# *Introducing the Do While…Loop and Do Until…Loop Repetition Statements*

## 7.4 Control Structures (Cont.)

From the previous discussions, we conclude that Visual Basic has 11 control statements

➔1 sequence structure,
➔3 types of selection statements and
➔7 types of repetition statements.

➢ All Visual Basic apps are formed by combining as many of each type of control statement as is necessary.

## 7.4 Control Statements (Cont.)

### 7.4.3 Repetition Structures

Visual Basic provides seven types of repetition statements for performing a statement or group of statements repeatedly:

1. *Do While...Loop*
2. Do Until...Loop
3. Do...Loop  While
4. Do...Loop  Until
5. While...End While
6. For...Next
7. For  Each...Next

## ***Comments in VB.net Code***

- **To explain the purpose of a program, or a statement, a comment statement is added**
  - ***For yourself and others***
- **Any statement beginning with an apostrophe (`) or REM is a comment**
- **Comments can be added to end of statements using only apostrophe**

## ***Using the Pmt Function***

- Calculates periodic payment on loan or investment
- Syntax: Pmt(*Rate*, *NPer*, *PV*)
  - *Rate:* interest rate per period
  - *NPer:* total number of payment periods (the term)
  - *PV:* present value of the loan or investment

# The ListBox control

- The ListBox  is always ued with the repeatition statements
- It allows users to view and/or select from multiple items in a list (Fig. 9.1).

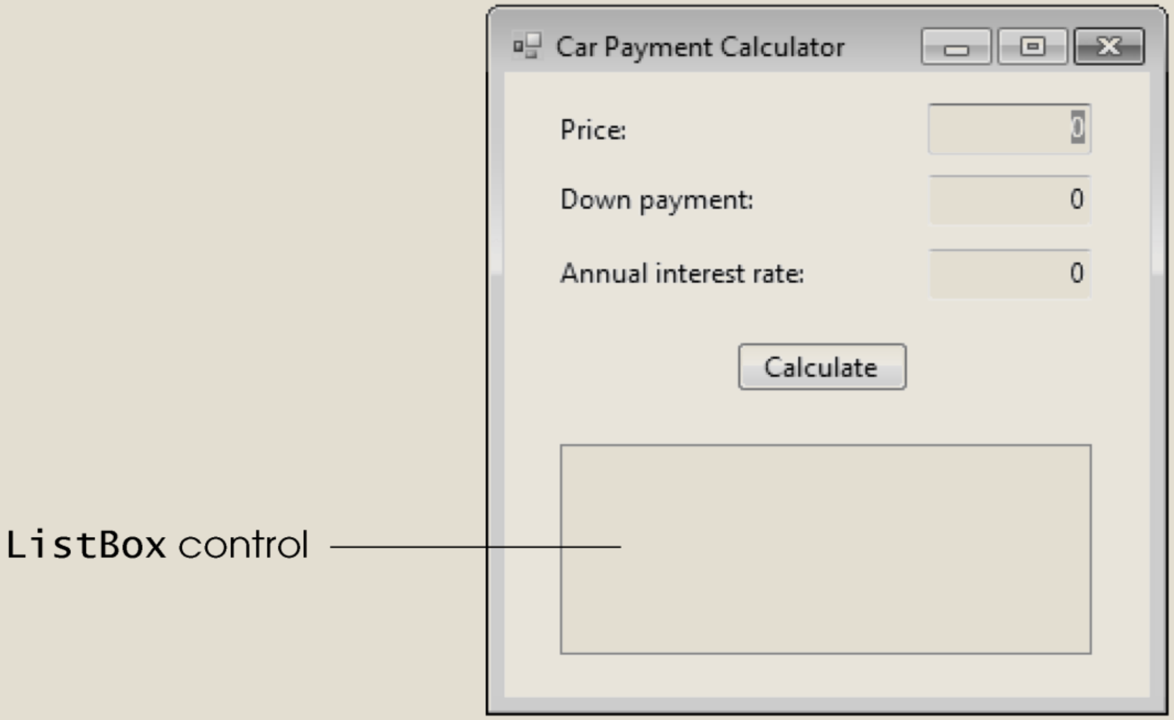
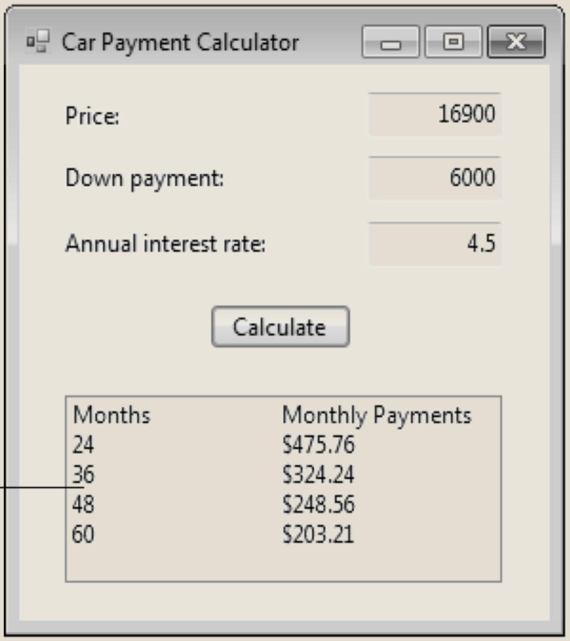


**Figure 9.1** **Car Payment Calculator** app before data has been entered

## The Car Payment Calculator Application

- Enter data to test the app then Click the Calculate Button.
- The app displays the monthly payment amounts in the ListBox (Fig. 9.3) in a tabular format.



| | |
|---|---|
| Price: | 16900 |
| Down payment: | 6000 |
| Annual interest rate: | 4.5 |

```
Months     Monthly Payments
24         $475.76
36         $324.24
48         $248.56
60         $203.21
```

**Figure 9.3** **Car Payment Calculator** app displaying calculation results.

## *Do While...Loop Repetition Statement*

- A **repetition statement** repeats actions, depending on the value of a condition.
- If you go to the grocery store with a list of items to purchase, you go through the list until you've put each item in your shopping cart
- This process is described by the following pseudocode statements:

```
Do while there are more items on my shopping list
    Put next item in cart
    Cross it off my list
```

## Do While…Loop Repetition Statement (Cont.)

- Using a Do While...Loop statement, this code finds the first power of 3 greater than 50.

```
Dim product As Integer = 3
Do While product <= 50
   product *= 3
Loop
```

- The condition in the Do While...Loop statement, *product <= 50*, is referred to as the *loop-continuation condition*.
- When the loop-continuation condition becomes false, the Do While...Loop statement terminates.

## Do While...Loop Repetition Statement (Cont.)

- The diagram in Fig. 9.4 illustrates the flow of control in the preceding Do While...Loop repetition statement.

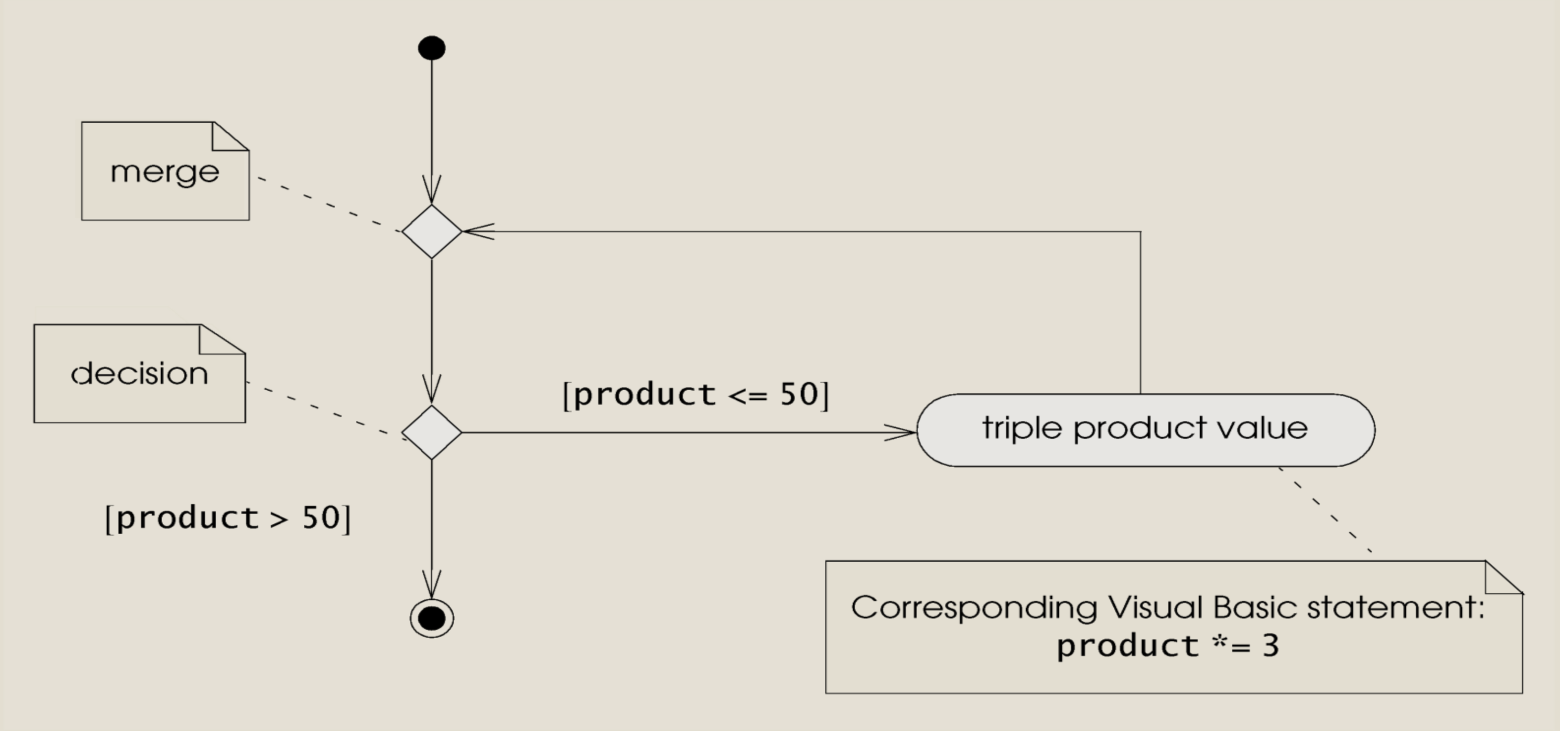


**Figure 9.4** Do While...Loop repetition statement UML activity diagram.

## 9.2 Do While...Loop Repetition Statement (Cont.)

- The transition arrow emerging from the action state points back to the merge, creating a loop.
- 
- The diamond-shaped merge symbol joins two flows of activity into one.

**Common Programming Error**

Provide in the body of every `Do While`…`Loop` statement an action that eventually causes the condition to become false. If you do not, the repetition statement never terminates, causing an error called an **infinite loop**. Such an error causes the app to "hang up." When an infinite loop occurs in your app, return to the IDE and select **Debug > Stop Debugging**.

## ***Do Until...Loop Repetition Statement***

- The following  statements describe the repetitive actions that occur during the same shopping trip:

```
Do until there are no more items on my shopping list
        Put next item in cart
    Cross it off my list
```

- Statements in the body of a Do Until...Loop are executed repeatedly for as long as the loop-termination condition remains `False`.

  This is known as *a* ***loop-termination condition***.

## Do Until…Loop Repetition Statement (Cont.)

- Using a Do Until...Loop, the same code to find the first power of 3 larger than 50 will be:

```
Dim product As Integer = 3
Do Until product > 50
       product *= 3
Loop
```

## Common Programming Error

Failure to provide the body of a `Do Until...Loop` statement with an action that eventually causes the condition in the `Do Until...Loop` to become true creates an infinite loop.

## Do Until...Loop Repetition Statement (Cont.)

- The UML activity diagram in Fig. 9.5 illustrates the flow of control for the Do Until...Loop repetition statement.

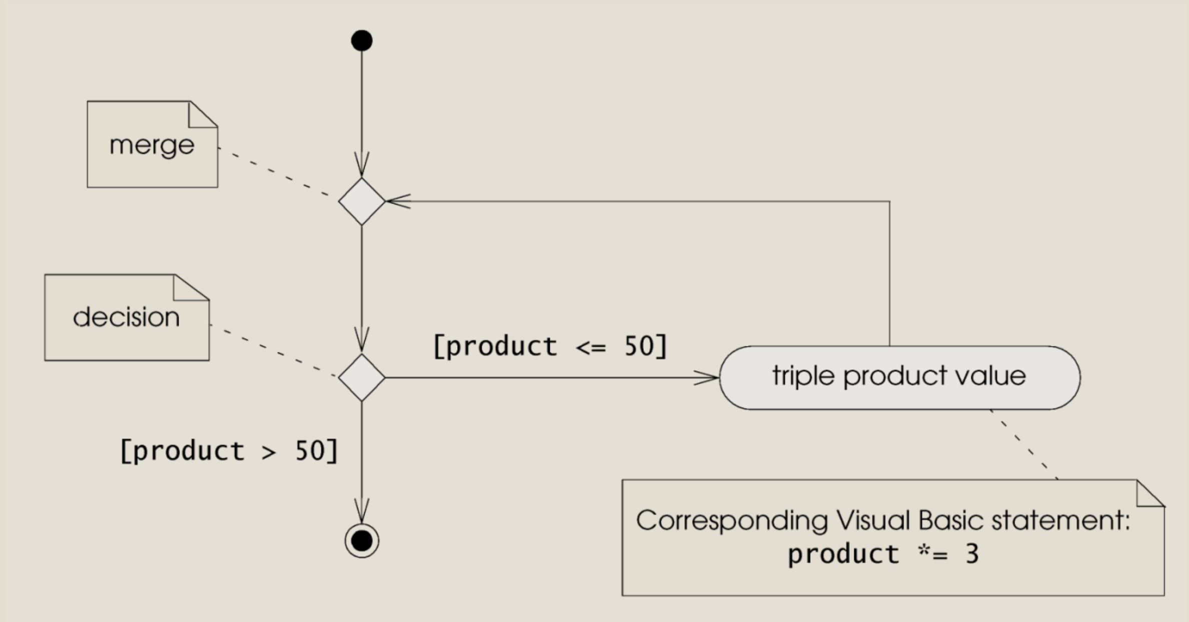


**Figure 9.5** Do Until...Loop repetition statement UML activity diagram.

## Constructing the Car Payment Calculator App

## App Requirements

*Typically, banks offer car loans for periods ranging from two to five years (24 to 60 months). Borrowers repay the loans in monthly installments. The amount of each monthly payment is based on the length of the loan, the amount borrowed and the interest rate. Create an app that allows the customer to enter the price of a car, the down-payment amount and the annual interest rate of the loan. The app should display the loan's duration in months and the monthly payments for two-, three-, four- and five-year loans. The variety of options allows the user to compare repayment plans and choose the most appropriate.*

## Constructing the Car Payment Calculator App

```
When the user clicks the Calculate Button
    Initialize loan length to two years
    Clear the ListBox of any previous calculation results
    Add a header to the ListBox

    Get car price from a TextBox
    Get down payment from a TextBox
    Get annual interest rate from a TextBox

    Calculate loan amount (car price – down payment)
    Calculate monthly interest rate (annual interest rate / 12)
```

## Constructing the Car Payment Calculator App (Cont.)

```
Do while loan length is less than or equal to five years
    Convert the loan length in years to number of months

    Calculate monthly payment based on loan amount,
    monthly interest rate and loan length in months

    Insert result into ListBox
    Increment loan length in years by one year
```

## Adding a ListBox to the Car Payment Calculator App

- Add a ListBox control from the Toolbox.

- Change the ListBox's Name property to paymentsListBox. Set the Location property to
24, 166 and the Size property to 230, 94 (Fig. 9.7).

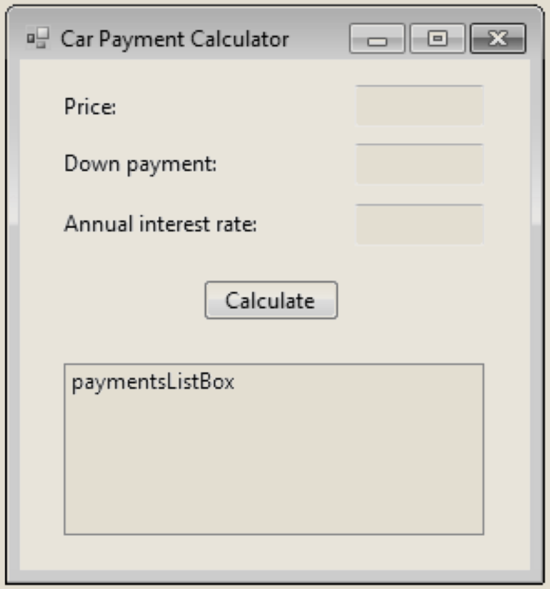


**Figure 9.7** `ListBox` added to **Car Payment Calculator** app's `Form`.

**Good Programming Practice**

Append the `ListBox` suffix to all `ListBox` control names.

**GUI Design Tip**

A `ListBox` should be large enough either to display all of its contents or to allow scrollbars to be used easily, if necessary.

## Using Code to Change a ListBox's Contents

- Double click the Calculate Button to generate the empty event handler (Fig. 9.8).
- To remove all content from the ListBox, call method *Clear of the ListBox's Items* property.
- This property enables you to add content to and remove content from the ListBox.
- The Items property returns an object that contains a list of items displayed in the ListBox.

```
   ' handles Calculate Button's Click event
   Private Sub calculateButton_Click(sender As System.Object,
      e As System.EventArgs) Handles calculateButton.Click

      ' remove text displayed in ListBox
      paymentsListBox.Items.Clear()
   End Sub ' calculateButton_Click
```

**Figure 9.8** Clearing the contents of a `ListBox`.

## Using Code to Change a ListBox's Contents (Cont.)

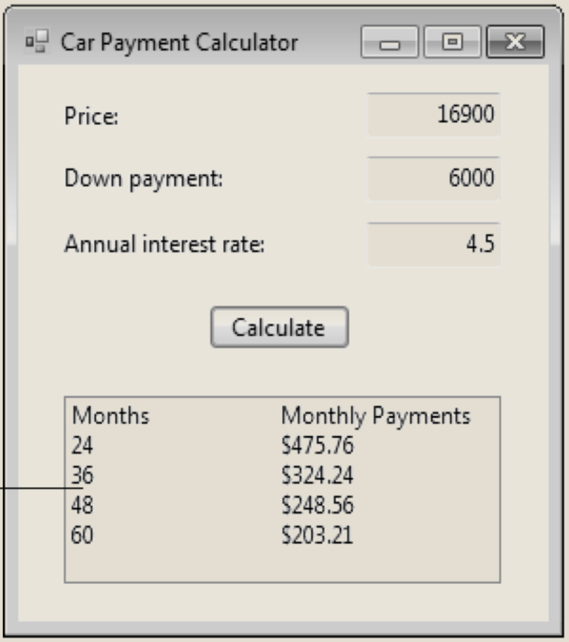
- The ListBox displays the number of monthly payments and the amount per payment.
- To clarify the information that's being displayed, we add a line of text—called a header—to ListBox using Method Add (lines 10–11 of Fig. 9.9)

```
 6          ' remove text displayed in ListBox
 7          paymentsListBox.Items.Clear()
 8
 9          ' add header to ListBox
10          paymentsListBox.Items.Add("Months" & ControlChars.Tab &
11             ControlChars.Tab & "Monthly Payments")
12      End Sub ' calculateButton_Click
```

**Figure 9.9** Adding a header to a ListBox.

# Using Code to Change a ListBox's Contents (Cont.)

- The ampersand (&) is the **string-concatenation operator—** it concantenates (combines) two operands into one string value by appending the right operand's text to the end of the left operand's text.
- In lines 10–11, the header is created by joining the values "Months" and "Monthly Payments" with two ControlChars.Tab constants—each inserts a tab character in the string to separate the columns (Fig. 9.3).



| | |
|---|---|
| Price: | 16900 |
| Down payment: | 6000 |
| Annual interest rate: | 4.5 |

```
Months      Monthly Payments
24          $475.76
36          $324.24
48          $248.56
60          $203.21
```

**Figure 9.3** **Car Payment Calculator** app displaying calculation results.

## Declaring Variables and Retrieving User Input

- The calculation requires the length in months, but the loop-continuation condition uses the number of years (Fig. 9.10).

```
   Private Sub calculateButton_Click(sender As System.Object,
      e As System.EventArgs) Handles calculateButton.Click

      Dim years As Integer = 2 ' repetition counter
      Dim months As Integer = 0 ' payment period
      Dim price As Decimal = 0 ' car price
      Dim downPayment As Decimal = 0 ' down payment
      Dim interest As Double = 0 ' interest rate
      Dim monthlyPayment As Decimal = 0 ' monthly payment
      Dim loanAmount As Decimal = 0 ' cost after down payment
      Dim monthlyInterest As Double = 0 ' monthly interest rate

      ' remove text displayed in ListBox
      paymentsListBox.Items.Clear()
```

Variables to store the length of the loan

Variables to store user input

Variables to store calculation results

**Figure 9.10** Variables for the **Car Payment Calculator** app.

## Declaring Variables and Retrieving User Input (Cont.)

- Line 26 (Fig. 9.11) divides the interest rate by 100—if the user enters 5, the interest rate is 0.05.

```
17
18         ' add header to ListBox
19         paymentsListBox.Items.Add("Months" & ControlChars.Tab &
20            ControlChars.Tab & "Monthly Payments")
21
22         ' retrieve user input and assign values
23         ' to their respective variables
24         downPayment = Val(downPaymentTextBox.Text)
25         price = Val(priceTextBox.Text)
26         interest = Val(interestTextBox.Text) / 100
```

**Figure 9.11** Retrieving input in the **Car Payment Calculator** app.

## Declaring Variables and Retrieving User Input (Cont.)

- The app computes the amount of the loan by subtracting the down payment from the price.
- These calculations need to occur only once, so they are placed before the Do While...Loop statement (Fig. 9.12).

```
        interest = Val(interestTextBox.Text) / 100

        ' determine amount borrowed and monthly interest rate
        loanAmount = price - downPayment
        monthlyInterest = interest / 12

```

**Figure 9.12** Determining amount borrowed and monthly interest rate.

## Calculating the Monthly Payment Amounts with a Do While...Loop Repetition Statement

- After you type line 33 and press Enter, the IDE will close the repetition statement by automatically adding the keyword Loop in line 35 (Fig. 9.13).

```
31
32          ' calculate payments for two, three, four and five year loans
33          Do While years <= 5
34
35          Loop
```

**Figure 9.13** Do While...Loop to calculate payments.

## Calculating the Monthly Payment Amounts with a Do While…Loop Repetition Statement (Cont.)

- This loop is an example of **counter-controlled repetition**.
- This uses a counter (years) to control the number of times that a set of statements executes.
- Counter-controlled repetition also is called **definite repetition**, because the number of repetitions is known before the repetition statement begins.

## Calculating the Monthly Payment Amounts with a Do While…Loop Repetition Statement (Cont.)

- The number of months changes with each iteration of this loop, and the calculation result changes based on the length of the payment period (Fig. 9.14).

```
' calculate payments for two, three, four and five year loans
Do While years <= 5
   ' calculate payment period
   months = 12 * years
Loop
```

**Figure 9.14** Converting the loan duration from years to months.

## Calculating the Monthly Payment Amounts with a Do While…Loop Repetition Statement (Cont.)

- The built-in Visual Basic function Pmt returns a Double value that specifies the monthly payment amount on a loan for a constant interest rate (monthlyInterest) and a given time period (months) (Fig. 9.15).
- Function Pmt's third argument—the amount borrowed in this example—is a negative value if it represents cash to be paid (as in this app) and a positive value if it represents cash to be received.

```
Do While years <= 5
   ' calculate payment period
   months = 12 * years

   ' calculate monthly payment using Pmt
   monthlyPayment =
      Pmt(monthlyInterest, months, -loanAmount)
Loop
```

**Figure 9.15** Pmt function returns monthly payment.

Amounts with a Do While…Loop Repetition Statement (Cont.)

- The number of monthly payments and the monthly payment amounts are displayed beneath the header in the ListBox.
- String.Format is used to display monthlyPayment in currency format (Fig. 9.16).

```
         ' calculate monthly payment using Pmt
         monthlyPayment =
             Pmt(monthlyInterest, months, -loanAmount)

         ' display payment value
         paymentsListBox.Items.Add(months & ControlChars.Tab &
             ControlChars.Tab & String.Format("{0:C}",
             monthlyPayment))
     Loop
```

**Figure 9.16** Displaying the number of months and the amount of each monthly payment.

## Amounts with a Do While…Loop Repetition Statement (Cont.)

- The counter variable years is incremented in each iteration until it equals 6 (Fig. 9.17).
- Then the loop-continuation condition (years <= 5) evaluates to False and the repetition ends.

```
               ' display payment value
               paymentsListBox.Items.Add(months & ControlChars.Tab &
                  ControlChars.Tab & String.Format("{0:C}",
                  monthlyPayment))

               years += 1 ' increment counter
            Loop
```

**Figure 9.17** Incrementing the counter.

- Figure 9.18 presents the source code for the app.

```
Public Class CarPaymentCalculatorForm
   ' handles Calculate Button's Click event
   Private Sub calculateButton_Click(sender As System.Object,
      e As System.EventArgs) Handles calculateButton.Click

      Dim years As Integer = 2 ' repetition counter
      Dim months As Integer = 0 ' payment period
      Dim price As Decimal = 0 ' car price
      Dim downPayment As Decimal = 0 ' down payment
      Dim interest As Double = 0 ' interest rate
      Dim monthlyPayment As Decimal = 0 ' monthly payment
      Dim loanAmount As Decimal = 0 ' cost after down payment
      Dim monthlyInterest As Double = 0 ' monthly interest rate

      ' remove text displayed in ListBox
      paymentsListBox.Items.Clear()

      ' add header to ListBox
      paymentsListBox.Items.Add("Months" & ControlChars.Tab &
         ControlChars.Tab & "Monthly Payments")

```

Clear the ListBox

Add a header to the ListBox

Figure 9.18 **Car Payment Calculator** app code. (Part 1 of 3.)

```
         ' retrieve user input and assign values
         ' to their respective variables
         downPayment = Val(downPaymentTextBox.Text)
         price = Val(priceTextBox.Text)
         interest = Val(interestTextBox.Text) / 100

         ' determine amount borrowed and monthly interest rate
         loanAmount = price - downPayment
         monthlyInterest = interest / 12

```

**Figure 9.18** **Car Payment Calculator** app code. (Part 2 of 3.)

o While...Loop repeats its body while years s less than or equal to 5

:alculate term in months

Calculate monthly payment amount by using Pmt function

Display number of months and monthly payment amount

Increment counter yearsto prepare to calculate the monthly payment

```
      ' calculate payments for two, three, four and five year loans
      Do While years <= 5
         ' calculate payment period
         months = 12 * years

         ' calculate monthly payment using Pmt
         monthlyPayment =
            Pmt(monthlyInterest, months, -loanAmount)

         ' display payment value
         paymentsListBox.Items.Add(months & ControlChars.Tab &
            ControlChars.Tab & String.Format("{0:C}",
            monthlyPayment))

         years += 1 ' increment counter
      Loop
   End Sub ' calculateButton_Click
End Class ' CarPaymentCalculatorForm
```

**Figure 9.18** **Car Payment Calculator** app code. (Part 3 of 3.)