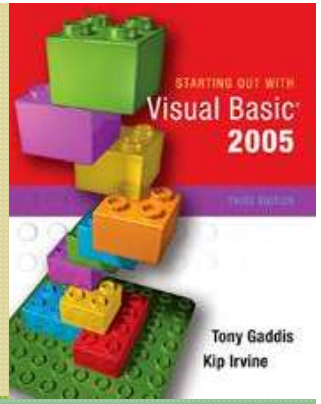
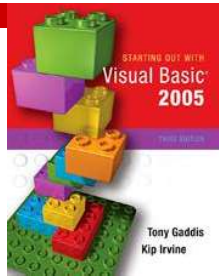



# 12.1 Classes and Objects

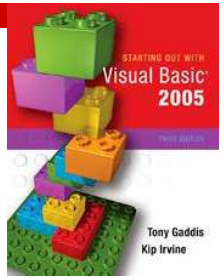
Classes Are Program Structures That Define Abstract Data Types and Are Used to Create Objects

# Abstract Data Types



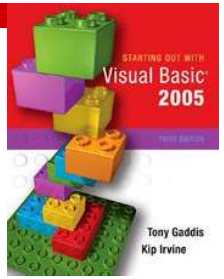

- An *abstract data type* (*ADT*) is a data type created by a programmer
- ADTs are important in computer science and object-oriented programming
- An *abstraction* is a model of something that includes only its general characteristics
- Dog is an abstraction
  - Defines a general type of animal but not a specific breed, color, or size
  - A dog is like a data type
  - A specific dog is an instance of the data type

# Classes



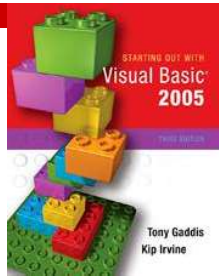

- A *class* is a program structure that defines an abstract data type
  - Must create the class first
  - Then can create instances of the class
- Class instances share common attributes
- VB forms and controls are classes
  - Each control in the toolbox, is its own class
  - When you place a button on a form you're creating an instance of the button class
  - An instance is also called an *object*

# Properties, Methods, and Events



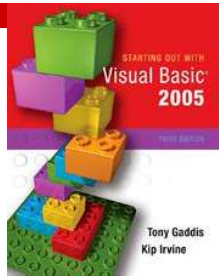

- Programs communicate with an object using the properties and methods of the class
- Class properties example: Buttons have Location, Text, and Name properties
- Class methods example: The Focus method functions identically for every single button
- Class event procedures: Each button in a form has a different click event procedure

# Object Oriented Design




- The challenge is to design classes that effectively cooperate and communicate
- Analyze program specifications to determine ADTs that best implement the specifications
- Classes are fundamental building blocks
  - Typically represent nouns of some type

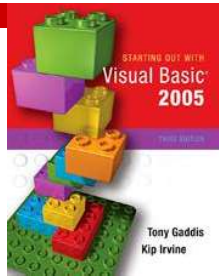
# Object Oriented Design Example




*Specifications:*
We need to keep a **list of students** that lets us track the courses they have completed. Each student has a **transcript** that contains all information about his or her completed courses. At the end of each semester, we will calculate the grade point average of each **student** . At times, users will search for a particular **course** taken by a student.

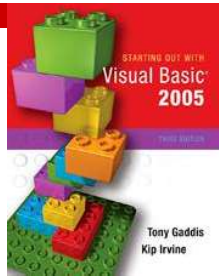
- Nouns from the specification above usually become classes in the program design
- Verbs such as *calculate* GPA and *search* become methods of those classes

# OOD Class Characteristics

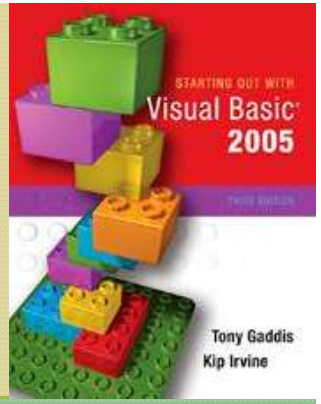



| Class | Attributes (properties) | Operations (methods) |
|---|---|---|
| Student | LastName, FirstName, IdNumber | Display, Input |
| StudentList | AllStudents, Count | Add, Remove, FindStudent |
| Course | Semester, Name, Grade,Credits | Display, Input |
| Transcript | CourseList, Count | Display, Search, CalcGradeAvg |

# Interface and Implementation




- *Class interface* is the portion of the class visible to the application programmer
- *Class implementation* is the portion of the class hidden from client programs
  - Private member variables
  - Private properties
  - Private methods
- Hiding of all data and procedures inside a class is referred to as *encapsulation*



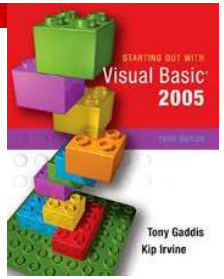

# 12.2 Creating a Class

To Create a Class in Visual Basic, You Create a Class Declaration

The Class Declaration Specifies the Member Variables, Properties, Methods, and Events That Belong to the Class

PEARSON Addison Wesley

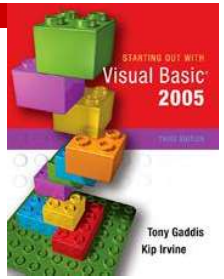
# Class Declaration




```
Public Class ClassName
        MemberDeclarations
End Class
```

- ClassName is the name of the class
- Examples of *MemberDeclarations* are presented in the following slides
- To create a new class:
  - Clicking Add New Item button on toolbar
  - Select Class from Add New Item dialog box
  - Provide a name for the class and click Add
  - Adds a new, empty class file (.vb) to project

# Member Variables



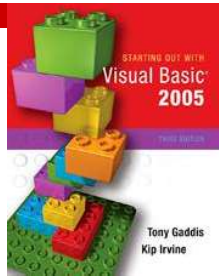

- A variable declared inside a class declaration
- Syntax:

  ***AccessSpecifer VariableName* As *DataType***

- AccessSpecifier may be Public or Private
- Example:

```
Public Class Student
   Public strLastName As String     'Student last name
   Public strFirstName As String    'Student first name
   Public strId As String           'Student ID number
End Class
```

# Creating an Instance of a Class




- A two step process creates a *class object*
- Declare a variable whose type is the class

```
Dim freshman As Student
```

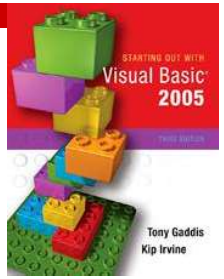
- Create instance of class with *New* keyword and assign the instance to the variable

```
freshman = New Student()
```

- *freshman* defined here as an *object variable*
- Can accomplish both steps in one statement

```
Dim freshman As New Student()
```

# Accessing Members



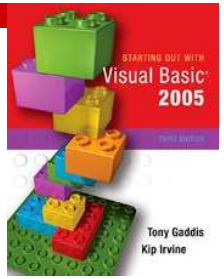

- Can work with Public member variables of a class object in code using this syntax:

  ***object.memberVariable***

- For example:
  - If *freshman* references a Student class object
  - And Student class has member variables *strFirstName*, *strLastName*, and *strID*
  - Can store values in member variables with

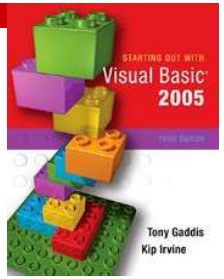
```
freshman.strFirstName = "Joy"
freshman.strLastName = "Robinson"
freshman.strId = "23G794"
```

# Property Procedure



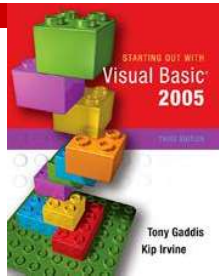

- A *property procedure* is a function that behaves like a property
- Controls access to property values
- Procedure has two sections: *Get* and *Set*
  - Get code executes when value is retrieved
  - Set code executes when value is stored
- Properties almost always declared Public to allow access from outside the class
- Set code often provides data validation logic

# Property Procedure Syntax




```
Public Property PropertyName() As DataType
        Get
                Statements
        End Get
        Set(ParameterDeclaration)
                Statements
        End Set
End Property
```

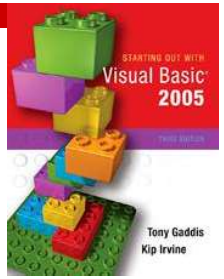
# Property Procedure Example




```
Public Class Student
    ' Member variables
    Private sngTestAvg As Single

    Public Property TestAverage() As Single
        Get
            Return sngTestAvg
        End Get
        Set(ByVal value As Single)
            If value >= 0.0 And value <= 100.0 Then
                sngTestAvg = value
            Else
                MessageBox.Show( _
                        "Invalid test average.", "Error")
            End If
        End Set
    End Property
End Class
```

# Setting and Validating a Property



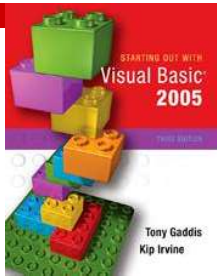

- TestAverage property is set as shown:

```
Dim freshman as New Student()
freshman.TestAverage = 82.3
```

- Passes 82.3 into value parameter of Set
  - If in the range 0.0 to 100.0, value is stored
  - If outside the range, message box displayed instead of value being stored

```
Set(ByVal value As Single)
        If value >= 0.0 And value <= 100.0 Then
                sngTestAvg = value
        Else
                MessageBox.Show( _
                "Invalid test average.", "Error")
        End If
End Set
```

# Read-Only Properties

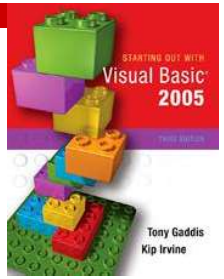



- Useful at times to make a property read-only
- Allows access to property values but cannot change these values from outside the class
- Use the *ReadOnly* keyword instead of *Public*

```
ReadOnly Property PropertyName() As DataType
    Get
        Statements
    End Get
End Property
```

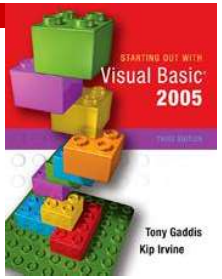
- This causes the propertyName to be read-only -- not settable from outside of the class

# Read-Only Property Example




```
' TestGrade property procedure
ReadOnly Property TestGrade() As Char
   Get
      If sngTestAverage >= 90
         return "A"
      Else If sngTestAverage >= 80
         return "B"
      Else If sngTestAverage >= 70
         return "C"
      Else If sngTestAverage >= 60
         return "D"
      Else
         return "F"
      End If
   End Get
End Property
```

# Object Removal & Garbage Collection

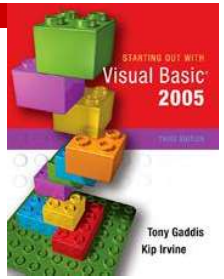



- Memory space is consumed when objects are instantiated
- Objects no longer needed should be removed
- Set object variable to *Nothing* so it no longer references the object

```
freshman = Nothing
```

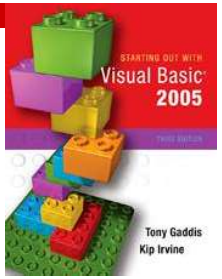
- Variable is a candidate for garbage collection when it no longer references an object
- The garbage collector runs periodically destroying objects no longer needed

# Going Out of Scope




- An *object variable* instantiated within a procedure is local to that procedure
- An object goes *out of scope* when
  - Referenced only by local variables and
  - The procedure ends
- Object removed once it goes out of scope
- An object instantiated in a procedure and assigned to a global variable is not removed
  - Reference remains when procedure ends

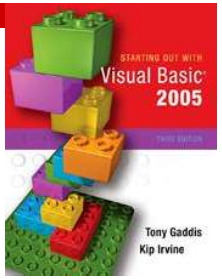
# Going Out of Scope, Example




```
Sub CreateStudent()
        Dim sophomore As Student
        sophomore = New Student()
        sophomore.FirstName = "Travis"
        sophomore.LastName = "Barnes"
        sophomore.IdNumber = "17H495"
        sophomore.TestAverage = 94.7
        g_studentVar = sophomore
End Sub
```

With this statement, *sophomore* will not go out of scope. Without this statement, it will go out of scope when the procedure ends. (*g_studentVar* is a module-level variable.)

# Comparing Object Variables



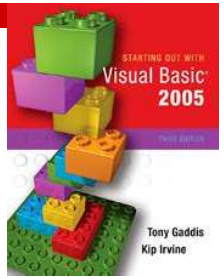

- Multiple variables can reference the same object

```
Dim collegeStudent As Student
Dim transferStudent As Student
collegeStudent = New Student()
transferStudent = collegeStudent
```

- Can test if two variables refer to same object
  - Must use the *Is* operator
  - The = operator cannot be used to test for this

```
If collegeStudent Is transferStudent Then
      ' Perform some action
End If
```

# IsNot & Nothing Object Comparisons




- Use the *IsNot* operator to determine that two variables do ***not*** reference the same object

```
If collegeStudent IsNot transferStudent Then
        ' Perform some action
End If
```

- Use the special value *Nothing* to determine if a variable has no object reference

```
If collegeStudent Is Nothing Then
        ' Perform some action
End If
```