Chapter 6

How to code methods and event handlers

Objectives

Applied

1. Given the specifications for a method, write the method.
2. Give the documentation for a method including its signature, use the method.
3. Give a portion of code that can be converted to a method call and a method, use refactoring to do the conversion.

Objectives (cont.)

Knowledge

1. Describe the signature of a method.
2. Explain the difference between passing an argument by value and by reference.
3. Describe the use of optional arguments.
4. Describe the advantages of passing arguments by name.
5. In general terms, describe the way event wiring works in C#, what you need to do to delete an event, and how you can use one event handler to handle two or more events.

The basic syntax for coding a method

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

A method with no parameters and no return type

```
private void DisableButtons()
{
    btnCalculate.Enabled = false;
    btnExit.Enabled = false;
}
```

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();
```

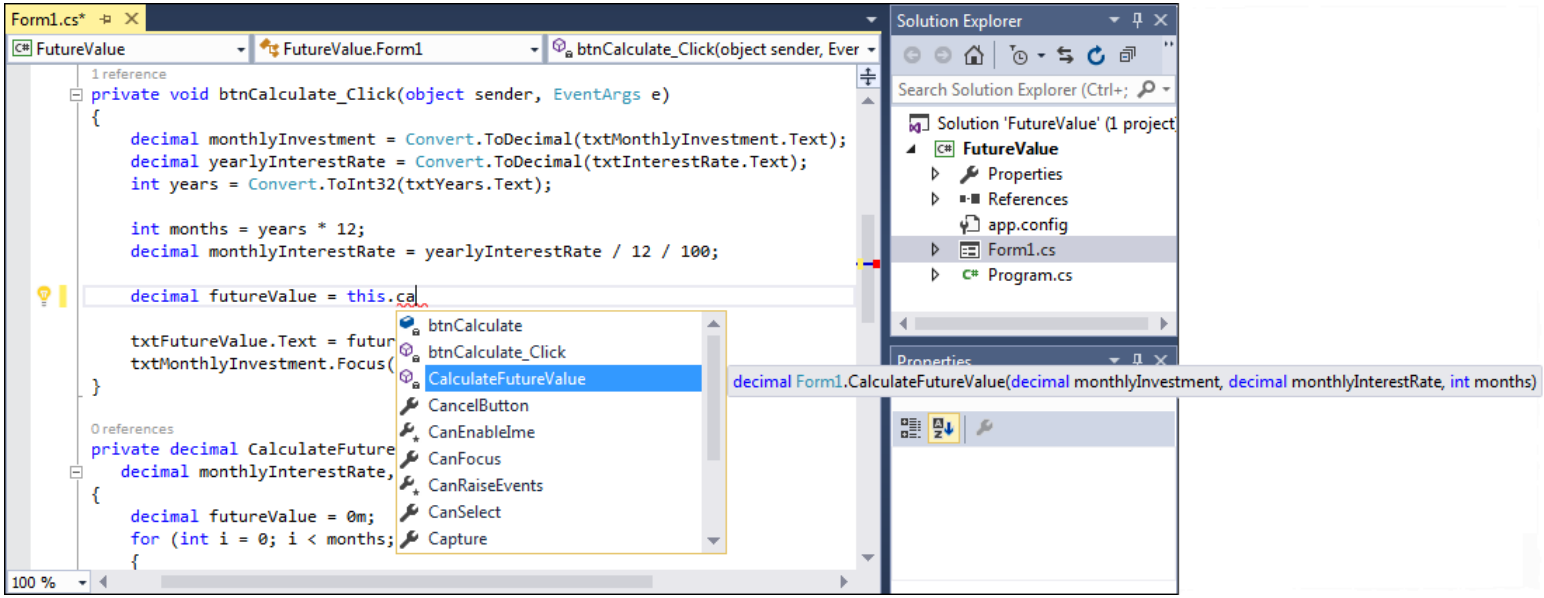
A statement that passes one argument

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

A statement that passes three arguments

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

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)
```

The btnCalculate_Click method with selected statements

```
private void btnCalculate_Click(object sender, EventArgs e)
{
    decimal monthlyInvestment =
        Convert.ToDecimal(txtMonthlyInvestment.Text);
    decimal yearlyInterestRate =
        Convert.ToDecimal(txtInterestRate.Text);
    int years = Convert.ToInt32(txtYears.Text);

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

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

    txtFutureValue.Text = futureValue.ToString("c");
    txtMonthlyInvestment.Focus();
}
```

The method call that replaces the selected code

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

The method that's added to the class

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

How to use refactoring to create a new method from existing code

1. Select the code you want to create a new method from, press Ctrl + period (.), and select Extract Method from the Quick Actions menu that's displayed. This adds a method named NewMethod to the class, replaces the selected code with a statement that calls the method, and displays the Rename dialog box for the method name.
2. Enter the name that you want to use for the new method and click the Apply button.

The basic syntax of the parameters in a parameter list

```
[{ref|out}] type variableName
```

A CalculateFutureValue method that uses the ref keyword

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

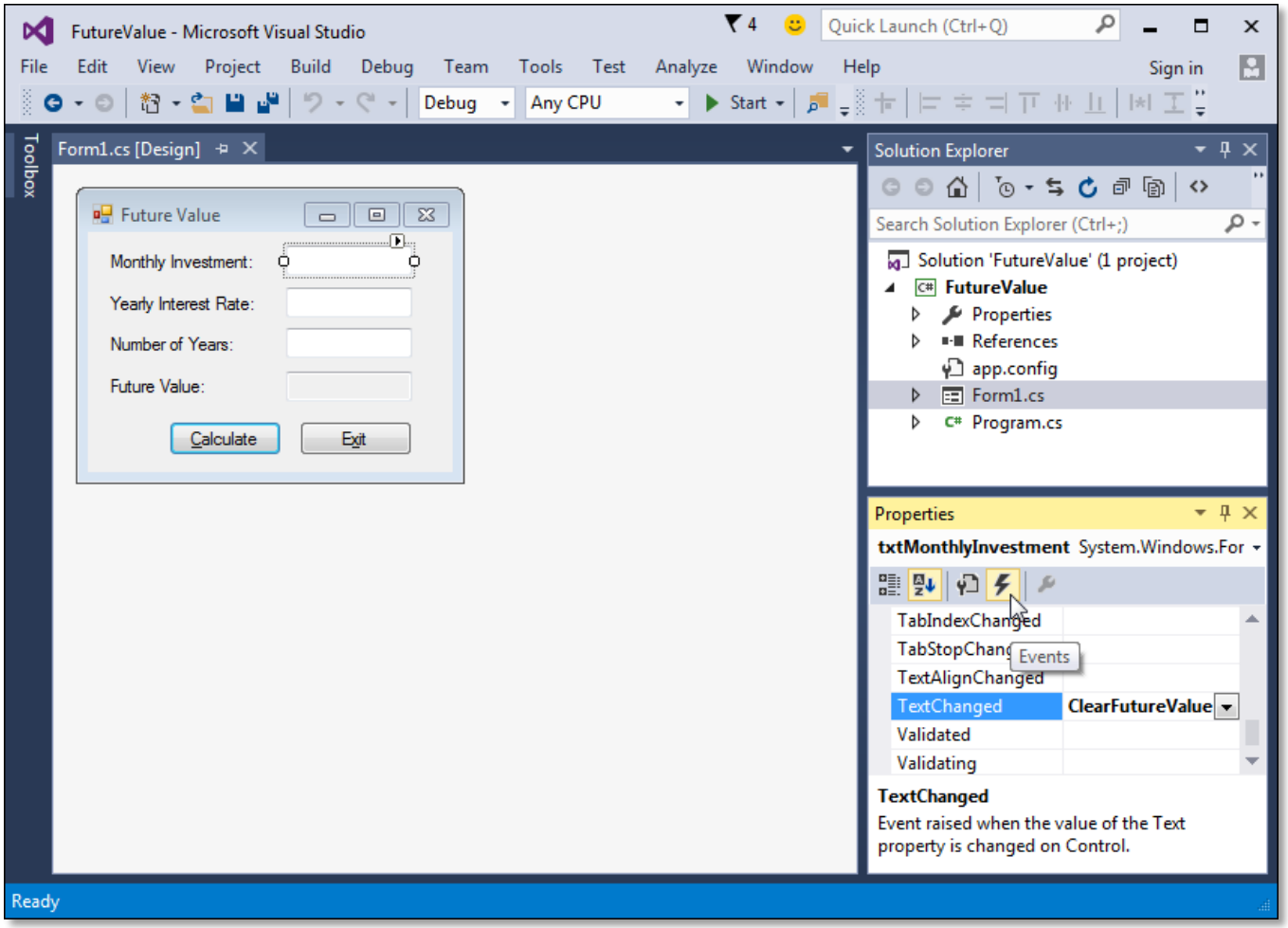
Code that works with the CalculateFutureValue method

```
decimal futureValue = 0m;
this.CalculateFutureValue(monthlyInvestment,
    monthlyInterestRate, months, ref futureValue);
txtFutureValue.Text = futureValue.ToString("c");
```

Code that works with a CalculateFutureValue method that uses the out keyword

```
decimal futureValue;
this.CalculateFutureValue(monthlyInvestment,
    monthlyInterestRate, months, out futureValue);
txtFutureValue.Text = futureValue.ToString("c");
```

The Events list for a text box control



The default event handler for a text box

The method declaration for the event handler

```
private void txtMonthlyInvestment_TextChanged(
object sender, EventArgs e)
{

}
```

The generated statement that wires the event to the event handler

```
this.txtMonthlyInvestment.TextChanged
    += new System.EventHandler(
       this.txtMonthlyInvestment_TextChanged);
```

A custom event handler for a text box

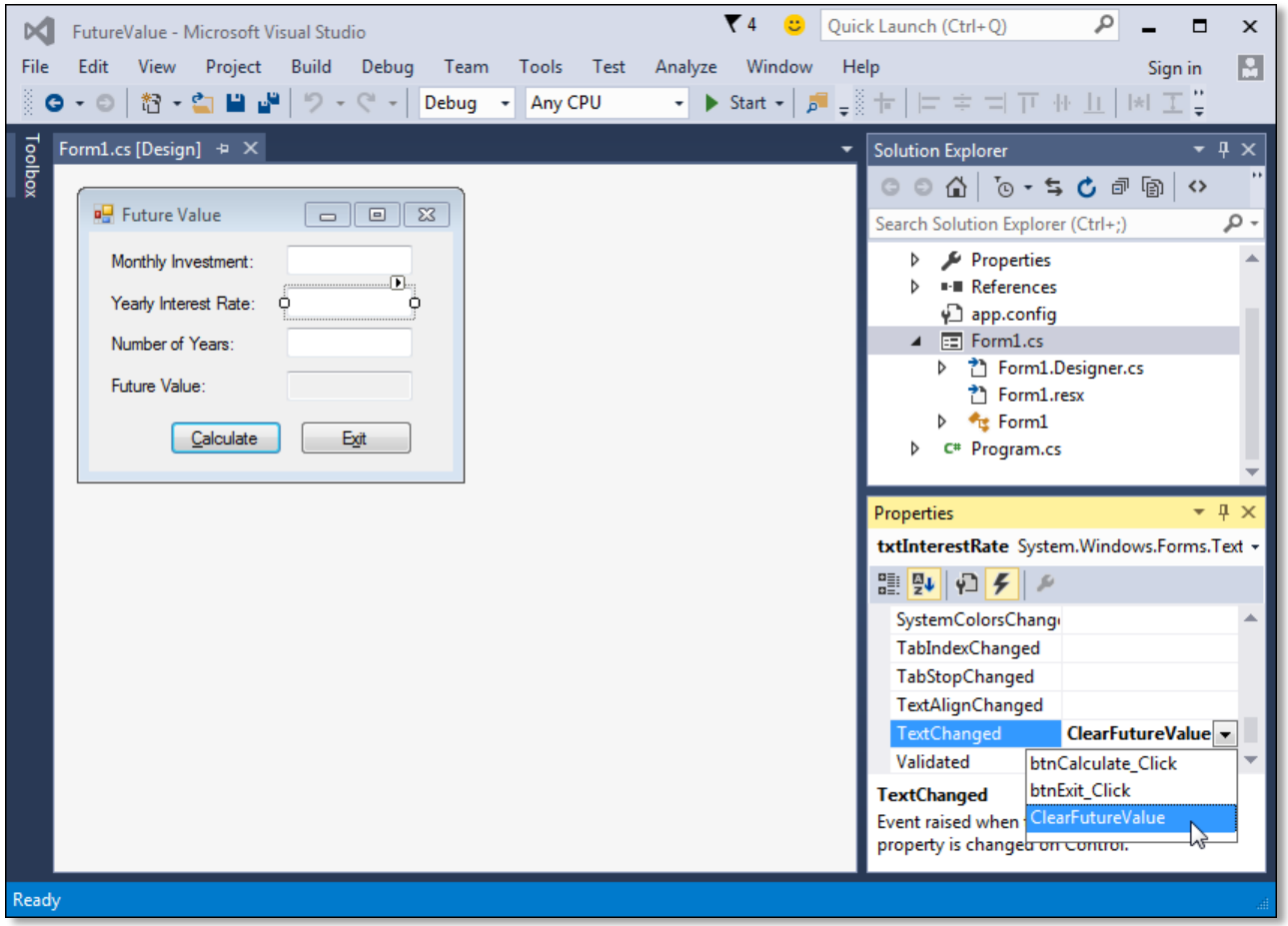
The method declaration for the event handler

```
private void ClearFutureValue(object sender, EventArgs e)
{

}
```

The generated statement that wires the event to the event handler

```
this.txtMonthlyInvestment.TextChanged +=
    new System.EventHandler(this.ClearFutureValue);
```

How to select an existing event handler for an event



How to wire an event to an existing event handler

1. Select the control in the Form Designer.
2. If necessary, click the Events button in the Properties window to display a list of the events for the control.
3. Click to the right of the event you want to handle and display the drop-down list.
4. Select the event handler you want to use for that event.

The code for the methods in the Future Value application

```
private void btnCalculate_Click(object sender, EventArgs e)
{
    decimal monthlyInvestment =
        Convert.ToDecimal(txtMonthlyInvestment.Text);
    decimal yearlyInterestRate =
        Convert.ToDecimal(txtInterestRate.Text);
    int years = Convert.ToInt32(txtYears.Text);

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

    decimal futureValue = this.CalculateFutureValue(
        monthlyInvestment, monthlyInterestRate, months);
    txtFutureValue.Text = futureValue.ToString("c");
    txtMonthlyInvestment.Focus();
}
```

The code for the methods in the Future Value application (cont.)

```
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;
}

private void btnExit_Click(object sender, EventArgs e)
{
    this.Close();
}

private void ClearFutureValue(object sender, EventArgs e)
{
    txtFutureValue.Text = "";
}
```

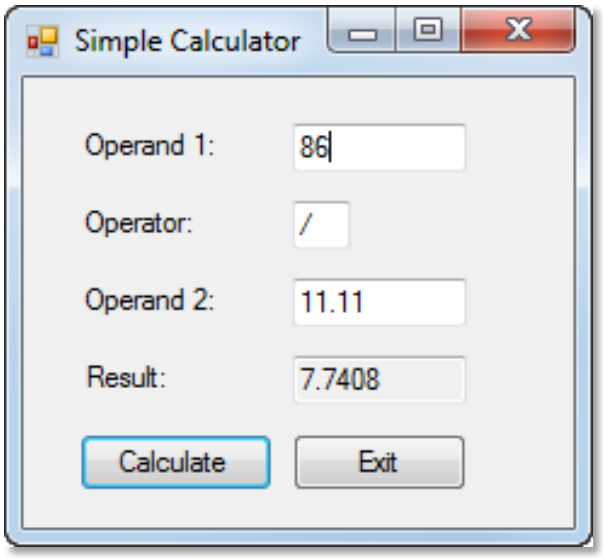
Some of the generated code in the Designer.cs file for the form

```
#region Windows Form Designer generated code

/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
    this.btnCalculate = new System.Windows.Forms.Button();
    this.txtInterestRate = new System.Windows.Forms.TextBox();
    this.txtMonthlyInvestment = new System.Windows.Forms.TextBox();
    ...
    //
    // btnCalculate
    //
    this.btnCalculate.Location = new System.Drawing.Point(54, 126);
    this.btnCalculate.Name = "btnCalculate";
    this.btnCalculate.Size = new System.Drawing.Size(75, 23);
    this.btnCalculate.TabIndex = 8;
    this.btnCalculate.Text = "&Calculate";
    this.btnCalculate.Click +=
        new System.EventHandler(this.btnCalculate_Click);
    //
    // txtInterestRate
    //
    this.txtInterestRate.Location = new System.Drawing.Point(132, 37);
```

Some of the generated code in the Designer.cs file for the form (cont.)

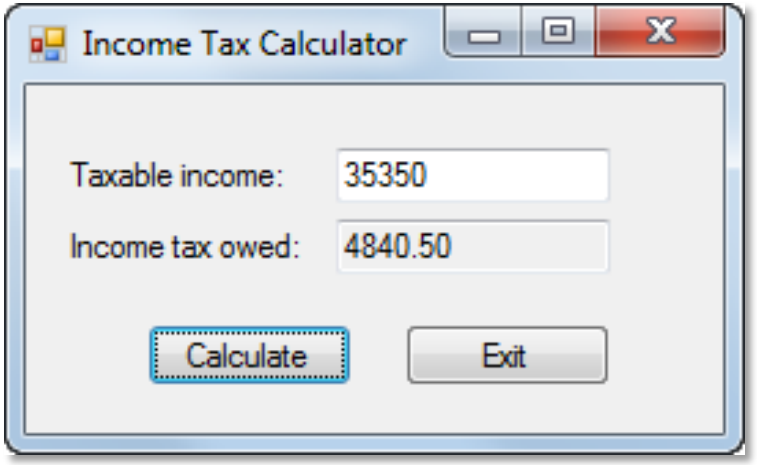
```
        this.txtInterestRate.Name = "txtInterestRate";
        this.txtInterestRate.Size = new System.Drawing.Size(84, 20);
        this.txtInterestRate.TabIndex = 3;
        this.txtInterestRate.TextChanged +=
            new System.EventHandler(this.ClearFutureValue);
        //
        // txtMonthlyInvestment
        //
        this.txtMonthlyInvestment.Location = new System.Drawing.Point(132, 9);
        this.txtMonthlyInvestment.Name = "txtMonthlyInvestment";
        this.txtMonthlyInvestment.Size = new System.Drawing.Size(84, 20);
        this.txtMonthlyInvestment.TabIndex = 1;
        this.txtMonthlyInvestment.TextChanged +=
            new System.EventHandler(this.ClearFutureValue);
        ...
}

#endregion

private System.Windows.Forms.Button btnCalculate;
private System.Windows.Forms.TextBox txtInterestRate;
private System.Windows.Forms.TextBox txtMonthlyInvestment;
...
```

Extra 6-1 Create a simple calculator



Perform an operation using two operands and an operator.

Extra 6-2 Add a method and an event handler to the income tax calculator



Add a method and another event handler to the income tax calculator of extra exercise 5-3.