

1 AN INTRODUCTION TO PROGRAMMING AND VB .NET

LEARNING OBJECTIVES

After reading this chapter, you will able to:

1. Understand the importance of information systems in organizations.
2. List and discuss the six computer operations.
3. Discuss the role of computer programs and programming languages in information systems.
4. Understand the concepts of object-oriented programming in Windows and in VB .NET.
5. List and discuss the steps in developing an application in VB .NET.

INFORMATION SYSTEMS IN BUSINESS

Many organizations are finding that in order to survive, they must be able to collect and process data efficiently and make the resulting information on their operations available to their employees. Successful organizations have found that the key to making this information available is having an effective information system that will carry out these operations. An **information system** is *the combination of technology (computers) and people that enables an organization to collect data, store them, and transform them into information*. To understand the concept of an information system fully, you need to understand the difference between data and information. **Data** are raw facts that are collected and stored by the information system. Data can be in the form of numbers, letters of the alphabet, images, sound clips, or even video clips. You are undoubtedly very familiar with many types of data, including names, dates, prices, and credit card numbers. By themselves, data are not very meaningful; however, when data are converted by the information system into **information**, the end result is meaningful. Once again, you are familiar with many forms of information, including written reports, lists, tables, and graphs. Information is what organizational employees use in their work.

To convert or process data into information electronically, software must direct the operations of the computer's operations. **Software** is composed of one or more lists of instructions called **programs**, and the process of creating these lists of instructions is termed **programming**. While computer hardware can be mass-produced on assembly lines like other consumer goods, software must be developed through the logical and creative capabilities of humans. Individuals or groups of individuals working together must develop the instructions that direct the operations of every com-

puter in use today. The same is true whether the instructions are for the computer that controls your car's fuel system, the computer that controls the space shuttle, or the computer that prints the checks for the business at which you work.

Programming in Information Systems

While a great deal of programming work goes on at large software firms like Microsoft or Adobe Systems, much more programming is done at companies that produce non-software goods and services. While you may think that these companies could buy off-the-shelf software like word processors or spreadsheets to run their business, in most cases companies must develop their own software to meet their particular needs. In fact, it has been said that the “software needed to be competitively different is generally not available from off-the-shelf packages” and that “...building...systems for unique [competitive] capability is often the single most important activity for an...organization.”¹ This means that no matter how good off-the-shelf software becomes, there is always going to be demand for programmers to work in businesses and not-for-profit organizations. In fact, the demand for information systems employees is accelerating and the future is very bright for persons trained in this field.

Programming is actually part of a much larger process known as **systems development**. This process involves a large scale effort to either create an entirely new information system or to update (maintain) an existing information system. In either case, systems development involves four primary steps: planning, analysis, design, and implementation. In the planning stage, it is decided what must be done to solve a problem or meet a need—create a new system, update an old one, or even, purchase a system from an outside source. Once it has been decided what must be done, the next step is to analyze the system that will be created. This may involve analyzing an existing system or analyzing the system that must be created. Once the analysis step is completed, the next step is to design the new or updated system. This design must be complete and detailed and leave nothing to chance or guesswork. Once the design is completed, the system can be implemented. It is in the implementation step that programming comes in. Programmers work with the results of the design step to create a series of computer programs that, together, will work as the needed information system. In many cases, the programmers will know little or nothing about the overall problem and must depend completely on the results of the design step. However, without the programming process, the information system will never be built or updated. Visual Basic .NET (VB .NET) and the entire Microsoft .NET framework is aimed at making this process possible.

Given that programming is such an important part of building and maintaining information systems for organizations of all sizes, it is easy to see why individuals interested in working in the field of information systems must have some knowledge of computer programming. This book is written with the purpose of helping you become capable of writing computer programs that will solve business-related problems.

COMPUTER OPERATIONS

Before we start our discussion of creating computer programs, it is useful to understand the six operations that all computers can carry out to process data into information. Understanding these operations will help you when you start writing programs.

1. Martin, James, *Cybercorp: The New Business Revolution*, New York: AMACOM Books, p. 104

These operations are the same regardless of whether we are discussing multi-user mainframe computers that handle large-scale processing, such as preparing the end-of-term grade rolls or processing the university payroll, or small personal computers that are used today by a large proportion of office workers in the United States and other developed countries. The six operations that a computer can perform are:

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

Let's now discuss each of these operations in a little more detail.

Input Data: For a computer to be able to transform data into information, it must first be able to accept input of the data. Data are typically input from a keyboard or mouse, but they can also come from other sources such as a barcode reader like those used at checkout terminals. Input can also come from some type of sensor or from a data file on computer disk. For example, with a word processor, the letters of the alphabet, numbers, and punctuation symbols are the data that are processed by the computer. New documents are created by entering data from the keyboard while existing documents are loaded from your hard drive or floppy disk.

Store data in memory: Once data have been input, they are stored in internal memory. Each memory location holding a piece of data is assigned a name, which is used by the instructions to perform the processing. Since the values in a memory location can change as the process occurs, the memory locations are called **variables**. 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. In the earliest days of computing, the instructions (program) were not stored in memory and had to be entered one at a time to process the data. When the *stored program* concept was developed by John von Neumann, it was a tremendous breakthrough. With a stored program, the instructions can be executed as fast as they can be retrieved from memory to convert the data into usable information.

Perform arithmetic on data: Once the data and instructions have been input and stored, arithmetic operations can be performed on the variables representing the data to process them into information. 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.

You may ask how a word processor or computer game works 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.

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 com-

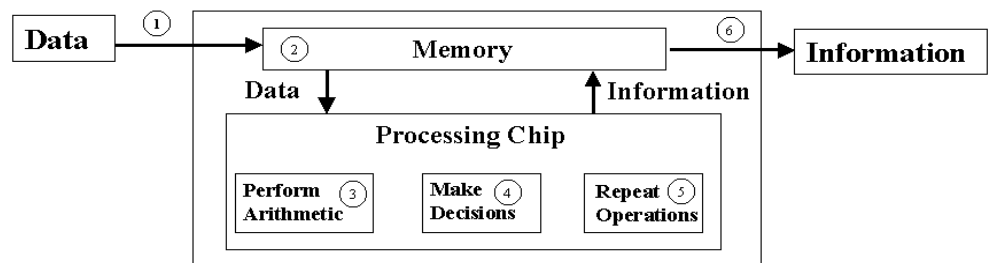
puter is checking each word to determine if it matches a word in the computer's dictionary. Based on the result of this comparison, the word is accepted or flagged for you to consider changing.

Repeat a group of actions any number of times: While you *could* carry out all of the above 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 from 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 above is an example of a repeated action: The program repeatedly checks words until it comes to the end of the document.

Output the results of processing: Once 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: displayed on a monitor, printed on paper, stored on disk, as instructions to a machine, 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 you probably will also print it for distribution to others.

These six operations are depicted in Figure 1-1, where each operation is numbered.

FIGURE 1-1. Six computer operations



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

While it can be quite daunting to create the logic to solve a problem, remember that all computer programs can be created with only three types of logic or, as they are known in programming, **control structures**. The three control structures are sequence, decision, and repetition.

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. Of course, it is up to the programmer to determine the proper sequence order 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 using the six operations of a computer within combinations of these three control structures. Once 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

Once you have developed the logic for solving the 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 VB .NET. These languages use a restricted vocabulary and a very structured syntax that the computer can understand. While a great deal of research is ongoing to create computers that can accept instructions using conversational English, currently no computers meet this criterion. So, until computers like C3-PO and R2D2, popularized in the *Star Wars* movies, are created, we are stuck with using these programming languages.

Within the computer, the data and instructions are represented in the binary number system as a series of zeros and ones. This form of representation is used because the computer's only two electrical states—on and off—correspond to 1 and 0. Using a string of transistors that act as switches, the computer can represent a number, character, or instruction as a series of on–off states. All processing is carried out in the binary number system. For example, the 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**, since 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.

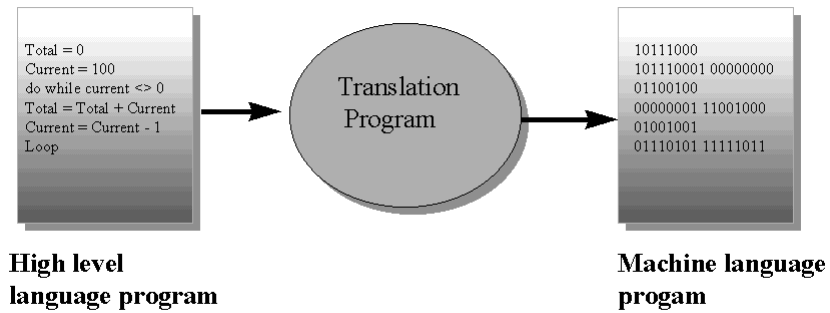
FIGURE 1-2.

Machine language program

Machine Language Command	Explanation
10111000 00000000 00000000	Set Total Value to 0
10111001 00000000 01100100	Set Current Value to 100
00000001 11001000	Add Current value to Total Value
01001001	Subtract 1 from Current Value
01110101 11111011	If Current value is not 0, repeat

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, like VB .NET which are referred to as **high-level languages** because they are close to the level of the human programmer rather than being close 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 interpreter, depending on whether the code was translated as a unit or line by line as shown in Figure 1-3.

FIGURE 1-3. Direct translation process



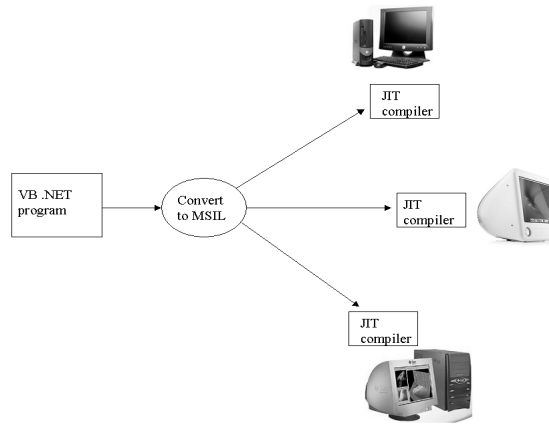
The problem with this approach is that different types of computers have different machine languages so a program would have to be translated differently for an Apple computer than for a Windows computer. To make it possible for the same program to run on all types of machine, the concept of the **just-in-time (JIT) compiler** was developed. With this approach, the high-level program is translated or compiled into an intermediate form that is machine-independent. The two approaches to this use of a just-in-time compiler are Java from Sun Microsystems and the .NET framework from Microsoft of which VB .NET 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 VB .NET program is compiled into MSIL, the just-in-time compiler on any computer can convert it into machine language for that particular machine. This process is shown in Figure 1-4 for MSIL.

Where the Java approach only works 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 VB .NET, C# (pronounced "c-sharp") .NET, and C++ (pronounced "c plus plus") .NET. This means that if you are using one of these language, it can be compiled in MSIL and combined with other programs in the MSIL and then sent to the JIT compiler, which for the .NET framework is called the *Common Language Runtime (CLR)*.

PROGRAMMING IN WINDOWS

As you are probably aware, most personal computers today run some form of the Microsoft Windows operating system such as Windows 95, Windows 98, Windows ME, Windows 2000, or Windows XP. With Windows being the primary operating sys-

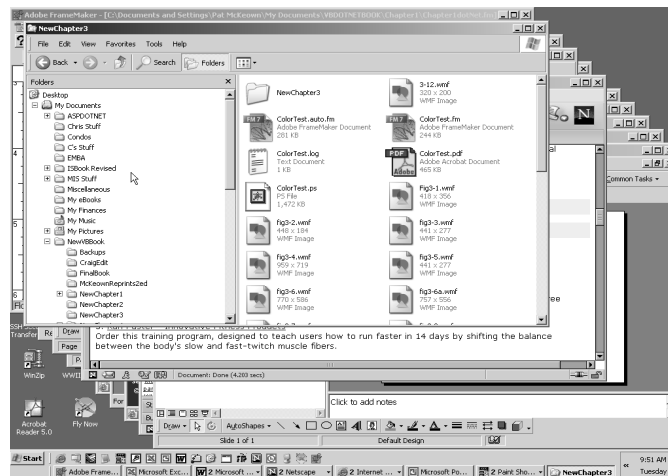
FIGURE 1-4. Use of MSIL and JIT compilers



tem for personal computers, learning to program in the Windows environment has become a critical skill for anybody 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 with its own boundaries. All components run in their own windows. For example, when you use your 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. Similarly, when an error message is displayed, this is done in a window. Finally, the menu bar and all of the icons or buttons on the toolbar across the top or side of your screen are also windows. Figure 1-5 shows a Windows XP screen with several types of windows displayed.

FIGURE 1-5. Windows in Windows XP



As a part of its operations, the Windows operating system is constantly monitoring all of the windows on the screen for signs of activity termed **events**. An event can be a mouse click or double-click, a keypress, or simply 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 set 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. While this may sound complicated, languages like VB .NET make it easier to create Windows-based applications that work with Windows 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 will take place and the order in which those actions will 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 the term or printing payroll checks at the end of the pay period. However, with the move toward widespread use of GUI, the trend is toward using event-driven programming.

The VB .NET Language

As discussed above, VB .NET is a computer language that has been developed to help you create programs that will work with the Windows operating system. It 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 VB .NET 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.

In addition to being event-driven, VB .NET is an **object-oriented (OO) language**; that is, it uses software objects which can respond to events. This is an important improvement over previous versions of Visual Basic which were *almost* object-oriented since they failed to have all of the characteristics of a true OO language. What distinguishes object oriented 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, then the code may not work with the new data. With object-oriented programming, the combination of code and data avoids this problem. For example, instead of writing code to deal with customers and then using this code with different customer data for each customer, we combine the code and data 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.

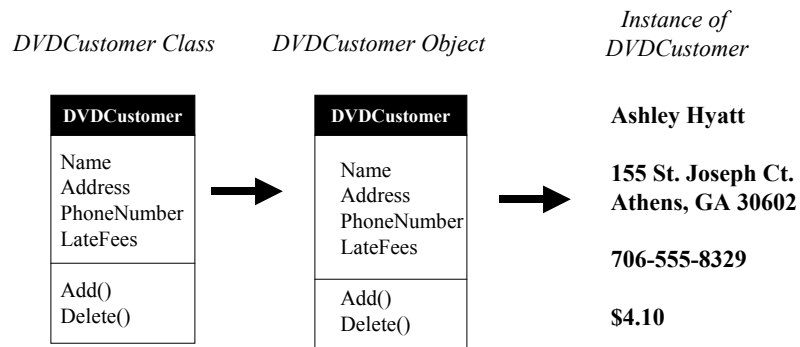
A primer to object-oriented programming

Chapter 8 of this book will deal with objects and object-oriented programming in detail, so we will just introduce you to some of the concepts of OO. First, consider the fact that each of the windows discussed above as a part of the Windows operating system is an object, as are a wide variety of other shapes, including buttons, click boxes, menus, and so on. There are also many objects in Windows that are unseen since they

are pure computer code, but have the same characteristics as visual objects. The beauty of VB .NET is that, unlike many other OO languages, you do not have to know how to create objects to use them. VB .NET automatically creates for you, the programmer, new instances of a many objects from a wide variety of built-in templates.

To understand Object-oriented programming, we 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 a class is to think of it as the *cookie cutter* and the actual object as the resulting *cookie*.² All of 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 concepts: properties and methods. **Properties** of objects are simply the attributes associated with the object such as their 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 are instances of a class called *DVDCustomer* which will have the properties and methods shown in Figure 1-6. Note that the *DVDCustomer* class has *Name*, *Address*, *PhoneNumber*, and *LateFees*. The class also has the *Add* and *Delete* methods to add and delete customers. Note that we have also created an object named *DVDCustomer* for a DVD rental store and an instance of this object for *Ashley Hyatt* that contains properties particular to her.

FIGURE 1-6.
Customer class,
object, and instance



Three key characteristics of OO programming are encapsulation, inheritance, and polymorphism. **Encapsulation** refers to a key concept: It should never be possible to work with variables in an object directly; they 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 change the values of the *DVDCustomer* object without going through the properties of the object; you can not get into the object except through the properties.

The second key concept in object-oriented programming, **inheritance**, refers to the capability to create *child* classes that descend from a *parent* class. This capability makes it easier to build a new child classes by having them inherit properties and methods from a parent class. For example, the class *DVDCustomer* inherits the properties

2. Cornell, Gary, *Visual Basic 5 from the Ground Up* (Berkeley: Osborne/McGraw-Hill, 1997), p. 376.

and methods from a more general *Customer* class, which itself inherits properties and methods from the even more general *Person* class.

Finally, **polymorphism** is related to inheritance in that a child class can inherit all of 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* from the *Customer* class and then adds the *LateFees* property that is particular to the *DVDCustomer*.

As another example, of objects, consider the an object with which you are familiar might be 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 we apply the KICK event to the soccer ball, then, depending on its 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 VB .NET, but the programmer must write the instructions to tell the object how to respond to an event.

Objects are combined with properties or methods by a period or dot, and objects are combined with events by an underline. Continuing the soccer ball example, we might have a property definition through the following statement:

Ball.Color = White

which defines the color of the ball.

Similarly, the roll method of the soccer ball is referenced by the dot property as shown below:

Ball.Roll

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

Ball_Kick

Working with VB .NET 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 or, as it is called in VB .NET, the *solution*. These instructions are referred to as the code for the program. The code for VB .NET is written in a much-updated form of one of the oldest computer languages around—Basic, which was first used in 1960. The version of Basic used in VB .NET has been improved in many ways, but it retains one of the key advantages of the original language compared to other languages: It is very easy to use and understand.

PROGRAMMING IN VB .NET

Creating an application using an OO programming language such as VB .NET 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 up 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 of the necessary messages between objects in Windows are handled by VB .NET, thereby significantly reducing the work you must do to create an application.

The manner in which you create a VB .NET project is also different from traditional programming. Instead of having to create an entire program before testing any part of it, with VB .NET you can use **interactive development** to create an object, write the code for it, and test it before going onto other objects. For example, assume a store named Vintage DVDs that rents only “old” movies on DVD has asked you to create a VB .NET project that includes calculating taxes on a DVD rental and sums



the taxes and price to compute the amount due. With VB .NET, 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.

While creating an application in VB .NET 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. These steps are:

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

It should be noted that it may be necessary to repeat or iterate through these steps to arrive at an acceptable final solution to the original problem.

In the next sections, we will 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. We will return to this example and expand it in subsequent chapters.

Step One: Define Problem

Before we can hope to develop any computer application, it is absolutely necessary to clearly define our objective, that is, the problem to be solved. Only then can we 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 maybe 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.

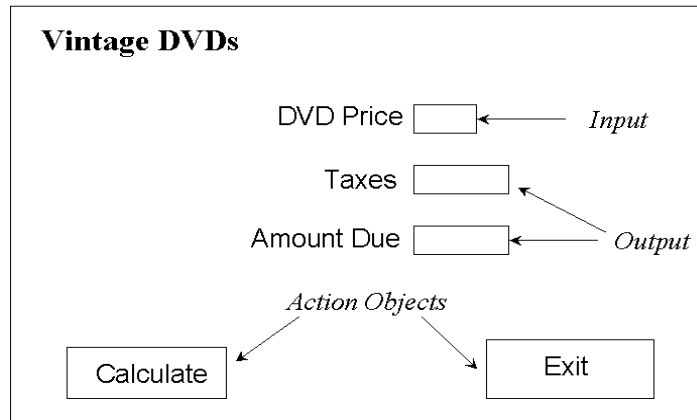
The problem identification step should include identification of the data to be *input* to the program and the desired results to be *output* from the program. Often these two items will be specified by a person or an agency other than the programmer. Much grief can be avoided if these input and output 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, or a program that correctly solves the wrong problem, or a combination of both! Therefore the programmer *must* spend as much time as is necessary to truly identify and understand the problem.

Because VB .NET 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 , you will 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 denoted as in Figure 1-7.



FIGURE 1-7. Sketch of interface for Vintage DVDs

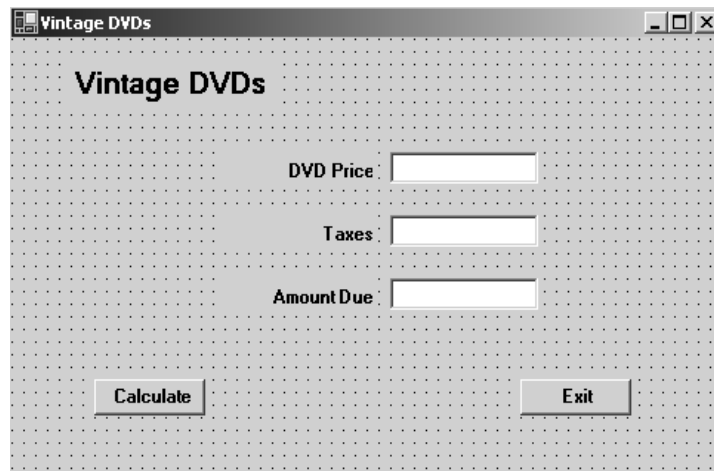


Step Two: Create Interface



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

FIGURE 1-8. Interface for Vintage DVDs



Step Three: Develop Logic for Action Objects

Once the problem has been clearly identified and the interface created, the next step is to develop the logic for the action objects in the interface. This is the step in the development process where 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. Once 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. Once you have developed an IPO Table and the pseudocode for each object, it is a very easy step to write a **procedure** in VB .NET that will carry out the necessary processing.

IPO Table

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 output: the amount of the sale. As mentioned earlier, in many cases the program designer will have no control over the input and output. They will be 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 the overall application, the overall design. Once 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 calculation button. IPO tables are needed for all objects that involve input, output, and processing. We won't do one for the Exit button since it simply terminates the project.

FIGURE 1-9. IPO Table for Calculate button

Input	Processing	Output
Video price	Taxes = 0.07 x Price Amount due = Price + Taxes	Taxes Amount due

Pseudocode

Once you have developed the IPO tables 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, since 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; it should be a personalized method 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 circumstances. 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. As an example of pseudocode, assume a program is needed to compare two values, Salary and Commission, and to output the smaller of the two. The pseudocode for this example is shown below:

```

Begin procedure
  Input Salary and Commission
  If Salary < Commission then
    Output Salary
  
```

```

Else
    Output Commission
End Decision
End procedure

```

In this pseudocode, it is easy to follow the procedure. Note that parts of it are indented to make it easier to follow the logic. The important point to remember about pseudocode is that it expresses the logic for the action object to the programmer in the same way that a computer language expresses it to the computer. In this way, pseudocode is like a personalized programming language.



Now let's write a pseudocode procedure for the Vintage DVDs Calculate object. Note that the pseudocode program matches the IPO Table shown in Figure 1-9.

```

Begin procedure
    Input DVD Price
    Taxes = 0.07 x DVD Price
    Amount Due = DVD Price + Taxes
    Output Taxes and Amount Due
End procedure

```

While we have only one object in our small example for which an IPO Table and pseudocode are needed, in most situations you will have numerous objects for which you will need to develop the logic using these tools.

Step Four: Write and Test Code for Action Objects

Once you have created the VB .NET interface and developed the logic for the action objects using IPO Tables and pseudocode, you must write procedures in VB .NET for each action object. This code should provide instructions to the computer to carry out one or more of the six operations listed earlier—that is, input data, store data in internal memory, perform arithmetic on data, compare two values and select one of two alternative actions, repeat a group of actions any number of times, and output the results of processing. While creating the interface is important, writing the code is the essence of developing an application.

Since you have not yet been introduced to the rules for writing code in VB .NET for the various objects, we will defer a full discussion of this step until Chapter 3 and beyond. However, you should be able to see the similarity between the VB .NET event procedure displayed in VB Code Box 1-1 and the pseudocode version shown earlier. The differences are due to the way VB .NET handles input and output. Input is from the Text property of the first textbox, named txtDVDPrice. Output goes to the Text property of the two textboxes named txtTaxes and txtAmountDue. There are also statements that begin with the word Dim, to declare the variables, and comment statements that begin with an apostrophe (').

Once you have written the code for an action object, the second part of this step is to test that object and correct any errors; don't wait until the entire project is completed. Use the interactive capabilities of VB .NET to test the code of each and every object as it is written. This process is referred to as **debugging**—trying to remove all of the errors or **“bugs.”**

Because VB .NET automatically checks each line of the code of an object for syntax or vocabulary errors, the debugging process is much easier than in other languages. However, even if all the syntax and vocabulary are correct, the code for an object still may be incorrect—either in the manner in which it carries out the logic or in the logic

CODE BOX 1-1. VB .NET computation of Taxes and Amount Due	<pre> Private Sub cmdCalc_Click(ByVal eventSender As System.Object, ByVal eventArgs As System.EventArgs) Handles cmdCalc.Click Const sngTaxRate As Single = 0.07 'Use local tax rate Dim decPrice as Decimal, decAmountDue As Decimal Dim decTaxes As Decimal decPrice = CDec(txtDVDPrice.Text) decTaxes = decPrice * sngTaxRate 'Compute taxes decAmountDue = decPrice + decTaxes 'Compute amount due txtTaxes.Text = CStr(decTaxes) txtAmountDue.Text = CStr(decAmountDue) txtDVDPrice.Text = Cstr(decPrice) End Sub </pre>
---	---

itself. The best way to find and correct such errors is to use **test data** for which the results are known in advance. If the results for the object do not agree with the results from the hand calculations, an error exists, either in the logic or in the hand calculations. After the hand calculations have been verified, the logic must be checked. For example, if the results of the Calculate button came out different from what was expected, then we would need to look for a problem in the data or the logic.

In the case of the Calculate button, we want to determine if the code shown in VB Code Box 1-1 will actually compute and output to the textboxes the *correct* taxes and amount due for the DVD price entered in the first textbox. Figure 1-10 shows the result of clicking the Calculate button for a DVD with a price of \$1.99. Note that the results, while correct, are not *exactly* what you might expect. Instead of rounded values of \$.14 for the taxes and \$2.13 for the amount due, the answers are the exact values of \$.1393 and \$2.1293. This is because we have not *formatted* the answers as dollar and cents. This will be done when we revisit this problem in Chapter 3.

While the answers for this set of test data are correct, this does not mean it will work for all test data. Testing requires that a wide variety of test data be used to assure that the code for the object works under all circumstances.

Since the Calculate button appears to work, we can now write the code for the Exit button, which consists of one instruction: End. If this command also works, then we are ready to move on to the next step in the application development process: testing the overall project.

FIGURE 1-10. Testin
g the Calculate
button

The screenshot shows a Windows-style application window titled "Vintage DVDs". The window has a title bar with standard minimize, maximize, and close buttons. The main content area is light gray and contains the following elements:

- A label "Vintage DVDs" at the top center.
- Three text boxes arranged vertically:
 - The first is labeled "DVD Price" and contains the text "1.99".
 - The second is labeled "Taxes" and contains the text ".1393".
 - The third is labeled "Amount Due" and contains the text "2.1293".
- At the bottom of the window, there are two buttons: "Calculate" on the left and "Exit" on the right.

Step Five: Test Overall Project

Once 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 Six: Document Your Project in Writing

An important part of writing any computer software is the documentation of the software. **Documentation** can be defined as *the written descriptions of the software that aid users and other programmers*. It includes both internal descriptions of the code instructions and external descriptions and instructions. 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. In VB Code Box 1-1, the statements beginning with an apostrophe (') are examples of internal documentation. 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 will be very difficult. In this text, because we will be explaining the code in detail, we do not include the level of internal documentation that *should* be found in the projects you create both here and in your work.



Tip: While internal documentation is shown as Step 6 of the program development process, it is more easily carried out if it is done *during* the development process rather than at the end.

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.

Final Comments on the Programming Process

Creating applications in a language like VB .NET is easy and fun. However, there is one caveat to this statement: You still must be able to develop the logic for the action objects and write the code to make them respond appropriately to events. In this text, we will spend the first five chapters concentrating on two things: showing you how to create a fairly simple interface and discussing the key elements involved in writing

code. While creating the interface can be very interesting, you should not lose sight of the overall goal, which is to produce applications that respond to events in an appropriate manner. The only way to make this happen is to be able to write code that works!



SUMMARY

TIP: No matter how good your interface looks, if the code does not work, the application will fail. Helping you create code that works is the purpose of this book.

At the beginning of the chapter, we said you would be able to do a number of things after reading it. Let's review those things here:

1. **Understand the importance of information systems in organizations.** Developing applications for information systems in organizations is the key job of information technology departments. While off-the-shelf software is of great use to individuals and organizations, in many cases organizations are finding that they must develop their own software to be competitive in today's world.
2. **List and discuss the six computer operations.** The six operations that all computers can carry out are:
 1. *Input data* that is then converted into information and output. Input can consist of numbers, text, or even instructions.
 2. *Store data in internal memory* for future processing. The data must be stored here so the processing chip can find it and carry out necessary transformations.
 3. *Perform arithmetic on data* to convert it into information. While it may appear that computers can work with text and graphics, in actuality, all they can do is to add and compare. However, by sophisticated means, it is possible to manipulate text and graphics by these two operations.
 4. *Compare two values and select one of two alternative actions* to make a decision. Without decisions, computing would be very boring since only one sequence of instructions could be carried out. With decisions made by comparing two values, a wide variety of sequences can be followed.
 5. *Repeat a group of actions any number of times* to carry out the most sophisticated processing operations. The ability to repeat is what really sets a computer apart from a handheld calculator and what enables it to carry out many of the more complex operations we generally associate with computing.
 6. *Output the results of processing* so that
3. **Discuss the role of computer programs and programming in information systems.** All information systems are a combination of people, hardware, and software and their purpose is to collect data, store them, and transform them into information. To convert or process data into information electronically, software must direct the operations of the computer's operations. Software is composed of one or more lists of instructions called programs, and the process of creating these lists of instructions is termed programming. Computer languages can be classified as high-level languages and machine languages. A high-level language uses English-like commands while machine language uses binary instructions that the computer can understand. A high-level language must be translated into machine language before the computer can use it. Most older languages used a direct translation to compile high-level languages into machine language, but the .NET languages use a

just-in-time compiler. In this process, the high-level language like VB .NET is converted into an intermediate form (MSIL) and then converted into machine language by the JIT compiler on each machine.

4. **Understand the concepts of object-oriented programming in Windows and in VB .NET.** Three key concepts in programming in Windows are windows, events, and messages. On the screen, windows are rectangular regions with boundaries. Signs of activity in the windows are events and windows send messages when an event occurs. Programming in Windows is known as event-driven programming as compared to older forms of programming which are procedural. VB .NET is an event-driven language that is also object-oriented since it uses software objects to respond to events. All objects have attributes which are known as properties and predefined activities known as methods which they can carry out. Programmers can write instructions to tell an object how to respond to an event.
5. **List and discuss the steps in developing an VB .NET application.** Next, we discussed the steps in the VB .NET programming process, which are as follows:
 1. Define problem
 2. Create interface
 3. Develop logic for action objects
 4. Write and test code for action objects
 5. Test overall project
 6. Document project in writing

Defining the problem involves determining the data to be *input* to the program and the desired results to be *output* from the program. Because VB .NET 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. The sketch should include the input and output objects and the objects for which code is needed to respond to events, the so-called action objects.

Creating the interface in VB .NET involves selecting visual objects from those available and place them on the form. This process should follow the sketch done earlier.

Developing logic for action objects involves thinking what each action object must do in response to an event. Two useful tools for designing programming applications are IPO (Input/Processing/Output) tables and pseudocode. IPO 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. Once 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 English rather than in a computer language. Once you have developed an IPO Table and the pseudocode for each object, it is a very easy step to write a procedure in VB .NET that will carry out the necessary processing.

The next step is to write and test the computer instructions in VB .NET that will carry out the logic for each action object. This is the most important step since the computer code actually implements the necessary logic to solve the problem.

Once the code for each action object has been tested 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.

The last step in writing any computer software is the documentation. Documentation can be defined as the written descriptions of the software that aid users and other programmers. It includes both internal descriptions of the code instructions and external descriptions and instructions. 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.

KEY TERMS

action objects	information	procedure
algorithm	information system	programming
bugs	interactive development	programming languages
code	interpretation	programs
compilation	IPO (Input/Processing/Output)	project
control structures	Tables	properties
data	logic	pseudocode
debugging	machine language	repetition control structure
decision control structure	message	sequence control structure
documentation	methods	software
event-driven programming	object-oriented event-driven	systems development
events	(OOED) language	test data
graphical user interface (GUI)	object-oriented language	variables
high-level language	procedural programming	window

EXERCISES

- Every month you collect your loose change in a jar on your dresser. At the end of the month, you sort and roll the coins and deposit them in your savings account. Describe what you would do using steps that follow the three basic control structures: sequence, decision and repetition. By following your steps, you should be able to handle one year worth of coins.
- Develop a list of properties, methods, and methods for each of the following common objects below.
 - a pet cat
 - an automobile
 - a video tape
 - a document created using word processing software
 - a list of customers and their contact information
- For the following scenarios, describe the inputs, outputs and processing steps using an IPO table.
 - You are balancing your checkbook. You have a stack of items that need to be added to the checkbook record including: deposit slips, ATM withdrawal receipts, and copies of checks used. You need to add the items and keep a running total of the balance for each item.
 - You are planning your schedule for the next school term. Assume that you will successfully complete your current courses.

- c. You have a personal web page on which you post a news page about your band/sports team/debate club/etc. You wish to automate the creation of your news page.
4. An algorithm is a step-by-step logical procedure for accomplishing a task or solving a problem. Write an algorithm that lists the steps that you would take for the following tasks or problems.
- a. You are going from your home to your first class of the day.
 - b. You are searching on the Internet for a course research topic.
 - c. You are attempting to best a friend while playing a simple game (tic-tac-toe, hangman, etc.).
5. Write a brief problem description for the programs that correspond to the following sets of IPO and pseudocode. What control structures are used in the logic for each set?
- a.

Input	Processes	Output
exchange rate amount in dollars	read exchange rate get amount in dollars calculate amount in foreign currency display amount in foreign currency	amount in foreign currency

Begin Procedure

Read exchange rate

Get amount in US dollars

Amount in foreign currency = exchange rate * amount in US dollars

Display amount in foreign currency

End Procedure

b.

Input	Processes	Output
tax rate item 1 price item 2 price item 3 price : item <i>k</i> price	read tax rate get prices for all items purchased calculate subtotal calculate sales tax calculate total price print subtotal, sales tax and total price	subtotal sales tax total price

Begin Procedure

Read tax rate

```

Repeat
  Get item price
Until all item prices obtained
Calculate subtotal = sum of all prices
Calculate sales tax = subtotal * tax rate
Calculate total price = subtotal + sales tax
Print subtotal, sales tax and total price
End Procedure

```

c.

Input	Processes	Output
grade average 1	get each grade average	letter grade 1
grade average 2	determine letter grade	letter grade 2
grade average 3		letter grade 3
:		:
grade average k		letter grade k

```

Begin Procedure
Repeat
  Get next grade average
  If grade average >= 90
    next letter grade = A
  Elself grade average >= 80
    next letter grade = B
  Elself grade average >= 70
    next letter grade = C
  Elself grade average >= 60
    next letter grade = D
  Else
    next letter grade = F
  End If
  Write next letter grade
Until all grades are assigned
End Procedure

```

PROJECTS

1. Assume that a student takes three quizzes and the score for each quiz is input. The output should be the average score on the three quizzes. Sketch the interface for this problem if textboxes will be used for input and output and a Compute button will calculate the average score. Also, create an IPO Table and pseudocode for the Compute button.
2. Chris Patrick works for the Shrub and Turf Lawn Care Company. He is paid a 10 percent commission on the value of lawn care contracts that he sells. Assume that the input includes the number of sales and the price charged for such contracts (assume it is the same for all contracts.) Output should include the total value of the sales and Chris's commission on the sales. Sketch the interface for this problem if textboxes are used for input and output. Assume that two buttons are used: one for computing total value of the sales and one for computing Chris's commission. For each button, develop

an IPO Table and the pseudocode procedure.

3. Acme, Inc., leases automobiles for its salespeople and wishes to create an application that will determine the gas mileage for each type of automobile. Input should include the make of the automobile, the beginning odometer reading, the ending odometer reading, and the gallons of gasoline consumed. Output should include the miles per gallon for the car being tested. Sketch the interface for this problem if textboxes are used for input and output. Assume that a Calculate button is used for computing the gas mileage. Develop an IPO Table and the pseudocode procedure for this Calculate button. [Note: Gas mileage = (Ending odometer reading – Beginning odometer reading)/Gallons used.]

4. Smith and Jones, Inc., wishes to determine the breakeven production volume for a new product. Breakeven volume is defined as the number of units that must be produced and sold for the total cost of production to equal the total revenue. The formula used to calculate the breakeven point is (Fixed cost of production)/(Selling price per unit – Variable cost per unit). The company also wants to know the Total revenue and Total cost values at the breakeven point where:

$$\text{Total revenue} = \text{Selling price} \times \text{Number produced}$$

$$\text{Total cost} = \text{Fixed cost} + (\text{Variable cost} \times \text{Number produced})$$

Input for this problem includes the Fixed cost of production, the Unit price, and the Unit cost for the new product. Output should include the Breakeven volume as well as the Total cost and Total revenue at the breakeven point. Sketch the interface for this problem if textboxes are used for input and output. Assume that one button is used for calculation of Breakeven volume and Total revenue/Total cost at the Breakeven volume. Develop an IPO Table and the pseudocode procedure for this button.

5. Cover-Your-Wall, Inc., specializes in selling wallpaper to "do-it-yourselfers." The company would like a computer program to determine the number of rolls needed to cover a room. This calculation depends on the area to be covered. This value is computed for a rectangular room with an eight-foot ceiling using the following formula:

$$\text{Room area} = (2 \times \text{length} \times 8) + (2 \times \text{width} \times 8) - (\text{window area}) - (\text{door area})$$

Then the number of rolls needed is found by:

$$\text{number of rolls} = (\text{room area})/(\text{roll area})$$

Design a project that will enable customers to enter data about their room and the type of wallpaper they are using and determine the number of rolls needed. Assume that input includes length and width of the room in feet, window area and door area for the room in square feet, and the roll area in square feet for the type of wallpaper being considered. Output should include the room area and the number of rolls needed to cover the room. Sketch the interface for this problem if textboxes are used for input and output. Assume that one button is for computing the room area and for computing the number of rolls needed. Develop IPO Tables and the pseudocode procedures for these buttons.

6. The loan officers of LowHomeLoans.com wish to provide a simple tool for computing the maximum loan payment that a borrower can expect to afford. They want to incorporate two "rules of thumb:"

1) The maximum monthly payment should not exceed 28% of the borrowers gross monthly income and

2) The maximum monthly payment should not exceed 36% of the borrowers gross monthly income minus monthly debt payments.

Here the monthly payment will include principle, interest, taxes and insurance. Assume that input includes the gross monthly income and the monthly debt payments. Output should include the maximum monthly payment based on gross income alone and maximum monthly payment based on gross income minus monthly debt. Sketch the interface for this problem using only textboxes, labels and command buttons. Develop an IPO table and the pseudocode procedures for these buttons.

7. Gregor Samsa was having difficulties getting up this morning. He had spent much of the night thinking about a project at the medium-sized exterminating company that he owns and operates. He has been losing money lately due to problems with the company's billing system and he feels that it should be upgraded or replaced. After looking around, he has found several alternative systems that might work for him. One criteria that he will use to choose between the alternative systems will be the net present value (NPV). As he attempts to roll out of bed, Gregor decides to have one of his IS people build him a simple NPV calculator. The NPV for a project may be calculated using:

$$NPV = -I_0 + (F_1/(1+k)) + (F_2/(1+k)^2) + \dots + (F_n/(1+k)^n) + (S_n/(1+k)^n)$$

Here, I_0 represents the initial investment; F_i represents the net cash flow in period i ; S_n represents the salvage value of the project at the end of its useful life (period n); k represents the minimum required annual rate of return; and n represents the lifetime of the project in years. Develop the IPO, sketch the interface, and write pseudocode for the action objects in your interface. You may assume that the useful life will be a maximum of 10 years. What changes would you need to make if this assumption is relaxed, that is, the program should work for any amount of years?

PROJECT: JOE'S TAX ASSISTANCE

"No! I'm sorry, Mrs. Twipple. You can't claim your cats as dependents."

"But I look after them all the time. I feed them, clean up after them, take them for walks,..."

"That's just not the way the tax laws work. Anyway, I think I have enough information to complete your return. Why don't you come back next week at the same time and I'll have something worked up for you." Joe gave a sigh of relief as Mrs. Twipple headed for the door.

Joe Jackson retired over a year ago after 30 years as an auditor for a government agency. Since retirement Joe had plenty of time for hobbies, visiting with his granddaughter Zoocy, and volunteering for various programs. Most recently, he had been spending one day a week with the Internal Revenue Service's Volunteer Income Tax Assistance (VITA) program, in which volunteers like Joe offer free tax assistance to people in the community, many of whom do not have the means or the education to complete their own returns.

To stretch his legs, Joe walked out into the hallway and over to the water fountain. The day was nearing an end, but he noticed at least three parties still in line for tax assistance. "It shouldn't take so long to help all these people," Joe mused. "There has got to be a way to speed up the process." He decided to put off thinking about this for the moment and asked the next couple to step into the office.

Later, on the drive home, Joe began to think of ways to improve his efficiency. The main problem, he thought, was how to handle all of the information obtained from each VITA client. At times, the amount of paper generated was enormous.

Finally a possible solution occurred to Joe: “Maybe Zooey can help me come up with something.”

His granddaughter, no doubt with some financial assistance from her parents, had presented Joe with a laptop at Christmas. “It’s time you joined the information age, Grandpa, and I’m going to help you do it,” she had explained. Since that time, she had shown Joe how to use various software programs, how to get “on-line” and “surf,” and even a thing or two about programming.

Joe came to a decision: “Yep, tomorrow I’ll explain to Zooey what I want to do and we can get to work programming. Maybe we can use that VB .NET thing-a-ma-bob that she was talking about.”

Questions

1. Think about the information needs that Joe would come across in his work with tax returns.
2. Would VB .NET be a useful tool in helping Joe handle these needs?
3. Develop an opening screen for Joe’s VITA program.

This material is protected by copyright and may not be downloaded, reproduced, stored in a retrieval system, modified, made available on a network, used to create derivative works, or transmitted in any form or by any means, except (i) in the United States, as permitted under Section 107 or 108 of the 1976 United States Copyright Act, or internationally, as permitted by other applicable national copyright laws, or (ii) as expressly authorized on this Web site, or (iii) with the prior written permission of Wiley