

# VISUAL BASIC .NET

## A Study Guide

### BASIC CONCEPTS

Visual Basic .NET (VB .NET) is a modern object-oriented programming language designed by Microsoft for Rapid Application Development (RAD). Any discussion of object-oriented involves the use of some important terms and concepts:

#### *OOP - object-oriented programming*

OOP involves using an object-oriented language to create a program that contains one or more objects

#### *OOD - object-oriented design*

OOD is the design methodology used to plan object-oriented programs whereby a problem is divided into one or more objects.

#### *Object*

An object is Anything that can be seen or touched. An object has attributes that describe it. An object has behaviors that the object can either perform or have performed on it. A clock, a car, a book, a form, a text window, and a button are all examples of objects.

#### *Control*

A control is a special type of object used in web and windows forms. Examples include command buttons, text boxes, labels, and check boxes.

#### *Class*

A class is a pattern or blueprint for creating an object. An object is an instance of a class.

#### *Encapsulation*

Encapsulation, to enclosed in a capsule, combines an object's attributes and behaviors into one package—a class; a car manufacturer encapsulates a car's attributes and behaviors into a class—a blueprint.

#### *Abstraction*

Abstraction is hiding the internal details of an object to prevent the user from making inadvertent changes to the object; some attributes and behaviors are hidden, while others are exposed; a car's steering wheel is exposed, while the engine is hidden.

#### *Inheritance*

Inheritance allows you to create one class from another class. The new class is called the derived class and the original class is called the base class. A car manufacturer can create a blueprint of the next year's car from the current year's blueprint.

### *Methods*

Methods define an object's behaviors. They are the operations that an object can perform and can be performed on itself.

Large programs may be organized into smaller, more manageable pieces, called methods. Once a method is written, we need no longer concern ourselves with its internal details (abstraction). We use the method as a whole and concentrate on the overall task.

### *Properties*

Properties are an object's attributes. Properties are the data associated with an object.

Methods must be provided with data in order to do their job. The data provided to methods are object properties. Properties may include distance, direction, color, size, and much more.

### *Function*

A Function is a method that returns a value.

### *Events*

Events are signals by which an object can notify other objects that something noteworthy has occurred. Examples including clicking on an object, entering text, and opening a new window.

### *Members*

An object's properties, methods, and events can be referred to as its members.

### *Scope*

Every method has a scope. Methods that reference every object in an application have global scope and are referred to as global-level methods. Methods that define behaviors for a single object have local scope.

An object in VB .NET is a self-contained unit that combines code and data. A class contains the code that defines the characteristics of an object. An object is an instance of a class. The process of creating an object from a class is called instantiation. More than one object instance can be created from a single class. A new class can inherit the characteristics of a base class. If you instantiate two or more instances of the same class, all of the objects have the same properties, methods, and events.

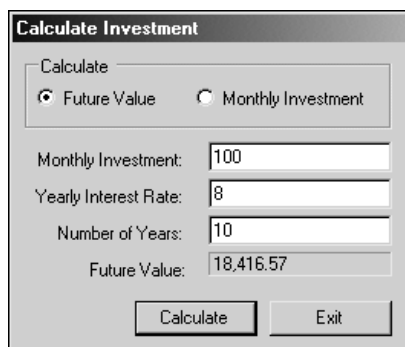
When we use the Alice Design Environment (DE) to design a world, the DE automatically generates code that creates objects based on classes from the class library. When we set up a world, we may view and set some of the properties of objects directly, prior to "playing" the world (running the program). We may view and set all of the properties for our objects during runtime using pre-written methods or we may write our own methods.

## VISUAL STUDIO AND THE .NET ENVIRONMENT

The **.NET framework** defines the environment in which we will be developing and executing Visual Basic .NET applications. Visual Studio .NET is a suite of products that includes these three programming languages; all run within the .NET framework.

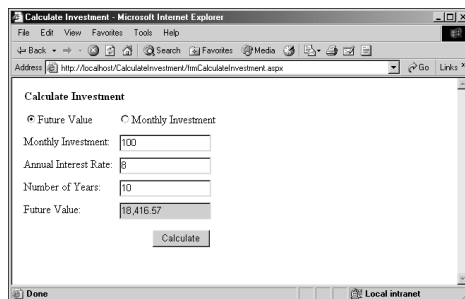
- **VB .NET** – Visual Basic .NET. Designed for Rapid Application Development (RAD)
- **Visual C# .NET** – Combines features of Java and C++ for RAD
- **Visual C++ .NET** – MS version of C++, a full featured language

Other languages have been developed for use within the .NET framework, but they are not part of Visual Studio. Visual Studio includes the Integrated Design Environment (IDE), a powerful tool used for developing applications in the .NET languages. Visual Studio .NET also comes with a version of MS SQL Server that runs on your workstation.



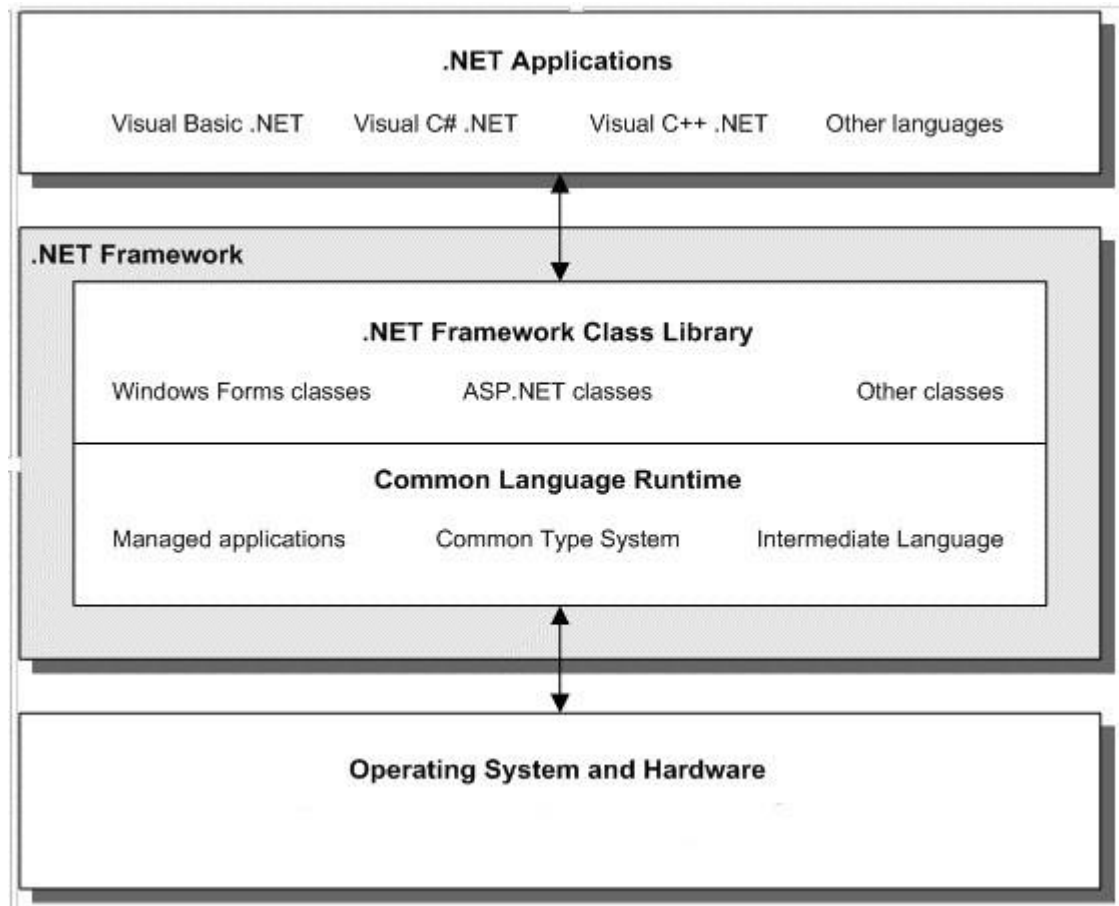
The screenshot shows a Windows Forms application titled "Calculate Investment". It features a "Calculate" section with two radio buttons: "Future Value" (selected) and "Monthly Investment". Below these are four text boxes: "Monthly Investment" (containing 100), "Yearly Interest Rate" (containing 8), "Number of Years" (containing 10), and "Future Value" (containing 18,416.57). At the bottom are "Calculate" and "Exit" buttons.

VB.NET is typically used to develop two types of applications. First, is a **Windows Forms application**. A Windows Forms application runs on the user's PC and consists of one or more Windows forms. These forms provide the GUI for the application. Each Windows form can contain controls like labels, text boxes, command buttons, radio buttons, and check boxes, and others.



The screenshot shows a Web Forms application running in Microsoft Internet Explorer. The browser window title is "Calculate Investment - Microsoft Internet Explorer". The address bar shows "http://localhost/CalculateInvestment/IntCalculateInvestment.aspx". The web form content is identical to the Windows Forms application, with the "Future Value" radio button selected and the "Calculate" button visible.

The other type of application is a **Web Forms application**. A web forms application consists of one or more web forms that provide the user interface. A Web Form runs in a web browser.



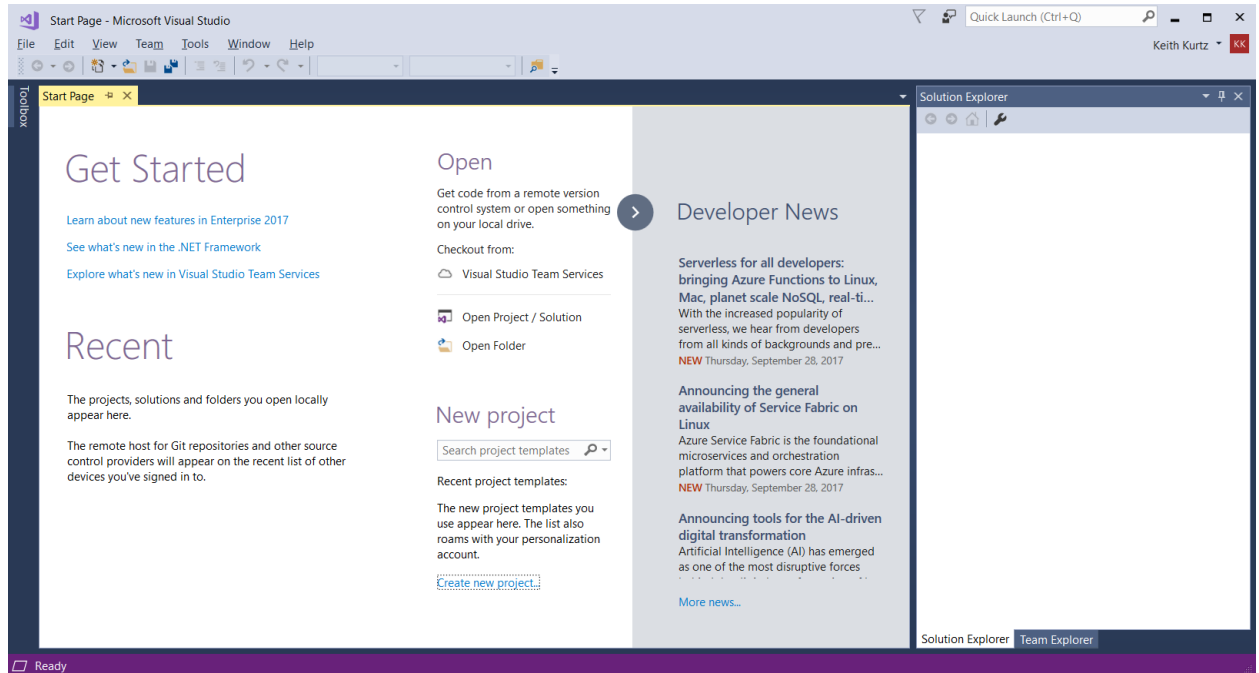
.NET applications do not access the operating system directly. They use the services of the .NET framework, which serves as an interface to the Operating System. The .NET framework includes the .NET Framework Class Library and the Common Language Runtime, or CLR. The .NET Framework Class Library contains hundreds of classes, or pre-written code that can be used by any of the .NET programming languages. The CLR is essential to running programs within the .NET framework. The CLR manages memory allocation, code execution and conversion, etc. The CLR also provides for the basic data types. This enables programmers to write in different languages while assuring compatibility between them.

### Projects and Solutions

Before we do anything else, we should distinguish between **Projects** and **Solutions**. A project is a container that holds Visual Basic source files and other files needed to create an assembly. A project may contain several files, but they all get compiled together into a single assembly. There is a .vbproj file that is used to keep track of all the files in a project. A Solution is a container that holds one or more projects. If there is only one project in a solution, there is not much to distinguish one from the other.

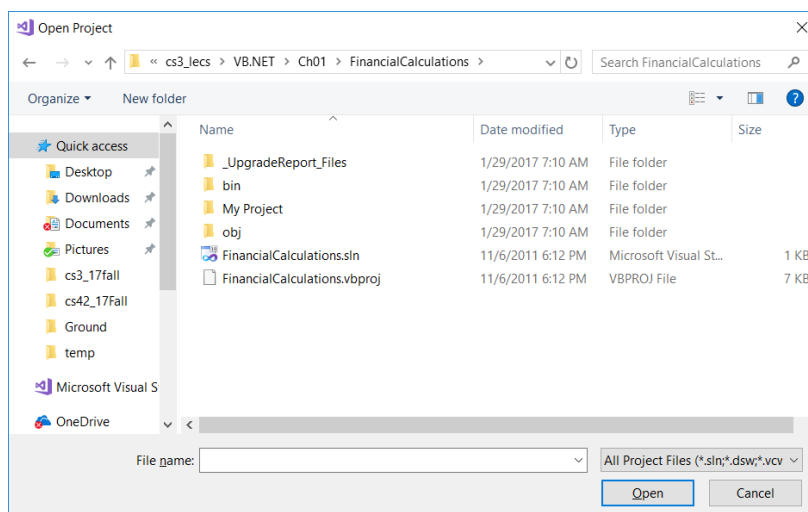
Multi-project solutions are useful if a group of programmers is working together, or if multiple languages are to be used. There is a .sln file that is used to keep track of all the projects in a solution.

## The Visual Studio Integrated Design Environment (IDE)



### THE VISUAL STUDIO START PAGE

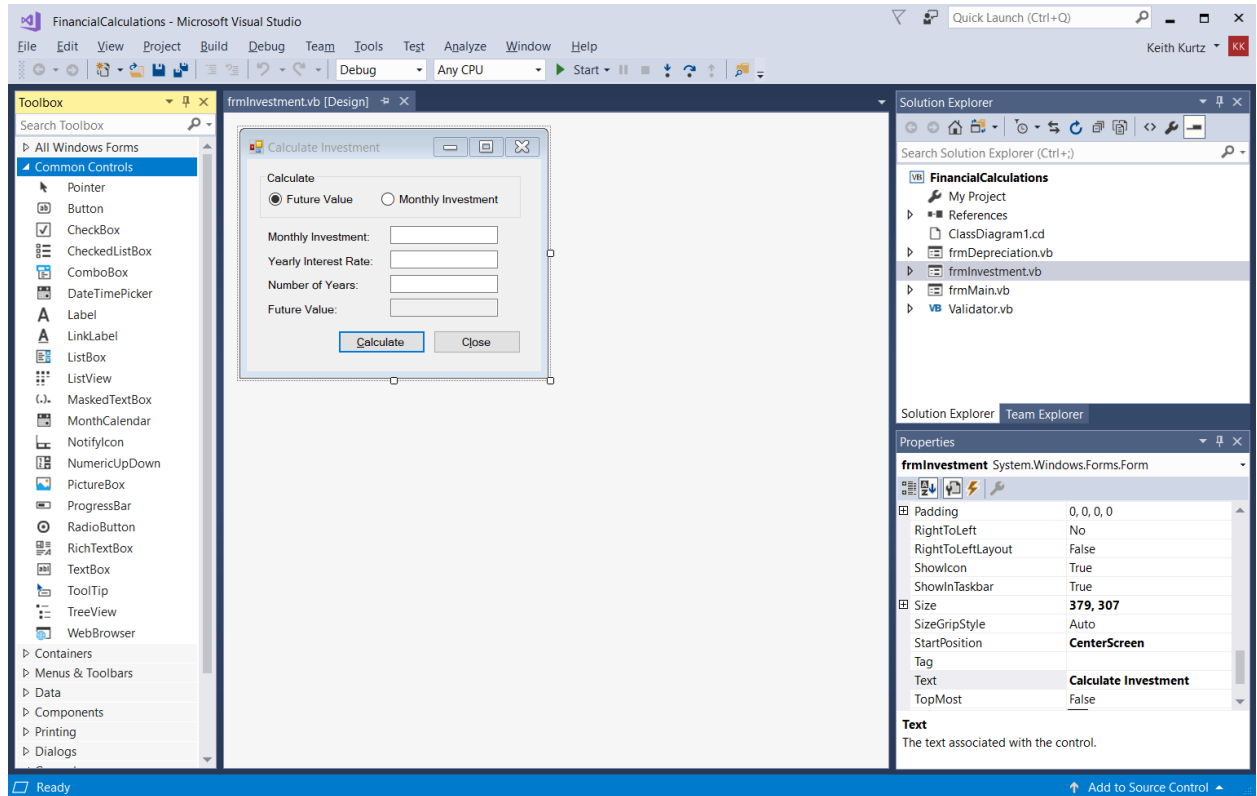
The Visual Studio start page is actually the home page of a built-in web browser. The browser is used here to display information and help pages. It uses hyperlinks, and other web objects that you are probably quite familiar with.



### THE OPEN PROJECT DIALOG BOX

To open a project, Click File... Open Project on the menu bar. Use the controls in the Open Project dialog box to locate and select the project or solution you want to open. In this case, we have opened the Financial Calculations project.

To close a project, you may either close Visual Studio or use the File... Close Project command.



### THE VISUAL STUDIO IDE WITH FRMINVESTMENT OPEN.

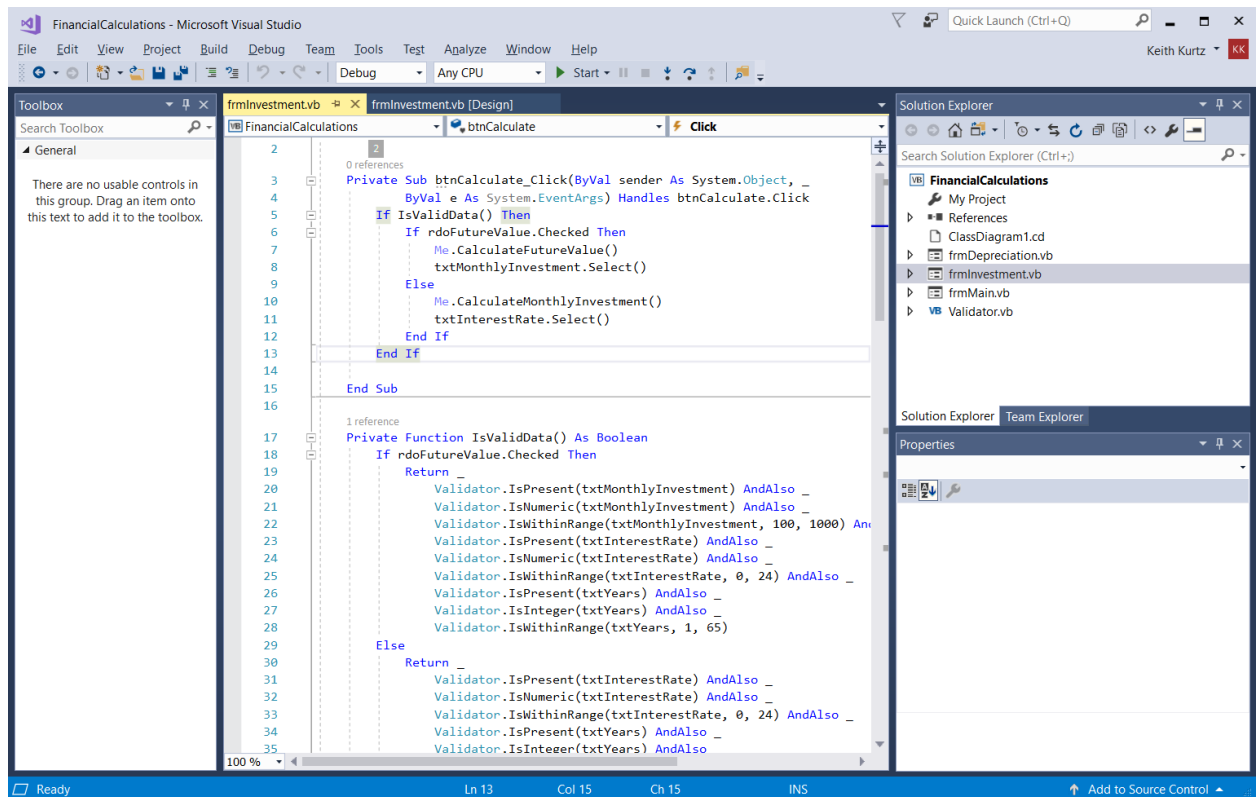
The main part of the IDE contains a number of tabbed windows. We use the Form Designer window to develop forms.

You may need to open the toolbox. Clicking on the tack will anchor it in place. The toolbox contains a variety of items that are mostly used to add controls to our forms.

The properties window is used to display and to modify the attributes, called properties, of every control in our form.

The Solution Explorer is used to manage the various project files.

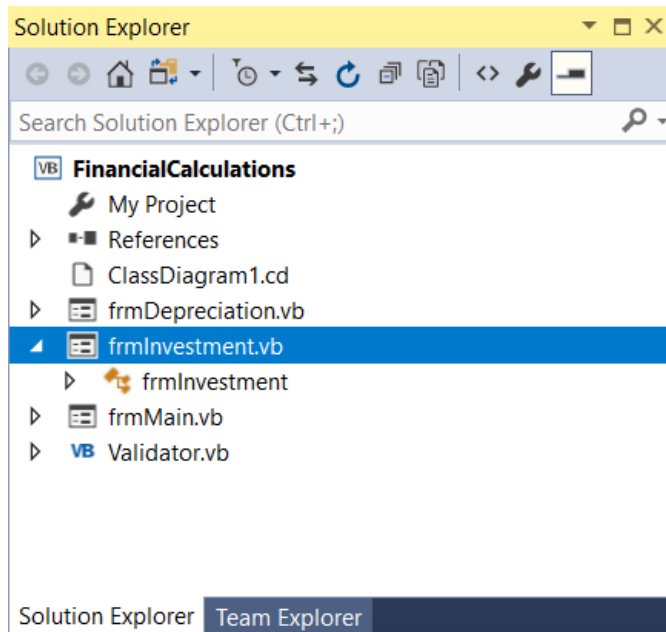
At the top of the screen is the title bar, the menu bar, and the standard toolbar. Various other toolbars may appear, as we need them.



## A CODE MODULE OPEN IN THE VISUAL STUDIO IDE.

Double-clicking on the `frmInvestment` module immediately below `frmInvestment.vb` in the Solution Explorer will open a code window with the code behind the form. It is a text editor with certain features meant to assist us in writing syntactically correct code.

Note that these two windows do not represent different files. They provide two views of the same VB source file.



The Solution Explorer is used to help keep track of all the files that comprise a .NET solution. You can use the (+) and (-) buttons to expand and collapse groups.

The VB source files all end with .vb. Form files and code files have different icons, but end with .vb. You may find it useful to use *frm* as the first three letters of each form file.

You will find that, in creating a .NET project in Visual Studio, a great many files are created. Be sure to keep all of these files together. They are necessary to proper management of your application.

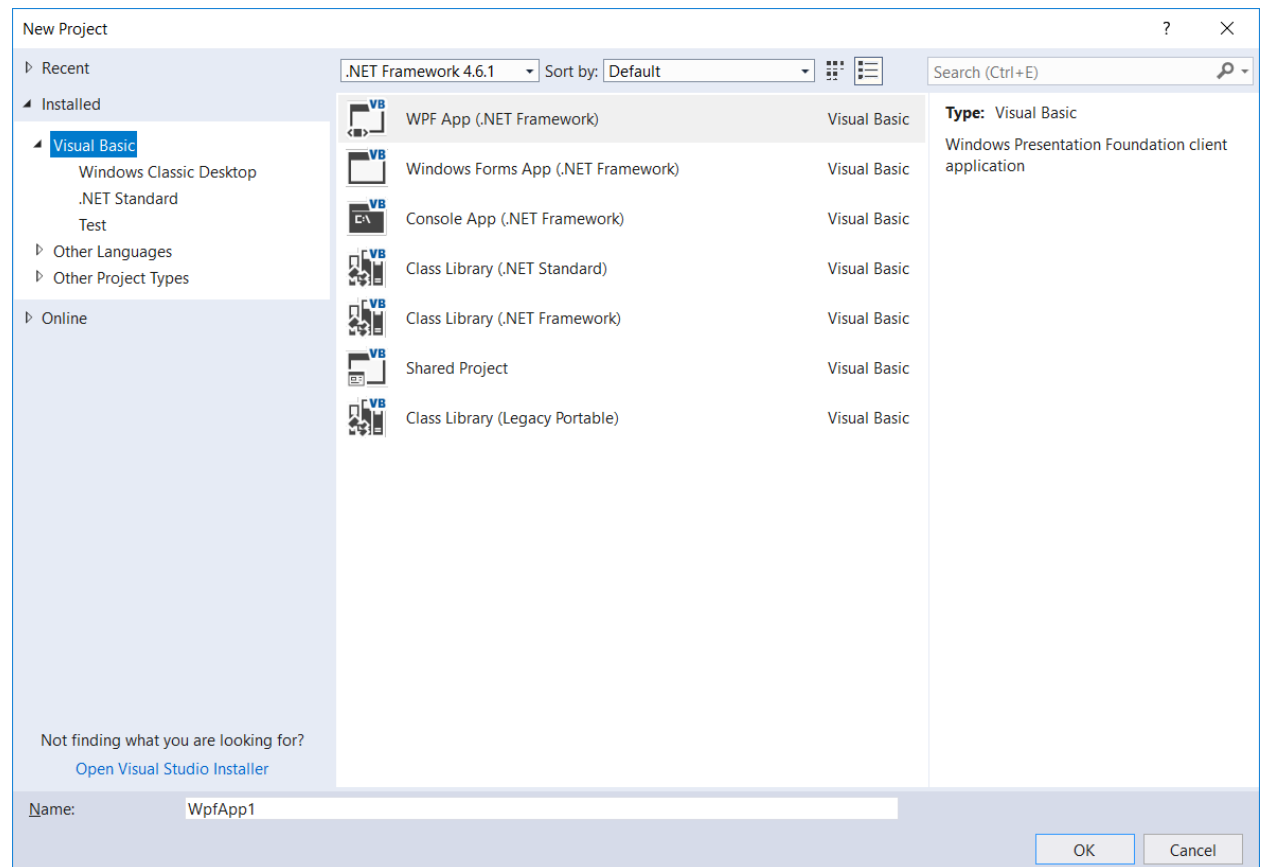
To run (debug) an application within the Visual Studio IDE, do one of the following:

- Click on the Start button in the toolbar
- Click on the Debug tab, then on Start Debugging
- Press F5



## CREATING PROJECTS

### The Invoice Total Form

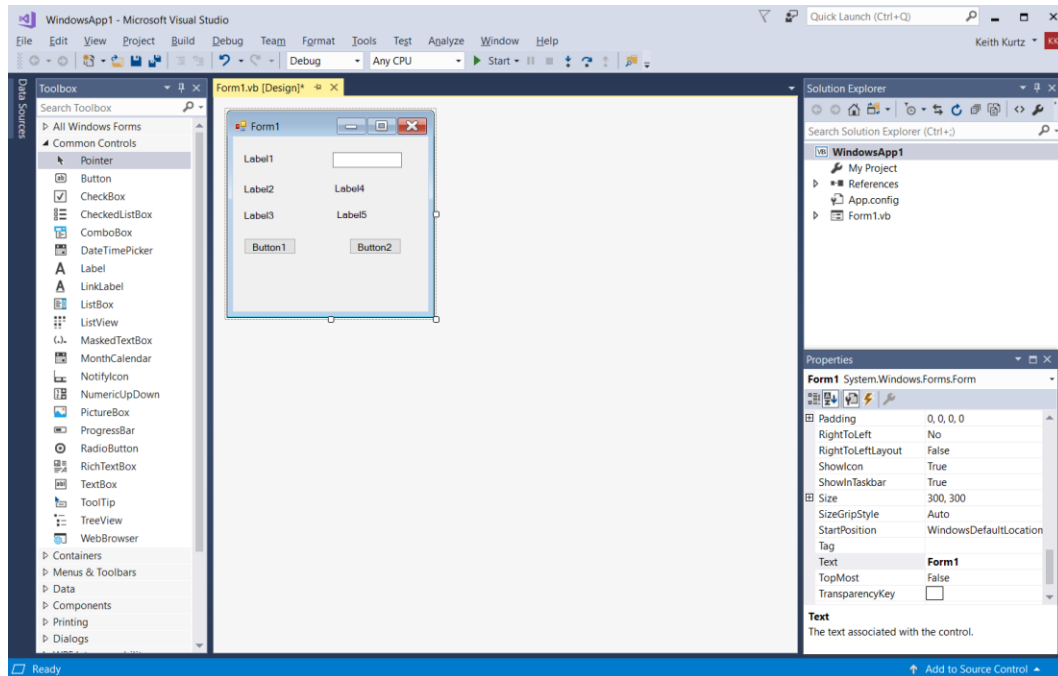


#### *The New Project dialog box.*

To create a new project in Visual Studio, Click on File...New Project. The New Project dialog box will open.

Be sure that Visual Basic is selected in the left-hand pane and Windows Forms Application is selected in the center pane. Type the name of the application, "InvoiceTotal" in this case, into the Name textbox at the bottom.

NOTE: You can control where your projects are saved by clicking Tools...Options and selecting Projects and Solutions, General in the left-hand pane of the dialog box that opens.



### One way to add a control to a form

- Select the control in the Toolbox.
- Click in the form where you want to place the control and drag the pointer on the form to size the control.

### Two other ways to add a control to a form

- Double-click on the control in the Toolbox.
- Drag the control from the Toolbox and drop it onto the form.

### How to select and work with controls

- To select a control on the form, click on it.

- To move a control, drag it.
- To size a selected control, drag one of its handles.

### How to select and move more than one control

- Hold down the Ctrl key as you click on each control.
- Click on a blank spot in the form and then drag around the controls.

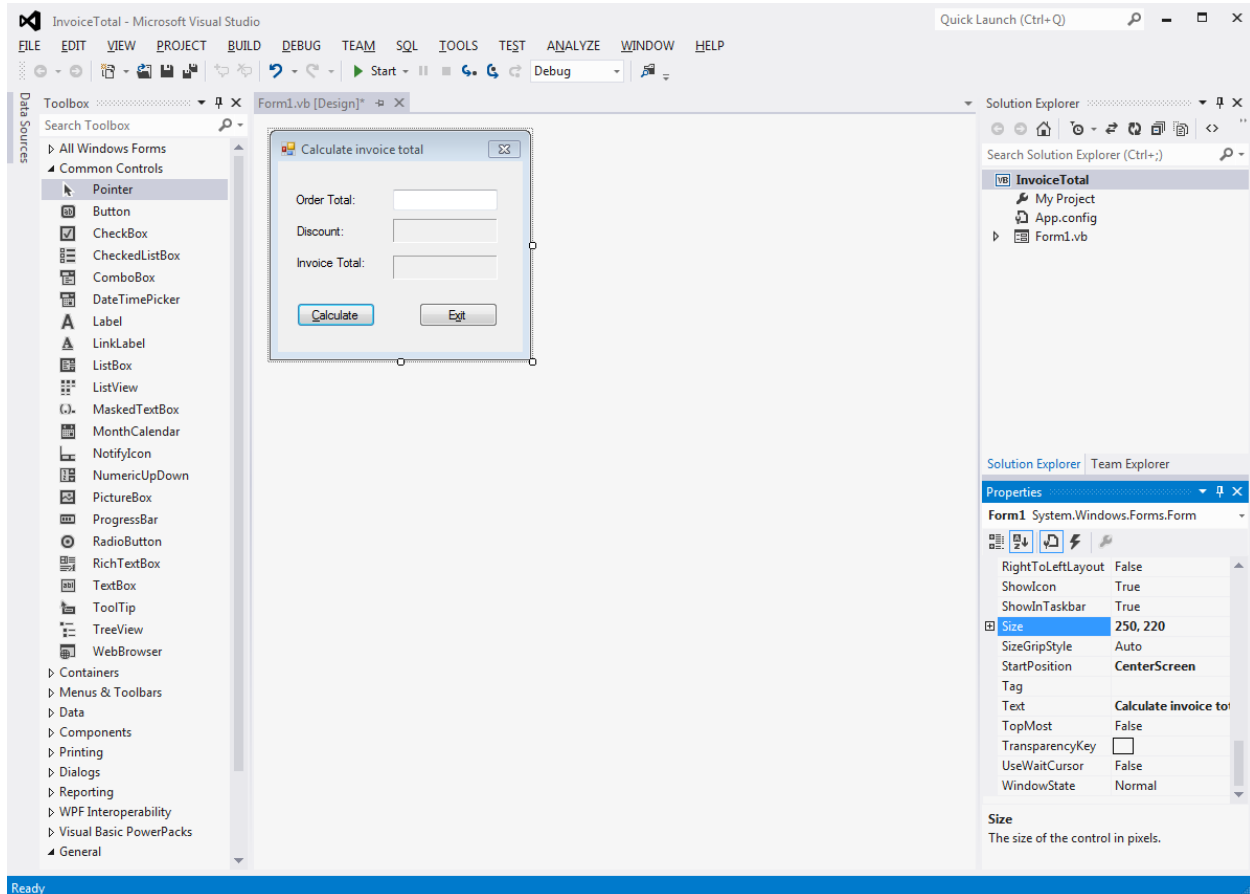
### How to change the size of the form

- Click on the form.
- Drag one of its handles.

A **label** is used by the application to display text. Users may not enter text into a label.

A **textbox** is used when a user must enter text into the application. The application may also write text to a textbox.

A **Button** is used to execute a command.



To create the form shown, select each control in turn and enter its properties in the Properties Window:

**Place the controls on the form and configure as follows:**

Default name	Property	Setting	Property	Setting
Label1	Text	Order total:	Location	15, 30
	TextAlign	MiddleRight	AutoSize	True
	TabIndex	0		
Label2	Text	Discount:	Location	15,60
	TextAlign	MiddleRight	AutoSize	True
	TabIndex	0		
Label3	Text	Invoice total:	Location	15,90
	TextAlign	MiddleRight	AutoSize	True
	TabIndex	0		

TextBox1	Name	txtOrderTotal	Location	110,27
	Text	(empty)	Size	100,22
	TabIndex	1		
Label4	Name	lblDiscountAmount	Location	110,55
	Text	(empty)	AutoSize	False
	TextAlign	MiddleLeft		
	TabIndex	0		
	BorderStyle	Fixed3D		
Label5	Name	lblInvoiceTotal	Location	110,90
	Text	(empty)	AutoSize	False
	TextAlign	MiddleLeft		
	TabIndex	0		
	BorderStyle	Fixed3D		
Button1	Name	btnCalculate	Location	8,135
	Text	&Calculate	Size	75,23
	TabIndex	2		
Button2	Name	btnExit	Location	135,135
	Text	E&xit	Size	75,23
	TabIndex	3		

#### The property settings for the form

Property	Setting
FormBorderStyle	FixedSingle
MaximizeBox	False
MinimizeBox	False
StartPosition	CenterScreen
Text	Calculate invoice total
AcceptButton	btnCalculate <i>(when Enter is pressed)</i>
CancelButton	btnExit <i>(when Esc is pressed)</i>
Size	250,220

When we click on a button, a *Click Event* occurs. We need to write the code for the click event that determines what actions are to occur when we click on each button. In the case of the Calculate Button, we want the application to apply the discount rate, display it in a label, calculate the Invoice Total, and display this in a different label. The following pseudocode may be used to plan the *Calculate Button Click Event Procedure*.

```
Declare variables
    decOrderTotal
    decDiscountAmount
    declInvoiceTotal

decOrderTotal = amount entered in txtOrderTotal
decDiscountAmount = decOrderTotal * 0.2
declInvoiceTotal = decOrderTotal - decDiscountAmount

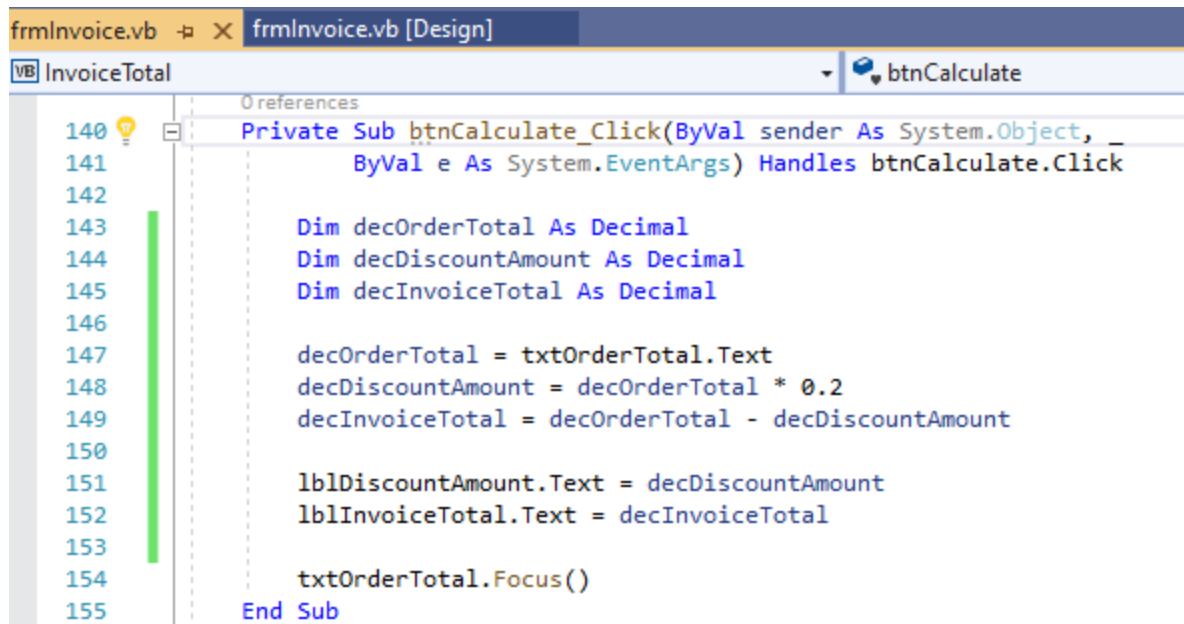
Display decDiscountAmount, declInvoiceTotal
Move focus to txtOrderTotal
```

The last step is used to move the cursor to the Textbox so that the application is ready for the next calculation.

You can open the code editor for a button's click event procedure by double-clicking on the button. This will open the code editor for the form and add the necessary syntax for a click event procedure for that button.

We can use the built-in *Intellisense* feature of Visual Studio to enter member names in the code editor:

- Type the object name followed by a *dot* to display a list of the available members for an object.
- Type the first few letters of the member name, so the Code Editor selects the first entry in the list that matches those letters.
- Once the correct member name is selected, press the Tab key to insert the member into your code.



```
frmInvoice.vb [Design]
VB InvoiceTotal btnCalculate
0 references
140 Private Sub btnCalculate_Click(ByVal sender As System.Object, _
141     ByVal e As System.EventArgs) Handles btnCalculate.Click
142
143     Dim decOrderTotal As Decimal
144     Dim decDiscountAmount As Decimal
145     Dim decInvoiceTotal As Decimal
146
147     decOrderTotal = txtOrderTotal.Text
148     decDiscountAmount = decOrderTotal * 0.2
149     decInvoiceTotal = decOrderTotal - decDiscountAmount
150
151     lblDiscountAmount.Text = decDiscountAmount
152     lblInvoiceTotal.Text = decInvoiceTotal
153
154     txtOrderTotal.Focus()
155 End Sub
```

Double-click on a button to open a code window and begin work on its click-event procedure.

The code for the Calculate Button Click Event Procedure is shown above.

The first three lines declare the three variables `decOrderTotal`, `decDiscountAmount`, and `decInvoiceTotal` as type decimal. Decimal variables are useful for working with dollar amounts. The prefix “dec” in each name is solely for use by the programmer as a reminder that this member is a variable of type decimal.

The next line of code stores the `Text` property of the text box in the `decOrderTotal` variable. The text property contains any text that is shown in the textbox and has presumably been entered by the user. This is an *assignment statement*. In an assignment statement, the expression to the right of the equals sign is evaluated and the result is stored in the variable on the left.

The next lines calculates the discount which is 20% (0.2) and stores the discount in the `decDiscountAmount` variable.

The next line of code calculates Invoice Total amount and stores the result in the `decInvoiceTotal` variable.

The next two lines display the discount amount and invoice total in their respective labels. By writing to the text property of a label, we cause text to appear.

Finally, the *Focus method* of the `txtOrderTotal` textbox is invoked, causing the cursor to move into the textbox.

```
End Sub

Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
    Me.Close()
End Sub
End Class
```

The click event procedure for the Exit button does nothing more than stop the application from executing. One of the easiest ways to do this is to close the form. We can use the *Close method* for this. The code above uses the alias for the current form, *Me*.

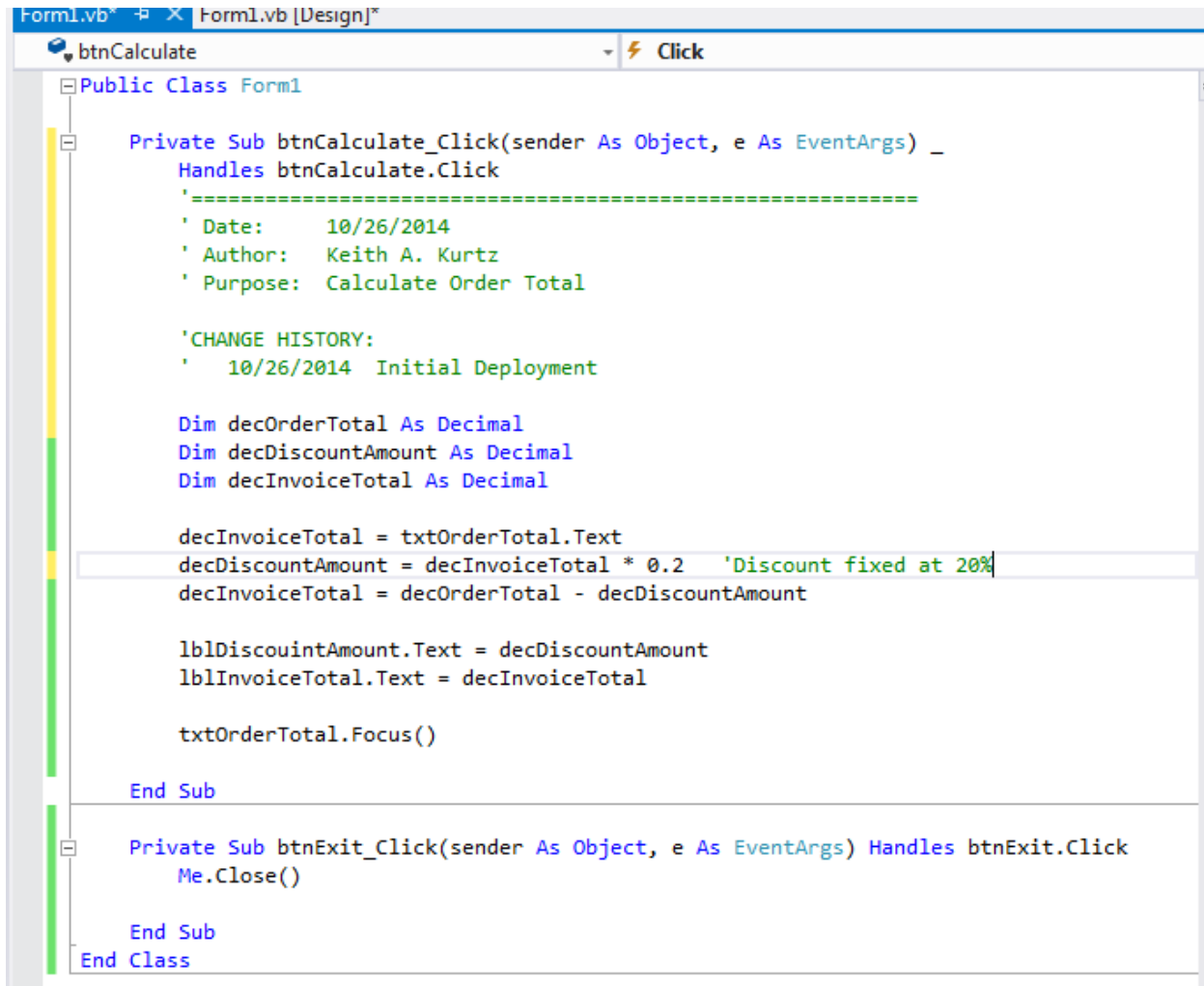
There are three ways to run a project inside of Visual Studio:

- Click the Start button located in the toolbar
- Press the F5 key
- Select Debug...Start Debugging on the menu

To save your project, select File...Save All. You may be prompted for a location to save your project.

## WORKING IN THE CODE EDITOR AND USING VARIABLES

Examine the following click event procedure:



```
Form1.vb*  Form1.vb [Design]*
btnCalculate Click
Public Class Form1
    Private Sub btnCalculate_Click(sender As Object, e As EventArgs) _
        Handles btnCalculate.Click
        '=====
        ' Date:      10/26/2014
        ' Author:    Keith A. Kurtz
        ' Purpose:   Calculate Order Total

        'CHANGE HISTORY:
        '  10/26/2014  Initial Deployment

        Dim decOrderTotal As Decimal
        Dim decDiscountAmount As Decimal
        Dim decInvoiceTotal As Decimal

        decInvoiceTotal = txtOrderTotal.Text
        decDiscountAmount = decInvoiceTotal * 0.2 'Discount fixed at 20%
        decInvoiceTotal = decOrderTotal - decDiscountAmount

        lblDiscountAmount.Text = decDiscountAmount
        lblInvoiceTotal.Text = decInvoiceTotal

        txtOrderTotal.Focus()

    End Sub

    Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
        Me.Close()

    End Sub
End Class
```

Note that first line is quite long and actually continues across two lines. This is done with the use of a *continuation character*, a space followed by the underscore. Without the continuation character, the line will be difficult to read as it may continue beyond the limits of the code editor window and require horizontal scrolling.

Comments are preceded by an apostrophe. Comments are ignored by the Visual Basic compiler and interpreter. They are for the use of the programmer and serve to help document the code. Comments may exist on their own line or follow code on the same line. Nothing may follow a comment on the same line.



## Data Types

The following data types are available in Visual Basic. You should always declare your variables with a data type.

Data type	Prefixes	Description
String	str or s	Any number of characters
Char	chr or c	A single character
Boolean	bln or b	A True or False value
Date	dtm	An integer that represents the date
Byte	byt or y	A positive integer value from 0 to 255
Short	srt or t	An integer from –32,768 to +32,767
Integer	int or i	Larger integer
Long	lng or l	Still larger integer
Decimal	dec or d	A number with up to 28 significant digits (integer and fraction)
Single	sng or f	A single-precision, floating-point number
Double	dbl or p	A double-precision, floating-point number
Object	obj or o	An address that refers to an object

When you declare a variable, you may choose to initialize the variable with a value, or allow Visual Basic to initialize your variable with a default value.

Data type	Default Initial Value
All numeric types	Zero (0)
Boolean	False
Char	Binary 0
String or Object	Nothing (it has no value)
Date	12:00 a.m., January 1, 0001

### *Basic Syntax for declaring and initializing variables*

**Dim**|**Private**|**Public**|**Static** variablename [**As** type] [= expression]

### *Typical variable declarations*

```
Dim sErrorMessage As String
Dim iIndex As Integer = 1
Dim tMonth As Short, dRate As Decimal
Dim iStatus, iRunningValue As Integer
Private bAddMode As Boolean = True
Public iUserStatus As Integer
Static iRunningValue As Integer
```

### Variable naming recommendations

- Start each name with its data type prefix in lowercase letters.
- Use camelcase to make the names easier to read.
- Assign meaningful names that are easy to remember.

### The syntax of an assignment statement

variablename = expression

### Typical assignment statements

```
iMonth = 1
iMonth = iMonth + 1
dDiscountAmount = dOrderTotal * .2
dInvoicetotal = dOrderTotal + dDiscountAmount
dChangePercent = _
    (dThisYTDSales - dLastYTDSales) / dLastYTDSales * 100
dArea = (dRadius ^ 2) * 3.1416
iMonth = 1
iMonth = iMonth + 1
dDiscountAmount = dOrderTotal * .2
dInvoicetotal = dOrderTotal + dDiscountAmount
dChangePercent = _
    (dThisYTDSales - dLastYTDSales) / dLastYTDSales * 100
dArea = (dRadius ^ 2) * 3.1416
```

### Arithmetic operators

Operator	Name
+	Addition
-	Subtraction
*	Multiplication
/	Division
\	Integer division
Mod	Modulo
^	Exponentiation
-	Negative sign

Other assignment operators

**Assume i = 13**

Operator	Example	Description	Result
+=	i += 5	i = i + 5	i = 18
-=	i -= 6	i = i - 6	i = 7
*=	i *= 2	i = i * 2	i = 26
/=	i /= 2	i = i / 2	i = 6
\=	i \= 3	i = i \ 3	i = 4
^=	i ^= 2	i = i ^ 2	i = 169

### *Order of precedence for arithmetic operations*

1. Exponentiation
2. Negative sign
3. Multiplication, division, integer division, and modulo
4. Addition and subtraction

Parentheses override the order of precedence. The operations in the innermost sets of parentheses are executed first.

Examples of arithmetic expressions

```
iVarX = 14          'assume for all examples
iVarY = 8           'assume for all examples
dVarA = 8.5         'assume for all examples
dVarB = 3.4         'assume for all examples
```

```
iResult = iVarX \ iVarY      'result = 1
iResult = iVarX mod iVarY    'result = 6
dResult = dVarA / dVarB      'result = 2.5
dResult = dVarA mod dVarB    'result = 1.7
iResult = -iVarY             'result = -8
iResult = iVarY + iVarX      'result = 6
iResult = iVarX \ iVarY      'result = 1
iResult = iVarX mod iVarY    'result = 6
dResult = dVarA / dVarB      'result = 2.5
dResult = dVarA mod dVarB    'result = 1.7
iResult = -iVarY             'result = -8
iResult = -iVarY + iVarX     'result = 6
```

```
iVarX = 10
iVarY = 5
```

```
iResultA = iVarX + iVarY * 5    'iResultA is 35
iResultB = (iVarX + iVarY) * 5  'iResultB is 75
dResultA = iVarX - iVarY * 5    'dResultA is -15
dResultB = (iVarX - iVarY) * 5  'dResultB is 25
```

## Casting

When different data types are combined in an expression, they must be *cast* into a common data type. This is usually the *broadest*, or most inclusive of the data types being used.

```
dVarA = 5.0
```

```
iVarB = 7
```

```
iVarC = 9
```

```
dResult = (dVarA + iVarB + iVarC) / 4    'dResult is 5.25
```

```
dVarA = iVarC                            'dVarA is 9.0
```

In the above examples, the integers are cast into type decimal. Type decimal is a broader data type because it includes all of the values available to an integer and more.

## Dates and Strings

### *Simple assignment statements for dates*

```
dtmStartDate = #June 1, 2001#
```

```
dtmStartDate = "June 1, 2001"
```

```
dtmStartDate = #6/1/2001#
```

### *How to concatenate character strings*

```
sFirstName = "Bob"
```

```
sLastName = "Smith"
```

```
sFullName = sFirstName & " " & sLastName
```

### *How to append one string to another string*

```
sFirstName = "Bob"
```

```
sLastName = "Smith"
```

```
sName = sFirstName & " "
```

```
sName = sName & sLastName
```

### *How to append with the &= operator*

```
sFirstName = "Bob"
```

```
sLastName = "Smith"
```

```
sName = sFirstName & " "
```

```
sName &= sLastName
```

## THE SELECTION STRUCTURE

The Selection Structure is used to make a decision or comparison and then, based on the result of that decision or comparison, to select one of two paths. The condition must result in either a true (yes) or false (no) answer. If the condition is true, the program performs one set of tasks. If the condition is false, there may or may not be a different set of tasks to perform.

The following pseudocode illustrates these two possibilities:

```
If condition is true Then
    perform these tasks
End If
Perform these tasks
whether condition is true
or false
```

```
If condition is true then
    perform these tasks
Else
    perform these tasks
End If
Perform these tasks
whether condition is true
or false
```

### *The syntax of the block If statement*

```
If condition Then
    statements
[ElseIf condition-n Then
    statements] ...
[Else
    statements]
End If
```

### *The syntax of the one-line If statement*

```
If condition Then statements [Else statements]
```

## Relational Operators

We use relational operators to create the condition used in an If statement. The result of a relational expression is always Boolean. That is, the result may only be True or False.

=	Equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
<>	Not equal to

These operators are evaluated from left to right, and are evaluated after any mathematical operators.

### *Expressions Containing Relational Operators*

$$10 + 3 < 5 * 2$$

- $5 * 2$  is evaluated first, giving 10
- $10 + 3$  is evaluated second, giving 13
- $13 < 10$  is evaluated last, giving false

$$7 > 3 * 4 / 2$$

- $3 * 4$  is evaluated first, giving 12
- $12 / 2$  is evaluated second, giving 6
- $7 > 6$  is evaluated last, giving true

### *Using Relational Operators in the condition*

A condition that checks if the value stored in the `intNum` variable is greater than 123

`intNum > 123`

A condition that checks if the value stored in the `strName` variable is "Mary Smith"

`UCase(strName) = "MARY SMITH"`

The second example raises a common issue. When users enter "Mary Smith," they type "Mary Smith," "mary smith," "MARY SMITH," or something else. Our intent is to check for the string M-A-R-Y-SPACE-S-M-I-T-H in upper or lower case. The `UCase` function converts all characters in a string to upper case, eliminating the problem. This function is used when we wish to ignore case. If case is important, then it should not be used.

### *Compound Logic*

Sometimes we wish to evaluate more than one condition in a single statement. The various conditions are connected logically using Not, And, or Or.

### *Logical Operators*

- |     |  |
|-----|--|
| Not | Reverses the truth value of the condition; false becomes true and true becomes false                           |
| And | All conditions connected by the And operator must be true for the compound condition to be true                |
| Or  | Only one of the conditions connected by the Or operator needs to be true for the compound condition to be true |

These operators are evaluated after any mathematical and relational operators. The order of precedence is Not, And, Or.

*Truth Table for Not Operator*

A	Not A
F	T
T	F

*Truth Table for And Operator*

A	B	A And B
F	F	F
F	T	F
T	F	F
T	T	T

*Truth Table for Or Operator*

A	B	A Or B
F	F	F
F	T	T
T	F	T
T	T	T

*Expressions Containing the And Logical Operator*

3 > 2 And 6 > 5

- 3 > 2 is evaluated first, giving true
- 6 > 5 is evaluated second, giving true
- true And true is evaluated last, giving true

10 < 25 And 6 > 5 + 1

- 5 + 1 is evaluated first, giving 6
- 10 < 25 is evaluated second, giving true
- 6 > 6 is evaluated third, giving false
- true And false is evaluated last, giving false

*Expression Containing the Or Logical Operator*

8 = 4 \* 2 Or 7 < 5

- 4 \* 2 is evaluated first, giving 8
- 8 = 8 is evaluated second, giving true
- 7 > 5 is evaluated third, giving false
- true Or false is evaluated last, giving true

*Optimization in Visual Basic*

If you use the And operator to combine two conditions, Visual Basic does not evaluate the second condition if the first condition is false. If you use the Or operator to combine two conditions, Visual Basic does not evaluate the second condition if the first condition is true.

*Order of precedence for conditional expressions*

1. All arithmetic operations
2. Relational operations
3. Logical operations in this order: Not, And, Or

### *Examples of Logical Operators used in the condition*

To pass a course, a student must have an average test score of at least 75 and an average project score of at least 35. The condition using the variables `sngTest` and `sngProj` is:

**`sngTest >= 75 And sngProj >= 35`**

Only people living in the state of Michigan who are over 65 years old receive a discount. The condition using the variables `strState` and `intAge` is:

**`UCase(strState) = "MICHIGAN" And intAge > 65`**

Only employees with job codes of 34 and 67 will receive a raise. The condition using the variable `intCode` is:

**`intCode = 34 Or intCode = 67`**

### *Nested Selection Structure*

A nested selection structure is one in which either the true path or the false path includes yet another selection structure. Any of the statements within either the true or false path of one selection structure may be another selection structure.

#### *Nested If in the true path*

```
If condition1 Then
    [instructions when condition1 is true]
    If condition2 Then
        [instructions when both condition1 and
        condition2 are true]
    Else
        [instructions when condition1 is true and
        condition2 is false]]
End If
Else
    [instructions when condition1 is false]]
End If
```

#### *Nested If in the false path*

```
If condition1 Then
    [instructions when condition1 is true]
Else
    If condition2 Then
        [instructions when condition1 is false and
        condition2 is true]
    Else
        [instructions when both condition1 and
        condition2 are false]]
    End If
End If
```



### *Nested If Example 1*

A selection structure that assigns a sales tax rate to the sngTax variable. The tax rate is determined by the state code stored in the intCode variable. Codes of 1 and 3 represent a 4% rate; a code of 2 represents a 5% rate. All other codes represent a 2% rate.

```
If intCode = 1 Or intCode = 3 Then
    sngTax = .04
Else
    If intCode = 2 Then
        sngTax = .05
    Else
        sngTax = .02
    End If
End If
```

### *Nested If Example 2*

A selection structure that assigns a bonus to the sngBonus variable. The bonus is determined by the salesperson's code (intCode) and, in some cases, by the sales amount (sngSales). If the code is 1 and the salesperson sold at least \$10,000, then the bonus is \$500; otherwise these salespeople receive \$200. If the code is 2 and the salesperson sold at least \$20,000, then the bonus is \$600; otherwise these salespeople receive \$550. All others receive \$150.

```
If intCode = 1 Then
    If sngSales >= 10000 Then
        sngBonus = 500
    Else
        sngBonus = 200
    End If
Else
    If intCode = 2 Then
        If sngSales >= 20000 Then
            sngBonus = 600
        Else
            sngBonus = 550
        End If
    Else
        sngBonus = 150
    End If
End If
```

It may appear that using compound logic will ease the clarity of this solution. The example below shows that this is not always the case.

```

If intCode = 1 And sngSales >= 10000 Then
    sngBonus = 500
Else
    If intCode = 1 And sngSales < 10000 Then
        sngBonus = 200
    Else
        If intCode = 2 And sngSales >= 20000 Then
            sngBonus = 600
        Else
            If intCode = 2 And sngSales < 20000          Then
                sngBonus = 550
            Else
                sngBonus = 150
            End If
        End If
    End If
End If

```

### *The Elself Clause*

When a selection is to be made from a range of values, the Elself structure may prove useful.

```

If iQuantity = 1 Or iQuantity = 2 Then
    dDiscount = 0
ElseIf iQuantity >= 3 And iQuantity <= 9 Then
    dDiscount = 0.1
ElseIf iQuantity >= 10 And iQuantity <=24 Then
    dDiscount = 0.2
ElseIf iQuantity >= 25 Then
    dDiscount = 0.3
Else
    dDiscount = 0
End If

```

The Elself clause is sometimes useful in the Nested If structure.

```

If sType = "Retail" Then
    If iQuantity <= 9 Then
        dDiscount = 0
    ElseIf iQuantity <= 19 Then
        dDiscount = .1
    ElseIf iQuantity >= 20 Then
        dDiscount = .2
    End If
Else '(sType <> "Retail")
    dDiscount = .4
End If

```

The comment following the keyword Else is used to clarify to the programmer the condition under which the Else clause will be executed.

### *The Decision Structure in the Invoice Total Application*

We can use an If statement to calculate the discount amount based on the dollar amount of the sale. In this case, a 20% discount will be applied to sales of at least \$100 and no discount will be applied to sales less than \$100.

```
Private Sub btnCalculate_Click(ByVal sender As System.Object, _  
    ByVal e As System.EventArgs) Handles btnCalculate.Click  
  
    Dim dOrderTotal As Decimal  
    Dim dDiscountPct As Decimal  
    Dim dDiscountAmount As Decimal  
    Dim dInvoiceTotal As Decimal  
  
    dOrderTotal = txtOrderTotal.Text  
    If dOrderTotal >= 100 Then  
        dDiscountPct = .2  
    Else  
        dDiscountPct = 0  
    End If  
    dDiscountAmount = dOrderTotal * dDiscountPct  
    dInvoiceTotal = dOrderTotal - dDiscountAmount  
  
    lblDiscountAmount.Text = dDiscountAmount  
    lblInvoiceTotal.Text = dInvoiceTotal  
    txtOrderTotal.Focus  
  
End Sub
```