How to code methods and event handlers

The basic syntax for coding a method

```
{public|private} returnType MethodName([parameterList])
{
    statements
}
```

A method with one parameter that returns a decimal value

```
private decimal GetDiscountPercent(decimal subtotal)
{
    decimal discountPercent = 0m;
    if (subtotal >= 500)
        discountPercent = .2m;
    else
        discountPercent = .1m;
    return discountPercent;
}
```

A method with three parameters that returns a decimal value

```
private decimal CalculateFutureValue(decimal
monthlyInvestment,
    decimal monthlyInterestRate, int months)
{
    decimal futureValue = 0m;
    for (int i = 0; i < months; i++)
    {
        futureValue = (futureValue + monthlyInvestment)
                    * (1 + monthlyInterestRate);
    }
    return futureValue;
}
```

The syntax for calling a method

```
[this.]MethodName([argumentList])
```

A statement that calls a method that has no parameters

```
this.DisableButtons();
```

A statement that passes one argument

```
decimal discountPercent =
this.GetDiscountPercent(subtotal);
```

A statement that passes three arguments

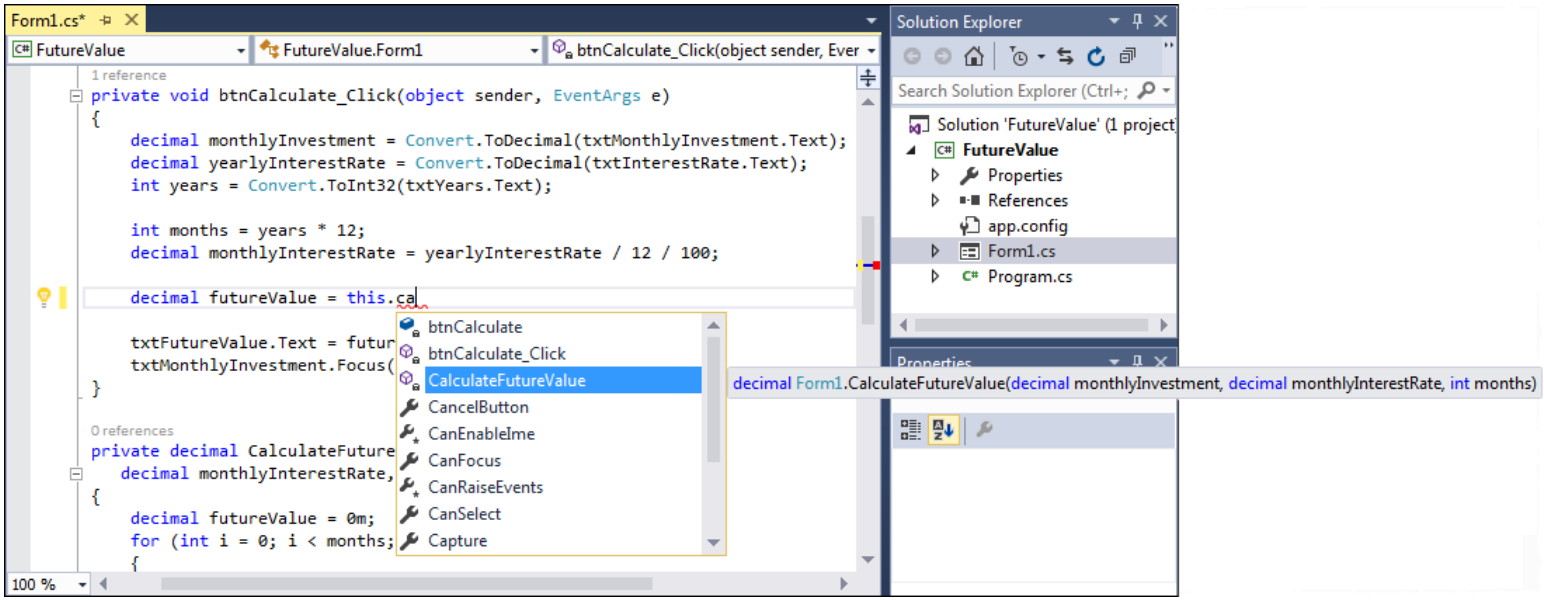
```
decimal futureValue = CalculateFutureValue(
    monthlyInvestment, monthlyInterestRate, months);
```

Future Value Example

```
//monthlyInvestment
Console.WriteLine("Enter in monthlyInvestment");
String monthlyInvestmentText = Console.ReadLine();
decimal monthlyInvestment = Convert.ToDecimal(monthlyInvestmentText);

//InterestRate
Console.WriteLine("Enter in InterestRate");
String InterestRateText = Console.ReadLine();
decimal yearlyInterestRate = Convert.ToDecimal(InterestRateText);

//years
Console.WriteLine("Enter in Years");
String YearsText = Console.ReadLine();
int years = Convert.ToInt32(YearsText);


 int months = years * 12;
 decimal monthlyInterestRate = yearlyInterestRate / 12 / 100;

 decimal futureValue = CalculateFutureValue(monthlyInvestment, monthlyInterestRate, months);

 String textfutureValue =  futureValue.ToString("c");
 Console.WriteLine("Future Value ={0}", textfutureValue);
 Console.ReadLine();
```

```
private static decimal CalculateFutureValue(decimal monthlyInvestment,
    decimal monthlyInterestRate, int months)
{
    decimal futureValue = 0m;
    for (int i = 0; i < months; i++)
    {
        futureValue = (futureValue + monthlyInvestment)
                    * (1 + monthlyInterestRate);
    }
    return futureValue;
}
```

The IntelliSense feature for calling a method



The syntax for an optional parameter

```
type parameterName = defaultvalue
```

The GetFutureValue method with two optional parameters

```
private decimal GetFutureValue(decimal monthlyInvestment,
        decimal monthlyInterestRate = 0.05m, int months = 12)
{
    decimal futureValue = 0m;
    for (int i = 0; i < months; i++)
    {
        futureValue = (futureValue + monthlyInvestment) *
                      (1 + monthlyInterestRate);
    }
    return futureValue;
}
```

A statement that passes arguments for all three parameters to the function

```
decimal futureValue =
    this.GetFutureValue(monthlyInvestment,
monthlyInterestRate, months)
```

A statement that omits the argument for the third parameter

```
decimal futureValue = this.GetFutureValue(
    monthlyInvestment, monthlyInterestRate)
```

Two statements that pass the arguments for two parameters by name

```
decimal futureValue =
    this.GetFutureValue(monthlyInvestment:monthlyInvestment,
                        months:months)
decimal futureValue =
    this.GetFutureValue(months:months,
                        monthlyInvestment:monthlyInvestment)
```

A statement that passes one argument by position and one by name

```
decimal futureValue =
    this.GetFutureValue(monthlyInvestment, months:months)
```

Passing Reference-Type Parameters

A variable of a reference type does not contain its data directly; it contains a reference to its data. When you pass a reference-type parameter by value, it is possible to change the data pointed to by the reference, such as the value of a class member. However, you cannot change the value of the reference itself; that is, you cannot use the same reference to allocate memory for a new class and have it persist outside the block. To do that, pass the parameter using the ref or out keyword. For simplicity, the following examples use `ref`.

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/passing-reference-type-parameters

Passing Reference Types by Value

The following example demonstrates passing a reference-type parameter, `arr`, by value, to a method, `Change`. Because the parameter is a reference to `arr`, it is possible to change the values of the array elements. However, the attempt to reassign the parameter to a different memory location only works inside the method and does not affect the original variable, `arr`.

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/passing-reference-type-parameters

Pass by Ref Example

```
class PassingRefByVal
{
    static void Change(int[] pArray)
    {
        pArray[0] = 888;  // This change affects the original element.
        pArray = new int[5] {-3, -1, -2, -3, -4};   // This change is local.
        System.Console.WriteLine("Inside the method, the first element is: {0}", pArray[0]);
    }

    static void Main()
    {
        int[] arr = {1, 4, 5};
        System.Console.WriteLine("Inside Main, before calling the method, the first element is: {0}", arr [0]);

        Change(arr);
        System.Console.WriteLine("Inside Main, after calling the method, the first element is: {0}", arr [0]);
    }
}
/* Output:
    Inside Main, before calling the method, the first element is: 1
    Inside the method, the first element is: -3
    Inside Main, after calling the method, the first element is: 888
*/
```

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/passing-reference-type-parameters

Pass by Ref Example

In the preceding example, the array, `arr`, which is a reference type, is passed to the method without the `ref` parameter. In such a case, a copy of the reference, which points to `arr`, is passed to the method. The output shows that it is possible for the method to change the contents of an array element, in this case from `1` to `888`. However, allocating a new portion of memory by using the new operator inside the `Change` method makes the variable `pArray` reference a new array. Thus, any changes after that will not affect the original array, `arr`, which is created inside `Main`. In fact, two arrays are created in this example, one inside `Main` and one inside the `Change` method.

Passing Reference Types by Reference

The following example is the same as the previous example, except that the `ref` keyword is added to the method header and call. Any changes that take place in the method affect the original variable in the calling program.

```
class PassingRefByRef
{
    static void Change(ref int[] pArray)
    {
        // Both of the following changes will affect the original variables:
        pArray[0] = 888;
        pArray = new int[5] {-3, -1, -2, -3, -4};
        System.Console.WriteLine("Inside the method, the first element is: {0}", pArray[0]);
    }

    static void Main()
    {
        int[] arr = {1, 4, 5};
        System.Console.WriteLine("Inside Main, before calling the method, the first element is: {0}", arr[0]);

        Change(ref arr);
        System.Console.WriteLine("Inside Main, after calling the method, the first element is: {0}", arr[0]);
    }
}
/* Output:
    Inside Main, before calling the method, the first element is: 1
    Inside the method, the first element is: -3
    Inside Main, after calling the method, the first element is: -3
*/
```

All of the changes that take place inside the method affect the original array in `Main`. In fact, the original array is reallocated using the `new` operator. Thus, after calling the `Change` method, any reference to `arr` points to the five-element array, which is created in the `Change` method.

Swapping Two Strings

Swapping strings is a good example of passing reference-type parameters by reference. In the example, two strings, `str1` and `str2`, are initialized in `Main` and passed to the `SwapStrings` method as parameters modified by the `ref` keyword. The two strings are swapped inside the method and inside `Main` as well.

```
class SwappingStrings
 {
     static void SwapStrings(ref string s1, ref string s2)
     // The string parameter is passed by reference.
     // Any changes on parameters will affect the original variables.
     {
         string temp = s1;
         s1 = s2;
         s2 = temp;
         System.Console.WriteLine("Inside the method: {0} {1}", s1, s2);
     }

     static void Main()
     {
         string str1 = "John";
         string str2 = "Smith";
         System.Console.WriteLine("Inside Main, before swapping: {0} {1}", str1, str2);

         SwapStrings(ref str1, ref str2);   // Passing strings by reference
         System.Console.WriteLine("Inside Main, after swapping: {0} {1}", str1, str2);
     }
 }
 /* Output:
     Inside Main, before swapping: John Smith
     Inside the method: Smith John
     Inside Main, after swapping: Smith John
*/
```

In this example, the parameters need to be passed by reference to affect the variables in the calling program. If you remove the `ref` keyword from both the method header and the method call, no changes will take place in the calling program.