Lab 7: Encapsulating Data and Methods

Exercise 1: Hiding Data Members

Task 1: Open the StressTesting solution

1. Log on to the 10266A-GEN-DEV virtual machine as **Student** with the password **Pa$$w0rd**.
2. Open Microsoft Visual Studio 2010:
   - Click **Start**, point to **All Programs**, click **Microsoft Visual Studio 2010**, and then click **Microsoft Visual Studio 2010**.
3. Open the StressTesting solution in the E:\Labfiles\Lab 7\Ex1\Starter folder:
   a. In Visual Studio, on the **File** menu, point to **Open**, and then click **Project/Solution**.
   b. In the **Open Project** dialog box, move to the **E:\Labfiles\Lab 7\Ex1 \Starter** folder, click **StressTesting.sln**, and then click **Open**.

Task 2: Declare fields in the StressTestCase class as private

1. Review the task list:
   a. If the task list is not already visible, on the **View** menu, click **Task List**.
   b. If the **Task List** is displaying **User Tasks**, in the drop-down list box click **Comments**.
2. In the task list, locate the **TODO - Modify the StressTestCase class to make members private** task, and then double-click this task. This task is located in the **StressTestCase** class.
3. In the **StressTestCase** class, remove the **TODO - Modify the StressTestCase class to make members private** comment, and then modify each field definition to make all of the fields private.

   Your code should resemble the following code example.

```
...
/// <summary>
/// Girder material type (enumeration type)
/// </summary>
private Material girderMaterial;

/// <summary>
/// Girder cross-section (enumeration type)
/// </summary>
private CrossSection crossSection;

/// <summary>
/// Girder length in millimeters
/// </summary>
private int lengthInMm;

/// <summary>
/// Girder height in millimeters
/// </summary>
private int heightInMm;

/// <summary>
/// Girder width in millimeters
/// </summary>
private int widthInMm;

/// <summary>
/// Details of test result (structure type)
/// Made nullable
/// </summary>
private TestCaseResult? testCaseResult;
...
```

Task 3: Build the project and correct errors

1. Build the project, and then review the error list.

   The project should fail to build because the code in the **doTests_Click** method in the test harness project attempts to access the fields in the **StressTestCase** class that are now private:

   a. On the **Build** menu, click **Build Solution**.

   b. If the error list is not automatically displayed, on the **View** menu, click **Error List**.

   c. If the error list is not showing errors, in the error list pane, click **Errors**.

2. Comment out the code that caused the errors that are shown in the error list. These errors are caused by six statements in the **doTests_Click** method:

   a. In the error list, double-click the first error. This error is located in the StressTest Test Harness solution, in the MainWindow.xaml.cs file.

   b. In the **MainWindow** class, in the **doTests_Click** method, comment out the six lines of code that raise errors.

   Your code should resemble the following code example.

```
...
private void doTests_Click(object sender, RoutedEventArgs e)
{
    ...
    //Material m = stc.girderMaterial;
    //CrossSection c = stc.crossSection;
    //int l = stc.lengthInMm;
    //int h = stc.heightInMm;
    //int w = stc.widthInMm;
    //tcr = stc.testCaseResult.Value;

    stc.PerformStressTest();
    ...
}
...
```

Task 4: Update unit tests to resolve errors

1. On the **Build** menu, click **Build Solution**. There should still be some errors.

   The remaining errors are located in the unit test project.

2. In the task list, locate the **TODO - Update unit tests to resolve errors** task, and then double-click this task. This task is located in the **StressTestCaseTest** unit test class.

3. In the **StressTestCaseConstructorTest** method, comment out the five **Assert** statements that cause errors.

   Your code should resemble the following code example.

```
...
public void StressTestCaseConstructorTest()
{
    ...
    //Assert.AreEqual(Material.Composite, target.girderMaterial);
    //Assert.AreEqual(CrossSection.CShaped, target.crossSection);
    //Assert.AreEqual(5000, target.lengthInMm);
    //Assert.AreEqual(32, target.heightInMm);
    //Assert.AreEqual(18, target.widthInMm);
}
...
```

4. Update the method to verify that the constructed object contains the correct member values by performing the following tasks:

Hint: You cannot access the member data directly because you have just declared private members. The **ToString** method returns a string representation of the object, including the member data.

   a. Before you instantiate the **target** object, declare a new string named **expected** and populate the string with the following data that represents the expected results of the test.

```
Material: Composite, CrossSection: CShaped, Length: 5000mm, Height:
32mm, Width: 18mm, No Stress Test Performed
```

Your code should resemble the following code example.

```
public void StressTestCaseConstructorTest()

    {

    ...

    string expected = "Material: Composite, CrossSection: CShaped,
Length: 5000mm, Height: 32mm, Width: 18mm, No Stress Test Performed";
    StressTestCase target = new StressTestCase(
        girderMaterial,
        crossSection,
        lengthInMm,
        heightInMm,
        widthInMm);

    ...
}
```

   b. At the end of the method, add an **Assert** statement that checks whether the **expected** string matches the output of the **target.ToString** method.

Your code should resemble the following code example.

```
public void StressTestCaseConstructorTest()
    {
    ...
    StressTestCase target = new StressTestCase(
        girderMaterial,
        crossSection,
        lengthInMm,
        heightInMm,
        widthInMm);
    ...
    Assert.AreEqual(expected, target.ToString());
}
```

5. Update the **StressTestCaseConstructorTest1** method and resolve the errors by performing the following tasks:

   a. Comment out the five existing **Assert** statements.

   b. Before the method creates the **target** object, create a new string that contains the expected result from a default **StressTestCase** class. This string is the same as the string that the previous test expects.

   c. At the end of the method, add an **Assert** statement that checks whether the **expected** string matches the output of the **target.ToString** method.

   Your code should resemble the following code example.

```
public void StressTestCaseConstructorTest1()
{
    string expected = "Material: StainlessSteel, CrossSection: IBeam,
Length: 4000mm, Height: 20mm, Width: 15mm, No Stress Test Performed";
    StressTestCase target = new StressTestCase();
    //Assert.AreEqual(Material.StainlessSteel, target.girderMaterial);
    //Assert.AreEqual(CrossSection.IBeam, target.crossSection);
    //Assert.AreEqual(4000, target.lengthInMm);
    //Assert.AreEqual(20, target.heightInMm);
    //Assert.AreEqual(15, target.widthInMm);
    Assert.AreEqual(expected, target.ToString());
}
```

6. Rebuild the solution and correct any errors:

   - On the **Build** menu, click **Build Solution**.

7. Run all of the tests in the solution, and then verify that all of the tests execute successfully:

   a. On the **Test** menu, point to **Run**, and then click **All Tests in Solution**.

b. Wait for the tests to run, and in the Test Results window, verify that all of the tests pass.

Exercise 2: Using Static Members to Share Data

Task 1: Open the StressTesting solution

- Open the StressTesting solution in the E:\Labfiles\Lab 7\Ex2\Starter folder. This solution contains a copy of the **StressTestCase** class with the public properties made private:

  a. In Visual Studio, on the **File** menu, point to **Open**, and then click **Project/Solution**.

  b. In the **Open Project** dialog box, move to the **E:\Labfiles\Lab 7\Ex2 \Starter** folder, click **StressTesting.sln**, and then click **Open**.

Task 2: Create a struct to hold the number of successes and failures

1. Review the task list:

   a. If the task list is not already visible, on the **View** menu, click **Task List**.

   b. If the **Task List** is displaying **User Tasks**, in the drop-down list box click **Comments**.

2. In the task list, locate the **TODO - Create the TestStatistics struct** task, and then double-click this task. This task is located in the **StressTestCase** class.

3. Delete the **TODO - Create the TestStatistics struct** comment, and then define a new public struct named **TestStatistics**, which has the following private members:

   a. An integer named **numberOfTestsPerformed**.

   b. An integer named **numberOfFailures**.

   Your code should resemble the following code example.

```
...
    public struct TestStatistics
    {
        private int numberOfTestsPerformed;
        private int numberOfFailures;
    }
}
```

4. Add a method to the **TestStatistics** struct named **IncrementTests**. The method should accept a Boolean parameter named *success*, but not return a value. Add code to the method to perform the following tasks:

   a. Increment the **numberOfTestsPerformed** member.

   b. If the *success* parameter is **false**, increment the **numberOfFailures** member.

   Your code should resemble the following code example.

```
public struct TestStatistics
{
    ...
    private int numberOfFailures;

    public void IncrementTests(bool success)
    {
        numberOfTestsPerformed++;
        if (!success)
        {
            numberOfFailures++;
        }
    }
}
```

5. Below the **IncrementTests** method, add a method named **GetNumberOfTestsPerformed**. This method should take no parameters and return an integer value. Add code to the method to return the value of the **numberOfTestsPerformed** member.

   Your code should resemble the following code example.

```
public struct TestStatistics
{
    ...
    public int GetNumberOfTestsPerformed()
    {
        return numberOfTestsPerformed;
    }
}
```

6. Below the **GetNumberOfTestsPerformed** method, add a method named **GetNumberOfFailures**. The method should take no parameters and return an integer value. Add code to the method to return the value of the **numberOfFailures** member.

   Your code should resemble the following code example.

```
public struct TestStatistics
{
    ...

    public int GetNumberOfFailures()
    {
        return numberOfFailures;
    }
}
```

7. Below the **GetNumberOfFailures** method, add an **internal** method named **ResetCounters**. The method should take no parameters and not return a value. Add code to the method to set both the **numberOfFailures** and the **numberOfTestsPerformed** members to zero.

   Your code should resemble the following code example.

```
public struct TestStatistics
{
    ...

    internal void ResetCounters()
    {
        numberOfFailures = 0;
        numberOfTestsPerformed = 0;
    }

}
```

8. Build the project and correct any errors:

   - On the **Build** menu, click **Build Solution**. Correct any errors.

Task 3: Modify the StressTestCase class to contain a TestStatistics object

1. In the task list, locate the **TODO - Add a TestStatistics field and method to the StressTestCase class** task, and then double-click this task. This task is located in the **StressTestCase** class.
2. Delete the **TODO - Add a TestStatistics field and method to the StressTestCase class** comment, and then declare a new private static member of type **TestStatistics** named **statistics**.

   Your code should resemble the following code example.

```
public class StressTestCase
{
    ...
    private TestCaseResult? testCaseResult;
    private static TestStatistics statistics;
    ...
}
```

3. Below the **statistics** member declaration, add a public static method named **GetStatistics**. The method should take no parameters, but should return a **TestStatistics** object. Add code to the method to return the value of the **statistics** member.

   Your code should resemble the following code example.

```
public class StressTestCase
{
    ...
    private static TestStatistics statistics;
    public static TestStatistics GetStatistics()
    {
        return statistics;
    }
    ...
}
```

4. Below the **GetStatistics** method, add a public static method named **ResetStatistics**. The method should take no parameters and should not return a value. Add code to the method to invoke the **ResetCounters** method on the **statistics** member.

   Your code should resemble the following code example.

```
public class StressTestCase
{
    ...
    public static TestStatistics GetStatistics()
    {
        return statistics;
    }

    public static void ResetStatistics()
    {
        statistics.ResetCounters();
    }
    ...
}
```

5. In the task list, locate the **TODO - Update the PerformStressTest method to handle statistics** task, and then double-click this task. This method is located in the **StressTestCase** class.

6. Delete the **TODO - Update the PerformStressTest method to handle statistics** comment, and in the **PerformStressTest** method, add code to invoke the **IncrementTests** method on the **statistics** member when a test either passes or fails. If the test passes, specify the value **true** as the argument to the **IncrementTests** method. If the test fails, specify the value **false** as the argument to the **IncrementTests** method.

   Your code should resemble the following code example.

```
public void PerformStressTest()
{
    ...
    if (Utility.rand.Next(10) == 9)
    {
        ...
        tcr.reasonForFailure = failureReasons[Utility.rand.Next(5)];

        statistics.IncrementTests(false);
    }
    else
    {
        tcr.result = TestResult.Pass;

        statistics.IncrementTests(true);
    }
    ...
}
```

Task 4: Display the statistics in the user interface

1. In the task list, locate the **TODO - Update the UI to display statistics** task, and then double-click this task. This task is located in the **MainWindow** class, at the end of the **doTests_Click** method.

2. At the end of the **doTests_Click** method, delete the comments and add code to perform the following tasks:

   a. Create a new **TestStatistics** object named **statistics**. Initialize the object with the value that is returned by calling the **StressTestCase**.**GetStatistics** method.

b. In the **statisticsLabel1** label, display the message "Number of tests: <*tests*>, Failures: <*failures*>", where *tests* is the number of tests that were executed, and *failures* is the number of tests that failed.

Hint: Set the **Content** property of a **Label** control to display a message in that control.

c. Invoke the **IncrementTests** method on the **statistics** object, and pass **true** as a parameter.

d. Invoke the static **GetStatistics** method on the **StressTestCase** object, and store the result in the statistics variable.

e. In the **statisticsLabel2** label, display the message "Number of tests: <*tests*>, Failures: <*failures*>", where *tests* is the number of tests that were executed, and *failures* is the number of tests that failed.

Note: This demonstrates the principle of passing or returning by value. When the code first calls the **GetStatistics** method, a copy of the value is returned from the **StressTestCase** object. Therefore, when the code calls the **IncrementTests** method, the update is performed on the copied value and not the original value. When the **GetStatistics** method is called for the second time, another copy of the original value is retrieved; therefore, both labels will display the same value.

Your code should resemble the following code example.

```
private void doTests_Click(object sender, RoutedEventArgs e)
{
    ...
    TestStatistics statistics = StressTestCase.GetStatistics();
    statisticsLabel1.Content = string.Format(
        "Number of tests: {0}, Failures: {1}",
        statistics.GetNumberOfTestsPerformed(),
        statistics.GetNumberOfFailures());
    statistics.IncrementTests(true);
    statistics = StressTestCase.GetStatistics();
    statisticsLabel2.Content = string.Format(
        "Number of tests: {0}, Failures: {1}",
        statistics.GetNumberOfTestsPerformed(),
        statistics.GetNumberOfFailures());
}
```

Task 5: Test the solution

1. Build the solution and correct any errors:
   - On the **Build** menu, click **Build Solution**. Correct any errors.
2. Run the application:
   - On the **Debug** menu, click **Start Without Debugging**.
3. In the MainWindow window, click **Run Stress Tests**, and then examine the statistics labels, which should both display the same values.
4. Close the MainWindow window, and then return to Visual Studio.

Task 6: Examine and run unit tests for the TestStatistics class

1. In the task list, locate the **TODO - Examine and run unit tests** task, and then double-click this task. This task is located in the StressTestClass_TestStatisticsTest file.
2. Examine the **GetNumberOfFailuresTest** method.

   This method creates a new **TestStatistics** object named **target** and then invokes the **IncrementTests** method twice, passing **false** as the parameter. The method then retrieves the number of failures from the **TestStatistics** object and uses an **Assert** statement to verify that the value is correct.
3. Examine the **GetNumberOfTestsPerformed** method.

   This method creates a new **TestStatistics** object named **target** and then invokes the **IncrementTests** method three times. The method then retrieves the number of tests that was performed from the **TestStatistics** object and uses an **Assert** statement to verify that the value is correct.
4. Examine the **IncrementTestsTest** method.

   This method creates a **TestStatistics** object named **target** and then invokes the **IncrementTests** method on this object four times. The method then retrieves the number of tests that were performed from the **target** object and uses an **Assert** statement to verify that the value is correct.
5. Run all of the tests in the solution, and then verify that all of the tests execute successfully:

   a. On the **Test** menu, point to **Run**, and then click **All Tests in Solution**.

   b. Wait for the tests to run, and in the Test Results window, verify that all of the tests pass.

Exercise 3: Implementing an Extension Method

Task 1: Open the StressTesting solution

- Open the StressTesting solution in the E:\Labfiles\Lab 7\Ex3\Starter folder. This solution contains a copy of the solution from the previous exercise:

  a. In Visual Studio, on the **File** menu, point to **Open**, and then click **Project/Solution**.

  b. In the **Open Project** dialog box, move to the **E:\Labfiles\Lab 7\Ex3 \Starter** folder, click **StressTesting.sln**, and then click **Open**.

Task 2: Define a new extension method

1. In the StressTest project, add a new public static class named **Extensions**, in a file named Extensions.cs:

   a. In Solution Explorer, right-click the **StressTest** project, point to **Add**, and then click **Class**.

   b. In the **Add New Item - StressTest** dialog box, in the **Name** box, type **Extensions** and then click **Add**.

   c. Modify the **Extensions** class definition. This class should be a **public static** class.

2. In the **Extensions** class, add a new public static extension method named **ToBinaryString**. The method should take a 64-bit integer parameter named *i* and return a string value.

Hint: To indicate that a method is an extension method, prefix the parameter with the **this** keyword.

Hint: You can use **long** as an alias for the **System.Int64** type.

Your code should resemble the following code example.

```
public static class Extensions
{
    public static string ToBinaryString(this long i)
    {
    }
}
```

3. In the **ToBinaryString** method, add code to create a string that holds the binary representation of the 64-bit integer value that is passed in the i integer, and return this string.

   Your code should resemble the following code example.

```
public static string ToBinaryString(this System.Int64 i)
{
    long remainder = 0;
    StringBuilder binary = new StringBuilder("");

    while (i > 0)
    {
        remainder = i % 2;
        i = i / 2;
        binary.Insert(0, remainder);
    }
    return binary.ToString();
}
```

Task 3: Modify the TestCaseResult struct to include a long field

1. Review the task list:
   a. If the task list is not already visible, on the **View** menu, click **Task List**.
   b. If the **Task List** is displaying **User Tasks**, in the drop-down list box click **Comments**.
2. In the task list, locate the **TODO - Modify the TestCaseResult struct** task, and then double-click this task. This task is located in the **TestCaseResult** struct.
3. In the **TestCaseResult** struct, delete the comment and add a public field of type **long** named **failureData**.

   Your code should resemble the following code example.

```
public struct TestCaseResult
{
    ...
    public string reasonForFailure;

    public long failureData;
}
```

Task 4: Modify the PerformStressTest method

1. In the task list, locate the **TODO - Update the PerformStressTest method** task, and then double-click this task. This task is located in the **StressTestCase** class, in the **PerformStressTest** method.
2. In the **PerformStressTest** method, delete the **TODO - Update the PerformStressTest method** comment, and then add code to update the **failureData** member of the **TestCaseResult** object with a random number to simulate the data that is retrieved from the stress-testing equipment.

Hint: Use the **Rand** member of the **Utility** static class to generate a random number. This method contains a method called **Next** that returns a random number in a specified range. Pass the value **int.MaxValue** as the parameter to the **Next** method to generate a random number between 0 and this value. The value **int.MaxValue** field specifies the maximum value that the integer type supports.

Your code should resemble the following code example.

```
public void PerformStressTest()
{
    ...
    tcr.reasonForFailure = failureReasons[Utility.rand.Next(5)];

    tcr.failureData = Utility.Rand.Next(int.MaxValue);

    statistics.IncrementTests(false);
    ...
}
```

Task 5: Display the failure data

1. In the task list, locate the **TODO - Update the UI to display the binary string** task, and then double-click this task. This task is located in the **MainWindow** class, in the **doTests_Click** method.
2. Modify the **doTests_Click** method to append the binary data that is contained in the **failureData** member to the failure information that is displayed in the user interface; append a space character followed by the result of the **ToBinaryString** method call to the end of the string that is added to the **resultList.Items** collection.

   Your code should resemble the following code example.

```
private void doTests_Click(object sender, RoutedEventArgs e)
{
    ...
    {
        ...
        if (stc.GetStressTestResult().HasValue)
        {
            tcr = (TestCaseResult)stc.GetStressTestResult().Value;

            // Modified in Exercise 3 to use extension method.
            resultList.Items.Add(tcr.result.ToString()
                + " " + tcr.reasonForFailure
                + " " + tcr.failureData.ToBinaryString());
        }
    }
    ...
}
```

Task 6: Test the solution

1. Build the solution and correct any errors:
   - On the **Build** menu, click **Build Solution**. Correct any errors.
2. Run the application:
   - On the **Debug** menu, click **Start Without Debugging**.
3. In the MainWindow window, click **Run Stress Tests**, and then verify that when an error occurs, binary data is displayed after the reason for the failure.
4. Close the MainWindow window, and then return to Visual Studio.

Task 7: Examine and run unit tests

1. In the task list, locate the **TODO - Review and run unit tests** task, and then double-click this task. This task is located in the **ExtensionsTest** class.
2. Examine the **ToBinaryStringTest** method.

   This method creates a long variable, i, with the value 8 and then creates a string variable, expected, with the value "1000". The method then invokes the **ToBinaryString** extension method on the long variable i and stores the result in a string named **actual**. The method then uses an **Assert** statement to verify that the expected and **actual** values are the same. The method then updates the long variable i with the value 10266 and the expected variable with the binary representation "10100000011010". Next, it directly calls the **ToBinaryString** method, passes the long variable i as a parameter, and stores the result of the method call in the actual variable. The method uses a second **Assert** statement to verify that the expected and actual values are the same.

3. Run all of the tests in the solution, and then verify that all of the tests execute successfully:

   a. On the **Test** menu, point to **Run**, and then click **All Tests in Solution**.

   b. Wait for all of the tests to run, and in the Test Results window, verify that all of the tests pass.