

INTERNATIONAL
EDITION



Visual Basic® 2012

How To Program

SIXTH EDITION

Paul Deitel • Harvey Deitel • Abbey Deitel




ALWAYS LEARNING

PEARSON



Visual Basic® 2012

HOW TO PROGRAM



SIXTH EDITION

selected at a time, including none at all. The text that appears alongside a **CheckBox** is called the **CheckBox label** and is specified with the **Text** property. If the **CheckBox** is “on” (checked), its **Checked** property has the Boolean value **True**; otherwise, it’s **False** (“off”). The app’s GUI is shown in Fig. 5.17 and the code is shown in Fig. 5.18.

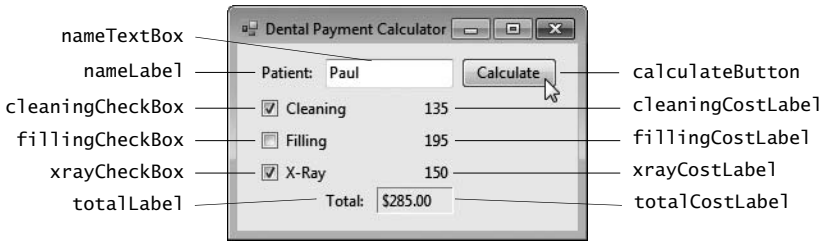


Fig. 5.17 | GUI for Dental Payment Calculator app.

```

1  ' Fig. 5.18: DentalPayment.vb
2  ' Dental Payment Calculator displays bill amount for a dental visit.
3  Public Class DentalPayment
4      ' calculate the bill amount
5      Private Sub calculateButtonClick(sender As Object,
6          e As EventArgs) Handles calculateButton.Click
7
8          ' if no name entered or no CheckBoxes checked, display message
9          If (nameTextBox.Text = String.Empty) OrElse
10             (Not cleaningCheckBox.Checked AndAlso
11              Not xrayCheckBox.Checked AndAlso
12              Not fillingCheckBox.Checked) Then
13
14              totalCostLabel.Text = String.Empty ' clear totalCostLabel
15
16              ' display an error message in a dialog
17              MessageBox.Show(
18                  "Please enter a name and check at least one item",
19                  "Missing Information", MessageBoxButtons.OK,
20                  MessageBoxIcon.Error)
21          Else ' add prices
22              ' total contains amount to bill patient
23              Dim total As Decimal = 0
24
25              ' if patient had a cleaning
26              If cleaningCheckBox.Checked Then
27                  total += Val(cleaningCostLabel.Text)
28              End If
29
30              ' if patient had a cavity filled
31              If fillingCheckBox.Checked Then
32                  total += Val(fillingCostLabel.Text)
33              End If

```

Fig. 5.18 | Dental Payment Calculator displays bill amount for a dental visit. (Part I of 2.)

```

34
35     ' if patient had an X-Ray taken
36     If xrayCheckBox.Checked Then
37         total += Val(xrayCostLabel.Text)
38     End If
39
40     ' display the total
41     totalCostLabel.Text = String.Format("{0:C}", total)
42 End If
43 End Sub ' calculateButtonClick
44 End Class ' DentalPayment

```

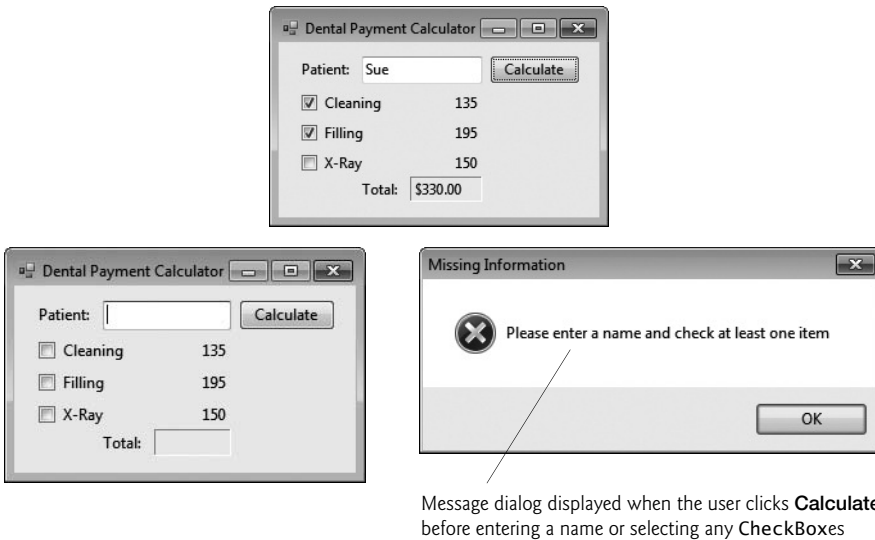


Fig. 5.18 | Dental Payment Calculator displays bill amount for a dental visit. (Part 2 of 2.)

Using Logical Operators

Recall from the problem statement that an error message should be displayed if the user does not enter a name or select at least one CheckBox. When the user presses the **Calculate** Button, lines 9–12 test these possibilities using logical operators **Not**, **OrElse** and **AndAlso**. The compound condition is split into two parts. If the condition in line 9 is **True**, the user did not enter a name, so an error message should be displayed. Short-circuit evaluation occurs with the **OrElse** operator, so the rest of the condition (lines 10–12) is ignored. If the user entered a name, then lines 10–12 check all three CheckBoxes. The expression

Not cleaningCheckBox.Checked

is **True** if the `cleaningCheckBox` is not checked—the `Checked` property has the value `False` and the `Not` operator returns `True`. If all three CheckBoxes are unchecked, the compound condition in lines 10–12 is **True**, so an error message should be displayed.

Calculating the Total of the Selected Services

If the user entered a name and selected at least one CheckBox, lines 21–42 calculate the total of the selected dental services. Each condition in lines 26, 31 and 36 uses the value

of one of the `CheckBox`'s `Checked` properties as the condition. If a given `CheckBox`'s `Checked` property is `True`, the body statement gets the value from the corresponding price `Label`, converts it to a number and adds it to the `total`. Then, line 41 displays the result in the `totalCostLabel`.

Displaying an Error Message Dialog

Class `MessageBox` creates message dialogs. We use a message dialog in lines 17–20 to display an error message to the user if the condition in lines 9–12 is `True`. `MessageBox` method `Show` displays the dialog. This method takes four arguments. The first is the `String` that's displayed in the message dialog. The second is the `String` that's displayed in the message dialog's title bar. The third and fourth are predefined constants that specify the `Button(s)` and the icon to show on the dialog, respectively.

Figure 5.19 shows the message box displayed by lines 17–20. It includes an **OK** button that allows the user to **dismiss** (that is, **close**) the message dialog by clicking the button. The program *waits* for the dialog to be closed before executing the next line of code—this type of dialog is known as a **modal dialog**. You can also dismiss the message dialog by clicking the dialog's close box—the button with an **X** in the dialog's upper-right corner.

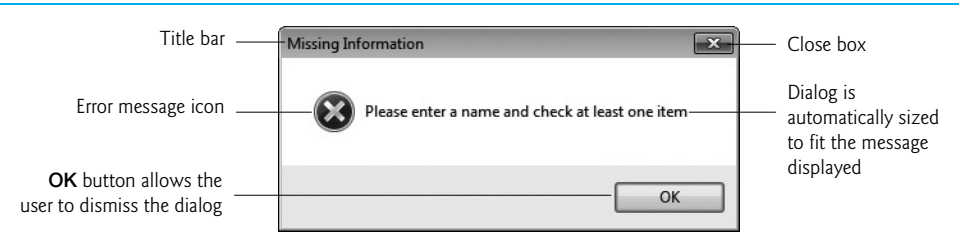


Fig. 5.19 | Message dialog displayed by calling `MessageBox.Show`.

Summary of Operator Precedence

Figure 5.20 displays the precedence of the operators introduced so far. The operators are shown from top to bottom in decreasing order of precedence.

Operators	Type
<code>^</code>	exponentiation
<code>+</code> <code>-</code>	unary plus and minus
<code>*</code> <code>/</code>	multiplicative operators
<code>\</code>	integer division
<code>Mod</code>	modulus
<code>+</code> <code>-</code>	additive operators
<code>&</code>	concatenation
<code><</code> <code><=</code> <code>></code> <code>>=</code> <code>=</code> <code><></code>	relational and equality
<code>Not</code>	logical NOT

Fig. 5.20 | Precedence of the operators discussed so far. (Part I of 2.)

Operators	Type
And AndAlso	logical AND
OrOrElse	logical inclusive OR
Xor	logical exclusive OR
= += -= *= /= \= ^= &=	assignment

Fig. 5.20 | Precedence of the operators discussed so far. (Part 2 of 2.)

5.12 Wrap-Up

In this chapter, we completed our introduction to the control statements that enable you to control the flow of execution. We demonstrated the `For...Next`, `Select...Case`, `Do...Loop While` and `Do...Loop Until` statements. Any algorithm can be developed using combinations of the sequence structure (that is, statements listed in the order in which they are to execute), the three types of selection statements—`If...Then`, `If...Then...Else` and `Select...Case`—and the seven types of repetition statements—`While...End While`, `Do While...Loop`, `Do Until...Loop`, `Do...Loop While`, `Do...Loop Until`, `For...Next` and `For Each...Next` (which we discuss in Chapter 7). We discussed how you can combine these building blocks using proven program-construction and problem-solving techniques. We showed how to use nested `For...Next` statements. We showed how to alter the flow of program control using the various `Exit` and `Continue` statements. You used the logical operators to form more complex conditional expressions in control statements. Chapter 6 discusses methods in greater depth and shows how to use them to organize larger programs as small, manageable pieces.

Summary

Section 5.2 For...Next Repetition Statement

- Counter-controlled repetition requires the name of a control variable, the initial value of the control variable, the increment (or decrement) by which the control variable is modified each time through the loop and the condition that tests for the final value of the control variable.
- The `For...Next` repetition statement specifies counter-controlled repetition details in a single line of code. In general, counter-controlled repetition should be implemented with `For...Next`.
- When the `For...Next` statement is reached, its control variable is initialized. Next, the *implied* loop-continuation condition is tested.
- The `To` keyword is required and is followed by the control variable's final value.
- The optional `Step` keyword specifies the increment. If `Step` and the value following it are omitted, the increment defaults to 1.
- If the loop-continuation condition is initially false, the `For...Next`'s body is not performed.
- When `Next` is reached, the control variable is incremented by the `Step` value, and the implied loop-continuation test is performed again.
- The first line of the `For...Next` statement sometimes is called the `For...Next` header.

- The general form of the For...Next statement is

```
For initialization To finalValue Step increment
statement
Next
```

where the *initialization* expression initializes the loop's control variable, *finalValue* determines whether the loop should continue executing and *increment* specifies the amount the control variable should be incremented each time through the loop.

- When the *initialization* expression in the For...Next statement header declares the control variable, the control variable can be used only in the body of the For...Next statement.
- A variable's scope specifies where the variable can be used in a program.
- The starting value, ending value and increment portions of a For...Next statement can contain arithmetic expressions.
- Local type inference determines a local variable's type based on its initializer value.
- You can also use local type inference with the control variable of a For...Next statement.
- The literal type character D (123.45D) indicates that a literal numeric value is of type Decimal. Other common literal type characters include C for Char ("T"C), F for Single (123.45F), S for Short (123S) and L for Long (123L).

Section 5.4 App: Interest Calculator

- The NumericUpDown control limits a user's choices to a range of values specified by its Minimum and Maximum properties. The Value property specifies the current value displayed in the control and can be used to obtain the current value. The Increment property specifies by how much the current number in the control changes when the user clicks the control's up or down arrow.
- When the user changes the value in a NumericUpDown control, a ValueChanged event occurs.
- Type Decimal should be used for monetary calculations.
- The built-in Val function obtains a value from a String of characters. The value returned is guaranteed to be a number. Val ignores whitespace. Val recognizes the decimal point as well as plus and minus signs that indicate whether a number is positive or negative. If Val receives an argument that cannot be converted to a number, it returns 0.
- The C ("currency") format specifier indicates that its corresponding value should be displayed in monetary format—with a dollar sign to the left and commas in the proper locations. This may differ based on your locale throughout the world.
- Other commonly used format specifiers include F for floating-point numbers (sets the number of decimal places to two), N for numbers (separates every three digits with a comma and sets the number of decimal places to two) and D for integers.

Section 5.6 Select...Case Multiple-Selection Statement

- A variable that's declared in the class, but outside all of the class's methods is called an instance variable. Such variables are special because they can be used by all of the class's methods.
- The Select...Case multiple-selection statement tests a variable or expression for each value that the variable or expression might assume, then takes different actions based on those values.
- The expression following the keywords Select Case is called the controlling expression. It's compared in order with each Case. If a matching Case is found, its code executes, then program control proceeds to the first statement after the Select...Case statement.
- Cases may contain single values, multiple values or ranges of values. Keyword To specifies a range.

- If no match occurs between the controlling expression's value and a Case, the optional Case Else executes. If there's not a Case Else, program control simply continues with the first statement after the Select...Case. When used, the Case Else must be the last Case.
- The required End Select keywords terminate the Select...Case statement.
- Case statements can use relational operators to determine whether the controlling expression satisfies a condition. Case Is < 0 tests for values less than 0.
- Multiple values can be tested in a Case statement by separating the values with commas.

Section 5.7 Do...Loop While and Do...Loop Until Repetition Statements

- The Do...Loop While repetition statement tests the loop-continuation condition after the loop body is performed—the loop body is always executed at least once.
- The Do...Loop Until repetition statement test the loop-termination condition after the loop body is performed—the loop body executes at least once.

Section 5.8 Using Exit to Terminate Repetition Statements

- When the Exit Do statement executes in a Do While...Loop, Do...Loop While, Do Until...Loop or Do...Loop Until statement, that repetition statement terminates and the program continues execution with the first statement after the repetition statement. Exit For and Exit While cause immediate exit from For...Next and While...End While loops, respectively.

Section 5.9 Using Continue in Repetition Statements

- A Continue statement terminates only the current iteration of a repetition statement and continues execution with the next iteration of the loop. Continue Do statement can be executed in a Do While...Loop, Do...Loop While, Do Until...Loop or Do...Loop Until statement. Continue For and Continue While can be used in For...Next and While...End While statements, respectively.
- Continue For causes a For...Next statement to increment the control variable, then evaluate the loop-continuation test. When Continue is used other repetition statements, the program evaluates the loop-continuation (or loop-termination) test immediately.
- Exit and Continue also can be used in nested control statements.

Section 5.10 Logical Operators

- A condition is an expression that results in a Boolean value—True or False.
- The logical operators can be used to form complex conditions.
- Operator And evaluates a complex condition to true if and only if both of its operands are true.
- Operator Or evaluates a complex condition to true unless both of its operands are false.
- Operator And has a higher precedence than the Or operator.
- The operators AndAlso andOrElse are similar to the And and Or operators, but an expression containing AndAlso or OrElse uses short-circuit evaluation.
- A condition containing the logical exclusive OR (Xor) operator is true if and only if one of its operands results in a true value and the other results in a false value. If both operands are true or both are false, the entire condition is false.
- The Not operator enables you to “reverse” the meaning of a condition.

Section 5.11 App: Dental Payment Calculator

- A CheckBox is known as a state button because it can be in the “on” state or the “off” state.
- When a CheckBox is selected, a check mark appears in the box. Any number of CheckBoxes can be selected at a time.