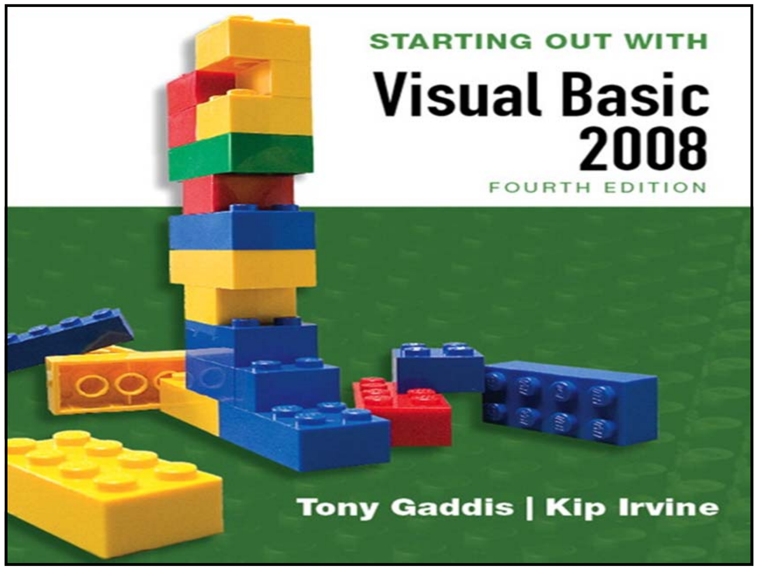

STARTING OUT WITH

Visual Basic 2008

FOURTH EDITION

Tony Gaddis | Kip Irvine

Chapter 4

Making Decisions and Working With Strings

Introduction

- This chapter covers the Visual Basic decision statements
  - **`If…Then`**
  - **`If…Then…Else`**
  - **`If…Then…ElseIf`**
  - **`Select Case`**
- It also discusses the use of
  - Radio Buttons
  - Message Boxes

4.1 The Decision Structure

The Decision Structure Allows a Program to Have More Than One Path of Execution

Order of Statement Execution

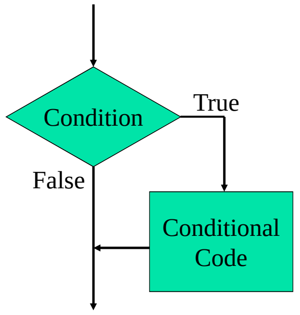
- Thus far, our code has been executed sequentially in a *sequence structure*
- To write meaningful programs we need multiple paths of execution
  - Some statements should be executed under certain circumstances in a *decision structure*
  - This chapter presents the means to execute statements conditionally
  - Next chapter presents the means to execute the same statements repeatedly

The Decision Structure

- Flowchart of a typical decision structure
- Evaluate the condition

  Is it cold outside?
- Execute or skip over some code

  If yes, wear a coat



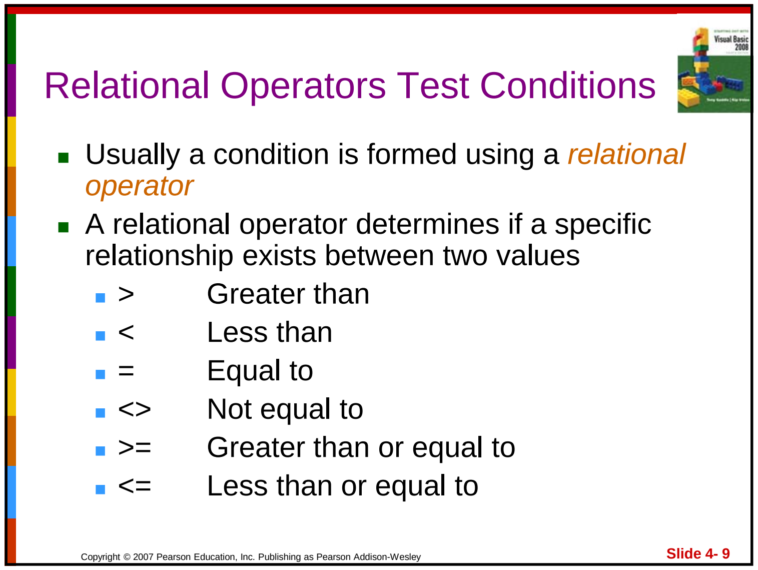
4.2 The If…Then Statement

The If…Then Statement Causes Other Statements to Execute Only Under a Certain Condition

If…Then Statement Syntax

```
If condition Then
     statement
     (more statements as needed)
End If
```

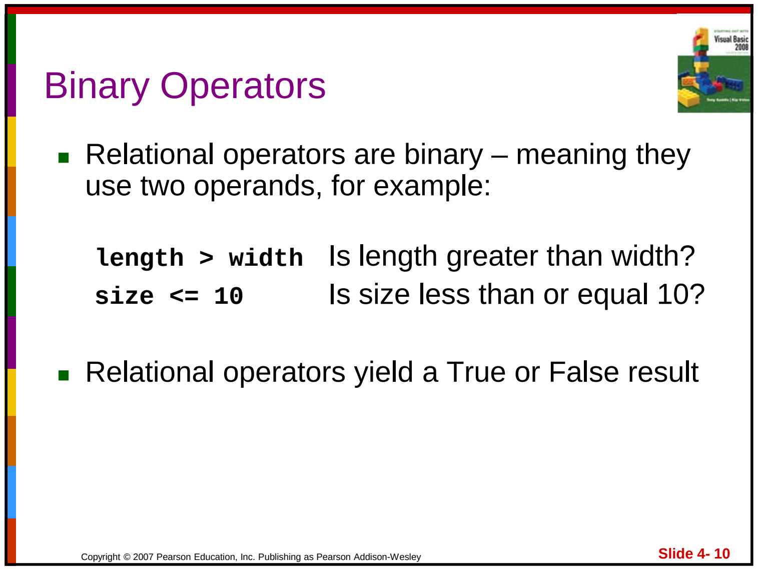
- New keywords used above:
  - **If**
  - **Then**
  - **End**



Relational Operators Test Conditions

- Usually a condition is formed using a *relational operator*
- A relational operator determines if a specific relationship exists between two values

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

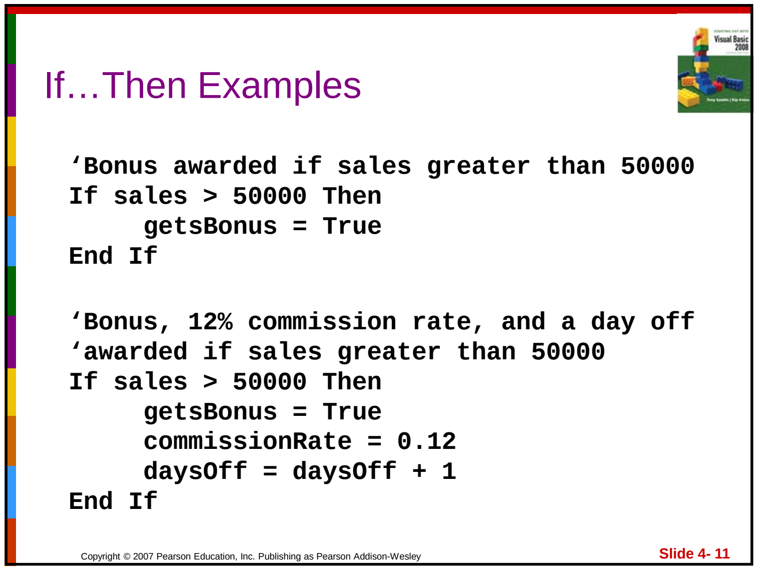


Binary Operators

- Relational operators are binary – meaning they use two operands, for example:

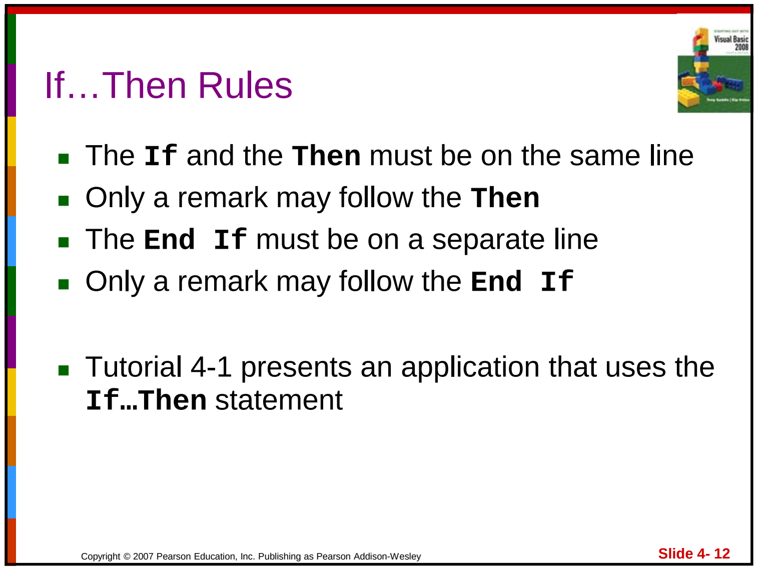
`length > width`	Is length greater than width?
`size <= 10`	Is size less than or equal 10?

- Relational operators yield a True or False result



If…Then Examples

```
'Bonus awarded if sales greater than 50000
If sales > 50000 Then
     getsBonus = True
End If

'Bonus, 12% commission rate, and a day off
'awarded if sales greater than 50000
If sales > 50000 Then
     getsBonus = True
     commissionRate = 0.12
     daysOff = daysOff + 1
End If
```



If…Then Rules

- The **`If`** and the **`Then`** must be on the same line
- Only a remark may follow the **`Then`**
- The **`End If`** must be on a separate line
- Only a remark may follow the **`End If`**

- Tutorial 4-1 presents an application that uses the **`If…Then`** statement

If…Then Programming Style

- The code between the **If…Then** and the **End If** is indented
- Visual Basic does not require this
- It is a convention among programmers to aid in the readability of programs
- By default, the Visual Basic editor will automatically do this indentation as you enter your program

Relational Operators with Math Operators

- Either or both relational operator operands may be mathematical expressions
- Math operators are evaluated before relational operators

```
If x + y > a - b Then
     lblMessage.Text = "It is true!"
End If
```

- **x+y** and **a-b** are evaluated first
- Each result is then compared using the > operator

Relational Operators With Function Calls

- Either or both relational operator operands may be function calls

```
If CInt(txtInput.Text) < 100 Then
     lblMessage.Text = "It is true!"
End If
```

Boolean Variables as Flags

- A *flag* is a Boolean variable that signals when some condition exists in the program
- Since a Boolean variable is either True or False, it can be used as the condition of an If
  - Since a Boolean variable already evaluates to True or False, an operator is not required

```
If blnQuotaMet Then
      lblMessage.Text = "You have met your sales quota"
End If
```

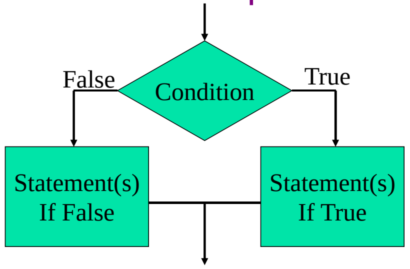
4.3 The If…Then…Else Statement

The If...Then...Else Statement Executes One Group of Statements If the Condition Is True and Another Group of Statements If the Condition Is False

If…Then vs. If…Then…Else

- The **If…Then** construct will execute or ignore a group of statements (do something or do nothing)
- The **If…Then…Else** construct will execute one group of statements or another group (do this or do that)
- Tutorial 4-2 contains an example of the **If…Then…Else** construct

If…Then…Else Example



```
If temperature < 40 Then
      lblMesage.Text = “A little cold, isn’t it?”
Else
      lblMesage.Text = “Nice weather we’re having!”
End If
```

4.4 The If…Then…ElseIf Statement

The If...Then…Elseif Statement Is Like a Chain of If...Then...Else Statements

They Perform Their Tests, One After the Other, Until One of Them Is Found to Be True

Two Mutually Exclusive Choices

- The **If…Then…Else** has two choices
  - The condition will either be True or False
  - So either the **Then** clause or **Else** clause will be executed
  - These are two mutually exclusive choices

Multiple Possible Choices

- The **If…Then…ElseIf** statement allows for an entire series of possible choices
- In pseudo code:

```
If it is very cold Then
     Wear a coat
Elseif it is chilly
     Wear a light jacket
Elseif it is windy
     Wear a windbreaker
Elseif it is hot
     Wear no jacket
```

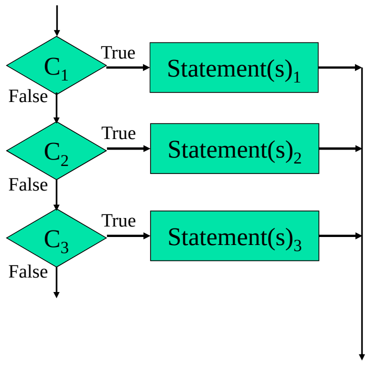
Multiple Possible Choices

- Each of the series of conditions in an **If…Then…ElseIf** is tested in sequence
- When a condition is true, the remaining conditions are ignored
- The order of the conditions is vital
  - Wrong order can result in wrong decision
  - What if it's chilly ***and*** windy?
  - If windy is tested before chilly, you'd go out with a windbreaker when you need a jacket

In Visual Basic Syntax

```
If condition1 Then
     Statement(s)1
Elseif condition2 Then
     Statements(s)2
Elseif condition3 Then
     Statements3
…
End If
```

In Flowchart Form



Example of ElseIf Usage

```
If sngAverage < 60 Then
      lblGrade.Text = "F"
ElseIf sngAverage < 70 Then
      lblGrade.Text = "D"
ElseIf sngAverage < 80 Then
      lblGrade.Text = "C"
ElseIf sngAverage < 90 Then
      lblGrade.Text = "B"
ElseIf sngAverage <= 100 Then
      lblGrade.Text = "A"
End If
```

- Does the order of these conditions matter?
- What happens if we reverse the order?

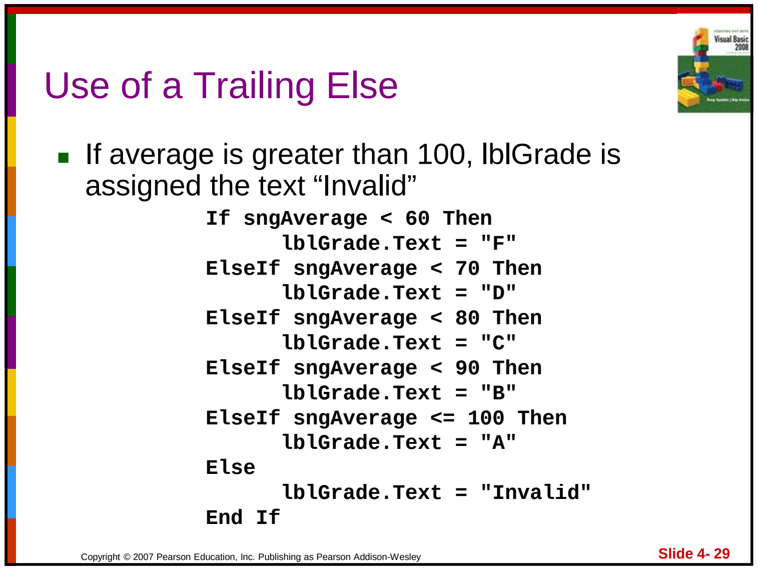
The Same Code Without ElseIf

```
If sngAverage < 60 Then
      lblGrade.Text = "F"
End If
If sngAverage < 70 Then
      lblGrade.Text = "D"
End If
If sngAverage < 80 Then
      lblGrade.Text = "C"
End If
If sngAverage < 90 Then
      lblGrade.Text = "B"
End If
If sngAverage <= 100 Then
      lblGrade.Text = "A"
End If
```

- Does this code function correctly? What is assigned to lblGrade for a 65 average? 75?

The (Optional) Trailing Else

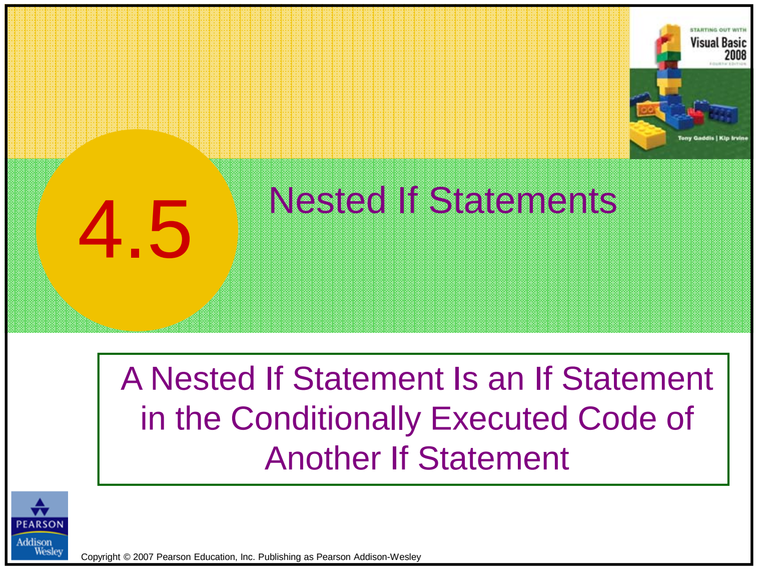
- A sequence of ElseIf's may end with a plain Else, called a *trailing Else*
- If none of the conditions are True, the trailing Else statement(s) will be executed



Use of a Trailing Else

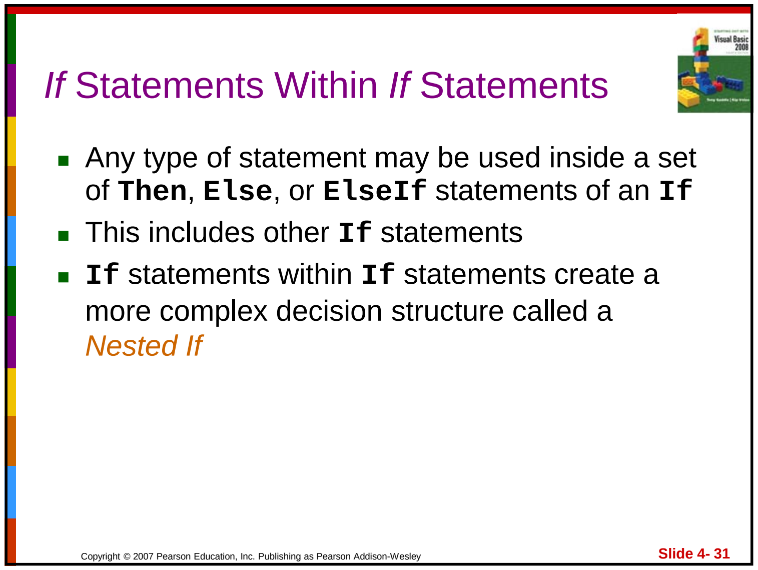
- If average is greater than 100, lblGrade is assigned the text “Invalid”

```
If sngAverage < 60 Then
      lblGrade.Text = "F"
ElseIf sngAverage < 70 Then
      lblGrade.Text = "D"
ElseIf sngAverage < 80 Then
      lblGrade.Text = "C"
ElseIf sngAverage < 90 Then
      lblGrade.Text = "B"
ElseIf sngAverage <= 100 Then
      lblGrade.Text = "A"
Else
      lblGrade.Text = "Invalid"
End If
```



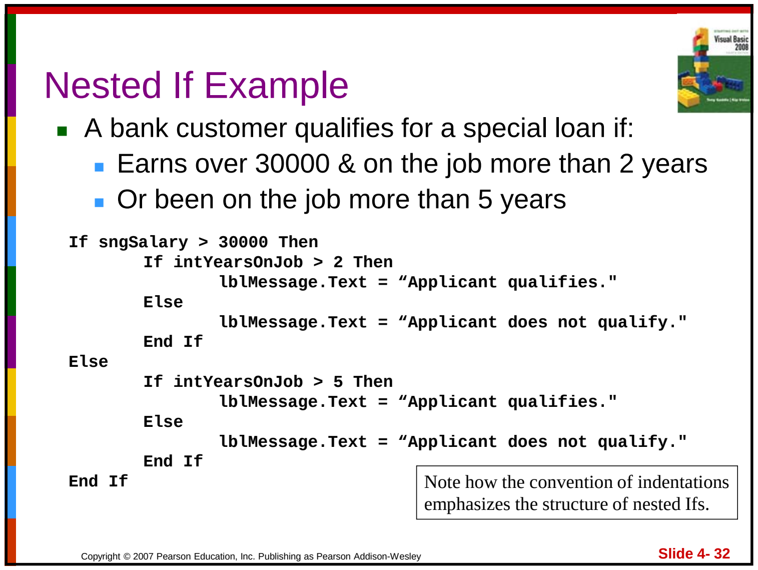
4.5 Nested If Statements

A Nested If Statement Is an If Statement in the Conditionally Executed Code of Another If Statement



If Statements Within *If* Statements

- Any type of statement may be used inside a set of **Then**, **Else**, or **ElseIf** statements of an **If**
- This includes other **If** statements
- **If** statements within **If** statements create a more complex decision structure called a *Nested If*



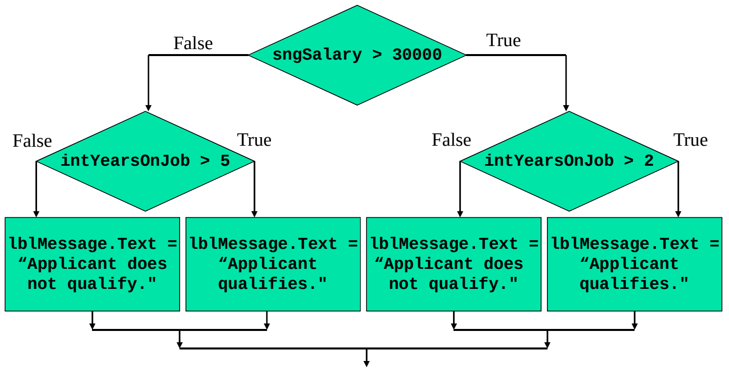
Nested If Example

- A bank customer qualifies for a special loan if:
  - Earns over 30000 & on the job more than 2 years
  - Or been on the job more than 5 years

```
If sngSalary > 30000 Then
      If intYearsOnJob > 2 Then
            lblMessage.Text = “Applicant qualifies."
      Else
            lblMessage.Text = “Applicant does not qualify."
      End If
Else
      If intYearsOnJob > 5 Then
            lblMessage.Text = “Applicant qualifies."
      Else
            lblMessage.Text = “Applicant does not qualify."
      End If
End If
```

Note how the convention of indentations emphasizes the structure of nested Ifs.

Flowchart Version



sngSalary > 30000
False
True
intYearsOnJob > 5
False
True
intYearsOnJob > 2
False
True
lblMessage.Text = “Applicant does not qualify."
lblMessage.Text = “Applicant qualifies."
lblMessage.Text = “Applicant does not qualify."
lblMessage.Text = “Applicant qualifies."

4.6 Logical Operators

Logical Operators Combine Two or More Relational Expressions Into a Single Expression

Visual Basic Logical Operators

Operator	Effect
And	Both operands must be true for the overall expression to be true, otherwise it is false
Or	One or both operands must be true for the overall expression to be true, otherwise it is false
Xor	One operand (but not both) must be true for the overall expression to be true, otherwise it is false
Not	Reverses the logical value of an expression

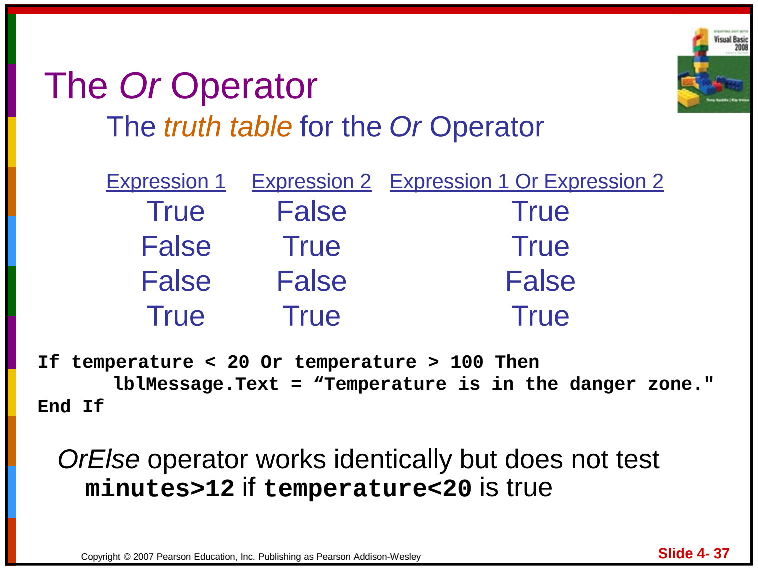
The *And* Operator

The *truth table* for the *And* Operator

Expression 1	Expression 2	Expression 1 And Expression 2
True	False	False
False	True	False
False	False	False
True	True	True

```
If temperature < 20 And minutes > 12 Then
    lblMessage.Text = “Temperature is in the danger zone."
End If
```

AndAlso operator works identically but does not test **`minutes>12`** if **`temperature<20`** is false



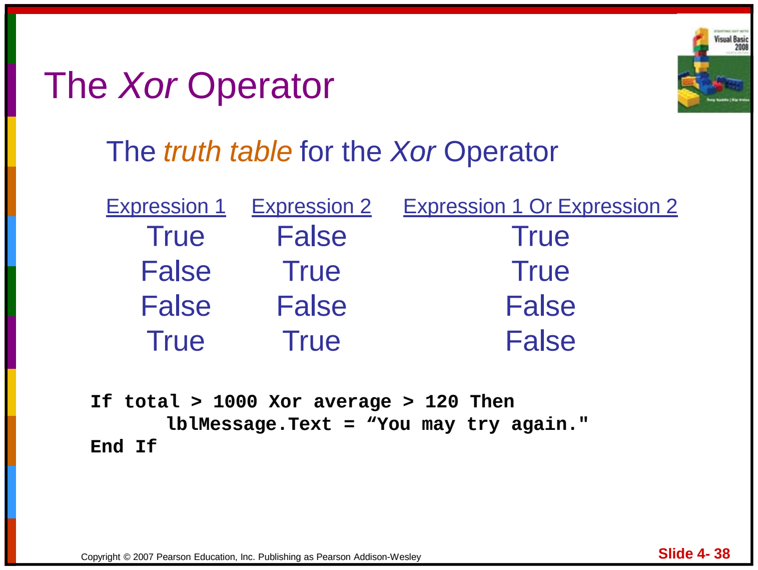
The *Or* Operator

The *truth table* for the *Or* Operator

Expression 1	Expression 2	Expression 1 Or Expression 2
True	False	True
False	True	True
False	False	False
True	True	True

```
If temperature < 20 Or temperature > 100 Then
     lblMessage.Text = “Temperature is in the danger zone."
End If
```

OrElse operator works identically but does not test **`minutes>12`** if **`temperature<20`** is true

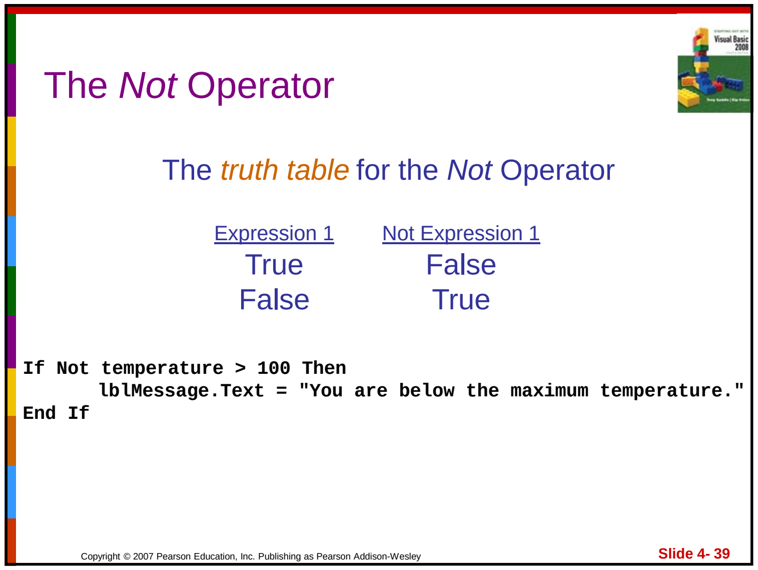


The *Xor* Operator

The *truth table* for the *Xor* Operator

Expression 1	Expression 2	Expression 1 Or Expression 2
True	False	True
False	True	True
False	False	False
True	True	False

```
If total > 1000 Xor average > 120 Then
     lblMessage.Text = “You may try again."
End If
```

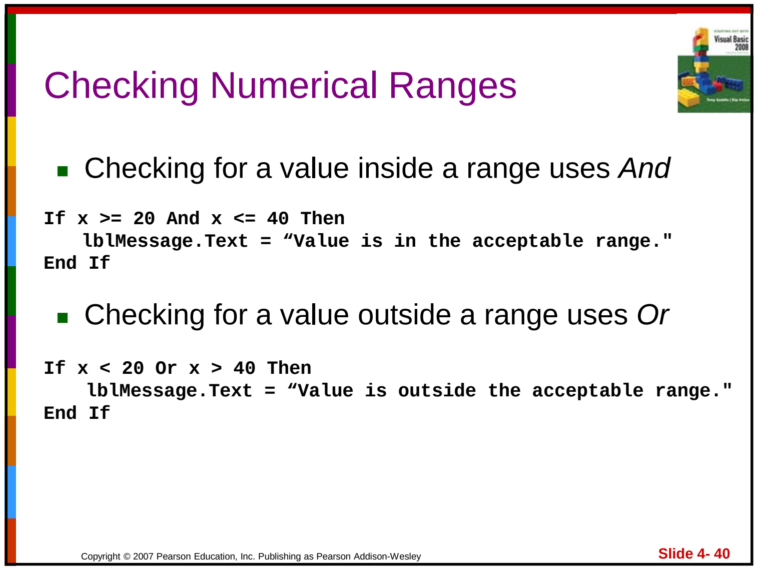


The *Not* Operator

The *truth table* for the *Not* Operator

Expression 1	Not Expression 1
True	False
False	True

```
If Not temperature > 100 Then
     lblMessage.Text = "You are below the maximum temperature."
End If
```



Checking Numerical Ranges

- Checking for a value inside a range uses *And*

```
If x >= 20 And x <= 40 Then
   lblMessage.Text = “Value is in the acceptable range."
End If
```

- Checking for a value outside a range uses *Or*

```
If x < 20 Or x > 40 Then
   lblMessage.Text = “Value is outside the acceptable range."
End If
```

Precedence of Logical Operators

- Logical operators have an order of precedence just as arithmetic operators do
- From highest to lowest precedence
  - Not
  - And
  - Or
  - Xor
- As with arithmetic operations, parentheses are often used to clarify order of operations

Precedence of Logical Operators

- For example, in the statement
  - `If x < 0 And y > 100 Or z = 50`
  - `x < 0 And y > 100` is evaluated first
  - If the And condition is true, we then evaluate
  - `True Or z = 50`
  - If the And condition is false, we then evaluate
  - `False Or z = 50`
- If the Or condition is to be evaluated first parentheses must be used
  - `If x < 0 And (y > 100 Or z = 50)`

Math, Relational, & Logical Operators

- Evaluate the following if a=5, b=7, x=100, y=30

  `If x > a * 10 And y < b + 20`

  Evaluating the math operators leaves us with

  `If x > 50 And y < 27`

  Evaluating the relational operators leaves

  `If True And False`

  Evaluating the logical operators leaves

  `False`
- Parentheses make order of operations clear

  `If (x > (a * 10)) And (y < (b + 20))`

4.7 Comparing, Testing, and Working With Strings

This Section Shows You How to Use Relational Operators to Compare Strings, and Discusses Several Intrinsic Functions That Perform Tests and Manipulations on Strings

Strings Can Be Compared

- Relational operators can be used to compare strings and string literals as well as numbers

```
strName1 = "Mary"
strName2 = "Mark"
If strName1 = strName2 Then
      lblMessage.Text = “Names are the same"
Else
      lblMessage.Text = “Names are NOT the same"
End If
```

```
If strMonth <> "October" Then
     ' statement
End If
```

How Are Strings Compared?

- Each character is encoded as a numerical value using the *Unicode* standard
- Letters are arranged in alphabetic order
  - The Unicode numeric code for A is less than the Unicode numeric code for B
- Characters of each string are compared one by one until a difference is found
  - **M a r y**
  - **M a r k**

Mary is greater than Mark because “**y**” has a Unicode value greater than “**k**”

How Are Strings Compared?

- Upper case letters do ***not*** have the same value as their lower case equivalents
  - Upper case letters are less than lower case
- The >, <, >=, and <= operators can be used with strings as well
- If one string is shorter than another, spaces are substituted for the missing characters
- Spaces have a lower value than letters
  - “Hi” has 2 spaces added if compared to “High”
  - “Hi ” is less than “High”

The Empty String

- A space (or blank) is considered a character
- An empty string is a string with no characters
  - A string with just spaces ***has*** characters in it
- The empty string is written as "", as in the following code that tests for no input:

```
If txtInput.Text = "" Then
      lblMessage.Text = "Please enter a value"
End If
```

ToUpper Method

- *ToUpper* method can be applied to a string
- Results in a string with lowercase letters converted to uppercase
- The original string is not changed

```
littleWord = "Hello"
bigWord = littleWord.ToUpper()
      ' littleWord retains the value "Hello"
      ' bigWord is assigned the value "HELLO"
```

ToLower Method

- The *ToLower* method performs a similar but opposite purpose
- Can be applied to a string
- Results in a string with the lowercase letters converted to uppercase
- The original string is not changed

```
bigTown = “New York"
littleTown = bigTown.ToLower()
      ' bigTown retains the value “New York"
      ' littleTown is assigned the value “new york"
```

A Handy Use for ToUpper or ToLower

- *ToUpper* or *ToLower* can be used to perform case insensitive comparisons of strings
- 1st comparison below is false “Hello”<>“hello”
- 2nd comparison is true
  - ToLower converts both strings to lower case
  - Causes “hello” to be compared to “hello”

```
word1 = "Hello“
Word2 = “hello”
If word1 = word2                      ‘false, not equal
If word1.ToLower() = word2.ToLower() ‘true, equal
```

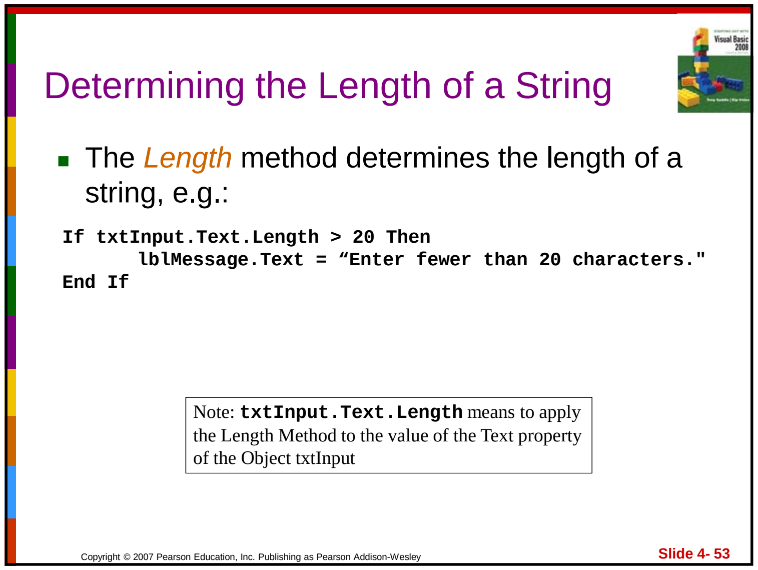
- Tutorial 4-6 demonstrates how this is used

IsNumeric Function

- This function accepts a string as an argument and returns True if the string contains a number

```
Dim strNumber as String
strNumber = “576”
If IsNumeric(strNumber)    ‘returns true
strNumber = “123abc”
If IsNumeric(strNumber)    ‘returns false
```

- Use *IsNumeric* function to determine if a given string contains numeric data

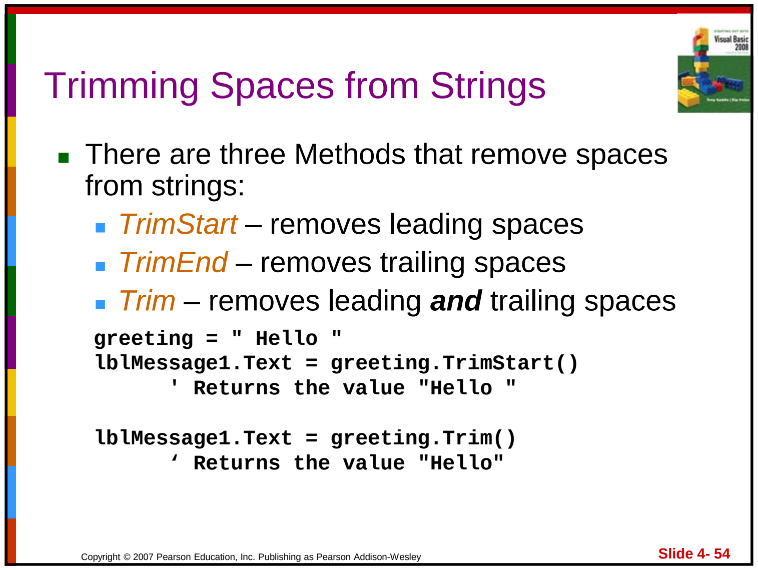


Determining the Length of a String

- The *Length* method determines the length of a string, e.g.:

```
If txtInput.Text.Length > 20 Then
      lblMessage.Text = “Enter fewer than 20 characters."
End If
```

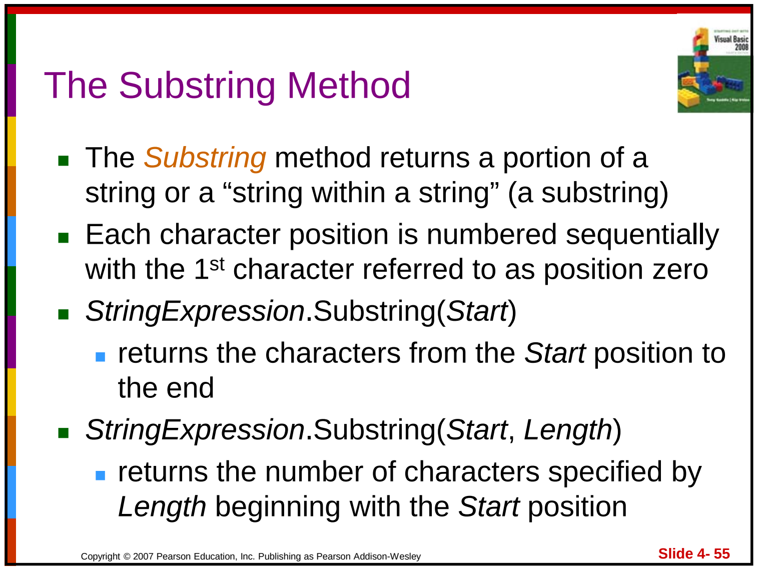
Note: **`txtInput.Text.Length`** means to apply the Length Method to the value of the Text property of the Object txtInput



Trimming Spaces from Strings

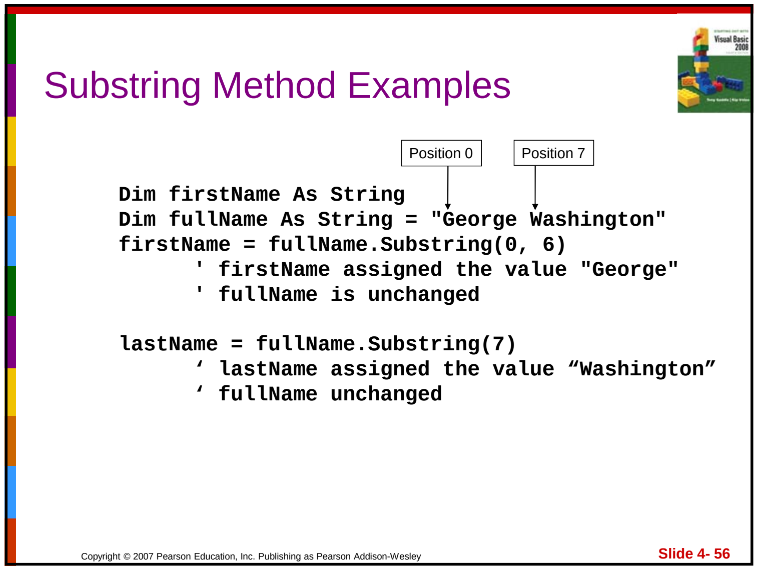
- There are three Methods that remove spaces from strings:
  - *TrimStart* – removes leading spaces
  - *TrimEnd* – removes trailing spaces
  - *Trim* – removes leading ***and*** trailing spaces

```
greeting = " Hello "
lblMessage1.Text = greeting.TrimStart()
      ' Returns the value "Hello "

lblMessage1.Text = greeting.Trim()
      ‘ Returns the value "Hello"
```



The Substring Method

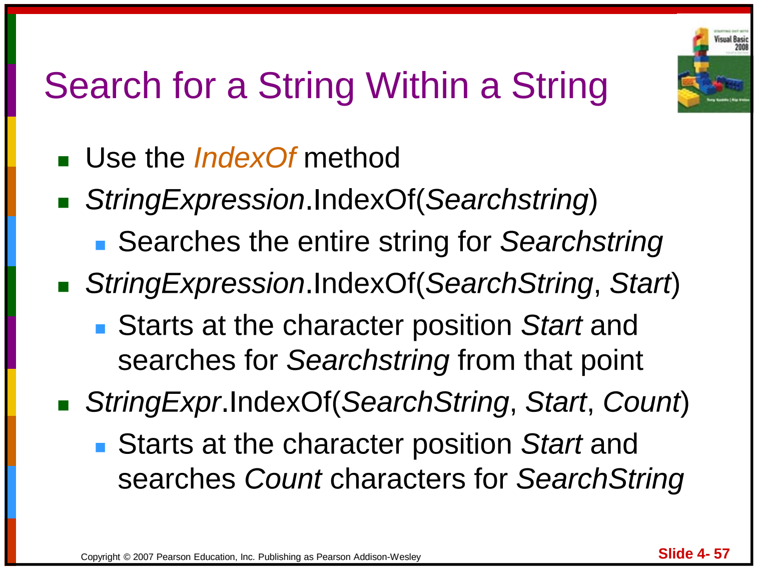
- The *Substring* method returns a portion of a string or a “string within a string” (a substring)
- Each character position is numbered sequentially with the 1st character referred to as position zero
- *StringExpression*.Substring(*Start*)
  - returns the characters from the *Start* position to the end
- *StringExpression*.Substring(*Start*, *Length*)
  - returns the number of characters specified by *Length* beginning with the *Start* position



Substring Method Examples

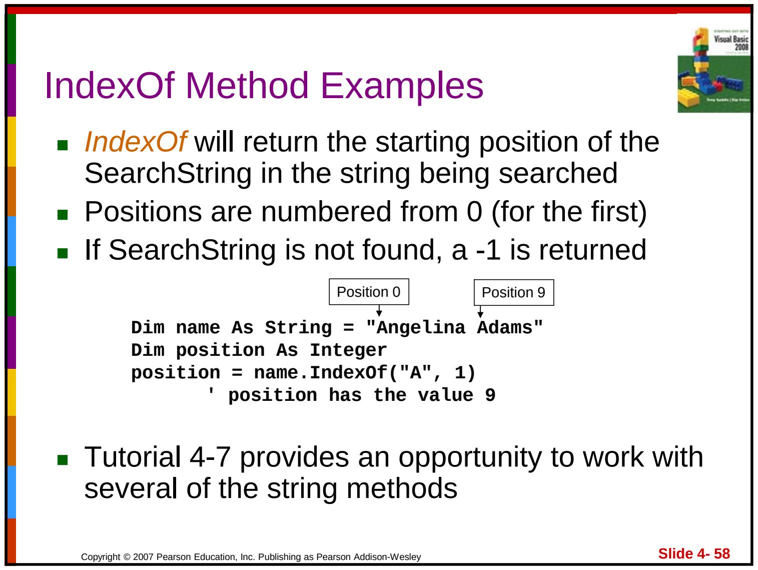
Position 0 Position 7

```
Dim firstName As String
Dim fullName As String = "George Washington"
firstName = fullName.Substring(0, 6)
      ' firstName assigned the value "George"
      ' fullName is unchanged

lastName = fullName.Substring(7)
      ‘ lastName assigned the value “Washington”
      ‘ fullName unchanged
```



Search for a String Within a String

- Use the *IndexOf* method
- *StringExpression*.IndexOf(*Searchstring*)
  - Searches the entire string for *Searchstring*
- *StringExpression*.IndexOf(*SearchString*, *Start*)
  - Starts at the character position *Start* and searches for *Searchstring* from that point
- *StringExpr*.IndexOf(*SearchString*, *Start*, *Count*)
  - Starts at the character position *Start* and searches *Count* characters for *SearchString*



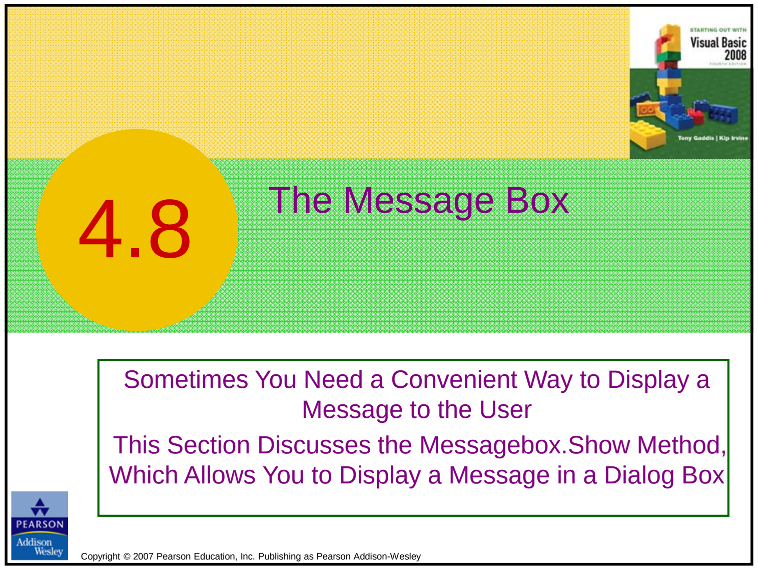
IndexOf Method Examples

- *IndexOf* will return the starting position of the SearchString in the string being searched
- Positions are numbered from 0 (for the first)
- If SearchString is not found, a -1 is returned

Position 0 Position 9

```
Dim name As String = "Angelina Adams"
Dim position As Integer
position = name.IndexOf("A", 1)
        ' position has the value 9
```

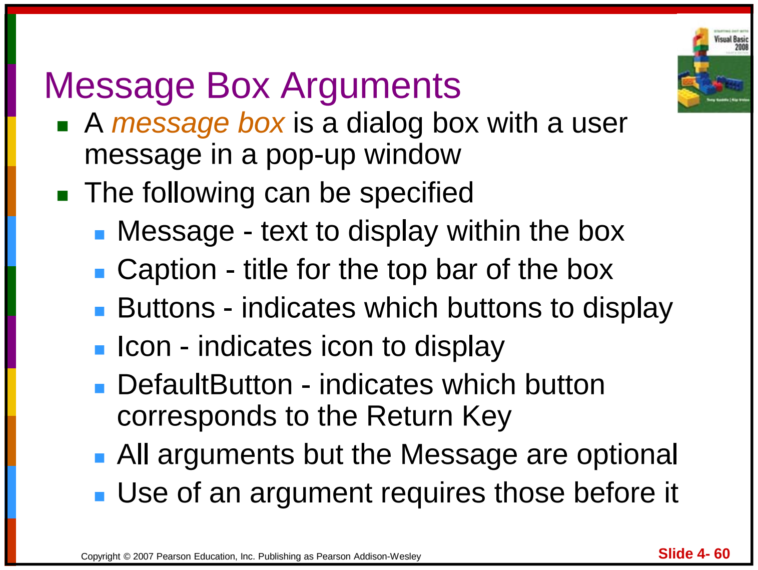
- Tutorial 4-7 provides an opportunity to work with several of the string methods



4.8 The Message Box

Sometimes You Need a Convenient Way to Display a Message to the User

This Section Discusses the Messagebox.Show Method, Which Allows You to Display a Message in a Dialog Box



Message Box Arguments

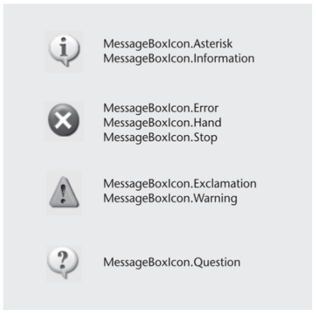
- A *message box* is a dialog box with a user message in a pop-up window
- The following can be specified
  - Message - text to display within the box
  - Caption - title for the top bar of the box
  - Buttons - indicates which buttons to display
  - Icon - indicates icon to display
  - DefaultButton - indicates which button corresponds to the Return Key
  - All arguments but the Message are optional
  - Use of an argument requires those before it

MessageBox Buttons Argument

MessageBoxButtons.AbortRetryIgnore
Displays Abort, Retry, and Ignore buttons
MessageBoxButtons.OK
Displays only an OK button
MessageBoxButtons.OKCancel
Displays OK and Cancel buttons
MessageBoxButtons.RetryCancel
Display Retry and Cancel buttons
MessageBoxButtons.YesNo
Displays Yes and No buttons
MessageBoxButtons.YesNoCancel
Displays Yes, No, and Cancel buttons

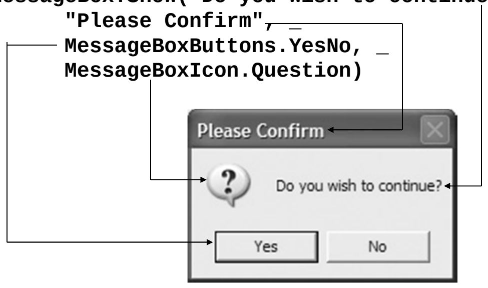
MessageBox Icon Argument

- The Icon argument specifies a particular type of icon to appear in the message box
- There are 4 possible icons shown to the left
- Note that some values show the same icon



Example Message Box

```
MessageBox.Show("Do you wish to continue?", _
          "Please Confirm", _
          MessageBoxButtons.YesNo, _
          MessageBoxIcon.Question)
```



Which Button Was Clicked

- *MessageBox* returns a value indicating which button the user clicked:
  - DialogResult.Abort
  - DialogResult.Cancel
  - DialogResult.Ignore
  - DialogResult.No
  - DialogResult.OK
  - DialogResult.Retry
  - DialogResult.Yes

Which Button Was Clicked Example

```
Dim result As Integer
result = MessageBox.Show("Do you wish to continue?", _
      "Please Confirm", MessageBoxButtons.YesNo)

If result = DialogResult.Yes Then
      ' Perform an action here
ElseIf result = DialogResult.No Then
      ' Perform another action here
End If
```

4.9 The Select Case Statement

In a Select Case Statement, One of Several Possible Actions Is Taken, Depending on the Value of an Expression

Select Case Statement

- Similar to **`If…Then…ElseIf`**
  - Performs a series of tests
  - Conditionally executes the first true condition
- *Select Case* is different in that:
  - A single test expression may be evaluated
  - The test expression is listed once
  - The possible values of the expression are then listed with their conditional statements
- **`Case Else`** may be included and executed if none of the values match the expression

Find Day of Week With Select Case

```
Select Case CInt(txtInput.Text)
   Case 1
      MessageBox.Show("Day 1 is Monday.")
   Case 2
      MessageBox.Show("Day 2 is Tuesday.")
   Case 3
      MessageBox.Show("Day 3 is Wednesday.")
   Case 4
      MessageBox.Show("Day 4 is Thursday.")
   Case 5
      MessageBox.Show("Day 5 is Friday.")
   Case 6
      MessageBox.Show("Day 6 is Saturday.")
   Case 7
      MessageBox.Show("Day 7 is Sunday.")
   Case Else
      MessageBox.Show("That value is invalid.")
End Select
```

Select Case With Multiple Values

```
Select Case strAnimal
   Case "Dogs", "Cats"
      MessageBox.Show ("House Pets")
   Case "Cows", "Pigs", "Goats"
      MessageBox.Show ("Farm Animals")
   Case "Lions", "Tigers", "Bears"
      MessageBox.Show ("Oh My!")
End Select
```

Select Case with Operators

```
Select Case intScore
   Case Is >= 90
      strGrade = “A”
   Case 80 to 89
      strGrade = “B”
   Case 70 to 79
      strGrade = “C”
   Case 60 to 69
      strGrade = “D”
   Case 0 to 59
      strGrade = “F”
   Case Else
      MessageBox.Show(“Invalid Score”)
End Select
```

- Tutorial 4-8 demonstrates the Select Case

4.10 Introduction to Input Validation

Input Validation Is the Process of Inspecting Input Values and Determining Whether They Are Valid

Validation Example

- Output is only as good as the input
  - “Garbage in, garbage out”
- *Input validation* is the process of inspecting user input to see that it meets certain rules
- The **`TryParse`** method verifies that an input value is in a valid numeric or date format
- Decision structures are often used to validate input

The **`TryParse`** Method

- Converts an input value to another format
  - Verifies that input of integers, decimals, dates, etc., are entered in an acceptable format
  - Returns Boolean value indicating True if conversion successful
  - Returns False if unsuccessful
- Each numeric variable type has a TryParse method
- Date & Boolean types include the TryParse method as well

Verify Integer Entry With **`TryParse`**

- Use Integer.TryParse method to convert value
  - txtInput.Text contains numeric string to convert
  - intResult receives converted value
  - TryParse returns True if input is an integer
  - TryParse returns False if input is not an integer

```
Dim intResult As Integer
If Integer.TryParse(txtInput.Text, intResult) Then
  lblMessage.Text = "Success!"
Else
  lblMessage.Text = "Error: an integer was not found"
End If
```

Verify Date Entry With **`TryParse`**

- Use Date.TryParse method to convert value
  - txtInput.Text contains date string to convert
  - datBirth receives converted value
  - TryParse returns True if input in date format
  - TryParse returns False if input not a valid date
  - Not used so Then clause indicates invalid date

```
Dim datBirth As Date
If Not Date.TryParse(txtInput.Text, datBirth) Then
  lblMessage.Text = “Not a valid date!“
End If
```

Using **`If`** To Check Range of Values

- Decision structures often used to validate input
- Example verifies that entries for both decSales and decAdvance are positive numbers

```
' Validate the input to ensure that
' no negative numbers were entered.
If decSales < 0 Or decAdvance < 0 Then
   MessageBox.Show("Please enter positive numbers" & _
   " for sales and/or advance pay.“, “Error”)
EndIf
```

4.11 Radio Buttons and Check Boxes

Radio Buttons Appear in Groups of Two or More and Allow the User to Select One of Several Possible Options

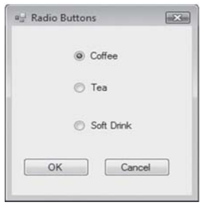
Check Boxes Allow an Item to Be Selected or Deselected by Checking or Unchecking a Box

Radio Buttons

- Used when only one of several possible options may be selected at one time
  - Car radio buttons select one station at a time
- May be placed in a group box
  - Group box defines a set of radio buttons
  - Can select only one button within a group box
  - Those on a form but not inside a group box are considered members of the same group
- Radio buttons have a boolean *Checked* property and a *CheckChanged* event

Checking Radio Buttons in Code

```
If radCoffee.Checked = True Then
       MessageBox.Show("You selected Coffee")
ElseIf radTea.Checked = True Then
       MessageBox.Show("You selected Tea")
ElseIf radSoftDrink.Checked = True Then
       MessageBox.Show("You selected a Soft Drink")
End If
```

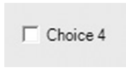


Check Boxes

- Unlike radio buttons, can select many check boxes at one time
- May also be placed in a group box
  - Not limited to one selection within a group box
  - Can select as many check boxes as you like within the same group box
- Check boxes also have a boolean *Checked* property and a *CheckChanged* event
- Tutorial 4-9 provides radio button and check box examples

Checking Check Boxes in Code

```
If chkChoice4.Checked = True Then
        MessageBox.Show("You selected Choice 4")
End If
```



4.12 Class-Level Variables

Class-level Variables Are Not Local to Any Procedure
In a Form They Are Declared Outside of Any Procedure, and May Be Accessed by Statements in Any Procedure in the Same Form

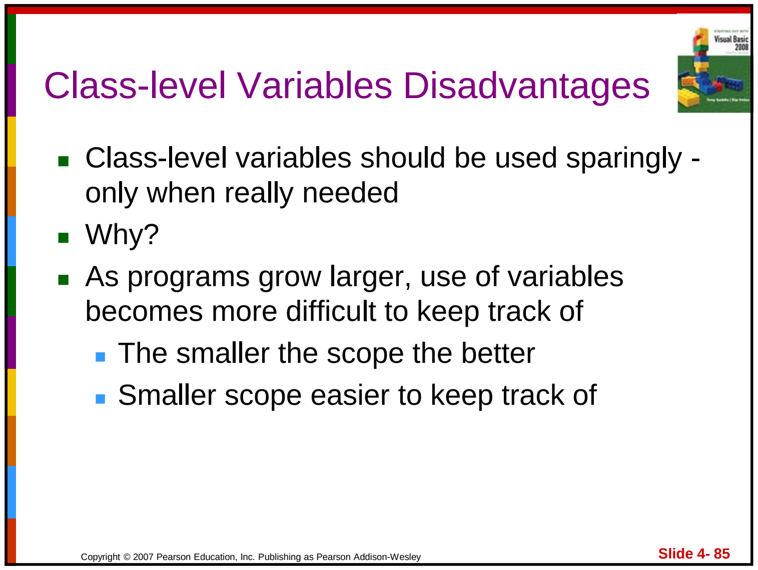
Advantages of Class-level Variables

- *Variable scope* refers to the portion of a program in which that variable is visible
- Variables declared inside a procedure or method have *local scope*
  - Only visible inside that procedure or method
- Sometimes a variable needs to be visible to many procedures or methods within a form
- Variables declared outside a procedure but within a form have *class scope*
  - These are visible throughout the form
  - Makes communication between procedures easy

Declaring a Class-Level Variable

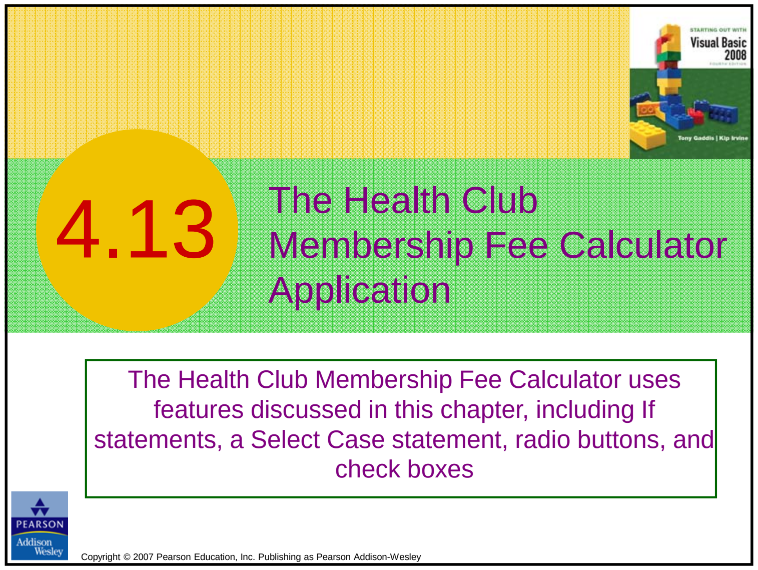
- decTotalSalary - class-level variable
  - Declared before first procedure in form class
- decWeeklyPay - local variable inside a procedure

```
Public Class Form1

Dim decTotalSalary As Decimal     ‘Class-level variable

Private Sub btnAddWeekly_Click(ByVal sender As System.Object,
  ByVal e As System.EventArgs) Handles btnAddWeekly.Click
      Dim decWeeklyPay As Decimal       'Local variable
      decWeeklyPay = CDec(txtPay.Text)
      decTotalSalary += decWeeklyPay
End Sub
```



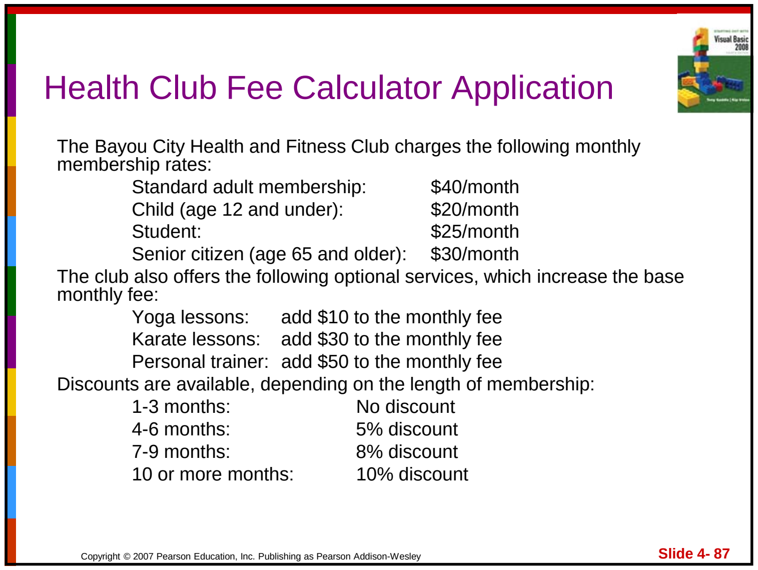
Class-level Variables Disadvantages

- Class-level variables should be used sparingly - only when really needed
- Why?
- As programs grow larger, use of variables becomes more difficult to keep track of
  - The smaller the scope the better
  - Smaller scope easier to keep track of



4.13 The Health Club Membership Fee Calculator Application

The Health Club Membership Fee Calculator uses features discussed in this chapter, including If statements, a Select Case statement, radio buttons, and check boxes



Health Club Fee Calculator Application

The Bayou City Health and Fitness Club charges the following monthly membership rates:

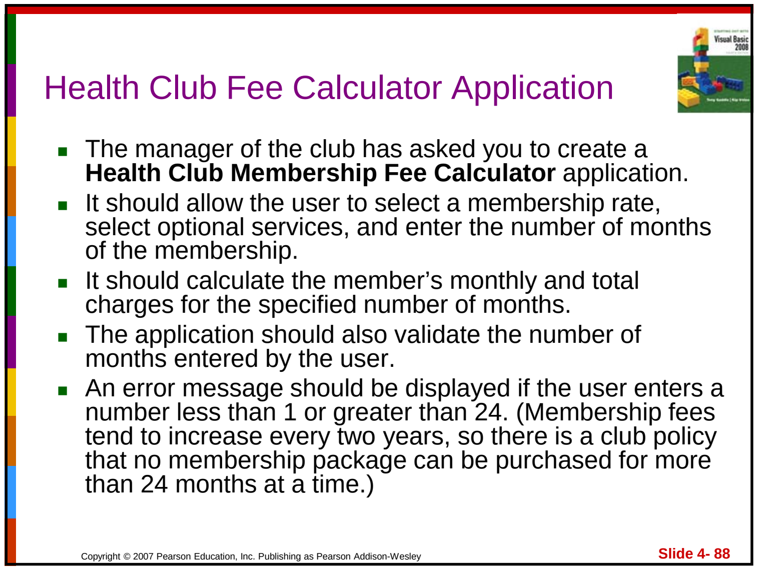
Standard adult membership:	$40/month
Child (age 12 and under):	$20/month
Student:	$25/month
Senior citizen (age 65 and older):	$30/month

The club also offers the following optional services, which increase the base monthly fee:

Yoga lessons:	add $10 to the monthly fee
Karate lessons:	add $30 to the monthly fee
Personal trainer:	add $50 to the monthly fee

Discounts are available, depending on the length of membership:

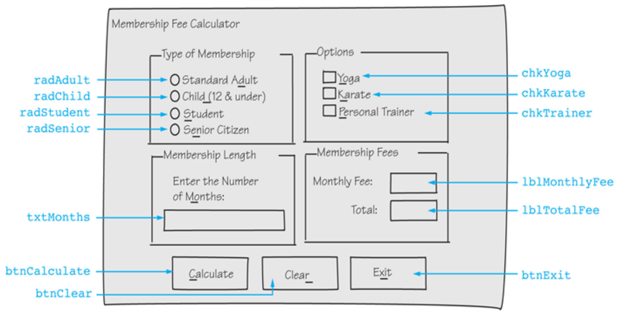
1-3 months:	No discount
4-6 months:	5% discount
7-9 months:	8% discount
10 or more months:	10% discount



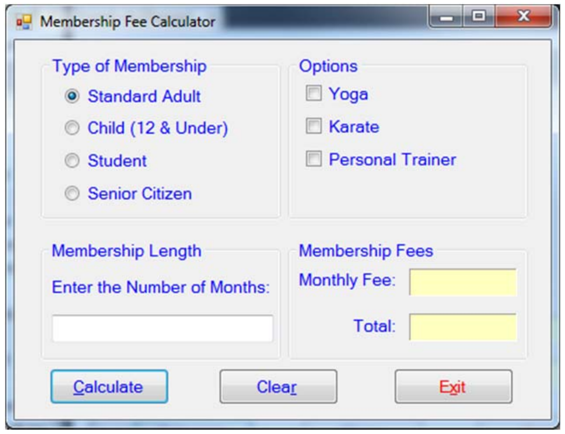
Health Club Fee Calculator Application

- The manager of the club has asked you to create a **Health Club Membership Fee Calculator** application.
- It should allow the user to select a membership rate, select optional services, and enter the number of months of the membership.
- It should calculate the member's monthly and total charges for the specified number of months.
- The application should also validate the number of months entered by the user.
- An error message should be displayed if the user enters a number less than 1 or greater than 24. (Membership fees tend to increase every two years, so there is a club policy that no membership package can be purchased for more than 24 months at a time.)

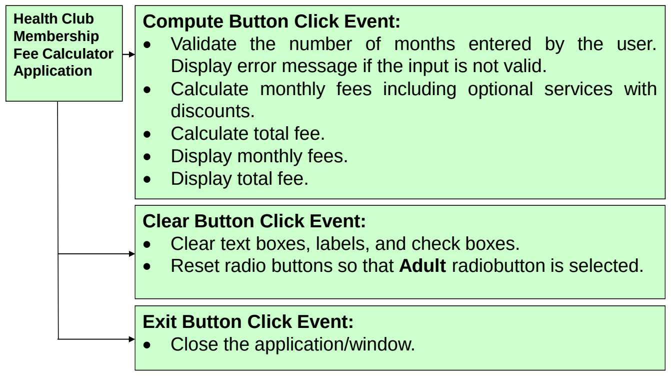
Health Club Fee Calculator Form



Health Club Fee Calculator Form



Algorithm Chart



Health Club Membership Fee Calculator Application

Compute Button Click Event:
- Validate the number of months entered by the user. Display error message if the input is not valid.
- Calculate monthly fees including optional services with discounts.
- Calculate total fee.
- Display monthly fees.
- Display total fee.

Clear Button Click Event:
- Clear text boxes, labels, and check boxes.
- Reset radio buttons so that **Adult** radiobutton is selected.

Exit Button Click Event:
- Close the application/window.

Compute Button Click Event:

- Validate the number of months entered by the user. Display error message if the input is not valid.
  - Check if the months entered is an integer:
```
If Months is not an integer then
    Display error message
End If
```
  - Check if the months entered is between 1 and 24 including 1 and 24:
```
If Months < 1 or Months > 24 then
    Display error message
End If
```

Compute Button Click Event:

- Calculate monthly fees including optional services with discounts.
  - Compute base fee:

```
If Adult radio button is selected then
        Fee = adult rate
Else If Child radio button is selected then
        Fee = Child rate
Else If Student radio button is selected then
        Fee = Student rate
Else If Senior radio button is selected then
        Fee = Senior rate
End If
```

Compute Button Click Event:

- Add options:

```
If Yoga check box is checked then
        Fee = Fee + Yoga
Else If Yoga check box is checked then
        Fee = Fee + Karate
Else If Yoga check box is checked then
        Fee = Fee + Trainer
End If
```

Compute Button Click Event:

- Compute discount based on # of months:

```
If months < 4 then
        Discount = 0
Else If months < 7 then
        Discount = Fee * .05
Else If months < 10 then
        Discount = Fee * .08
Else If months >= 10 then
        Discount = Fee * .1
End If
```

- Compute base fee with discount:

```
Fee = Fee – Discount
```

Compute Button Click Event:

- Calculate total fee.

```
Total = Fee * Months
```

- Display monthly fees.

```
FeeLable.Text = Fee.ToString("c")
```

- Display total fee.

```
Total.Text = Total.ToString("c")
```

Clear Button Click Event:

- Clear all text boxes,

  TextBoxName.Clear()
- Clear labels,

  LabeName.Text=""
- Clear check boxes.

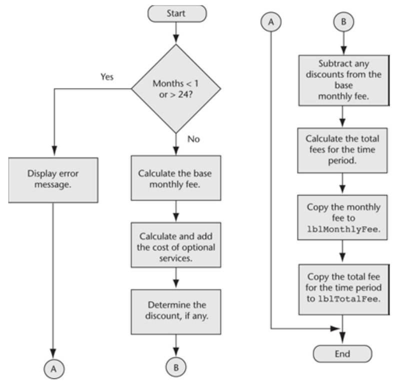
  CheckeBoxName.Checked = False
- Reset radio buttons so that Adult radiobutton is selected.

  RadioButtonName.Checked = True

Exit Button Click Event:

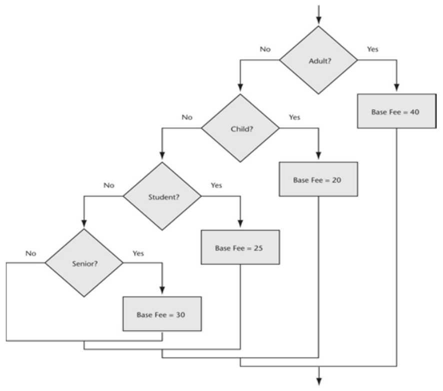
Close()

Calculate Button Click Event Flowchart


Start
Months < 1 or > 24?
Yes
No
Display error message.
Calculate the base monthly fee.
Calculate and add the cost of optional services.
Determine the discount, if any.
A
B
A
B
Subtract any discounts from the base monthly fee.
Calculate the total fees for the time period.
Copy the monthly fee to lblMonthlyFee.
Copy the total fee for the time period to lblTotalFee.
End

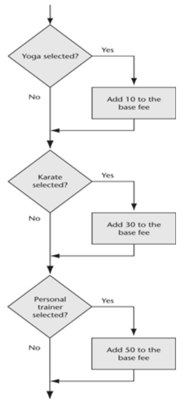
Base Monthly Fee Calculation Flowchart

Uses a series of **`ElseIf`** statements



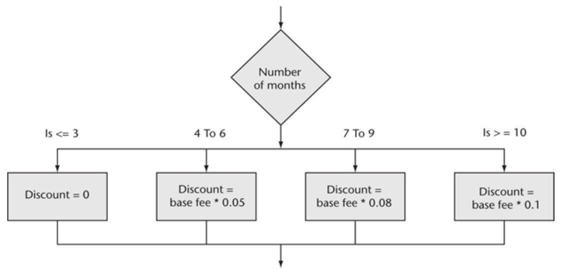
Calculate Optional Services Flowchart

Uses several **`If...Then`** statements



Compute Discount Flowchart

Uses a **`Select Case`** statement


Number of months
Is <= 3
4 To 6
7 To 9
Is > = 10
Discount = 0
Discount = base fee * 0.05
Discount = base fee * 0.08
Discount = base fee * 0.1