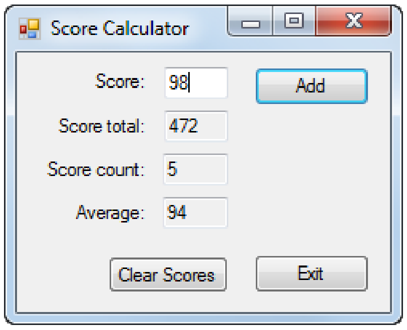
Week 9 – Exercise 1 Accumulate test score data

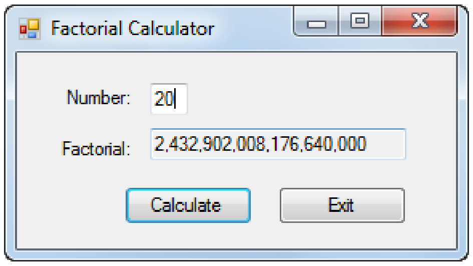
In this exercise, you'll create a form that accepts one or more scores from the user. Each time a score is added, the score total, score count, and average score are calculated and displayed.



1. Start a new WPF project named ScoreCalculator.
2. Add labels, text boxes, and buttons to the default form and set the properties of the form and its controls so they appear as shown above. When the user presses the Enter key, the Click event of the Add button should fire. When the user presses the Esc key, the Click event of the Exit button should fire.
3. Declare two class variables to store the score total and the score count.
4. Create an event handler for the Click event of the Add button. This event handler should get the score the user enters, calculate and display the score total, score count, and average score, and move the focus to the Score text box. It should provide for integer entries, but you can assume that the user will enter valid integer values.
5. Create an event handler for the Click event of the Clear Scores button. This event handler should set the two class variables to zero, clear the text boxes on the form, and move the focus to the Score text box.
6. Create an event handler for the Click event of the Exit button that closes the form.
7. Test the application to be sure it works correctly.

Week 9 – Exercise 2 Calculate the factorial of a number

In this exercise, you'll create a WPF application that accepts an integer from the user and then calculates the factorial of that integer.



Number: 20

Factorial: 2,432,902,008,176,640,000

The factorial of an integer is that integer multiplied by every positive integer less than itself. A factorial number is identified by an exclamation point following the number. Here's how you calculate the factorial of the numbers 1 through 5:

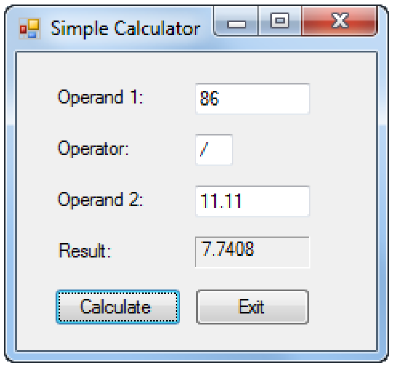
1! = 1	which equals 1
2! = 2 * 1	which equals 2
3! = 3 * 2 * 1	which equals 6
4! = 4 * 3 * 2 * 1	which equals 24
5! = 5 * 4 * 3 * 2 * 1	which equals 120

To be able to store the large integer values for the factorial, this application can't use the Int32 data type.

1. Start a new WPF project named Factorial
2. Add labels, text boxes, and buttons to the default form and set the properties of the form and its controls so they appear as shown above. When the user presses the Enter key, the Click event of the Calculate button should fire. When the user presses the Esc key, the Click event of the Exit button should fire.
3. Create an event handler for the Click event of the Calculate button. This event handler should get the number the user enters, calculate the factorial of that number, display the factorial with commas but no decimal places, and move the focus to the Number text box. It should return an accurate value for integers from 1 to 20. (The factorial of the number 20 is shown in the form above.)
4. Create an event handler for the Click event of the Exit button that closes the form.
5. Test the application to be sure it works correctly.

Week 9 – Exercise 3 Create a simple calculator

In this exercise, you'll create a form that accepts two operands and an operator from the user and then performs the requested operation.



Operand 1: 86
Operator: /
Operand 2: 11.11
Result: 7.7408

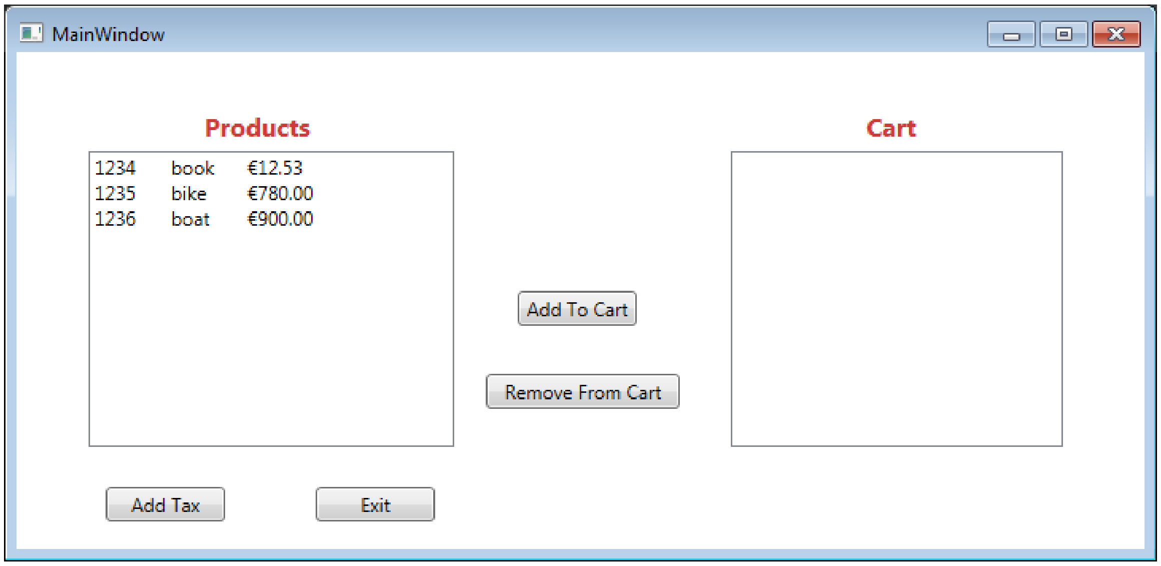
1. Start a new project named SimpleCalculator.
2. Add labels, text boxes, and buttons to the default form and set the properties of the form and its controls so they appear as shown above. When the user presses the Enter key, the Click event of the Calculate button should fire. When the user presses the Esc key, the Click event of the Exit button should fire.
3. Code a private method named Calculate that performs the requested operation and returns a decimal value. This method should accept the following arguments:

Argument	Description
decimal operand1	The value entered for the first operand.
string operator1	One of these four operators: +, -, *, or /.
decimal operand2	The value entered for the second operand.

4. Create an event handler for the Click event of the Calculate button. This event handler should get the two numbers and operand the user enters, call the Calculate method to get the result of the calculation, display the result rounded to four decimal places, and move the focus to the Operand 1 text box.
5. Create an event handler for the Click event of the Exit button that closes the form.
6. Create an event handler that clears the Result text box if the user changes the text in any of the other text boxes.
7. Test the application to be sure it works correctly.

Week 9 – Exercise 4 Create a simple Shopping Cart

In this exercise, you'll create the Shopping Cart WPF application that was worked on in the Lecture.



1234	book	€12.53
1235	bike	€780.00
1236	boat	€900.00

1. Create the XAML interface shown above with the 4 buttons and 2 ListBoxes. Names and associated click handler as follows:

```
Name=listBoxProducts
Name=listBoxCart
Name=btnAddTax                  Click="btnAddTax_Click"
Name=btnExit                    Click="btnExit_Click"
Name=btnAddToCart               Click="btnAddToCart_Click"
Name=btnRemoveFromCart          Click="btnRemoveFromCart_Click"
```

2. Create a Product class with auto-implemented methods for ID, Name and Price.
3. In the MainWindow.xaml.cs file create 3 Products.
4. Store the 3 Products in a Product Array.
5. Create another Product Array to store Cart Products.
6. Set the Loaded property in the XAML code for "Window" to "OnWindowLoad"

```
<Window x:Class="ShoppingCart.MainWindow"
    .
    .
    .

  Title="MainWindow" Height="350" Width="724.808" Loaded="OnWindowLoaded">
```

7. Right click "OnWindowLoad" and select "Go to Definition", this generates the method signature in the .CS code.
   Add the following code to put the Array of Products on the Products List box:

```
  listBoxProducts.ItemsSource = Products;
```

8. Override the Product ToString() method to show ID, Name and Price. Test the application and ensure the list of products is displayed.
9. Add another auto implemented property to the Product to hold Tax and add a void method call AddTax(double taxRate), this calculates the tax based on product price and stores it in the Tax property.
10. Double click the Add Tax button and in the corresponding method loop through all Products in the Products Array and apply a tax value. You now need to empty the List Box and add the array again. To empty the List Box use:

    ```
                listBoxProducts.ItemsSource = "";
    ```

11. Update the Product.ToString() method and show the Tax if it is set.
12. Click the Add Tax button and verify the Tax amount now appears along-side each product.
13. Using the click handler for the Add to Cart button when an item is selected in the product list box add it to the Array of Cart products and display the Array on the Cart list box. Also check an item is selected before attempting to add it to the Cart Array.

    ```
    Product selectedProduct = listBoxProducts.SelectedItem as Product;
    if (selectedProduct != null)
    {
       // also verify the product is not already on the Cart
    }
    ```
14. Using the click handler for the Remove from Cart button when an item is selected in the cart list box remove it from the Array of Cart products, empty the Cart list box and redisplay the Array on the Cart list box.
15. Fully test the adding and removing of Products from/to the Cart.