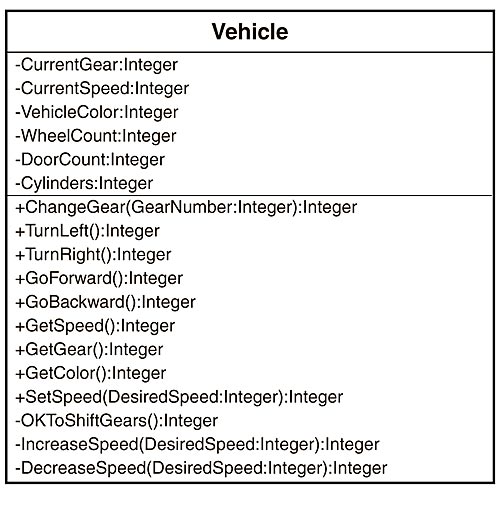
**UML Class Diagrams** (Visual Basic .NET Primer Plus By Jack Purdum Published Apr 11, 2003 by Sams.)

The standard UML diagram, or notation, for a class is a rectangle that's divided into three compartments. Starting at the top of the class diagram, the first compartment contains name of the class. The second compartment contains attributes of the class. (You'll also hear the attributes of a class referred to as the *properties* or *variables* of the class.) The third compartment contains class methods. The methods of the class tell us what the class can do.

[Figure 3.2](javascript:popUp('/content/images/chap3_0672324857/elementLinks/03fig02.jpg')) shows a class diagram with the class name Vehiclein the first compartment. The standard naming convention for class names is to begin the class name with an uppercase letter. If the class name contains more than one word, each word in the class name is in uppercase. For example, class names that follow this naming convention would include Vehicle, PassengerCar, and IncomeStatement. There are no spaces between the words in a class name.

[**Figure 3.2**](javascript:popUp('/content/images/chap3_0672324857/elementLinks/03fig02.jpg')) **A sample UML class diagram for a class named Vehicle.**

**Class Properties (Attributes)**

In our example, compartment two has six Integer attributes. In a more complete example, there could be dozens of attributes for a class. Visual Basic .NET tends to refer to these attributes as the *properties* of the class. Each property can assume different values. Collectively, the current values of the properties describe the state of a class object. For example, if the CurrentGear value is 4 and CurrentSpeed is 55, it seems reasonable to assume that the Vehicle object is going forward at 55 miles per hour. Therefore, the state of the object is such that it is moving forward at 55 mph. If we can draw an English analogy, the properties of an object are like the nouns of a sentence.

**Plus Signs and Minus Signs**

You'll notice that each entry in the second compartment has a minus sign in front of it. In the third compartment, some entries have plus signs, whereas others have minus signs. The plus sign (+) signifies that the item on that line is available outside the class. Stated in a different way, the plus signs means that we can use these items to affect the state of the class object. The plus signs, therefore, denote Public elements of the class.

If an entry is prefixed with a minus sign (-), it means that the item is available for use only within the class itself. The item is not visible or accessible outside of the class. The minus signs, therefore, mark the Private elements of a class.

As a general rule, making an item Private is a good thing and is consistent with our goal of hiding data whenever possible. This is an integral part of the idea of encapsulation that we discussed in Chapter 2. By encapsulating the data, we minimize the chance of inadvertently changing the data in some part of the program outside of the class. This makes finding program errors much easier.

The plus and minus signs, therefore, denote the access specifiers for each element in a class. (There is a third access specifier named Protected that's denoted by the sharp symbol, #. We won't discuss this access specifier until Chapter 16, "Class Properties.") You can think of the plus signs as defining the way you interact with class objects, whereas the minus signs tells you what's available only to the class itself. (We'll delve into these program elements in Chapter 15, "Encapsulation.")

**Class Methods (Operations)**

Compartment three lists the operations that are available in the class. Although UML notation calls the items in compartment three *operations*, when using Visual Basic .NET, the tendency is to refer to these operations as *methods*. Methods tell the programmer how they must interact with the class. If you look at the names of the items in compartment three, each seems to imply an action of some sort. If properties are the nouns of a sentence, methods are the verbs.

You'll often hear other programmers refer to the methods as the *procedures* of a class. Even though there's nothing seriously wrong with this term, you should remember that procedures can exist outside of a class. However, methods are almost always tied to a particular class object.

**Class Methods with Arguments**

Sometimes a method needs outside information to perform its task. For example, in Figure 3.1, we see the following line:

+SetSpeed(DesiredSpeed:Integer):Integer

The plus sign says that SetSpeed() is a Public method, so it can be used in conjunction with a class object. Between the parentheses, we see that a data value is passed to the SetSpeed() method of the Vehicle class. This data value is given the name DesiredSpeed and it is an Integer data type. (You'll learn more about data types in the next chapter. For now, just think of DesiredSpeed as a number.)

After the closing parentheses, we see a colon (:) followed by the word Integer. This means that SetSpeed() will return an integer value to the part of the program that wanted to use the SetSpeed() method. Although we can't be sure at this point, the return value is probably used to indicate whether or not we were able to set the speed. After all, things can go wrong. For example, SetSpeed() probably doesn't work too well if the Vehicle object isn't running. Likewise, setting the speed to 150 is probably not wise if the current gear is Reverse.

Now look at the two lines:

-IncreaseSpeed(DesiredSpeed:Integer):Integer

-DecreaseSpeed(DesiredSpeed:Integer):Integer

These look very similar in terms of their purpose and use. In fact, such similarities should tip you off that it might be possible to simplify things. Could we replace these two methods with the following method

-ChangeSpeed(DesiredSpeed:Integer):Integer

with the understanding that if DesiredSpeed is positive, it represents the amount we want to raise the speed whereas a negative number is the amount we want to decrease the speed? For example, consider what the following code might do:

Dim MyVehicle as New Vehicle

Dim ObjectSpeed as integer

' Some code that does something...

ObjectSpeed = MyVehicle.GetSpeed()

ObjectSpeed = MyVehicle.ChangeSpeed(-ObjectSpeed)

First we create a Vehicle object named MyVehicle. Then we call the GetSpeed() method for the MyVehicle object. Notice how a dot (called the dot operator) separates the object name from the method name. All Visual Basic .NET objects use this syntax format.

Let's assume that the vehicle is presently traveling at 55 mph. The value of ObjectSpeed is assigned the value of *55*. If we then pass the negative value of the current speed (that is, -55) to the ChangeSpeed() method, we can cause the car to stop.

I realize that there are a lot of syntax details that you don't understand about this example right now. Not to worry. We'll cover all of those details in subsequent chapters. For now, however, I just want you to get an idea of what a UML class diagram is and the type of information it conveys.

Why are some of the methods marked with minus signs? Obviously this means they are Private methods and not available outside of the class. Such methods are helper methods that are used internally by the class itself to accomplish its tasks. For example, the method OkToShiftGears() might check the CurrentGear and CurrentSpeed values to see if it's safe to shift gears. If the CurrentSpeed is 55, it might not be a good idea to shift into Reverse right now. Therefore, the ChangeGear()method might call OkToShiftGears() to help it decide whether it's safe to change gears.

The idea behind a UML class diagram is that is gives you a quick, concise overview of what the class does and how you as a programmer are expected to interact with it. If you think of a class as a black box, the minus signs indicate things inside the black box you shouldn't mess around with as a user of the class. The plus signs indicate the means you *must* use to interact with the properties and methods of the class. In other words, the Public items of a class define the interface for the class object and how you must interact with it.

**Programmer's Tip**

Always keep in mind that it's the Public methods that dictate how programmers must use your class. As long as you don't change how these methods are used, your interface with those programmers doesn't change. This lends consistency to your class.

On the other hand, you can change the Private methods to your heart's content and never need to worry about getting the programmers ticked off at you because you've changed the way they work. After all, they never see or use the Private methods anyway. This usually means keeping the Public methods as simple as possible and using the Private methods to manage the details of the task at hand.

We'll revisit UML class diagrams again in later chapters. For now, just review the material in this section to become comfortable with the nature of information UML class diagrams provide.