

Chapter 8

Dental Payment App

Introducing CheckBoxes and Message Dialogs





Objectives

In this chapter, you'll:

- Use `CheckBoxes` to allow users to select options.
- Use dialogs to display messages.
- Use logical operators to form more powerful conditions.



Outline

- 8.1** Test-Driving the **Dental Payment** App
- 8.2** Designing the **Dental Payment** App
- 8.3** Using CheckBoxes
- 8.4** Using a Dialog to Display a Message
- 8.5** Logical Operators
- 8.6** Designer-Generated Code
- 8.7** Wrap-Up



App Requirements

A dentist's office administrator wishes to create an app that employees can use to bill patients. The app must allow the user to enter the patient's name and specify which services were performed during the visit. The app will then calculate the total charges. If a user attempts to calculate a bill before any services are specified, or before the patient's name is entered, an error message will be displayed informing the user that necessary input is missing.

Test-Driving the Dental Payment App

- A CheckBox is a small square that either is blank or contains a check mark (☒).
- Open the Dental Payment application (Fig. 8.1).

Dental Payment

Dental Payment Form

Patient name:

<input type="checkbox"/> Cleaning	35
<input type="checkbox"/> Cavity Filling	150
<input type="checkbox"/> X-Ray	85

Total:

Calculate

CheckBox controls (unchecked)

Figure 8.1 Running the completed **Dental Payment** app.

Test-Driving the Dental Payment App (Cont.)

- Leave the **Patient name:** field blank, and deselect any CheckBoxes that you've selected before clicking the **Calculate** Button.
- An error message appears (Fig. 8.2); close it by clicking the OK button.

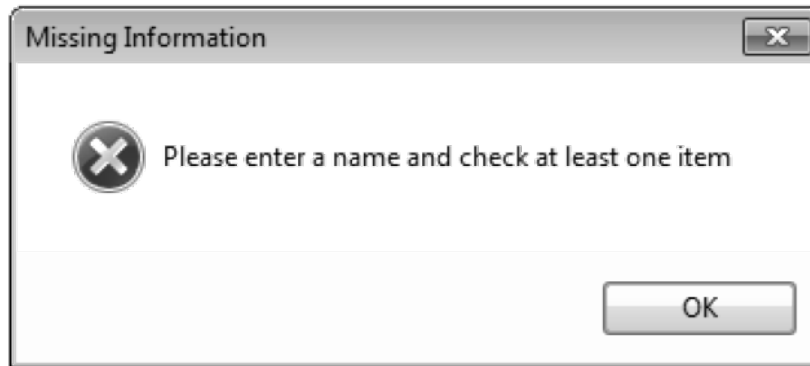


Figure 8.2 Message dialog appears when no name is entered and/or no CheckBoxes are selected.

Test-Driving the Dental Payment App (Cont.)



- Run the application and enter information (Fig. 8.3).

The screenshot shows a window titled "Dental Payment" with a standard Windows-style title bar (minimize, maximize, close buttons). The form inside is titled "Dental Payment Form". It contains a text input field for "Patient name:" with the value "Bob Jones". Below this is a list of services, each with a checked checkbox and a corresponding price:

Service	Price
<input checked="" type="checkbox"/> Cleaning	35
<input checked="" type="checkbox"/> Cavity Filling	150
<input checked="" type="checkbox"/> X-Ray	85

At the bottom right, there is a "Total:" label next to an empty text input field, and a "Calculate" button below it. An annotation "CheckBox controls (checked)" with three lines points to the checkboxes for Cleaning, Cavity Filling, and X-Ray.

Figure 8.3 Dental Payment app with input entered.

Test-Driving the Dental Payment App (Cont.)

- Uncheck the **Cavity Filling CheckBox** (Fig. 8.4).

CheckBox control
(unchecked)

Dental Payment Form	
Patient name:	Bob Jones
<input checked="" type="checkbox"/> Cleaning	35
<input type="checkbox"/> Cavity Filling	150
<input checked="" type="checkbox"/> X-Ray	85
Total:	
<button>Calculate</button>	

Figure 8.4 Dental Payment app with input changed.

Test-Driving the Dental Payment App (Cont.)

- Click the **Calculate** button to calculate the total (Fig. 8.5).

The screenshot shows a window titled "Dental Payment" with a standard Windows-style title bar (minimize, maximize, close buttons). The main content area is titled "Dental Payment Form". It contains a text input field for "Patient name:" with the value "Bob Jones". Below this is a list of services with checkboxes and associated costs:

Service	Cost
<input checked="" type="checkbox"/> Cleaning	35
<input type="checkbox"/> Cavity Filling	150
<input checked="" type="checkbox"/> X-Ray	85

At the bottom right, there is a "Total:" label next to a text input field containing "\$120.00". Below the total field is a "Calculate" button.

Figure 8.5 Dental Payment app with total calculated.

8.2 Designing the Dental Payment App

When the user clicks the “Calculate” Button

- Clear previous output

- If user has not entered a patient name or has not selected any CheckBoxes

 - Display message in dialog

- Else

 - Initialize the total to zero

 - If “Cleaning” CheckBox is selected

 - Add cost of a cleaning to the total

 - If “Cavity Filling” CheckBox is selected


 - Add cost of receiving a cavity filling to the total

 - If “X-Ray” CheckBox is selected

 - Add cost of receiving an x-ray to the total

 - Format total to be displayed as currency

 - Display total



Action	Control/Class/Object	Event
Label all the app's controls	titleLabel, nameLabel, totalLabel, cleanCostLabel, fillingCostLabel, xrayCostLabel	App is run
	calculateButton	Click
Clear previous output	totalResultLabel	
If user has not entered a patient name or has not selected any CheckBoxes	nameTextBox, cleanCheckBox, cavityCheckBox, xrayCheckBox	
Display message in dialog	MessageBox	

Figure 8.6 ACE table for **Dental Payment** app. (Part 1 of 2.)

Else Initialize the total to zero		
If “Cleaning” CheckBox is selected Add cost of a cleaning to the total	cleanCheckBox	
If “Cavity Filling” CheckBox is selected Add cost of receiving a cavity filling to the total	cavityCheckBox	
If “X-Ray” CheckBox is selected Add cost of receiving an x-ray to the total	xrayCheckBox	
Format total to be displayed as currency	String	
Display total	totalResultLabel	

Figure 8.6 ACE table for **Dental Payment** app. (Part 2 of 2.)

8.3 Using CheckBoxes

- A CheckBox is known as a **state button** because it can be in the on/off [true/false] state.
- The text that appears alongside a CheckBox is called the **CheckBox label**.
- If the CheckBox is checked, the **Checked property** contains the Boolean value `True`; otherwise, it contains `False`.



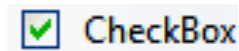
GUI Design Tip

A CheckBox's label should be descriptive and brief. When a CheckBox's label contains more than one word, use book-title capitalization.

Adding CheckBoxes to the Form

- Add a CheckBox to the Form by double clicking the icon in the **Toolbox**. Repeat this process until three CheckBoxes have been added to the Form.

- *Customizing the CheckBoxes.*



- First, set the AutoSize property to False and change the Size to 122, 24 for all three CheckBoxes.
- Name the first CheckBox cleanCheckBox, set its Location to 22, 113 and set its Text to Cleaning.
- Name the second CheckBox cavityCheckBox, set its Location to 22, 160 and set its Text to Cavity Filling.
- Name the final CheckBox xrayCheckBox, set its Location to 22, 207 and set its Text to X-Ray.



GUI Design Tip


Align groups of CheckBoxes either horizontally or vertically.

Adding the Calculate Button's Event Handler

- Double click the **Calculate** Button on the Form to create an event handler (Fig. 8.7).

```
2      ' handles Click event
3      Private Sub calculateButton_Click(sender As System.Object,
4          e As System.EventArgs) Handles calculateButton.Click
5
6          ' clear text displayed in Label
7          totalResultLabel.Text = String.Empty
8
9          ' total contains amount to bill patient
10         Dim total As Decimal = 0
11
12         ' if patient had a cleaning
13         If cleanCheckBox.Checked = True Then
14             total += Val(cleanCostLabel.Text)
15         End If
16
17         ' if patient had a cavity filled
18         If cavityCheckBox.Checked = True Then
19             total += Val(fillingCostLabel.Text)
20         End If
21
```

Figure 8.7 Using the Checked property. (Part 1 of 2)



```
22      ' if patient had an X-ray taken
23      If xrayCheckBox.Checked = True Then
24          total += Val(xrayCostLabel1.Text)
25      End If
26
27      ' display the total
28      totalResultLabel1.Text = String.Format("{0:C}", total)
29  End Sub ' calculateButton_Click
```

Figure 8.7 Using the Checked property. (Part 2 of 2)

Adding the Calculate Button's Event Handler (Cont.)

- Select the **Cleaning** CheckBox, and click the **Calculate** Button. The **Total:** field now displays **\$35.00** (Fig. 8.8).

The screenshot shows a window titled "Dental Payment" with a standard Windows-style title bar (minimize, maximize, close buttons). The main content area is titled "Dental Payment Form". It contains a text input field for "Patient name:". Below this are three items, each with a checkbox and a price: "Cleaning" with a price of 35, "Cavity Filling" with a price of 150, and "X-Ray" with a price of 85. At the bottom right, there is a "Total:" label next to a text input field displaying "\$0.00". Below the total field is a "Calculate" button. An arrow points from the text annotation to the "\$0.00" value in the total field.

Service	Price
<input type="checkbox"/> Cleaning	35
<input type="checkbox"/> Cavity Filling	150
<input type="checkbox"/> X-Ray	85

Total: \$0.00

Calculate

App calculates a bill of \$0.00 when no **CheckBoxes** are checked

Figure 8.8 App running without input.

8.4 Using a Dialog to Display a Message

- This dialog (Fig. 8.9) contains:
- a title bar
 - a close box
 - a message
 - an OK button
 - an icon indicating the tone of the message

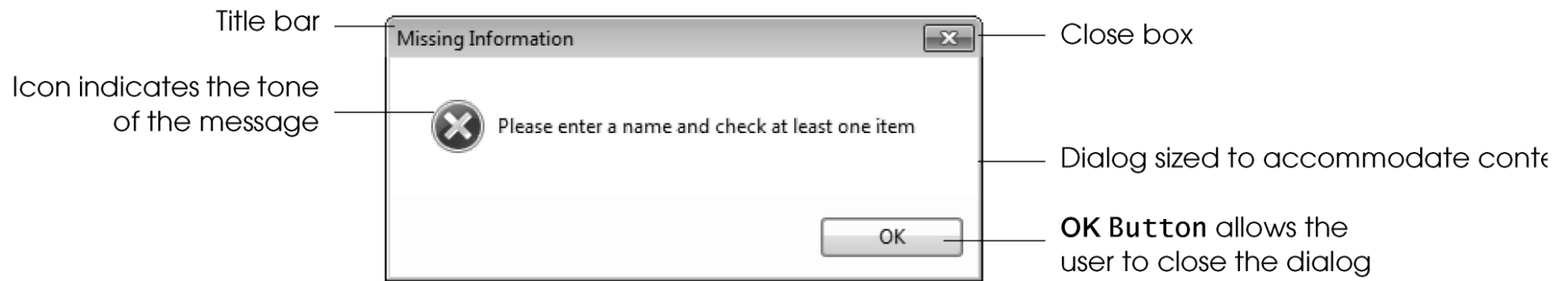


Figure 8.9 Dialog displayed by the app.




GUI Design Tip

Text displayed in a dialog should be descriptive and brief.

Displaying a Message Dialog Using `MessageBox.Show`

- The message should display only if the user does not enter the patient's name.
- If no data has been entered, the expression `nameTextBox.Text = String.Empty` evaluates to `True` (Fig. 8.10).




```
6      ' clear text displayed in Label
7      totalResultLabel.Text = String.Empty
8
9      ' if no name entered, display an error message
10     If nameTextBox.Text = String.Empty Then
11
12     End If
```

Figure 8.10 Adding an If...Then statement to the calculateButton Click event handler.

Displaying a Message Dialog Using `MessageBox.Show` (Cont.)

- `MessageBox.Show` has four arguments separated by commas (Fig. 8.11).
- The first argument specifies the dialog text.
 - The second argument specifies the title bar text.
 - The third argument indicates which `Button` to display.
 - The fourth argument indicates which icon appears.




```
6      ' clear text displayed in Label
7      totalResultLabel.Text = String.Empty
8
9      ' if no name entered, display an error message
10     If nameTextBox.Text = String.Empty Then
11
12         ' display an error message in a dialog
13         MessageBox.Show(
14             "Please enter a name and check at least one item",
15             "Missing Information", MessageBoxButtons.OK,
16             MessageBoxIcon.Error)
17     Else ' add prices
```

Change End If to Else —

Figure 8.11 Message dialog code that displays a message to users.

Displaying a Message Dialog Using `MessageBox.Show` (Cont.)

- Insert the keywords `End If` to close the `If...Then...Else...` statement (Fig. 8.12).




```
37         ' display the total
38         totalResultLabel.Text = String.Format("{0:C}", total)
39     End If
40 End Sub ' calculateButton_Click
```

Figure 8.12 Ending the If...Then...Else statement.

Displaying a Message Dialog Using `MessageBox.Show` (Cont.)

- Figure 8.13 displays the entire method `calculateButton_Click` after the new code has been added.



```
2      ' handles Click event
3  Private Sub calculateButton_Click(sender As System.Object,
4      e As System.EventArgs) Handles calculateButton.Click
5
6      ' clear text displayed in Label
7      totalResultLabel.Text = String.Empty
8
9      ' if no name entered, display an error message
10     If nameTextBox.Text = String.Empty Then
11
12         ' display an error message in a dialog
13         MessageBox.Show(
14             "Please enter a name and check at least one item",
15             "Missing Information", MessageBoxButtons.OK,
16             MessageBoxIcon.Error)
```

Figure 8.13 calculateButton_Click event handler. (Part 1 of 2.)

```

17         Else ' add prices
18
19         ' total contains amount to bill patient
20         Dim total As Decimal = 0
21
22         ' if patient had a cleaning
23         If cleanCheckBox.Checked = True Then
24             total += Val(cleanCostLabel.Text)
25         End If
26
27         ' if patient had a cavity filled
28         If cavityCheckBox.Checked = True Then
29             total += Val(fillingCostLabel.Text)
30         End If
31
32         ' if patient had an X-ray taken
33         If xrayCheckBox.Checked = True Then
34             total += Val(xrayCostLabel.Text)
35         End If
36
37         ' display the total
38         totalResultLabel.Text = String.Format("{0:C}", total)
39     End If
40 End Sub ' calculateButton_Click

```

Figure 8.13 calculateButton_Click event handler. (Part 2 of 2.)

Displaying a Message Dialog Using `MessageBox.Show` (Cont.)

- Run the application (Fig. 8.14).

Dental Payment

Dental Payment Form

Patient name:

☐ Cleaning 35

☐ Cavity Filling 150

☐ X-Ray 85

Total:


User must now enter a name

App calculates a bill of
\$0.00 when a name is
entered and no
CheckBoxes are checked

Figure 8.14 Total calculated without any CheckBoxes selected.

8.4 Using a Dialog to Display a Message (Cont.)

- Figure 8.15 lists the available `MessageBoxButtons` constants.



MessageBoxButtons Constants	Description
OK	OK Button. Allows the user to <i>acknowledge</i> a message.
OKCancel	OK and Cancel Buttons. Allow the user to <i>continue</i> or <i>cancel</i> an operation.
YesNo	Yes and No Buttons. Allow the user to <i>respond</i> to a question.
YesNoCancel	Yes , No and Cancel Buttons. Allow the user to <i>respond</i> to a question or <i>cancel</i> an operation.
RetryCancel	Retry and Cancel Buttons. Allow the user to <i>retry</i> or to <i>cancel</i> an operation that has <i>failed</i> .
AbortRetryIgnore	Abort , Retry and Ignore Buttons. When one of a series of operations has <i>failed</i> , these Buttons allow the user to <i>abort</i> the entire sequence, <i>retry</i> the <i>failed</i> operation or <i>ignore</i> the <i>failed</i> operation and <i>continue</i> .

Figure 8.15 Message dialog MessageBoxButtons constants.

8.4 Using a Dialog to Display a Message (Cont.)

- Some of the available icon constants are shown in Fig. 8.16.




MessageBoxIcon Constants	Icon	Description
Exclamation		Icon containing an exclamation point. Typically used to caution the user against potential problems.
Information		Icon containing the letter “i.” Typically used to display information about the state of the app.
None		No icon is displayed.
Error		Icon containing an × in a red circle. Typically used to alert the user to errors or critical situations.

Figure 8.16 Some message dialog MessageBoxIcon constants.

8.5 Logical Operators

- **Simple conditions** such as `count <= 10`, `total > 1000`, and `number <> value` use only one condition with one of the operators `>`, `<`, `>=`, `<=`, `=` or `<>`.
- **Logical operators** can be used to form complex conditions.
- The logical operators are:
 - `And`
 - `AndAlso`
 - `Or`
 - `OrElse`
 - `Xor`
 - `Not`



Error-Prevention Tip

Always write the *simplest* condition possible by limiting the number of logical operators used. Conditions with many logical operators can be hard to read and can introduce subtle bugs into your apps.


8.5 Logical Operators (Cont.)

➤ Using AndAlso

- To ensure that two conditions are *both* true in an application, use the logical AndAlso operator as follows:

```
If genderTextBox.Text = "Female" AndAlso age >= 65 Then  
    seniorFemales += 1  
End If
```

- Figure 8.17 illustrates the outcome of using the AndAlso operator with two expressions. Such tables are called truth tables.



<i>expression1</i>	<i>expression2</i>	<i>expression1 AndAlso expression2</i>
False	False	False
False	True	False
True	False	False
True	True	True

Figure 8.17 Truth table for the AndAlso operator.

8.5 Logical Operators (Cont.)

➤ Using OrElse

- To ensure that either *or* both of two conditions are true, use the **OrElse** operator.


```
If (semesterAverage >= 90) OrElse (finalExam >= 90) Then  
    MessageBox.Show("Student grade is A", "Student Grade", _  
        MessageBoxButtons.OK, MessageBoxIcon.Information)  
End If
```

- Figure 8.18 provides a truth table for the OrElse operator.
- Note that the AndAlso operator has a higher precedence.



Error-Prevention Tip

When writing conditions that contain combinations of **AndAlso** and **OrElse** operators, use parentheses to ensure that the conditions evaluate properly. Otherwise, logic errors could occur because **AndAlso** has *higher* precedence than **OrElse**.



<i>expression1</i>	<i>expression2</i>	<i>expression1 OrElse expression2</i>
False	False	False
False	True	True
True	False	True
True	True	True

Figure 8.18 Truth table for the OrElse operator.

8.5 Logical Operators (Cont.)

➤ Short-Circuit Evaluation

- The following expression stops immediately if `genderTextBox.Text` is not equal to "Female"

`(genderTextBox.Text = "Female") AndAlso (age >= 65)`

- The evaluation of the second expression is irrelevant; once the first expression is known to be false, the whole expression must be false.
- This way of evaluating logical expressions, called **short-circuit evaluation**, requires fewer operations and takes less time.

➤ Visual Basic also provides the And and Or operators, which do not short-circuit.

8.5 Logical Operators (Cont.)

➤ Using Xor

- 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.
- Figure 8.19 provides a truth table for the Xor operator.



<i>expression1</i>	<i>expression2</i>	<i>expression1 Xor expression2</i>
False	False	False
False	True	True
True	False	True
True	True	False

Figure 8.19 Truth table for the logical exclusive OR (Xor) operator.

8.5 Logical Operators (Cont.)

➤ Using Not

- Visual Basic's Not operator enables you to “reverse” the meaning of a condition.

```
If Not (grade = value) Then  
    displayLabel.Text = "They are not equal!"  
End If
```

- The parentheses around the condition `grade = value` improve the readability of the condition.
- Figure 8.20 provides a truth table for the Not operator.



<i>expression</i>	<i>Not expression</i>
False	True
True	False

Figure 8.20 Truth table for the **Not** operator (logical negation).

Using Logical Operators in Complex Expressions


- Note the use of OrElse and AndAlso (Fig. 8.21).
- Note that users must enter a name and select at least one CheckBox before they click the **Calculate** Button, or the dialog will be displayed.



add the highlighted code

```
9      ' if no name entered or no CheckBox checked, display message
10     If (nameTextBox.Text = String.Empty) OrElse
11         (cleanCheckBox.Checked = False AndAlso
12          xrayCheckBox.Checked = False AndAlso
13          cavityCheckBox.Checked = False) Then
```


Figure 8.21 Using the AndAlso and OrElse logical operators.



Using logical operators and the Checked property of a CheckBox	10 [1 Public Class DentalPaymentForm 2 ' handles Click event 3 Private Sub calculateButton_Click(sender As System.Object, 4 e As System.EventArgs) Handles calculateButton.Click 5 6 ' clear text displayed in Label 7 totalResultLabel.Text = String.Empty 8 9 ' if no name entered or no CheckBox checked, display message 10 If (nameTextBox.Text = String.Empty) OrElse 11 (cleanCheckBox.Checked = False AndAlso 12 xrayCheckBox.Checked = False AndAlso 13 cavityCheckBox.Checked = False) Then 14 15 ' display an error message in a dialog 16 MessageBox.Show(_ 17 "Please enter your name and check at least one item", 18 "Missing Information", MessageBoxButtons.OK 19 MessageBoxIcon.Error) 20 Else ' add prices 21 22 ' total contains amount to bill patient 23 Dim total As Decimal = 0 24
--	------	---

Displaying a MessageBox	16 [15 ' display an error message in a dialog 16 MessageBox.Show(_ 17 "Please enter your name and check at least one item", 18 "Missing Information", MessageBoxButtons.OK 19 MessageBoxIcon.Error) 20 Else ' add prices 21 22 ' total contains amount to bill patient 23 Dim total As Decimal = 0 24
----------------------------	------	---

Figure 8.22 Code for the **Dental Payment** app. (Part 1 of 2.)



	25	' if patient had a cleaning
Determine whether	26	If cleanCheckBox.Checked = True Then
cleanCheckBox is	27	total += Val(cleanCostLabel.Text)
checked	28	End If
	29	
Determine whether	30	' if patient had a cavity filled
cavityCheckBox is	31	If cavityCheckBox.Checked = True Then
checked	32	total += Val(fillingCostLabel.Text)
	33	End If
	34	
Determine whether	35	' if patient had an X-ray taken
xrayCheckBox is	36	If xrayCheckBox.Checked = True Then
checked	37	total += Val(xrayCostLabel.Text)
	38	End If
	39	
	40	' display the total
	41	totalResultLabel.Text = String.Format ("{0:C}", total)
	42	End If
	43	End Sub ' calculateButton_Click
	44	End Class ' DentalPaymentForm

Figure 8.22 Code for the **Dental Payment** app. (Part 2 of 2.)

8.6 Designer-Generated Code

- Like the variables you've declared, GUI controls also must be declared before they're used.
- When you work in **Design** view, Visual Basic automatically declares the controls for you.
- To improve the readability of your application code, Visual Basic “hides” the GUI declarations and other GUI code it generates in a *separate* file.
- Click the **Show All Files** button in the **Solution Explorer**, then click the plus (+) sign next to `DentalPayment.vb` to expand its node.

8.6 Designer-Generated Code (Cont.)

- Figure 8.23 shows some of the declarations that the IDE generated.

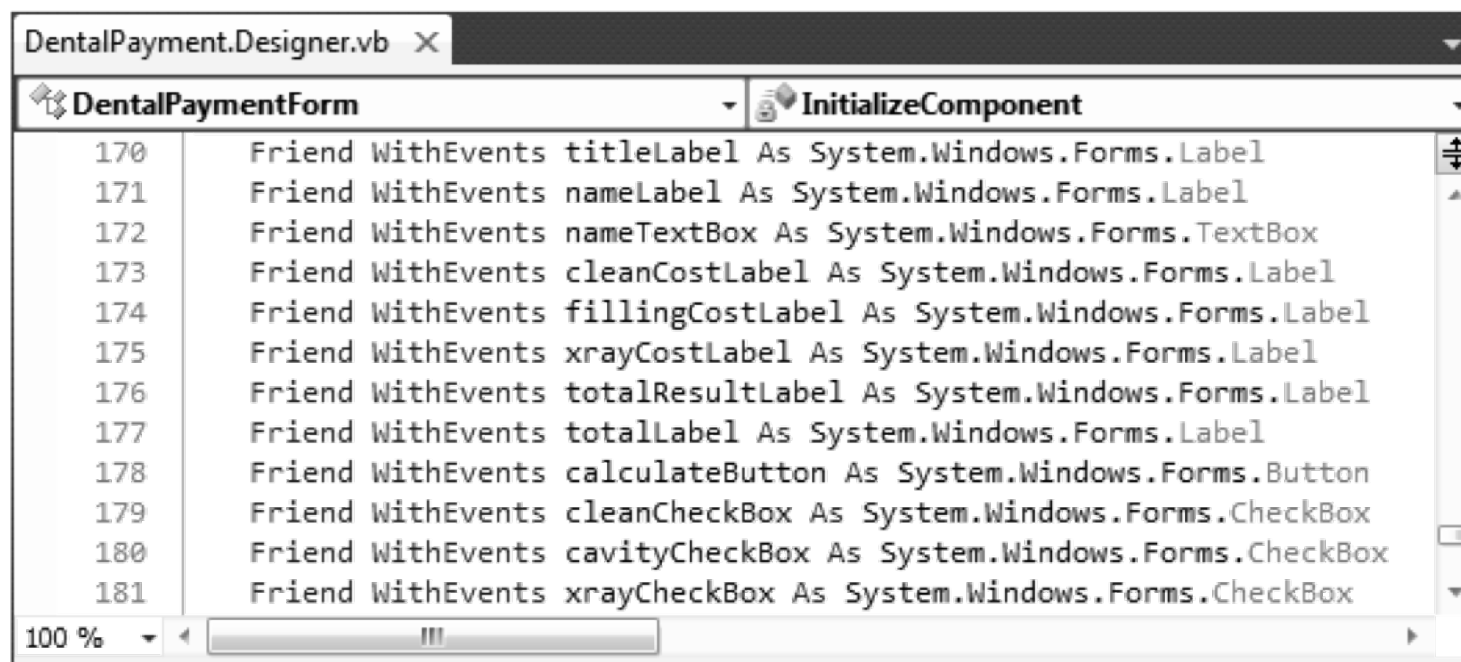
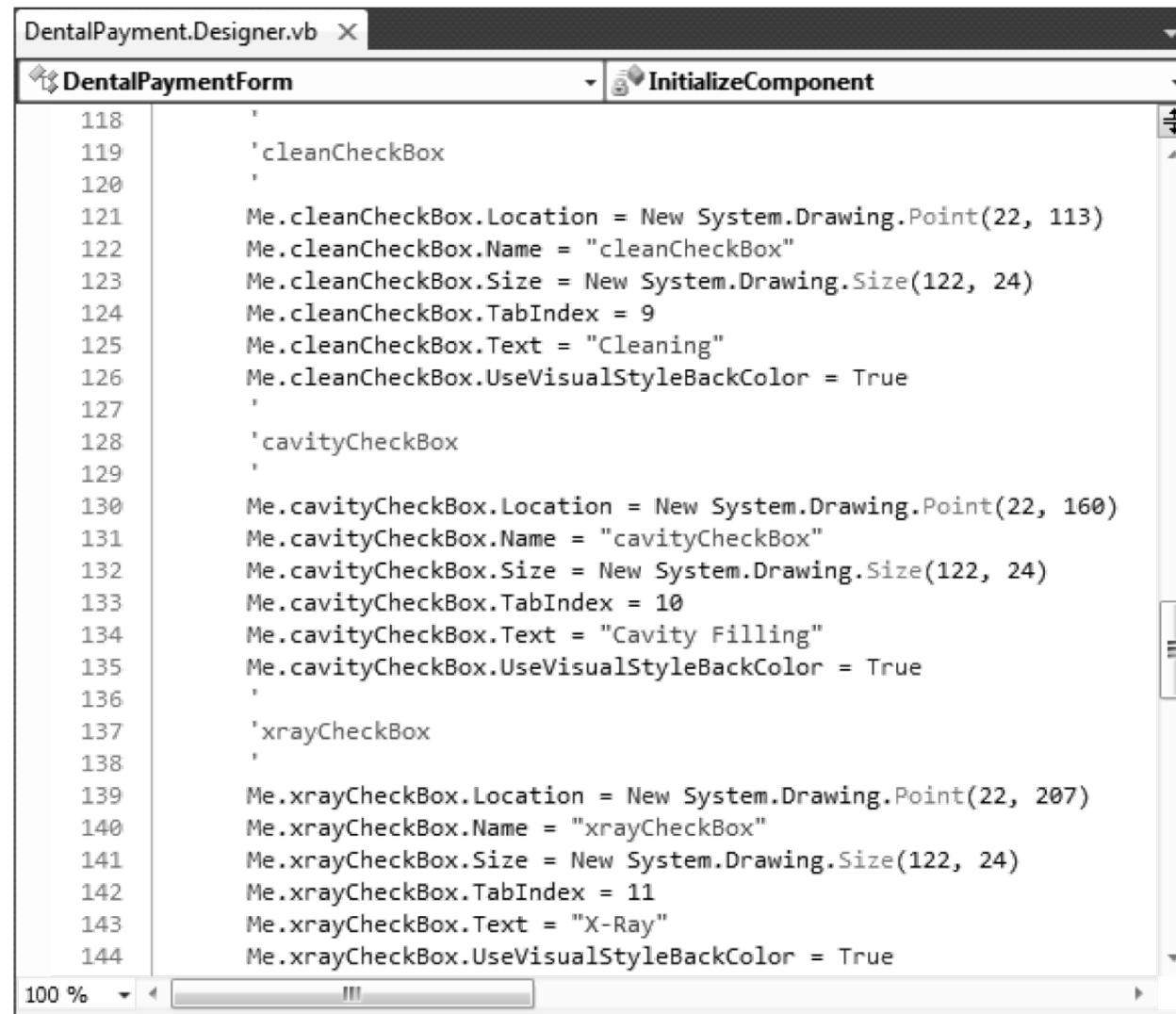


Figure 8.23 GUI declarations for the controls in the **Dental Payment** app.

8.6 Designer-Generated Code (Cont.)

- Figure 8.24 shows some of the statements that the IDE generated when you set the properties of the **CheckBoxes**.



```
DentalPayment.Designer.vb X
DentalPaymentForm InitializeComponent
118 '
119 'cleanCheckBox
120 '
121 Me.cleanCheckBox.Location = New System.Drawing.Point(22, 113)
122 Me.cleanCheckBox.Name = "cleanCheckBox"
123 Me.cleanCheckBox.Size = New System.Drawing.Size(122, 24)
124 Me.cleanCheckBox.TabIndex = 9
125 Me.cleanCheckBox.Text = "Cleaning"
126 Me.cleanCheckBox.UseVisualStyleBackColor = True
127 '
128 'cavityCheckBox
129 '
130 Me.cavityCheckBox.Location = New System.Drawing.Point(22, 160)
131 Me.cavityCheckBox.Name = "cavityCheckBox"
132 Me.cavityCheckBox.Size = New System.Drawing.Size(122, 24)
133 Me.cavityCheckBox.TabIndex = 10
134 Me.cavityCheckBox.Text = "Cavity Filling"
135 Me.cavityCheckBox.UseVisualStyleBackColor = True
136 '
137 'xrayCheckBox
138 '
139 Me.xrayCheckBox.Location = New System.Drawing.Point(22, 207)
140 Me.xrayCheckBox.Name = "xrayCheckBox"
141 Me.xrayCheckBox.Size = New System.Drawing.Size(122, 24)
142 Me.xrayCheckBox.TabIndex = 11
143 Me.xrayCheckBox.Text = "X-Ray"
144 Me.xrayCheckBox.UseVisualStyleBackColor = True
```

Figure 8.24 Statements that configure the CheckBox properties.