**Exercise 1: Building a Simple File Editor**

**Task 1: Open the SimpleEditor project**

**Task 2: Display a dialog box to accept a file name from the user**

|  |  |
| --- | --- |
| 1. | Display the MainWindow.xaml window.  The MainWindow window implements a very simple text editor. The main part of the window contains atext box that a user can use to display and edit text. The **Open** button enables the user to open a file, and the**Save** button enables the user to save the changes to the text back to a file. You will add the code thatimplements the logic for these two buttons. |
| 2. | Review the task list. |
| 3. | Locate the task **TODO - Implement a method to get the file name**. Double-click this task.  This task is located in the MainWindow.xaml.cs class file. |
| 4. | Delete the comment, and then define a new private method named **GetFileName** that accepts no parametersand returns a string value that holds the file name that the user specified. |
| 5. | In the method body, declare a new string member named **fname**, and then initialize it with the **String.Empty**value. |
| 6. | At the end of the collection of **using** statements at the top of the file, add a statement to bring the**Microsoft.Win32** namespace into scope. |
| 7. | In the **GetFileName** method, after the statement that declares the fname variable, add code to the method toperform the following actions:   |  |  | | --- | --- | | a. | Create a new instance of the**OpenFileDialog** dialog box, named **openFileDlg**. | | b. | Set the **InitialDirectory** property of **openFileDlg** to point to the E:\Labfiles\Lab 5\Ex1\Starter folder.  **Note**: When including file paths in code, you should prefix the string with the @ symbol. Thissymbol instructs the C# compiler to treat any '\' characters as literals rather than escapecharacters. | | c. | Set the **DefaultExt** property of **openFileDlg**to ".txt"; | | d. | Set the **Filter** property of **openFileDlg**to **"Text Documents (.txt)|\*.txt"**. | |
| 8. | Add code to perform the following tasks:   |  |  | | --- | --- | | a. | Call the **ShowDialog** method of **openFileDlg**, and then save the result.  **Note**: The value that **ShowDialog** returns is a nullable Boolean value, so save the result in anullable Boolean variable. | | b. | If the result is **true**, assign the value of the **FileName** property of **openFileDlg** to the fname variable. | |
| 9. | At the end of the method, return the value in the fname variable. |

**Task 3: Implement a new class to read and write text to a file**

|  |  |
| --- | --- |
| 1. | Add a new class named **TextFileOperations** to the FileEditor project.  You will use this class to wrap some common file operations. This scheme enables you to change the way inwhich files are read from or written to without affecting the rest of the application. |
| 2. | At the top of the class file, add a statement to bring the **System.IO** namespace into scope. |
| 3. | In the **TextFileOperations** class, add a public static method named **ReadTextFileContents**. The methodshould accept a string parameter named *fileName*, and return a string object. |
| 4. | In the **ReadTextFileContents** method, add code to return the entire contents of the text file whose path isspecified in the *fileName* parameter.  **Hint**: Use the static **ReadAllText** method of the **File** class. |
| 5. | Below the **ReadTextFileContents** method, add a public static method named **WriteTextFileContents**. Themethod should not return a value type, and should accept the following parameters:   |  |  | | --- | --- | | a. | A string parameter named *fileName*. | | b. | A string parameter named *text*. | |
| 6. | In the **WriteTextFileContents** method, add code to write the text that is contained in the *text* parameter tothe file that is specified in the *fileName* parameter.  **Hint**: Use the static **WriteAllText** method of the **File** class. |
| 7. | Build the solution and correct any errors. |

**Task 4: Update the MainWindow event handlers to consume the TextFileOperations class**

|  |  |
| --- | --- |
| 1. | In the task list, locate the task **TODO - Update the OpenButton\_Click method**. Double-click this task.  This task is located in the **OpenButton**\_**Click** method of the **MainWindow** class. |
| 2. | Remove the comment, and then add code to perform the following tasks:   |  |  | | --- | --- | | a. | Invoke the **GetFileName** method. Store the result of the method in the **fileName**member. | | b. | If **fileName** is not an empty string, call the static **ReadTextFileContents** method of the**TextFileOperations** class, and then pass *fileName* as the parameter. Store the result in the **Text**property of the editor **TextBox** control in the WPF window. | |
| 3. | In the task list, locate the task **TODO - Update the SaveButton\_Click method**. Double-click this task.  This task is located in the **SaveButton**\_**Click** method of the **MainWindow** class. |
| 4. | In the **SaveButton**\_**Click** method, remove the comment, and then add code to perform the following tasks:   |  |  | | --- | --- | | a. | Check that the **fileName** member is not an empty string. | | b. | If **fileName** is not an empty string, call the static **WriteTextFileContents** method of the**TextFileOperations** class. Pass **fileName** and the **Text**property of the editor **TextBox** control as theparameters. | |
| 5. | Build the solution and correct any errors. |
| 6. | Start the application without debugging. |
| 7. | In the MainWindow window, click **Open**. |
| 8. | In the **Open** dialog box, move to the **E:\Labfiles\Lab 5\Ex1\Starter** folder, click **Commands.txt**, and thenclick **Open**. |
| 9. | In the MainWindow window, verify that the text in the following code example is displayed in the editor**TextBox** control. |

Move x, 10

Move y, 20

If x < y Add x, y

If x > y & x < 20 Sub x, y

Store 30

This is the text from the Commands.txt file.

|  |  |
| --- | --- |
| 10. | Change the **Store 30** line to **Save 50**, and then click **Save**. |
| 11. | Close the MainWindow window. |
| 12. | Using Windows Explorer, move to the **E:\Labfiles\Lab 5\Ex1\Starter** folder. |
| 13. | Open the Commands.txt file by using Notepad. |
| 14. | In Notepad, verify that the last line of the file contains the text **Save 50**. |
| 15. | Close Notepad and return to Visual Studio. |

**Task 5: Implement test cases**

|  |  |
| --- | --- |
| 1. | In the task list, locate the task **TODO - Complete Unit Tests**. Double-click this task.  This task is located in the **TextFileOperationsTest** class. |
| 2. | Remove the comment. |
| 3. | Examine the **ReadTextFileContentsTest1** method, and then uncomment the commented line.  This method creates three strings:   |  |  | | --- | --- | | a. | The **fileName** string contains the path of a prewritten file that contains specific content. | | b. | The **expected** string contains the contents of the prewritten file, including formatting and escapecharacters. | | c. | The **actual** string is initialized by calling the **ReadTextFileContents** method that you justimplemented. |   The test method then uses an **Assert** statement to verify that the **expected** and **actual** strings are the same. |
| 4. | Examine the **WriteTextFileContentsTest1** method, and then uncomment the commented line.  This method creates two strings:   |  |  | | --- | --- | | a. | The **fileName** string contains the path of a nonexistent file, which the method will create when run. | | b. | The **text** string contains some text that the method will write to the file. |   The method calls the **WriteTextFileContents** method, passing the **fileName** and **text** strings as parameters.This creates the file at the specified location, and writes to the file. The method then creates a further string,**expected**, by calling the **File**.**ReadAllText** method and reading the text from the written file. The methodthen checks that the text string and the expected string are the same, before deleting the file that was createdduring the test. |
| 5. | Run all tests in the solution, and verify that all tests execute correctly. |

**Exercise 2: Making the Editor XML Aware**

**The applications that control a robotic device read the instructions from the file and then encode them as anXML document before passing them to the instruction execution module on the device. For example,imagine that a text file contains the instructions in the following code example.**

Move x, 10

Move y, 20

If x < y Add x, y

If x > y Sub x, y

Store 30

The control applications will wrap them in a pair of XML tags, as the following code example shows.

<ControlApplication>

<Instructions Code = "

Move x, 10

Move y, 20

If x < y Add x, y

If x > y Sub x, y

Store 30"

/>

</ControlApplication>

However, some of the data in these instructions can contain characters such as ">" and "<" that might bemisinterpreted as XML tags rather than data.

In this exercise, you will modify the WPF application to look for data that contains XML tags in the text file as itis read in and encode this data as XML escape sequences before displaying it. For example, the "<" character willbe replaced with "&gt;", the ">" symbol will be replaced with "&lt;", and so on. The WPF application will use afile stream to read the data.

The main tasks for this exercise are as follows:

|  |  |
| --- | --- |
| 1. | Open the starter project. |
| 2. | Add a new method to filter XML characters to the **TextFileOperations** class. |
| 3. | Update the user interface to invoke the new method. |
| 4. | Implement test cases. |

**Task 1: Open the starter project**

|  |  |
| --- | --- |
| • | Open the SimpleEditor solution in the E:\Labfiles\Lab 5\Ex2\Starter folder.  This project is a completed version of the SimpleEditor project from Exercise 1. |

**Task 2: Add a new method to filter XML characters to the TextFileOperations class**

|  |  |
| --- | --- |
| 1. | Review the task list. |
| 2. | In the task list, locate the **TODO - Implement a new method in the TextFileOperations class** task.Double-click this task.  This task is located in the **TextFileOperations** class. |
| 3. | Remove the comment, and then add a new public static method named **ReadAndFilterTextFileContents**.The method should accept a string parameter named *fileName*, and return a string. |
| 4. | In the **ReadAndFilterTextFileContents** method, add the following local variables:   |  |  | | --- | --- | | a. | A **StringBuilder** object named **fileContents**, initialized to a new instance of the **StringBuilder** class. | | b. | An integer variable called charCode. | |
| 5. | Add a statement that instantiates a **StreamReader** object, named **fileReader**, by using the *fileName*parameter. |
| 6. | Add a **while** statement that reads each character in the **StreamReader** object until the end of the file isreached.  **Hint**: Use the **Read** method of the **StreamReader** class to read the next character from a stream.This method returns –1 if there is no more data. |
| 7. | In the **while** block, add a **switch** statement that evaluates the charCode variable.  In the **switch** statement, add **case** statements for each of the characters in the following table. In eachstatement, append the **fileContent** **StringBuilder** object with the alternative representation shown in thetable.   | **charCode** | **Standard representation** | **Alternative representation** | | --- | --- | --- | | 34 | " (straight quotation mark) | &quot; | | 38 | & (ampersand) | &amp; | | 39 | ' (apostrophe) | &apos; | | 60 | < (less than) | &lt; | | 62 | > (greater than) | &gt; | |
| 8. | Add a default **case** statement that appends the actual character read from the stream to the **fileContentStringBuilder** object.  **Note**: The **Read** method returns the value read from the file as an integer and stores it in thecharCode variable. You must cast this variable to a character before you append it to the end of the**StringBuilder** object. |
| 9. | At the end of the method, return the contents of the **fileContent StringBuilder** object as a string. |
| 10. | Build the solution and correct any errors. |

**Task 3: Update the user interface to invoke the new method**

|  |  |
| --- | --- |
| 1. | In the task list, locate the **TODO - Update the UI to use the new method** task. Double-click this task.  This task is located in the **OpenButton\_Click** method of the **MainWindow**.**xaml**.**cs** class. |
| 2. | Delete the comment, and then modify the line of code that calls the**TextFileOperations.ReadTextFileContents** method to call the**TextFileOperations.ReadAndFilterTextFileContents** method instead. Pass the **fileName** field as theparameter, and then save the result in the **Text** property of the editor **TextBox** control. |
| 3. | Build the solution and correct any errors. |
| 4. | Start the application without debugging. |
| 5. | In the MainWindow window, click **Open**. |
| 6. | In the **Open** dialog box, move to the **E:\Labfiles\Lab 5\Ex2\Starter** folder, click **Commands.txt**, and thenclick **Open**. |
| 7. | In the MainWindow window, verify that the text in the following code example is displayed in the editor**TextBox** control. |

Move x, 10

Move y, 20

If x &lt; y Add x, y

If x &gt; y &amp; x &lt; 20 Sub x, y

Store 30

This is the text from the Commands.txt file. Notice that the **<**, **>**, and **&** characters have been replaced with the text**&lt;**, **&gt;**, and **&amp;**.

|  |  |
| --- | --- |
| 8. | Close the MainWindow window and return to Visual Studio. |

**Task 4: Implement test cases**

|  |  |
| --- | --- |
| 1. | In the task list, locate the **TODO - Complete Unit Tests** task. Double-click this task.  This task is located in the **TextFileOperationsTest** class. |
| 2. | Examine the **ReadAndFilterTextFileContentsTest** method, and then uncomment the commented line.  This method creates three strings:   |  |  | | --- | --- | | a. | The **filename** string contains the path of a prewritten file that contains specific content. | | b. | The **expected** string contains the contents of the prewritten file, including formatting and escapecharacters. | | c. | The **actual** string is initialized by calling the**ReadAndFilterTextFileContents** method that you justimplemented. |   The test method then uses an **Assert** statement to verify that the **expected** and **actual** strings are the same.  This method is complete, and requires no further work. |
| 3. | Run all tests in the solution, and verify that all tests execute correctly. |