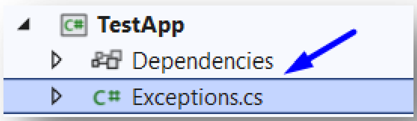
Exercises: Unit Testing Exceptions

Tasks for exercise in class and for homework to the course "Programming Advanced for QA" @ SoftUni.

Submit your solutions here: https://judge.softuni.org/Contests/4493

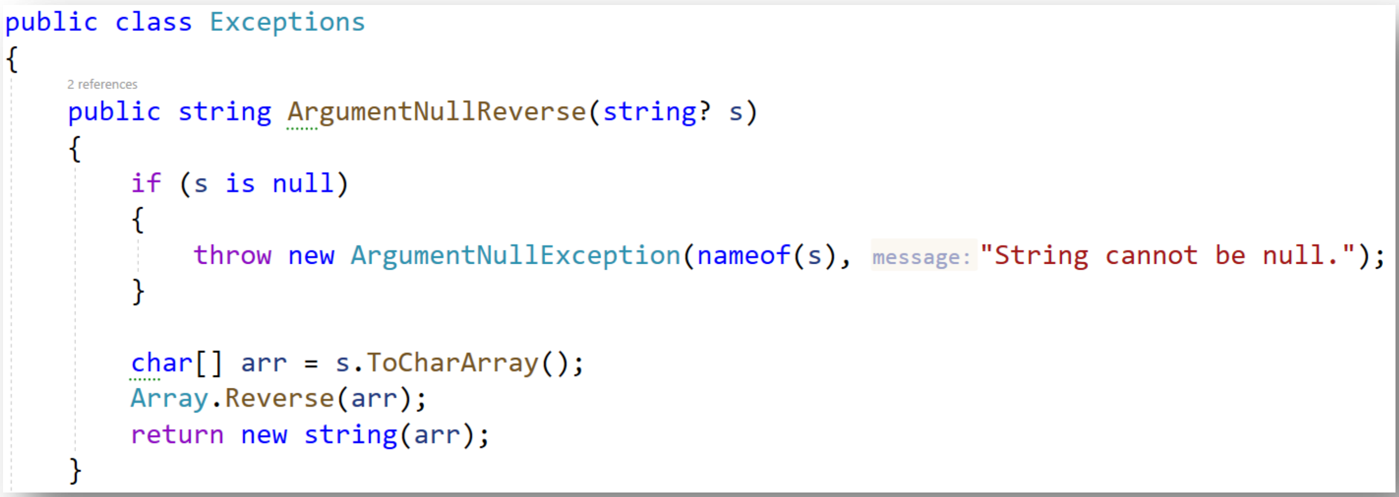
1. Unit Test: Reverse – Argument Null Exception

Look at the **provided skeleton** and examine the **`Exceptions.cs`** class that you will test:



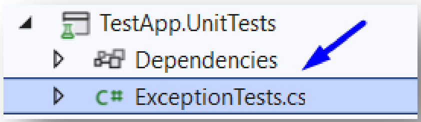
The class has **multiple methods each showing the use of different exceptions**.

The first method, **`ArgumentNullReverse()`**, takes in a **string and reverses it**. If the string is **null** an **`ArgumentNullException`** is thrown:

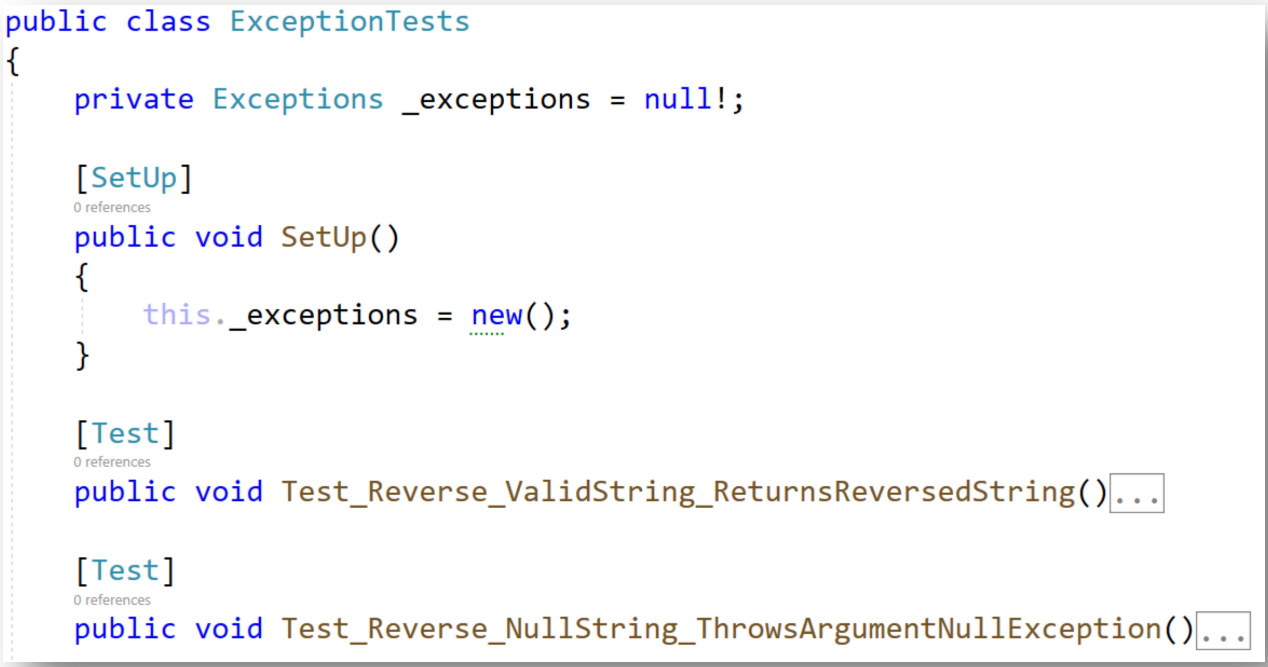


```
public class Exceptions
{
    2 references
    public string ArgumentNullReverse(string? s)
    {
        if (s is null)
        {
            throw new ArgumentNullException(nameof(s), message: "String cannot be null.");
        }

        char[] arr = s.ToCharArray();
        Array.Reverse(arr);
        return new string(arr);
    }
```

Then, look at the tests inside the **`ExceptionTests.cs`** class:

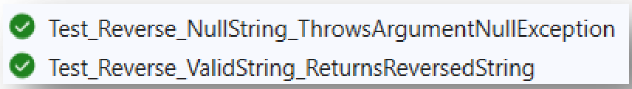


Notice the use of a **setup method**, so each test has a brand new **exception instance** to use.



```
public class ExceptionTests
{
    private Exceptions _exceptions = null!;

    [SetUp]
    0 references
    public void SetUp()
    {
        this._exceptions = new();
    }

    [Test]
    0 references
    public void Test_Reverse_ValidString_ReturnsReversedString()...

    [Test]
    0 references
    public void Test_Reverse_NullString_ThrowsArgumentNullException()...
```

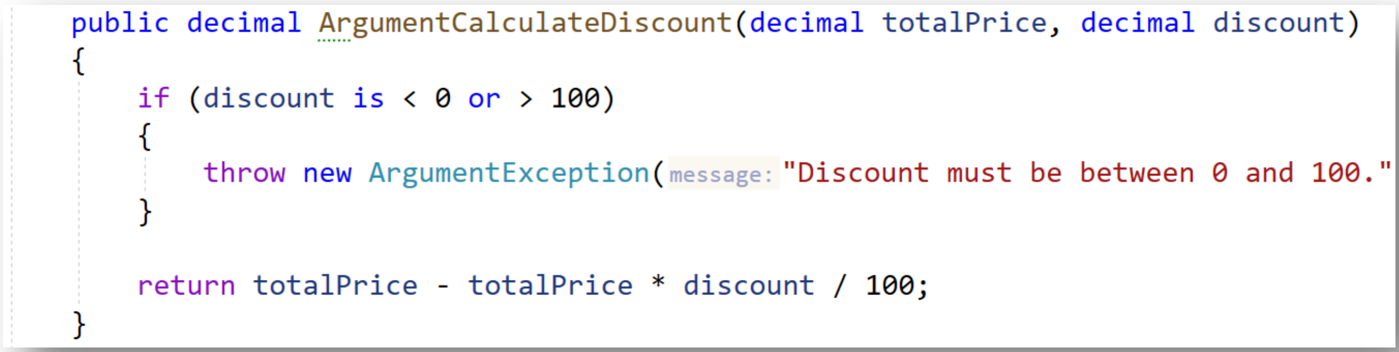
The two tests are **partially finished**, and your task is to finish them. The tests should run when you're finished:



```
✓ Test_Reverse_NullString_ThrowsArgumentNullException
✓ Test_Reverse_ValidString_ReturnsReversedString
```

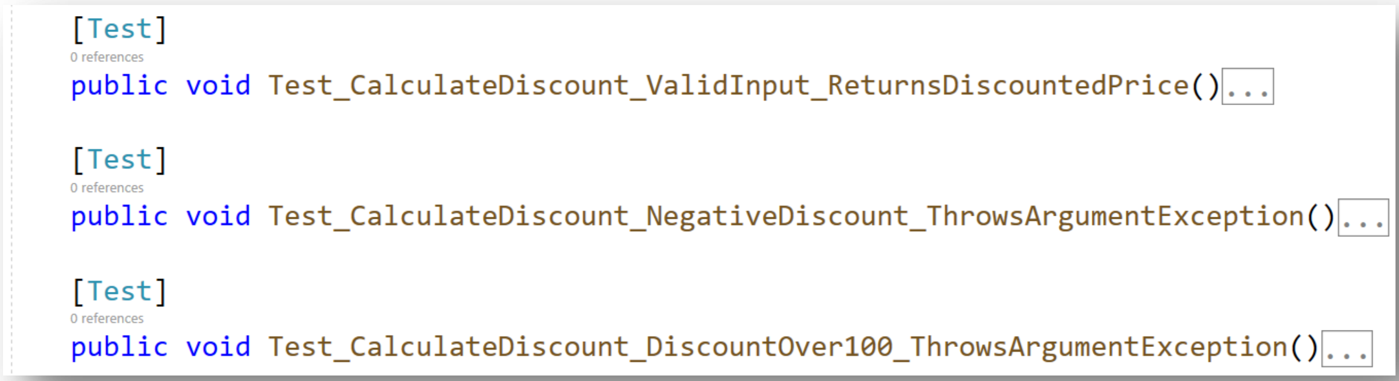
2. Unit Test: Calculate Discount – Argument Exception

The **`ArgumentCalculateDiscount()` method** takes in a **total price decimal**, and **discount decimal**. It calculates and returns the discounted price. If the discount is **lower than 0 or higher than 100** an **`ArgumentException`** is thrown:



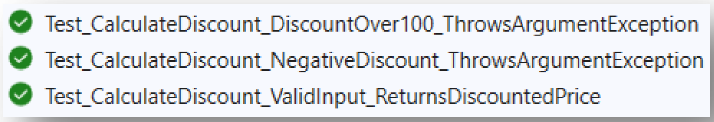
```
public decimal ArgumentCalculateDiscount(decimal totalPrice, decimal discount)
{
    if (discount is < 0 or > 100)
    {
        throw new ArgumentException(message: "Discount must be between 0 and 100."
    }

    return totalPrice - totalPrice * discount / 100;
}
```

Now, look at the tests:



```
[Test]
0 references
public void Test_CalculateDiscount_ValidInput_ReturnsDiscountedPrice()...

[Test]
0 references
public void Test_CalculateDiscount_NegativeDiscount_ThrowsArgumentException()...

[Test]
0 references
public void Test_CalculateDiscount_DiscountOver100_ThrowsArgumentException()...
```

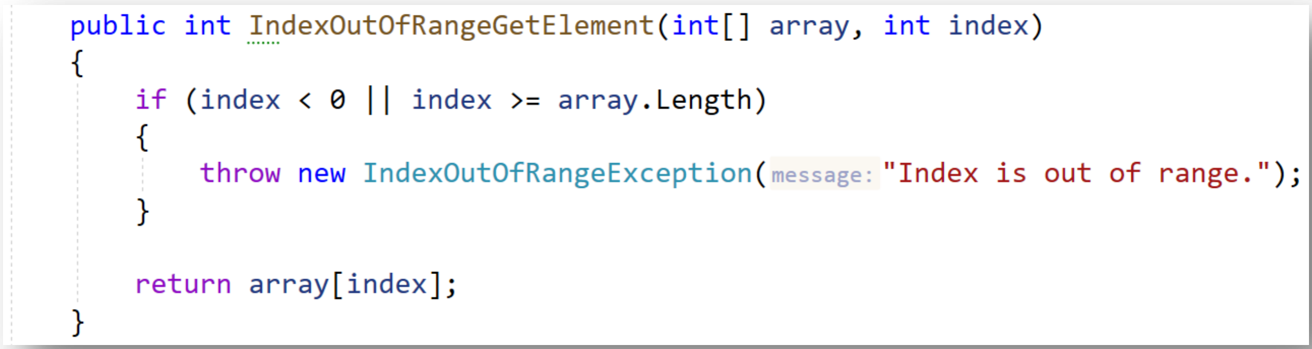
You are given **two partially finished** test, the rest are **empty,** and your task is to finish them. The tests should run when you're finished:



```
✓ Test_CalculateDiscount_DiscountOver100_ThrowsArgumentException
✓ Test_CalculateDiscount_NegativeDiscount_ThrowsArgumentException
✓ Test_CalculateDiscount_ValidInput_ReturnsDiscountedPrice
```

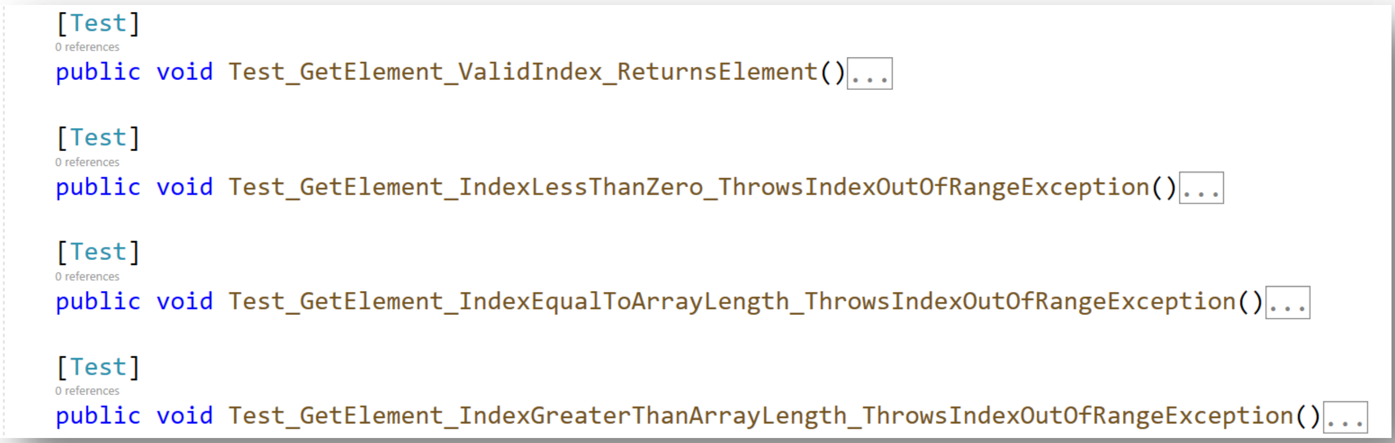
3. Unit Test: Get Element – Index out of Range Exception

The **`IndexOutOfRangeGetElement()` method** takes in an **array of integers**, and an **index**. It **retrieves** the element from the **array** at the **given index**. If the index is **lower than 0 or higher / equal** to the **length** an **`IndexOutOfRangeException`** is thrown:



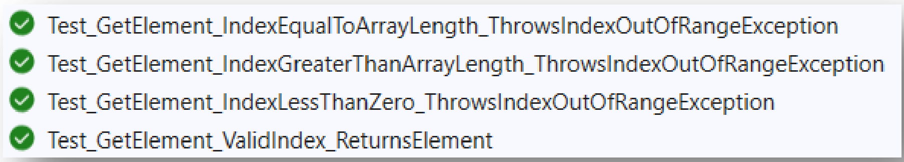
```
public int IndexOutOfRangeGetElement(int[] array, int index)
{
    if (index < 0 || index >= array.Length)
    {
        throw new IndexOutOfRangeException(message: "Index is out of range.");
    }

    return array[index];
}
```

Now, look at the tests:



```
[Test]
0 references
public void Test_GetElement_ValidIndex_ReturnsElement()...

[Test]
0 references
public void Test_GetElement_IndexLessThanZero_ThrowsIndexOutOfRangeException()...

[Test]
0 references
public void Test_GetElement_IndexEqualToArrayLength_ThrowsIndexOutOfRangeException()...

[Test]
0 references
public void Test_GetElement_IndexGreaterThanArrayLength_ThrowsIndexOutOfRangeException()...
```

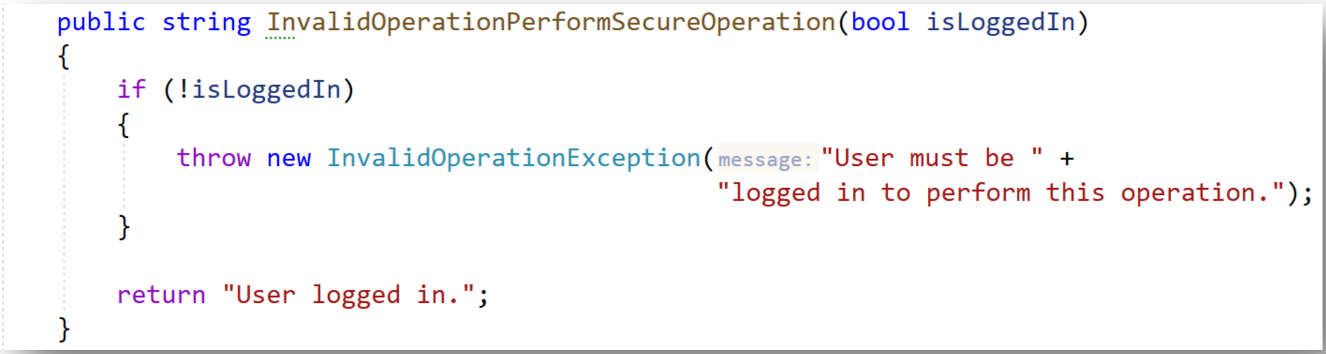
You are given **one partially finished** test, the rest are **empty,** and your task is to finish them. The tests should run when you're finished:



```
✓ Test_GetElement_IndexEqualToArrayLength_ThrowsIndexOutOfRangeException
✓ Test_GetElement_IndexGreaterThanArrayLength_ThrowsIndexOutOfRangeException
✓ Test_GetElement_IndexLessThanZero_ThrowsIndexOutOfRangeException
✓ Test_GetElement_ValidIndex_ReturnsElement
```

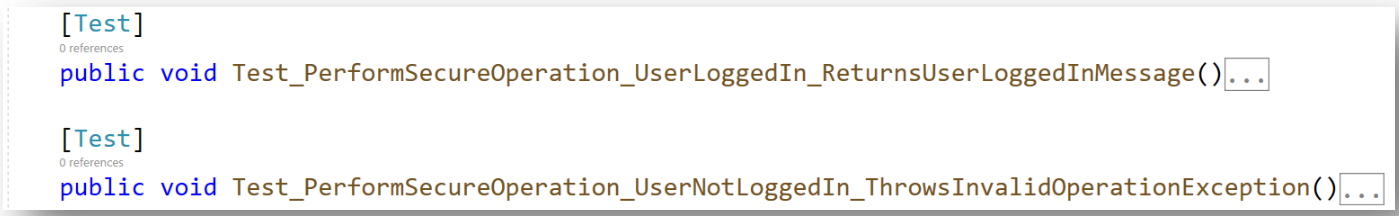
4. Unit Test: Perform Secure Operation – Invalid Operation Exception

The **`InvalidOperationPerformSecureOperation()` method** takes in a **Boolean indicating if the user is logged in**. If the user is not logged in an **`InvalidOperationException`** is thrown:



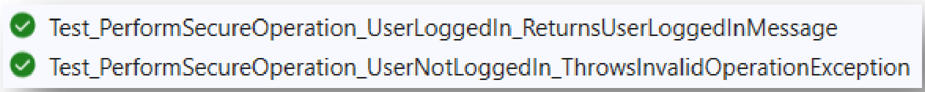
```
public string InvalidOperationPerformSecureOperation(bool isLoggedIn)
{
    if (!isLoggedIn)
    {
        throw new InvalidOperationException(message: "User must be " +
                                            "logged in to perform this operation.");
    }

    return "User logged in.";
}
```

Now, look at the tests:



```
[Test]
0 references
public void Test_PerformSecureOperation_UserLoggedIn_ReturnsUserLoggedInMessage()...

[Test]
0 references
public void Test_PerformSecureOperation_UserNotLoggedIn_ThrowsInvalidOperationException()...
```

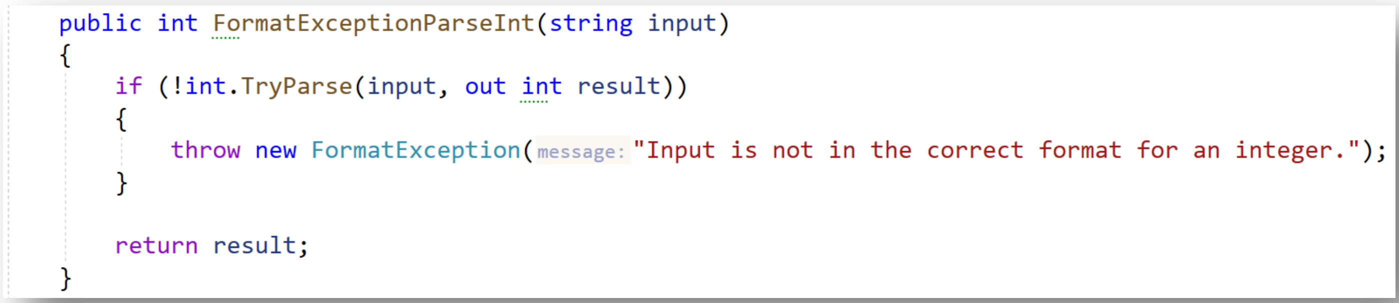
You are given **two empty** tests, and your task is to finish them. The tests should run when you're finished:



```
✓ Test_PerformSecureOperation_UserLoggedIn_ReturnsUserLoggedInMessage
✓ Test_PerformSecureOperation_UserNotLoggedIn_ThrowsInvalidOperationException
```

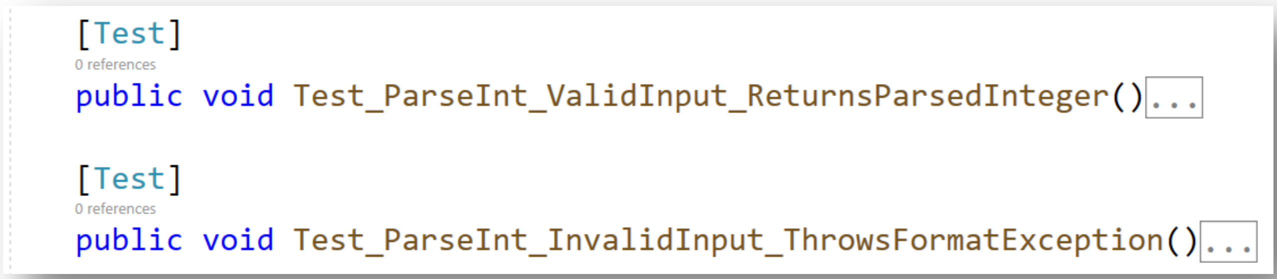
5. Unit Test: Parse Int - Format Exception

The **`FormatExceptionParseInt()` method** takes in a **string as input** and tries to **parse** it into an **integer**. If the string is not a valid number a **`FormatException`** is thrown:



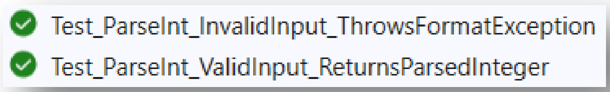
```
public int FormatExceptionParseInt(string input)
{
    if (!int.TryParse(input, out int result))
    {
        throw new FormatException(message: "Input is not in the correct format for an integer.");
    }

    return result;
}
```

Now, look at the tests:



```
[Test]
0 references
public void Test_ParseInt_ValidInput_ReturnsParsedInteger()...

[Test]
0 references
public void Test_ParseInt_InvalidInput_ThrowsFormatException()...
```

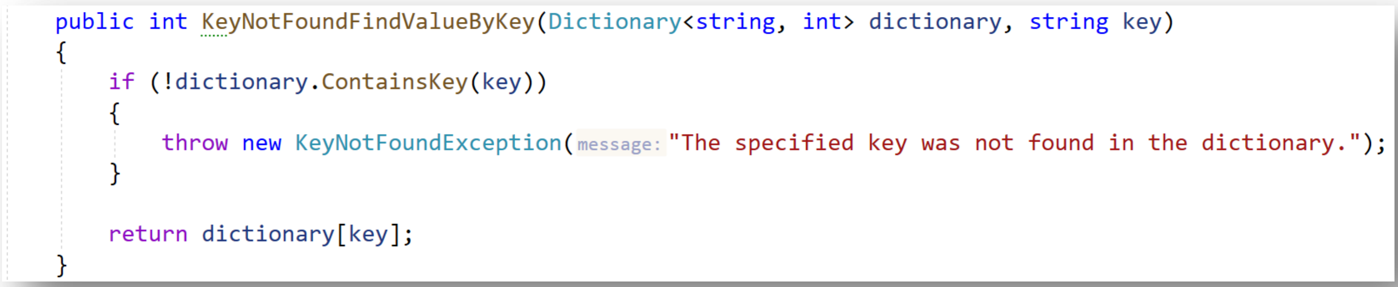
You are given **two empty** tests, and your task is to finish them. The tests should run when you're finished:



```
✓ Test_ParseInt_InvalidInput_ThrowsFormatException
✓ Test_ParseInt_ValidInput_ReturnsParsedInteger
```

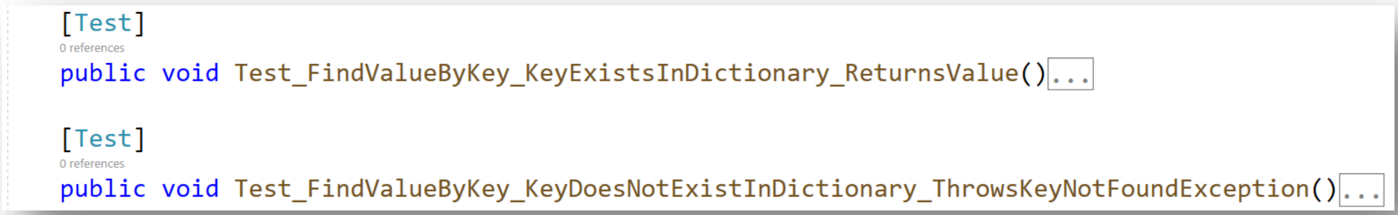
6. Unit Test: Find Value by Key – Key Not Found Exception

The **`KeyNotFoundFindValueByKey()` method** takes in a **dictionary**, and a **string representing a key from the dictionary**. If the key does not exist in the dictionary a **`KeyNotFoundException`** is thrown:



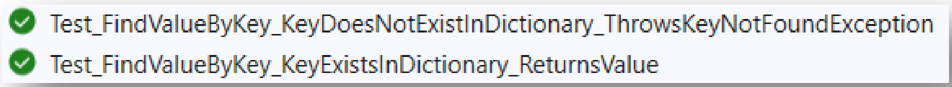
```
public int KeyNotFoundFindValueByKey(Dictionary<string, int> dictionary, string key)
{
    if (!dictionary.ContainsKey(key))
    {
        throw new KeyNotFoundException(message: "The specified key was not found in the dictionary.");
    }

    return dictionary[key];
}
```

Now, look at the tests:



```
[Test]
0 references
public void Test_FindValueByKey_KeyExistsInDictionary_ReturnsValue()...

[Test]
0 references
public void Test_FindValueByKey_KeyDoesNotExistInDictionary_ThrowsKeyNotFoundException()...
```

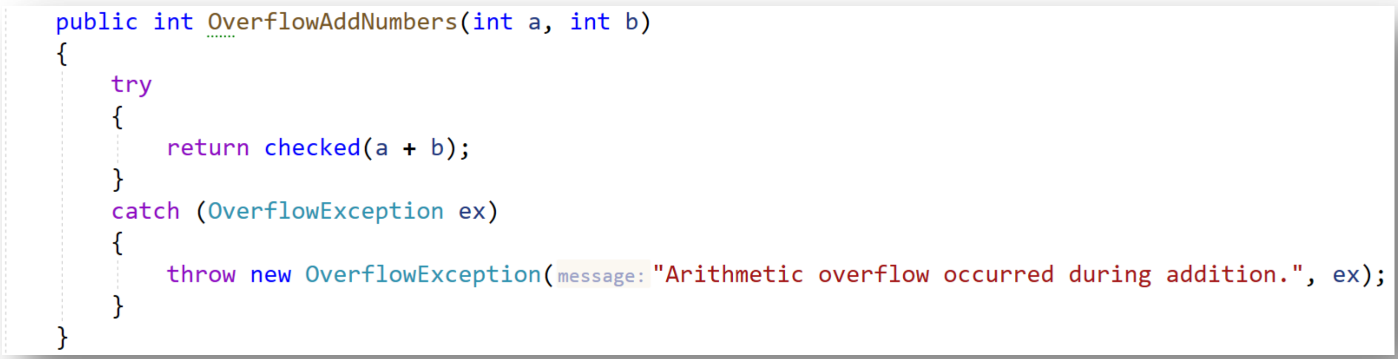
You are given **two empty** tests, and your task is to finish them. The tests should run when you're finished:



```
✓ Test_FindValueByKey_KeyDoesNotExistInDictionary_ThrowsKeyNotFoundException
✓ Test_FindValueByKey_KeyExistsInDictionary_ReturnsValue
```

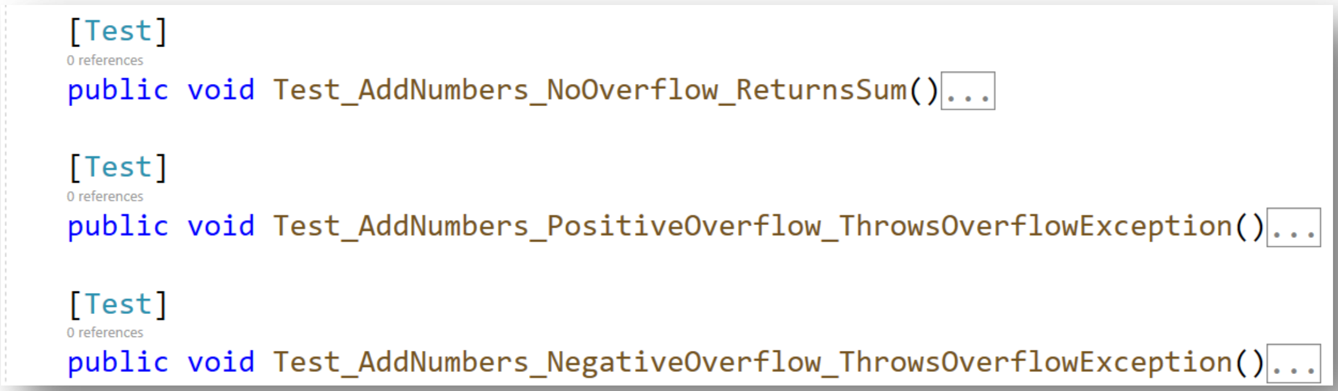
7. Unit Test: Add Numbers – Overflow Exception

The **`OverflowAddNumbers()` method** takes in **two numbers to be summed together**. If summing the numbers **overflows the integer type** a **`OverflowException`** is thrown:

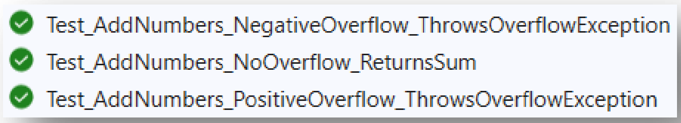


```
public int OverflowAddNumbers(int a, int b)
{
    try
    {
        return checked(a + b);
    }
    catch (OverflowException ex)
    {
        throw new OverflowException(message: "Arithmetic overflow occurred during addition.", ex);
    }
}
```

Now, look at the tests:

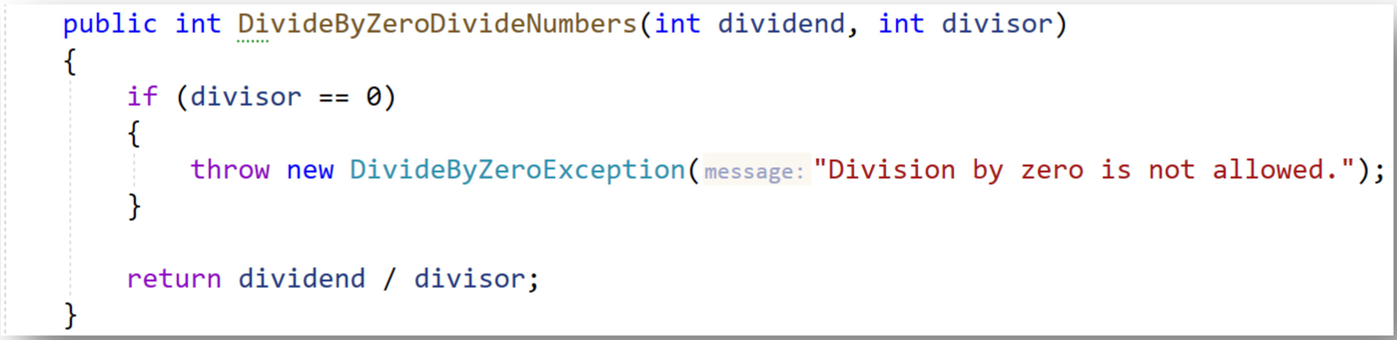

```
[Test]
0 references
public void Test_AddNumbers_NoOverflow_ReturnsSum()...

[Test]
0 references
public void Test_AddNumbers_PositiveOverflow_ThrowsOverflowException()...

[Test]
0 references
public void Test_AddNumbers_NegativeOverflow_ThrowsOverflowException()...
```

You are given **three empty** tests, and your task is to finish them. The tests should run when you're finished:

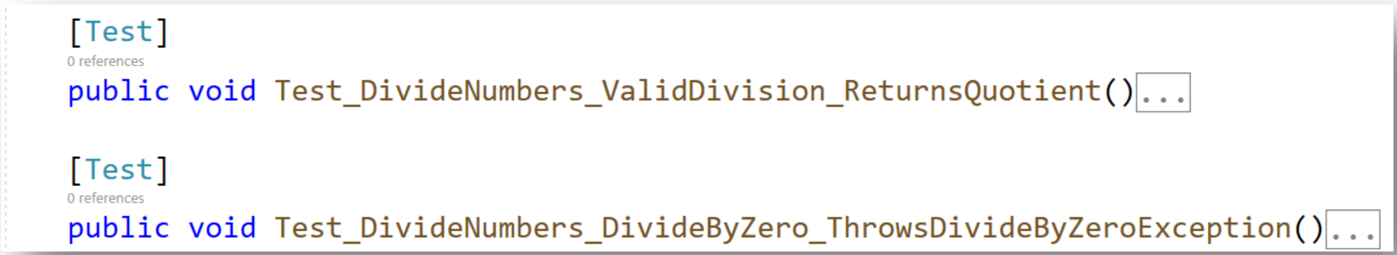

```
✓ Test_AddNumbers_NegativeOverflow_ThrowsOverflowException
✓ Test_AddNumbers_NoOverflow_ReturnsSum
✓ Test_AddNumbers_PositiveOverflow_ThrowsOverflowException
```

8. Unit Test: Divide Numbers – Divide by Zero Exception

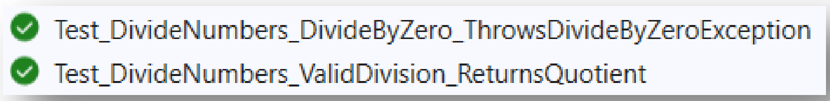
The **`DivideByZeroDivideNumbers()` method** takes in a **two numbers to be divided**. If the **divisor is 0** a **`DivideByZeroException`** is thrown:


```
public int DivideByZeroDivideNumbers(int dividend, int divisor)
{
    if (divisor == 0)
    {
        throw new DivideByZeroException(message: "Division by zero is not allowed.");
    }

    return dividend / divisor;
}
```

Now, look at the tests:


```
[Test]
0 references
public void Test_DivideNumbers_ValidDivision_ReturnsQuotient()...

[Test]
0 references
public void Test_DivideNumbers_DivideByZero_ThrowsDivideByZeroException()...
```

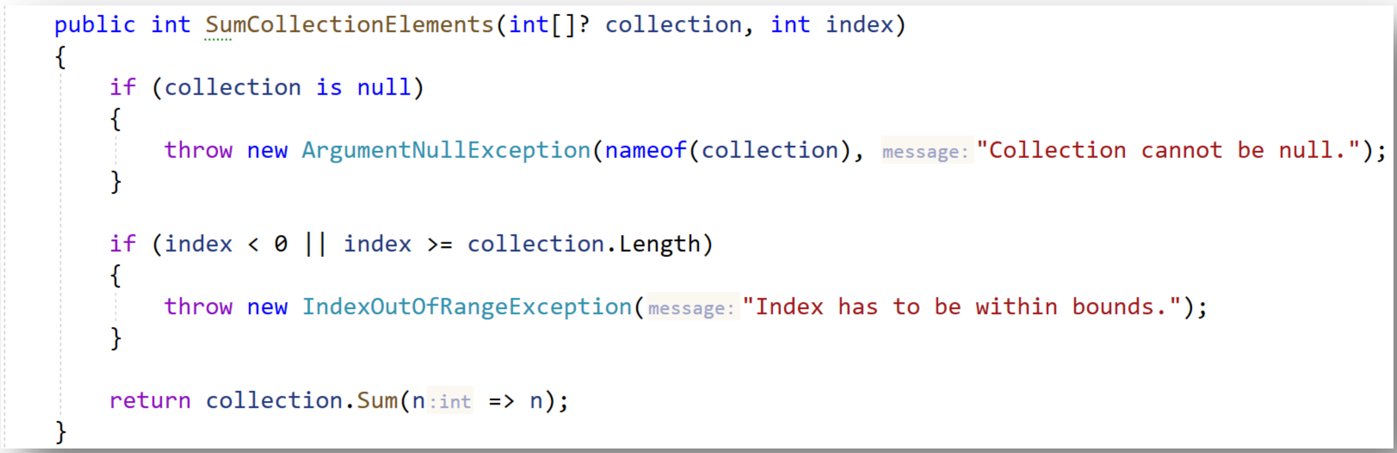
You are given **two empty** tests, and your task is to finish them. The tests should run when you're finished:



```
✓ Test_DivideNumbers_DivideByZero_ThrowsDivideByZeroException
✓ Test_DivideNumbers_ValidDivision_ReturnsQuotient
```

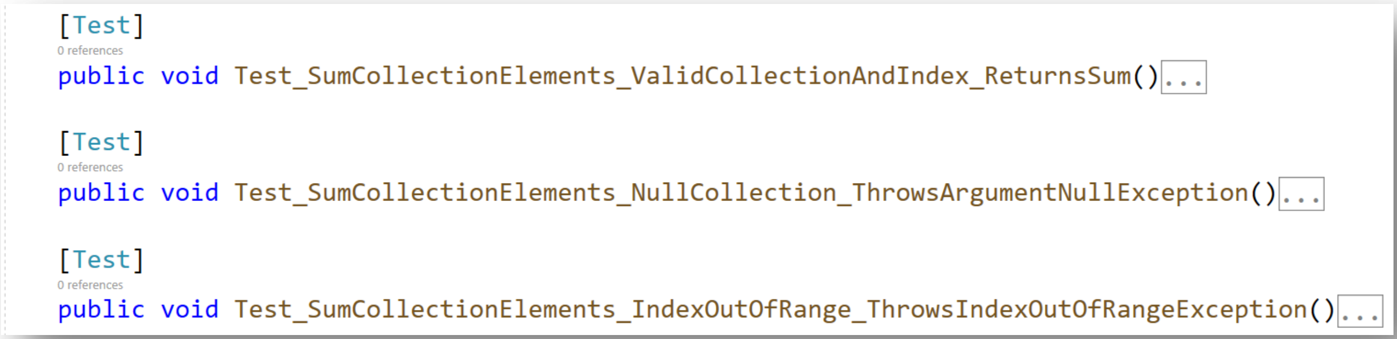
9. Unit Test: Sum Collection Elements

The **`SumCollectionElements()` method** takes in an **array of integers**, and an **index**. If the collection is **null** an **`ArgumentNullException`** is thrown, or if the **index is out of bounds** an **`IndexOutOfRangeException`** is thrown:



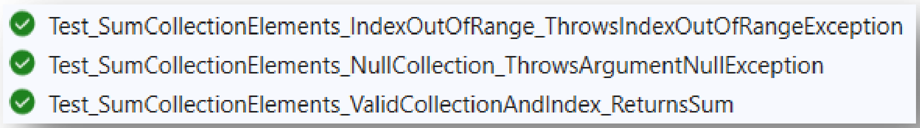
```
public int SumCollectionElements(int[]? collection, int index)
{
    if (collection is null)
    {
        throw new ArgumentNullException(nameof(collection), message: "Collection cannot be null.");
    }

    if (index < 0 || index >= collection.Length)
    {
        throw new IndexOutOfRangeException(message: "Index has to be within bounds.");
    }

    return collection.Sum(n:int => n);
}
```

Now, look at the tests:



```
[Test]
0 references
public void Test_SumCollectionElements_ValidCollectionAndIndex_ReturnsSum()...

[Test]
0 references
public void Test_SumCollectionElements_NullCollection_ThrowsArgumentNullException()...

[Test]
0 references
public void Test_SumCollectionElements_IndexOutOfRange_ThrowsIndexOutOfRangeException()...
```

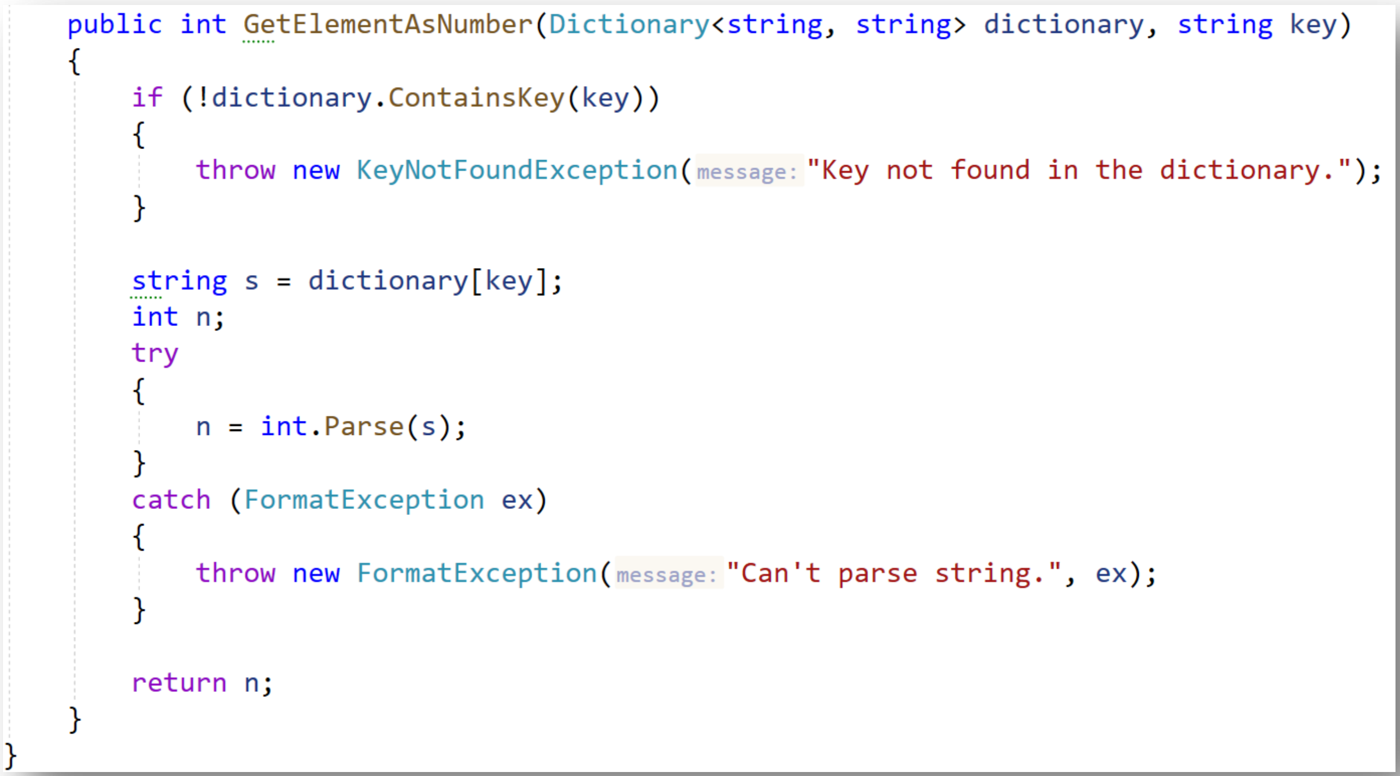
You are given **three empty** tests, and your task is to finish them. The tests should run when you're finished:



```
✓ Test_SumCollectionElements_IndexOutOfRange_ThrowsIndexOutOfRangeException
✓ Test_SumCollectionElements_NullCollection_ThrowsArgumentNullException
✓ Test_SumCollectionElements_ValidCollectionAndIndex_ReturnsSum
```

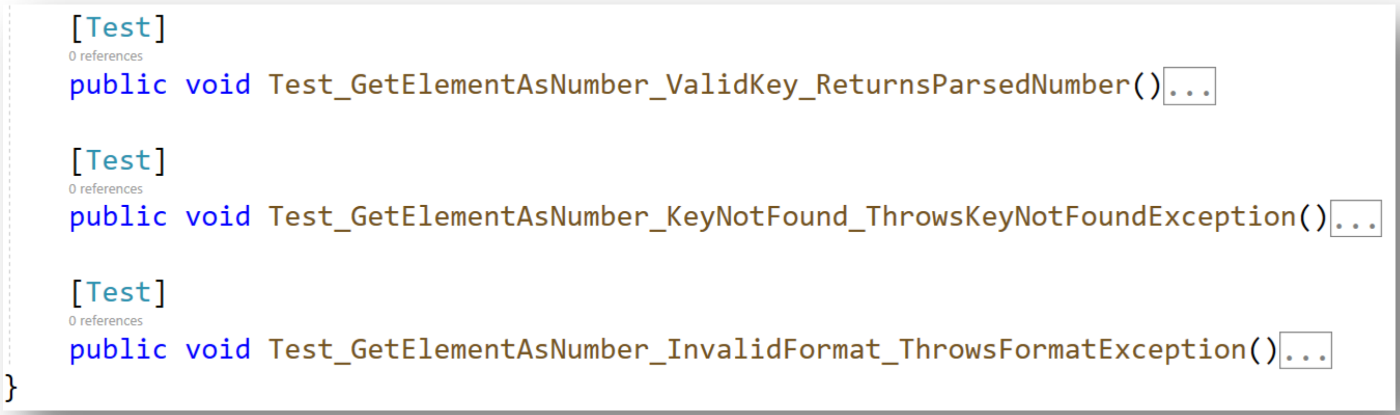
10. Unit Test: Get Element as Number

The **`GetElementAsNumber()` method** takes in a **dictionary**, and a **string representing a key from the dictionary**. If the key does not exist in the dictionary a **`KeyNotFoundException`** is thrown, if the value **cannot be converted to integer** a **`FormatException`** is thrown:



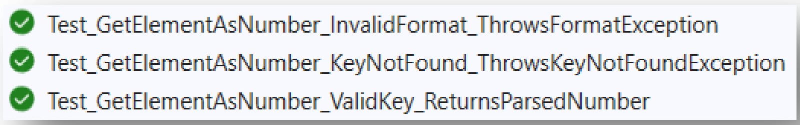
```
    public int GetElementAsNumber(Dictionary<string, string> dictionary, string key)
    {
        if (!dictionary.ContainsKey(key))
        {
            throw new KeyNotFoundException(message: "Key not found in the dictionary.");
        }

        string s = dictionary[key];
        int n;
        try
        {
            n = int.Parse(s);
        }
        catch (FormatException ex)
        {
            throw new FormatException(message: "Can't parse string.", ex);
        }

        return n;
    }
}
```

Now, look at the tests:



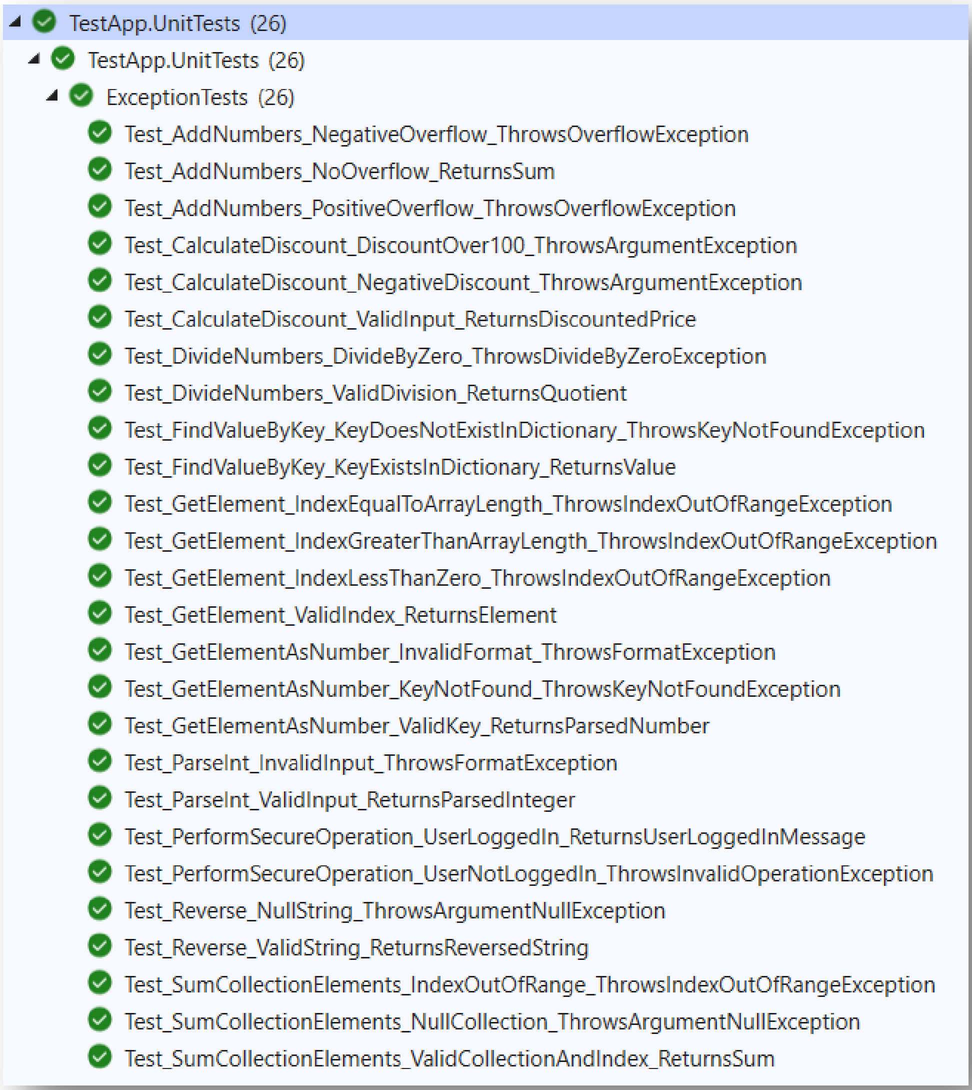
```
    [Test]
    0 references
    public void Test_GetElementAsNumber_ValidKey_ReturnsParsedNumber()...

    [Test]
    0 references
    public void Test_GetElementAsNumber_KeyNotFound_ThrowsKeyNotFoundException()...

    [Test]
    0 references
    public void Test_GetElementAsNumber_InvalidFormat_ThrowsFormatException()...
}
```

You are given **two empty** tests, and your task is to finish them. The tests should run when you're finished:



```
✓ Test_GetElementAsNumber_InvalidFormat_ThrowsFormatException
✓ Test_GetElementAsNumber_KeyNotFound_ThrowsKeyNotFoundException
✓ Test_GetElementAsNumber_ValidKey_ReturnsParsedNumber
```

At the end make sure all your tests run:



```
◢ ✓ TestApp.UnitTests (26)
  ◢ ✓ TestApp.UnitTests (26)
    ◢ ✓ ExceptionTests (26)
        ✓ Test_AddNumbers_NegativeOverflow_ThrowsOverflowException
        ✓ Test_AddNumbers_NoOverflow_ReturnsSum
        ✓ Test_AddNumbers_PositiveOverflow_ThrowsOverflowException
        ✓ Test_CalculateDiscount_DiscountOver100_ThrowsArgumentException
        ✓ Test_CalculateDiscount_NegativeDiscount_ThrowsArgumentException
        ✓ Test_CalculateDiscount_ValidInput_ReturnsDiscountedPrice
        ✓ Test_DivideNumbers_DivideByZero_ThrowsDivideByZeroException
        ✓ Test_DivideNumbers_ValidDivision_ReturnsQuotient
        ✓ Test_FindValueByKey_KeyDoesNotExistInDictionary_ThrowsKeyNotFoundException
        ✓ Test_FindValueByKey_KeyExistsInDictionary_ReturnsValue
        ✓ Test_GetElement_IndexEqualToArrayLength_ThrowsIndexOutOfRangeException
        ✓ Test_GetElement_IndexGreaterThanArrayLength_ThrowsIndexOutOfRangeException
        ✓ Test_GetElement_IndexLessThanZero_ThrowsIndexOutOfRangeException
        ✓ Test_GetElement_ValidIndex_ReturnsElement
        ✓ Test_GetElementAsNumber_InvalidFormat_ThrowsFormatException
        ✓ Test_GetElementAsNumber_KeyNotFound_ThrowsKeyNotFoundException
        ✓ Test_GetElementAsNumber_ValidKey_ReturnsParsedNumber
        ✓ Test_ParseInt_InvalidInput_ThrowsFormatException
        ✓ Test_ParseInt_ValidInput_ReturnsParsedInteger
        ✓ Test_PerformSecureOperation_UserLoggedIn_ReturnsUserLoggedInMessage
        ✓ Test_PerformSecureOperation_UserNotLoggedIn_ThrowsInvalidOperationException
        ✓ Test_Reverse_NullString_ThrowsArgumentNullException
        ✓ Test_Reverse_ValidString_ReturnsReversedString
        ✓ Test_SumCollectionElements_IndexOutOfRange_ThrowsIndexOutOfRangeException
        ✓ Test_SumCollectionElements_NullCollection_ThrowsArgumentNullException
        ✓ Test_SumCollectionElements_ValidCollectionAndIndex_ReturnsSum
```