GRAND CANYON UNIVERSITY™

CST-150 Activity 4 Guide

Contents

Part 1

Creating Methods and Writing to a File

Overview

Software developers need an underlying base or foundation how to modularize their code by writing methods. Upon completion of this assignment, students will be able to define a method using C# syntax, distinguish between void and value-returning methods, pass data to methods, and invoke methods.

Execution

Execute this assignment according to the following guidelines:

1. Activity 4 is a continuation of Activity 3 Part 1.
    a. Open Activity 3 in Visual Studio.
    b. Open Visual Studio and select "Open a project or solution" as shown in Figure 1.
    c. Select "CST-150 Activity 3" and open the project by selection the solution file "CST-150 Activity 3.sln." The .sln is the solution file.

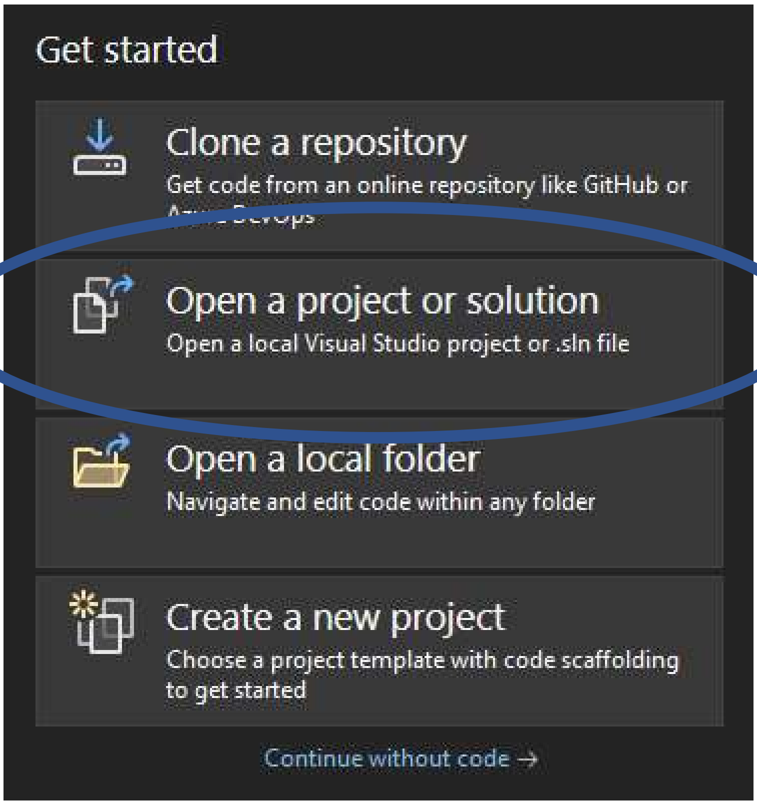
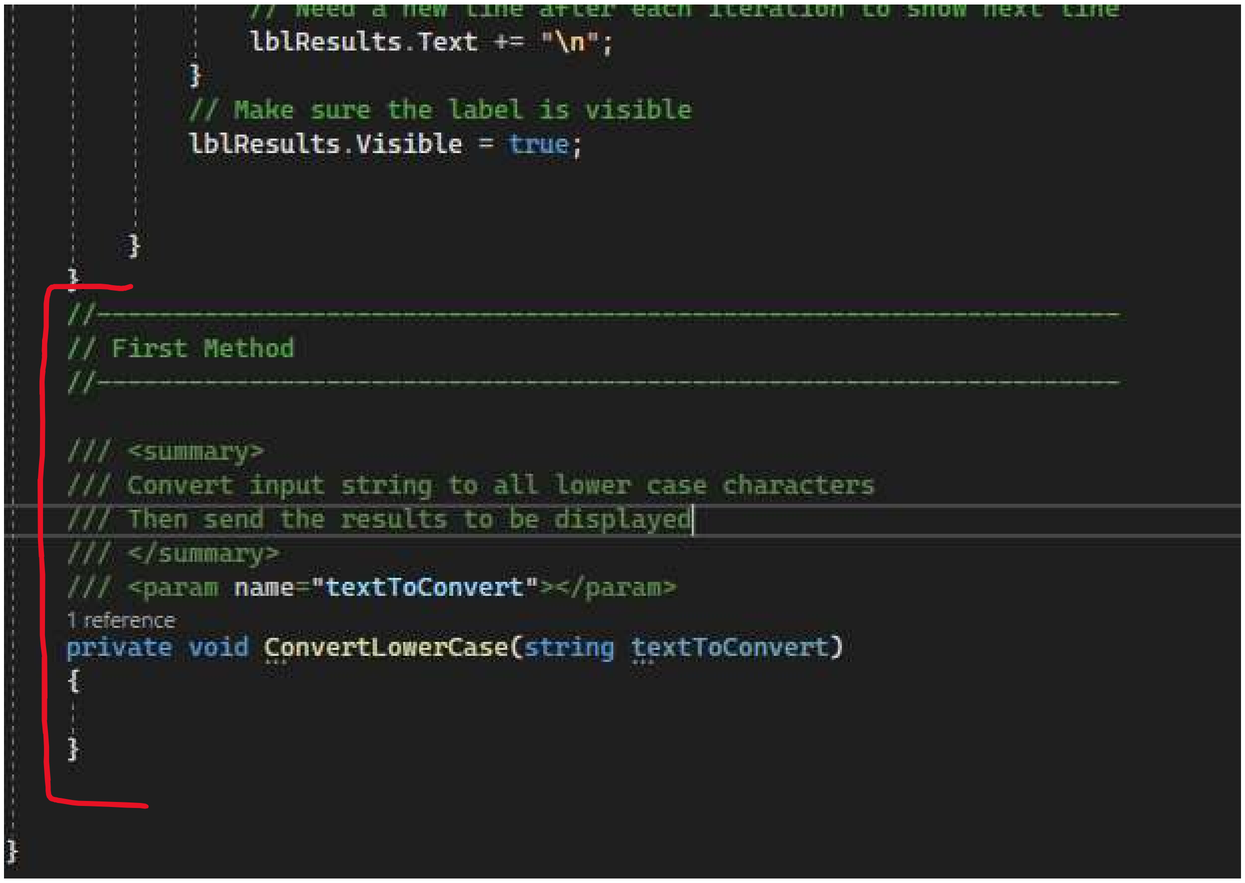

Get started
Clone a repository
Get code from an online repository like GitHub or Azure DevOps
Open a project or solution
Open a local Visual Studio project or .sln file
Open a local folder
Navigate and edit code within any folder
Create a new project
Choose a project template with code scaffolding to get started
Continue without code →

Figure 1: Open an existing solution.

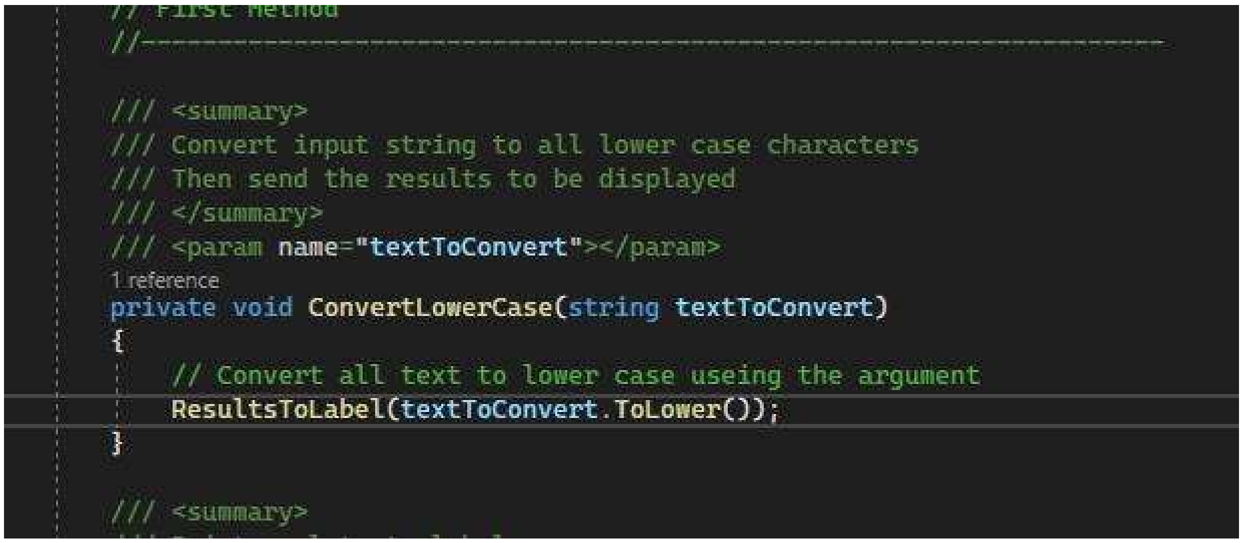
2. Create First Method
    a. Convert all the words to lowercase.
    b. Access Modifier as private since this method will only be accessed within this method as shown in Figure 1.
    c. We will not be returning anything so return type is void.
    d. Method name is "ConvertLowerCase" using PascalCasing.
    e. The parameter data that we need to receive must be the string that we read in from the file (datatype name) syntax.



```
            // Need a new line after each iteration to show next line
            lblResults.Text += "\n";
        }
        // Make sure the label is visible
        lblResults.Visible = true;


    }
}
//-------------------------------------------------------------------
// First Method
//-------------------------------------------------------------------

/// <summary>
/// Convert input string to all lower case characters
/// Then send the results to be displayed
/// </summary>
/// <param name="textToConvert"></param>
1 reference
private void ConvertLowerCase(string textToConvert)
{

}


}
```

Figure 2: First Method – Convert to Lower Case

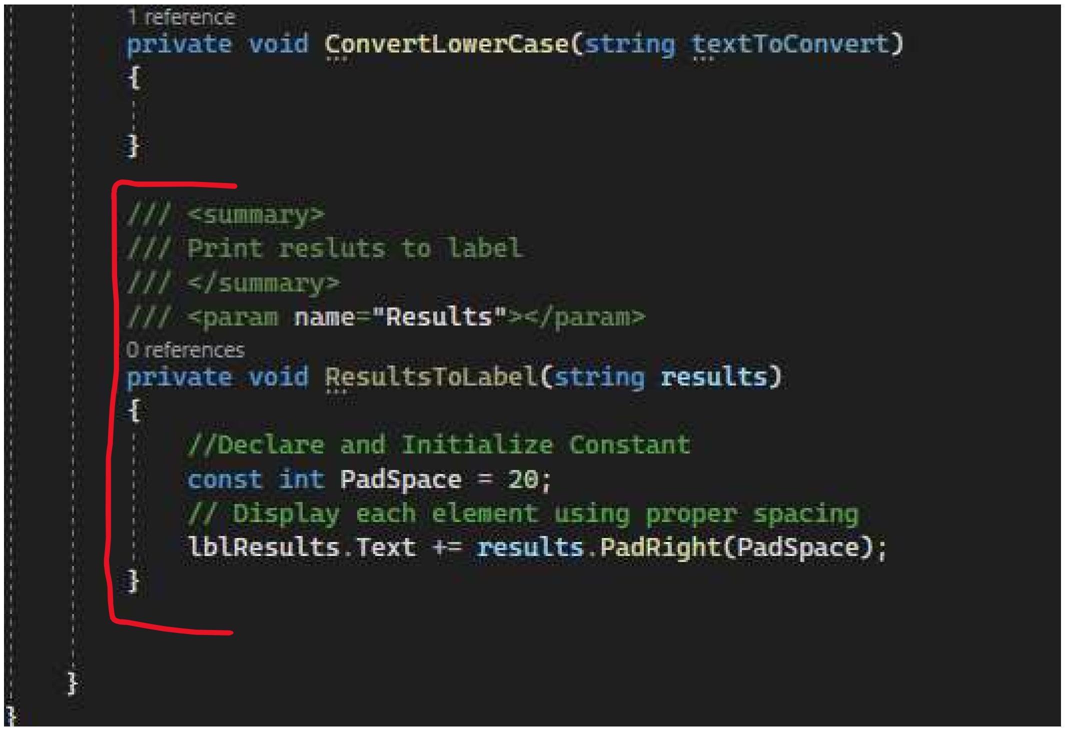
f. Place a block of code that will print out the text in lowercase. This statement calls the next method that we will be creating as shown in Figure 3.



```
    // First Method
    //-------------------------------------------------------------------

    /// <summary>
    /// Convert input string to all lower case characters
    /// Then send the results to be displayed
    /// </summary>
    /// <param name="textToConvert"></param>
    1 reference
    private void ConvertLowerCase(string textToConvert)
    {
        // Convert all text to lower case useing the argument
        ResultsToLabel(textToConvert.ToLower());
    }

    /// <summary>
```

Figure 3: Make all text lower case.

3. Create Second Method.
    a. Print results to the label.
    b. Access Modifier as private since this method will only be accessed within this method as shown in Figure 4.
    c. We will not be returning anything so return type is void.
    d. Method name is "ResultsToLabel" using PascalCasing.
    e. The parameter data that we need to receive must be the string that is shown in the label (datatype name) syntax.

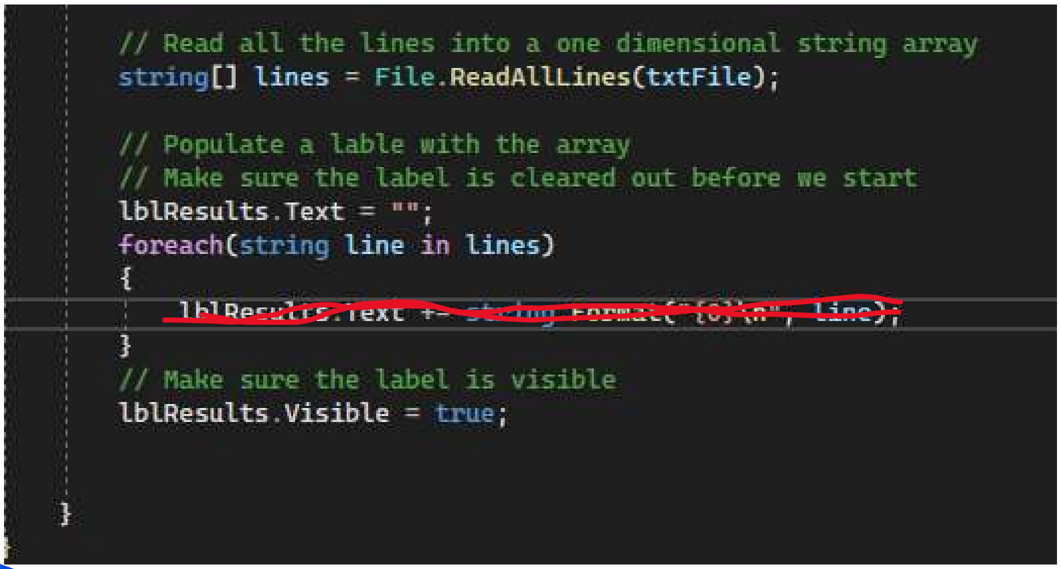


```
        1 reference
        private void ConvertLowerCase(string textToConvert)
        {

        }

        /// <summary>
        /// Print resluts to label
        /// </summary>
        /// <param name="Results"></param>
        0 references
        private void ResultsToLabel(string results)
        {
            //Declare and Initialize Constant
            const int PadSpace = 20;
            // Display each element using proper spacing
            lblResults.Text += results.PadRight(PadSpace);
        }

    }
}
```

Figure 4: Method 2 - Print Results to Label.

?

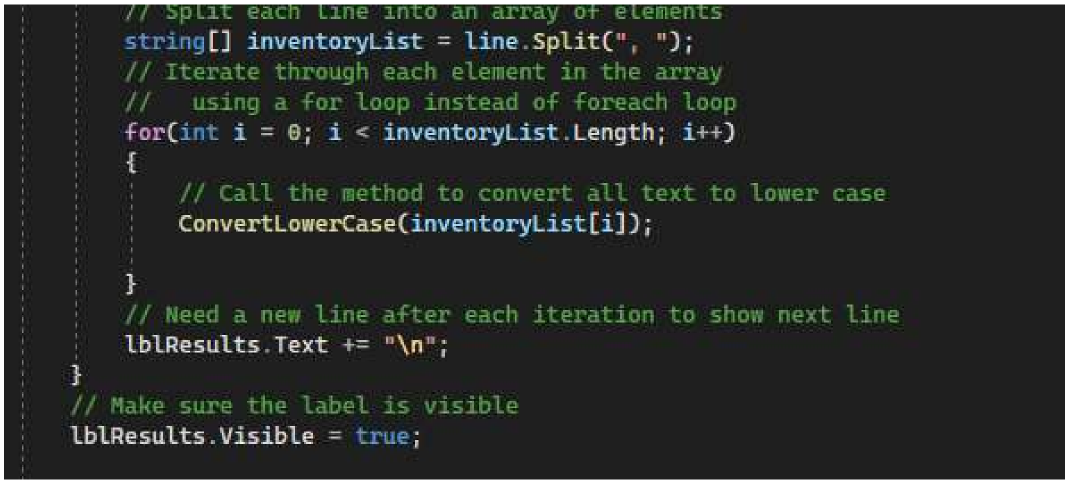
f. Then, remove the lblResults.Text printing to label line of code from the method "BtnReadFileClickEvent" as is shown in Figure 5.



```
        // Read all the lines into a one dimensional string array
        string[] lines = File.ReadAllLines(txtFile);

        // Populate a lable with the array
        // Make sure the label is cleared out before we start
        lblResults.Text = "";
        foreach(string line in lines)
        {
            ~~lblResults.Text += string.Format("{0}\n", line);~~
        }
        // Make sure the label is visible
        lblResults.Visible = true;

    }
}
```

Figure 5: Remove Print to Label.

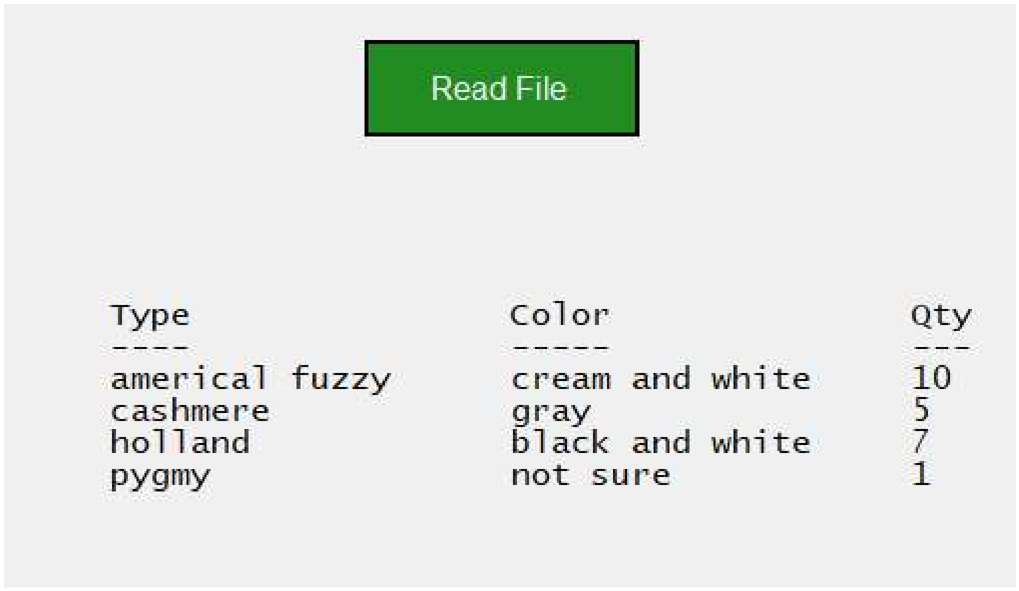
g. In BtnReadFileClickEvent, add in a call to convert the text to lowercase as shown in Figure 6. The lowercase method will call this method to print the results.



```
        // Split each line into an array of elements
        string[] inventoryList = line.Split(", ");
        // Iterate through each element in the array
        //   using a for loop instead of foreach loop
        for(int i = 0; i < inventoryList.Length; i++)
        {
            // Call the method to convert all text to lower case
            ConvertLowerCase(inventoryList[i]);

        }
        // Need a new line after each iteration to show next line
        lblResults.Text += "\n";
    }
    // Make sure the label is visible
    lblResults.Visible = true;
```

Figure 6: Call Method to Convert to Lower-Case.

h. Figure 7 shows example results being shown in the label.



```
Type                Color               Qty
----                -----               ---
americal fuzzy      cream and white     10
cashmere            gray                5
holland             black and white     7
pygmy               not sure            1
```

Figure 7: Sample Results.

4. Update the Form.
   a. In the main form, add a label and comboBox as shown in Figure 8.
   b. Label name: lblSelectRow
   c. Label text: Select Row
   d. comboBox name: cmbSelectRow
   e. We will dynamically populate the comboBox in the code behind.

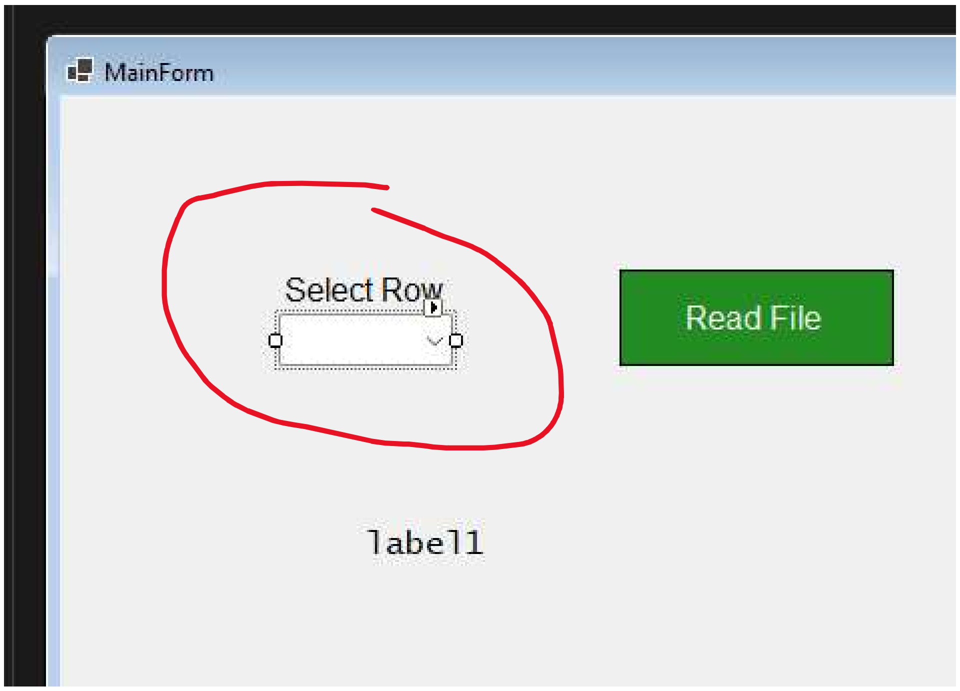
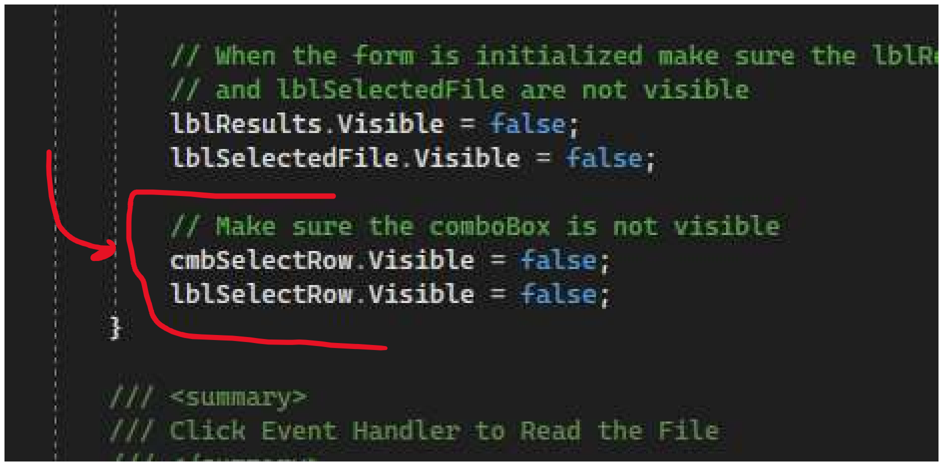


Figure 8: Add Label and comboBox controls.

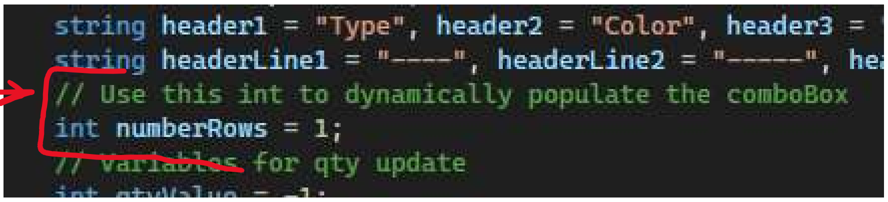
5. Manage the comboBox in the Code Behind.
    a. Back to the code behind "FrmMain.cs."
    b. Be sure the label and comboBox we just added are not visible when the program is first run. In the Class Constructor, add the following lines as shown in Figure 9.



```
        // When the form is initialized make sure the lblR
        // and lblSelectedFile are not visible
        lblResults.Visible = false;
        lblSelectedFile.Visible = false;

        // Make sure the comboBox is not visible
        cmbSelectRow.Visible = false;
        lblSelectRow.Visible = false;
    }

    /// <summary>
    /// Click Event Handler to Read the File
```

Figure 9: Hide the label and comboBox.

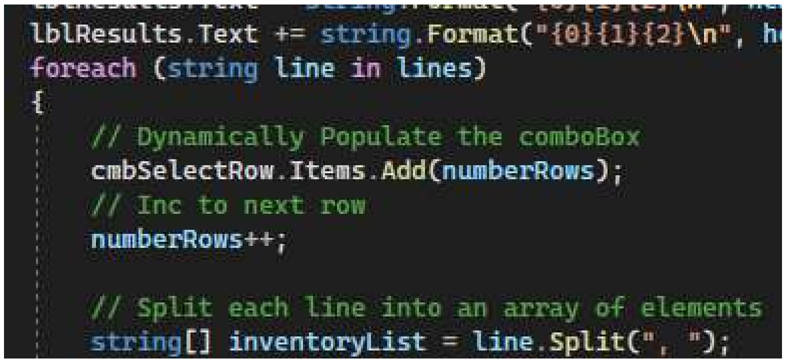
    c. Add the pointer to the variables as shown in Figure 10. Start this at 1 to point to row 1 in the inventory.



```
string header1 = "Type", header2 = "Color", header3 = 
string headerLine1 = "----", headerLine2 = "-----", he
// Use this int to dynamically populate the comboBox
int numberRows = 1;
// variables for qty update
int qtyValue = -1;
```

Figure 10: Add Variable Pointer.

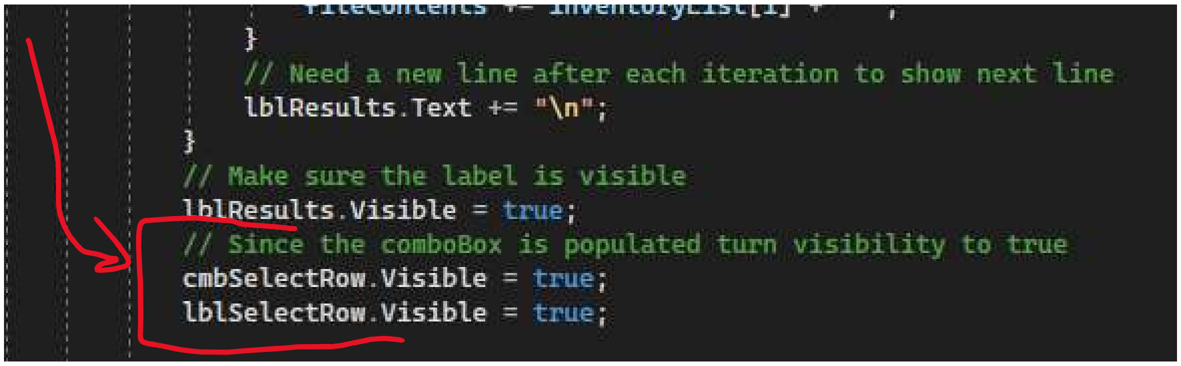
d. In the foreach loop, dynamically populate the comboBox as shown in Figure 11.



```
lblResults.Text += string.Format("{0}{1}{2}\n", h
foreach (string line in lines)
{
    // Dynamically Populate the comboBox
    cmbSelectRow.Items.Add(numberRows);
    // Inc to next row
    numberRows++;

    // Split each line into an array of elements
    string[] inventoryList = line.Split(", ");
```

Figure 11: Dynamically Populate comboBox.

e. Outside the foreach loop, be sure the label and comboBox are visible as shown in Figure 12.



```
        }
        // Need a new line after each iteration to show next line
        lblResults.Text += "\n";
    }
    // Make sure the label is visible
    lblResults.Visible = true;
    // Since the comboBox is populated turn visibility to true
    cmbSelectRow.Visible = true;
    lblSelectRow.Visible = true;
```

Figure 12: Be sure the label and comboBox are visible.

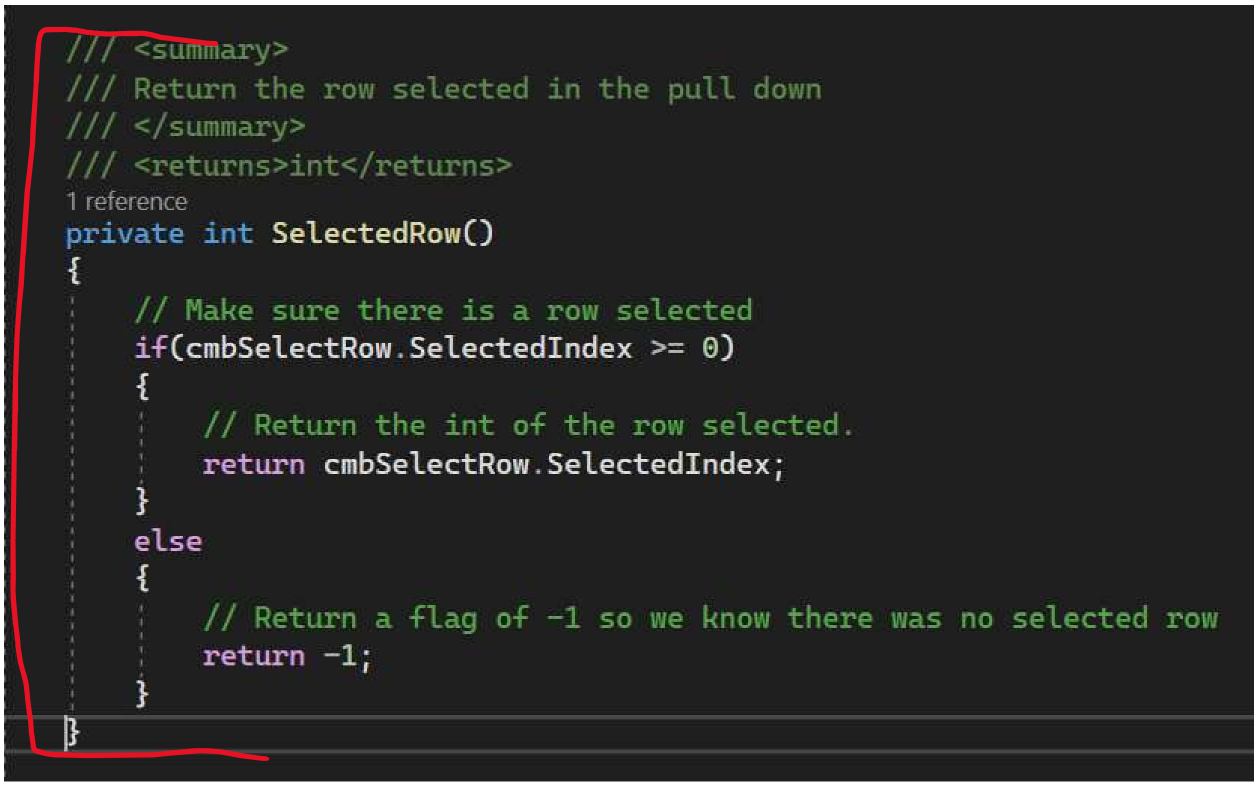
6. Create Third Method

a. Method is to determine which row was selected via the comboBox as shown in Figure 13.

b. This method is returning the row selected as an int.

c. If no row is selected, return a -1; otherwise, return the row selected.

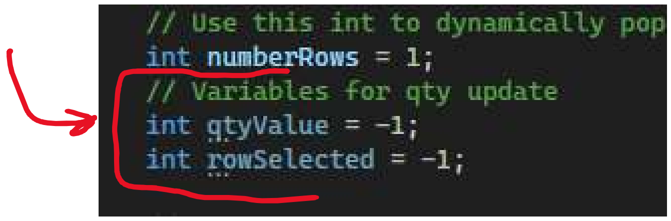
d. Row will start with index 0.



```
/// <summary>
/// Return the row selected in the pull down
/// </summary>
/// <returns>int</returns>
1 reference
private int SelectedRow()
{
    // Make sure there is a row selected
    if(cmbSelectRow.SelectedIndex >= 0)
    {
        // Return the int of the row selected.
        return cmbSelectRow.SelectedIndex;
    }
    else
    {
        // Return a flag of -1 so we know there was no selected row
        return -1;
    }
}
```

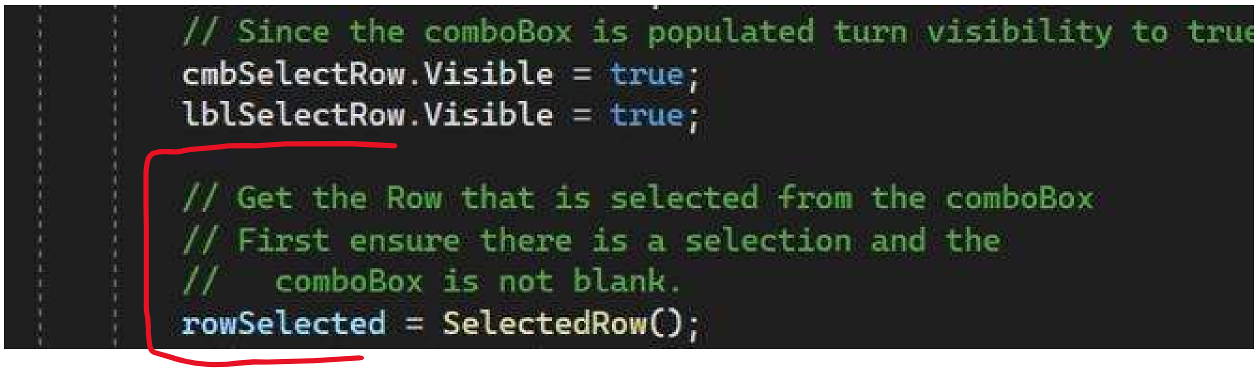
Figure 13: Method 3

7. Back up to the "BtnReadFileClickEvent" handler.
   a. Test the selected Row method.
   b. First declare and initialize a variable as shown in Figure 14 to be used for the row selected.
   c. Invoke the method as shown in Figure 15.
   d. Verify the proper row as selected by using debug tools in Visual Studio.



```
// Use this int to dynamically pop
int numberRows = 1;
// Variables for qty update
int qtyValue = -1;
int rowSelected = -1;
```

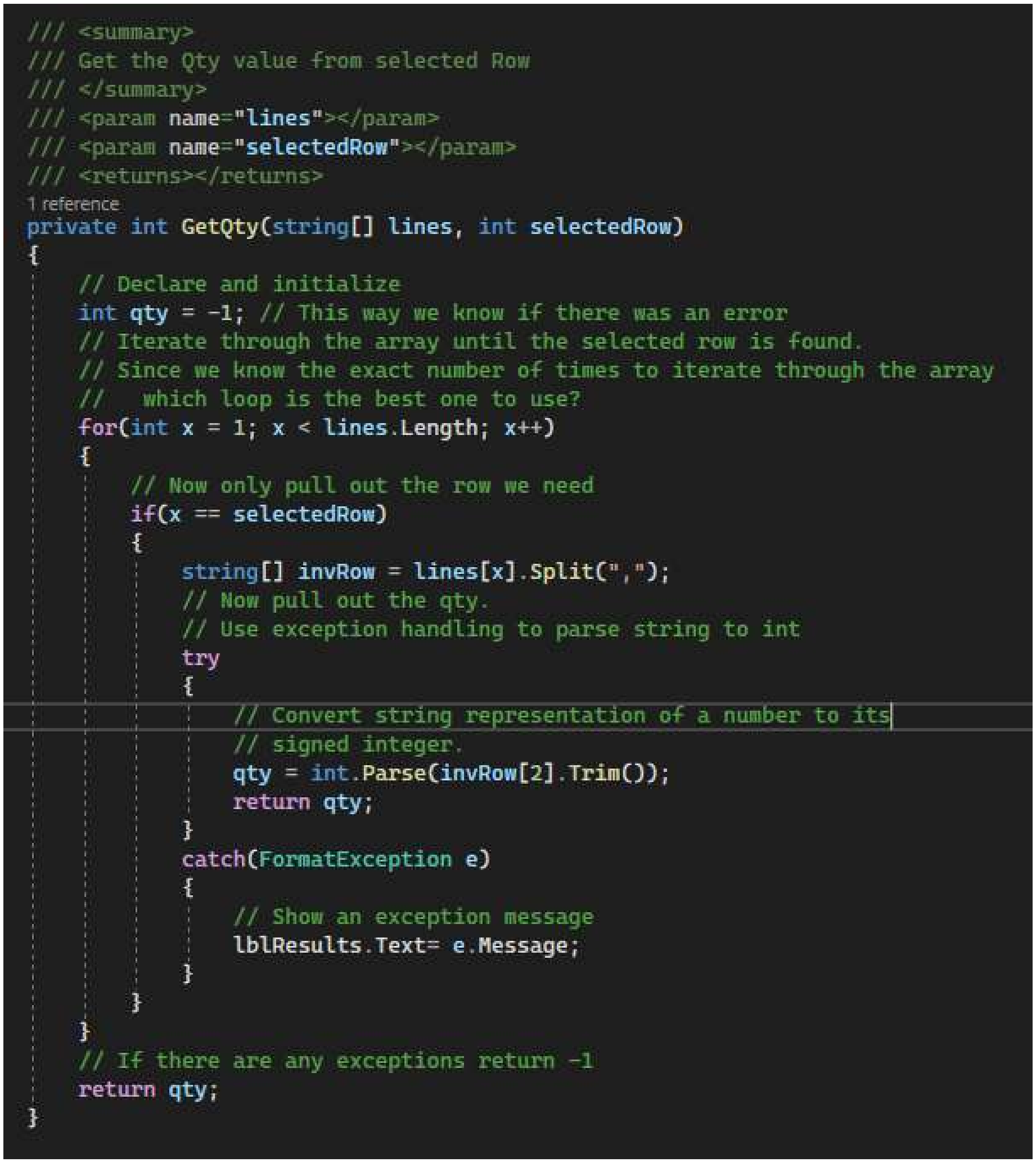
Figure 14: Add variable to track row selected.



```
// Since the comboBox is populated turn visibility to true
cmbSelectRow.Visible = true;
lblSelectRow.Visible = true;

// Get the Row that is selected from the comboBox
// First ensure there is a selection and the
//   comboBox is not blank.
rowSelected = SelectedRow();
```

Figure 15: Invoke the method.

8. Create Fourth Method
    e. Now, we have the selected row.
    f. Read the qty from selected row and parse to int using a try catch to manage any exceptions in the Parse.
    g. Follow the comments to fully understand the functionality of this code as shown in Figure 16.

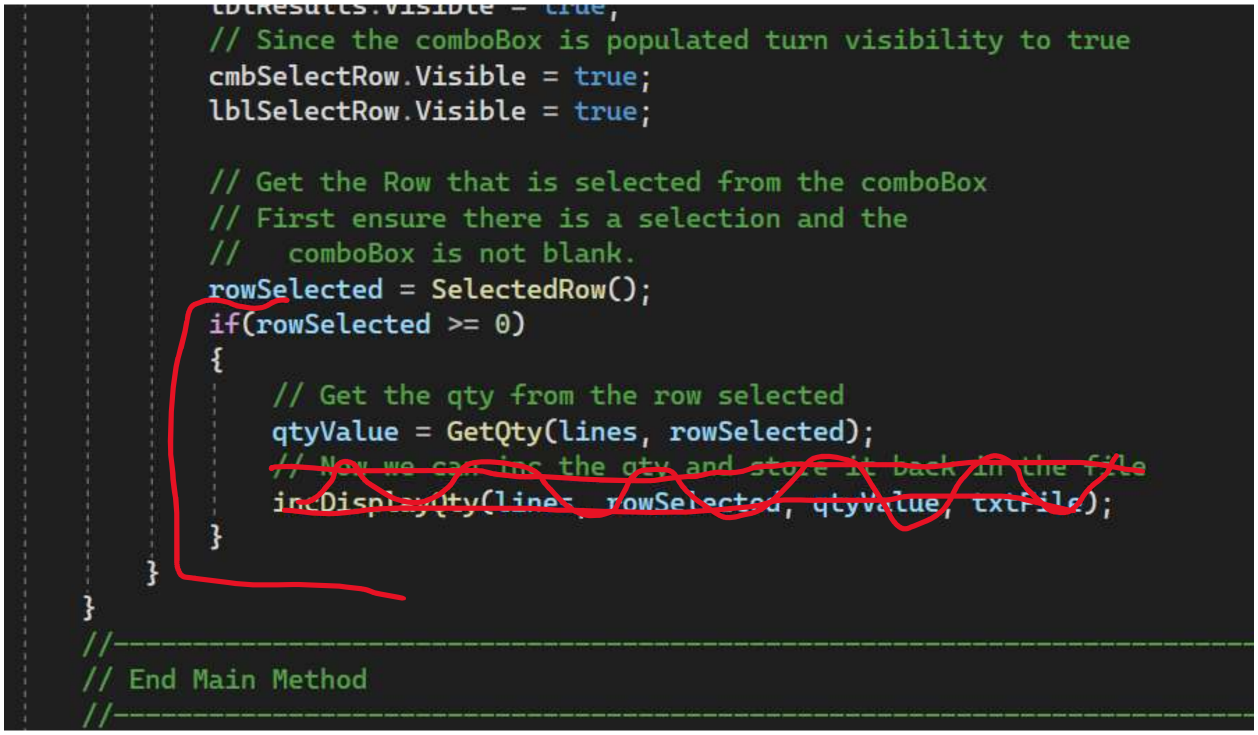


```
/// <summary>
/// Get the Qty value from selected Row
/// </summary>
/// <param name="lines"></param>
/// <param name="selectedRow"></param>
/// <returns></returns>
1 reference
private int GetQty(string[] lines, int selectedRow)
{
    // Declare and initialize
    int qty = -1; // This way we know if there was an error
    // Iterate through the array until the selected row is found.
    // Since we know the exact number of times to iterate through the array
    //   which loop is the best one to use?
    for(int x = 1; x < lines.Length; x++)
    {
        // Now only pull out the row we need
        if(x == selectedRow)
        {
            string[] invRow = lines[x].Split(",");
            // Now pull out the qty.
            // Use exception handling to parse string to int
            try
            {
                // Convert string representation of a number to its
                // signed integer.
                qty = int.Parse(invRow[2].Trim());
                return qty;
            }
            catch(FormatException e)
            {
                // Show an exception message
                lblResults.Text= e.Message;
            }
        }
    }
    // If there are any exceptions return -1
    return qty;
}
```

Figure 16: Get the Qty.

h. Back up to the "BtnReadFileClickEvent" handler.

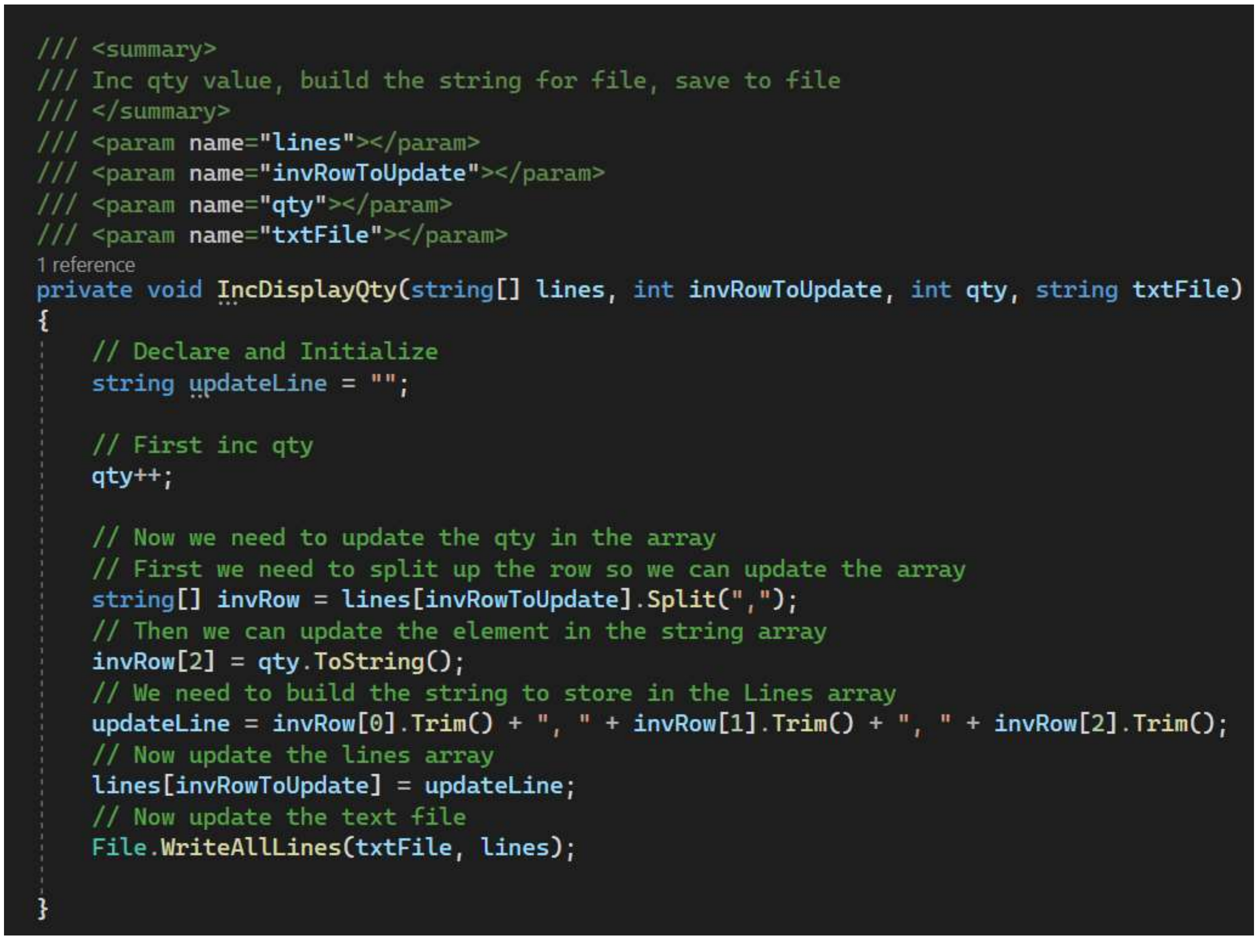
i. Inside the if statement, invoke this new method as shown in Figure 17 and test the functionality works correctly using debug tools.



```
        lblResults.Visible = true;
        // Since the comboBox is populated turn visibility to true
        cmbSelectRow.Visible = true;
        lblSelectRow.Visible = true;

        // Get the Row that is selected from the comboBox
        // First ensure there is a selection and the
        //   comboBox is not blank.
        rowSelected = SelectedRow();
        if(rowSelected >= 0)
        {
            // Get the qty from the row selected
            qtyValue = GetQty(lines, rowSelected);
            ~~// Now we can inc the qty and store it back in the file~~
            ~~incDisplayQty(lines, rowSelected, qtyValue, txtFile);~~
        }
    }
}
//------------------------------------------------------------------
// End Main Method
//------------------------------------------------------------------
```

Figure 17: Invoke the GetQty method.

9. Create Fifth Method
    a. Now that we have qty value as an int.
    b. We need to increment the qty by one and then write the updated data back to the text file as shown in Figure 18.



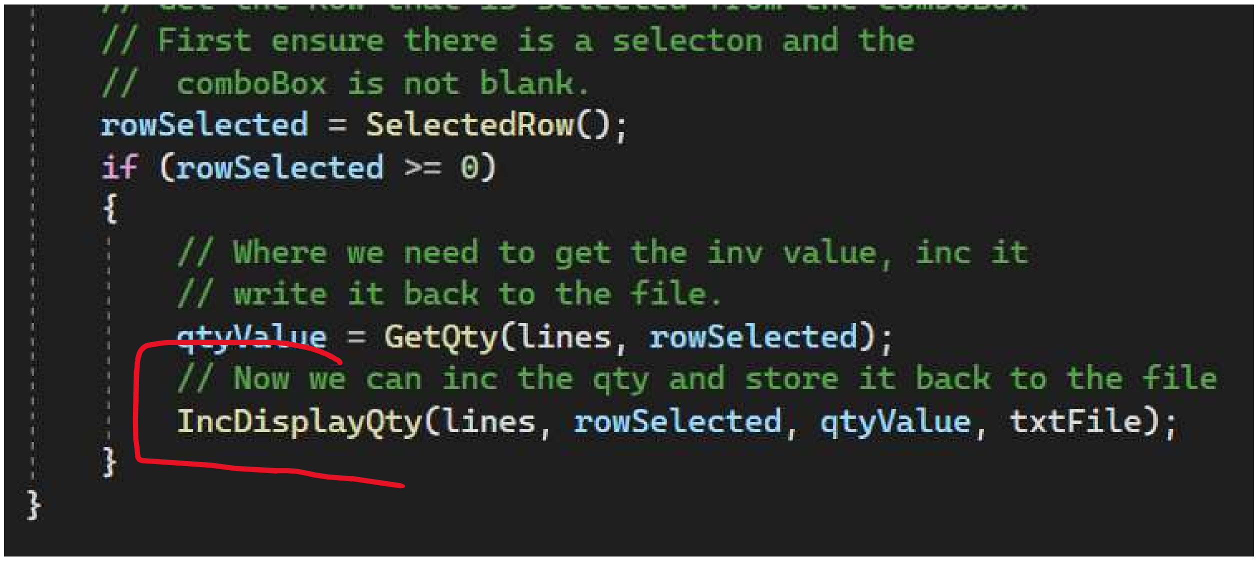
```
/// <summary>
/// Inc qty value, build the string for file, save to file
/// </summary>
/// <param name="lines"></param>
/// <param name="invRowToUpdate"></param>
/// <param name="qty"></param>
/// <param name="txtFile"></param>
1 reference
private void IncDisplayQty(string[] lines, int invRowToUpdate, int qty, string txtFile)
{
    // Declare and Initialize
    string updateLine = "";

    // First inc qty
    qty++;

    // Now we need to update the qty in the array
    // First we need to split up the row so we can update the array
    string[] invRow = lines[invRowToUpdate].Split(",");
    // Then we can update the element in the string array
    invRow[2] = qty.ToString();
    // We need to build the string to store in the Lines array
    updateLine = invRow[0].Trim() + ", " + invRow[1].Trim() + ", " + invRow[2].Trim();
    // Now update the lines array
    lines[invRowToUpdate] = updateLine;
    // Now update the text file
    File.WriteAllLines(txtFile, lines);

}
```

Figure 18: Inc Qty and Save File

c. Back up to the "BtnReadFileClickEvent" handler.

d. Inside the if statement, invoke this new method as shown in Figure 19 and test the functionality works correctly using debug tools.

e. Run the app and view the text file to ensure the proper inventory item was inc by one.



```
    // First ensure there is a selecton and the
    //  comboBox is not blank.
    rowSelected = SelectedRow();
    if (rowSelected >= 0)
    {
        // Where we need to get the inv value, inc it
        // write it back to the file.
        qtyValue = GetQty(lines, rowSelected);
        // Now we can inc the qty and store it back to the file
        IncDisplayQty(lines, rowSelected, qtyValue, txtFile);
    }
}
```

Figure 19: Invoke the IncDisplayQty method.

10. Submit the Activity as described in the digital classroom.
    a. Be sure to use the text file that is provided by your instructor before taking screenshots.

Part 2

Methods with Return Data Types and Parameters

Overview

Write a Windows Form Application that uses a button event handler to invoke methods. The event handler should only invoke the methods and have minimal business or display logic. Use a label to display the results. All methods must include the proper C# syntax while keeping the main method clean.

Execution

Execute this assignment according to the following guidelines:

1. Start a new Visual Studio Project.
   a. Start Visual Studio and select "Create a new project" as is shown in Figure 20.

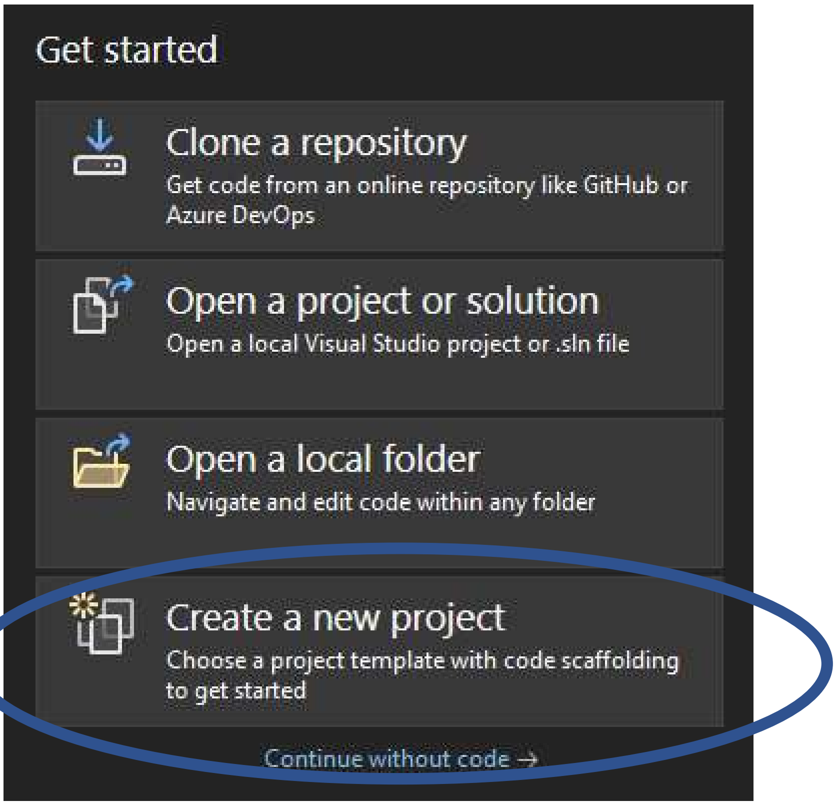
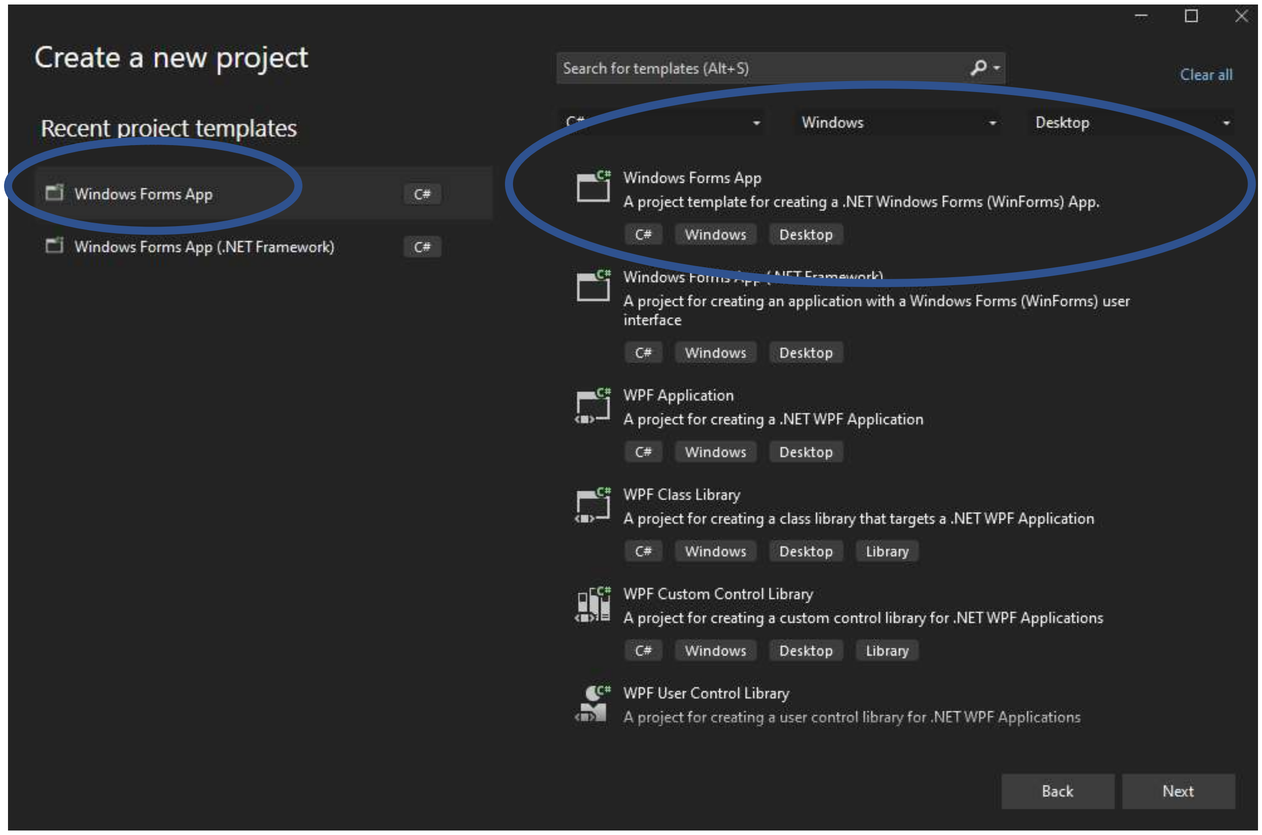

Get started
Clone a repository
Get code from an online repository like GitHub or Azure DevOps
Open a project or solution
Open a local Visual Studio project or .sln file
Open a local folder
Navigate and edit code within any folder
Create a new project
Choose a project template with code scaffolding to get started
Continue without code →

Figure 20: Select "Create a new project."

   b. Select the Project Template "WindowsFormsApp" as shown in Figure 21. Be sure to select "Windows Forms App." DO NOT select the one with (.NET Framework).


Create a new project

Recent project templates

Windows Forms App C#

Windows Forms App (.NET Framework) C#

Search for templates (Alt+S)

Clear all

C# Windows Desktop

Windows Forms App
A project template for creating a .NET Windows Forms (WinForms) App.
C# Windows Desktop

Windows Forms App (.NET Framework)
A project for creating an application with a Windows Forms (WinForms) user interface
C# Windows Desktop

WPF Application
A project for creating a .NET WPF Application
C# Windows Desktop

WPF Class Library
A project for creating a class library that targets a .NET WPF Application
C# Windows Desktop Library

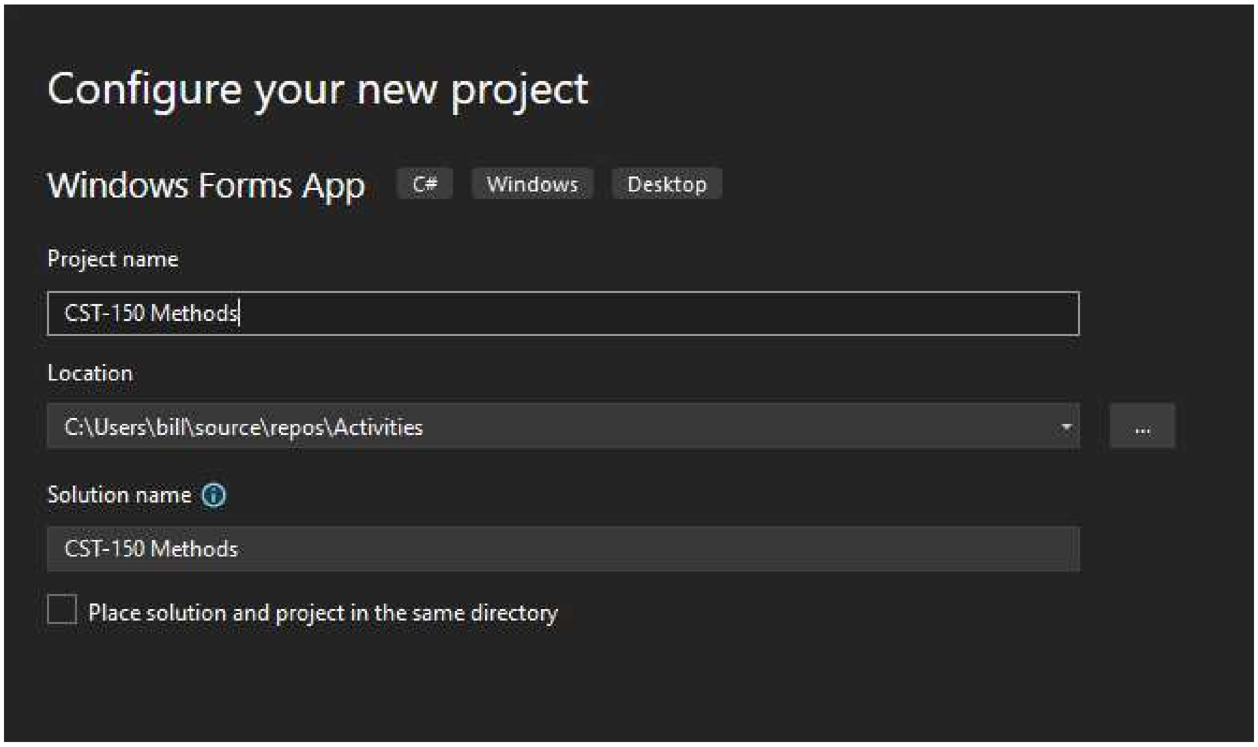
WPF Custom Control Library
A project for creating a custom control library for .NET WPF Applications
C# Windows Desktop Library

WPF User Control Library
A project for creating a user control library for .NET WPF Applications

Back Next

Figure 21: Select the Project Template.

c. Enter the Project name: "CST-150 Methods" as shown in Figure 22 followed by the "Next" button. Be sure the check box is not checked for "Place solution and project in the same directory."


Configure your new project

Windows Forms App C# Windows Desktop

Project name: CST-150 Methods

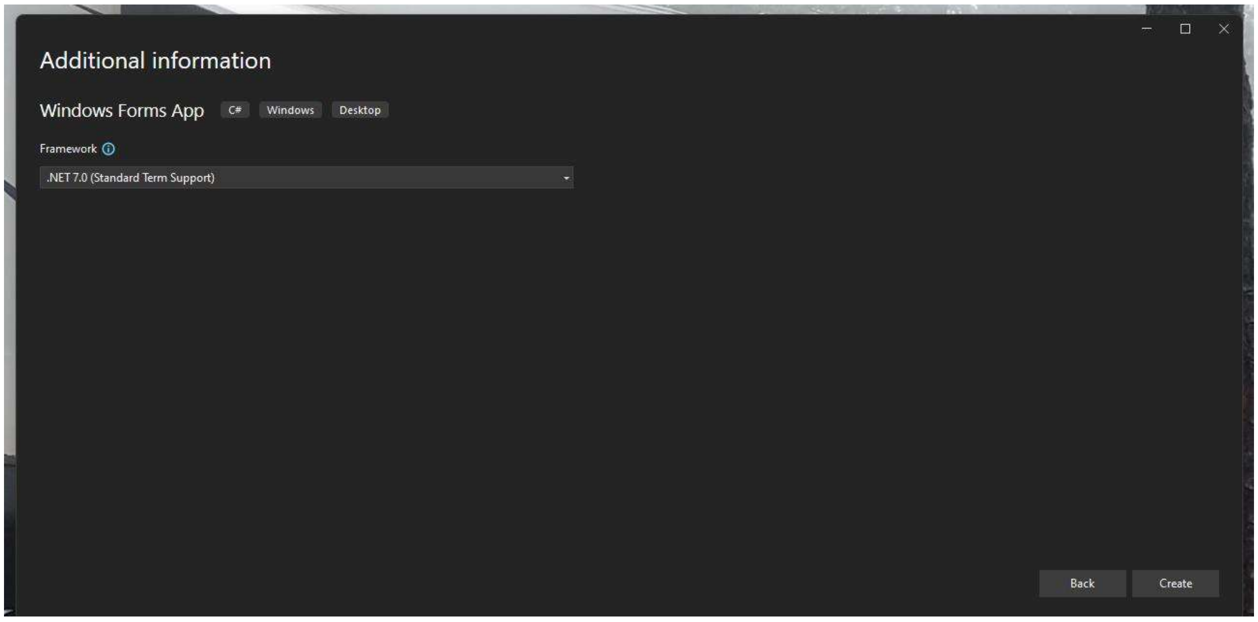
Location: C:\Users\bill\source\repos\Activities

Solution name: CST-150 Methods

Place solution and project in the same directory

Figure 22: Enter Project Name.

d. Select the Framework as is shown in Figure 23 followed by the "Create" button. These in class applications have all been tested using .NET 7.0. (The instructor may have the class us a different .NET version.)

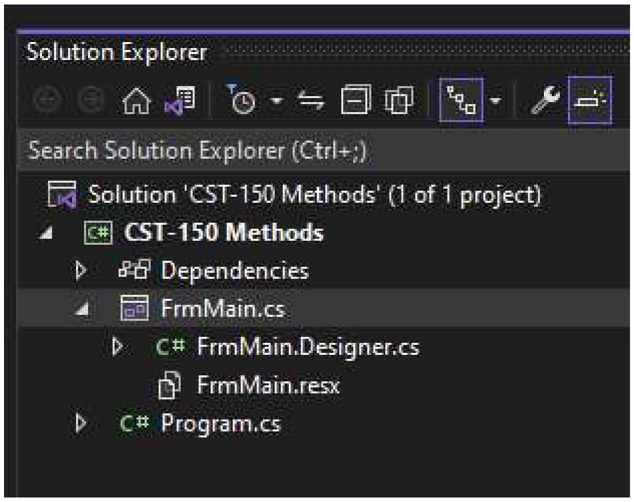

Additional information

Windows Forms App C# Windows Desktop

Framework: .NET 7.0 (Standard Term Support)

Figure 23: Select Framework.

2. Rename the form to FrmMain.cs as is shown in Figure 24.



Solution Explorer

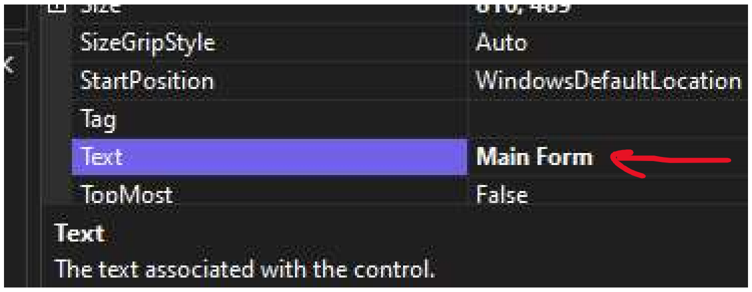
Search Solution Explorer (Ctrl+;)

```
Solution 'CST-150 Methods' (1 of 1 project)
  CST-150 Methods
    Dependencies
    FrmMain.cs
      FrmMain.Designer.cs
      FrmMain.resx
    Program.cs
```

Figure 24: Rename Form

3. Verify the Application is Working Correctly.
   a. Follow the steps outlined in Activity 3.

4. Configure the FrmMain
   a. Change the form Text property to "Main Form" as is shown in Figure 25 and Figure 26.



SizeGripStyle	Auto
StartPosition	WindowsDefaultLocation
Tag	
Text	Main Form
TopMost	False

Text
The text associated with the control.

Figure 25: Configure the Form Text Property

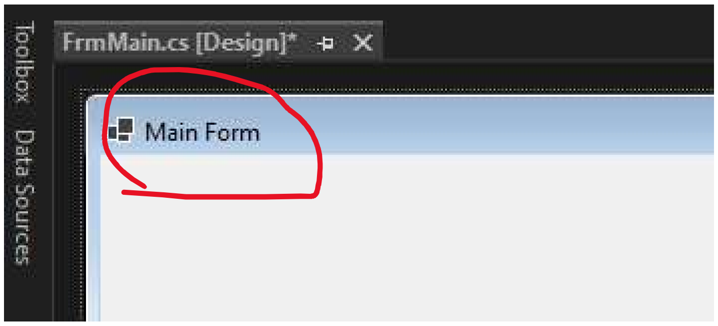
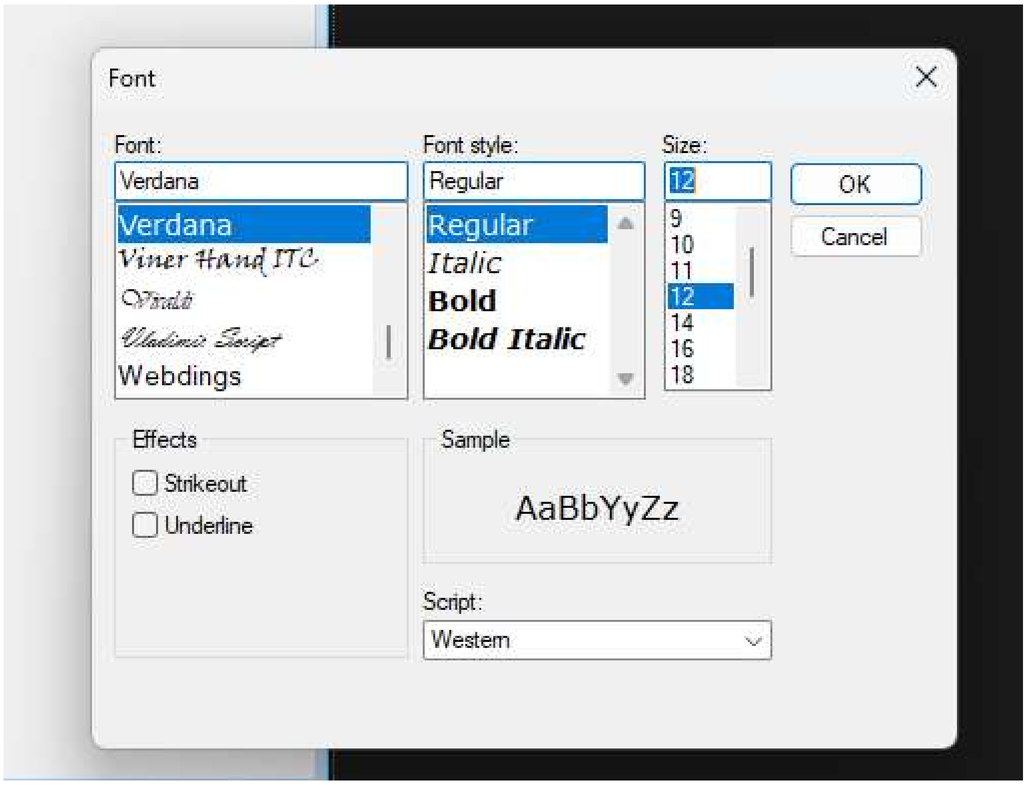


Figure 26: Main Form

b. Change the font on the FrmMain as shown in Figure 27 in the properties and then select ok.


Font

Font: Verdana

Font style: Regular

Size: 12

Script: Western

Effects: ☐ Strikeout ☐ Underline

Sample: AaBbYyZz

Figure 27: Configure Font on Form

c. Add a button and label to the form. Name the button "btnMain" and change the text to "Execute Methods." Auto size property should be true for the button. Then name the label "lblResults" as is shown in Figure 28.

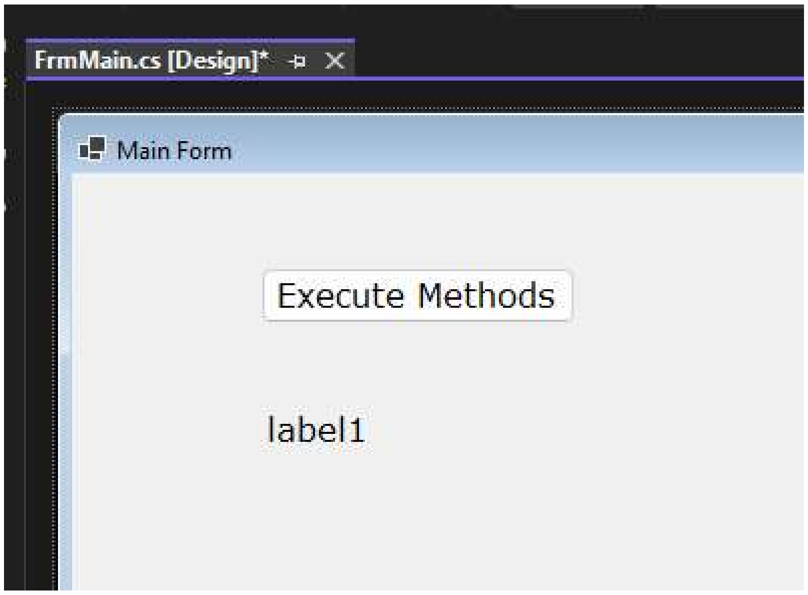
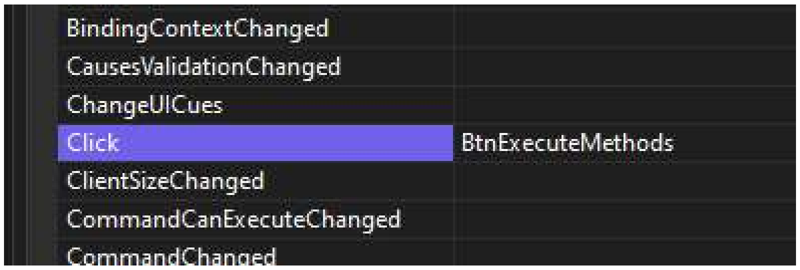


Figure 28: FrmMain

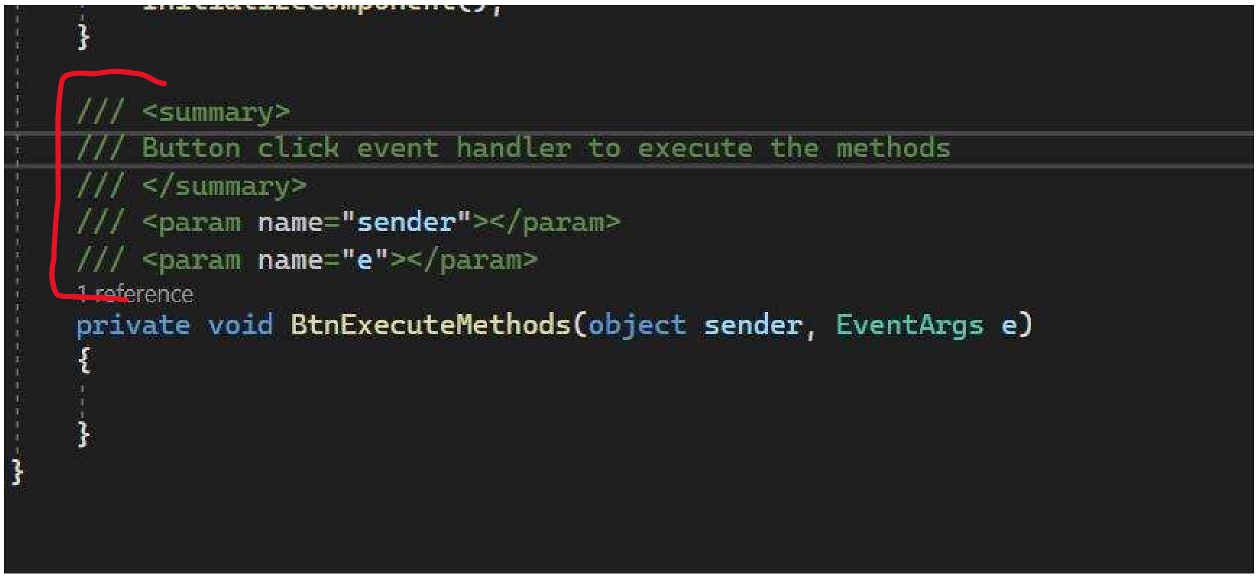
d. Select the button and create a Click Event Handler named "BtnExecuteMethods" as shown in Figure 29.



BindingContextChanged	
CausesValidationChanged	
ChangeUICues	
Click	BtnExecuteMethods
ClientSizeChanged	
CommandCanExecuteChanged	
CommandChanged	

Figure 29: Click Event Handler

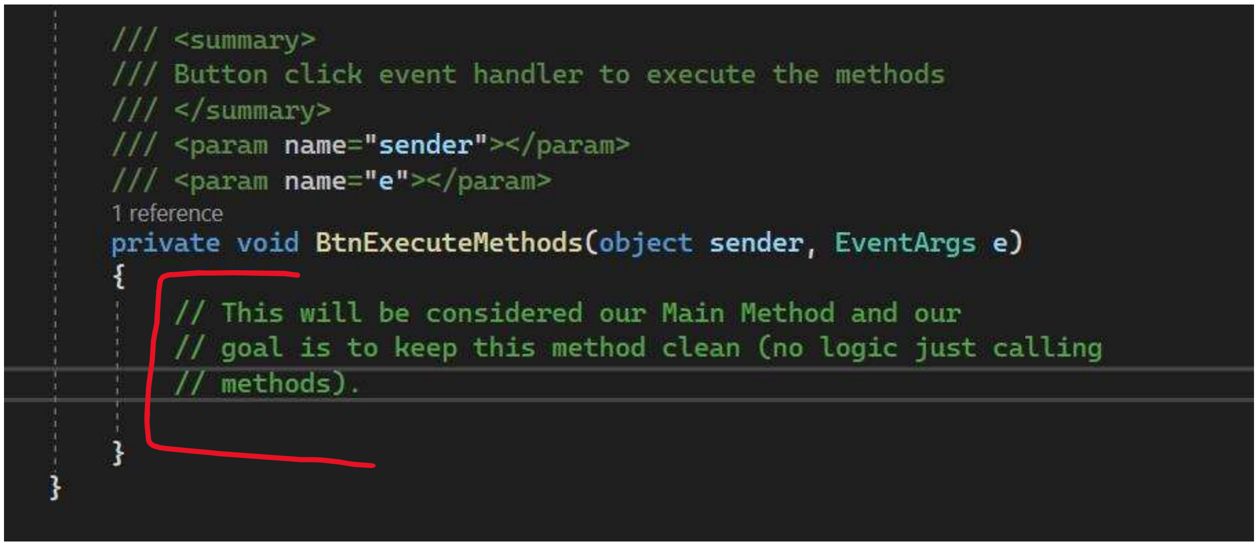
e. In the method, add the method level comments for the event handler as shown in Figure 30.



```
    }

    /// <summary>
    /// Button click event handler to execute the methods
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    1 reference
    private void BtnExecuteMethods(object sender, EventArgs e)
    {

    }
}
```

Figure 30: Add the summary to the Event Handler

5. Configure the FrmMain.cs code behind.
    a. This is our goal for the app as shown in Figure 31.

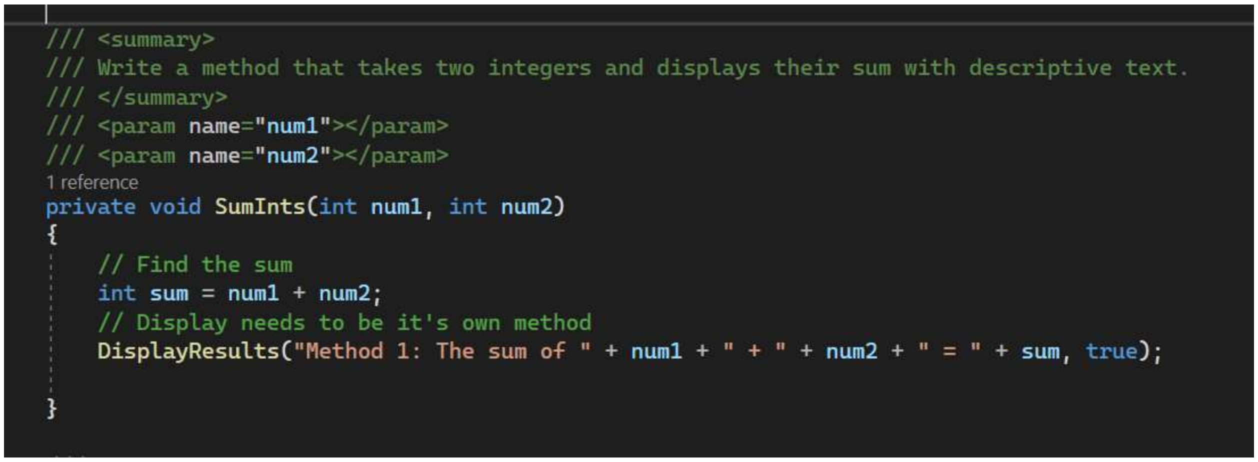


```
    /// <summary>
    /// Button click event handler to execute the methods
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    1 reference
    private void BtnExecuteMethods(object sender, EventArgs e)
    {
        // This will be considered our Main Method and our
        // goal is to keep this method clean (no logic just calling
        // methods).

    }
}
```

Figure 31: Goal for the App

    b. We need to first create a method so we can call it. Requirements: Write a method that takes two integers and displays their sum with descriptive text.

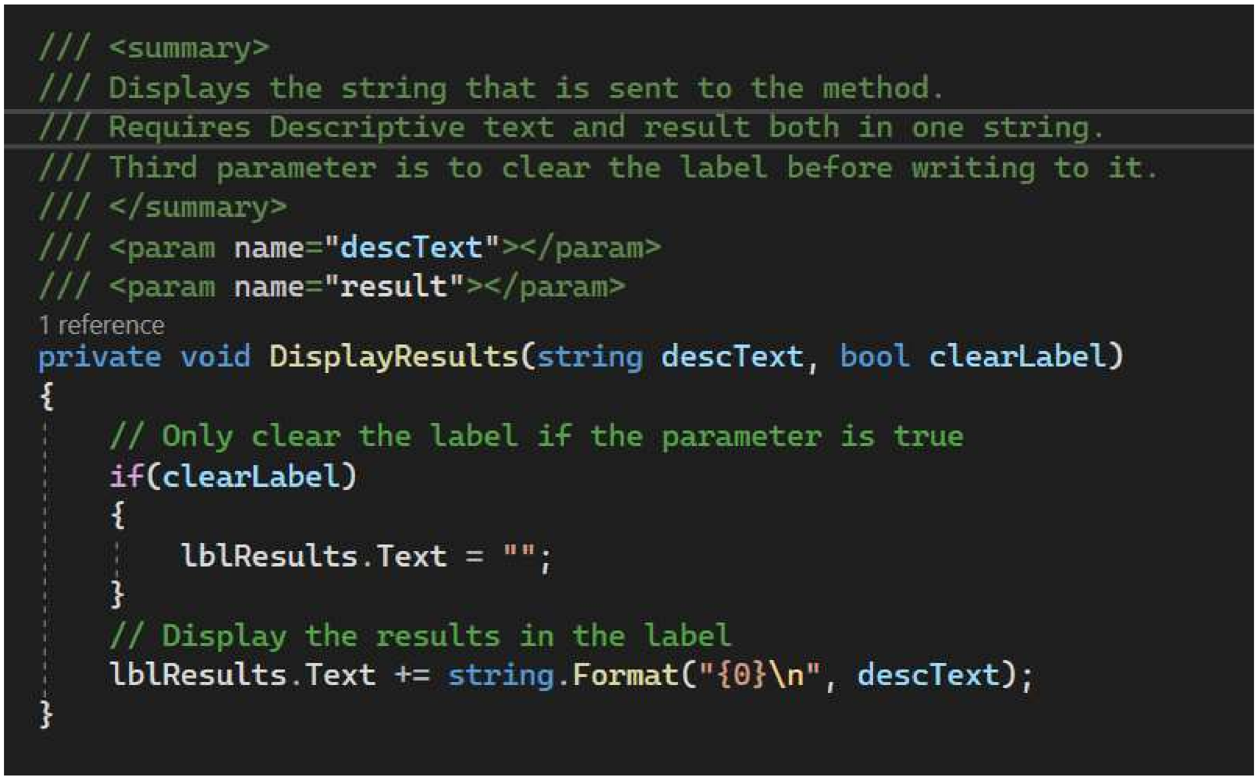
c. This first method is called SumInts as shown in Figure 32. Include the two parameters that are of a data type int. There is no return for this method, so return type is "void." DisplayResults is a method that we will right shortly that has one function to display a string.



```
/// <summary>
/// Write a method that takes two integers and displays their sum with descriptive text.
/// </summary>
/// <param name="num1"></param>
/// <param name="num2"></param>
1 reference
private void SumInts(int num1, int num2)
{
    // Find the sum
    int sum = num1 + num2;
    // Display needs to be it's own method
    DisplayResults("Method 1: The sum of " + num1 + " + " + num2 + " = " + sum, true);

}
```

Figure 32: Method 1

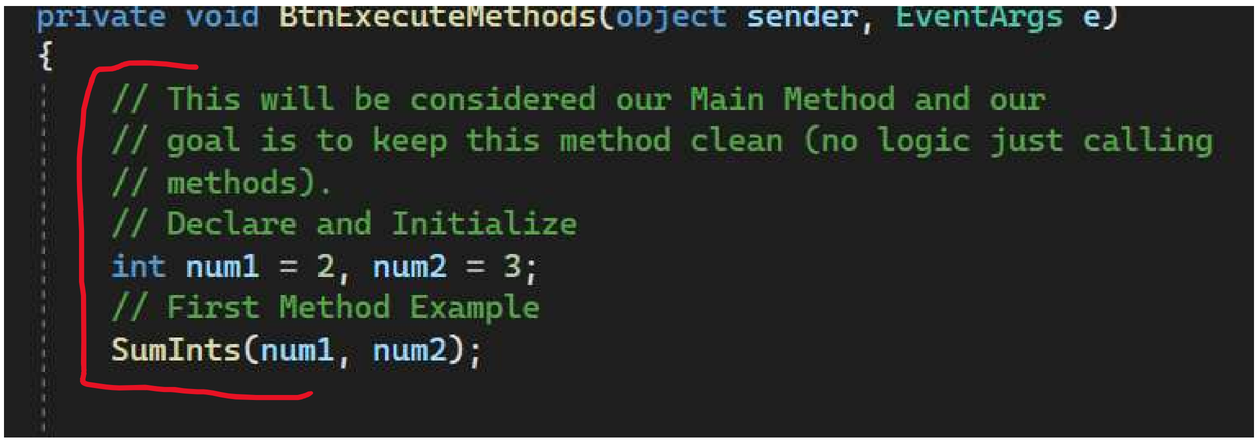
d. Create the Display Results method as shown in Figure 33. The method level comments should describe exactly the purpose of this method. Be sure every method has method level comments "<summary>."



```
/// <summary>
/// Displays the string that is sent to the method.
/// Requires Descriptive text and result both in one string.
/// Third parameter is to clear the label before writing to it.
/// </summary>
/// <param name="descText"></param>
/// <param name="result"></param>
1 reference
private void DisplayResults(string descText, bool clearLabel)
{
    // Only clear the label if the parameter is true
    if(clearLabel)
    {
        lblResults.Text = "";
    }
    // Display the results in the label
    lblResults.Text += string.Format("{0}\n", descText);
}
```

Figure 33: Display Results Method

e. Back up the to the Event Handler and prepare the event handler method to call the first method as shown in Figure 34.

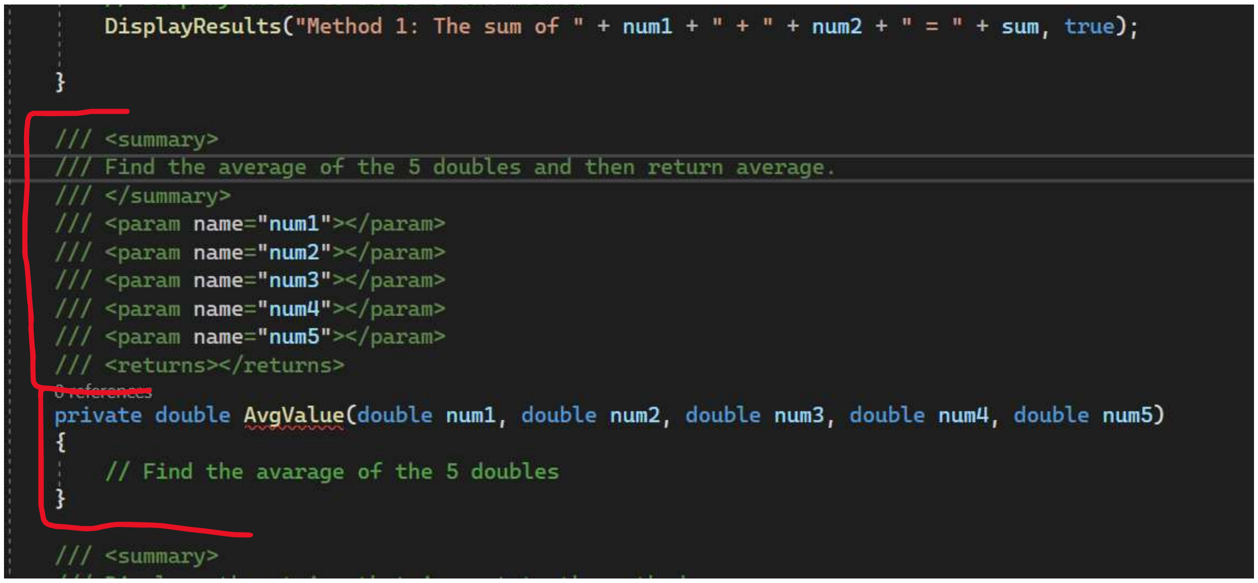


```
private void BtnExecuteMethods(object sender, EventArgs e)
{
    // This will be considered our Main Method and our
    // goal is to keep this method clean (no logic just calling
    // methods).
    // Declare and Initialize
    int num1 = 2, num2 = 3;
    // First Method Example
    SumInts(num1, num2);
```

Figure 34: Call First Method

6. Configure the FrmMain for Method 2

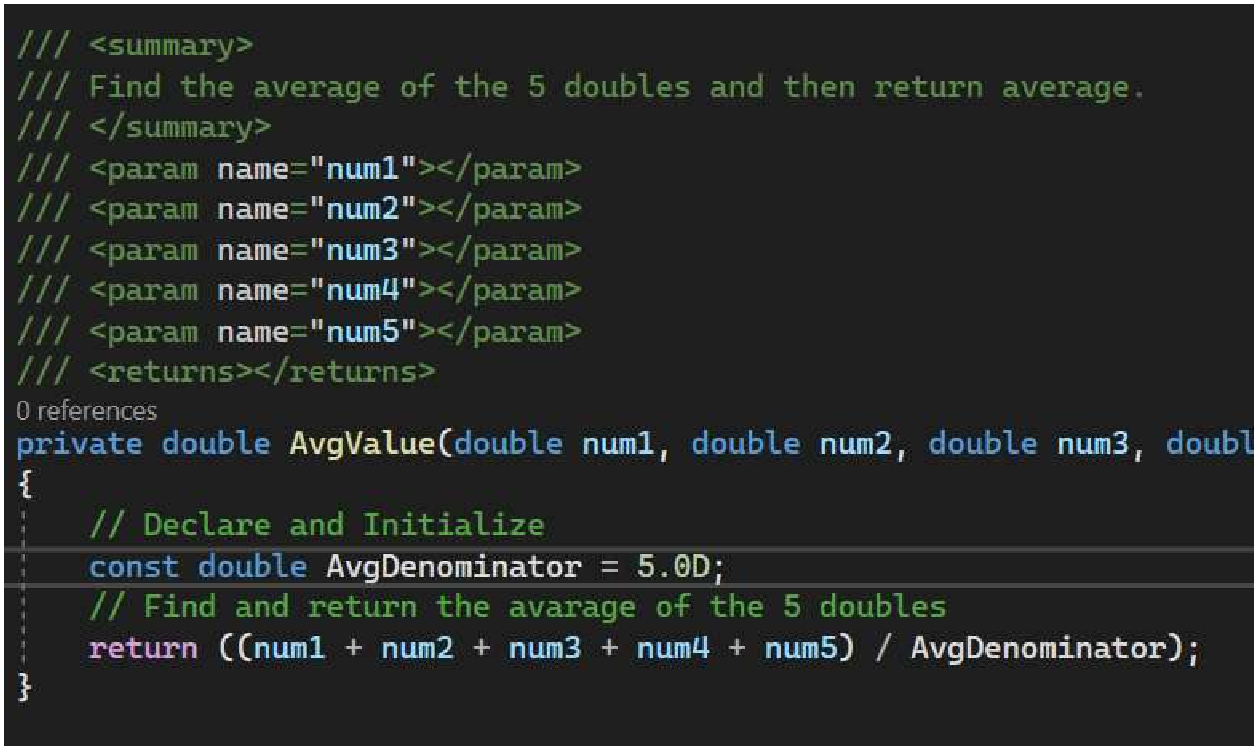
a. Write a method that takes five doubles and returns their average. Display the average value with descriptive text as shown in Figure 35.



```
    DisplayResults("Method 1: The sum of " + num1 + " + " + num2 + " = " + sum, true);

}

/// <summary>
/// Find the average of the 5 doubles and then return average.
/// </summary>
/// <param name="num1"></param>
/// <param name="num2"></param>
/// <param name="num3"></param>
/// <param name="num4"></param>
/// <param name="num5"></param>
/// <returns></returns>
0 references
private double AvgValue(double num1, double num2, double num3, double num4, double num5)
{
    // Find the avarage of the 5 doubles
}

/// <summary>
```

Figure 35: Method 2 Step 1

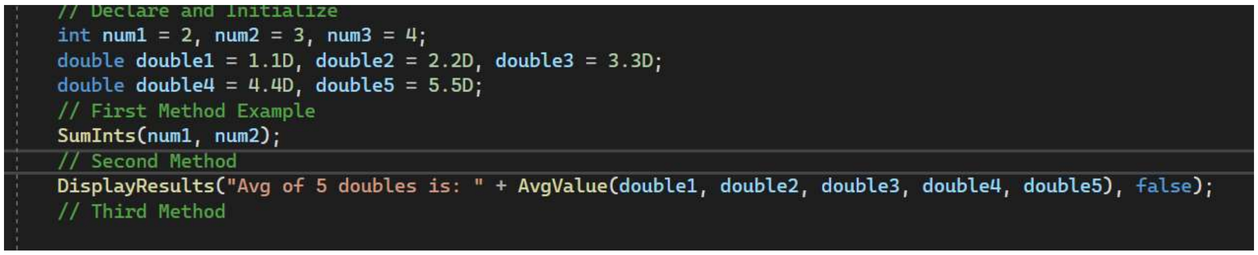
b. Create the logic for this method and return the double. Be sure the const is a double; if we have double/int we get double as a result as shown in Figure 36.



```
/// <summary>
/// Find the average of the 5 doubles and then return average.
/// </summary>
/// <param name="num1"></param>
/// <param name="num2"></param>
/// <param name="num3"></param>
/// <param name="num4"></param>
/// <param name="num5"></param>
/// <returns></returns>
0 references
private double AvgValue(double num1, double num2, double num3, doubl
{
    // Declare and Initialize
    const double AvgDenominator = 5.0D;
    // Find and return the avarage of the 5 doubles
    return ((num1 + num2 + num3 + num4 + num5) / AvgDenominator);
}
```

Figure 36: Method 2 Step 2

c. Back to the Event Handler and call this method and display the results as shown in Figure 37.



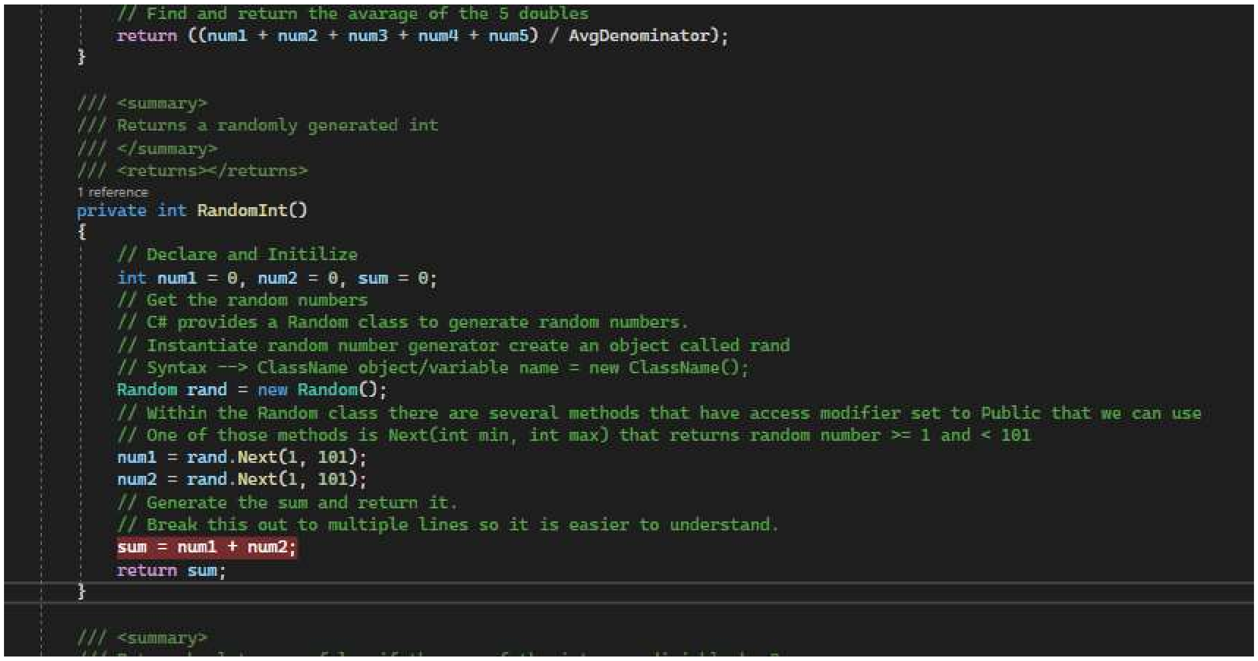
```
    // Declare and Initialize
    int num1 = 2, num2 = 3, num3 = 4;
    double double1 = 1.1D, double2 = 2.2D, double3 = 3.3D;
    double double4 = 4.4D, double5 = 5.5D;
    // First Method Example
    SumInts(num1, num2);
    // Second Method
    DisplayResults("Avg of 5 doubles is: " + AvgValue(double1, double2, double3, double4, double5), false);
    // Third Method
```

Figure 37: Call Method 2

7. Configure the FrmMain for Method 3

   a. Write a method that returns the sum of two randomly generated integers. Display the sum with descriptive text as shown in Figure 38.

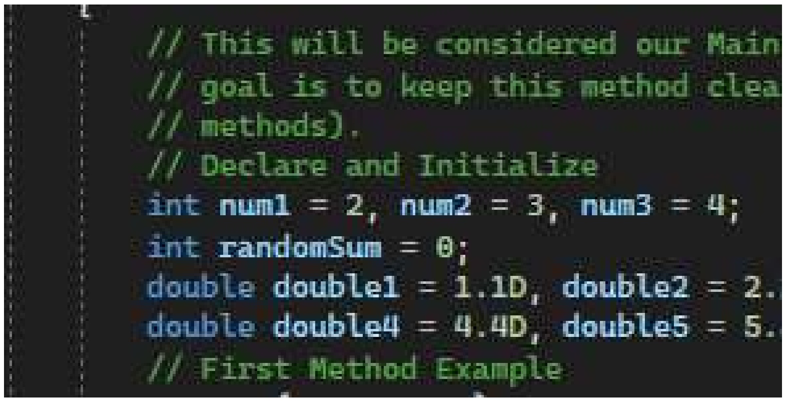
   b. Follow the comments provided in the method to understand the process in this method.



```
    // Find and return the avarage of the 5 doubles
    return ((num1 + num2 + num3 + num4 + num5) / AvgDenominator);
}

/// <summary>
/// Returns a randomly generated int
/// </summary>
/// <returns></returns>
1 reference
private int RandomInt()
{
    // Declare and Initilize
    int num1 = 0, num2 = 0, sum = 0;
    // Get the random numbers
    // C# provides a Random class to generate random numbers.
    // Instantiate random number generator create an object called rand
    // Syntax --> ClassName object/variable name = new ClassName();
    Random rand = new Random();
    // Within the Random class there are several methods that have access modifier set to Public that we can use
    // One of those methods is Next(int min, int max) that returns random number >= 1 and < 101
    num1 = rand.Next(1, 101);
    num2 = rand.Next(1, 101);
    // Generate the sum and return it.
    // Break this out to multiple lines so it is easier to understand.
    sum = num1 + num2;
    return sum;
}

/// <summary>
```

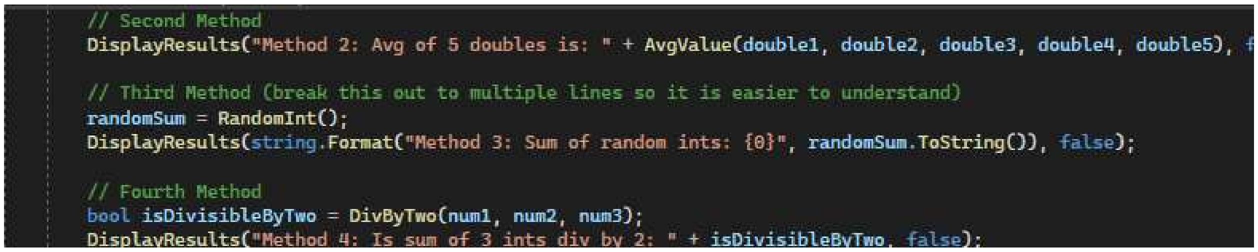
Figure 38: Random Number Generator

c. Back to the Event Handler and call this method and display the results as shown in Figure 39 and Figure 40.



```
    // This will be considered our Main
    // goal is to keep this method clea
    // methods).
    // Declare and Initialize
    int num1 = 2, num2 = 3, num3 = 4;
    int randomSum = 0;
    double double1 = 1.1D, double2 = 2.
    double double4 = 4.4D, double5 = 5.
    // First Method Example
```

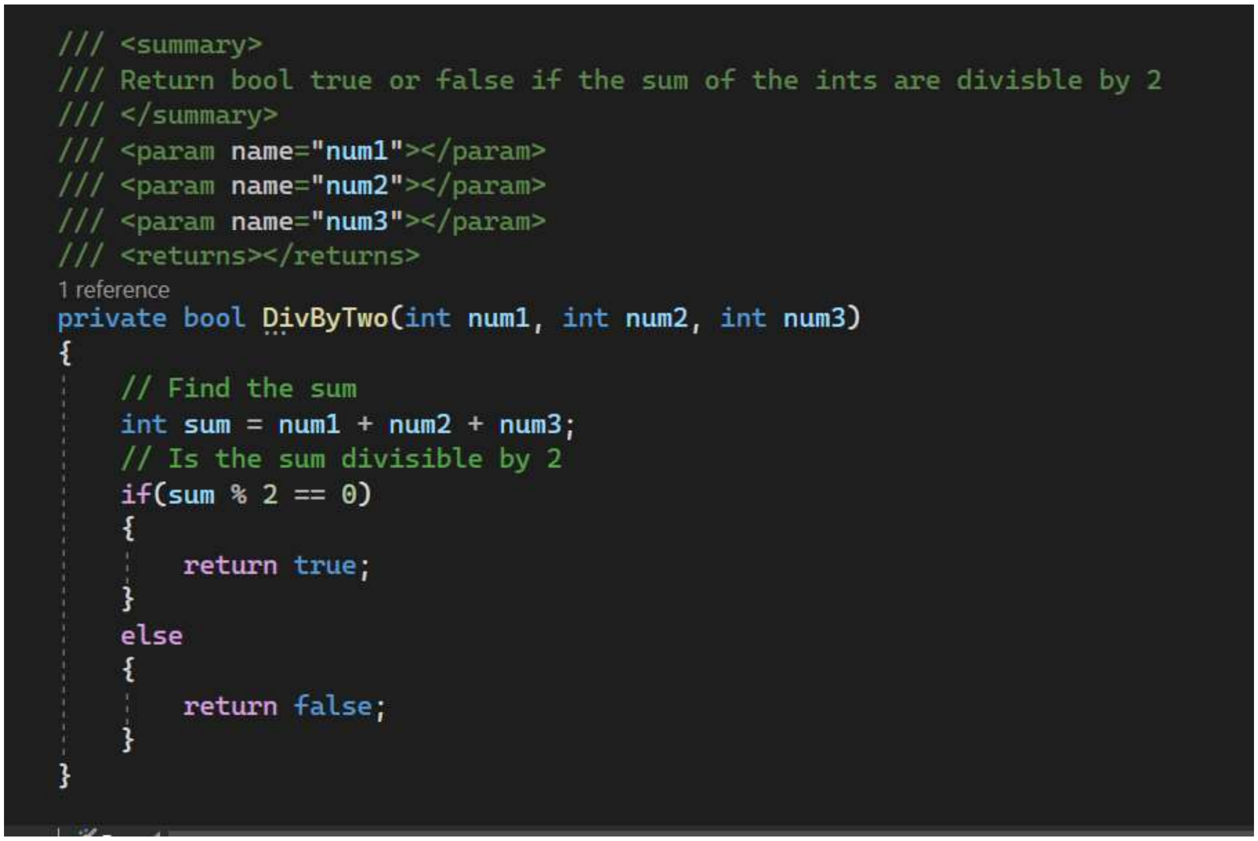
Figure 39: Declare and Initialize



```
// Second Method
DisplayResults("Method 2: Avg of 5 doubles is: " + AvgValue(double1, double2, double3, double4, double5), f

// Third Method (break this out to multiple lines so it is easier to understand)
randomSum = RandomInt();
DisplayResults(string.Format("Method 3: Sum of random ints: {0}", randomSum.ToString()), false);

// Fourth Method
bool isDivisibleByTwo = DivByTwo(num1, num2, num3);
DisplayResults("Method 4: Is sum of 3 ints div by 2: " + isDivisibleByTwo, false);
```

Figure 40: Call the method.

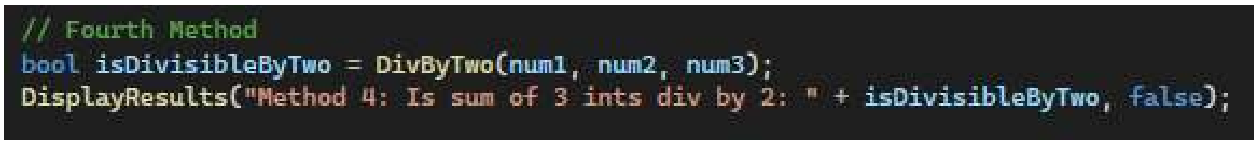
8. Configure the FrmMain for Method 4
    a. Write a method that takes three integers and returns true if their sum is divisible by 2, false otherwise. Display the results with descriptive text as shown in Figure 41.



```
/// <summary>
/// Return bool true or false if the sum of the ints are divisble by 2
/// </summary>
/// <param name="num1"></param>
/// <param name="num2"></param>
/// <param name="num3"></param>
/// <returns></returns>
1 reference
private bool DivByTwo(int num1, int num2, int num3)
{
    // Find the sum
    int sum = num1 + num2 + num3;
    // Is the sum divisible by 2
    if(sum % 2 == 0)
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

Figure 41: Method 4

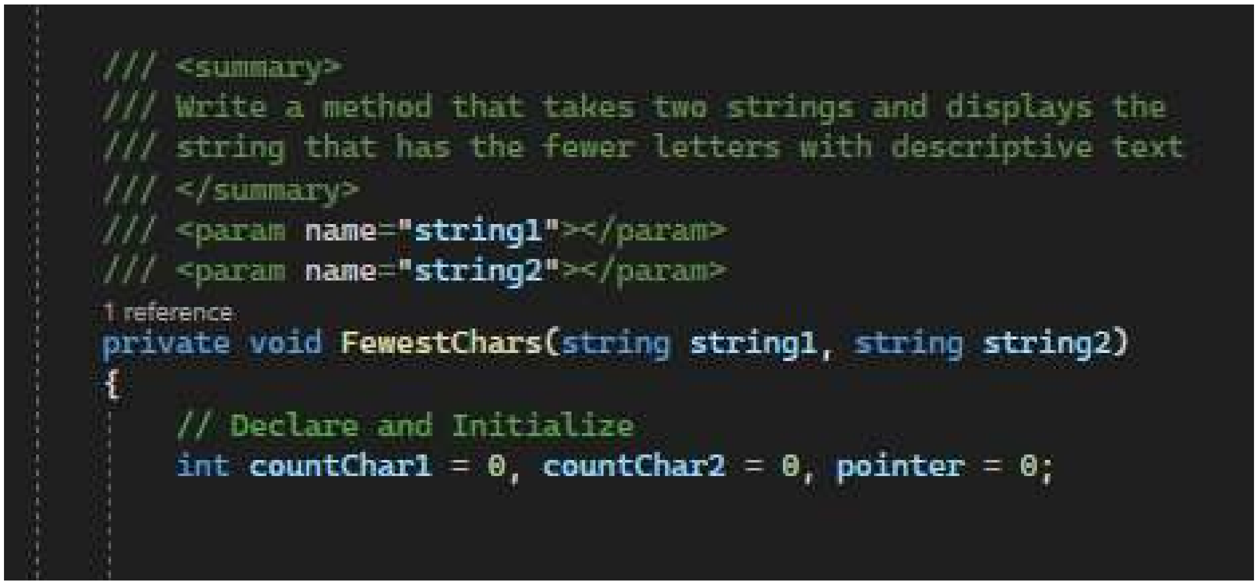
    b. Back to the Event Handler and call this method and display the results as shown in Figure 42.



```
// Fourth Method
bool isDivisibleByTwo = DivByTwo(num1, num2, num3);
DisplayResults("Method 4: Is sum of 3 ints div by 2: " + isDivisibleByTwo, false);
```

Figure 42: Call Method 4

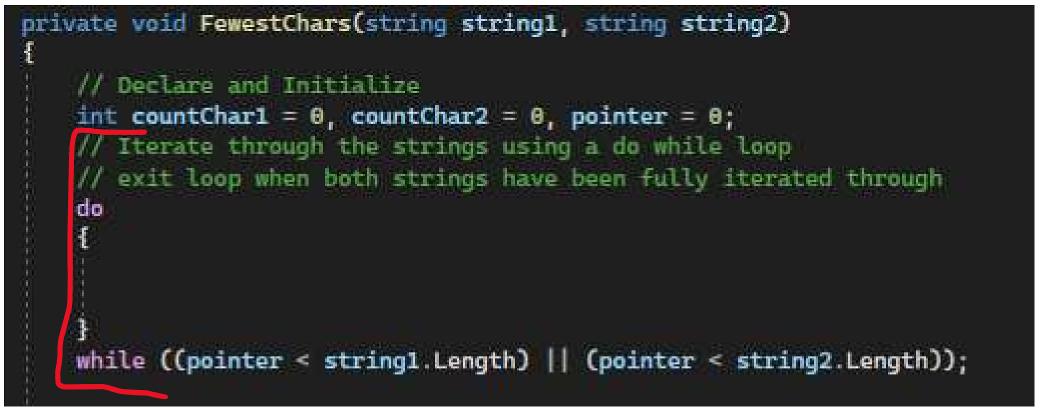
9. Configure the FrmMain for Method 5
    a. Write a method that takes two strings and displays the string that has the fewer letters with descriptive text as shown in Figure 43.



```
/// <summary>
/// Write a method that takes two strings and displays the
/// string that has the fewer letters with descriptive text
/// </summary>
/// <param name="string1"></param>
/// <param name="string2"></param>
1 reference
private void FewestChars(string string1, string string2)
{
    // Declare and Initialize
    int countChar1 = 0, countChar2 = 0, pointer = 0;
```

Figure 43: Method 5

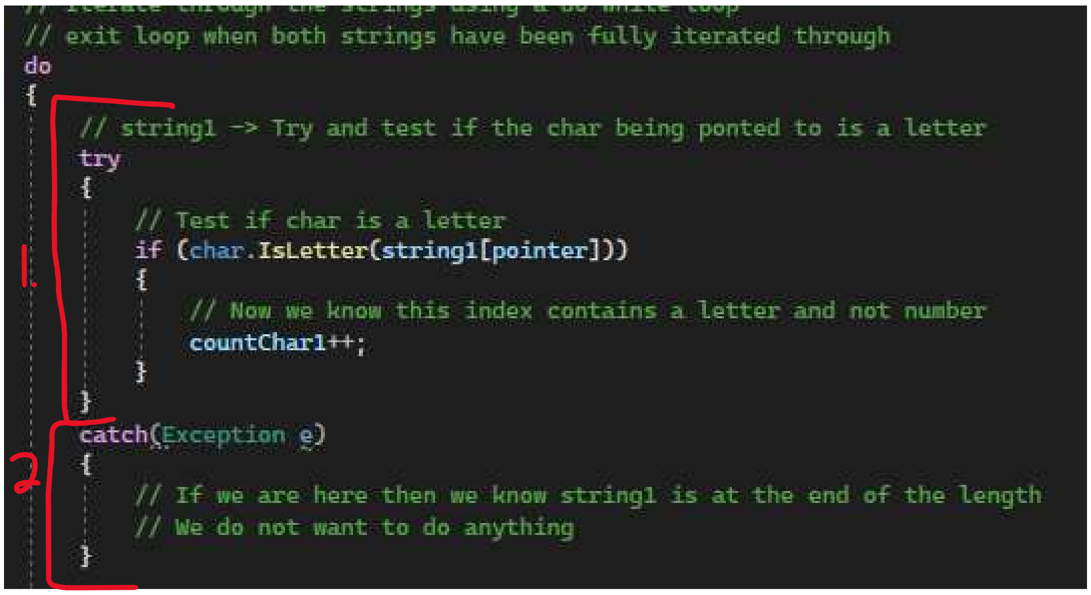
b. Now, we can start the do while loop as shown in Figure 44.



```
private void FewestChars(string string1, string string2)
{
    // Declare and Initialize
    int countChar1 = 0, countChar2 = 0, pointer = 0;
    // Iterate through the strings using a do while loop
    // exit loop when both strings have been fully iterated through
    do
    {


    }
    while ((pointer < string1.Length) || (pointer < string2.Length));
```

Figure 44: do while loop.

c. Populate the first do while loop as shown in Figure 45. Be sure to write every comment that will step you through the design process. This first do while is for the first string.

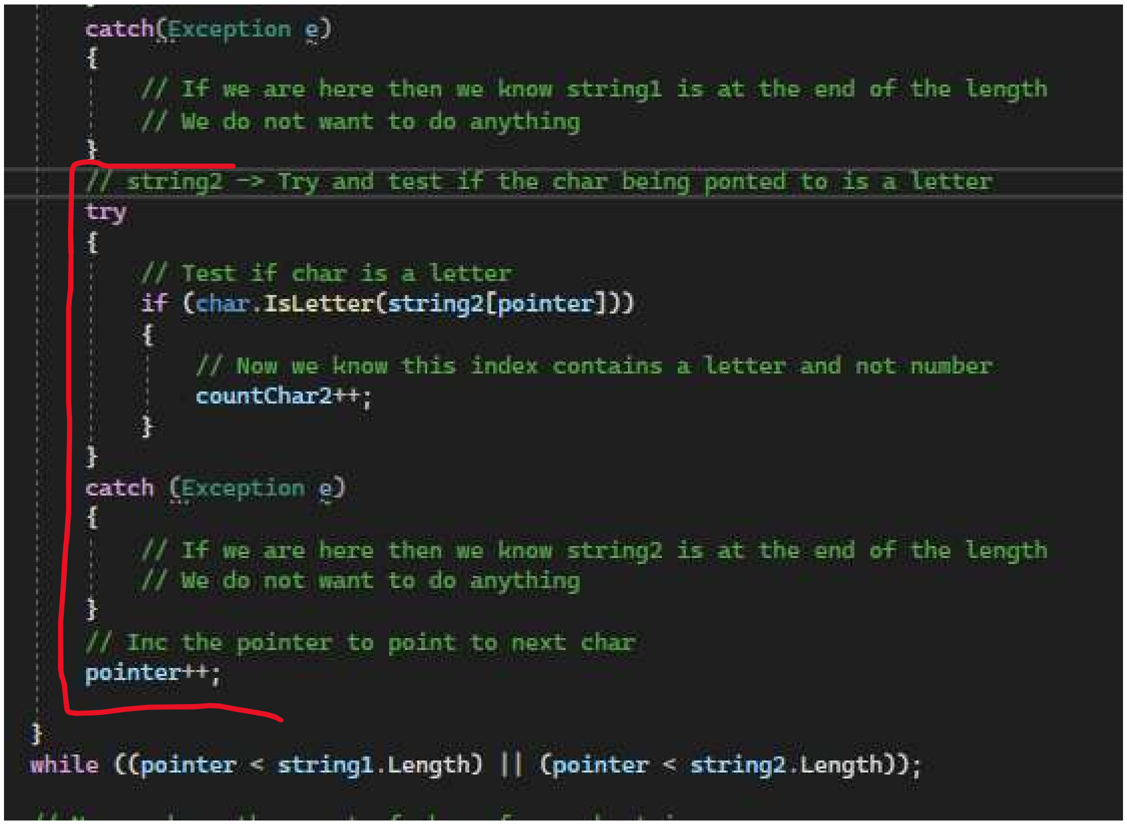


```
// exit loop when both strings have been fully iterated through
do
{
    // string1 -> Try and test if the char being ponted to is a letter
    try
    {
        // Test if char is a letter
        if (char.IsLetter(string1[pointer]))
        {
            // Now we know this index contains a letter and not number
            countChar1++;
        }
    }
    catch(Exception e)
    {
        // If we are here then we know string1 is at the end of the length
        // We do not want to do anything
    }
```

Figure 45: Populate do while loop.

d. Populate the second do while loop as shown in Figure 46. Be sure to write every comment that will step you through the design process. This second do while is for the second string.

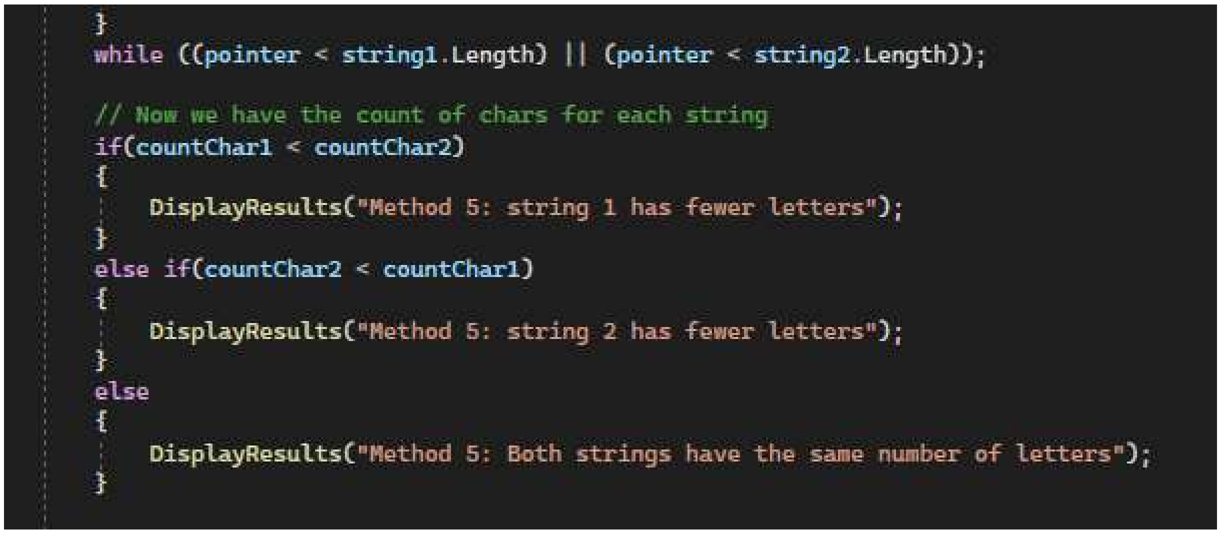
e. Typically, we would see this and notice we need to create a method, so this code is not repeated. Take it upon yourself to simplify this code by combining these two do while loops.



```
    catch(Exception e)
    {
        // If we are here then we know string1 is at the end of the length
        // We do not want to do anything
    }
    // string2 -> Try and test if the char being ponted to is a letter
    try
    {
        // Test if char is a letter
        if (char.IsLetter(string2[pointer]))
        {
            // Now we know this index contains a letter and not number
            countChar2++;
        }
    }
    catch (Exception e)
    {
        // If we are here then we know string2 is at the end of the length
        // We do not want to do anything
    }
    // Inc the pointer to point to next char
    pointer++;

}
while ((pointer < string1.Length) || (pointer < string2.Length));
```

Figure 46: Second do while loop

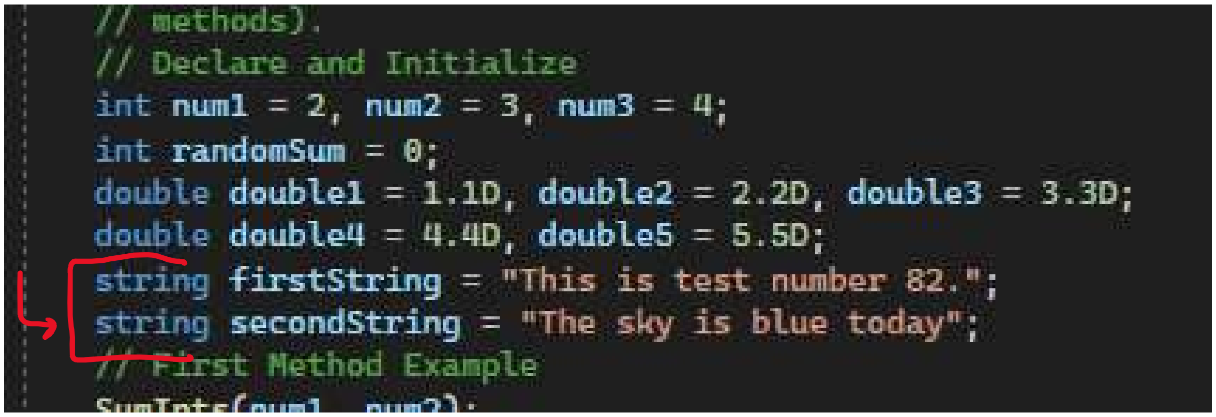
f. Now that we know the number of characters in each string, compare them and display the results as shown in Figure 47.



```
}
while ((pointer < string1.Length) || (pointer < string2.Length));

// Now we have the count of chars for each string
if(countChar1 < countChar2)
{
    DisplayResults("Method 5: string 1 has fewer letters");
}
else if(countChar2 < countChar1)
{
    DisplayResults("Method 5: string 2 has fewer letters");
}
else
{
    DisplayResults("Method 5: Both strings have the same number of letters");
}
```

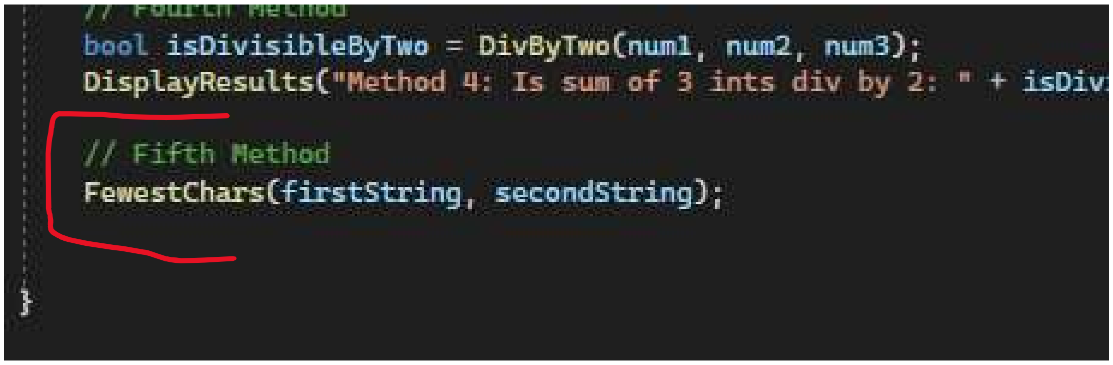
Figure 47: Display the Results

g. Go back to the Event Handler and call this method and display the results as shown in Figure 49Figure 42. First, we need to declare and initialize the two strings to test as shown in Figure 48.



```
// methods).
// Declare and Initialize
int num1 = 2, num2 = 3, num3 = 4;
int randomSum = 0;
double double1 = 1.1D, double2 = 2.2D, double3 = 3.3D;
double double4 = 4.4D, double5 = 5.5D;
string firstString = "This is test number 82.";
string secondString = "The sky is blue today";
// First Method Example
SumInts(num1, num2);
```

Figure 48: Declare and Initialize

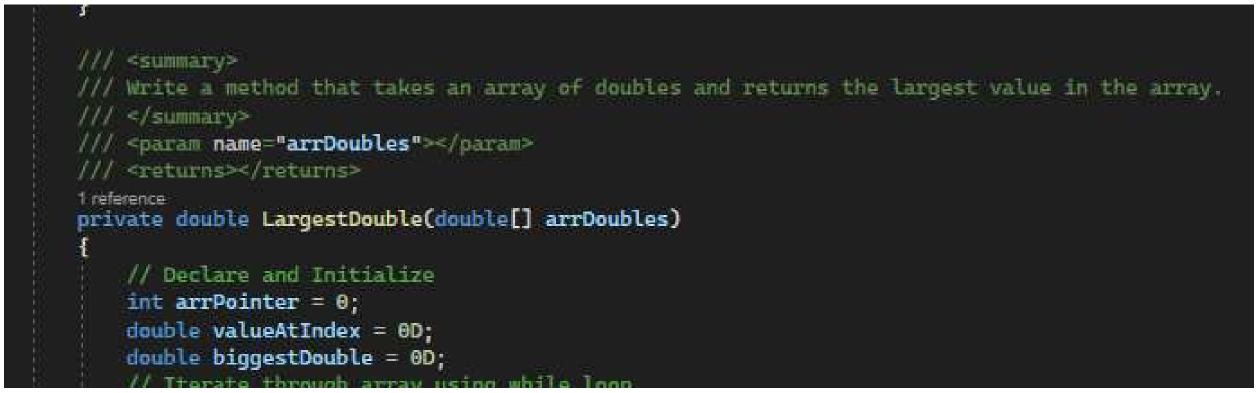


```
// Fourth Method
bool isDivisibleByTwo = DivByTwo(num1, num2, num3);
DisplayResults("Method 4: Is sum of 3 ints div by 2: " + isDiv

// Fifth Method
FewestChars(firstString, secondString);


}
```

Figure 49: Call method 5

10. Configure the FrmMain for Method 6

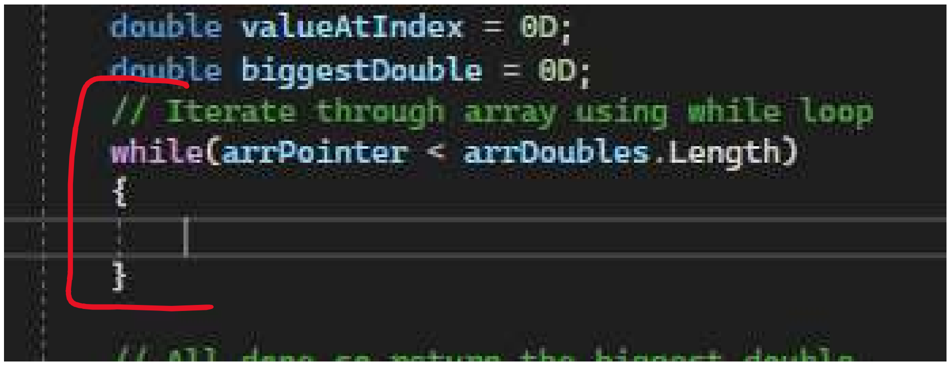
a. Write a method that takes an array of doubles and returns the largest value in the array. Display the double value with descriptive text as shown in Figure 50.



```
/// <summary>
/// Write a method that takes an array of doubles and returns the largest value in the array.
/// </summary>
/// <param name="arrDoubles"></param>
/// <returns></returns>
1 reference
private double LargestDouble(double[] arrDoubles)
{
    // Declare and Initialize
    int arrPointer = 0;
    double valueAtIndex = 0D;
    double biggestDouble = 0D;
```

Figure 50: Method 6

b. Use a while loop to iterate through the array. We would typically use foreach loop, but we wanted to do something different and show how while loop is used as shown in Figure 51.

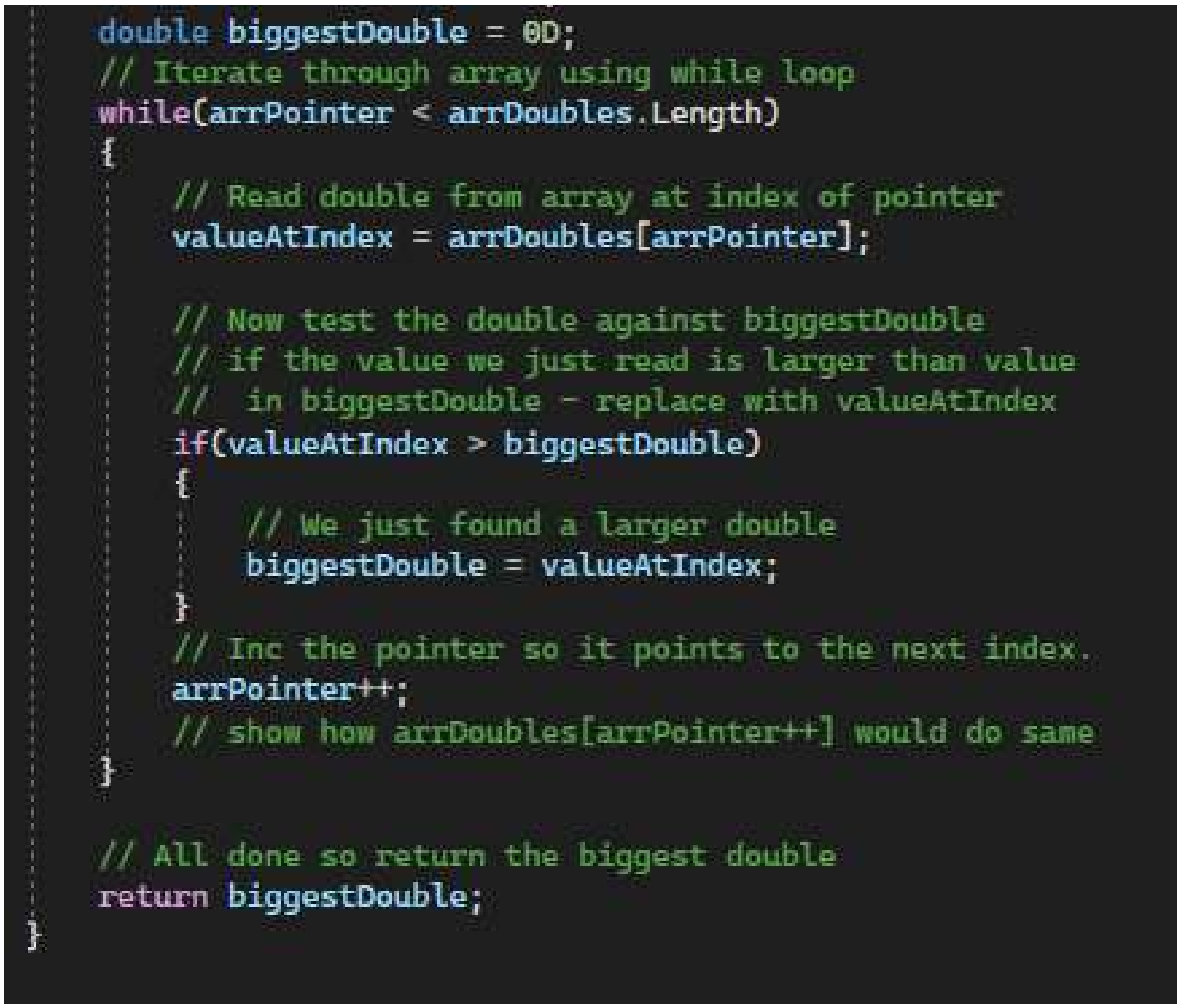


```
    double valueAtIndex = 0D;
    double biggestDouble = 0D;
    // Iterate through array using while loop
    while(arrPointer < arrDoubles.Length)
    {

    }
```

Figure 51: While Loop

c. Inside the while loop, check for the largest double as shown in Figure 52.

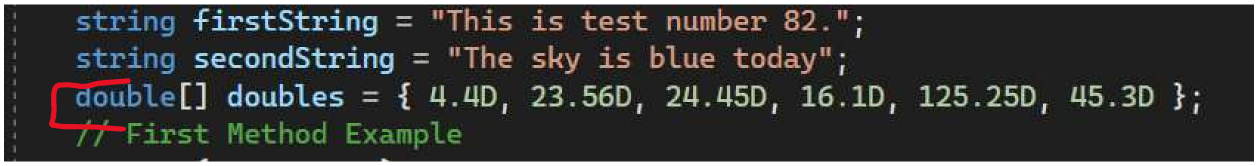
d. Then, return the largest double.



```
    double biggestDouble = 0D;
    // Iterate through array using while loop
    while(arrPointer < arrDoubles.Length)
    {
        // Read double from array at index of pointer
        valueAtIndex = arrDoubles[arrPointer];

        // Now test the double against biggestDouble
        // if the value we just read is larger than value
        //  in biggestDouble - replace with valueAtIndex
        if(valueAtIndex > biggestDouble)
        {
            // We just found a larger double
            biggestDouble = valueAtIndex;
        }
        // Inc the pointer so it points to the next index.
        arrPointer++;
        // show how arrDoubles[arrPointer++] would do same
    }

    // All done so return the biggest double
    return biggestDouble;
}
```

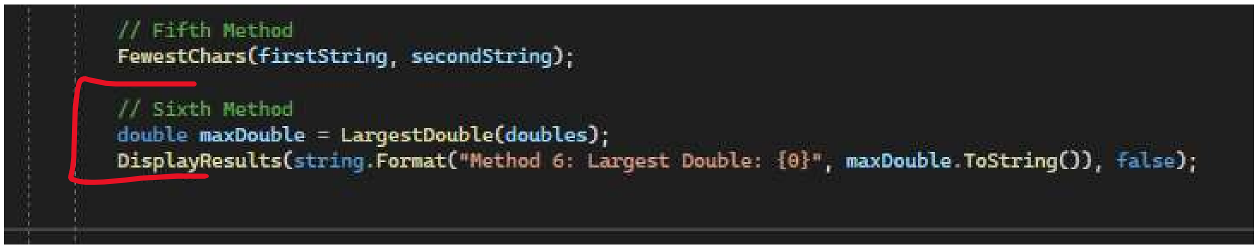
Figure 52: Check for Largest Double

a. Back to the Event Handler and call this method and display the results as shown in Figure 54Figure 42. First, we need to declare and initialize the array to test as shown in Figure 53Figure 48.



```
    string firstString = "This is test number 82.";
    string secondString = "The sky is blue today";
    double[] doubles = { 4.4D, 23.56D, 24.45D, 16.1D, 125.25D, 45.3D };
    // First Method Example
```

Figure 53: Declare and initialize for the double.



```
// Fifth Method
FewestChars(firstString, secondString);

// Sixth Method
double maxDouble = LargestDouble(doubles);
DisplayResults(string.Format("Method 6: Largest Double: {0}", maxDouble.ToString()), false);
```

Figure 54: Show results for Method 6.

11. Your Turn to Complete Methods 7, 8, and 9.
    b. Use the assignment in the digital classroom to guide you through the method requirements. You have the tools to complete these based on previous methods written, as well as the textbook.

12. Submit the Activity as described in the digital classroom.