Introduction to Classes and Objects
You will see something new. Two things. And I call them Thing One and Thing Two.

– Dr. Theodor Seuss Geisel

Nothing can have value without being an object of utility.

– Karl Marx
Your public servants serve you right.
   – Adlai E. Stevenson

Knowing how to answer one who speaks,
To reply to one who sends a message.
   – Amenemope
OBJECTIVES

In this chapter you will learn:

- What classes, objects, methods, instance variables and properties are.
- How to declare a class and use it to create an object.
- How to implement a class’s behaviors as methods.
- How to implement a class’s attributes as instance variables and properties.
- How to call an object’s methods to make them perform their tasks.
OBJECTIVES

- The differences between instance variables of a class and local variables of a method.
- How to use a constructor to ensure that an object’s attributes are initialized when the object is created.
- The differences between value types and reference types.
- How to use properties to ensure that only valid data is placed in attributes.
4.1 Introduction

4.2 Classes, Objects, Methods and Instance Variables

4.3 Declaring a Class with a Method and Instantiating an Object of a Class

4.4 Declaring a Method with a Parameter

4.5 Instance Variables and Properties

4.6 Value Types and Reference Types

4.7 Initializing Objects with Constructors

4.8 Validating Data with set Accessors in Properties

4.9 (Optional) Software Engineering Case Study: Identifying the Classes in the ATM Requirements Document
4.2 Classes, Objects, Methods and Instance Variables

• In Visual Basic, a program unit called a class houses one or more methods.
  – Performing a task in a program requires a method.

• Just as a car is built from engineering drawings, you build an object from a class.

• A car receives messages, such as pressure on the brake pedal, telling it to internally perform complex actions.

• Cars have individual attributes such as color, model, and gas left in the tank.
• Right click the project name in the **Solution Explorer** and select **Add > Class**…

• Enter the name of your new file (**GradeBook.vb**).
' Fig. 4.1: GradeBook.vb

Class declaration with one method.

Public Class GradeBook
    ' display a welcome message to the GradeBook user
    Public Sub DisplayMessage()
        Console.WriteLine("Welcome to the Grade Book!")
    End Sub ' DisplayMessage
End Class ' GradeBook

Fig. 4.1 | Class declaration with one method.
4.3 Declaring a Class with a Method and Instantiating an Object of a Class (Cont.)

• A method that begins with keyword `Public` can be called from outside its class.

• **Keyword Sub** indicates that this method will perform a task but will not return any information to its **calling method**.

• By convention, method names begin with an uppercase letter and all subsequent words in the name begin with a capital letter.

• The parentheses after the method name are used to indicate any **parameters**.
• Typically, you cannot call a method that belongs to another class until you create an object of that class.

• Each new class you create becomes a new type in Visual Basic.
  – This is one reason why Visual Basic is known as an extensible language.
• Variable `gradeBook` is initialized with the result of the **object creation expression** `New GradeBook()` (Fig. 4.2).

```vbnet
Module GradeBookTest

' Main begins program execution
Sub Main()
  ' initialize gradeBook to refer to a new GradeBook object
  Dim gradeBook As New GradeBook()

  ' call gradeBook's DisplayMessage method
  gradeBook.DisplayMessage()

End Sub ' Main
End Module ' GradeBookTest

Welcome to the Grade Book!
```

**Fig. 4.2** | Creating an object of class `GradeBook` and calling its `DisplayMessage` method.
4.3 Declaring a Class with a Method and Instantiating an Object of a Class (Cont.)

• Figure 4.3 presents a **UML class diagram** for class *GradeBook*.
  - The top compartment contains the name of the class.
  - The middle compartment contains the class’s **attributes**.
  - The bottom compartment contains the class’s **operations**.

• The plus sign (+) in front of the operation name indicates a public operation.

![UML class diagram](image)

**Fig. 4.3** | UML class diagram indicating that class *GradeBook* has a public `DisplayMessage()` operation.
• A method can require parameters to perform its task.
• A method call supplies values—called arguments—for each parameter.

Fig. 4.4 | Class declaration with one method that has a parameter.
Module GradeBookTest

Sub Main()
    ' initialize gradeBook to reference a new gradeBook object
    Dim gradeBook As New GradeBook()

    ' prompt for the course name
    Console.WriteLine("Please enter the course name:")

    ' read the course name
    Dim nameOfCourse As String = Console.ReadLine()

    Console.WriteLine() ' output a blank line

Fig. 4.5 | Creating a GradeBook object and passing a String to its DisplayMessage method. (Part 1 of 2.)
Please enter the course name:
CS101 Introduction to Visual Basic Programming

Welcome to the grade book for
CS101 Introduction to Visual Basic Programming!

**Fig. 4.5** Creating a *GradeBook* object and passing a *String* to its *DisplayMessage* method. (Part 2 of 2.)
4.4 Declaring a Method with a Parameter (Cont.)

• Each parameter must have a type and an identifier.
• Multiple parameters can be listed, separated by commas.
• The number of arguments in a method call must match the number of parameters.
4.4 Declaring a Method with a Parameter (Cont.)

• The UML class diagram in Fig. 4.6 models class GradeBook.

![UML class diagram](image)

*Fig. 4.6* | UML class diagram indicating that class GradeBook has a DisplayMessage operation with a courseName parameter of type String.
4.5 Instance Variables and Properties

• Variables declared in the body of a method are known as **local variables** and can be used only inside that method.

• Attributes are represented as **instance variables** and are declared inside a class declaration (but outside of other parts of the class).

• Each object of a class maintains its own copy of an instance variable.
• Class **GradeBook** (Fig 4.7) now maintains the course name as an instance variable so that it can be stored and modified.

```vbnet
Public Class GradeBook
    Private courseNameValue As String ' course name for this GradeBook

    ' property CourseName
    Public Property CourseName() As String
        Get ' retrieve courseNameValue
            Return courseNameValue
        End Get

        Set(ByVal value As String) ' set courseNameValue
            courseNameValue = value ' store the course name in the object
        End Set
    End Property ' CourseName

    ' Fig. 4.7: GradeBook.vb
    ' GradeBook class that contains instance variable courseNameValue
    ' and a property to get and set its value.
```

**Outline**

- **GradeBook.vb**
  - (1 of 2)

  **Instance variable courseNameValue is declared as a String.**

  **Property’s Get accessor**

  **Property’s Set accessor**

**Fig. 4.7** | GradeBook class that contains a courseNameValue instance variable and a CourseName property. (Part 1 of 2.)
`GradeBook.vb`

(2 of 2)

Fig. 4.7 | GradeBook class that contains a courseNameValue instance variable and a CourseName property. (Part 2 of 2.)
4.5 Instance Variables and Properties (Cont.)

- **Private** variables, methods and properties are accessible only in the class in which they are declared.
- Declaring **private** instance variables is known as **information hiding**.
  - Variable `courseNameValue` is hidden in the object.
4.5 Instance Variables and Properties (Cont.)

Software Engineering Observation 4.1

Precede every instance variable declaration, method declaration and property declaration with an access modifier. In most cases, instance variables should be declared Private, and methods and properties should be declared Public. Instance variables are Private by default, and methods and properties are Public by default.

Software Engineering Observation 4.2

Declaring the instance variables of a class as Private and the methods of the class as Public facilitates debugging, because problems with data manipulations are localized to the class’s methods and properties.
• The `Get` accessor begins with the keyword `Get` and ends with the keywords `End Get`.
  – The accessor’s body contains a `Return` statement, which consists of the keyword `Return` followed by an expression.

• Using the value of `gradeBook.CourseName` implicitly executes `CourseName`’s `Get` accessor:

```csharp
Dim theCourseName As String = gradeBook.CourseName
```
• The **Set** accessor begins with the keyword **Set** and ends with the keywords **End Set**.

• Assigning a value to `gradeBook.CourseName` implicitly executes `CourseName`'s **Set** accessor:
  
  ```
  gradeBook.CourseName = "CS100 Introduction to Computers"
  ```

• The order in which methods and properties are declared in a class does not determine when they are called at execution time.

• Either property accessor can be omitted (making a property read-only or write-only).
• Module GradeBookTest (Fig. 4.8) creates a GradeBook object and demonstrates property CourseName.

• Variables have a default initial value.
  – String is a reference type; the default value for reference types is Nothing.

Fig. 4.8 | Creating and manipulating a GradeBook object (invoking properties). (Part 1 of 2.)
' read course name
Dim theName As String = Console.ReadLine()

gradeBook.CourseName = theName ' set the CourseName (invokes Set)
Console.WriteLine() ' output a blank line

' display welcome message including the course name (invokes Get)
gradeBook.DisplayMessage()

End Sub ' Main
End Module ' GradeBookTest

Initial course name is:

Please enter the course name:
CS101 Introduction to Visual Basic Programming

Welcome to the grade book for
CS101 Introduction to Visual Basic Programming!

Fig. 4.8 | Creating and manipulating a GradeBook object (invoking properties). (Part 2 of 2.)
4.5 Instance Variables and Properties (Cont.)

*GradeBook*’s *UML Class Diagram with a Property*

- The property is listed as a public attribute (Fig. 4.9), as indicated by the plus (+) sign.
- Using the word “Property” in guillemets (« and ») helps distinguish it from other attributes.
- To indicate that an attribute or operation is private, use a minus sign (–).

*Fig. 4.9* | UML class diagram indicating that class *GradeBook* has a *courseName* attribute of type *String*, one property and one method.
Software Engineering Observation 4.3

Accessing *Private* data through *Set* and *Get* accessors not only protects the instance variables from receiving invalid values, but also hides the internal representation of the instance variables from that class’s clients. Thus, if representation of the data changes (often, to reduce the amount of required storage or to improve performance), only the properties’ implementations need to change—the clients’ implementations need not change as long as the services provided by the properties are preserved.
4.6 Value Types and Reference Types

- Data types in Visual Basic are divided into **value types** and **reference types**.
  - A variable of a value type simply contains a value of that type (Fig. 4.10).

```
Dim count As Integer = 7

count
```

A variable (`count`) of a value type (`Integer`) contains a value (7) of that type.

**Fig. 4.10** | Value type variable.
4.6 Value Types and Reference Types (Cont.)

• A variable of a reference type contains the memory address where the data referred to by that variable is stored.

• Reference type instance variables are initialized by default to the value Nothing.

• A reference to an object is used to invoke the object’s methods and access the object’s properties.

gradeBook.CourseName = theName  
' set the CourseName (invokes Set)
4.7 Initializing Objects with Constructors

• A constructor initializes an object of the class.
• The `New` key word calls the class’s constructor to perform the initialization (Fig. 4.11).

```vbnet
Dim gradeBook As New GradeBook("
"CS101 Introduction to Visual Basic Programming")
```

![Diagram](image)

**Fig. 4.11** | Reference type variable.
Fig. 4.12 | GradeBook class with a constructor that receives a course name. (Part 1 of 2.)
Set(ByVal value As String) ' set courseNameValue
  courseNameValue = value ' store the course name in the object
End Set
End Property ' CourseName

' display a welcome message to the GradeBook user
Public Sub DisplayMessage()
  ' use property CourseName to display the
  ' name of the course this GradeBook represents
  Console.WriteLine("Welcome to the grade book for " _
    & vbCrLf & CourseName & "!")
End Sub ' DisplayMessage
End Class ' GradeBook

Fig. 4.12 | GradeBook class with a constructor that receives a course name. (Part 2 of 2.)
Module GradeBookTest

' Main begins program execution
Sub Main()
    ' create GradeBook object
    Dim gradeBook1 As New GradeBook("CS101 Introduction to Visual Basic Programming")
    Dim gradeBook2 As New GradeBook("CS102 Data Structures in Visual Basic")

Fig. 4.13 | Constructor used to initialize GradeBook objects. (Part 1 of 2.)
```vbnet
' display initial value of CourseName for each GradeBook
Console.WriteLine( "gradeBook1 course name is: " & gradeBook1.CourseName)
Console.WriteLine( "gradeBook2 course name is: " & gradeBook2.CourseName)
End Sub ' Main
End Module ' GradeBookTest
```

```
gradeBook1 course name is: CS101 Introduction to Visual Basic Programming
gradeBook2 course name is: CS102 Data Structures in Visual Basic
```

**Fig. 4.13 |** Constructor used to initialize GradeBook objects. (Part 2 of 2.)

**Error-Prevention Tip 4.1**

Unless default initialization of your class’s instance variables is acceptable, provide a constructor to ensure that these variables are properly initialized with meaningful values when each new object of your class is created.
4.7 Initializing Objects with Constructors (Cont.)

Adding the Constructor to Class GradeBook’s UML Class Diagram

• The UML class diagram of Fig. 4.14 models the class GradeBook.

• The word “constructor” is between guillemets (« and »).

Fig. 4.14 | UML class diagram indicating that class GradeBook has a constructor that has a name parameter of type String.
• Suppose the system only allows course names of 25 characters or less.

• We now enhance class `GradeBook`’s property `CourseName` to perform this **data validation** (Fig. 4.15)

```vbnet
' GradeBook class with a property that performs validation.
Public Class GradeBook

    Private courseNameValue As String ' course name for this GradeBook

    ' constructor initializes CourseName with String supplied as argument
    Public Sub New(ByVal name As String)
        CourseName = name ' validate and store course name
    End Sub ' New

    ' property that gets and sets the course name; the Set accessor
    ' ensures that the course name has at most 25 characters
    Public Property CourseName() As String
        Get ' retrieve courseNameValue
            Return courseNameValue
        End Get

Fig. 4.15 | Method declarations for class GradeBook with a CourseName property that validates the length of instance variable courseNameValue. (Part 1 of 3.)
Fig. 4.15 | Method declarations for class `GradeBook` with a `CourseName` property that validates the length of instance variable `courseNameValue`. (Part 2 of 3.)
35 ' display a welcome message to the GradeBook user
36 Public Sub DisplayMessage()
37 ' this statement uses property CourseName to get the
38 ' name of the course this GradeBook represents
39 Console.WriteLine("Welcome to the grade book for " _
    & vbCrLf & CourseName & "!")
40 End Sub ' DisplayMessage
41 End Class ' GradeBook

Fig. 4.15 | Method declarations for class GradeBook with a CourseName property that validates the length of instance variable courseNameValue. (Part 3 of 3.)
4.8 Validating Data with Set Accessors in Properties (Cont.)

- **Length** returns the number of characters in the **String**.
- **Substring** returns part of an existing **String**.
- To display double quotes in a string, you use two double quotes in a row.
- The first 25 characters are assigned to **courseNameValue** to maintain a **consistent state**.
• Testing the GradeBook object (Fig. 4.16).

```vbnet
Module GradeBookTest
    ' Main begins program execution
    Sub Main()
        ' create two GradeBook objects;
        ' initial course name of gradeBook1 is too long
        Dim gradeBook1 As New GradeBook("CS101 Introduction to Visual Basic Programming")
        Dim gradeBook2 As New GradeBook("CS102 VB Data Structures")

        ' display each GradeBook's course name (by invoking Get)
        Console.WriteLine("gradeBook1's initial course name is: ", gradeBook1.CourseName)
        Console.WriteLine("gradeBook2's initial course name is: ", gradeBook2.CourseName)

        Console.WriteLine()  ' display blank line

        ' place in gradeBook1's course name a valid-length String
        gradeBook1.CourseName = "CS101 VB Programming"
    End Sub
End Module
```

Fig. 4.16 | Creating and manipulating a GradeBook object in which the course name is limited to 25 characters in length. (Part 1 of 2.)
' display each GradeBook's course name (by invoking Get)
Console.WriteLine( _
  "gradeBook1's course name is: " & gradeBook1.CourseName)
Console.WriteLine( _
  "gradeBook2's course name is: " & gradeBook2.CourseName)
End Sub ' Main
End Module ' GradeBookTest

Name "CS101 Introduction to Visual Basic Programming" exceeds maximum length (25). Limiting course name to first 25 characters.

gradeBook1's initial course name is: CS101 Introduction to Vis
gradeBook2's initial course name is: CS102 VB Data Structures

gradeBook1's course name is: CS101 VB Programming
gradeBook2's course name is: CS102 VB Data Structures

Fig. 4.16 | Creating and manipulating a GradeBook object in which the course name is limited to 25 characters in length. (Part 2 of 2.)
Error-Prevention Tip 4.2

The benefits of data integrity are not automatic simply because instance variables are made Private—you must provide appropriate validity checking and report the errors.

Error-Prevention Tip 4.3

Set accessor that set the values of Private data should verify that the intended new values are proper; if they are not, the Set accessor should place the Private instance variables into an appropriately consistent state.
4.9 Software Engineering Case Study: Identifying the Classes in the ATM Requirements Document

*Identifying the Classes in a System*

- Figure 4.17 lists the nouns in the requirements document.

<table>
<thead>
<tr>
<th>Nouns and noun phrases in the requirements document</th>
</tr>
</thead>
<tbody>
<tr>
<td>bank</td>
</tr>
<tr>
<td>ATM</td>
</tr>
<tr>
<td>user</td>
</tr>
<tr>
<td>customer</td>
</tr>
<tr>
<td>transaction</td>
</tr>
<tr>
<td>account</td>
</tr>
<tr>
<td>balance</td>
</tr>
</tbody>
</table>

*Fig. 4.17* | Nouns and noun phrases in the requirements document.
4.9 Software Engineering Case Study: Identifying the Classes in the ATM Requirements Document (Cont.)

• We do not need to model “bank,” “user” or “customer” as a class, because they are not parts of the ATM system itself.

• We do not model “$20 bill” or “deposit envelope” as classes. These are physical objects in the real world, but they are not part of what is being automated.

• Monetary values are best represented as attributes. The account number and PIN represent attributes of a bank account.
4.9 Software Engineering Case Study: Identifying the Classes in the ATM Requirements Document (Cont.)

• Each class is modeled as a rectangle with three compartments:
  – Top compartment: name of the class.
  – Middle compartment: class’s attributes.
  – Bottom: class’s operations.

Fig. 4.18 | Representing a class in the UML using a class diagram.
4.9 Software Engineering Case Study: Identifying the Classes in the ATM Requirements Document (Cont.)

• Class diagrams also show the relationships between the classes of the system (Fig. 4.19).
  – Hide class attributes and operations to create readable diagrams.

• currentTransaction is a role name for the Withdrawal object.

• An association is a relationship between classes.

• Multiplicity values indicate how many objects of each class participate in the association.

Fig. 4.19 | Class diagram showing an association among classes.
4.9 Software Engineering Case Study: Identifying the Classes in the ATM Requirements Document (Cont.)

- The UML can model many types of multiplicity (Fig. 4.20).

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>None</td>
</tr>
<tr>
<td>1</td>
<td>One</td>
</tr>
<tr>
<td>$m$</td>
<td>An integer value</td>
</tr>
<tr>
<td>0..1</td>
<td>Zero or one</td>
</tr>
<tr>
<td>$m, n$</td>
<td>$m$ or $n$</td>
</tr>
<tr>
<td>$m..n$</td>
<td>At least $m$, but not more than $n$</td>
</tr>
<tr>
<td>*</td>
<td>Any nonnegative integer (zero or more)</td>
</tr>
<tr>
<td>0..*</td>
<td>Zero or more (identical to *)</td>
</tr>
<tr>
<td>1..*</td>
<td>One or more</td>
</tr>
</tbody>
</table>

Fig. 4.20 | Multiplicity types.
In Fig. 4.21, the **solid diamonds** indicate that class **ATM** has a **composition** relationship.

The class that has the composition symbol on its end of the association line is the whole, and the classes on the other end are the parts.

**Fig. 4.21** | Class diagram showing composition relationships.
4.9 Software Engineering Case Study: Identifying the Classes in the ATM Requirements Document (Cont.)

• In the “has-a” relationship:
  – Only one class can represent the whole.
  – A part may belong to only one whole at a time, although the part may be removed and attached to another whole.

• Hollow diamonds are used to indicate aggregation.
  – A personal computer “has a” monitor, but the two parts can exist independently, and the monitor can be attached to multiple computers.
• Figure 4.22 shows a class diagram.
  – ATM has a one-to-one relationship with class BankDatabase.
  – BankDatabase has a one-to-many relationship with class Account.

• Only the BankDatabase can access and manipulate an account directly.
4.9 Software Engineering Case Study: Identifying the Classes in the ATM Requirements Document (Cont.)

**Fig. 4.22** | Class diagram for the ATM system model.