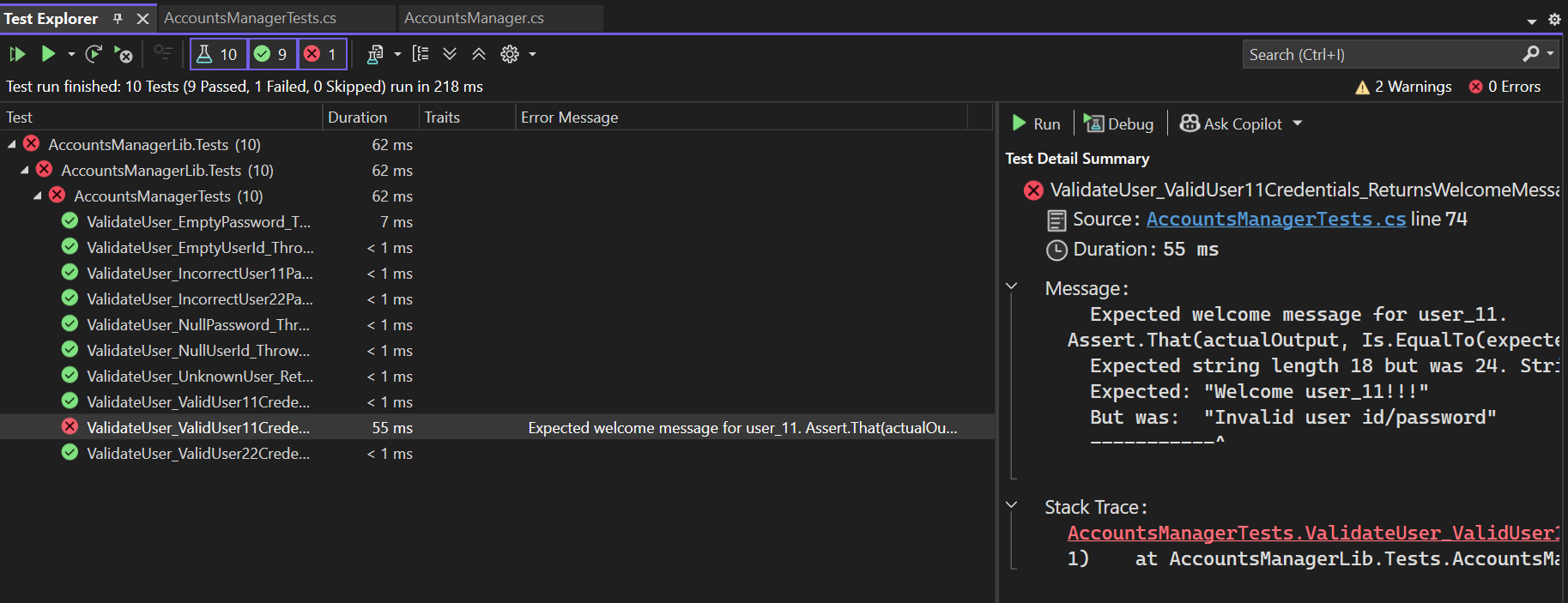


### Breaking the test by modifying the source project functionality.

*Code*

|  |
| --- |
| using System; using System.Collections.Generic; using System.Linq; using System.Text; using System.Threading.Tasks;  namespace AccountsManagerLib { public class AccountsManager { public string ValidateUser(string userId, string password) { string outputMsg;  if (string.IsNullOrEmpty(userId) || string.IsNullOrEmpty(password))  {  throw new FormatException("Both user id and password are mandatory");  }   //changed the password below to secret@wrong11 from secret@user11  if ((userId.Equals("user\_11") && password.Equals("secret@wrong11")) || (userId.Equals("user\_22") && password.Equals("secret@user22")))  {  outputMsg = string.Format("Welcome {0}!!!", userId);  }  else  {  outputMsg = "Invalid user id/password";  }   return outputMsg;  } }  } |

*Output*



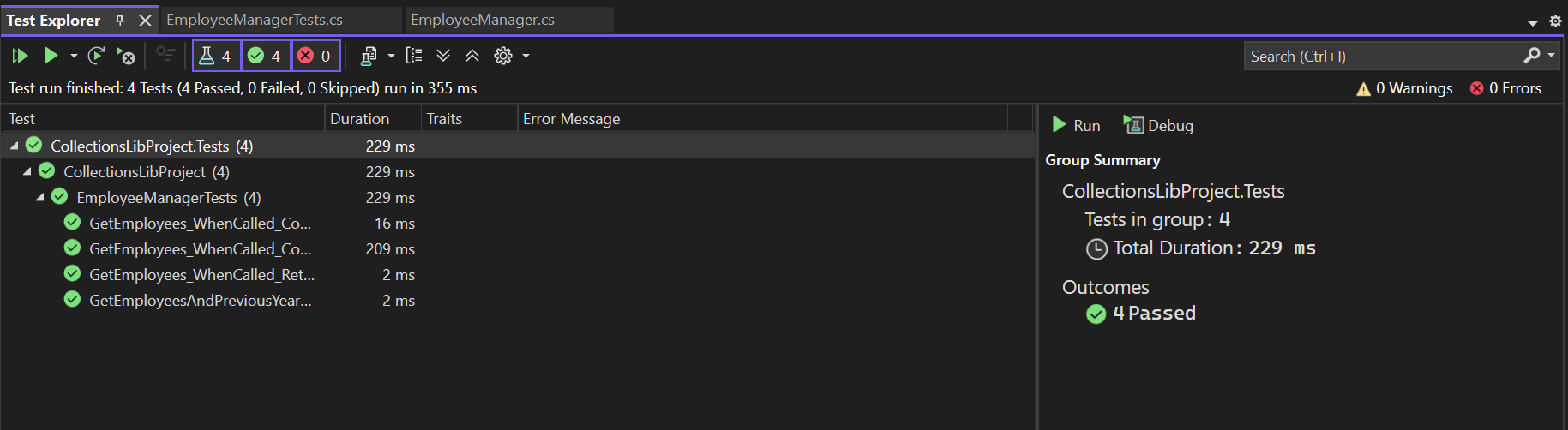
*AccountManagerTests.cs remains the same, with only changes made to AccountManager.cs, i.e. changed the* ***user\_11*** *password from* ***secret@user11*** *to* ***secret@wrong11****, hence highlighting the exact discrepancy between the expected and actual results, which proves the tests are effective at catching regressions and* *breaking the test.*

## 5.NUnit CollectionsLib Project – EmployeeManager

Code

|  |
| --- |
| using CollectionsLib; using NUnit.Framework; using NUnit.Framework.Legacy; using System;  namespace CollectionsLibProject { [TestFixture] public class EmployeeManagerTests { private EmployeeManager \_employeeManager; [SetUp] public void Setup() { \_employeeManager = new EmployeeManager(); }  // No null items in collection  [Test]  [Description("Verifies that the collection returned by GetEmployees contains no null items.")]  public void GetEmployees\_WhenCalled\_ContainsNoNullItems()  {  List<Employee> employees = \_employeeManager.GetEmployees();   CollectionAssert.AllItemsAreNotNull(employees,  "The employee collection should not contain any null items.");   Assert.That(employees, Has.None.Null);  }   //Verify employee with ID 100 exists  [Test]  [Description("Verifies that an employee with ID 100 exists in the collection.")]  public void GetEmployees\_WhenCalled\_ContainsEmployeeWithId100()  {  List<Employee> employees = \_employeeManager.GetEmployees();   Assert.That(employees, Has.Some.Property("EmpId").EqualTo(100),  "The collection should contain an employee with ID 100.");  }   //Check if the collection is unique  [Test]  [Description("Verifies that the collection returned by GetEmployees contains only unique employees.")]  public void GetEmployees\_WhenCalled\_ReturnsAUniqueCollection()  {  List<Employee> employees = \_employeeManager.GetEmployees();   CollectionAssert.AllItemsAreUnique(employees,  "All employees in the collection should be unique based on their EmpId.");   Assert.That(employees, Is.Unique,  "All employees in the collection should be unique based on their EmpId.");  }   //Compare two collections  [Test]  [Description("Verifies GetEmployees and GetEmployeesWhoJoinedInPreviousYears are the same collection.")]  public void GetEmployeesAndPreviousYearEmployees\_WhenCompared\_AreEquivalent()  {  List<Employee> allEmployees = \_employeeManager.GetEmployees();  List<Employee> previousYearEmployees = \_employeeManager.GetEmployeesWhoJoinedInPreviousYears();   CollectionAssert.AreEquivalent(allEmployees, previousYearEmployees,  "The complete employee list and the filtered list should be equivalent.");   Assert.That(allEmployees, Is.EquivalentTo(previousYearEmployees),  "The complete employee list and the filtered list should be equivalent.");  } }  } |

Output

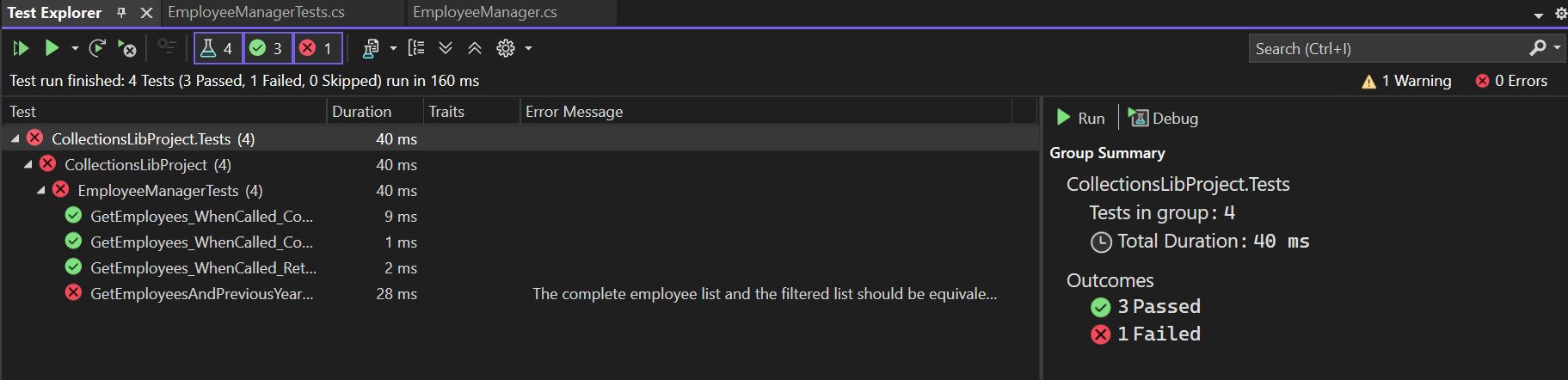


### After adding a new Employee

Code

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;    namespace CollectionsLib  {  public class Employee  {  public int EmpId { get; set; }  public string EmpName { get; set; }  public double Salary { get; set; }  public DateTime DOJ { get; set; }    public override bool Equals(object obj)  {  if(obj == null || !(obj is Employee))  {  return false;  }    Employee other = (Employee)obj;  return this.EmpId == other.EmpId;  }    public override int GetHashCode()  {  return this.EmpId.GetHashCode();  }  }    public class EmployeeManager  {  private static readonly List<Employee> employees;    static EmployeeManager()  {  employees = new List<Employee>  {  new Employee { EmpId=100, EmpName="John",DOJ=DateTime.Now.AddYears(-5),Salary=30000},  new Employee { EmpId=101, EmpName="Mary",DOJ=DateTime.Now.AddYears(-2),Salary=10000},  new Employee { EmpId=102, EmpName="Steve",DOJ=DateTime.Now.AddYears(-2),Salary=10000},  new Employee { EmpId=103, EmpName="Allen",DOJ=DateTime.Now.AddYears(-7),Salary=50000},    //new employee added below  new Employee { EmpId=104, EmpName="Jane", DOJ=DateTime.Now, Salary=60000}  };  }    public List<Employee> GetEmployees()  {  return employees;  }  public List<Employee> GetEmployeesWhoJoinedInPreviousYears()  {  int currentYear = DateTime.Now.Year;  return employees.FindAll(x=>x.DOJ.Year<currentYear);  }  }  } |

Output



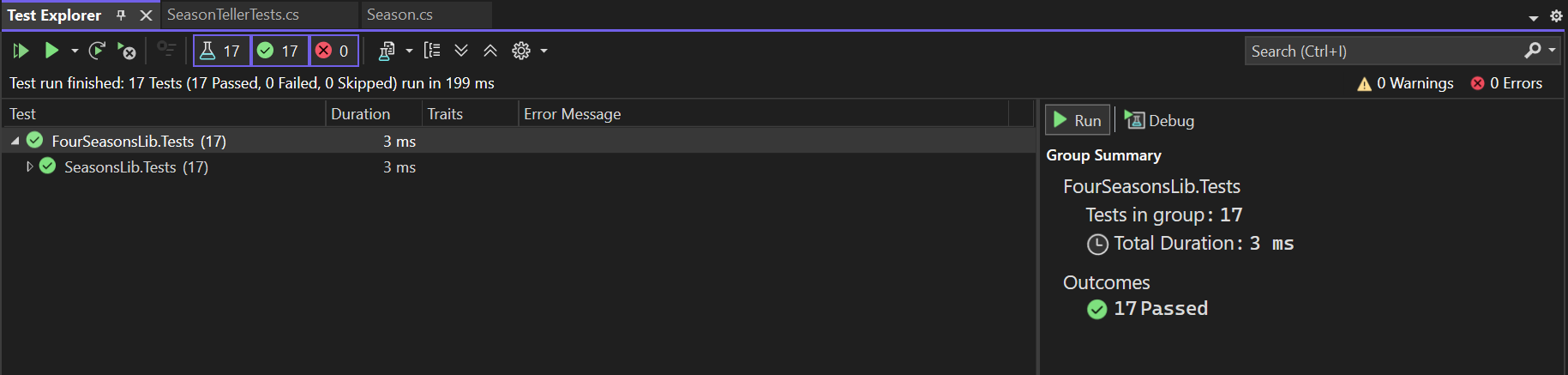
*Thus, after adding the new employee, the previous year list was not equal to the current year, and the test case failed as shown.*

## 6.NUnit FourSeasonsLib Project – SeasonTeller Class

*Code*

|  |
| --- |
| using NUnit.Framework; using SeasonsLib; using System.Collections.Generic;  namespace SeasonsLib.Tests { [TestFixture] public class SeasonTellerTests { private static readonly object[] ValidMonthTestCases = { new object[] { "February", "Spring" }, new object[] { "March", "Spring" }, new object[] { "April", "Summer" }, new object[] { "May", "Summer" }, new object[] { "June", "Summer" }, new object[] { "July", "Monsoon" }, new object[] { "August", "Monsoon" }, new object[] { "September", "Monsoon" }, new object[] { "October", "Autumn" }, new object[] { "November", "Autumn" }, new object[] { "December", "Winter" }, new object[] { "January", "Winter" }, };  private static IEnumerable<TestCaseData> InvalidMonthTestCases()  {  yield return new TestCaseData("Hello", "Invalid Season");  yield return new TestCaseData("Monsoon", "Invalid Season");  yield return new TestCaseData("", "Invalid Season");  yield return new TestCaseData("Sept", "Invalid Season");  yield return new TestCaseData("Febr", "Invalid Season");  }   [Test, TestCaseSource(nameof(ValidMonthTestCases))]  public void DisplaySeasonBy\_ValidMonth\_ReturnsCorrectSeason(string monthName, string expectedSeason)  {  var seasonTeller = new SeasonTeller();  var actual = seasonTeller.DisplaySeasonBy(monthName);  Assert.That(actual, Is.EqualTo(expectedSeason));  }   [Test, TestCaseSource(nameof(InvalidMonthTestCases))]  public void DisplaySeasonBy\_InvalidMonth\_ReturnsInvalidSeason(string monthName, string expectedMessage)  {  var seasonTeller = new SeasonTeller();  var actual = seasonTeller.DisplaySeasonBy(monthName);  Assert.That(actual, Is.EqualTo(expectedMessage));  } }  } |

*Output*



As mentioned in HandsOn, the two ways of **TestCaseSource** have been used, i.e.

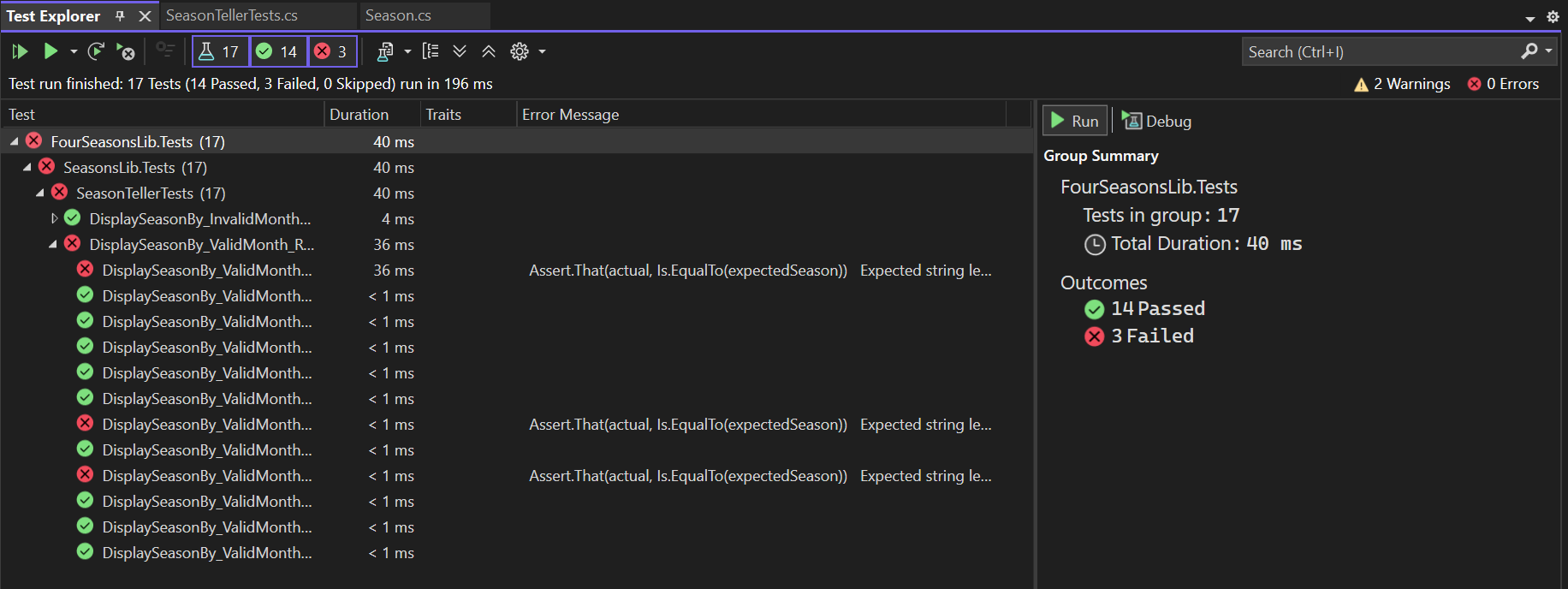
* Straightforward way: used in **ValidMonthTestCases**
* Alternate way: used in **InvalidMonthTestCases**

### Breaking the Tests

*Code*

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;    namespace SeasonsLib  {  public class SeasonTeller  {  public string DisplaySeasonBy(string monthName)  {  string seasonName;    if (monthName.Equals("February", StringComparison.OrdinalIgnoreCase) || monthName.Equals("March", StringComparison.OrdinalIgnoreCase))  {  seasonName = "Spring";  }  else if (monthName.Equals("April", StringComparison.OrdinalIgnoreCase) || monthName.Equals("May", StringComparison.OrdinalIgnoreCase) || monthName.Equals("June", StringComparison.OrdinalIgnoreCase))  {  seasonName = "Summertime"; //replaced with Summertime instead of Summer  }  else if (monthName.Equals("July", StringComparison.OrdinalIgnoreCase) || monthName.Equals("August", StringComparison.OrdinalIgnoreCase) || monthName.Equals("September", StringComparison.OrdinalIgnoreCase))  {  seasonName = "Monsoon";  }  else if (monthName.Equals("October", StringComparison.OrdinalIgnoreCase) || monthName.Equals("November", StringComparison.OrdinalIgnoreCase))  {    seasonName = "Autumn";    }  else if (monthName.Equals("December", StringComparison.OrdinalIgnoreCase) || monthName.Equals("January", StringComparison.OrdinalIgnoreCase))  {    seasonName = "Winter";    }  else  {  return "Invalid Season";    }    return seasonName;  }  }  } |

*Output*



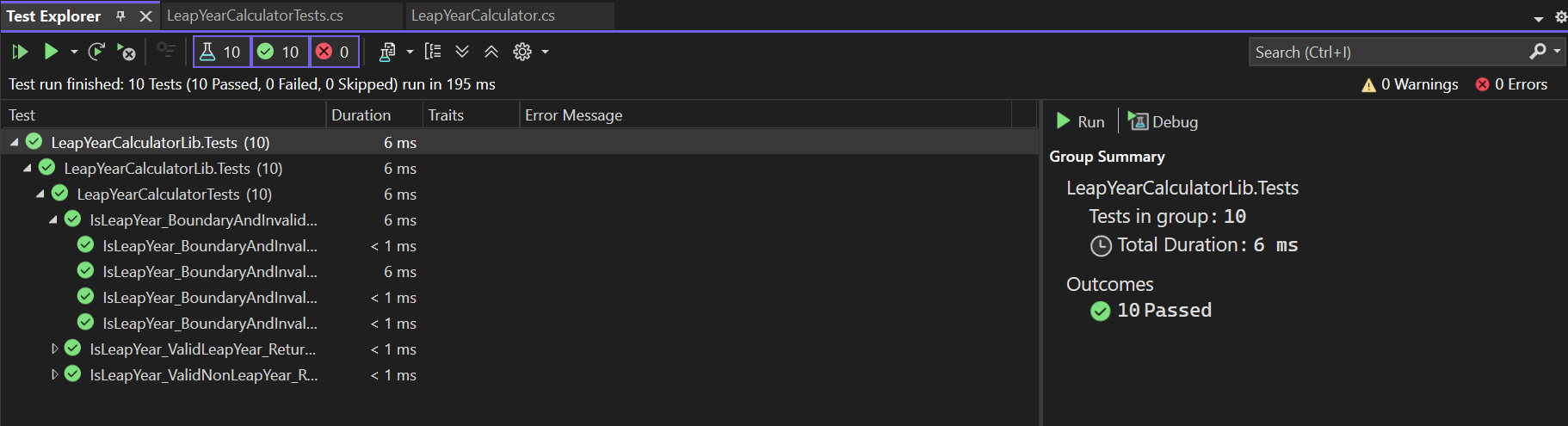
*Here, I changed Summer to Summertime, which led to breaking the Tests.*

## 7.NUnit LeapYearCalculatorLib Project – LeapYearCalulator

*Code*

|  |
| --- |
| using NUnit.Framework; using System;  namespace LeapYearCalculatorLib.Tests { [TestFixture] public class LeapYearCalculatorTests { private LeapYearCalculator \_calculator;  [SetUp]  public void Setup()  {  \_calculator = new LeapYearCalculator();  }   [TestCase(2024, 1)]  [TestCase(2000, 1)]  [TestCase(1600, -1)]  public void IsLeapYear\_ValidLeapYear\_ReturnsOne(int year, int expected)  {  int actual = \_calculator.IsLeapYear(year);  Assert.That(actual, Is.EqualTo(expected));  }   [TestCase(2023, 0)]  [TestCase(1900, 0)]  [TestCase(2100, 0)]  public void IsLeapYear\_ValidNonLeapYear\_ReturnsZero(int year, int expected)  {  int actual = \_calculator.IsLeapYear(year);  Assert.That(actual, Is.EqualTo(expected));  }   [TestCase(1600, -1)]  [TestCase(9999, 0)]  [TestCase(1752, -1)]  [TestCase(10000, -1)]  public void IsLeapYear\_BoundaryAndInvalidCases\_ReturnsExpected(int year, int expected)  {  int actual = \_calculator.IsLeapYear(year);  Assert.That(actual, Is.EqualTo(expected));  } }  } |

*Output*

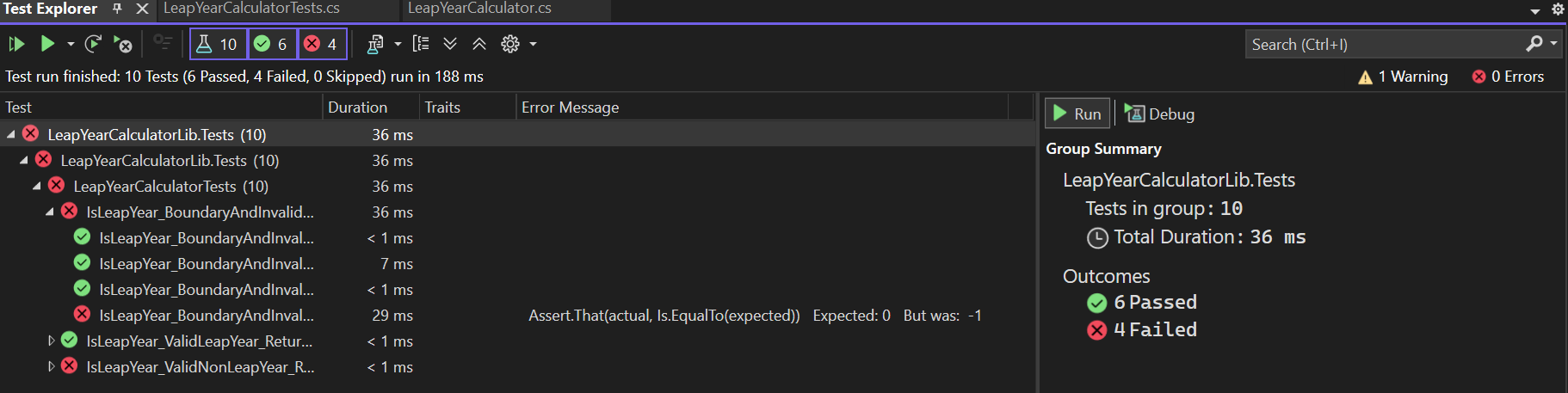


### Breaking the Tests by modifying the source code

Code

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;    namespace LeapYearCalculatorLib  {  public class LeapYearCalculator  {    public int IsLeapYear(int year)  {  int output;    if (year<1753||year>9999)  {  return -1;  }    if (((year % 4 == 0) && (year % 100 != 0)) || (year % 400 == 0))  output = 1;  else  output = -1; //returns -1 instead of 0    return output;    }  }  } |

Output



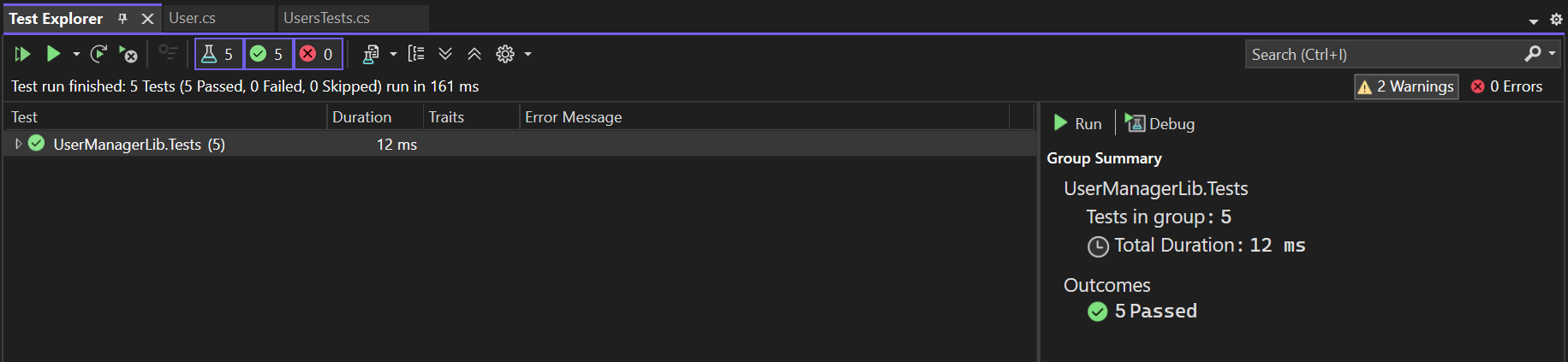
*Changed the return value to –1 from 0, if the year given is not a leap year, which led to breaking of the Tests.*

## 8.Nunit UserManagerLib Project – UserManager Class

*Code*

|  |
| --- |
| using NUnit.Framework; using System;  namespace UserManagerLib.Tests { [TestFixture] public class UsersTests { private User \_user;  [SetUp]  public void Setup()  {  \_user = new User();  }   [Test]  public void CreateUser\_ValidPAN\_NoExceptionThrown()  {  var user = new User()  {  PANCardNo = "ABCDE1234F"  };   Assert.That(() => \_user.CreateUser(user), Throws.Nothing);  }   [TestCase(null)]  [TestCase("")]  public void CreateUser\_NullOrEmptyPAN\_ThrowsNullReferenceException(string? pan)  {  var user = new User  {  PANCardNo = pan  };   Assert.That(() => \_user.CreateUser(user), Throws.Exception.TypeOf<NullReferenceException>());  }   [TestCase("ABC123")] // Less than 10 chars   [TestCase("ABCDEFGHIJKL")] // More than 10 chars   public void CreateUser\_InvalidLengthPAN\_ThrowsFormatException(string pan)  {  var user = new User  {  PANCardNo = pan  };   Assert.That(() => \_user.CreateUser(user), Throws.Exception.TypeOf<FormatException>());  } }  } |

*Output*

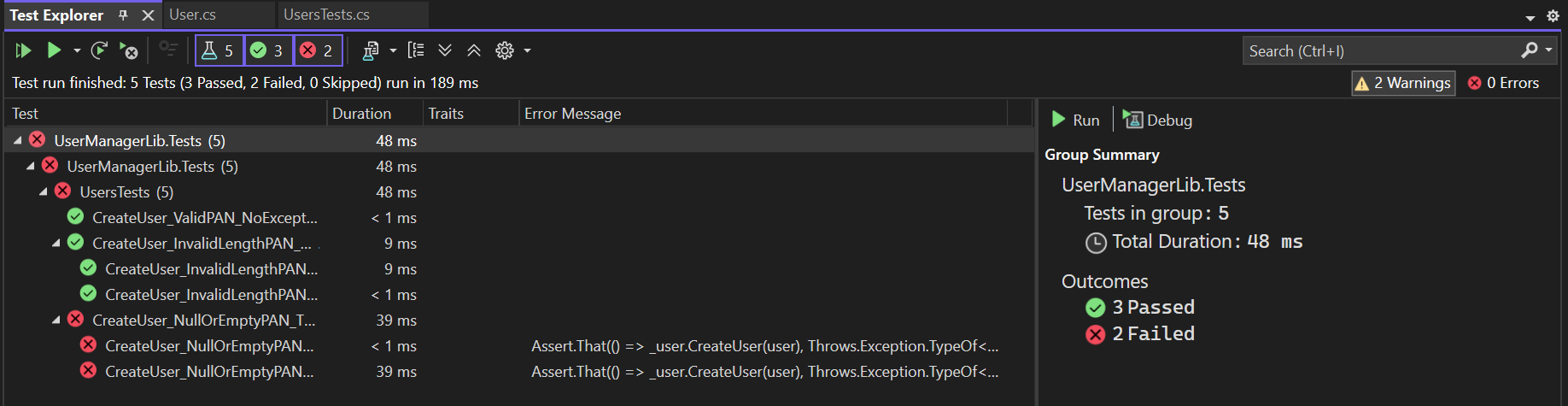


### Breaking the Tests by modifying the source code

Code

|  |
| --- |
| using NUnit.Framework; using System;  namespace UserManagerLib.Tests { [TestFixture] public class UsersTests { private User \_user;  [SetUp]  public void Setup()  {  \_user = new User();  }   [Test]  public void CreateUser\_ValidPAN\_NoExceptionThrown()  {  var user = new User()  {  PANCardNo = "ABCDE1234F"  };   Assert.That(() => \_user.CreateUser(user), Throws.Nothing);  }   [TestCase(null)]  [TestCase("")]  public void CreateUser\_NullOrEmptyPAN\_ThrowsNullReferenceException(string? pan)  {  var user = new User  {  PANCardNo = pan  };   Assert.That(() => \_user.CreateUser(user), Throws.Exception.TypeOf<NullReferenceException>());  }   [TestCase("ABC123")] // Less than 10 chars   [TestCase("ABCDEFGHIJKL")] // More than 10 chars   public void CreateUser\_InvalidLengthPAN\_ThrowsFormatException(string pan)  {  var user = new User  {  PANCardNo = pan  };   Assert.That(() => \_user.CreateUser(user), Throws.Exception.TypeOf<FormatException>());  } }  } |

Output



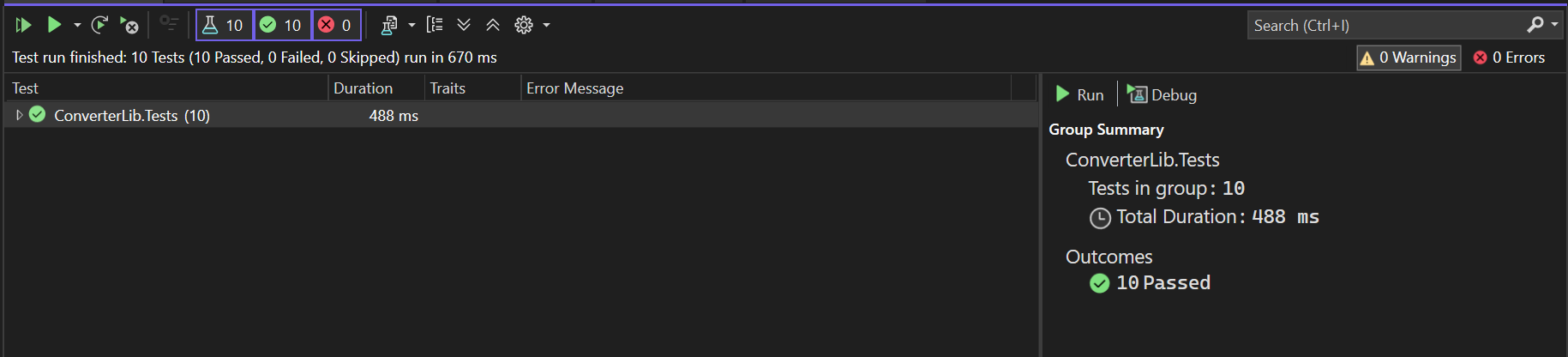
*Changed the to return “Valid” instead of throwing a null exception, leading to breaking the Tests.*

## 9.NUnit ConverterLib Project – ConverterTests Class

Code

|  |
| --- |
| using NUnit.Framework; using Moq; using ConverterLib; using CurrencyConverterApp;  namespace ConverterLib.Tests { [TestFixture] public class ConverterTests { private Mock \_mockRateFeed; private Converter \_converter;  [SetUp]  public void Setup()  {  \_mockRateFeed = new Mock<IDollarToEuroExchangeRateFeed>();  \_converter = new Converter(\_mockRateFeed.Object);  }   [TestCase(100, 0.85, 85)]  [TestCase(50, 0.9, 45)]  public void USDToEuro\_ValidRate\_ReturnsExpected(double dollars, double rate, double expectedEuro)  {  \_mockRateFeed.Setup(feed => feed.GetActualUSDollarValue()).Returns(rate);  double result = \_converter.USDToEuro(dollars);  Assert.That(result, Is.EqualTo(expectedEuro).Within(0.01));  }   [TestCase(0, 273.15)]  [TestCase(-273.15, 0)]  public void CelsiusToKelvin\_ValidInput\_ReturnsKelvin(double celsius, double expected)  {  Assert.That(\_converter.CelsiusToKelvin(celsius), Is.EqualTo(expected).Within(0.01));  }   [TestCase(1, 2.205)]  [TestCase(10, 22.05)]  public void KilogramToPound\_ValidInput\_ReturnsPounds(double kg, double expected)  {  Assert.That(\_converter.KilogramToPound(kg), Is.EqualTo(expected).Within(0.01));  }   [TestCase(1.609, 1)]  [TestCase(16.09, 10)]  public void KilometerToMile\_ValidInput\_ReturnsMiles(double km, double expected)  {  Assert.That(\_converter.KilometerToMile(km), Is.EqualTo(expected).Within(0.01));  }   [TestCase(3.785, 1)]  [TestCase(7.57, 2)]  public void LiterToGallon\_ValidInput\_ReturnsGallons(double liter, double expected)  {  Assert.That(\_converter.LiterToGallon(liter), Is.EqualTo(expected).Within(0.01));  } }  } |

Output

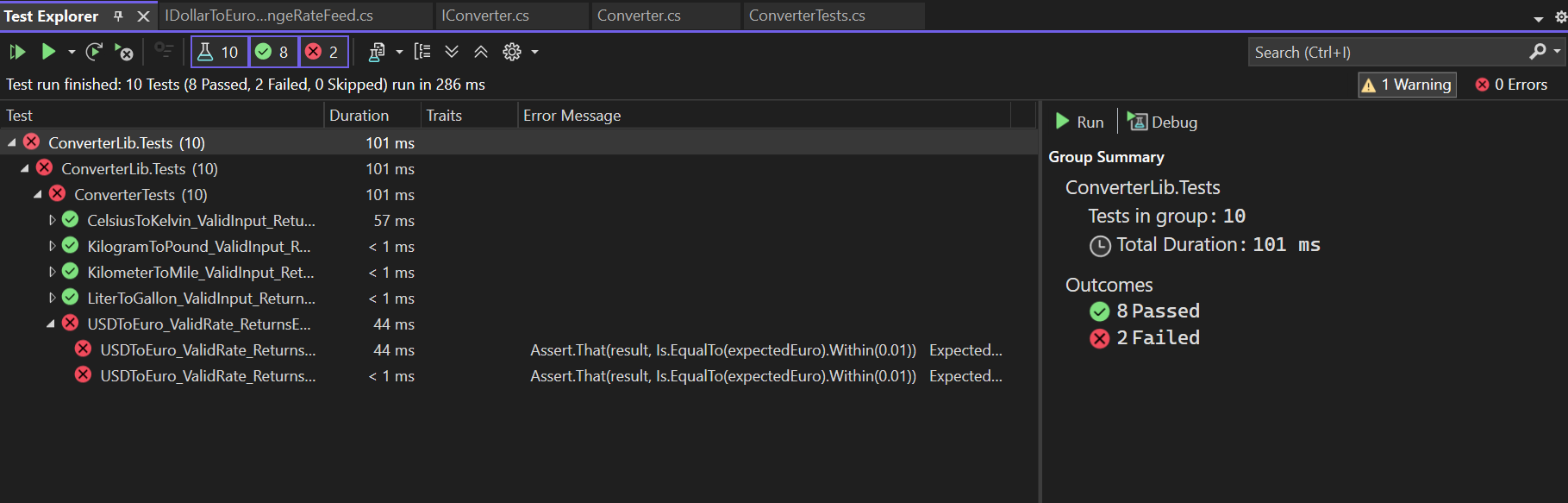


### Breaking the Tests by modifying the source code

Code

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using CurrencyConverterApp;    namespace ConverterLib  {  public class Converter : IConverter  {  IDollarToEuroExchangeRateFeed \_exchangeRateFeed;  public Converter(IDollarToEuroExchangeRateFeed exchangeRateFeed)  {  \_exchangeRateFeed = exchangeRateFeed;  }    public double CelsiusToKelvin(double celsius)  {  return celsius + 273.15;  }    public double KilogramToPound(double kilogram)  {  return kilogram \* 2.205;  }    public double KilometerToMile(double kilometer)  {  return kilometer / 1.609;  }    public double LiterToGallon(double liter)  {  return liter / 3.785;  }    public double USDToEuro(double dollar)  {  return dollar + \_exchangeRateFeed.GetActualUSDollarValue(); //changed \* -> +  }  }  } |

Output



*Changed the USDToEURO logic where*

***return dollar \* \_exchangeRateFeed.GetActualUSDollarValue()******;***

***to***

***return dollar + \_exchangeRateFeed.GetActualUSDollarValue();***

***Which led to breaking of the Tests.***