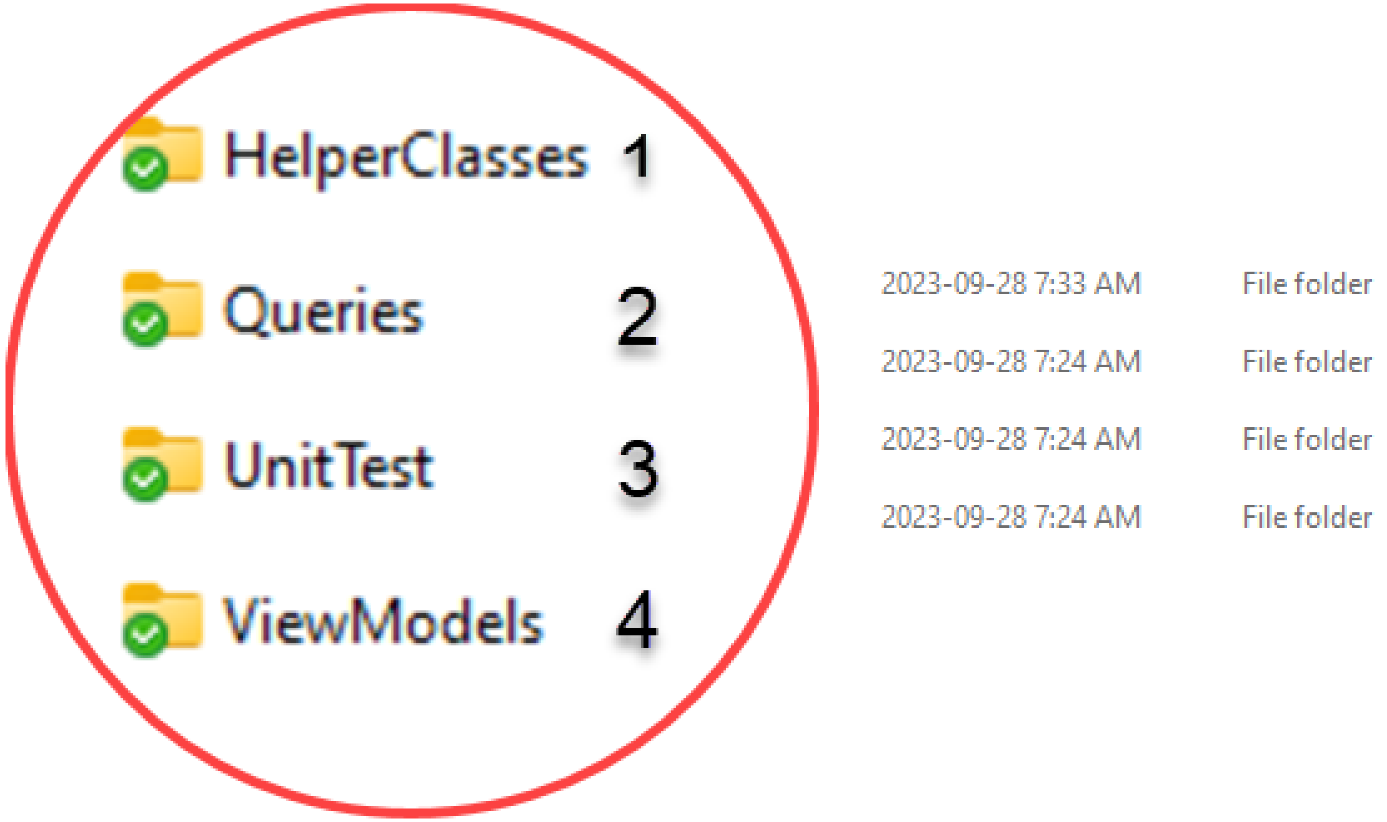
Session 4: Unit Testing in MVVM - A Deep Dive into the Model (Part 2)

Show us the code

Folder Layout

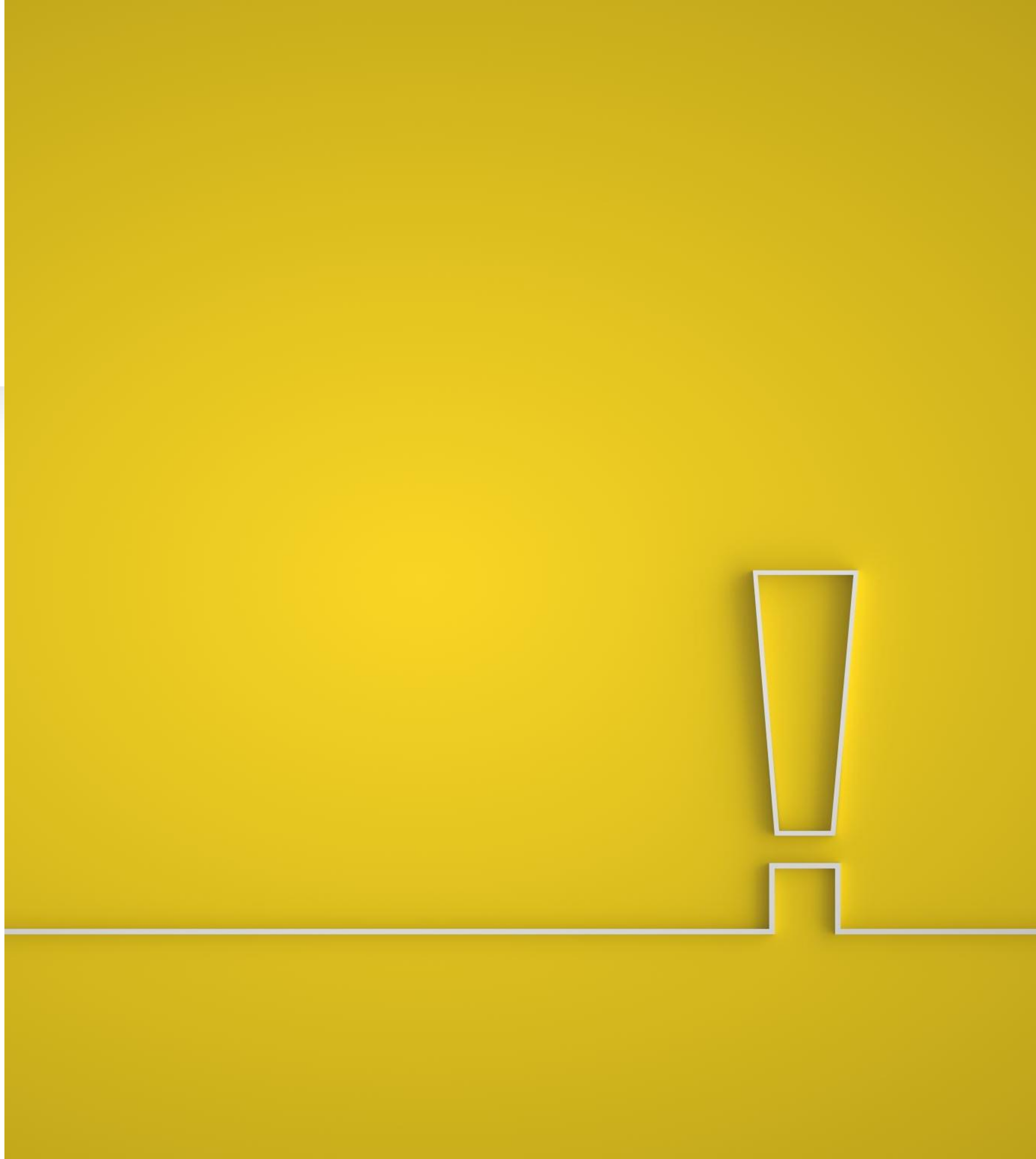
1. HelperClasses: A folder for storing classes that are used to support unit tests.
2. Queries: Where the individual methods are stored
3. UnitTest: Where the unit tests are stored for testing the methods
4. ViewModels: Where the view models/classes are stored.



HelperClasses 1	2023-09-28 7:33 AM	File folder
Queries 2	2023-09-28 7:24 AM	File folder
UnitTest 3	2023-09-28 7:24 AM	File folder
ViewModels 4	2023-09-28 7:24 AM	File folder

Session's Objective

- Install OLTP-DMIT2018
- Customer Search (GetCustomers)
    - Define the rules.
    - Create “Unit Tests” based on the rules using Arrange, Act, and Assert patterns.
    - Create the method for returning a list of customers (GetCustomers).
    - Refactor using Arrange, Act, and Assert patterns.
- Customer Retrieval (GetCustomer)
    - Define the rules.
    - Create “Unit Tests” based on the rules using Arrange, Act, and Assert patterns.
    - Create the method for returning a single customer (GetCustomer).
    - Refactor using Arrange, Act, and Assert patterns.



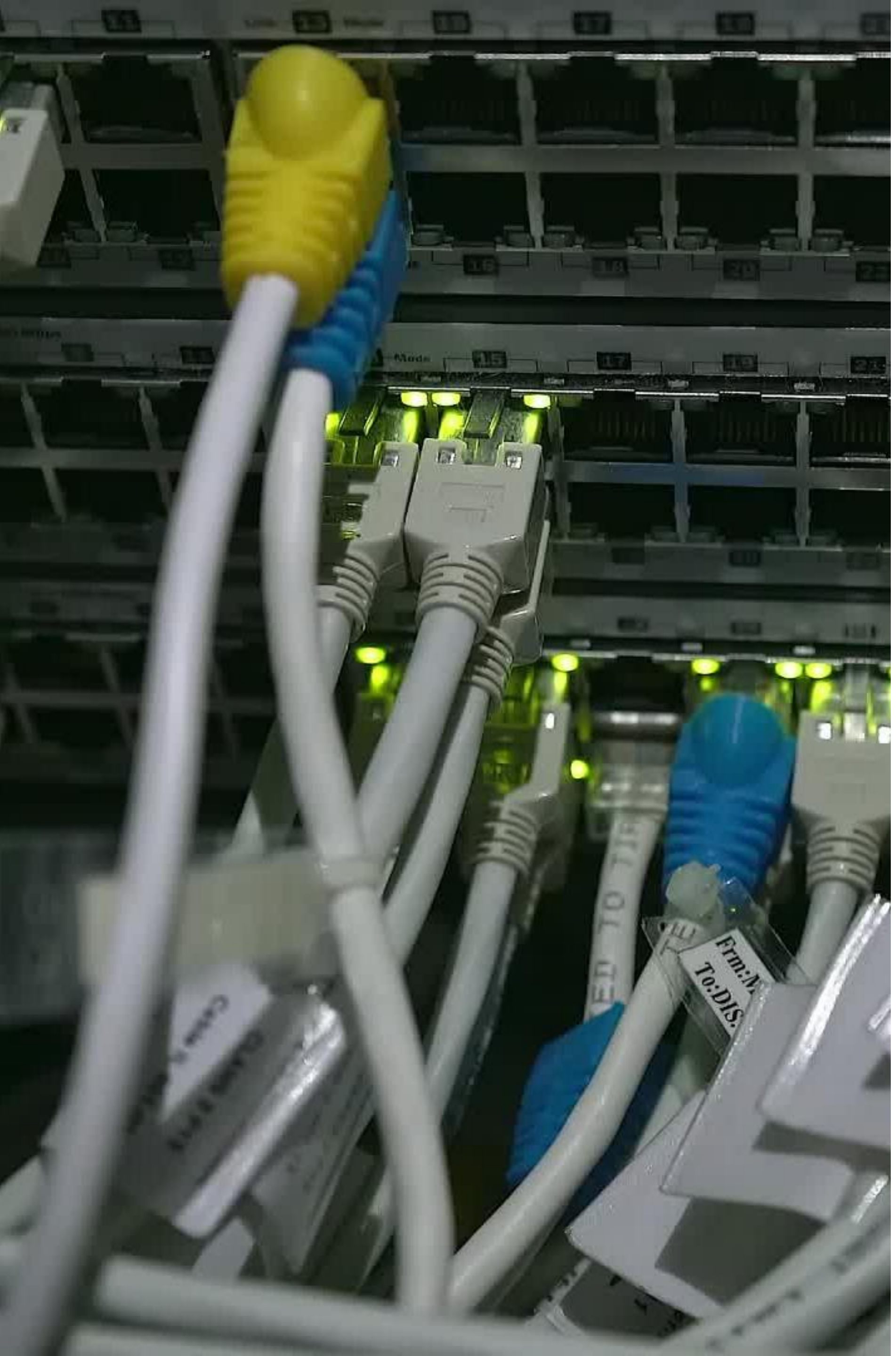
Customer Search - Defining the rules!

What are our rules?

Business Rules

- These are processing rules that need to be satisfied for valid data
- rule: Both the “last name” and “phone number” cannot be empty.

  Only able to retrieve customers if we know who we are searching for.
- rule: The remove from view flag must be false. Otherwise, this is considered a deleted record (Soft Delete).



Install OLTP DMIT2018 Database

1. Download the OLTP DMIT2018 database from Moodle.
2. Install the OLTP-DMIT2018 bacpac file. Ensure that you set the database name to OLTP-DMIT2018.
3. Add a connection in LINQPad to the OLTP-DMIT2018 database.

Creating Our First Test

1. Create a LINQ file for the customer tests (collection of tests for customers)

   UnitTest\CustomerTests.linq

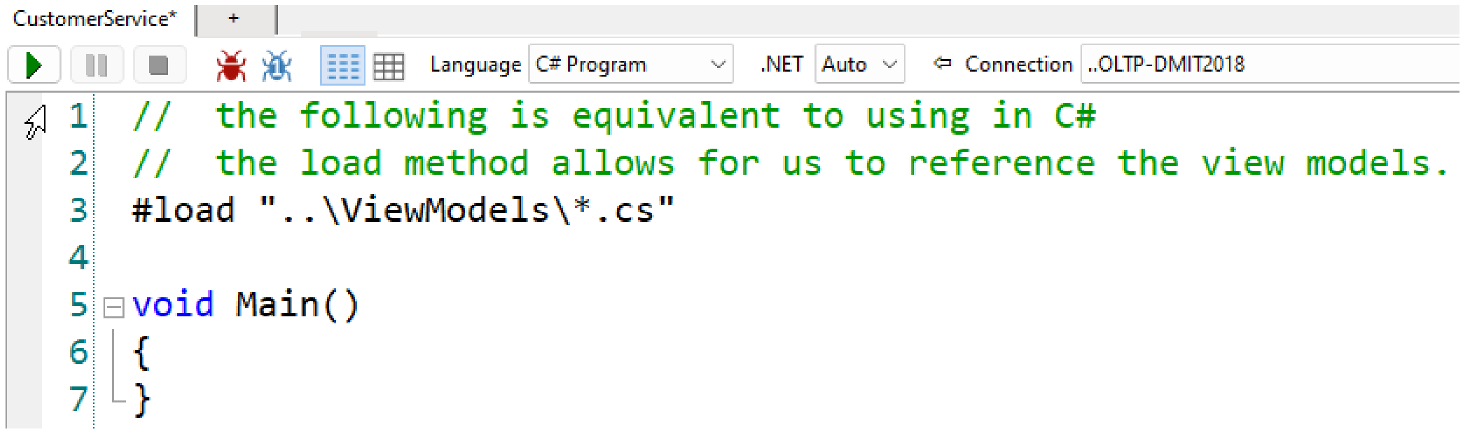
2. Create a LINQ file for “Get Customer” method (searching for customers).

   Queries\CustomerService.linq

NOTE: The CustomerService file will mimic our C# CustomerService.cs file.

Initial Setup of CustomerService

- In LINQPad, select a new query and set the language to "C# Language."
- Set the "Connection" to "OLTP-DMIT2018".
- Add the "#load" method to reference the view model

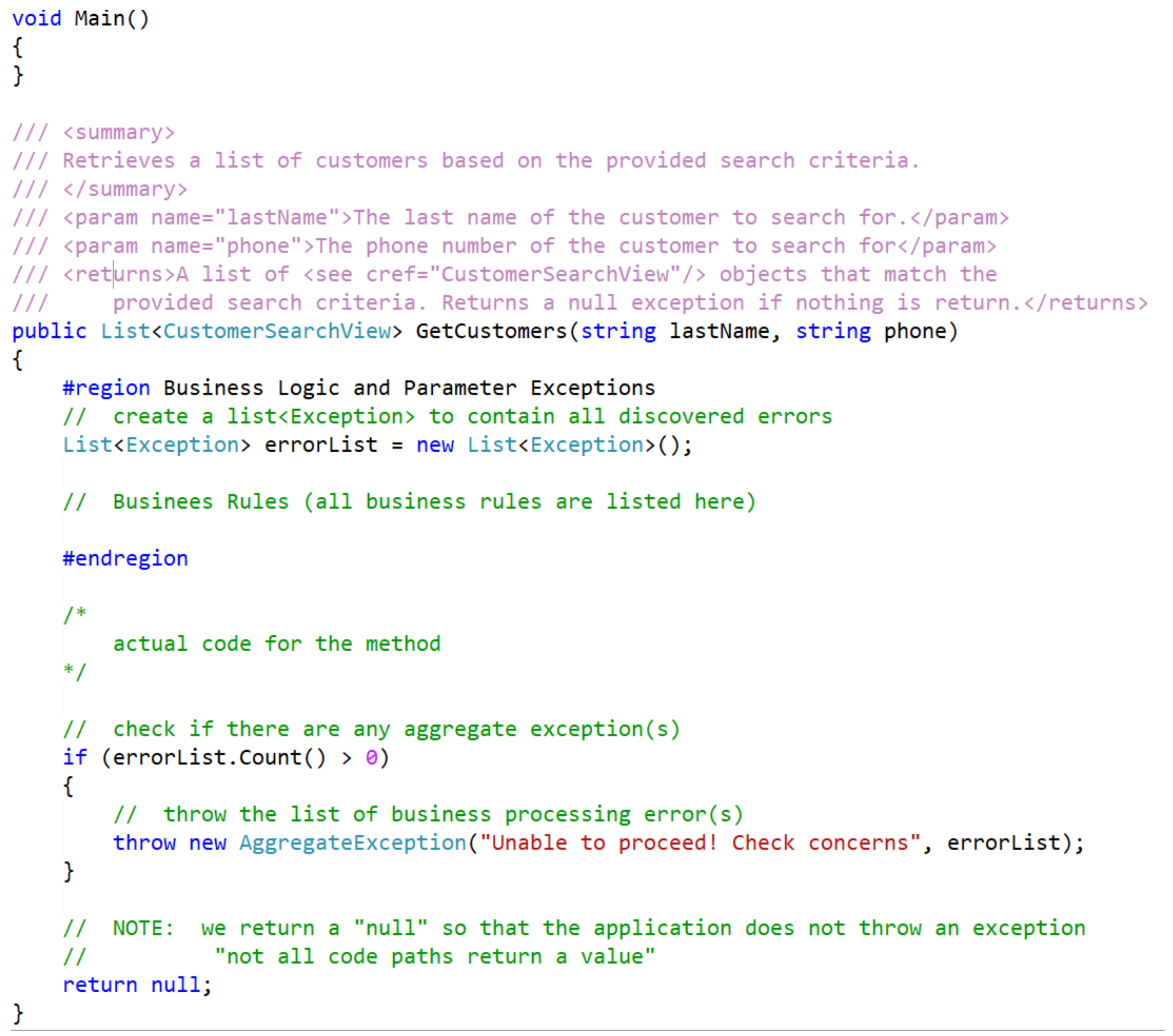


Language: C# Program
Connection: ..OLTP-DMIT2018

```
//  the following is equivalent to using in C#
//  the load method allows for us to reference the view models.
#load "..\ViewModels\*.cs"

void Main()
{
}
```

Build the Skeleton of our GetCustomers Method

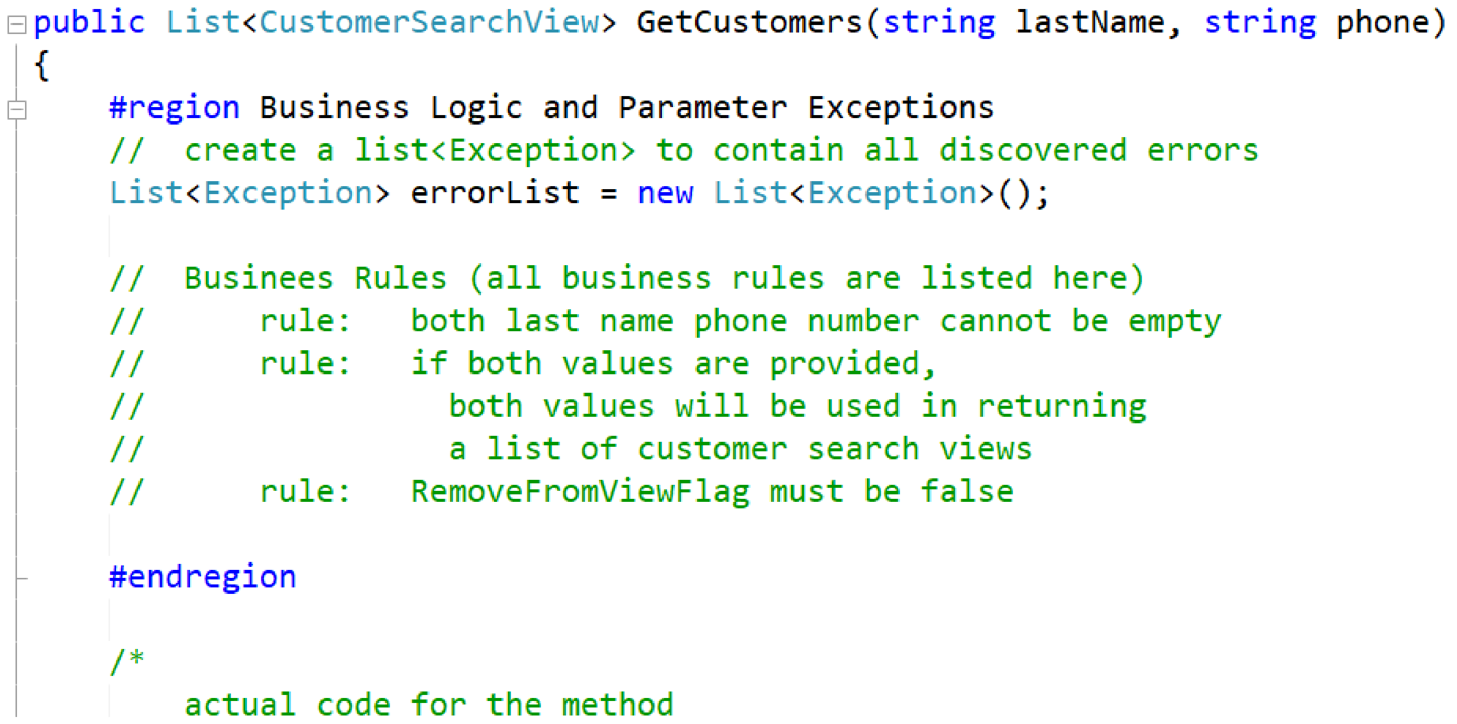
- Create the method signature for the “GetCustomer” method.
- Stub out the “Business Logic and Parameter Exceptions” area
- Add the “AggregateException” check.



```
void Main()
{
}

/// <summary>
/// Retrieves a list of customers based on the provided search criteria.
/// </summary>
/// <param name="lastName">The last name of the customer to search for.</param>
/// <param name="phone">The phone number of the customer to search for</param>
/// <returns>A list of <see cref="CustomerSearchView"/> objects that match the
///     provided search criteria. Returns a null exception if nothing is return.</returns>
public List<CustomerSearchView> GetCustomers(string lastName, string phone)
{
    #region Business Logic and Parameter Exceptions
    //  create a list<Exception> to contain all discovered errors
    List<Exception> errorList = new List<Exception>();

    //  Businees Rules (all business rules are listed here)

    #endregion

    /*
        actual code for the method
    */

    //  check if there are any aggregate exception(s)
    if (errorList.Count() > 0)
    {
        //  throw the list of business processing error(s)
        throw new AggregateException("Unable to proceed! Check concerns", errorList);
    }

    //  NOTE:  we return a "null" so that the application does not throw an exception
    //          "not all code paths return a value"
    return null;
}
```

Adding Our Rules

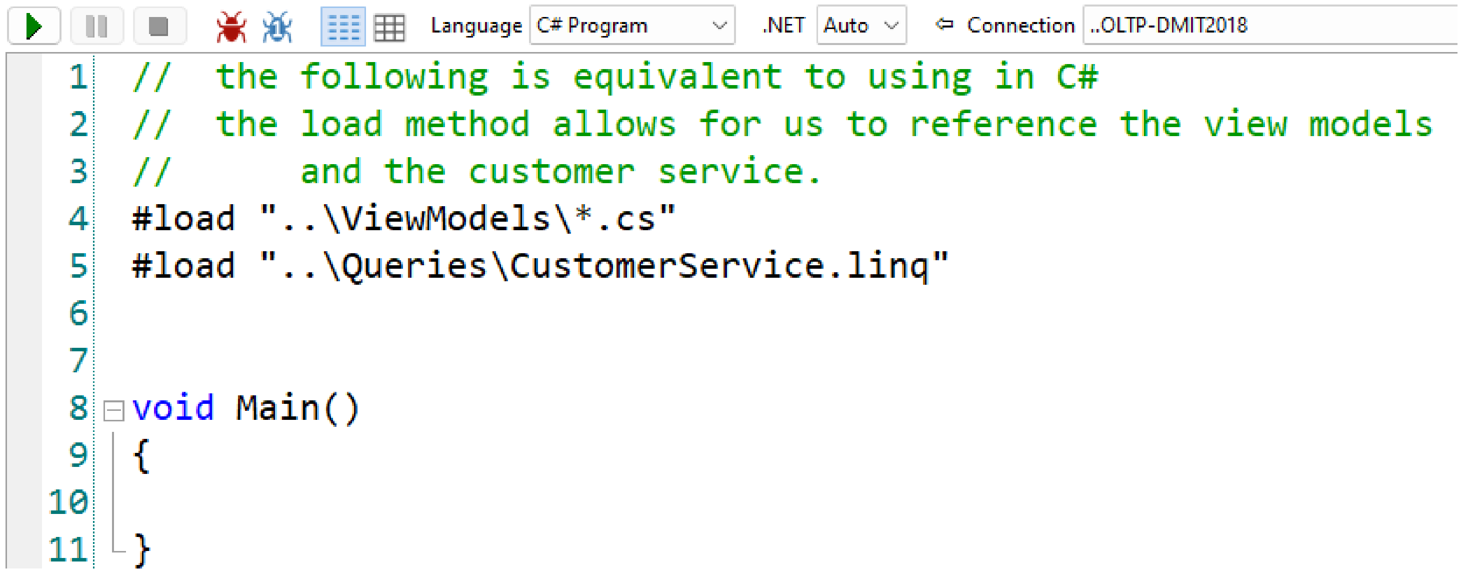
- Both the **last name** and the **phone number** cannot be empty.
- If both values are provided, both values will be used in returning a list of customer search views.
- RemoveFromViewFlag (Soft Delete) must be set to false.



```
public List<CustomerSearchView> GetCustomers(string lastName, string phone)
{
    #region Business Logic and Parameter Exceptions
    //  create a list<Exception> to contain all discovered errors
    List<Exception> errorList = new List<Exception>();

    //  Businees Rules (all business rules are listed here)
    //      rule:   both last name phone number cannot be empty
    //      rule:   if both values are provided,
    //                both values will be used in returning
    //                a list of customer search views
    //      rule:   RemoveFromViewFlag must be false

    #endregion

    /*
       actual code for the method
```

Initial Setup of CustomerService

- In LINQPad, select a new query and set the language to "C# Language."
- Set the "Connection" to "OLTP-DMIT2018".
- Add the "#load" method to reference the view model
    - NOTE: We now included a reference to the CustomerService.linq so that we can have access to the GetCustomer method.

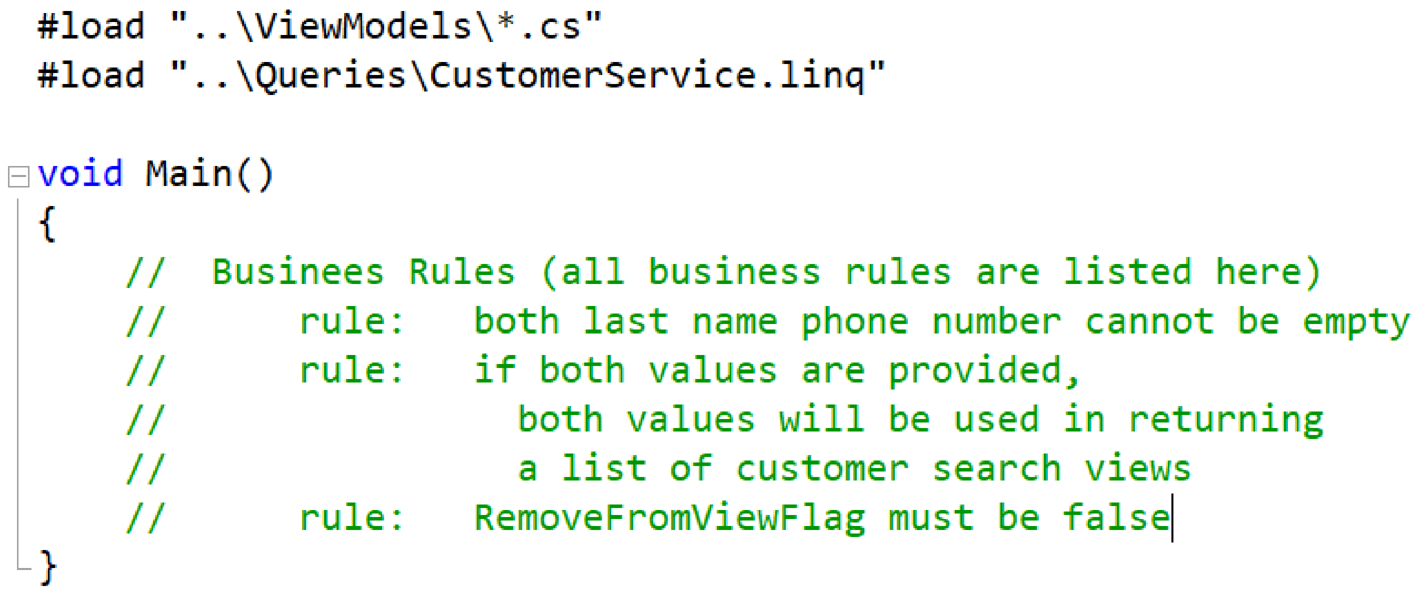


Language C# Program .NET Auto Connection ..OLTP-DMIT2018

```
//  the following is equivalent to using in C#
//  the load method allows for us to reference the view models
//      and the customer service.
#load "..\ViewModels\*.cs"
#load "..\Queries\CustomerService.linq"


void Main()
{

}
```

List All Requirements and Tests that are needed.

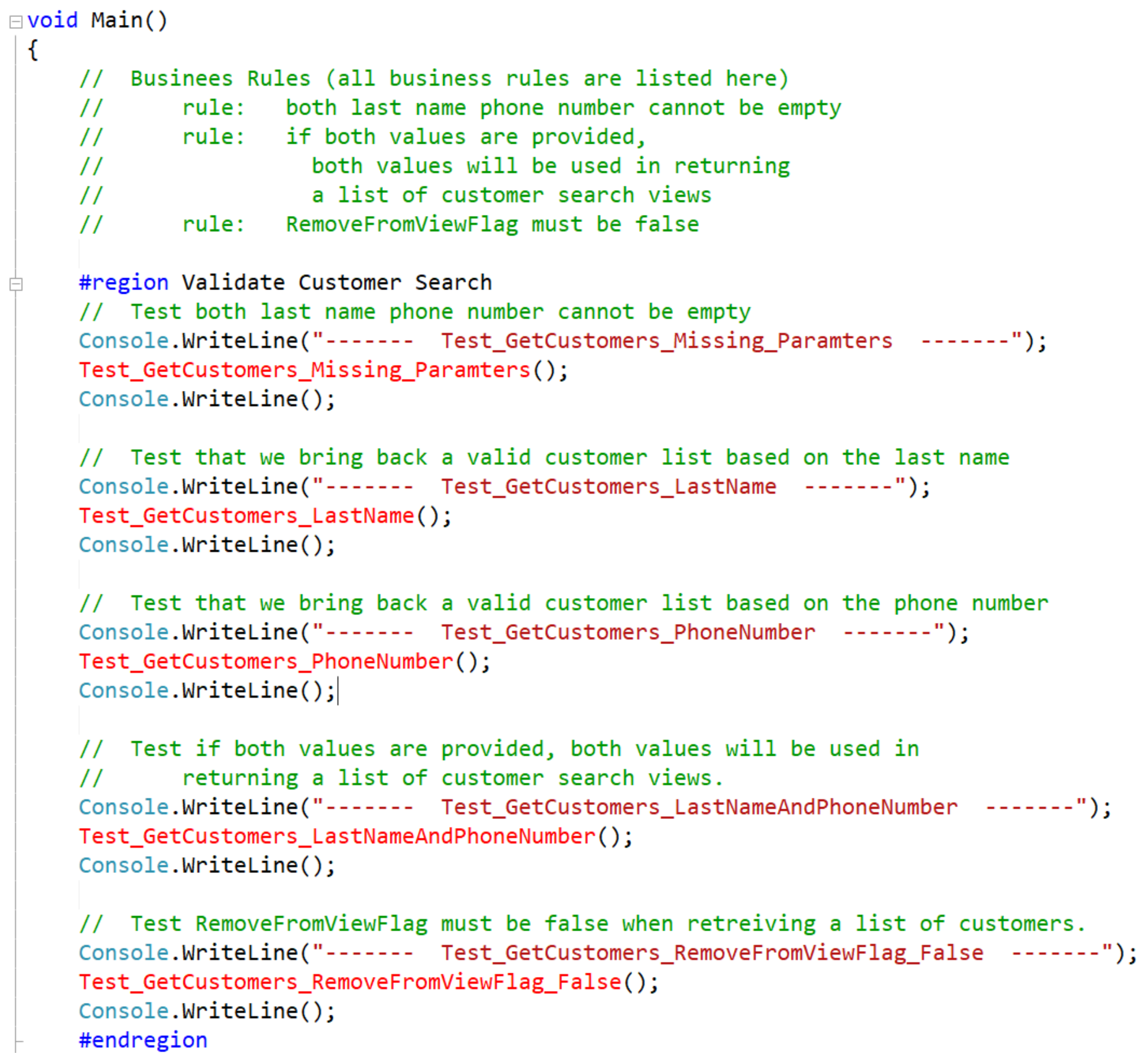
- GetCustomers
- Parameters
    - string lastName
    - string phoneNumber
- Return:
    - list<CustomerSearchView>
- Rules
    - Both last names and phone numbers cannot be empty.
    - If both values are provided, both values will be used in returning a list of customer search views.
    - RemoveFromViewFlag must be false.



```
#load "..\ViewModels\*.cs"
#load "..\Queries\CustomerService.linq"

void Main()
{
    //  Businees Rules (all business rules are listed here)
    //      rule:   both last name phone number cannot be empty
    //      rule:   if both values are provided,
    //                both values will be used in returning
    //                a list of customer search views
    //      rule:   RemoveFromViewFlag must be false
}
```

Create Placeholders for Tests

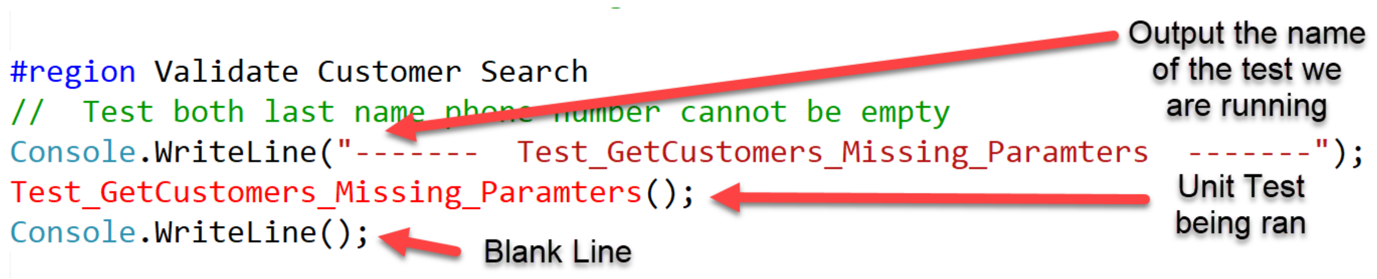
- Create a list of tests based on the requirements and rules. These will be placeholders as we build the rules



```
void Main()
{
        //  Businees Rules (all business rules are listed here)
        //      rule:   both last name phone number cannot be empty
        //      rule:   if both values are provided,
        //                 both values will be used in returning
        //                 a list of customer search views
        //      rule:   RemoveFromViewFlag must be false

        #region Validate Customer Search
        //  Test both last name phone number cannot be empty
        Console.WriteLine("-------  Test_GetCustomers_Missing_Paramters  -------");
        Test_GetCustomers_Missing_Paramters();
        Console.WriteLine();

        //  Test that we bring back a valid customer list based on the last name
        Console.WriteLine("-------  Test_GetCustomers_LastName  -------");
        Test_GetCustomers_LastName();
        Console.WriteLine();

        //  Test that we bring back a valid customer list based on the phone number
        Console.WriteLine("-------  Test_GetCustomers_PhoneNumber  -------");
        Test_GetCustomers_PhoneNumber();
        Console.WriteLine();

        //  Test if both values are provided, both values will be used in
        //      returning a list of customer search views.
        Console.WriteLine("-------  Test_GetCustomers_LastNameAndPhoneNumber  -------");
        Test_GetCustomers_LastNameAndPhoneNumber();
        Console.WriteLine();

        //  Test RemoveFromViewFlag must be false when retreiving a list of customers.
        Console.WriteLine("-------  Test_GetCustomers_RemoveFromViewFlag_False  -------");
        Test_GetCustomers_RemoveFromViewFlag_False();
        Console.WriteLine();
        #endregion
```

Create Placeholders for Tests (Continue)

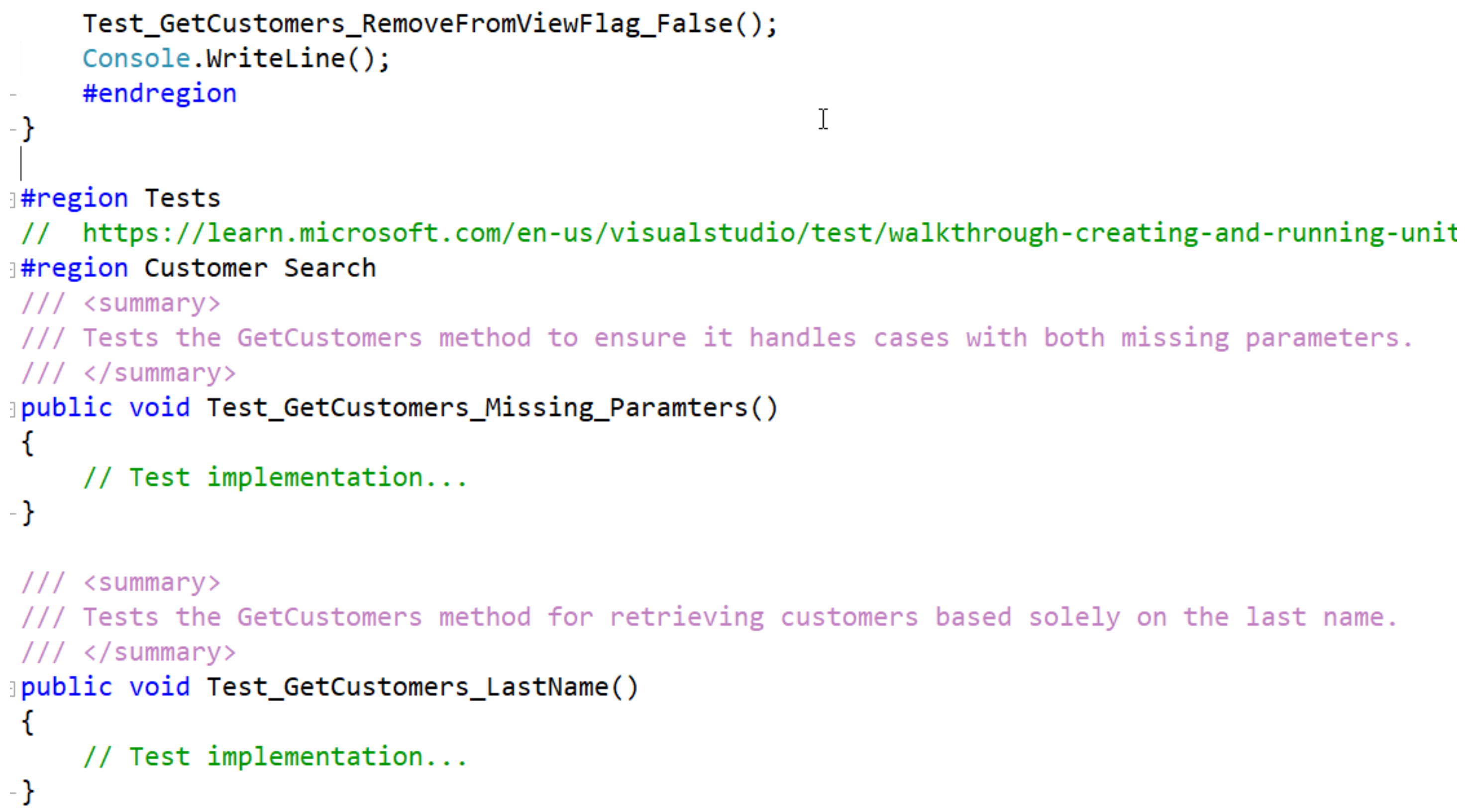
- Unit Test(s) layout pattern.



```
#region Validate Customer Search
//  Test both last name phone number cannot be empty
Console.WriteLine("-------  Test_GetCustomers_Missing_Paramters  -------");
Test_GetCustomers_Missing_Paramters();
Console.WriteLine();
```

Create Methods for Placeholders

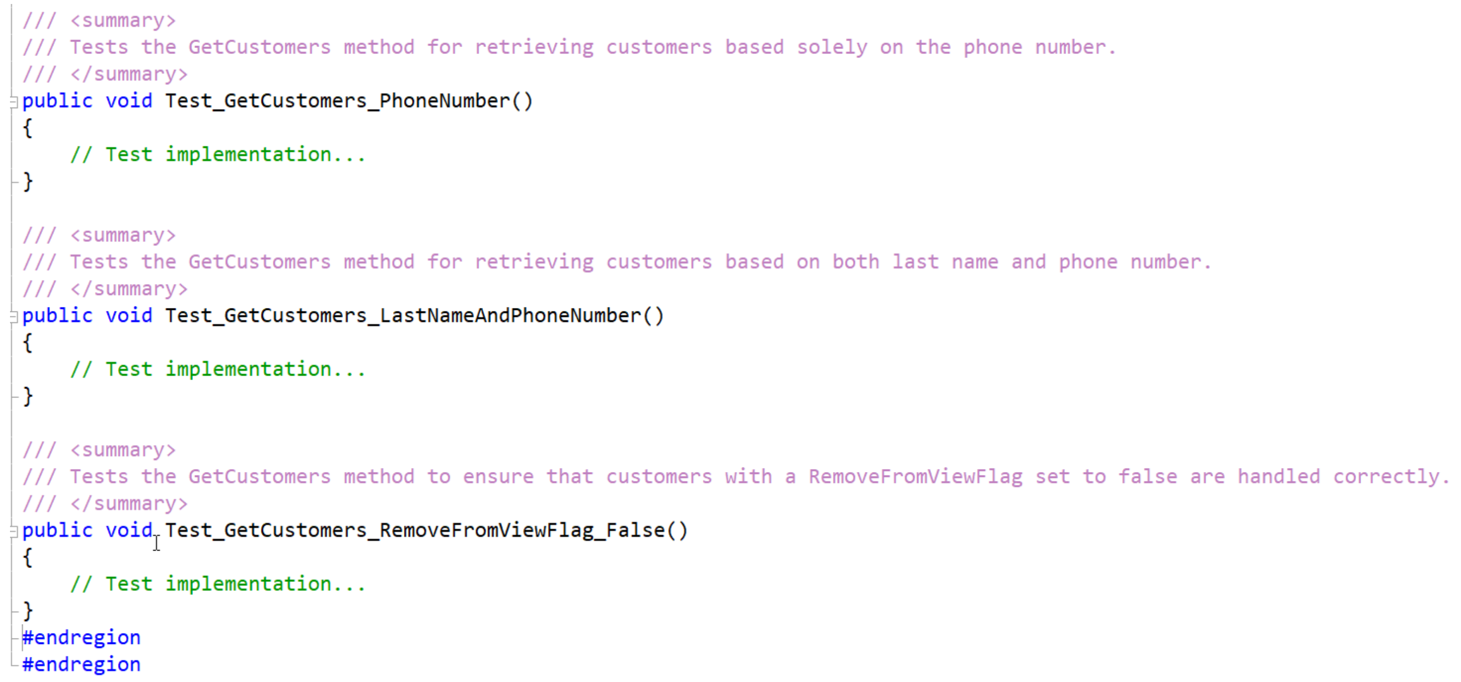
- https://learn.microsoft.com/en-us/visualstudio/test/walkthrough-creating-and-running-unit-tests-for-managed-code?view=vs-2022



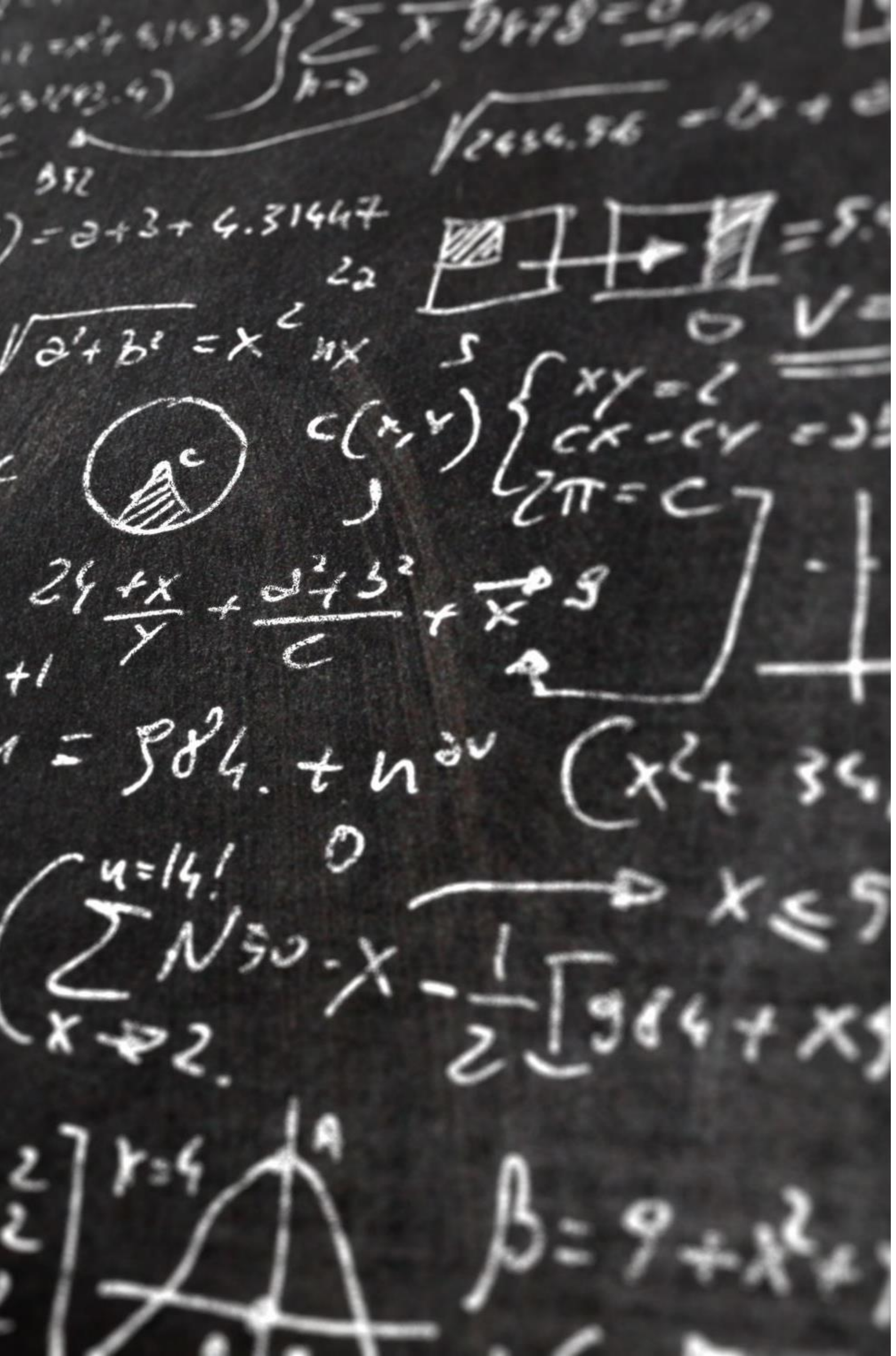
```
    Test_GetCustomers_RemoveFromViewFlag_False();
    Console.WriteLine();
    #endregion
}

#region Tests
//  https://learn.microsoft.com/en-us/visualstudio/test/walkthrough-creating-and-running-unit
#region Customer Search
/// <summary>
/// Tests the GetCustomers method to ensure it handles cases with both missing parameters.
/// </summary>
public void Test_GetCustomers_Missing_Paramters()
{
    // Test implementation...
}

/// <summary>
/// Tests the GetCustomers method for retrieving customers based solely on the last name.
/// </summary>
public void Test_GetCustomers_LastName()
{
    // Test implementation...
}
```

Create Methods for Placeholders

- Continue



```
/// <summary>
/// Tests the GetCustomers method for retrieving customers based solely on the phone number.
/// </summary>
public void Test_GetCustomers_PhoneNumber()
{
    // Test implementation...
}

/// <summary>
/// Tests the GetCustomers method for retrieving customers based on both last name and phone number.
/// </summary>
public void Test_GetCustomers_LastNameAndPhoneNumber()
{
    // Test implementation...
}

/// <summary>
/// Tests the GetCustomers method to ensure that customers with a RemoveFromViewFlag set to false are handled correctly.
/// </summary>
public void Test_GetCustomers_RemoveFromViewFlag_False()
{
    // Test implementation...
}
#endregion
#endregion
```

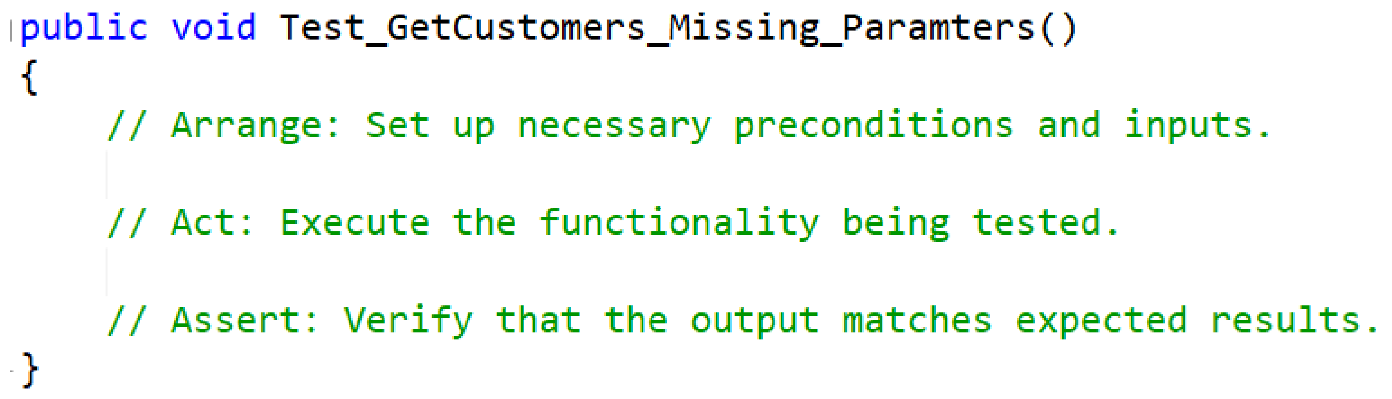


Reviewing what test can be implemented at this time.

- Only the test “Test_GetCustomers_Missing_Parameters” can be coded at this time. We are not relying on the known data in the customer table.
- The other four tests require specific data from the customer table. Therefore, we must implement the Add/Edit method before moving forward.
  - `Test_GetCustomers_LastName`
  - `Test_GetCustomers_PhoneNumber`
  - `Test_GetCustomers_LastNameAndPhoneNumber`
  - `Test_GetCustomers_RemoveFromViewFlag_False`

Creating the “Get Customers Missing Parameters” Unit Test

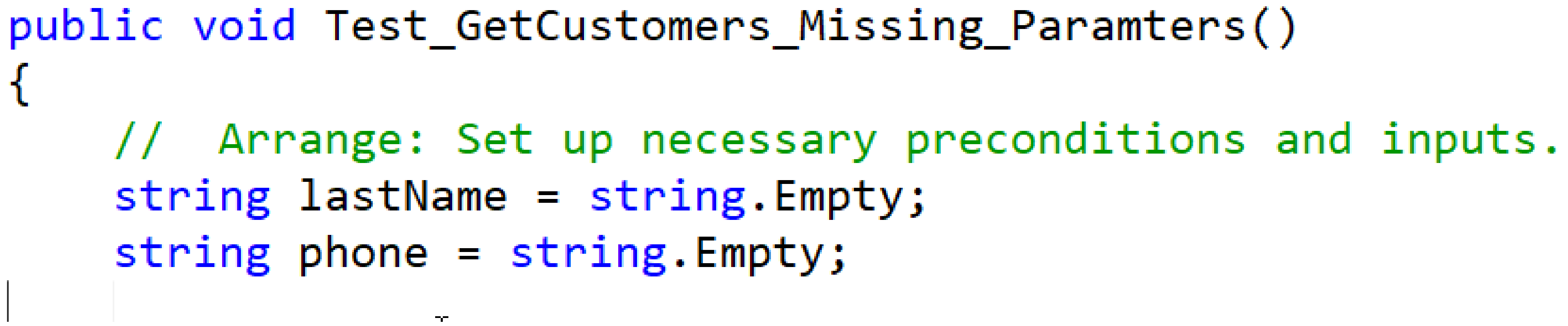
- Add the unit test AAA template so that we remember what we are creating.



```
public void Test_GetCustomers_Missing_Paramters()
{
    // Arrange: Set up necessary preconditions and inputs.

    // Act: Execute the functionality being tested.

    // Assert: Verify that the output matches expected results.
}
```

Arrange

- We must set up variables to store our initial data for the "Arrange" phase.



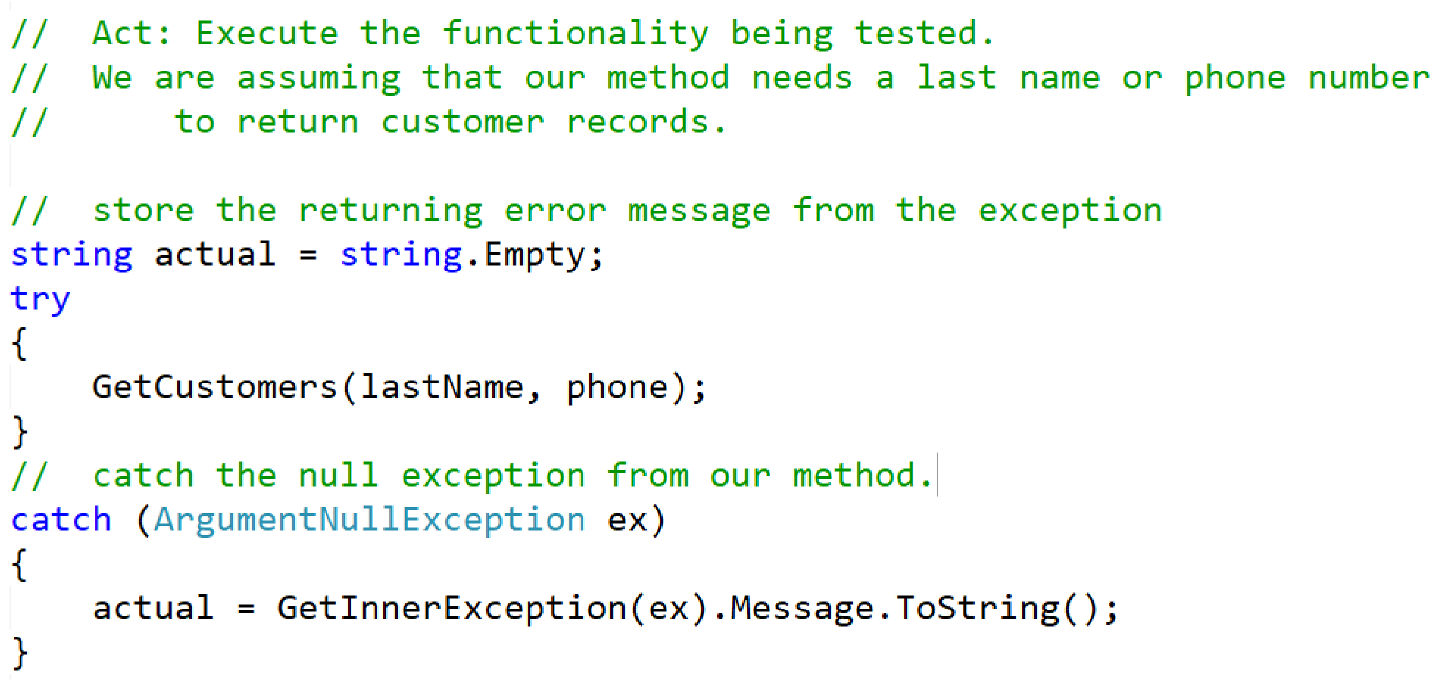
```
public void Test_GetCustomers_Missing_Paramters()
{
    //  Arrange: Set up necessary preconditions and inputs.
    string lastName = string.Empty;
    string phone = string.Empty;
```

Act

- For the "Act" phase, we'll call the "GetCustomers" method. However, this method hasn't been finished yet. So, because of this, we can use the “Test First”.
  - Test Fail.
  - Refactor to get the test working.
  - Test Pass.
  - Refactor to clean up the code.

Act - Continue

- When an exception is thrown, we will always get a “well form” exception message from the method that we are calling that is presented to the end user. We will be comparing this message (actual) to the expected message.



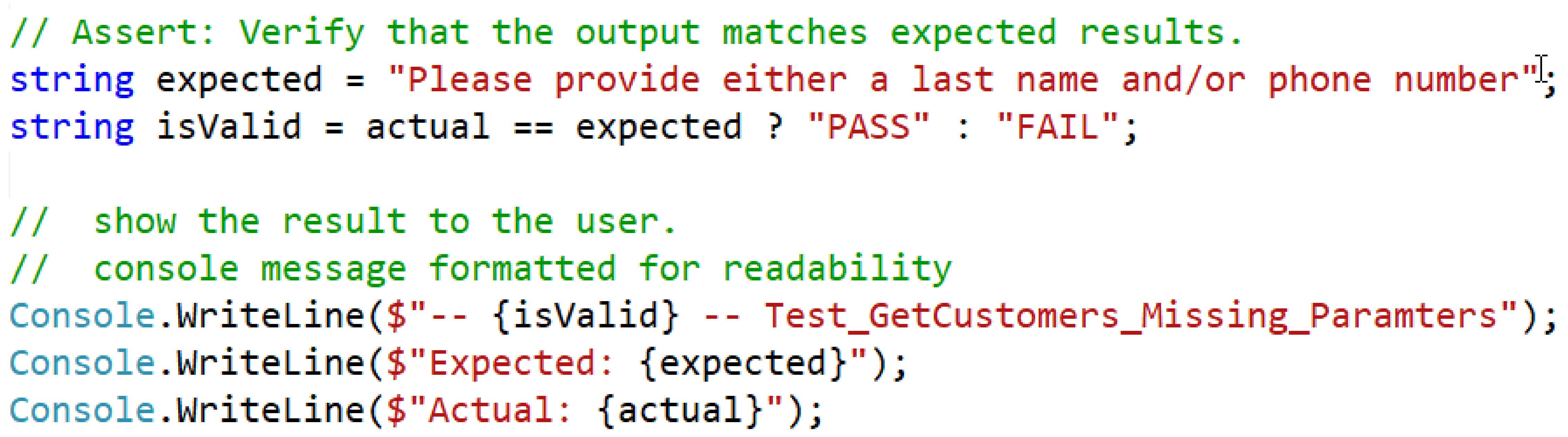
```
//  Act: Execute the functionality being tested.
//  We are assuming that our method needs a last name or phone number
//      to return customer records.

//  store the returning error message from the exception
string actual = string.Empty;
try
{
    GetCustomers(lastName, phone);
}
//  catch the null exception from our method.
catch (ArgumentNullException ex)
{
    actual = GetInnerException(ex).Message.ToString();
}
```

Add GetInnerException Method

```
/// <summary>
/// Retrieves the innermost exception from a given exception.
/// </summary>
/// <param name="ex">The exception from which to extract the innermost exception.</param>
/// <returns>The innermost exception,
///     or the original exception if it has no inner exceptions.
/// </returns>
private Exception GetInnerException(Exception ex)
{
    while (ex.InnerException != null)
        ex = ex.InnerException;
    return ex;
}
```

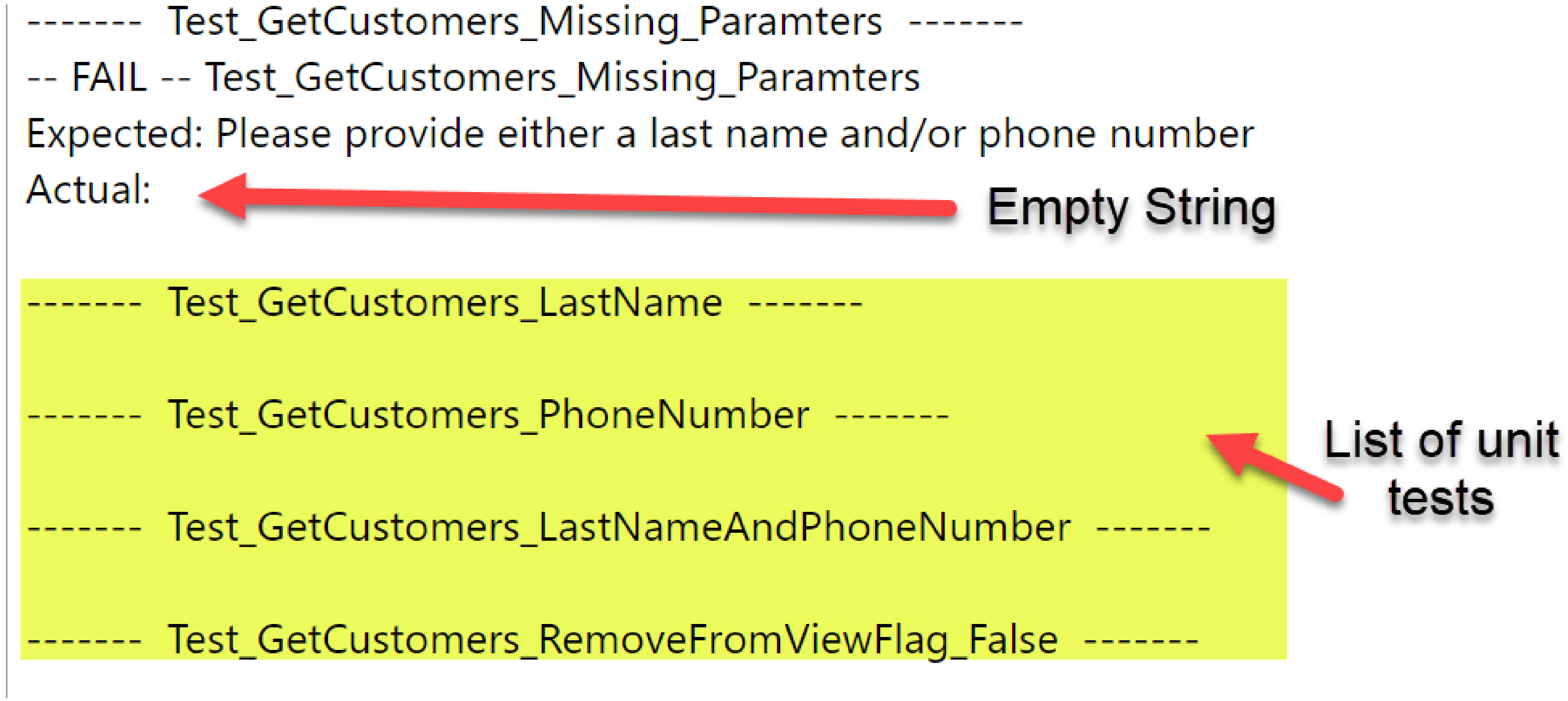
Arrange

- For the “Arrange" phase, we must evaluate the return exception message with what we expect the message to be.



```
// Assert: Verify that the output matches expected results.
string expected = "Please provide either a last name and/or phone number";
string isValid = actual == expected ? "PASS" : "FAIL";

//  show the result to the user.
//  console message formatted for readability
Console.WriteLine($"-- {isValid} -- Test_GetCustomers_Missing_Paramters");
Console.WriteLine($"Expected: {expected}");
Console.WriteLine($"Actual: {actual}");
```

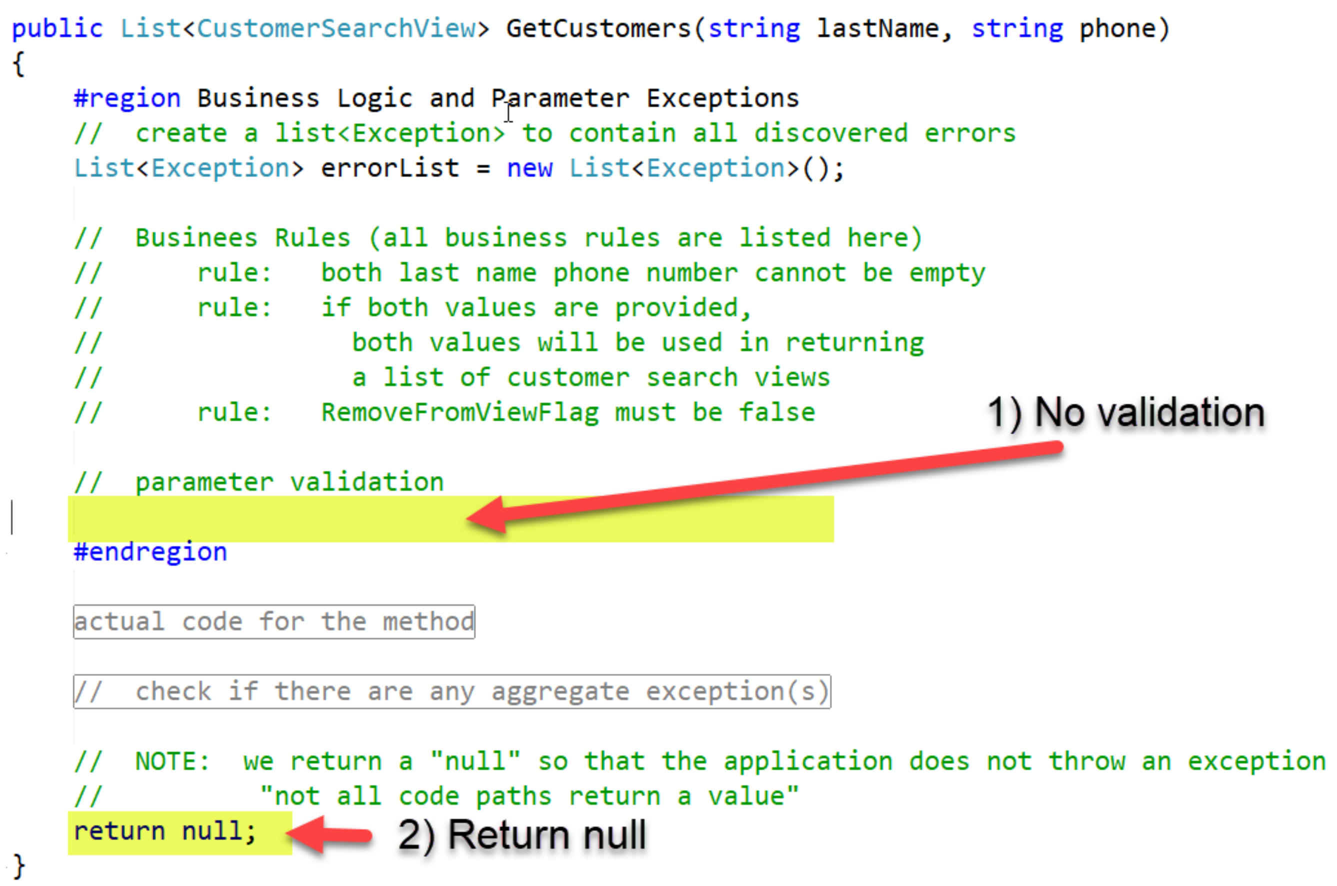
Unit Test Result



```
-------  Test_GetCustomers_Missing_Paramters  -------
-- FAIL -- Test_GetCustomers_Missing_Paramters
Expected: Please provide either a last name and/or phone number
Actual:

-------  Test_GetCustomers_LastName  -------

-------  Test_GetCustomers_PhoneNumber  -------

-------  Test_GetCustomers_LastNameAndPhoneNumber  -------

-------  Test_GetCustomers_RemoveFromViewFlag_False  -------
```

Why did this test fail?

1. In the GetCustomers method, we have not created any exception(s) in the " Business Logic and Parameter Exceptions" section.
2. The returning result is a "null," so the LINQ query would run.

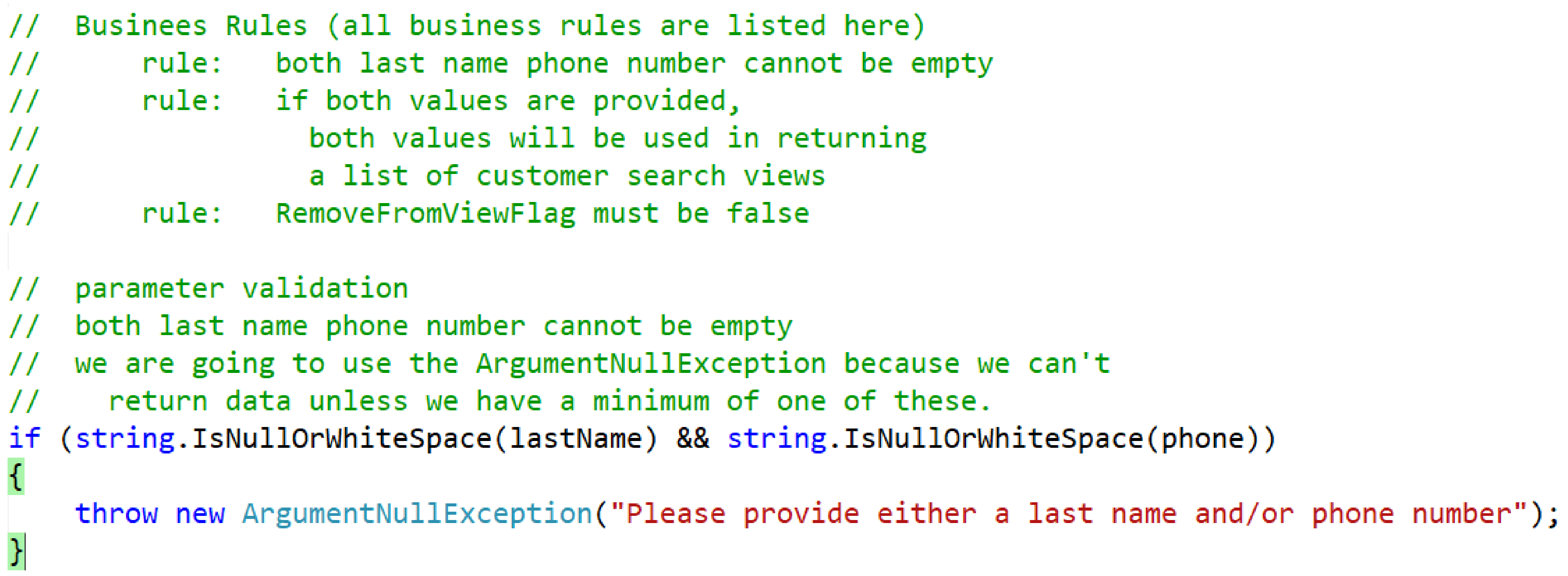


```
public List<CustomerSearchView> GetCustomers(string lastName, string phone)
{
    #region Business Logic and Parameter Exceptions
    //  create a list<Exception> to contain all discovered errors
    List<Exception> errorList = new List<Exception>();

    //  Businees Rules (all business rules are listed here)
    //      rule:   both last name phone number cannot be empty
    //      rule:   if both values are provided,
    //                both values will be used in returning
    //                a list of customer search views
    //      rule:   RemoveFromViewFlag must be false

    //  parameter validation

    #endregion

    actual code for the method

    //  check if there are any aggregate exception(s)

    //  NOTE:  we return a "null" so that the application does not throw an exception
    //           "not all code paths return a value"
    return null;
}
```

Refactoring – Red

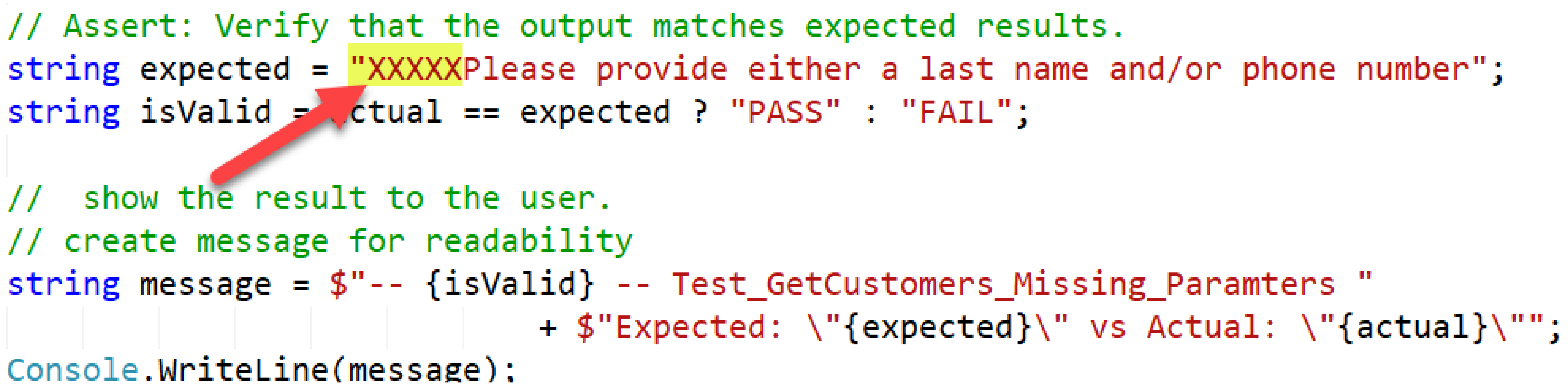
- Write a failing test.
  - We are going to complete the validation for missing parameters.
  - We will refactor the expected result to an invalid “expected” result.
  - Run the unit test “Test_GetCustomers_Missing_Paramters“ to verify we get a failed result.

Complete the Validation for Missing Parameters

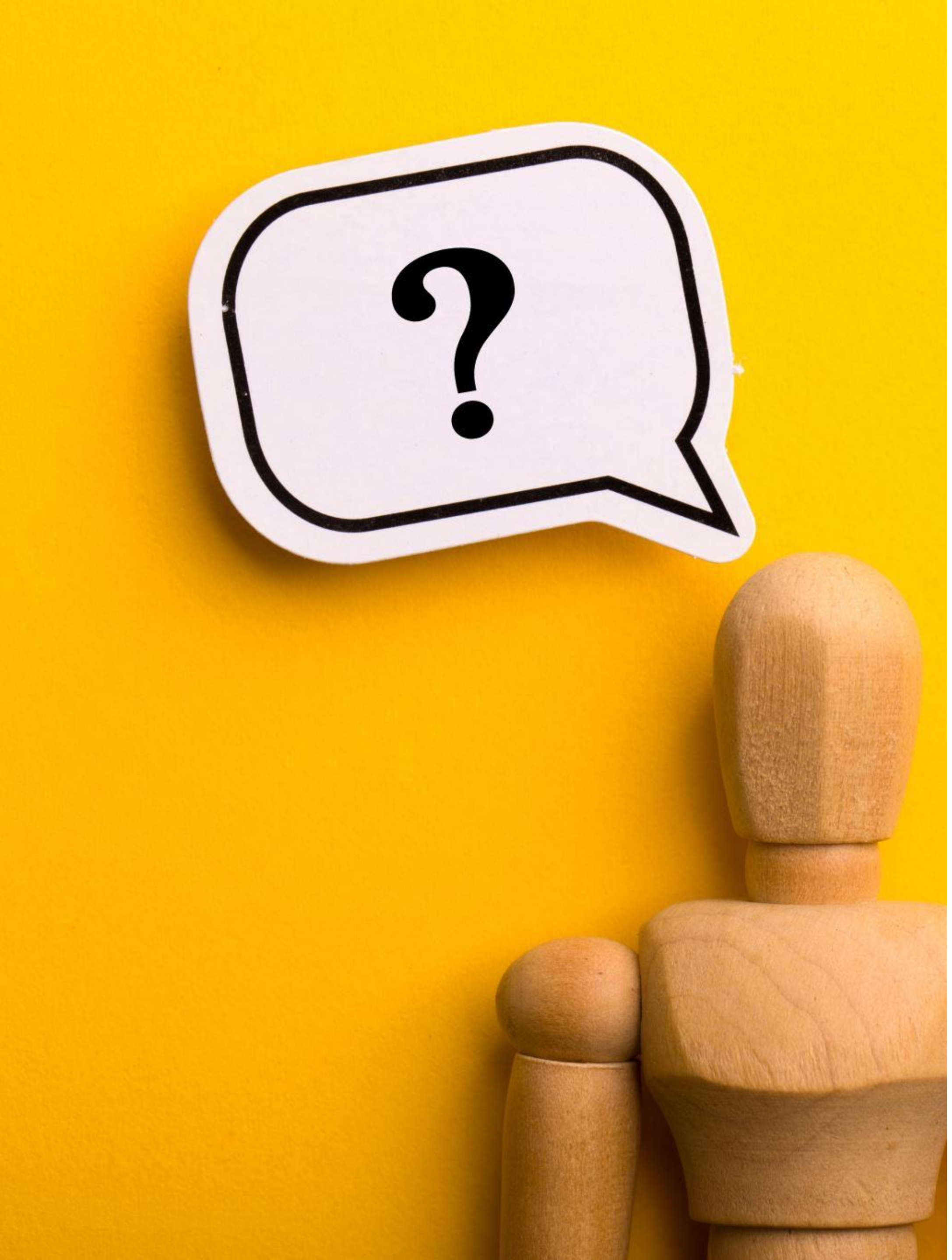


```
//  Businees Rules (all business rules are listed here)
//      rule:   both last name phone number cannot be empty
//      rule:   if both values are provided,
//                both values will be used in returning
//                a list of customer search views
//      rule:   RemoveFromViewFlag must be false

//  parameter validation
//  both last name phone number cannot be empty
//  we are going to use the ArgumentNullException because we can't
//    return data unless we have a minimum of one of these.
if (string.IsNullOrWhiteSpace(lastName) && string.IsNullOrWhiteSpace(phone))
{
    throw new ArgumentNullException("Please provide either a last name and/or phone number");
}
```

Refactor the “expected” Result to an Invalid Result



```
// Assert: Verify that the output matches expected results.
string expected = "XXXXXPlease provide either a last name and/or phone number";
string isValid = actual == expected ? "PASS" : "FAIL";

//  show the result to the user.
// create message for readability
string message = $"-- {isValid} -- Test_GetCustomers_Missing_Paramters "
                            + $"Expected: \"{expected}\" vs Actual: \"{actual}\"";
Console.WriteLine(message);
```

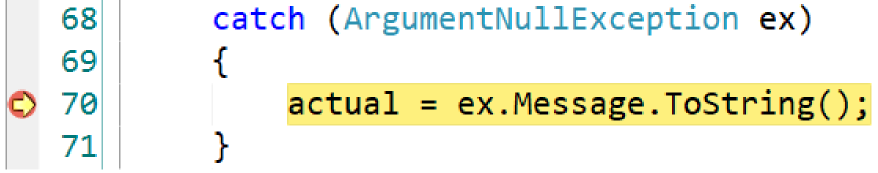


Run the Unit Test

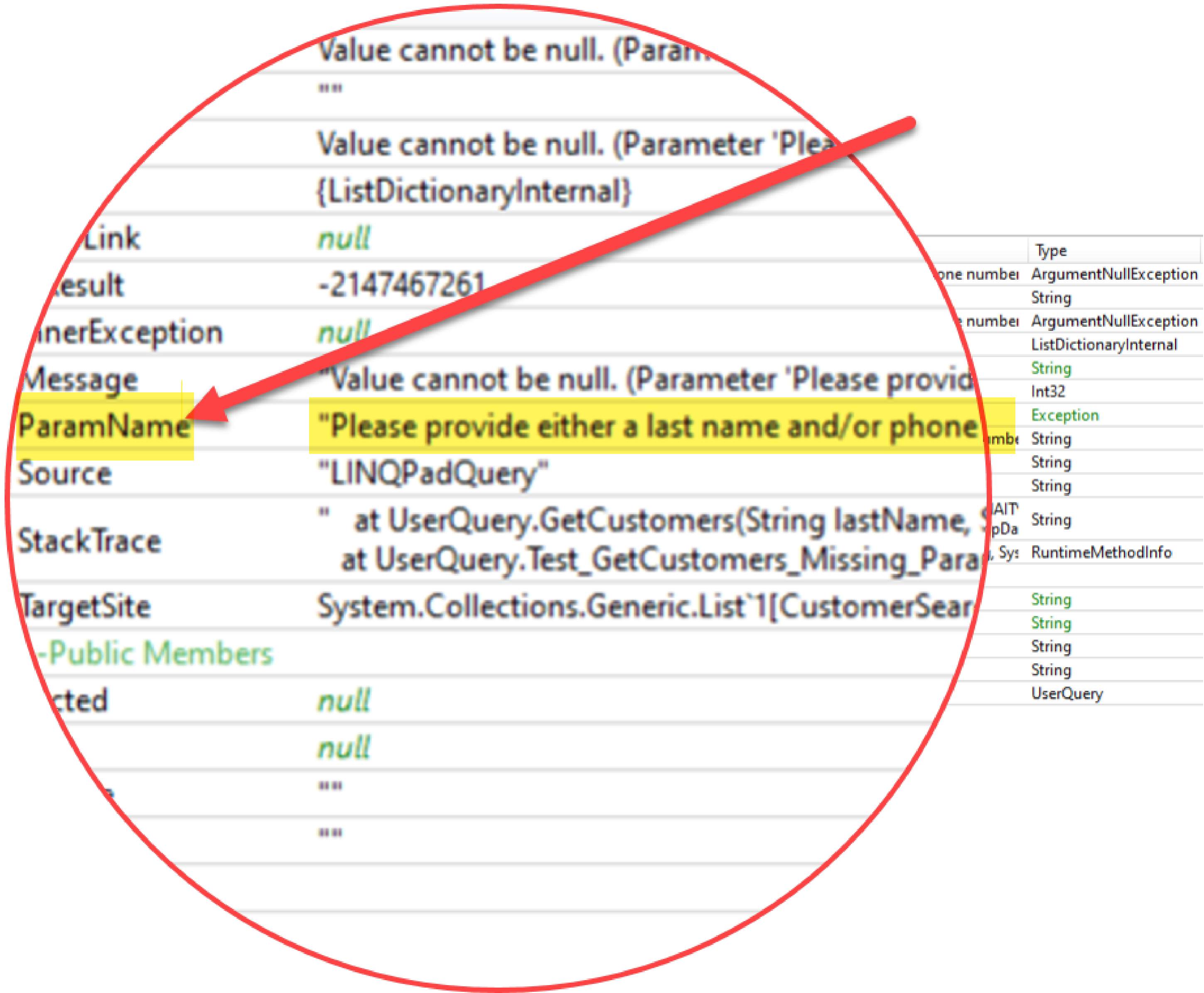
- There are 2 issues with the output.
  - 1) The return message includes information about the exception.
    - “Value cannot be null. (Parameter ‘” and “’)“
  - 2) The two values are different Even if we did not have the exception information.
    - Expected: XXXXXPlease provide either a last name and/or phone number
    - Actual: Please provide either a last name and/or phone number

Remove the Exception Information.

- Place a breakpoint on the exception and review the debug information.



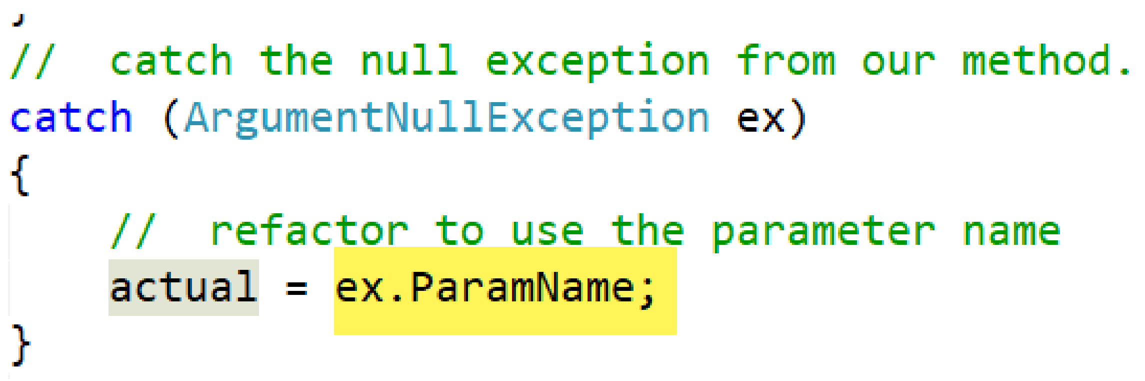
```
catch (ArgumentNullException ex)
{
    actual = ex.Message.ToString();
}
```



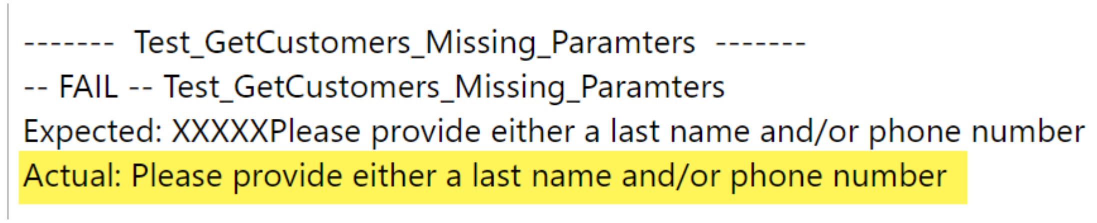
```
                 Value cannot be null. (Param
                 ""
                 Value cannot be null. (Parameter 'Plea
                 {ListDictionaryInternal}
Link             null
esult            -2147467261
nerException     null
Message          "Value cannot be null. (Parameter 'Please provid
ParamName        "Please provide either a last name and/or phone
Source           "LINQPadQuery"
StackTrace       "   at UserQuery.GetCustomers(String lastName, S
                  at UserQuery.Test_GetCustomers_Missing_Para
TargetSite       System.Collections.Generic.List`1[CustomerSear
-Public Members
cted             null
                 null
                 ""
                 ""
```

	Type
one number	ArgumentNullException
	String
number	ArgumentNullException
	ListDictionaryInternal
	String
	Int32
	Exception
mbe	String
	String
	String
AIT pDa	String
Sys	RuntimeMethodInfo
	String
	String
	String
	String
	UserQuery

Refactor “Actual” Result



```
//  catch the null exception from our method.
catch (ArgumentNullException ex)
{
    //  refactor to use the parameter name
    actual = ex.ParamName;
}
```



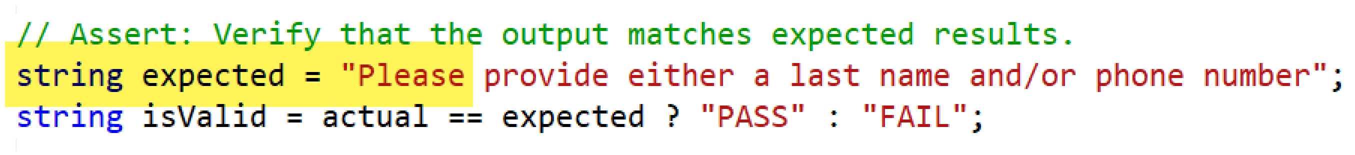
```
-------  Test_GetCustomers_Missing_Paramters  -------
-- FAIL -- Test_GetCustomers_Missing_Paramters
Expected: XXXXXPlease provide either a last name and/or phone number
Actual: Please provide either a last name and/or phone number
```

Refactoring – Green

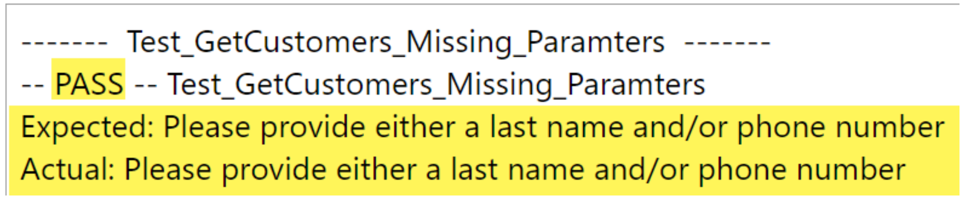
- Write a passing test.
  - We will refactor the expected result to a valid “expected” result.
  - Run the unit test “Test_GetCustomers_Missing_Paramters“ to verify we get a passing result.

Refactor a Valid “expected” Result.

- Remove the “XXXXX” from the beginning of the expected result.
- Run the unit test Test_GetCustomers_Missing_Parameters.



```
// Assert: Verify that the output matches expected results.
string expected = "Please provide either a last name and/or phone number";
string isValid = actual == expected ? "PASS" : "FAIL";
```



```
-------  Test_GetCustomers_Missing_Paramters  -------
-- PASS -- Test_GetCustomers_Missing_Paramters
Expected: Please provide either a last name and/or phone number
Actual: Please provide either a last name and/or phone number
```

Statement Regarding Slide Accuracy and Potential Revisions

- Please note that the content of these PowerPoint slides is accurate to the best of my knowledge at the time of presentation. However, as new research and developments emerge, or to enhance the learning experience, these slides may be subject to updates or revisions in the future. I encourage you to stay engaged with the course materials and any announcements for the most current information