**Exercise 1: Using Enumerations to Specify Domains**

**In this exercise, you will define enumerations that represent different materials under stress (stainless steel,aluminum, reinforced concrete, and titanium) and the cross-section of the girders (I-Beam, Box, Z-Shaped,and C-Shaped). You will also define another enumeration called TestResult that represents the results of astress test.**

The main tasks for this exercise are as follows:

|  |  |
| --- | --- |
| 1. | Open the Enumeration solution. |
| 2. | Add enumerations to the **StressTest** namespace. |
| 3. | Retrieve the enumeration values. |
| 4. | Display the selection results. |
| 5. | Test the solution. |

**Task 1: Open the Enumerations solution**

|  |  |
| --- | --- |
| 1. | Open the Enumerations solution in the Ex1\Starter folder. |

**Task 2: Add enumerations to the StressTest namespace**

|  |  |  |
| --- | --- | --- |
| 1. | Review the task list. | |
| 2. | Locate the **TODO - Implement Material, CrossSection, and TestResult enumerations** task, and thendouble-click this task. This task is located in the StressTestType.cs file. |
| 3. | In the **StressTest** namespace, define a new enumeration named **Material**. The enumeration should have thefollowing values:   |  |  | | --- | --- | | a. | **StainlessSteel** | | b. | **Aluminum** | | c. | **ReinforcedConcrete** | | d. | **Composite** | | e. | **Titanium** | |
| 4. | Below the **Material** enumeration, define a new enumeration named **CrossSection**. The enumeration shouldhave the following values:   |  |  | | --- | --- | | a. | **IBeam** | | b. | **Box** | | c. | **ZShaped** | | d. | **CShaped** | |
| 5. | Below the **CrossSection** enumeration, define a new enumeration named **TestResult**. The enumerationshould have the following values:   |  |  | | --- | --- | | a. | **Pass** | | b. | **Fail** | |
| 6. | Build the solution and correct any errors. |

**Task 3: Retrieve the enumeration values**

|  |  |
| --- | --- |
| 1. | In the TestHarness project, display the MainWindow.xaml window.  The purpose of the TestHarness project is to enable you to display the values from each of the enumerations.When the application runs, the three lists are populated with the values that are defined for each of theenumerations. The user can select an item from each list, and the application will construct a string from the corresponding enumerations. |
| 2. | In the task list, locate the **TODO - Retrieve user selections from the UI** task, and then double-click thistask. This task is located in the **MainWindow**.**xaml**.**cs** class. |
| 3. | Remove the comment, and add code to the **selectionChanged** method to perform the following tasks:   |  |  | | --- | --- | | a. | Create a **Material** object called **selectedMaterial** and initialize it to the  value of the **SelectedItem** property in the **materials** list box. | | b. | Create a **CrossSection** object called **selectedCrossSection** and initialize it to  the value of the**SelectedItem** property in the **crosssections** list box. | | c. | Create a **TestResult** object called **selectedTestResult** and initialize it to  the value of the **SelectedItem**property in the **testresults** list box.  **Hint:** The **SelectedItem** property of a **ListBox** control has the **object** type.  You must cast thisproperty to the appropriate type when you assign it to an enumeration variable. | |

**Task 4: Display the selection results**

|  |  |
| --- | --- |
| 1. | In the **selectionChanged** method, after the code that you added in the previous task, add a statement tocreate a new **StringBuilder** object named **selectionStringBuilder**. |

|  |  |
| --- | --- |
| 2. | Add a **switch** statement to evaluate the selectedMaterial variable. In the **switch** statement, add **case**statements for each potential value of the **Material** enumeration. In each **case** statement, add code to appendthe text "Material: *<selectedMaterial>*, " to the **selectionStringBuilder** object. Substitute the text "*<selectedMaterial>*" in this string with the corresponding value for the selectedMaterial variable that isshown in the following table. |
| **Material enumeration value** | **<*selectedMaterial*> string** |
| **Material.StainlessSteel** | Stainless Steel |
| **Material.Aluminum** | Aluminum |
| **Material.ReinforcedConcrete** | Reinforced Concrete |
| **Material.Composite** | Composite |
| **Material.Titanium** | Titanium |

|  |  |
| --- | --- |
| 3. | Add another **switch** statement to evaluate the selectedCrossSection variable. In this **switch** statement, add**case** statements for each potential value of the **CrossSection** enumeration. In each **case** statement, add codeto append the text "Cross-section: *<selectedCrossSection>*," to the **selectionStringBuilder** object.Substitute the text "*<selectedCrossSection>*" in this string with the corresponding value for theselectedCrossSection variable that is shown in the following table. |
| **Material enumeration value** | **<*selectedCrossSection*> string** |
| **CrossSection**.**IBeam** | I-Beam |
| **CrossSection**.**Box** | Box |
| **CrossSection**.**ZShaped** | Z-Shaped |
| **CrossSection**.**CShaped** | C-Shaped |

|  |  |
| --- | --- |
| 4. | Add a final **switch** statement to evaluate the **selectedTestResult** member. In the **switch** statement, add **case**statements for each potential value of the **TestResult** enumeration. In each **case** statement, add code toappend the text "Result: *<selectedTestResult>*." to the **selectionStringBuilder** object. Substitute the text "*<selectedTestResult>*" in this string with the corresponding value for the selectedTestResult variable that isshown in the following table. |
| **Material enumeration value** | | **<*selectedTestResult*> string** |
| **TestResult.Pass** | | Pass |
| **TestResult.Fail** | | Fail |

|  |  |
| --- | --- |
| 5. | At the end of the **selectionChanged** method, add code to display the string that is constructed by using the**selectionStringBuilder** object in the **Content** property of the **testDetails** label. |

**Task 5: Test the solution**

|  |  |  |
| --- | --- | --- |
| 1. | Build the application and correct any errors. | |
| 2. | Run the application. | |
| 3. | In the MainWindow window, in the **Material** list, click **Titanium**, in the **CrossSection** list, click **Box**, andthen in the **Result** list, click **Fail**.  At the bottom of the window, verify that the label updates with your selections. |
| 4. | Experiment by selecting further values from all three lists, and verify that with each change, the labelupdates to reflect the changes. |
| 5. | Close the application, and then return to Visual Studio. |

**Exercise 2: Using a Struct to Model a Simple Type**

**In this exercise, you will define a type called TestCaseResult that holds the result of a stress test. It will havethe following public fields:**

|  |  |
| --- | --- |
| • | **Result** : **TestResult** |
| • | **ReasonForFailure**: **string** |

This type is small, so it is best implemented as a struct. You will provide a constructor that initializes these fields.

The main tasks for this exercise are as follows:

|  |  |
| --- | --- |
| 1. | Open the Structures solution. |
| 2. | Add the **TestCaseResult** structure. |
| 3. | Add an array of **TestCaseResult** objects to the user interface project. |
| 4. | Fill the **results** array with data. |
| 5. | Display the array contents. |
| 6. | Test the solution. |

**Task 1: Open the Structures solution**

|  |  |
| --- | --- |
| • | Open the Structures solution in the E:\Labfiles\Lab 6\Ex2\Starter folder. |

**Task 2: Add the TestCaseResult structure**

|  |  |  |
| --- | --- | --- |
| 1. | Review the task list: | |
| 2. | In the task list, locate the **TODO - Declare a Structure** task, and then double-click this task. This task islocated in the StressTestTypes.cs file. | |
| 3. | Delete the comment, and then declare a new structure named **TestCaseResult**. In the **TestCaseResult**structure, add the following members:   |  |  | | --- | --- | | a. | A **TestResult** object named **Result**. | | b. | A **string** object named **ReasonForFailure**. | |

**Task 3: Add an array of TestCaseResult objects to the user interface project**

|  |  |
| --- | --- |
| 1. | In the TestHarness project, display the MainWindow.xaml window.  This project simulates running stress tests and displays the results. It tracks the number of successful andfailed tests, and for each failed test, it displays the reason for the failure. |
| 2. | In the task list, locate the **TODO - Declare a TestCaseResult array** task, and then double-click this task. |
| 3. | Remove the comment, and then declare a new array of **TestCaseResult** objects named **results**. |

**Task 4: Fill the results array with data**

|  |  |
| --- | --- |
| 1. | In the **RunTests**\_**Click** method, after the statement that clears the **reasonsList** list, add code to initialize the**results** array. Set the array length to **10**. |
| 2. | Below the statement that creates the array, add code that iterates through the items in the array and populateseach one with the value that the static **GenerateResult** method of the **TestManager** class returns. The**GenerateResult** method simulates running a stress test and returns a **TestCaseResult** object that containsthe result of the test and the reason for any failure. |

**Task 5: Display the array contents**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| • | Locate the comment **TODO - Display the TestCaseResult data**. Delete the comment, and then add code thatiterates through the **results** array. For each value in the array, perform the following tasks:   |  |  | | --- | --- | | a. | Evaluate the **result** value. If the **result** value is **TestResult**.**Pass**, increment the **passCount** value. | | b. | If the **result** value is **TestResult.Fail**, increment the **failCount** value, and add the **ReasonForFailure** stringto the **reasonsList** list box that is displayed in the window.  **Note:** To add an item to a list box, you use the **ListBox**.**Items**.**Add** method and pass the item to addto the list as a parameter to the method. | |

**Task 6: Test the solution**

|  |  |
| --- | --- |
| 1. | Build the application and correct any errors. |
| 2. | Run the application. |
| 3. | In the MainWindow window, click **Run Tests**.  Verify that the **Successes** and **Failures** messages are displayed. Also verify that a message appears in the**Failures** list if failures occur. |
| 4. | Click **Run Tests** again to simulate running another batch of tests and display the results of these tests. |
| 5. | Close the application, and then return to Visual Studio. |

**Exercise 3: Using a Class to Model a More Complex Type**

**In this exercise, you will define another type called StressTestCase that represents a stress test case for agirder. This type will be more complex than the TestCaseResult struct and is best implemented as a class.The StressTestCase class will have the following public data members:**

|  |  |
| --- | --- |
| • | **girderMaterial**: **MaterialType** |
| • | **crossSection**: **CrossSection** |
| • | **lengthInMm**: **int** |
| • | **heightInMm**: **int** |
| • | **widthInMm**: **int** |
| • | **testCaseResult**: **TestCaseResult** |

You will also define two constructors: a default constructor that initializes these fields (apart from **testCaseResult**)to default values and an overloaded constructor that enables a programmer to specify nondefault values. You willthen add the following public methods to the class:

|  |  |
| --- | --- |
| • | **PerformStressTest**. This method will simulate performing a stress test and set the result to indicate whetherthe test passed or failed, together with a reason for failure. |
| • | **GetStressTestResult**. This method will return the value of the **testCaseResult** field. |
| • | **ToString**. This method will return a representation of the object as a string for display purposes. |

The main tasks for this exercise are as follows:

|  |  |
| --- | --- |
| 1. | Open the Classes solution. |
| 2. | Define the **StressTestCase** class. |
| 3. | Add a parameterized constructor and a default constructor to the class. |
| 4. | Add the **PerformStressTest** and **GetStressTestResult** methods to the class. |
| 5. | Override the **ToString** method to return a custom string representation |
| 6. | Create an array of **StressTestCase** objects. |
| 7. | Display the **StressTestCases** collection. |
| 8. | Test the solution. |
| 9. | Examine and run unit tests. |

**Task 1: Open the Classes solution**

|  |  |
| --- | --- |
| • | Open the Classes solution in the E:\Labfiles\Lab 6\Ex3\Starter folder. |

**Task 2: Define the StressTestCase class**

|  |  |
| --- | --- |
| 1. | In the TestHarness project, display the MainWindow.xaml window.  This project is an extended version of the test harness from the previous two exercises. In addition tosimulating stress-test results, it displays the details of the girder under test. |
| 2. | Review the task list. |
| 3. | In the task list, locate the **TODO - Add the StressTestCase class** task, and then double-click this task. |
| 4. | Remove the comment, and then add code to declare a public class named **StressTestCase** with the followingpublic members:   |  |  | | --- | --- | | a. | A **Material** object named **GirderMaterial**. | | b. | A **CrossSection** object named **CrossSection**. | | c. | An integer named **LengthInMm**. | | d. | An integer named **HeightInMm**. | | e. | An integer named **WidthInMm**. | | f. | A **TestCaseResult** object named **TestCaseResult**. | |

**Task 3: Add a parameterized constructor and a default constructor to the class**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1. | Below the member declarations, add a constructor for the **StressTestCase** class that accepts the followingparameters:   |  |  | | --- | --- | | a. | A **Material** object named **girderMaterial**. | | b. | A **CrossSection** object named **crossSection**. | | c. | An integer named **lengthInMm**. | | d. | An integer named **heightInMm**. | | e. | An integer named **widthInMm**. |   In the constructor, add code to store the value for each parameter in the corresponding member.  **Hint:** In the constructor, to make it clear which items are member variables and which items areparameters, use the **this** keyword (which represents the current object) with all member variables. |
| 2. | Above the constructor, add a default constructor.  **Hint:**A default constructor is a constructor that accepts no parameters and implements functionalityto create a default instance of a class. |

In the default constructor, initialize the members of the **StressTestCase** object with default values by using theparameterized constructor and the data that are shown in the following table.

| **Parameter name** | **Value** |
| --- | --- |
| *girderMaterial* | **Material.StainlessSteel** |
| *crossSection* | **CrossSection.IBeam** |
| *lengthInMm* | **4000** |
| *heightInMm* | **20** |
| *widthInMm* | **15** |

**Hint:** Remember that you can invoke one constructor directly from another by using the syntax in thefollowing code example.

public MyDefaultConstructor() : this(parameter1, parameter2, ...)

{

...

}

**Task 4: Add the PerformStressTest and GetStressTestResult methods to the class**

|  |  |
| --- | --- |
| 1. | Below the class constructors, add code to declare a new method named **PerformStressTest**. The**PerformStressTest** method should take no parameters and should not return a value.  This method will simulate performing a stress test and then populate a **StressTestCase** object with thedetails of the test. |
| 2. | In the **PerformStressTest** method, create an array of strings called **failureReasons** that contains thefollowing values:   |  |  | | --- | --- | | a. | **"Fracture detected"** | | b. | **"Beam snapped"** | | c. | **"Beam dimensions wrong"** | | d. | **"Beam warped"** | | e. | **"Other"** | |
| 3. | Add a statement that invokes the **Next** method of the static **Rand** method of the **Utility** class. Pass the value**10** as a parameter.  **Note:** The **Utility.Rand.Next** method accepts an integer parameter and then returns a random integervalue between zero and the value of the integer parameter. In this case, the method will return aninteger between 0 and 9. |

If the value that the **Rand** method returns is 9, add code to perform the following tasks:

|  |  |  |
| --- | --- | --- |
| a. | Set the **TestCaseResult**.**Result** member value to **TestResult.Fail**. | |
| b. | Invoke the **Utility.Rand.Next** method with a parameter value of *5*. Store the result in a new integer membernamed **failureCode**. |
| c. | Set the **TestCaseResult.ReasonForFailure** value to the value in the **failureReasons** array that the**failureCode** value indicates.  **Note:** This code simulates a 10 percent chance of a test case failing. The **failureReasons** arraycontains five possible causes of failure, and this code selects one of these causes at random. |
| 4. | If the **Rand** method returns a value other than 9, add code to set the **TestCaseResult.Result** member valueto **TestResult.Pass**. |
| 5. | Below the **PerformStressTest** method, add a public method named **GetStressTestResult**, which accepts noparameters and returns a **TestCaseResult** object. |
| 6. | In the **GetStressTestResult** method, add code to return a reference to the **TestCaseResult** member. |

**Task 5: Override the ToString method to return a custom string representation**

|  |  |
| --- | --- |
| 1. | Below the **GetStressTestResult** method, add the following public method named **ToString**.  **Note:** This overrides the **ToString** method that is inherited from the **object** type. You will see moreabout inheritance in a later module.  ...  public class StressTestCase  {  ...  **public override string ToString()**  **{**  **}**  }  ... |
| 2. | In the **ToString** method, add code to return a string with the format shown in the following code example,where each value in angle brackets is replaced with the corresponding member in the class. |

Material: <girderMaterial>, CrossSection: <crossSection>, Length: <lengthInMm>mm,

Height: <heightInMm>mm, Width:<widthInMm>mm.

**Hint:**Use the **String.Format** method to build the string.

**Task 6: Create an array of StressTestCase objects**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1. | In the task list, locate the **TODO - Create an array of sample StressTestCase objects** task, and thendouble-click this task. This task is located in the **MainWindow.xaml.cs** class. | | | | |
| 2. | Remove the comment, and add a private method named **CreateTestCases**. The **CreateTestCases** methodshould accept no parameters and return an array of **StressTestCase** objects. | | | | |
| 3. | In the **CreateTestCases** method, add code to create an array of **StressTestCase** objects named**stressTestCases**. The array should be able to hold 10 objects. | | | | |
| 4. | Add code to generate 10 **StressTestCase** objects, and store each of them in the **stressTestCases** array. Usethe following table to determine the parameters to pass to the constructor for each instance. | | | | |
| **Array   position** | **Material** | **CrossSection** | **Length** | **Height** | **Width** |
| 0 | Use default constructor | | | | |
| 1 | **Material.Composite** | **CrossSection.CShaped** | 3500 | 100 | 20 |
| 2 | Use default constructor | | | | |
| 3 | **Material.Aluminium** | **CrossSection.Box** | 3500 | 100 | 20 |
| 4 | Use default constructor | | | | |
| 5 | **Material.Titanium** | **CrossSection.CShaped** | 3600 | 150 | 20 |
| 6 | **Material.Titanium** | **CrossSection.ZShaped** | 4000 | 80 | 20 |
| 7 | **Material.Titanium** | **CrossSection.Box** | 5000 | 90 | 20 |
| 8 | Use default constructor | | | | |
| 9 | **Material.StainlessSteel** | **CrossSection.Box** | 3500 | 100 | 20 |

|  |  |
| --- | --- |
| 5. | At the end of the method, return the **stressTestCases** array. |

**Task 7: Display the StressTestCases collection**

|  |  |
| --- | --- |
| 1. | In the task list, locate the **TODO - Iterate through the StressTestCase samples displaying the results**task, and then double-click this task. This task is located in the **doTests\_Click** method that runs when theuser clicks **Run Stress Tests**. |
| 2. | Remove the comment, and then add code to invoke the **CreateTestCases** method. Store the result of themethod call in a new array of **StressTestCase** objects named **stressTestCases**. |
| 3. | Add code to create a **StressTestCase** object named **currentTestCase** and a **TestCaseResult** object named**currentTestResult**. You will add code to instantiate these objects shortly. |
| 4. | Add code that iterates through the **StressTestCase** objects in the **stressTestCases** array. For each**StressTestCase** object, add code to perform the following tasks:   |  |  | | --- | --- | | a. | Set the **currentTestCase** object to refer to the **StressTestCase** object. | | b. | Invoke the **currentTestCase**.**PerformStressTest** method on the **currentTestCase** object. | | c. | Add the **currentTestCase** object to the **testList** list box that is displayed in the window. | | d. | Invoke the **currentTestCase**.**GetStressTestResult** method, and store the result in the**currentTestResult** object. | | e. | Add a string to the **resultList** list box that is displayed in the window. This string should consist of the**currentTestResult**.**Result** value and the **currentTestResult**.**ReasonForFailure** message. | |

**Task 8: Test the solution**

|  |  |  |
| --- | --- | --- |
| 1. | Build the solution and correct any errors. | |
| 2. | Run the application. | |
| 3. | In the MainWindow window, click **Run Stress Tests**.  Verify that the **Girder Tested** list contains a list of different girder compositions and the **Results** listcontains a series of test results. |
| 4. | Click **Run Stress Tests** again. You should see a different set of results. | |
| 5. | Close the application, and then return to Visual Studio | |

**Task 9: Examine and run unit tests**

|  |  |
| --- | --- |
| 1. | In the task list, locate the **TODO - Examine and Run Unit Tests** task, and then double-click this task. Thistask is located in the **StressTestCaseTest** class. |
| 2. | Examine the **StressTestCaseConstructorTest** method.  This method uses the parameterized constructor to create a new **StressTestCase** object the uses definedvalues. The method then uses a series of **Assert** statements to ensure that the properties of the created objectmatch the values that are passed to the constructor. |
| 3. | Examine the **StressTestCaseConstructorTest1** method.  This method uses the default constructor to create a new **StressTestCase** object, passing no parameters. Themethod then uses a series of **Assert** statements to ensure that the properties of the created object match theintended default values. |
| 4. | Examine the **GetStressTestResultTest** method.  This method creates a new **StressTestCase** object and then retrieves a **TestCaseResult** object by calling the**StressTestCase**.**GetStressTestResult** method. The test method then uses **Assert** statements to ensure thatthe **TestCaseResult.Result** and **TestCaseResult.ReasonForFailure** properties contain the expected values. |
| 5. | Examine the **PerformStressTestTest** method.  This method creates a **StressTestCase** object, calls the **PerformStressTest** method, and then retrieves the**TestCaseResult**object. The method then checks that, if the test failed, the**TestCaseResult.ReasonForFailure** member contains some text. If the test passed, the method uses **Assert**statements to verify that the **ReasonForFailure** member contains no data. The method iterates 30 times. |
| 6. | Examine the **ToStringTest** method.  This method creates a default **StressTestCase** object, and then verifies that the object's **ToString** methodreturns a string that contains the correct details. |
| 7. | Run all of the tests in the solution, and verify that all of the tests execute successfully. | |

**Exercise 4: Using a Nullable Struct**

**In this exercise, you will modify the constructor for the StressTestCase class to initialize the testCaseResultfield to null (to indicate no result yet). However, you cannot set a value-type field or variable to a referencevalue such as null. Therefore, you will convert the testCaseResult field to a nullable field to support nullvalues. You will then modify the methods in the StressTestCase class, and the test harness that invokes theGetStressTestResult method that displays the result of a test case, to dereference the value of this structthrough the Value property.**

The main tasks for this exercise are as follows:

|  |  |
| --- | --- |
| 1. | Open the NullableStructs solution. |
| 2. | Modify the **TestCaseResult** field to make it nullable. |
| 3. | Modify the parameterized constructor to initialize the **TestCaseResult** member. |
| 4. | Modify the **PerformStressTest** method. |
| 5. | Modify the **GetStressTestResult** method. |
| 6. | Modify the **GetStressTestResult** method call. |
| 7. | Test the solution. |
| 8. | Update the unit tests. |

**Task 1: Open the NullableStructs solution**

|  |  |
| --- | --- |
| • | Open the NullableStructs solution in the E:\Labfiles\Lab 6\Ex4\Starter folder. |

**Task 2: Modify the TestCaseResult field to make it nullable**

|  |  |  |
| --- | --- | --- |
| 1. | Review the task list. | |
| 2. | In the task list, locate the **TODO - Make TestCaseResult nullable** task, and then double-click this task.This task is located in the **StressTestTypes** class. |
| 3. | Remove the comment, and then modify the **TestCaseResult** member definition to allow it to store a nullvalue. |

**Task 3: Modify the parameterized constructor to initialize the TestCaseResult member**

|  |  |
| --- | --- |
| • | In the **StressTestCase** parameterized constructor, remove the comment **TODO – Initialize TestCaseResult tonull**, and then add code to initialize the **TestCaseResult** member to null. |

**Task 4: Modify the PerformStressTest method**

|  |  |
| --- | --- |
| 1. | In the **PerformStressTest** method, remove the comment **TODO – Update the PerformStressTest methodand work with the nullable type**, and then add code to declare a new TestCaseResult variable namedcurrentTestCase. |
| 2. | Modify the **if** statement to perform the following tasks:   |  |  | | --- | --- | | a. | In all instances, modify the **currentTestCase** object rather than the **TestCaseResult** member. | | b. | At the end of the **if** block, assign the **currentTestCase** object to the **TestCaseResult** member. | |
| 3. | Modify the **else** block to perform the following tasks:   |  |  | | --- | --- | | a. | Modify the **currentTestCase** object rather than the **TestCaseResult** member. | | b. | At the end of the **if** block, store the **currentTestCase** object in the **TestCaseResult** member. | |

**Task 5: Modify the GetStressTestResult method**

|  |  |
| --- | --- |
| • | In the **GetStressTestResult** method, modify the method definition to return a nullable **TestCaseResult** value. |

**Task 6: Modify the GetStressTestResult method call**

|  |  |
| --- | --- |
| 1. | In the task list, locate the **TODO - Modify call to GetStressTestResult method to handle nulls** task, andthen double-click this task. |
| 2. | Remove the comment, and then modify the code to create a nullable **TestCaseResult** object named**currentTestResult**. |
| 3. | In the **for** block, after retrieving the value of the **currentTestResult** object from the**currentStressTest**.**GetStressTestResult** method, add code to check whether the **currentTestResult** objectcontains a value. If a value exists, add a string that contains the **StressTestResult** **Result** and**ReasonForFailure** properties to the **resultList** list box. |

**Task 7: Test the solution**

|  |  |
| --- | --- |
| 1. | Build the solution and correct any errors. |
| 2. | Run the application. |
| 3. | In the MainWindow window, click **Run Stress Tests**.  Verify that the application functions in the same way as before. |
| 4. | Close the application, and then return to Visual Studio. |

**Task 8: Update the unit tests**

|  |  |  |
| --- | --- | --- |
| 1. | In the task list, locate the **TODO - Examine and run unit tests updated to deal with nullable type** task,and then double-click this task. This task is located in the **StressTestCaseTest** class.  **Note:** Most of the test cases are identical to those in Exercise 3. The only changes are in the**GetStressTestResult** and **PerformStressTestTest** methods. | |
| 2. | Examine the **GetStressTestResult** method.  This method creates a new **StressTestCase** object. It then evaluates the **HasValue** property on the result ofthe **GetStressTestResult** method call to verify that property contains no value. The test then calls the**PerformStressTest** method, which generates a **TestCaseResult** value in the **StressTestCase** object. The testmethod again evaluates the **HasValue** property to verify that a value now exists. |
| 3. | Examine the changes to the **PerformStressTestTest** method.  This method creates a **StressTestCase** object and then calls the **PerformStressTest** method on that object.The method calls the **GetStressTestResult** method on the **StressTestCase** object and stores the result in alocal nullable **TestCaseResult** object. The method then uses an **Assert** statement to evaluate the **HasValue**property of the **TestCaseResult** object to verify that the result is not null. The method then evaluates the**Value** property of the **TestCaseResult** object to determine whether the result indicates that the stress testfailed or passed. If the stress test failed, an **Assert** statement is used to verify that the **ReasonForFailure**string contains a value. If the stress test passed, an **Assert** statement is used to verify that the**ReasonForFailure** string is null. The method iterates 30 times. |
| 4. | Run all of the tests in the solution, and verify that all of the tests execute successfully. |
| 5. | Close Visual Studio. | | |