AN INTRODUCTION TO PROGRAMMING AND VISUAL BASIC

Starting Point

Go to www.wiley.com/college/petroutsos to assess your knowledge of the basics of programming and Visual Basic.
Determine where you need to concentrate your effort.

What You’ll Learn in This Chapter

▲ Operations all computers perform
▲ Basic control structures
▲ Programming languages and the need for just-in-time compilation
▲ Programming in Windows
▲ Object-oriented programming
▲ The programming process
▲ The purpose and key features of the Visual Basic programming environment
▲ The relationship between a solution and a project
▲ How to create Windows application projects
▲ Forms and form properties
▲ Controls
▲ The role of events in Windows programming and how to build a simple program that responds to events
▲ How to test code

After Studying This Chapter, You’ll Be Able To

▲ Identify the types of operations a computer carries out in order to prepare to write a program
▲ Identify the steps required to prepare to write a program
▲ Install and get help in Visual Studio
▲ Write pseudocode as preparation for creating a program
▲ Create an IPO table as preparation for creating a program
▲ Create a Windows application project
▲ Use the Solution Explorer window to manage the files in a project
▲ Customize the Visual Studio environment to make it easier to work in
▲ Add a form to a project
▲ Set form properties by using the Properties window
▲ Add TextBox, Label, and Button controls to a form
▲ Set control properties by using the Properties window
▲ Add code to a form event
▲ Add code to a control event
▲ Test code by debugging
INTRODUCTION

This chapter introduces the fundamentals of programming using Microsoft Visual Studio 2005 and Microsoft Visual Basic 2005. The first section introduces you to some basic programming concepts and the role that Visual Studio and Visual Basic play in program development. In the second section, you will learn about solutions and projects and become familiar with the Visual Studio development environment. In the third section, you will learn about Windows Forms. A Windows form allows you to create the user interface for a Windows application. In the fourth section, you will look at the basic steps for adding user interface elements, known as controls, to a Windows form. In the final section, you will learn about event-driven programming and write your first code.

1.1 Getting Familiar with Programming

No doubt, you have used many applications. But have you thought about what goes on behind the scenes or the tools used to create that application? In this section, you’ll discover the difference between source code and a compiled program. Next, you’ll learn some basics about the development environment you will be using: Visual Studio. After that, you’ll learn the fundamental steps for preparing to install and installing Visual Studio. Finally, you’ll learn about the documentation and help features available for learning about Visual Studio and Visual Basic.

1.1.1 An Introduction to Programming

Before you begin to think about creating a computer program, it’s useful to understand the six operations that all computers can carry out. Understanding these operations will help you when you start writing programs. These are the six operations a computer can perform:

1. Input data
2. Store data in internal memory
3. Perform arithmetic on data
4. Compare two values and select one of two alternative actions
5. Repeat a group of actions any number of times
6. Output the results of processing

These six operations are depicted in Figure 1-1, where each operation is numbered. Let’s discuss each of these operations in a little more detail:

1. **Input data**: For a computer to be able to process data, it must first be able to accept input of data. Data is typically input from a keyboard or
mouse, but it can also come from other sources, such as a bar-code reader like those used at checkout terminals. Input can also come from some type of sensor or from a data file on a computer disk. For example, with a word processor, the letters of the alphabet, numbers, and punctuation symbols form the data that is processed by the computer. New documents are created by entering data from the keyboard, and existing documents are loaded from a hard drive or flash memory.

2. **Store data in memory**: After data has been input, it is stored in internal memory. Each memory location that holds data is assigned a name, which the instructions use to perform the processing. Because the values in a memory location can change as the process occurs, the memory locations are called **variables**. For example, the current balance in your checking account would typically be stored in a single memory location and be identified by a variable name. The instructions for processing this data are also stored in memory.

3. **Perform arithmetic on data**: Once data and instructions have been input and stored, arithmetic operations can be performed on the variables that represent the data. This includes addition, subtraction, multiplication, division, and raising to a power. The processing chip of the computer carries out these operations by retrieving the data from memory and then performing the processing based on instructions from the programmer. How does a word processor or computer game work if all the computer can do is perform arithmetic? The answer is that everything in a computer—numbers, letters, graphics, and so on—is represented by numbers, and all processing is handled through some type of arithmetic operation.

4. **Compare two values and select one of two alternative actions**: To do anything other than the simplest processing, a computer must be able to choose between two sets of instructions to execute. It does this by
comparing the contents of two memory locations and, based on the result of that comparison, executing one of two groups of instructions. For example, when you carry out the spell-checking operation, the computer checks each word to determine whether it matches a word in the computer's dictionary. Based on the result of this comparison, the word is accepted or flagged for you to consider changing.

5. Repeat a group of actions any number of times: Although you could carry out all the preceding operations with a typewriter or handheld calculator, repeating actions is something the computer does better than any person or any other type of machine. Because a computer never tires or becomes bored, it can be instructed to repeat some action as many times as needed, without fear of an error occurring due to the constant repetition. The capability of a computer to repeat an operation is what most clearly sets it apart from all other machines. The spell-checking operation mentioned earlier is an example of a repeated action: The program repeatedly checks words until it comes to the end of the document.

6. Output the results of processing: After the processing has been completed and the required information generated, to be of any use, the information must be output. Output of processed information can take many forms: It can be displayed on a monitor, printed on paper, stored as files on a disk, and so on. Output is accomplished by retrieving information from a memory location and sending it to the output device. For example, when you complete your work with a word processor, the resulting information is displayed on your monitor and saved to a file, and you may also print it for distribution to others.

1.1.2 Programs and Programming

To carry out any of the six operations just discussed, you must be able to provide instructions to the computer, in the form of a program. The most important thing about programming is that it is a form of problem solving, and the objective is to develop the step-by-step process—the logic—that will solve the problem. Step-by-step logic of this type is referred to as an algorithm. You have worked with algorithms before; a set of directions to a party is an algorithm, as is a recipe to make spaghetti sauce or to bake a cake. For a computer program, you must develop a set of instructions for solving a problem, using only the six operations of a computer. This is the most difficult part of programming.

Many times, a program fails to work because the programmer attempts to write the program before developing the correct algorithm for solving the problem. Only after you have developed the logic of the solution can you consider actually writing the instructions for the computer.
Control Structures
All computer programs are created using just three types of logic, or, as they are known in programming, control structures: the sequence, decision, and repetition control structures.

The sequence control structure includes the input, storage, arithmetic, and output computer operations discussed earlier. It is so called because all four of these operations can be performed without any need to make a decision or repeat an operation. At its simplest, sequence means one program instruction follows another, in order. The programmer must determine the proper sequence for the instructions.

The decision control structure is the same as the decision-making computer operation discussed earlier. It enables the programmer to control the flow of operations by having the user or data determine which operation is to be performed next.

Finally, the repetition control structure is used to repeat one or more operations. The number of repetitions depends on the user or the data, but the programmer must include a way to terminate the repetition process.

All algorithms are created by performing the six operations of a computer, using combinations of these three control structures. When you learn how to create the logic for these three control structures, you will find that writing meaningful and useful programs is a matter of combining the structures to create more complex logic.

Programming Languages
When you have developed the logic for solving a problem, you can think about writing the actual instructions that the computer will use in implementing the logic. Computer programs must be written in one of various programming languages, such as Visual Basic. Each of these languages uses a restricted vocabulary and a very structured syntax (similar to grammar) that the computer can understand. Although a great deal of research is ongoing to create computers that can accept instructions using conversational English, currently no computers can do so.

Within a computer, the data and instructions are represented in the binary number system as a series of 0s and 1s. This form of representation is used because the computer has only two electrical states, on and off, corresponding to 1 and 0. Using a string of transistors that act as switches, the computer can represent a number, a character, or an instruction as a series of on/off states. All processing is carried out in the binary number system. For example, a computer carries out all arithmetic in binary instead of in the decimal number system that humans use. The binary form of the instructions is called machine language because this is the language that computers use to carry out their operations. An example of the machine language statements necessary to sum the digits 1 to 100 for a computer using an Intel CPU chip is shown in Figure 1-2.
Programming the very first computers, which had to be done in binary, was very difficult and time-consuming. Now, we have English-like programming languages, such as Visual Basic, that are referred to as **high-level languages** because they are closer to the level of the human programmer than to the level of the machine. Before the statements in a high-level program can be used to direct the actions of a computer, they must be translated into machine language. Files on a Windows-based computer with an .exe file extension are machine language programs that have been translated from some high-level language. They can be executed with no translation because they are already in a binary form. Until recently, this was a direct translation from high-level language to machine language by a software program known as a **compiler** or an interpreter, depending on whether the code was translated as a unit or line by line; Figure 1-3 shows how the direct translation process works.

The problem with the direct translation approach is that different types of computers have different machine languages, so a program would have to be translated differently for an Apple computer than for a Windows computer. To make it possible for the same program to run on all types of computers, the concept of

---

**Figure 1-2**

**Machine Language Command**

<table>
<thead>
<tr>
<th>Command</th>
<th>Translation</th>
</tr>
</thead>
<tbody>
<tr>
<td>10111000 00000000 00000000</td>
<td>Set Total Value to 0</td>
</tr>
<tr>
<td>10111001 00000000 01100100</td>
<td>Set Current Value to 100</td>
</tr>
<tr>
<td>00000001 11001000</td>
<td>Add Current Value to Total Value</td>
</tr>
<tr>
<td>01001001</td>
<td>Subtract 1 from Current Value</td>
</tr>
<tr>
<td>01110101 11111011</td>
<td>If Current Value is not 0, repeat</td>
</tr>
</tbody>
</table>

A machine language program.

**Figure 1-3**

The direct translation process.
the just-in-time (JIT) compiler was developed. With this approach, a high-level program is translated or compiled into an intermediate form that is machine independent. The two approaches to this use of a JIT compiler are Java from Sun Microsystems and the .NET Framework from Microsoft, of which Visual Basic 2005 is a part. In the case of Java, the intermediate form is called bytecode, and for the .NET Framework, it is called Microsoft Intermediate Language (MSIL). Once converted, a Visual Basic 2005 program is compiled into MSIL; the JIT compiler on any computer can convert it into machine language for that particular machine. This process for MSIL is shown in Figure 1-4.

Whereas the Java approach works only for programs written in Java, the .NET Framework approach works for all languages that have been revised to work under that framework. At this time, these include Visual Basic, C# (pronounced “c-sharp”), and C++ (pronounced “c plus plus”) .NET. This means that if you are using one of these languages, it can be compiled in MSIL and combined with other programs in MSIL and then sent to the JIT compiler, which for the .NET Framework is called the Common Language Runtime (CLR).

1.1.3 Programming in Windows

As you are probably aware, most personal computers today run some form of the Microsoft Windows operating system, such as Windows 2000, Windows XP, or Windows Server 2003. With Windows being the primary operating system for personal computers, learning to program in the Windows environment has become a critical skill for anybody who is interested in working in information systems. To program in Windows, you first need to understand a little about how Windows works.
To understand the workings of Windows, you need to understand three key concepts: windows, events, and messages. A window is any rectangular region on the screen that has its own boundaries. All visible components run in their own windows. For example, when you use a word processor, a document window displays the text you are entering and editing. When you retrieve a file, you do this from a dialog box that is a window that displays on top of other windows. Similarly, when an error message is displayed, it is done in a window. Figure 1-5 shows a Windows XP screen with several types of windows displayed.

As a part of its operations, the Windows operating system constantly monitors all the windows on the screen for signs of activity, termed events. An event can be a mouse click or a double-click, a keypress, or a change in a window caused by an entry of text in it.

When an event occurs, the corresponding window sends a message to the operating system, which processes the message and then broadcasts it to other windows. When they receive a message, the other windows take actions based on their own sets of instructions. Programming in Windows requires that you learn how to work with windows, events, and messages. For this reason, programming in Windows is usually termed event-driven programming because all actions are driven by events. Although this may sound complicated, languages such as Visual Basic make it easy to create Windows-based applications by providing you with the necessary tools.
Event-driven programming is quite different from traditional approaches to programming, where the program itself controls the actions that take place and the order in which those actions occur. With traditional programs, execution of the program starts with the first instruction and continues through the remaining instructions, making decisions as to which instructions will be executed, depending on the data that are input. The main program may use smaller subprograms to handle parts of the processing. This type of programming is referred to as **procedural programming**, and it works very well for such activities as processing a large number of grades at the end of a term or printing payroll checks at the end of a pay period. However, with the move toward widespread use of graphical user interfaces (GUIs), the trend is toward using event-driven programming.

Visual Basic is an event-driven language that does not follow a predefined sequence of instructions; it responds to events to execute different sets of instructions, depending on which event occurs. The order in which events—such as mouse clicks, keystrokes, or even other sets of instructions—occur controls the order of events in Visual Basic and other event-driven languages. For that reason, an event-driven program can execute differently each time it is run, depending on what events occur.

### 1.1.4 Introduction to Object-Oriented Programming

In addition to being event driven, Visual Basic is an **object-oriented (OO) language**. An OO language is one that encapsulates functionality into classes. The beauty of Visual Basic is that, unlike with many other OO languages, you do not have to know how to create objects to use them. Visual Basic automatically creates for you, the programmer, new instances of many objects from a wide variety of built-in templates.

What distinguishes OO programming from earlier languages is that objects combine programming instructions, or **code**, with data. Previous attempts to structure programs in such a way that large problems could be broken down into smaller problems separated the code from the data. The problem with this approach is that if the data changes, the code might not work with the new data. With OO programming, the combination of code and data avoids this problem. For example, instead of writing code to deal with customers and then using that code with different customer data for each customer, the code and data are combined into an **object** for each customer. The objects for multiple customers are very similar, with the exception of the data component, so you can use them in similar ways.

To understand OO programming, you need to understand a number of concepts and terminology. First, in order to create an object, you must first create a **class**—that is, a template with data and procedures from which objects are created. One way of looking at this is to think of a class as a cookie cutter and
the actual object as the resulting cookie. All the actual work in creating an object is accomplished in creating the class; an object is created by defining it to be an instance of a class.

Objects have two key elements: properties and methods. Some objects also have events. Properties of objects are simply the attributes associated with the object, such as the object’s name, color, and so on. Methods are a set of predefined activities that an object can carry out. For example, consider the customer objects mentioned earlier; they could be instances of a class called DVDCustomer, which has have the properties and methods shown in Figure 1-6. Note that the DVDCustomer class has Name, Address, PhoneNumber, and LateFees properties. The class also has the Add and Delete methods to add and delete customers. Note that there is also an object named DVDCustomer for a DVD rental store, and there is an instance of this object for Ashley Hyatt that contains properties particular to her.

Three key characteristics of OO programming are encapsulation, inheritance, and polymorphism. Encapsulation refers to the requirement that it should never be possible to work with variables in an object directly; the variables must be addressed through the object’s properties and methods. This implies a black-box view of an object, in which the programmer does not need to know what is going on inside the object but only needs to know how to work with the object’s methods and properties. For example, you would not be able to change the values of the DVDCustomer object without going through the properties of the object; you cannot get into the object except through the properties.

Inheritance refers to the capability to create child classes that descend from a parent class. This capability makes it easier to build new child classes by having them inherit properties and methods from a parent class. For example, the class DVDCustomer inherits the properties and methods from a more general
Customer class, which itself inherits properties and methods from an even more general Person class.

Finally, **polymorphism** is related to inheritance in that a child class can inherit all the characteristics and capabilities of the parent class but then add or modify some of them so the child class is different from the parent class. For example, the DVDCustomer class inherits the Name, Address, and PhoneNumber properties from the Customer class and then adds the LateFees property that is particular to DVDCustomer.

As another example of objects, consider a soccer ball. The SoccerBall class inherits properties and methods from the more general Ball class. These properties include diameter, weight, color, and so on. Methods for the soccer ball include rolling and bouncing. If you apply the Kick event to the soccer ball, then, depending on the ball's diameter and weight, it will roll and bounce a certain distance. It is important to note that the instructions for a method are already a part of Visual Basic, but the programmer must write the instructions to tell the object how to respond to an event. You combine objects with properties or methods by using a period, or dot, and you combine objects with events by using an underscore character. Continuing the soccer ball example, you might have a property definition through the following statement to define the color of the ball:

```vbnet
Ball.Color = White
```

Similarly, the Roll method of the soccer ball is referenced by the dot property as shown here:

```vbnet
Ball.Roll
```

Finally, the Kick event is applied to the soccer ball as follows, causing the ball to roll:

```vbnet
Ball.Kick
```

Working with Visual Basic involves combining objects with the instructions on how each object should respond to a given event. For example, you might have a button for which the instructions are to display a message; instructions for another button might be to exit the program. These instructions are referred to as the **code** for the program.

### 1.1.5 The Programming Process

Creating an application using an OO programming language such as Visual Basic is much easier than working with a traditional programming language. Instead of having to develop the logic for the entire program as you would with a procedural language, you can divide the program logic into small, easily handled parts by working with objects and events. For each object, you determine the events that you want the object to respond to and then develop code to have
the object provide the desired response. All the necessary messages between objects in Windows are handled by Visual Basic, thereby significantly reducing the work you must do to create an application.

The manner in which you create a Visual Basic project is also different from traditional programming. Instead of having to create an entire program before testing any part of it, with Visual Basic, you can use interactive development to create an object, write the code for it, and test it before going on to other objects. For example, assume that a store named Vintage DVDs that rents only “old” movies on DVD has asked you to create a Visual Basic project that calculates taxes on a DVD rental and sums the taxes and price to compute the amount due. With Visual Basic, you can create the objects and code to calculate the taxes and amount due and test them to ensure their correctness before going on to the rest of the project.

Although creating an application in Visual Basic is easier than working with a procedural language, you still need to follow a series of steps to ensure correctness and completeness of the finished product:

1. Define the problem.
2. Create an interface.
3. Develop logic for action objects.
4. Write and test code for action objects.
5. Test the overall project.
6. Document the project in writing.

Note that it may be necessary to repeat or iterate through these steps to arrive at an acceptable final solution to the original problem. The following sections discuss each of these steps and apply them to a part of the situation just mentioned—that is, creating an application to calculate the taxes and amount due on a DVD rental.

**Step 1: Define the Problem**

Before you can develop any computer application, it is absolutely necessary to clearly define your objective—that is, the problem to be solved. Only then can you begin to develop the correct logic to solve the problem and incorporate that logic into a computer application. Ensuring that the correct problem is being solved requires careful study of why a problem exists. Maybe an organization is currently handling some repetitive process manually and wants to use a computer to automate it. Or perhaps management has a complicated mathematical or financial problem that cannot be solved by hand. Or maybe a situation has occurred or will occur that cannot be handled by an existing program.

The problem identification step should include identification of the data to be input to the program and the desired results to be output from the program. Often,
these two items are specified by a person or an agency other than the programmer. Much grief can be avoided if these input and output requirements are incorporated into the programmer's thinking at this early stage of program development. Unclear thinking at this stage may cause the programmer to write a program that does not correctly solve the problem at hand, a program that correctly solves the wrong problem, or a combination of both. Therefore, the programmer must spend as much time as necessary to truly identify and understand the problem.

Because Visual Basic is a visual language, a good way to understand what is required to solve the problem is to sketch the interface, showing the various objects that will be part of the project. Not only does this help you understand the problem, it is also a good way for you to communicate your understanding to other people. As a part of this sketch, you should denote the input and output objects and the objects for which code is needed to respond to events, the so-called action objects. A sketch of the proposed solution for the DVD rental problem is shown in Figure 1-7. In looking at the solution, you see one input—the price of the DVD—and two outputs—the taxes and the amount due. There are also two action objects—a calculation button and an exit button. If there are multiple forms, they should all be sketched, with input, output, and action objects.

**Step 2: Create an Interface**

After you have defined the problem and, using a sketch of the interface, decided on the objects that are necessary for your project, you are ready to create the interface. Creating the interface with Visual Basic is quite easy: You select objects from those available and place them on the form. This process should follow the sketch done earlier. Although you have not yet been introduced to the wide variety of objects available for creating Visual Basic projects, you can work on the logic for the Vintage DVDs problem with just four types of objects: the form, buttons for action, text boxes for input and output, and labels for descriptors. The interface is shown in Figure 1-8.
Step 3: Develop Logic for Action Objects

After you have clearly identified the problem and created the interface, the next step is to develop the logic for the action objects in the interface. This is the step in the development process during which you have to think about what each action object must do in response to an event. No matter how good your interface, if you don’t develop the appropriate logic for the action objects, you will have great difficulty creating a project that solves the problem defined earlier.

To help with this logical development for the action objects, there are two useful tools for designing programming applications: IPO tables and pseudocode. IPO (input/processing/output) tables show the inputs to an object, the required outputs for that object, and the processing that is necessary to convert the inputs into the desired outputs. When you have an IPO table for an object, you can write a pseudocode procedure to complete the logic development step.

Writing pseudocode involves writing the code for the object in structured English rather than in a computer language. After you have developed an IPO table and the pseudocode for each object, it is easy to write a procedure in Visual Basic to carry out the necessary processing.

Let’s begin by developing the logic for the Calculate button, using an IPO table. The IPO table for the Calculate button has as input the price of a DVD. The processing involves the calculation necessary to compute the desired
1.1 GETTING FAMILIAR WITH PROGRAMMING

output: the amount of the sale. As mentioned earlier, in many cases, the program designer has no control over the input and output, which are specified by somebody else—either the person for whom the application is being developed or, if you are a member of a team and are working on one part of an overall application, the overall design. After you are given the specified input and output, your job is to determine the processing necessary to convert the inputs into desired outputs. Figure 1-9 shows the IPO table for the Calculate button. IPO tables are needed for all objects that involve input, output, and processing. (We won’t create an IPO table for the Exit button because it simply terminates the project.)

After you have developed the IPO table for each action object, you should then develop a pseudocode procedure for each one. Pseudocode is useful for two reasons. First, you can write the procedure for the object in English, without worrying about the special syntax and grammar of a computer language. Second, pseudocode provides a relatively direct link between the IPO table and the computer code for the object because you use English to write instructions that can then be converted into program instructions. Often, this conversion from pseudocode statement to computer language instruction is virtually line for line.

There are no set rules for writing pseudocode; you can personalize the method you use for going from the IPO table to the computer program. The pseudocode should be a set of clearly defined steps that enables a reader to see the next step to be taken under any possible circumstance. Also, the language and syntax should be consistent so that the programmer will be able to understand his or her own pseudocode at a later time.

Let’s write a pseudocode procedure for the Vintage DVDs Calculate object. Note that this pseudocode program follows the information in the IPO table shown in Figure 1-9:

```
Begin procedure
  Input DVD Price
  Taxes = 0.07 × DVD Price
  Amount due = DVD Price + Taxes
  Output Taxes and Amount due
End procedure
```
This small example has only one object for which an IPO table and pseudocode are needed. However, in most situations, there are numerous objects for which you need to develop the logic using these tools.

**Step 4: Write and Test Code for Action Objects**

After you have created the Visual Basic interface and developed the logic for the action objects, using IPO tables and pseudocode, you must write procedures in Visual Basic for each action object. This code should provide instructions to the computer to carry out one or more of the six operations listed earlier. Although creating the interface is important, writing the code is the essence of developing an application.

After you have written the code for an action object, the second part of this step is to test that object and correct any errors; you should not wait until the entire project is complete. You should use the interactive capabilities of Visual Studio to test the code of each and every object as it is written. This process is referred to as **debugging**, and it involves trying to remove all of the errors, or **bugs**.

**Step 5: Test the Overall Project**

After you have tested the code for each action object individually, the next step is to test the overall project and correct any errors that may still exist or that may be the result of incorrect communication between objects. At this stage, it is necessary to determine whether the results obtained from the project meet the objectives outlined in the problem definition step. If the project does not meet the final user’s needs, then the developer must analyze the results and the objectives to find out where they diverge. After the analysis, the developer should trace through the program development procedure and correct the algorithm, IPO tables, pseudocode, and final code for one or more objects to find the cause of the difference between the objectives and the final project.

**Step 6: Document the Project in Writing**

An important part of writing any computer software is the documentation of the software. Documentation helps users by providing instructions and suggestions on using the software. Documentation helps other programmers who may need to make changes or correct the programs.

Internal documentation usually includes comments within the program that are intermingled with the program statements to explain the purpose and logic of the program elements. This type of documentation is essential to the maintenance of software, especially by someone other than the original programmer. By being able to read the original programmer’s purpose for a part of a program or a program statement, a different programmer can make any needed corrections.
or revisions. Without internal documentation, it can be extremely difficult for anyone to understand the purpose of parts of the program. And if a programmer is unclear about what’s going on in the program, making needed changes is very difficult.

Written documentation includes books, manuals, and pamphlets that give instructions on using the software and also discuss the objectives and logic of the software. The documentation should include a user's guide and programmer documentation. The user’s guide provides complete instructions on accessing the software, entering data, interpreting output, and understanding error messages. The programmer documentation should include various descriptive documents that allow for maintenance of the software. These may include pseudocode of sections of the program, a listing of the program, and a description of required input values and the resulting output.

### FOR EXAMPLE

#### Creating an Area Calculator

Say that you work for a construction company. You need to build a program that can accept the length and width of a room, in feet and inches, and calculate its area in both inches and feet and inches. The program will be a Windows application.

You analyze the program requirements. You determine that the application needs two actions: a Calculate action and an Exit action.

The Calculate action needs to accept four input values: WidthFeet, WidthInches, LengthFeet, and LengthInches. These values will be entered by a user. You also determine that the program must output two values: TotalAreaInches and TotalAreaFeet. TotalAreaFeet is a decimal value that is the area calculated in square feet. TotalAreaInches is a whole number that is the area calculated in square inches.

You sketch the IPO table shown in Figure 1-10.

![Figure 1-10](Continued)
You write the following pseudocode for the procedure:

```
Begin Procedure
Input LengthFeet, LengthInches, WidthFeet, WidthInches
TotalLengthInches = LengthFeet * 12 + LengthInches
TotalWidthInches = WidthFeet * 12 + WidthInches
TotalAreaInches = TotalLengthInches * TotalWidthInches
TotalAreaFeet = TotalAreaInches / 144
End Procedure
```

You sketch the interface and identify four fields used for input, two fields use for output, and two buttons.

You implement the interface by using a Visual Basic Windows form. You add TextBox controls for the Input fields and Label controls for the output fields. You also add Label controls to identify each input field. Next, you write and test the code for both buttons. After testing all functionality, you create the documentation.

SELF-CHECK

1. Describe how an application written in Visual Basic 2005 is converted to machine language.
2. Compare a class, an object, and an instance.
3. Define encapsulation.
4. Identify the six steps in creating a Windows application.

1.2 Working with Visual Studio

Visual Studio is the integrated development environment (IDE) used to create programs with Visual Basic. An IDE is an application that provides tools for creating, debugging, and modifying application code. In the next sections, you will learn how to install Visual Studio and how to use the online help system.

1.2.1 Installing Visual Studio

You can avoid frustrations when installing Visual Studio by first making sure your system can run Visual Studio. You don’t want to begin the installation only
to have the Setup Wizard display the dreadful message that it cannot install the application because your computer is not capable of supporting it. Visual Studio has the following hardware requirements:

▲ 600MHz or faster CPU
▲ 192MB or more of RAM
▲ 2GB of disk space

Your computer probably meets or surpasses these requirements if you purchased it within the past few years.

Typically, software—not hardware—is at the center of installation troubles. Not everyone keeps up-to-date with Microsoft’s latest service packs. A service pack is a group of enhancements to Windows that usually patch security gaps in the original Windows release. Visual Studio requires you to have certain service packs installed on your computer before running the Setup Wizard. You can download service packs free of charge from www.microsoft.com. To install Visual Studio, you must be running one of the following:

▲ Windows 2000 with Service Pack 4 (SP4)
▲ Windows XP with SP2
▲ Windows Server 2003 with SP1

The installation of Visual Studio is straightforward and should take an hour to an hour and a half. You begin the installation by closing down any applications that are open on your computer. Next, you place Installation Disc 1 into your CD drive, and Windows automatically starts the Setup Wizard, which walks you through the installation. You also use the Setup Wizard to uninstall Visual Studio and to add or remove components to the installation that have not yet been installed. During the installation process, you have the opportunity to install all the components, select the components you want to install, or accept the default components that Microsoft suggests you install. You probably want to install the components that are selected by default. If you need a feature that you do not install now, sometime in the future, you can use the Setup Wizard again to install any component that was not originally installed. You will probably not need to add a new component later, however, because nearly all the components that you need are part of the default installation.

1.2.2 Getting Help with Visual Studio and Visual Basic

As you learn to program in Visual Basic, you may often need to look up how to do something in the documentation. This is normal. Even experienced programmers rely heavily on the documentation to find the best way to accomplish
a task. The .NET Framework is a rich library with a huge number of objects. No one can be expected to remember how to use them all. Fortunately, Visual Studio and Visual Basic provide powerful tools for gaining access to the information you need.

The most straightforward use of the documentation requires little more than clicking on or in the object that you have a question about and pressing the F1 key to launch context-sensitive help. For instance, in any project, you can click somewhere on an object and press F1. The Microsoft Document Explorer launches documentation about that object loaded. The Document Explorer offers a sophisticated set of tools, mostly represented in the Document Explorer toolbar, shown in Figure 1-11. The tools you find there give you various ways to access the documentation, as follows. The toolbar’s first section has navigation buttons, a refresh button, and font size maintenance. It also has the following buttons:

▲ **How Do I:** The How Do I button has preset questions that relate to the selected topic and may help with general queries about certain types of development. If you are stuck, give it a try.

▲ **Search:** Clicking the Search button allows for phrase searching.

▲ **Index and Contents:** The Index button and Contents button allow browsing through the index or table of contents of the documentation.

▲ **Help Favorites:** You can save favorites in the Help Favorites just as you can in Internet Explorer. The double arrow is handy: It synchronizes the Contents panel with the page you are currently viewing.

▲ **Ask a Question:** The Ask a Question button takes you directly to the NNTP Newsgroups (using a web-based viewer) hosted by Microsoft, where you can ask questions and have them answered by Microsoft MVPs, authors, and other experts.

These options are only one part of the documentation in Visual Studio. IntelliSense shows information from the user documentation when you rest the mouse cursor over a piece of code. The Properties window shows the documentation for a property when it is selected. And don’t overlook the online tools that Microsoft provides. You can open the Help menu and select Technical Support to access a wealth of information available on the Web, right from inside Visual Studio.
1.3 Creating Solutions and Projects

When you create an application, you create multiple files. Some of these files contain source code (the human-readable instructions the program will execute) and are compiled as part of your application. You might also create graphics files, documentation files, and data files. Visual Studio allows you to organize all the files associated with an application into a project. A single project is suitable for a simple application, but a more complex application might include multiple projects. A solution allows you to organize the files for one or more projects.

1.3.1 Creating a Windows Application Project

All applications are created as a project. It is fairly easy to create a new project, as you’ll learn shortly. You use the New Project dialog box (shown in Figure 1-12), in which you select a project type and template. Visual Studio enables you to create projects in Visual Basic and other programming languages, as well. Here, you select Visual Basic as the project type.

In the New Project dialog box, several templates are displayed in the Templates window. A template contains the basic ingredients needed to create...
a specific kind of Visual Basic program. You see the templates Windows Application, Class Library, Windows Control Library, ASP.NET Web Application, ASP.NET Web Service, and others. You’ll start learning to program by using the **Windows Application** template. A Windows application creates an .exe when it is compiled.

After you select Windows Application as the template for your project, a new Windows application project opens, showing a blank form and the Properties window. The **blank form** is where you create the screens that allow a user to interact with your application. You use the Properties window to define characteristics of the blank form. The default settings are sufficient for most applications you build. However, you probably want to change the text that appears at the top of the form to something that better describes the purpose of the form.

### 1.3.2 Using the Solution Explorer Window

Solutions and projects hold forms and other project files in folders. In fact, solutions and projects are represented by folders in the Visual Studio Projects directory of your My Documents folder. The **Solution Explorer** is Visual Studio’s tool that allows you to manage the files in a project.

If you envision your projects like folders, you can imagine that you would group similar folders together in a folder. That’s what solutions do: They are both physically and logically folders full of projects.
1.3 CREATING SOLUTIONS AND PROJECTS

Figure 1-13 shows the important files in a project, along with a whole bunch of buttons above to help to manage them.

To open a file, you double-click the file’s icon or name. To rename, copy, or delete a file, you right-click the file and choose the desired action from the context-sensitive menu that appears. In the Solution Explorer, you can also make a new folder and move files into it. And you can right-click a project to add a new form or support file.

The buttons above the files themselves are the most significant part of the Solution Explorer. They are, from left to right:

▲ Properties: Opens the Properties window.
▲ Show All Files: Shows hidden files.
▲ Refresh: Checks the solution folder for new files that may have been added by another tool.

View Designer: Opens the selected file in Design view. You use Design view to design what your application looks like.

View Class Diagram: Opens the Class Designer for the project. The Class Designer allows you to view how the classes in the project are related. You’ll learn about classes later in the book. For now, just think of them as being definitions for objects in a project.

1.3.3 Customizing the Visual Studio Environment

You can customize the Visual Studio environment to make it easier to work in. All the tools, windows, and views are part of an Integrated Development Environment (IDE) that each user can customize. This makes organization of your personal development space a lot easier.

Most often, you will want to move around a tool window to put it in a more convenient spot. You can display a tool window in the following ways:

Floating: A floating window is very mobile. You can drag it around by its handles to place it anywhere you want.

Dockable: When you drag a dockable window, Visual Studio gives you the option to dock the window. When you dock a window, you anchor it to an edge so that it always stays in that location. There are several options for docking a window. If you drag the window over the top, bottom, left, or right arrow, it docks to that side. When a window is docked, it has a thumbtack that you can pin or unpin. When pinned, it stays on the side, moving the Design view over. When unpinned, it slides out of the way, toward the side it is pinned to.

Tabbed: You can drag a window to the center and have it become a tab at the top of the view window, like the Form1.vb and Form2.vb files in Figure 1-14. If you drag the window to the center of the four-pointed star, it makes the window a tab in the other central windows.

FOR EXAMPLE

Configuring Your Environment

Say you are creating an application and want to cause the Properties window to float above other windows so you can move it to a convenient place. You click its title bar and drag it away from its docked position. Later, you decide that you want it above the Solution Explorer window, so you move it over the Solution Explorer window and highlight the arrow that points up.
1.4 Understanding Forms

When you create a Windows application project, Visual Studio adds a default form to the project. You use a form to create the program's user interface. Each form in an application acts as a window or a dialog box.

In the following sections, you'll learn how to set properties on the default form. You'll also learn how to create additional forms in a project.
1.4.1 Defining a Form

A form is a member of the System.Windows.Forms.Form class. System.Windows.Forms is a namespace. The .Net Framework uses namespaces to organize and group classes that are related. A form has all the properties, methods, and events defined by the Form class. For instance, the Form class has a Show method that allows you to display the form. Each object based on the Form class in a project has a name. You use that name when accessing the form in code.

1.4.2 Working with Forms in Design View

When you open a Windows application project, you see the Design view. The Design view is where the GUI work takes place. Generally speaking, anytime you are working with what the form looks like, not code, you are working with the Design view. The term designer window refers to the actual place you do the work. The term Design view refers to the state of the file.

In general, Design view is the core part of Visual Studio. Many of the other tools in Visual Studio depend on the Design view, in fact, and are disabled when you use another view, such as Code view, to work on a project.

The designer tabs have the word [Design] in the tab name, as shown in Figure 1-15, to indicate that you are using the Design view. Tabs are used in

Figure 1-15

A form in Design view.
1.4 UNDERSTANDING FORMS

the Design and Code views. A light gray tab represents views of files that are open but not active. An asterisk (*) next to the filename means that you’ve made changes but not yet saved the file.

The white tab is active and contains the editable form. When you have more than one form open, you can edit only the active form. You can drag the tabs to the left and right to change their order. Right-clicking a tab opens a menu from which you can choose several screen-management options, as shown in Figure 1-16. You can save and close files from this menu, or get information, such as the current path or the containing folder. You can display files as horizontal or vertical tab groups. Tab groups make it easier to copy information from one form into another. For example, you can have one set of pages on the top half of the screen and another on the bottom half, and you can copy from one and paste into the other without changing screens.

1.4.3 Setting Form Properties

A form has multiple properties that determine how it looks and acts. As you have already learned, a property is a value that describes an object. You can set
properties at design time or at runtime. Setting a property at design time means setting it when you are creating the application. Setting a property at runtime (runtime refers to the state an application is in when it is executing) means setting it by writing code.

You set a property at design time by using the Properties window. The Properties window, shown in Figure 1-17, is usually on the right side of the screen. It contains all the editable values associated with a form. If the Properties window isn’t on the right side of the screen, you can find it by opening the View menu and choosing Properties, or you can press F4.

At the top of the Properties window, you see the form element whose properties are listed. The Properties window has a toolbar. For now, just notice that the leftmost button causes the properties to be listed by type. The second button causes the properties to be listed in alphabetical order. A description of the selected property is shown beneath the list of properties.

You set a property at runtime by using code like this:

```
form_name.propertyname = propertyvalue
```
1.4 UNDERSTANDING FORMS

For example, to set the Text property of Form1 to “My First Program,” you use the following code:

```csharp
Form1.Text = "My First Program"
```

The Text property sets the text that displays in the form’s title bar. Notice that “My First Program” is enclosed in quotation marks. That’s because it is a string literal value. **String** is a programming term for textual data. A **literal** is a value that you type directly into code.

A form has a large number of properties. You’ll have a chance to use many of them as you work your way through this book. For now, let’s just look at a few of the important ones:

▲ **(Name):** Specifies the name you will use to reference an object in code.
   The (Name) property is really not a property at all. You can change (Name) in the Properties window, but you cannot change it in code.
▲ **BackColor:** Sets the background color of the form.
▲ **ControlBox:** Is set to True or False to determine whether the icon to the left of the form’s title should be displayed. ControlBox shows a Windows-controlled menu when it is clicked.
▲ **FormBorderStyle:** Determines whether the form is resizable, a dialog, or a toolbox window.
▲ **HelpButton:** Determines whether a Help button appears on the title bar.
▲ **MaximizeBox:** Determines whether the Maximize button appears on the title bar.
▲ **MaximumSize:** Determines the maximum size for the window.
▲ **MinimizeBox:** Determines whether the Minimize button appears on the title bar.
▲ **MininumSize:** Determines the minimum size for the window.
▲ **ShowInTaskbar:** Determines whether the window should be listed in the Windows taskbar.
▲ **Size:** Determines the starting size of the window. Sets the height and width.
▲ **StartPosition:** Determines where the form is displayed when it is first shown.

1.4.4 Adding a Form to a Project

Most Windows applications you build require multiple forms. For example, you might have an application that has a main form but that allows users to configure the application by using a Properties dialog. The Properties dialog would be a separate form.

To add a form to a project, you right-click the project name in Solution Explorer, open the Add menu, and choose Windows Form. Visual Studio has a number of **templates** for standard forms, as shown in Figure 1-18. You can choose one of them or you can choose Windows Form to add a blank form. You give the form a name and click Add.
If your project has multiple forms, you need to designate one as the startup form. The **startup form** is the one that displays when the application first launches. To designate a startup form, you right-click the project in Solution Explorer and choose Properties. On the Application tab, you select the appropriate form from the Startup Form drop-down list, as shown in Figure 1-19.

**Figure 1-19**

Setting the startup form.
### FOR EXAMPLE

#### Using Multiple Forms

Say that you are writing an application that allows a user to perform various types of measurement conversion. The main form has four buttons: Distance, Temperature, Volume, and Weight. Each button displays a form that performs the relevant conversions. You configure the main form as the startup form.

You set the Text property and the (Name) property for each form as shown in Table 1-1.

<table>
<thead>
<tr>
<th>Form</th>
<th>Text</th>
<th>(Name)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main form</td>
<td>Measurement Converter</td>
<td>frmMain</td>
</tr>
<tr>
<td>Distance calculations form</td>
<td>Distance Calculator</td>
<td>frmDistance</td>
</tr>
<tr>
<td>Temperature calculations form</td>
<td>Temperature Calculator</td>
<td>frmTemp</td>
</tr>
<tr>
<td>Volume calculations form</td>
<td>Volume Calculator</td>
<td>frmVolume</td>
</tr>
<tr>
<td>Weight calculations form</td>
<td>Weight Calculator</td>
<td>frmWeight</td>
</tr>
</tbody>
</table>

The forms do not require minimize or maximize buttons. Therefore, you set the MinimizeBox and MaximizeBox properties to False.

To show the Distance Calculator form, you use the following code:

```vbnet
frmDistance.Show
```

### SELF-CHECK

- List some reasons you might need multiple forms in a project.
- To enable users to minimize, maximize, and resize a window, what properties should you set?
- Describe a situation in which you would set the Text property of a form at runtime instead of at design time.

### 1.5 Using Controls

Controls are special objects that allow you to add functionality to a form. Like forms (and pretty much everything else in Visual Basic), they are defined by classes. This means they have properties, methods, and events.
In the following sections, you’ll learn how to add controls to a form. Although a large number of controls are available, the following sections focus on three simple but commonly used controls: the Label, Textbox, and Button controls.

1.5.1 Using the Toolbox to Add Controls

On the left side of the project window is a tab called the toolbox that expands into a list of tools when you hover the mouse over it. You use these tools to transform a blank form into an interactive screen for an application. The tools are organized into groups. You select the name of the group to see a list of tools in that group. You are likely to use the Common Controls tool group (see Figure 1-20) more than the other groups because it contains buttons, check boxes, and other controls that are frequently used in applications.

You can keep the toolbox open by first opening it with the mouse and then clicking the pushpin icon at the top of the toolbox. You can then drag and drop tools onto your form without having to open and close the toolbox each time. You click the pushpin icon again to return the toolbox to its tab position.

When you want to use a control on a form, you drag that control from the toolbox and drop it onto the form. You can adjust the control on the form by first selecting the control. Sizing handles appear around the control. You can then do any of the following:

Figure 1-20

The Common Controls tool group.
1.5 USING CONTROLS

▲ Change the size of the control by dragging the sizing handles.
▲ Reposition the control on the blank form by dragging the center of the control to the new location.
▲ Remove a control by selecting it and pressing Delete.

You can adjust the size of a control on a form to fit your design by using the sizing handles that appear when you select the control. Visual Basic changes the Height and Width properties of the control to reflect the new size as you resize the control.

There are typically eight sizing handles around a control (see Figure 1-21). One sizing handle is in each corner of a control, and another is on each side of the control.

The right- and left-side sizing handles move the corresponding size out and in to make the control wider or narrower. The top- and bottom-side sizing handles move the corresponding side up and down, making the control taller or shorter. The corner sizing handles resize the control in any direction. As a corner sizing handle is dragged on an angle, the sides of the control attached to that handle move up and out or down and in, depending on the direction that the sizing handle is dragged.

Professional developers usually keep the size of a type of control the same throughout an application. For example, all Button controls are adjusted to the size of the Button control that has the longest button text (i.e., the text that appears on the button).

1.5.2 Setting Control Properties

Each control has a set of properties that define a characteristic or behavior of the control. The kinds of properties that are available depend on the type of control. For example, the FontColor property of the Label control sets the color of the label. The list of properties appears in the Properties window when you select
the control. You can select a control from the drop-down list in the Properties window, or you can select a control by clicking it on the form.

Like a form, each control you add to a form also has a name. You can set the (Name) property for the control at design time through the Properties window, but you cannot set it at runtime.

You can also set a control’s location on the form and the control’s size through its properties. The control’s location on the form is set through the X and Y properties. To access them in the Properties window, you expand Location. X and Y refer to the coordinates of the control’s top-left corner in relation to the form’s top-left corner. You can adjust the size of a control by setting the Width and Height properties, which are located in the Properties window under Size. You can also set them in code. However, to do so you use the Top property and the Left property:

```vba
lblName.Top = 40
lblName.Left = 10
```

This code sets the leftmost pixel of the control to 40 pixels from the edge of the form and the topmost pixel of the control to 10 pixels from the top of the form.

**1.5.3 Adding a TextBox Control to a Form**

A TextBox control enables a user to enter text into a field. Text can include letters, numbers, and other characters. You use a TextBox control anytime a user needs to be able to type a value.

You should rename the TextBox control, using a name that reflects the content of the text box. For example, you might name a TextBox control txtCustomerFirstName if the text box will contain the first name of a customer. Naming your TextBox controls in this way makes them more meaningful when you refer to them in your code. Using a prefix that identifies the control as a TextBox control (e.g., txt) is also useful for helping you understand the code you have written. You can change the name of a TextBox control by selecting the control, right-clicking, and clicking Properties from the pop-up menu. Then you type the new name in the (Name) property.

By default, the TextBox control accepts one line of text from the user and has only the two horizontal sizing handles. However, you can have the user enter multiple lines into a TextBox control by selecting the right arrow located near the upper-right corner of the text box. The Tasks pop-up menu appears (see Figure 1-22). You select the MultiLine check box to convert from a single-line text box to a multiline text box. After you make this selection, the eight sizing handles appear around the TextBox control. You can use them to resize the TextBox control to display multiple lines of text.

You access the text entered in a TextBox control by using the TextBox control’s Text property. For example, to set the value of txtCustomerFirstName in code, you use this:

```vba
txtCustomerFirstName.Text = "John"
```
You can also retrieve the text a user types into a TextBox control and use it to set another property. For example, to set the Text property of txtHomeState to the value stored in txtWorkState, you use this:

```vbnet
txtHomeState.Text = txtWorkState.Text
```

By default, a user can edit the contents of a TextBox control. However, sometimes you place information into a TextBox control that you do not want the user to change, such as a customer account number. You can prevent the user from changing the contents of a TextBox control by setting the ReadOnly property of the TextBox control to True. The ReadOnly property tells Visual Basic whether to permit changes to the contents of the TextBox control. You set the ReadOnly property by using the following code:

```vbnet
txtAccountNumber.ReadOnly = True
```

Notice that you do not enclose True in quotes. True is a literal value, but it is not a string. It is a Boolean. A **Boolean** is a special type of property that can only be set to True or False.

The default values of the TextBox control’s properties may or may not be appropriate for your application. You can change the default values by using the Properties list. You highlight the property and then enter the new value for the property.

The following are some properties that are commonly changed:

- **MaxLength**: Sets the maximum number of characters a user can enter into the text box.
- **Multiline**: Determines whether one or multiple lines can be entered into the text box. You set the Multiline property to False for a single line or
True for multiple lines. If you are allowing multiple lines, you might also want to enable either horizontal or vertical scrollbars.

▲ **ScrollBars**: Determines whether horizontal scrollbars, vertical scrollbars, both, or neither are added to the text box.

▲ **Font**: Designates a font for text entered into the text box; you can choose from a list of available fonts.

▲ **ForeColor and BackColor**: Sets the foreground and background colors for the text box, respectively; you can choose a color from a palette of available colors.

▲ **TextAlign**: Aligns text within the text box; options are Left, Center, and Right.

### 1.5.4 Adding a Label Control to a Form

A **Label control** is used to display text on a form that identifies controls, provides instructions for using the application, and generally communicates to the user. For example, a label is typically used to identify a TextBox control; for example, the text “Customer First Name:” could be a Label control placed to the left or at the top of a TextBox control on a form.

You can modify the appearance of a Label control by changing its properties in the Properties window. You can use the Text property to change the text of a Label control and the ForeColor and BackColor properties to change the foreground and background colors of a Label control, respectively. You use the Font property to change the font of the Label control’s text.

By default, a Label control’s AutoSize property is set to True. This means that the control is automatically resized horizontally as you enter text. You can move the control, but you cannot change its size. When the AutoSize property is set to True, all text is displayed on a single line. If you want to wrap text or manage the size of the Label control yourself, you set the AutoSize property to False.

### 1.5.5 Adding a Button Control to a Form

A **Button control** simulates a push button on a form and is used to start, confirm, or cancel an operation. For example, you can use an OK button on a dialog box. The user confirms an action by clicking OK.

You should rename the Button control to reflect the action of the button (e.g., btnOK, btnCancel). You rename a Button control by setting its (Name) property.

You can modify the appearance of the label on a Button control by changing other Button control properties. You change the text of the label on a Button control by using the Button control’s Text property. You change the font of the label by choosing a new value for the Font property. You click the Font property to display a list of other available fonts. You can reposition a label on a Button control by using the TextAlign property. You can click the TextAlign
property to see a diagram that illustrates where you can position the text. The choices are top, middle, or bottom and left, center, or right. You specify values by using the `ContentAlignment` enumeration. An enumeration is a set of descriptive words mapped to numeric values. For example, to have the text aligned bottom center, you use this code:

```
btnSetLabel.TextAlign = ContentAlignment.BottomCenter
```

You can prevent a Button control from being accidentally pushed by setting the Enabled property of the Button control to False. There are two ways to set the Enabled property: You can either open the Properties window and change the Enabled property to False or, from you can use the following code:

```
btnOK.Enabled = False
```

In this case, `btnOK` is the Button control name, and Enabled is the name of the property. You enable the Button control by changing the Enabled property to True or by adding the following code:

```
btnOK.Enabled = True
```

### FOR EXAMPLE

**Designing a Temperature Conversion Form**

Say that you are writing an application that allows a user to perform various types of measurement conversion. The main form has four buttons: Distance, Temperature, Volume, and Weight. Each button displays a form that performs the relevant conversions. You are designing the temperature conversion form. One way to design it would be to add one TextBox control named `txtTemp`, one Label control named `lblConvertedTemp`, and two Button controls: `btnConvertToCelsius` and `btnConvertToFahrenheit`.

- Identify the properties you would use to reposition a control in code.
- Compare and contrast the Label control and the TextBox control. Give examples of when you would use each.
- List some uses of the Button control.
- Describe the steps you would take to change the text of a Label control in code.
1.6 Writing an Event-Driven Program

A Visual Basic application is an event-driven application. As you have learned, an event is something that happens when an application runs, such as the clicking of a button. You can write code to respond to an event that is important to an application and ignore other events because Visual Basic has a default response for every event.

For example, you can write code that executes when the user clicks a button because that event is important to the application. You do not have to write code that executes when a form is loaded unless you want something special to happen when that event occurs.

1.6.1 Understanding Form Events

A form event is an event that occurs when something happens to a form. The default form event is the Load event, which occurs when a form is loaded into memory. An event is associated with a special type of subprocedure known as an event procedure or event handler. The syntax (i.e., the general grammar for using a subprocedure, function, or method) for a form's Load event procedure looks like this:

```vbs
Private Sub frmMain_Load (ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
'Enter your code here
End Sub
```

In this case, the event belongs to a form named frmMain. Notice that the event is named with the name of the form, an underscore, and then the name of the event. A space and then an underscore character are used to wrap the text to the next line. Visual Basic sees each line as a separate line of code unless an underscore is used. In this case, we have to wrap to the next line so that the code can fit on the page of the book. In the development environment, there is no wrap.

Next, you see two arguments in parentheses. You shouldn’t worry too much about them for now. An argument is used to pass data to a procedure. In this case, the operating system is passing data to the frmMain_Load event procedure. You’ll learn more about arguments later in the book.

You know that this is an event procedure because of the Handles keyword. The Handles keyword identifies the event the procedure will handle. In this case, it will handle the Form class's Load event. MyBase is a special word that refers to an object’s base class (i.e., the class on which the object is based). The base class for frmMain is System.Windows.Forms.Form.

The following line of code has been added:

'Enter your code here
This line is not really code; it is a comment. A comment is a line of code that is not compiled and executed. Instead, you use it to document your code. In Visual Basic, you add a comment to your code by preceding the line with an apostrophe. A quick way to comment lines of code is to select them and click the Comment Out button on the toolbar (see Figure 1-23). The Uncomment button removes the comment apostrophes.

The event procedure terminates with End Sub. This tells the compiler that the subprocedure is over.

The following are some other form events you might use:

▲ **Activated**: Occurs when a form becomes the active form (i.e., is brought into the foreground).
▲ **Shown**: Occurs when a form displays.
▲ **Deactivated**: Occurs when a different window becomes active.
▲ **FormClosing**: Occurs before a form closes.
▲ **Move**: Occurs when a user moves the window.

You add code to a form event if you want to do something in response to the event. For example, you might want to display a dialog, asking a user to confirm whether a form should close. To do so, you add code to the FormClosing event.

### 1.6.2 Using the Code Window

You place code that you want to execute in response to an event in the event procedure. When you double-click the form or a control, the event procedure for the default event displays. If you have not added code to the default event, a new event procedure is created. You can avoid creating a new event procedure by clicking View Code. Visual Studio displays the code page without creating a new event procedure. The code window is shown in Figure 1-24.

If you add an event procedure for a control or a form and then change that control's name, the name of the event procedure does not change automatically. You need to change the name in the event procedure manually.

The left drop-down box at the top of the window contains a list of objects on the form and the name of the form. If the object in the left box is a control or a form's events, the right drop-down box contains a list of events associated
with the events in the left box. You can select an event from the list, and Visual Basic .NET displays the corresponding event procedure.

The problem with using Code view is that you need to know what to type. To help get started, you can use some very productive code-based tools, such as IntelliSense. When you type an object name and then a period, a special context-sensitive menu (the IntelliSense menu) appears, as shown in Figure 1-25. This menu shows the properties and methods the object supports.

1.6.3 Understanding Control Events

Most controls support events. One of those events is the default event, which is the event procedure that is created when you double-click a control in Design view. The following are the default events for the controls we have discussed so far:
1.6 WRITING AN EVENT-DRIVEN PROGRAM

▲ **Button:** The Click event occurs when the user clicks the left mouse button on the button.

▲ **TextBox:** The TextChanged event occurs each time the Text property changes.

▲ **Label:** The Click event occurs when the user clicks the left mouse button on the label.

You add code to control events to cause something to happen when an event occurs. For example, to cause the Text property of the Label control to be set to the same value as the Text property in the Label control when a user clicks the btnSetLabel Button control, you add the following code:

```csharp
Private Sub btnSetLabel_Click (_
    ByVal sender As System.Object, ByVal e As _
    System.EventArgs) Handles btnSetLabel.Click
    lblName.Text = txtName.Text
End Sub
```

Controls support other types of events as well. For example, the TextBox control supports a LostFocus event that occurs when the user moves focus away from the control by pressing Tab or clicking somewhere else on the screen. You might want to use the LostFocus event to process the text entered instead of using the TextChanged event. The TextChanged event occurs each time a character in the control is changed. It is important to understand the circumstances under which an event fires to make sure it is the best event to execute a specific piece of code.

1.6.4 Testing Your Code

You can run an application inside the IDE to test it. This is known as debugging. To start the application, you either open the Debug menu and choose Start Debugging, click the Start Debugging icon on the toolbar (refer to Figure 1-16), or press F5. Your application starts, and you can test each feature where you have added code to make sure it works the way it should.

When you have finished testing, you can return to the design environment by doing one of the following:

▲ Close the application.

▲ Open the Debug menu and choose Stop Debugging.

▲ Press Ctrl+Alt+Break.

▲ Click the Stop Debugging button on the toolbar.
FOR EXAMPLE

Coding the Temperature Conversion Form

Say you are writing an application that allows a user to perform various types of measurement conversion. The main form has four buttons: Distance, Temperature, Volume, and Weight. Each button displays a form that performs the relevant conversions. On the Temperature Conversion form, you add one TextBox control named txtTemp, one Label control named lblConvertedTemp, and two Button controls: btnConvertToCelsius and btnConvertToFahrenheit. You add code to the Click event of each button to retrieve data from TextBox.Text, perform the necessary calculation, and set the Text property of lblConvertedTemp to the converted value.

SELF-CHECK

• List the steps you would take to view the code for the Click event for btnStart.
• List the steps you would take to add code to an event procedure that is not a control’s default event procedure.
• Describe how IntelliSense can help you program faster and more accurately.
• Identify each part of the event procedure syntax.
• List the steps you would take to test the txtAddress_TextChanged event procedure.

SUMMARY

In this chapter, you have learned the fundamental concepts and procedures necessary to begin programming in Visual Basic. You have learned about fundamental programming concepts, including event-driven programming and object-oriented programming. You have also learned the steps you must take to create a program. You have learned that Visual Studio is an IDE that provides you with the tools you need to write code and compile a Visual Basic application. You have also learned that Visual Studio uses projects to organize the files needed for an application and solutions to group multiple related projects. You have also learned that forms are used to provide the user interface for a Windows application and that a form is actually an object of the System.Windows.Forms.Form
class. You have learned how to add TextBox, Label, and Button controls to a
form and set control properties to create a user interface. You have also learned
that Windows programs are based on events, and you have learned how to write
a simple event procedure.

### KEY TERMS

<table>
<thead>
<tr>
<th>Action object</th>
<th>Event-driven programming</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm</td>
<td>Event handler</td>
</tr>
<tr>
<td>Argument</td>
<td>Event procedure</td>
</tr>
<tr>
<td>Base class</td>
<td>Form</td>
</tr>
<tr>
<td>Binary number system</td>
<td>Form event</td>
</tr>
<tr>
<td>Blank form</td>
<td>Handles keyword</td>
</tr>
<tr>
<td>Boolean</td>
<td>High-level language</td>
</tr>
<tr>
<td>Bug</td>
<td>Inheritance</td>
</tr>
<tr>
<td>Button control</td>
<td>Input/Processing/Output (IPO) Table</td>
</tr>
<tr>
<td>Bytecode</td>
<td>Instance</td>
</tr>
<tr>
<td>Class</td>
<td>Integrated Development</td>
</tr>
<tr>
<td>Class Designer</td>
<td>Environment (IDE)</td>
</tr>
<tr>
<td>Code</td>
<td>IntelliSense</td>
</tr>
<tr>
<td>Code view</td>
<td>Interactive development</td>
</tr>
<tr>
<td>Common Language Runtime (CLR)</td>
<td>IPO table</td>
</tr>
<tr>
<td>Comment</td>
<td>JIT compiler</td>
</tr>
<tr>
<td>Compiler</td>
<td>Label control</td>
</tr>
<tr>
<td>Control</td>
<td>Literal</td>
</tr>
<tr>
<td>Control structure</td>
<td>Machine language</td>
</tr>
<tr>
<td>Debugging</td>
<td>Method</td>
</tr>
<tr>
<td>Decision control structure</td>
<td>Microsoft Intermediate Language</td>
</tr>
<tr>
<td>Design view</td>
<td>MSIL</td>
</tr>
<tr>
<td>Designer window</td>
<td>MyBase</td>
</tr>
<tr>
<td>Dialog box</td>
<td>Namespace</td>
</tr>
<tr>
<td>Document Explorer</td>
<td>.NET Framework</td>
</tr>
<tr>
<td>Dock</td>
<td>Object</td>
</tr>
<tr>
<td>Encapsulation</td>
<td>Object-oriented language</td>
</tr>
<tr>
<td>Enumeration</td>
<td>Pinned</td>
</tr>
<tr>
<td>Event</td>
<td>Polymorphism</td>
</tr>
<tr>
<td>Procedural programming</td>
<td>Startup form</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Procedure</td>
<td>String</td>
</tr>
<tr>
<td>Programming language</td>
<td>Syntax</td>
</tr>
<tr>
<td>Project</td>
<td>Tab group</td>
</tr>
<tr>
<td>Property</td>
<td>Template</td>
</tr>
<tr>
<td>Pseudocode</td>
<td>TextBox control</td>
</tr>
<tr>
<td>Repetition control structure</td>
<td>Toolbox</td>
</tr>
<tr>
<td>Runtime</td>
<td>Variable</td>
</tr>
<tr>
<td>Sequence control structure</td>
<td>Visual Basic</td>
</tr>
<tr>
<td>Service pack</td>
<td>Visual Studio</td>
</tr>
<tr>
<td>Solution</td>
<td>Window</td>
</tr>
<tr>
<td>Solution Explorer</td>
<td>Windows Application</td>
</tr>
<tr>
<td>Source code</td>
<td></td>
</tr>
</tbody>
</table>
ASSESS YOUR UNDERSTANDING

Go to www.wiley.com/college/petroutsos to evaluate your knowledge of the basics of programming and Visual Basic. 
Measure your learning by comparing pre-test and post-test results.

Summary Questions

1. Source code must be compiled before it can execute. True or False?
2. A computer performs all processing through arithmetic. True or False?
3. Which of the following is not a control structure?
   (a) decision control structure  
   (b) input control structure  
   (c) repetition control structure  
   (d) sequence control structure  
4. Each bit of data in a computer is represented by a number in the base-10 system. True or False?
5. Procedural programming is used for Windows applications that have a GUI. True or False?
6. A/an _____________ is an object loaded in memory.
   (a) class  
   (b) procedure  
   (c) instance  
   (d) variable  
7. When programming with Visual Basic, you must wait to test your application until every object has been coded. True or False?
8. Which of the following do you identify during the problem identification step?
   (a) action objects  
   (b) controls you will add to the form  
   (c) calculations you will perform  
9. Creating an interface is the first step in the programming process. True or False?
10. When discussing program design, what does the acronym IPO stand for?
    (a) Internet public objects  
    (b) input/processing/output  
    (c) international programming organization  
    (d) initial process organization
11. If you have designed a good-looking interface, testing the code is not important. True or False?
12. At which stage of a project do you determine whether the results obtained meet the objectives outlined in the problem definition step?
   (a) documentation
   (b) coding
   (c) testing the overall project
   (d) designing the interface
13. If a program has enough comments, they are the only documentation required. True or False?
14. Which of the following provides a program written in Visual Studio 2005 with access to operating system features and libraries for performing various functions?
   (a) IntelliSense
   (b) the CLR
   (c) the .NET Framework
15. Which of the following operating system configurations will run Visual Studio 2005?
   (a) Windows XP with SP1
   (b) Windows Server 2003 with no service pack
   (c) Windows 2000 Professional with SP4
16. You have added a control to a form. How can you easily get help on how to use the control?
   (a) Select the control and press F1.
   (b) Right-click the control and choose Help.
   (c) Right-click the control and choose View Code.
   (d) Select the control and press F2.
17. A Windows application project creates a dynamic link library (DLL) when compiled. True or False?
18. The Solution Explorer window lists all the files in a project. True or False?
19. A docked window in Visual Studio can be pinned or unpinned. True or False?
20. Which namespace contains the Form object used in a Windows application?
   (a) System.Windows.Forms
   (b) System.Forms
   (c) Windows.Application.Forms
   (d) Application.Forms
21. An asterisk (*) next to a filename in a tab means that you have not made changes since you changed the file. True or False?

22. Which form property cannot be changed in code?
   (a) Height
   (b) BackColor
   (c) (Name)
   (d) ShowInTaskbar

23. A project can have only a single startup form. True or False?

24. Which sizing handle would you use to make a control both taller and wider?
   (a) a corner handle
   (b) the right handle
   (c) the left handle
   (d) the bottom handle

25. Which pair of properties do you use to set a control’s location on a form?
   (a) Top, Left
   (b) Bottom, Right
   (c) X, Y
   (d) PositionX, PositionY

26. By default, a TextBox control allows a user to enter multiple lines of text. True or False?

27. A Label control can wrap text only if the AutoSize property is set to True. True or False?

28. Which Button control property setting prevents a button from accidentally being pushed?
   (a) Activated = False
   (b) Enabled = False
   (c) Deactivated = True
   (d) Disabled = True

29. Which event is the default event for a form?
   (a) FormClosing
   (b) Activated
   (c) Load
   (d) Move

30. In Code view, you must select an object in the left-hand drop-down list before you can select an event. True or False?
31. What is the default event of the Label control?
   (a) Click
   (b) LostFocus
   (c) TextChanged
   (d) Change

32. Which key do you press to cause an application to start up?
   (a) F1
   (b) F3
   (c) F5
   (d) F7

**Applying This Chapter**

1. You are planning to install Visual Studio. Your computer has the following configuration: 800MHz processor, 512MB RAM, 20GB hard disk with 1GB free disk space, and Windows 2000 Professional with SP3. Can you install Visual Studio? If not, what changes must you make?

2. You are building an application that will include several image files. You want to make the image files easy to find. What should you do?

3. What is wrong with the following code?
   
   ```
   frmConfiguration.Text = Configuration Settings
   ```

4. You add a TextBox control to a form. You cannot increase its height. Why not?

5. You add the following code to a form:
   
   ```
   Private Sub txtName_TextChanged _
   (ByVal sender As System.Object, ByVal e As _
   System.EventArgs) Handles txtName.TextChanged
   txtName.ReadOnly = True
   End Sub
   ```

   You press F5 to run the application and type in the txtName field. You can enter only a single character. Why?
YOU TRY IT

Installing Visual Studio
You plan to install Visual Studio to support application development using Visual Basic 2005. You do not plan to develop any applications using Visual C# or Visual C++. You have not yet decided whether you are going to build Windows applications or web applications.

1. Discuss the benefits and drawbacks of installing the Visual C# language component.
2. Discuss the benefits and drawbacks of installing web development components.
3. What can you do if you do not install the web development components and later decide to build a web-based application?

Configuring a Project
You are creating an order entry application for a customer. The customer has provided you with a requirements document and the Excel spreadsheet that is currently used to enter orders.

1. Discuss the benefits of adding these files to your project.
2. How would you ensure that these files were easy to identify?

Creating Forms
You are creating an order entry application for a customer. The application must allow users to view customer details and inventory details as well as enter orders. The application must also allow users to enter configuration settings. It should also display an About dialog box.

1. Identify the forms you need to add to the project.
2. Which form should you designate as the startup form?

Designing a Form
You are creating a form that will be used in a customer service application. The form will be used to enter details about a service call. The form will retrieve information from a database that stores customer information, based on a customer's name. The customer's name must not be modified after the data is retrieved. However, any other information can. The form must show the customer's name, phone number, and address. It must allow a user to enter a description of the problem and the resolution. The user must be able to submit the data to a database and print the data.

1. Sketch the form. Identify which types of controls you should use for each element.
2. Will any of the controls be multiline?
3. Will any of the controls be ReadOnly at design time?

Programming Events
You are creating a form that will be used in a customer service application. The form will be used to enter details about a service call. The form will retrieve information from a database that stores customer information, based on a customer's name. The customer's name must not be modified after the data is retrieved. However, any other information can. The form must show the customer's name, phone number, and address. It must allow a user to enter a description of the problem and the resolution. The phone number field can contain only numeric characters. The user must be able to submit the data to a database and print the data. The user must be prompted when to save data when the form is closed.

1. Identify the events procedures you need to create to provide the needed functionality.