NEAR EAST UNIVERSITY

Faculty of Engineering

Department of Computer Engineering

STOCK CONTROL MANAGAMENT

Graduation Project
COM-400

Student: Menderes BOZKURT

Supervisor: Ümit İLHAN

Nicossia - 2004

TABLE OF CONTENTS

ACKNOWLDGMENTS

First of all I would like to thank Mr. UMIT ILHAN for his endless ıd untiring support and help and his persistence, in the course of the preparation of this oject.

Under his guidance, I have overcome many difficulties that I face during the various stages of the preparation of this project.

Finally, I would like to thank my family, especially my father who na ıe is Mr HASAN BOZKURT and my brothers who name is Mr MURAT BOZKURT MUSA BOZKURT , IBRAHIM BOZKURT and ALİI BOZKURT , .Their love and uidance saw me through doubtul times. Their never-ending belief in me and their encouragement has been a crucial and a very strong pillar that has held me t ;ether

ABSTRACT

As the information age has effected every aspect of our life, t· need for computerizing many information systems has raised.

Once of the important branches that are effected by information revc ıtion is the computer programming languages.

This project is concerned about using compueter program ı Pharmacy management system . It is written using Visual Basic 6.0 programming l: guage and used Microsoft Access Database language for databases. Visual Basic is on of the best and easy programming languages.

This project is accomlete Pharmacy management program, tha covers all services needed in most Pharmacy, such as computer related informati(,madicine, goods and many other Pharmacy management related services.

Before coming to this point, this project has gone through some impc tant steps;

- First one was the requirements definition for which I h d to go to some Pharmacy and study their systems.
- The second steps were designing the system and sof\ ıre that is intended to serve an integrated Pharmacy management sys ı m.
- The final steps was the implementation of the design on t : computer using Visual Basic Language.

INTRODUCTION

Visual Basic is a Microsoft Windows programming Language.Visu Basic programs are created in an Integrated Development Environment (IDE) . T e IDE allows the programmer to create , run and debug Visual Basic programs conv iently. IDEs allow a programmer to create working programs in a fraction of the tin that it would normally take to code programs without using IDEs. The process of apidly creating an application is typically referred to as Rapid Application Deve pment (RAD). Visual Basic is the world's most widely used RAD language.

Visual Basic is derived from the BASIC programming language. Visu Basic is a distinctly different language providing powerfull features such as graphi al user interfaces, even handling, access to the Win32 API, object-oriented featur , error handling, structured programming, and much more.

The Visual Basic IDE allows Windows programs to be created wit ut the need for the programmer to be a Windows programming export.

Microsoft provides several version of Visual Basic, namely the l arning Edition , the Professional Edition and the Enterprice Edition. The Learning Edition provides fundemantal programming capabilities than the Learning Edition an is the choice of many programmers to write Visual Basic applications. The E erprice Edition is used for developing large-scale computing systems that meet the eds of substandial organizations.

Visual Basic is an interpreted language. However , the professio al and Enterprice Edition allows Visual Basic code to be compiled to native code.

Visual Basic evolved from BASIC(Beginner's All purpose S nbolic Instruction Code). Basic was developed in the mid 1960's by Professor John Kemeny and Thomas Kurtz of Darthmouth College as a language for writing simple programs. BASIC's primary purpose was to help people learn how to progran

The widespread use of BASIC with various types of computers (so etimes called hardware platforms) led to many enhancements to the language. W ith the development of the Microsoft windows graphical user interface (GUI) in e late

1980s and the early 1990s, the natural evolution of BASIC was Visual Basi which was created by Microsoft Corporation in 1991.

Until Visual Basic appeared, develoing Microsoft Windov -based applications was a diffucult and cumbersome process. Visual Basic greatly si plifies Windows application development. Since 1991 six versions have been releas l, with the latest-Visual Basic 6-appearing in september 1998.

After a brief explanation about the Visual Basic 6.0 and the de loping layers, I hope that you will find the necessary information that you need about the Visual Basic even if you are a text based programmer.

CHAPTER1

Visual Basic Program

1.1.VB Advantages

So what makes VB a great programming language? The answer is simply that B provides more of the actual code for a programmer than any other non-visual programming language.

If you've ever programmed in the older BASIC or other command line progra ning language, then you'll remember that the programmer had to write the code for e entire user interface. Todays windows, buttons, lists, and other application fea res such as menus were not built-in to the BASIC programming language. Progra mers had to create the code for these features on their own!

As much as 80% of a programmer's time was spent writing code to create the er interface to his applications (the visual interface). To eliminate this huge drain n a programmer's time, Microsoft has provided Visual Basic with the built-in c ability to create the user interface using nothing more than a mouse!

This built-in interface creation capability has had the further benefit of stand dizing on the user interface to Windows applications. Today, users can move fi m one Windows program to another and see the same basic interface tools to wor with - allowing them to concentrate solely on the unique capabilities of the applicati .

The bottom line is that you can create an entire application shell (the user i erface) very quickly and then spend most of your time working on the feature which differentiate you application from its competition.

1.2.Very First Visual Basic Program

Visual Basic lets you build a complete and functional Windows applic on by dropping a bunch of controls on a form and writing some code that execut when something happens to those controls or to the form itself. For instance, you c write code that executes when a form loads or unloads or when the user resizes it. L ewise, you can write code that executes when the user clicks on a control or types w ile the control has the input focus.

This programming paradigm is also known as *event-driven programming* ecause your application is made up of several event procedures executed in an ord that's dependent on what happens at run time. The order of execution can't, in ge ral, be foreseen when the program is under construction. This programming model ntrasts with the procedural approach, which was dominant in the old days.

This section offers a quick review of the event-driven model and uses a ample application as a context for introducing Visual Basic's intrinsic controls, w h their properties, methods, and events. This sample application, a very simple one, ueries the user for the lengths of the two sides of a rectangle, evaluates its perim er and area, and displays the results to the user. Like all lengthy code examples and p grams illustrated in this book, this application is included on the companion CD.

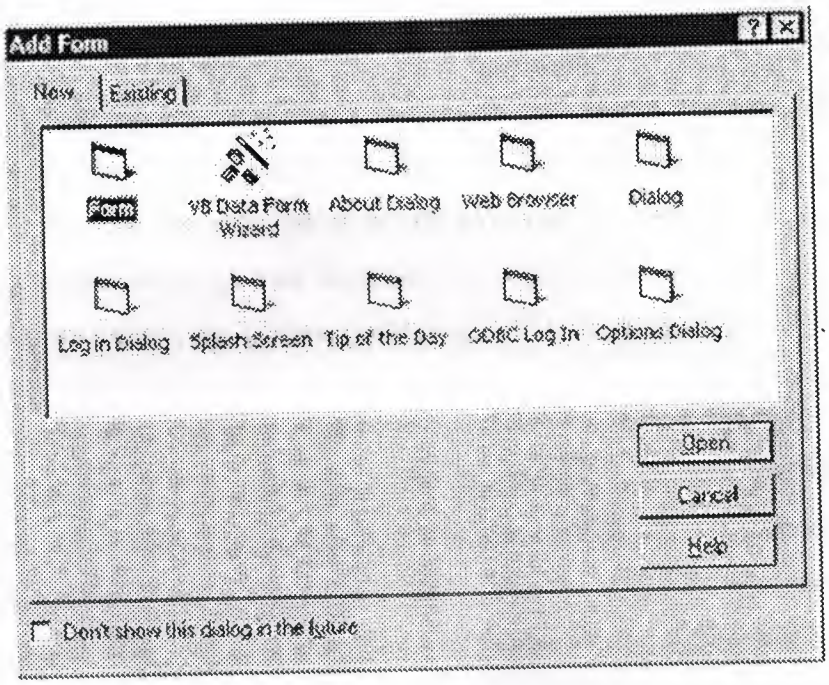
1.3.The Form Object

After this long introductory description of properties, methods, and events at are common to most Visual Basic objects, it's time to see the particular features all of them individually. The most important visible object is undoubtedly the Form object because you can't display any control without a parent Form. Conversely, u can write some moderately useful applications using only forms that have no co ols on them. In this section, I'll show a number of examples that are centered o forms' singular features.

You create a new form at design time using the Add Form command from the Project menu or by clicking on the corresponding icon on the standard toolbar. u can create forms from scratch, or you can take advantage of the many form te plates provided by Visual Basic 6. If you don't see the dialog box shown in Fig e 2-7, invoke the Options command from the Tools menu, click the Environment b, and select the topmost check box on the right.

Feel free to create new form templates when you need them. A form templat doesn't necessarily have to be a complex form with many controls on it. Even an em y form with a group of properties carefully set can save you some precious ti e. For

example, see the Dialog Form template provided by Visual Basic. To produ e your custom form templates, you just have to create a form, add any necessary cont ols and code, and then save it in the \Template\Forms directory. (The complete path o Visual Basic's template directory can be read and modified in the Environment ta of the Options dialog box.)



1.4.Adding Controls to a Form

We're ready to get practical. Launch the Visual Basic IDE, and select a Stand d EXE project. You should have a blank form near the center of the work are More accurately, you have a *form designer*, which you use to define the appearan of the main window of your application. You can also create other forms, if you ne them, and you can create other objects as well, using different designers (the Use ontrol and UserDocument designers, for example). Other chapters of this book are evoted to such designers.

One of the greatest strengths of the Visual Basic language is that programm rs can design an application and then test it without leaving the environment. But yo should be aware that designing and testing a program are two completely different t ks. At *design time,* you create your forms and other visible objects, set their proper s, and write code in their event procedures. Conversely, at *run time* you monitor th effects of your programming efforts: What you see on your screen is, more or less, w t your end users will see. At run time, you can't invoke the form designer, and you h e only a limited ability to modify the code you have written at design time. For insta e, you can modify existing statements and add new ones, but you can't add new pro dures, forms, or controls. On the other hand, at run time you can use some diagnos c tools that aren't available at design time because they would make no sense in that ontext (for example, the Locals, the Watches, and the Call Stack windows).

To create one or more controls on a form's surface, you select the control t e that you want from the Toolbox window, click on the form, and drag the mous cursor until the control has the size and shape you want. (Not all controls are r izable. Some, such as the Timer control, will allow you to drag but will return their original size and shape when you release the mouse button.) Alternatively, u can place a control on the form's surface by double-clicking its icon in the Tool x: this action creates a control in the center of the form. Regardless of the met d you follow, you can then move and resize the control on the form using the mouse

TIP

If you need to create multiple controls of the same type, you can follov this three-step procedure: First, click on the control's icon on the Toolbo window while you keep the Ctrl key pressed. Next, draw multiple controls b clicking the left button on the form's surface and then dragging the cursor Finally, when you're finished creating controls, press the Escape key or clic the Pointer icon in the upper left corner of the Toolbox.

To complete our Rectangle sample application, we need four TextBox contrc —two for entering the rectangle's width and height and two for showing the ı sulting perimeter and area, as shown in Figure 1-8. Even if they aren't strictly requiı l from an operational point of view, we also need four Label controls for clarif ıg the purpose of each TextBox control. Finally we add a CommandButton contro named *Evaluate* that starts the computation and shows the results.

Place these controls on the form, and then move and resize them as depicted i Figure 1-8. Don't worry too much if the controls aren't perfectly aligned because ɔu can later move and resize them using the mouse or using the commands in the ˈormat menu.

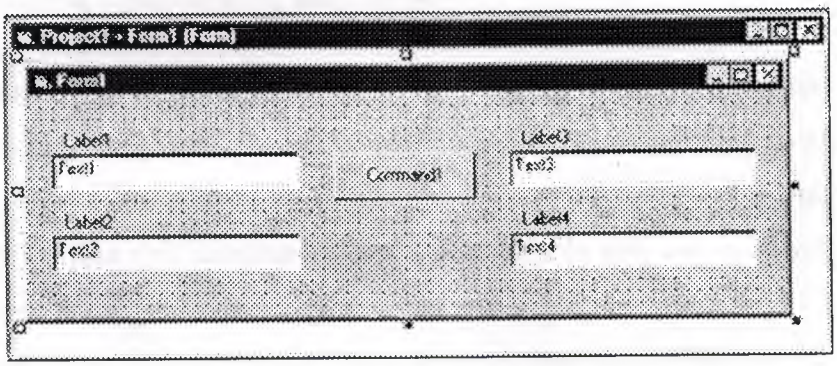


Figure 1-8 The Rectangle Demo form at design time, soon after the place ıent of its controls.

1.5.Setting Properties of Controls

Each control is characterized by a set of properties that define its beha ɔr and appearance. For instance, Label controls expose a *Caption* property that corı sponds to the character string displayed on the control itself, and a *BorderStyle* prop ty that affects the appearance of a border around the label. The TextBox contro ; most important property is *Text*, which corresponds to the string of characters that ppears within the control itself and that can be edited by the user.

In all cases, you can modify one or more properties of a control by selec ıg the control in the form designer and then pressing F4 to show the Properties windı v. You

can scroll through the contents of the Properties window until the propert you're interested in becomes visible. You can then select it and enter a new value.

Using this procedure, you can modify the *Caption* property of all four Label ontrols to *&Width*, *&Height*, *&Perimeter*, and *&Area*, respectively. You will note at the ampersand character doesn't appear on the control and that its effect is to u derline the character that follows it. This operation actually creates a *hot key* and ass iates it with the control. When a control is associated with a hot key, the user can uickly move the focus to the control by pressing an Alt+*x* key combination, as you r rmally do within most Windows applications. Notice that only controls exposing a *aption* property can be associated with a hot key. Such controls include the Label. Frame, CommandButton, OptionButton, and CheckBox.

A quick way to select all the controls on a form is to click anywhere o the form and press the Ctrl+A key combination. After selecting all controls you can deselect a few of them by clicking on them while pressing the Shift o Ctrl key. Note that this shortcut doesn't selectcontrols that are contained i other controls.When you select a group of controls and then press the F4 key the Properties windowdisplays only the properties that are common to all th selected controls. The onlyproperties that are exposed by any control are *Lef Top*, *Width*, and *Height*. If youselect a group of controls that display a string o characters, such as the TextBox,Label, and CommandButton controls in ou Rectangle example, the *Font* property isalso available and can therefore b selected. When you double-click on the *Font* item in the Properties window, Font dialog box appears. Let's select a Tahoma font and set its size to 1 points.

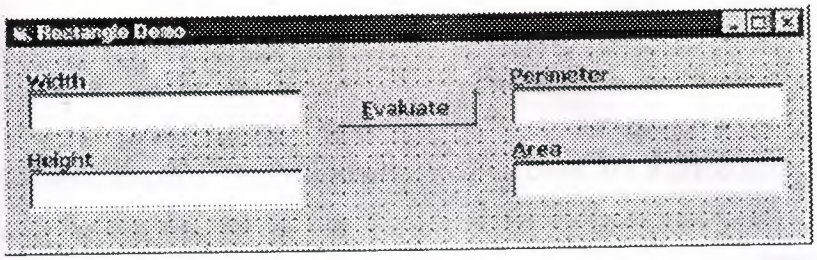


Figure 1-9. The Rectangle Demo form at design time, after setting the c ntrols' proper*ties.*

TIP

When a control is created from the Toolbox, its *Font* property reflect he font of the parent form. For this reason, you can often avoi individual font settings by changing the form's *Font* property befor placing any controls on the form itself.

1.6.Naming Controls

One property that every control has and that's very important to Visu Basic programmers is the *Name* property. This is the string of characters that ident ies the control in code. This property can't be an empty string, and you can't have two or more controls on a form with the same name. The special nature of this pr erty is indirectly confirmed by the fact that it appears as (Name) in the Properties ndow, where the initial parenthesis serves to move it to the beginning of the property st.

When you create a control, Visual Basic assigns it a default name. For exan le, the first TextBox control that you place on the form is named *Text1*, the secon one is named *Text2*, and so forth. Similarly, the first Label control is named *Label1*, nd the first CommandButton control is named *Command1*. This default naming sche e frees you from having to invent a new, unique name each time you create a contro Notice that the *Caption* property of Label and CommandButton controls, as well as e *Text* property of TextBox controls, initially reflect the control's *Name* property, but he two properties are independent of each other. In fact, you have just modified the *aption*

and *Text* properties of the controls in the Rectangle Demo form without affect g their *Name* properties.

Because the *Name* property identifies the control in code, it's a good habit to n dify it so that it conveys the meaning of the control itself. This is as important as s lecting meaningful names for your variables. In a sense, most controls on a form are special variables whose contents are entered directly by the user.

Microsoft suggests that you always use the same three-letter prefix for all the ontrols of a given class. The control classes and their recommended prefixes are show in

Table 1-1.Table 1-1. Standard three-letter prefixes for forms and all i trinsic controls.

Control Class	*Prefix*	*Control Class*	*Prefix*
CommandButton	cmd	Data	dat
TextBox	txt	HScrollBar	hsb
Label	lbl	VScrollBar	vsb
PictureBox	pic	DriveListBox	drv
OptionButton	opt	DirListBox	dir
CheckBox	chk	FileListBox	fil
ComboBox	cbo	Line	lin
ListBox	lst	Shape	shp
Timer	tmr	OLE	ole
Frame	fra	Form	Frm

For instance, you should prefix the name of a TextBox control with txt, the n ne of a Label control with lbl, and the name of a CommandButton control with cmd Forms

should also follow this convention, and the name of a form should be prefi› d with the *frm* string. This convention makes a lot of sense because it lets you ded e both the control's type and meaning from its name. This book sticks to this aming convention, especially for more complex examples when code readability is at take.

In our example, we will rename the Text1 through Text4 controls as t› Width, txtHeight, txtPerimeter, and txtArea respectively. The Command1 control vill be renamed cmdEvaluate, and the four Label1 through Label4 controls will be named lblWidth, lblHeight, lblPerimeter, and lblArea, respectively. However, ple e note that Label controls are seldom referred to in code, so in most cases you can le e their

1.7.Adding Code

Up to this point, you have created and refined the user interface of your prog m and created an application that in principle can be run. (Press F5 and run it to c nvince yourself that it indeed works.) But you don't have a useful application yet. o turn your pretty but useless program into your first working application, you nee to add some code. More precisely, you have to add some code in the *Click* even of the cmdEvaluate control. This event fires when the user clicks on the Evaluate b tton or presses its associated hot key (the Alt+E key combination, in this case).

To write code within the *Click* event, you just select the cmdEvaluate control d then press the F7 key, or right-click on it and then invoke the View Code comma d from the pop-up menu. Or you simply double-click on the control using the lef mouse button. In all cases, the code editor window appears, with the flashing curso ocated between the following two lines of code:

```
Private Sub cmdEvaluate_Click()
End Sub
```

Visual Basic has prepared the template of the *Click* event procedure for you, d you have to add one or more lines of code between the *Sub* and *End Sub* staten nts. In this simple program, you need to extract the values stored in the txtWi th and

txtHeight controls, use them to compute the rectangle's perimeter and area, an assign the results to the txtPerimeter and txtArea controls respectively:

```
Private Sub cmdEvaluate_Click()
    ' Declare two floating point variables.
    Dim reWidth As Double, reHeight As Double
    ' Extract values from input TextBox controls.
    reWidth = CDbl(txtWidth.Text)
    reHeight = CDbl(txtHeight.Text)
    ' Evaluate results and assign to output text boxes.
    txtPerimeter.Text = CStr((reWidth + reHeight) * 2)
    txtArea.Text = CStr(reWidth * reHeight)
End Sub
```

1.8.Running and Debugging the Program

You're finally ready to run this sample program. You can start its execution ir several ways: By invoking the Start command from the Run menu, by click ıg the corresponding icon on the toolbar, or by pressing the F5 key. In all cases, y ı'll see the form designer disappear and be replaced (but not necessarily in the same osition on the screen) by the real form. You can enter any value in the leftmost xtBox controls and then click on the Evaluate button (or press the Alt+E key combin ion) to see the calculated perimeter and area in the rightmost controls. When you're ı ıished, end the program by closing its main (and only) form.

CAUTION

You can also stop any Visual Basic program running in the environment b invoking the End command from the Run menu, but in general this isn't good approach because it prevents a few form-related events—namely th

QueryUnload and the *Unload* events—from firing. In some cases, these even procedures contain the so-called *clean-up code,* for example, statements tha close a database or delete a temporary file. If you abruptly stop the executio of a program, you're actually preventing the execution of this code. As general rule, use the End command only if strictly necessary.This program i so simple that you hardly need to test and debug it. Of course, this wouldn't b true for any real-world application. Virtually all programs need to betested an debugged, which is probably the most delicate (and often tedious) part of programmer's job. Visual Basic can't save you from this nuisance, but at leas it offers so many tools that you can often complete it very quickly. To se some Visual Basic debugging tools in action, place a breakpoint on the firs line of the *Click* event procedure while the program is in design mode. Yo can set a breakpoint by moving the text cursor to the appropriate line and the invoking the

Toggle Breakpoint command from the Debug menu or pressing the F9 short it key. You can also set and delete breakpoints by left-clicking on the gray vertical s ip that runs near the left border of the code editor window. In all cases, the line on w ich the breakpoint is set will be highlighted in red.

After setting the breakpoint at the beginning of the *Click* event procedure, pre s F5 to run the program once again, enter some values in the Width and Height fie ls, and then click on the Evaluate button. You'll see the Visual Basic environment ent break mode, and you are free to perform several actions that let you better understan what's actually going on:

- Press F8 to execute the program one statement at a time. The Visu Basic instruction that's going to be executed next—that is, the current statem nt —is highlighted in yellow.
- Show the value of an expression by highlighting it in the code win w and then pressing F9 (or selecting the Quick Watch command from th Debug menu). You can also add the selected expression to the list of values d played in the Watch window, as you can see in Figure 1-10.

- An alternative way to show the value of a variable or a property is to n ›ve the mouse cursor over it in the code window; after a couple of seconds, ; yellow *data tip* containing the corresponding value appears.
- Evaluate any expression by clicking on the Immediate window and typ ıg ? or *Print* followed by the expression. This is necessary when you need to 'aluate the value of an expression that doesn't appear in the code window.
- You can view the values of all the local variables (but not express ns) by selecting the Locals command from the View menu. This com ınd is particularly useful when you need to monitor the value of many local ı riables and you don't want to set up a watching expression for each one.
- You can affect the execution flow by placing the text cursor on the s tement that you want to execute next and then selecting the Set Next S tement command from the Debug menu. Or you can press the Ctrl+ 9 key combination. You need this technique to skip over a piece of code at you don't want to execute or to reexecute a given block of lines without r tarting the program.

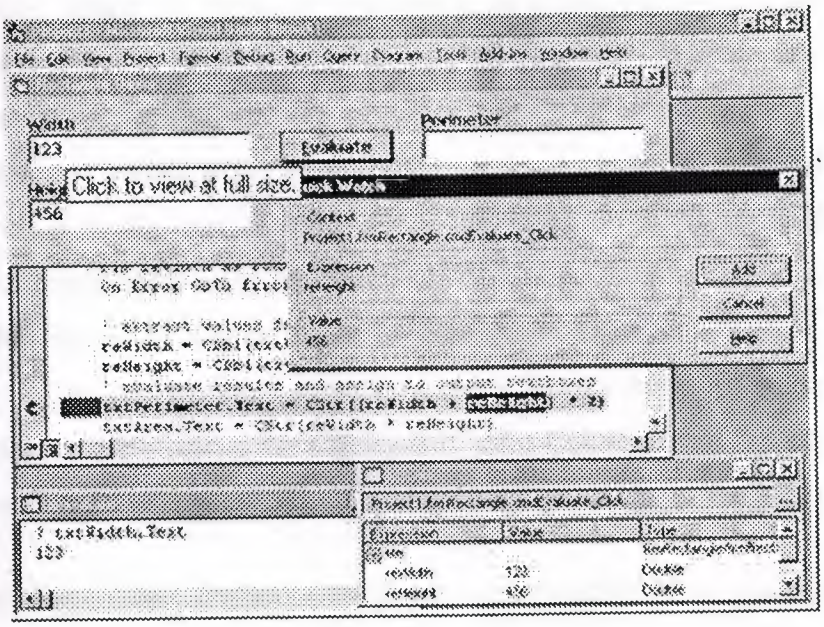


Figure 1-10. The Rectangle Demo program in break mode, with severa debug tools activated.

1.9.Refining the Sample Program

Our first Visual Basic project, Rectangle.vbp, is just a sample program, but t s is no excuse not to refine it and turn it into a complete and robust, albeit trivial, app ation.

The first type of refinement is very simple. Because the txtPerimeter and xtArea controls are used to show the results of the computation, it doesn't make nse to make their contents editable by the user. You can make them read-only f lds by setting their *Locked* property to True. (A suggestion: select the two controls, p ss F4, and modify the property just once.) Some programmers prefer to use Label co rols to display result values on a form, but using read-only TextBox controls as an advantage: The end user can copy their contents to the clipboard and pas those contents into another application.

A second refinement is geared toward increasing the application's consiste y and usability. Let's suppose that your user uses the Rectangle program to deter ne the perimeter and area of a rectangle, takes note of the results, and then enter a new width or a new height (or both). Unfortunately, an instant before your user c cks on the Evaluate button the phone rings, engaging the user in a long conversatio When he or she hangs up, the form shows a plausible, though incorrect, result. How an you be sure that those values won't be mistaken for good ones? The solution is imple, indeed: as soon as the user modifies either the txtWidth or the txtHeight xtBox controls, the result fields must be cleared. In Visual Basic, you can accomp sh this task by trapping each source control's *Change* event and writing a cc ple of statements in the corresponding event procedure. Since *Change* is the dèfa event for TextBox controls—just as the *Click* event is for CommandButtons contr —you only have to double-click the txtWidth and txtHeight controls on the form desi ner to

have Visual Basic create the template for the corresponding event procedures This is the code that you have to add to the procedures:

```
Private Sub txtWidth_Change()
    txtPerimeter.Text = ""
    txtArea.Text = ""
End Sub
```

```
Private Sub txtHeight_Change()
    txtPerimeter.Text = ""
    txtArea.Text = ""
End Sub
```

Note that you don't have to retype the statements in the *txtHeight*'s *Chan* event procedure: just double-click the control to create the *Sub ... End Sub* templ e, and then copy and paste the code from the *txtWidth_Click* procedure. Whe you're finished, press F5 to run the program to check that it now behaves as expected

The purpose of the next refinement that I am proposing is to increase the p gram's robustness. To see what I mean, run the Rectangle project and press the l aluate button without entering width or height values: the program raises a Type M match error when trying to extract a numeric value from the txtWidth control. If thi were a real-world, compiled application, such an *untrapped* error would cause the ap ication to end abruptly, which is, of course, unacceptable. All errors should be trap ed and dealt with in a convenient way. For example, you should show the user w re the problem is and how to fix it. The easiest way to achieve this is by setting up n error handler in the *cmdEvaluate_Click* procedure, as follows. (The lines you would dd are in boldface.)

```
Private Sub cmdEvaluate_Click()
    ' Declare two floating point variables.
    Dim reWidth As Double, reHeight As Double
    On Error GoTo WrongValues
    ' Extract values from input textbox controls.
    reWidth = CDbl(txtWidth.Text)
    reHeight = CDbl(txtHeight.Text)
    Ensure that they are positive values.
    If reWidth <= 0 Or reHeight <= 0 Then GoTo WrongValues
    ' Evaluate results and assign to output text boxes.
    txtPerimeter.Text = CStr((reWidth + reHeight) * 2)
    txtArea.Text = CStr(reWidth * reHeight)
    Exit Sub
WrongValues:
    MsgBox "Please enter valid Width and Height values",
vbExclamation
```

```
End Sub
```

Note that we have to add an *Exit Sub* statement to prevent the *MsgBox* statem ıt from being erroneously executed during the normal execution flow. To see how he *On Error* statement works, set a breakpoint on the first line of this procedure, un the application, and press the F8 key to see what happens when either of the ·xtBox controls contains an empty or invalid string.

1.10.Ready, Compile, Run!

Visual Basic is a very productive programming language because it allow: you to build and test your applications in a controlled environment, without first pro ıcing a compiled executable program. This is possible because Visual Basic conve s your source code into *p-code* and then interprets it. P-code is a sort of inte ıediate language, which, because it's not executed directly by the CPU, is slower t ın real natively compiled code. On the other hand, the conversion from source code t p-code takes only a fraction of the time needed to deliver a compiled application. ıis is a great productivity bonus unknown to many other languages. Another bene of p-code is that you can execute it step-by-step while the program is runnin in the environment, investigate the values of the variables, and—to some exter —even modify the code itself. This is a capability that many other languages don't ave or have acquired only recently; for example, the latest version of Microsoft Vis ıl C++ has it. By comparison, Visual Basic has always offered this feature which undoubtedly contributed to making it a successful language.At some time du ng the program development, you might want to create an executable (EXE) prograr There are several reasons to do this: compiled programs are often (much) fas r than interpreted ones, users don't need to install Visual Basic to run your applicat ın, and you usually don't want to let other people peek at your source code. Visu Basic makes the compilation process a breeze: when you're sure that your appli tion is completed, you just have to run the Make *projectname* command from the Fil nenu.

It takes a few seconds to create the Rectangle.exe file. This executabl file is independent of the Visual Basic environment and can be executed in the sam way as any other Windows application—for example, from the Run command of t e Start menu. But this doesn't mean that you can pass this EXE file to another ı er and

expect that it works. All Visual Basic programs, in fact, depend on a nu ber of ancillary files—most notably the MSVBVM60.DLL file, a part of the Visu Basic runtime—and won't execute accurately unless all such files are correctly inst led on the target system.,

For this reason, you should never assume that a Visual Basic program will ex ute on every Windows system because it's working on your computer or on other co puters in your office. (If your business is software development, it's highly probable hat the Visual Basic environment is installed on all the computers around you.) stead, prepare a standard installation using the Package and Deployment Wizard, nd try running your application on a clean system. If you develop software profes onally, you should always have such a clean system at hand, if possible with st the operating system installed. If you're an independent developer, you probably on't be inclined to buy a complete system just to test your software. I found a very sin le and relatively inexpensive solution to this dilemma: I use one computer with re ovable hard disks, so I can easily test my applications under different system config ations. And since a clean system requires only hundreds of megabytes of disk spa . I can recycle all of my old hard disks that aren't large enough for any other use.

Before I conclude this chapter, you should be aware of one more det l. The compilation process doesn't necessarily mean that you aren't using p-code In the Visual Basic jargon, *compiling* merely means *creating an executable file*. In f ct, you can compile to p-code, even if this sounds like an oxymoron to a developer oming from another language. (See Figure 1-11.) In this case, Visual Basic creates an EXE

file that embeds the same p-code that was used inside the development envir nment. That's why you can often hear Visual Basic developers talking about *p-c le* and *native-code* compilations to better specify which type of compilation they're r ferring to.

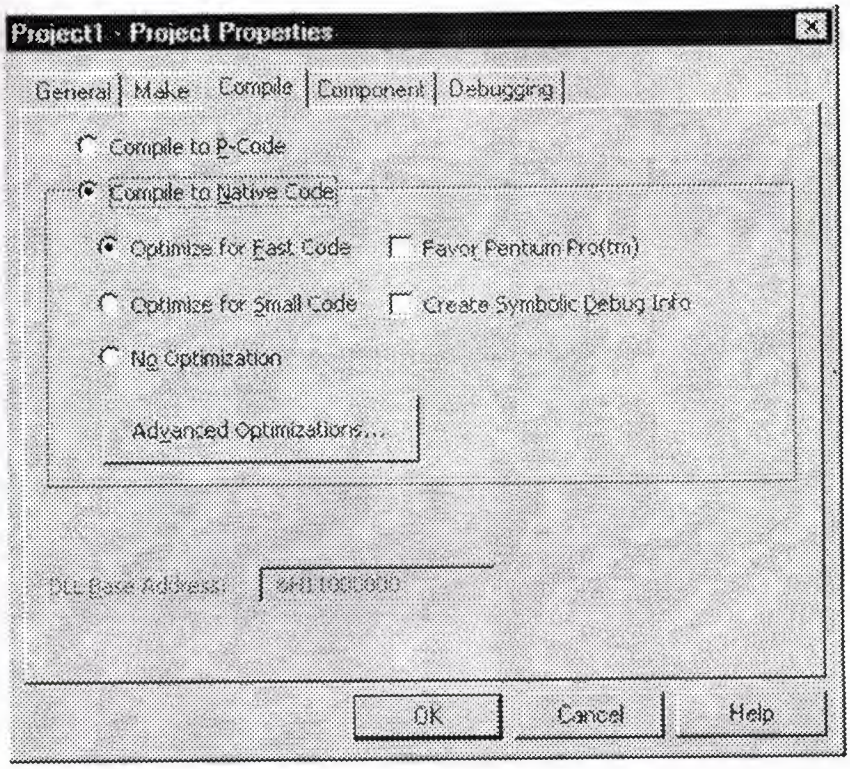


Figure 1-11. You can opt to compile to p-code or native code in the Com ile tab of the Project Properties dialog.

In general, such p-code-compiled programs run at the same speed as int preted programs within the IDE, so you're missing one of the biggest benefit of the compilation process. But here are a few reasons why you might decide to cr te a p-code executable:

- P-code-compiled executables are often smaller than programs com led to native code. This point can be important if you're going to distrib e your application over the Internet or when you're creating ActiveX controls hat are embedded in an HTML page.
- P-code compilation is often faster than native code compilation, so y might prefer to stick to p-code when you compile the program in the test p se. (A few types of applications can't be tested within the IDE, most otably multithreaded components.)
- If your application spends most of its time accessing databases or re awing windows, compilation to native code doesn't significantly imp ve its

performance because the time spent executing Visual Basic code is only a fraction of the total execution time.

We've come to the end of this *tour de force* in the Visual Basic IDE. In this chapter, I've illustrated the basics of Visual Basic development, and I hope I've given you a taste of how productive this language can be. Now you're ready to move to the next chapters, where you can learn more about forms and controls and about how to make the best of their properties, methods, and events.

CHAPTER 2

DATABASE AND ACCESS

2.1.Why is the computer necessary in our life

Computer software has become a driving force; it is a powerful force that s Decision-making and serves as a basis for modern investigation and problem solving.Com uters have become a key factor that gives products and services that modern look ,its e bedded in systems of all kinds;medical,industrial,military,entertainment,even office-based pr ducts.

A Computer system in a service management record can promise better speed and efficiency with almost no change of effors.

2.2.How to develop a database application

The steps involved in database application development any relational data base pplication there are always the same basic steps to follow.Microsoft Access is a relation data base management system because all data is stored in an Access data base in the for of simple tables.Another name for a table is relation.

The steps of Access database design like this

- Database design
- Tables design
- Forms design
- Query design
- Report design
- Macro design
- Modüles design

2.3.Relational database

DBMS has established themselves as one of the primary means for data torage for information based systems ranging from large business applications to simpl pc based programs.However a relational database management system (RDBMS) is the sy em used to work with data management operations more than 15 years,and still improvir ,providing more sophisticated storage,retriaval systems.Relational database manageme t systems

provides organisations with ability to handle huge ammount of data and chan ing it into meaningful information.

2.4.The facilities of access

Microsoft Access is relational DBMS(Database Management System) with all ıe features necessary to develop and use a data base application.The facilities it offers can l ɔ found on most modern relational DBMSs and all versions of Access.

- Tables are where all the data is stored.They are usually linked by relati ıships.
- Queries are the way you extract data from the database
- Forms are the method used for input and display of database data.
- Reports are used to display nicely formatted data on paper.
- Macros are sets of simple commands that execute sequences (' database operations.
- Modules are used to stare general-purpose VB database program code.

2.5.Visual basic and Access

Microsoft Acess is the DBMS(Database Management System) VB and Access in developing data base applications is that for non-trivial database applications,VB ffers more flexibility to the developper then the VB comes with Access.Access database using VB program code and setting properties.

First method of linking VB forms to Access databases called the data conti l.The data control is a simple VB control that you drag on to a VB form to link it to y(ır choosen database.The data can be displayed and updated using tiedtext boxes,list b xes,combo boxes,and grids.

2.5.1.DAO(Data Access objects)

The DAO approach to database programming often requires more code ,bu like SQL compared to the Qury Design View,offers greater control to the database progr nmer over what's going on his/her application.

Data Access Objects are things like databases,recordsets,table and query defi itions, and fields,Rather than tying a recor set to a data control when we use DAO we sha allow our programs to create and manipulate recordsets.

2.5.2.ADO(Active X Data Objects)

The ADO programming is in principle very similar to DAO programming b t cointains some new commands.ADO is Microsoft's new approach to database program ing which aims to give the programmer a more consistent way of connecting to a bro l range of different types of data source.

2.6.The application of Access

MS Access is begin used as the development tool,and the application is going to b a single user application,which means its going to be instolled on one machine,this applica on however may be used by more than one user on many computers sharing the same ables by using simple advancements.

For a new database,after having specified the database name and path as above,yo will be confronted with the following window.

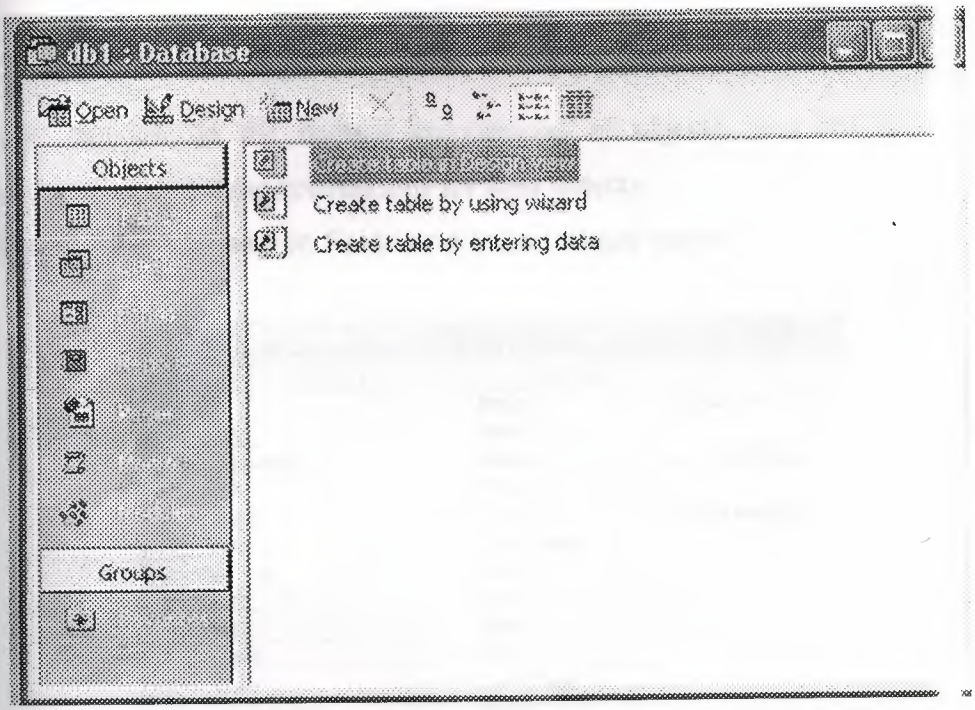


Figure1.1. The window of database

This window shows that there are notables in database yet.Click new button.

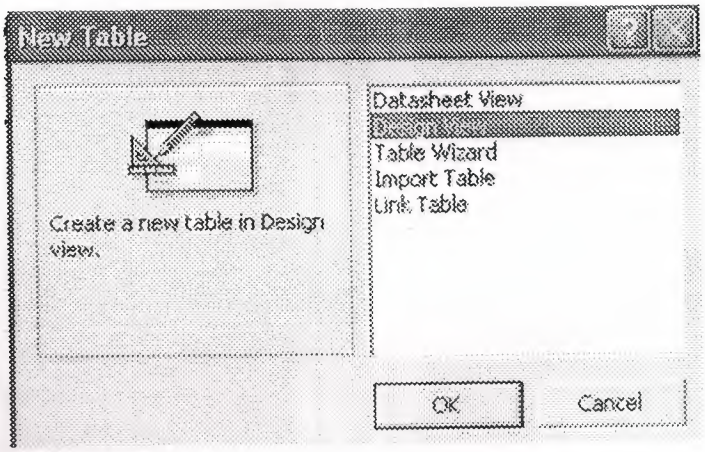


Figure 1.2. The window is type of table design

sellect the Design View by clicking on the listbox and then the OK button.Design View gives us more control over the design of our database than either the Table Wizard or the Datasheet view.Import Table is used to bring in data from an existing database and Link Table is database to an external table.

2.6.1. Tables Design

In my project's table designing with primary key. Guide Lines for making a database project.

The database consists of one tables;

Table1

Please pay attention on the naming conventions of objects, you are required to use appropriate names using these conventions for your objects.

TheTable one have got eleven fields one is its unique name

Table1 : Tablo

Alan Adı	Veri Türü	
id	Metin	numaralar
name	Metin	
explanation	Metin	urun aciklamasi
piece	Metin	
a_item	Metin	active madde
l_date	Tarih/Saat	
shelf_no	Metin	
price	Metin	
f_tel	Metin	
f_name	Metin	
b_date	Tarih/Saat	
p_explenation	Metin	

Figure 1.3 The Table1

NEAR EAST UNIVERSITY

Faculty of Engineering

Department of Computer Engineering

STOCK CONTROL MANAGAMENT

Graduation Project
COM-400

Student: Menderes BOZKURT

Supervisor: Ümit İLHAN

Nicossia - 2004

TABLE OF CONTENTS

ACKNOWLDGMENTS

First of all I would like to thank Mr. UMIT ILHAN for his endless ıd untiring support and help and his persistence, in the course of the preparation of this oject.

Under his guidance, I have overcome many difficulties that I face during the various stages of the preparation of this project.

Finally, I would like to thank my family, especially my father who na ıe is Mr HASAN BOZKURT and my brothers who name is Mr MURAT BOZKURT MUSA BOZKURT , IBRAHIM BOZKURT and ALİI BOZKURT , .Their love and uidance saw me through doubtul times. Their never-ending belief in me and their encouragement has been a crucial and a very strong pillar that has held me t ;ether

ABSTRACT

As the information age has effected every aspect of our life, t· need for computerizing many information systems has raised.

Once of the important branches that are effected by information revc ıtion is the computer programming languages.

This project is concerned about using compueter program ı Pharmacy management system . It is written using Visual Basic 6.0 programming l: guage and used Microsoft Access Database language for databases. Visual Basic is on of the best and easy programming languages.

This project is accomlete Pharmacy management program, tha covers all services needed in most Pharmacy, such as computer related informati(,madicine, goods and many other Pharmacy management related services.

Before coming to this point, this project has gone through some impc tant steps;

- First one was the requirements definition for which I h d to go to some Pharmacy and study their systems.
- The second steps were designing the system and sof\ ıre that is intended to serve an integrated Pharmacy management sys ı m.
- The final steps was the implementation of the design on t : computer using Visual Basic Language.

INTRODUCTION

Visual Basic is a Microsoft Windows programming Language.Visu Basic programs are created in an Integrated Development Environment (IDE) . T e IDE allows the programmer to create , run and debug Visual Basic programs conv iently. IDEs allow a programmer to create working programs in a fraction of the tin that it would normally take to code programs without using IDEs. The process of apidly creating an application is typically referred to as Rapid Application Deve pment (RAD). Visual Basic is the world's most widely used RAD language.

Visual Basic is derived from the BASIC programming language. Visu Basic is a distinctly different language providing powerfull features such as graphi al user interfaces, even handling, access to the Win32 API, object-oriented featur , error handling, structured programming, and much more.

The Visual Basic IDE allows Windows programs to be created wit ut the need for the programmer to be a Windows programming export.

Microsoft provides several version of Visual Basic, namely the l arning Edition , the Professional Edition and the Enterprice Edition. The Learning Edition provides fundemantal programming capabilities than the Learning Edition an is the choice of many programmers to write Visual Basic applications. The E erprice Edition is used for developing large-scale computing systems that meet the eds of substandial organizations.

Visual Basic is an interpreted language. However , the professio al and Enterprice Edition allows Visual Basic code to be compiled to native code.

Visual Basic evolved from BASIC(Beginner's All purpose S nbolic Instruction Code). Basic was developed in the mid 1960's by Professor John Kemeny and Thomas Kurtz of Darthmouth College as a language for writing simple programs. BASIC's primary purpose was to help people learn how to progran

The widespread use of BASIC with various types of computers (so etimes called hardware platforms) led to many enhancements to the language. W ith the development of the Microsoft windows graphical user interface (GUI) in e late

1980s and the early 1990s, the natural evolution of BASIC was Visual Basi which was created by Microsoft Corporation in 1991.

Until Visual Basic appeared, develoing Microsoft Windov -based applications was a diffucult and cumbersome process. Visual Basic greatly si plifies Windows application development. Since 1991 six versions have been releas l, with the latest-Visual Basic 6-appearing in september 1998.

After a brief explanation about the Visual Basic 6.0 and the de loping layers, I hope that you will find the necessary information that you need about the Visual Basic even if you are a text based programmer.

CHAPTER1

Visual Basic Program

1.1.VB Advantages

So what makes VB a great programming language? The answer is simply that B provides more of the actual code for a programmer than any other non-visual programming language.

If you've ever programmed in the older BASIC or other command line progra ning language, then you'll remember that the programmer had to write the code for e entire user interface. Todays windows, buttons, lists, and other application fea res such as menus were not built-in to the BASIC programming language. Progra mers had to create the code for these features on their own!

As much as 80% of a programmer's time was spent writing code to create the er interface to his applications (the visual interface). To eliminate this huge drain n a programmer's time, Microsoft has provided Visual Basic with the built-in c ability to create the user interface using nothing more than a mouse!

This built-in interface creation capability has had the further benefit of stand dizing on the user interface to Windows applications. Today, users can move fi m one Windows program to another and see the same basic interface tools to wor with - allowing them to concentrate solely on the unique capabilities of the applicati .

The bottom line is that you can create an entire application shell (the user i erface) very quickly and then spend most of your time working on the feature which differentiate you application from its competition.

1.2.Very First Visual Basic Program

Visual Basic lets you build a complete and functional Windows applic on by dropping a bunch of controls on a form and writing some code that execut when something happens to those controls or to the form itself. For instance, you c write code that executes when a form loads or unloads or when the user resizes it. L ewise, you can write code that executes when the user clicks on a control or types w ile the control has the input focus.

This programming paradigm is also known as *event-driven programming* ecause your application is made up of several event procedures executed in an ord that's dependent on what happens at run time. The order of execution can't, in ge ral, be foreseen when the program is under construction. This programming model ntrasts with the procedural approach, which was dominant in the old days.

This section offers a quick review of the event-driven model and uses a ample application as a context for introducing Visual Basic's intrinsic controls, w h their properties, methods, and events. This sample application, a very simple one, ueries the user for the lengths of the two sides of a rectangle, evaluates its perim er and area, and displays the results to the user. Like all lengthy code examples and p grams illustrated in this book, this application is included on the companion CD.

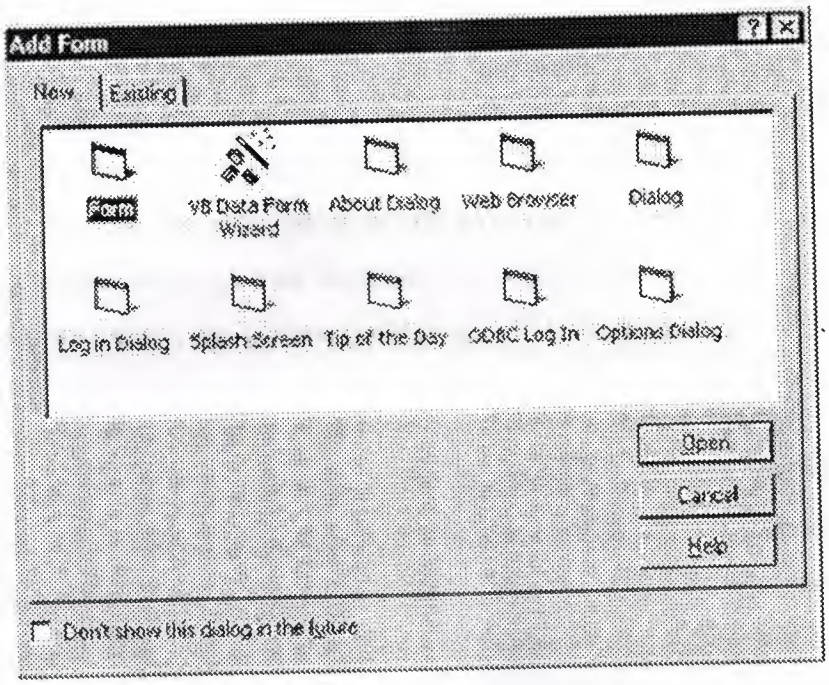
1.3.The Form Object

After this long introductory description of properties, methods, and events at are common to most Visual Basic objects, it's time to see the particular features all of them individually. The most important visible object is undoubtedly the Form object because you can't display any control without a parent Form. Conversely, u can write some moderately useful applications using only forms that have no co ols on them. In this section, I'll show a number of examples that are centered o forms' singular features.

You create a new form at design time using the Add Form command from the Project menu or by clicking on the corresponding icon on the standard toolbar. u can create forms from scratch, or you can take advantage of the many form te plates provided by Visual Basic 6. If you don't see the dialog box shown in Fig e 2-7, invoke the Options command from the Tools menu, click the Environment b, and select the topmost check box on the right.

Feel free to create new form templates when you need them. A form templat doesn't necessarily have to be a complex form with many controls on it. Even an em y form with a group of properties carefully set can save you some precious ti e. For

example, see the Dialog Form template provided by Visual Basic. To produ e your custom form templates, you just have to create a form, add any necessary cont ols and code, and then save it in the \Template\Forms directory. (The complete path o Visual Basic's template directory can be read and modified in the Environment ta of the Options dialog box.)



1.4.Adding Controls to a Form

We're ready to get practical. Launch the Visual Basic IDE, and select a Stand d EXE project. You should have a blank form near the center of the work are More accurately, you have a *form designer*, which you use to define the appearan of the main window of your application. You can also create other forms, if you ne them, and you can create other objects as well, using different designers (the Use ontrol and UserDocument designers, for example). Other chapters of this book are evoted to such designers.

One of the greatest strengths of the Visual Basic language is that programm rs can design an application and then test it without leaving the environment. But yo should be aware that designing and testing a program are two completely different t ks. At *design time,* you create your forms and other visible objects, set their proper s, and write code in their event procedures. Conversely, at *run time* you monitor th effects of your programming efforts: What you see on your screen is, more or less, w t your end users will see. At run time, you can't invoke the form designer, and you h e only a limited ability to modify the code you have written at design time. For insta e, you can modify existing statements and add new ones, but you can't add new pro dures, forms, or controls. On the other hand, at run time you can use some diagnos c tools that aren't available at design time because they would make no sense in that ontext (for example, the Locals, the Watches, and the Call Stack windows).

To create one or more controls on a form's surface, you select the control t e that you want from the Toolbox window, click on the form, and drag the mous cursor until the control has the size and shape you want. (Not all controls are r izable. Some, such as the Timer control, will allow you to drag but will return their original size and shape when you release the mouse button.) Alternatively, u can place a control on the form's surface by double-clicking its icon in the Tool x: this action creates a control in the center of the form. Regardless of the met d you follow, you can then move and resize the control on the form using the mouse

TIP

If you need to create multiple controls of the same type, you can follov this three-step procedure: First, click on the control's icon on the Toolbo window while you keep the Ctrl key pressed. Next, draw multiple controls b clicking the left button on the form's surface and then dragging the cursor Finally, when you're finished creating controls, press the Escape key or clic the Pointer icon in the upper left corner of the Toolbox.

To complete our Rectangle sample application, we need four TextBox contrc —two for entering the rectangle's width and height and two for showing the ı sulting perimeter and area, as shown in Figure 1-8. Even if they aren't strictly requiı l from an operational point of view, we also need four Label controls for clarif ıg the purpose of each TextBox control. Finally we add a CommandButton contro named *Evaluate* that starts the computation and shows the results.

Place these controls on the form, and then move and resize them as depicted i Figure 1-8. Don't worry too much if the controls aren't perfectly aligned because ɔu can later move and resize them using the mouse or using the commands in the ˈormat menu.

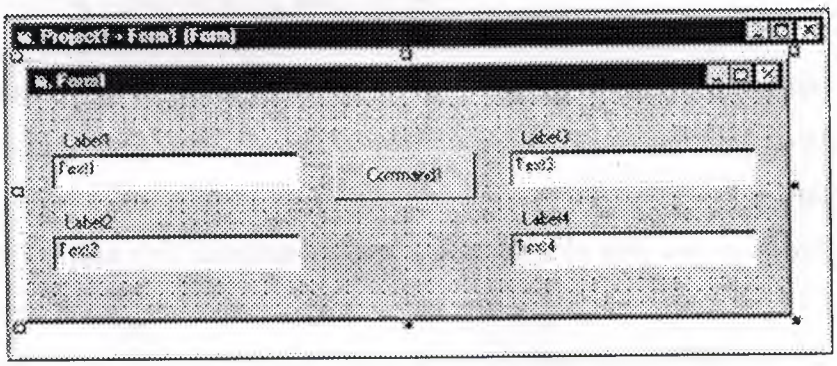


Figure 1-8 The Rectangle Demo form at design time, soon after the place ıent of its controls.

1.5.Setting Properties of Controls

Each control is characterized by a set of properties that define its beha ɔr and appearance. For instance, Label controls expose a *Caption* property that corı sponds to the character string displayed on the control itself, and a *BorderStyle* prop ty that affects the appearance of a border around the label. The TextBox contro ; most important property is *Text*, which corresponds to the string of characters that ppears within the control itself and that can be edited by the user.

In all cases, you can modify one or more properties of a control by selec ıg the control in the form designer and then pressing F4 to show the Properties windı v. You

can scroll through the contents of the Properties window until the propert you're interested in becomes visible. You can then select it and enter a new value.

Using this procedure, you can modify the *Caption* property of all four Label ontrols to *&Width*, *&Height*, *&Perimeter*, and *&Area*, respectively. You will note at the ampersand character doesn't appear on the control and that its effect is to u derline the character that follows it. This operation actually creates a *hot key* and ass iates it with the control. When a control is associated with a hot key, the user can uickly move the focus to the control by pressing an Alt+*x* key combination, as you r rmally do within most Windows applications. Notice that only controls exposing a *aption* property can be associated with a hot key. Such controls include the Label. Frame, CommandButton, OptionButton, and CheckBox.

A quick way to select all the controls on a form is to click anywhere o the form and press the Ctrl+A key combination. After selecting all controls you can deselect a few of them by clicking on them while pressing the Shift o Ctrl key. Note that this shortcut doesn't selectcontrols that are contained i other controls.When you select a group of controls and then press the F4 key the Properties windowdisplays only the properties that are common to all th selected controls. The onlyproperties that are exposed by any control are *Lef Top*, *Width*, and *Height*. If youselect a group of controls that display a string o characters, such as the TextBox,Label, and CommandButton controls in ou Rectangle example, the *Font* property isalso available and can therefore b selected. When you double-click on the *Font* item in the Properties window, Font dialog box appears. Let's select a Tahoma font and set its size to 1 points.

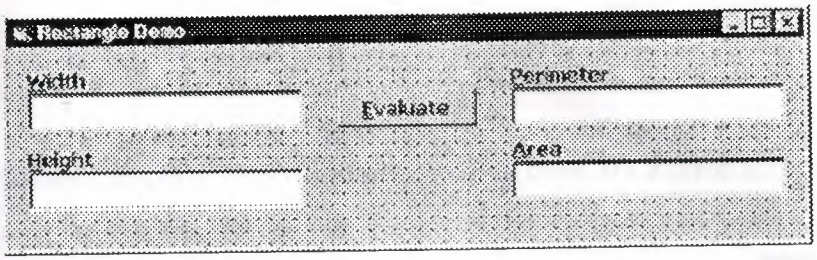


Figure 1-9. The Rectangle Demo form at design time, after setting the c ntrols' proper*ties.*

TIP

When a control is created from the Toolbox, its *Font* property reflect he font of the parent form. For this reason, you can often avoi individual font settings by changing the form's *Font* property befor placing any controls on the form itself.

1.6.Naming Controls

One property that every control has and that's very important to Visu Basic programmers is the *Name* property. This is the string of characters that ident ies the control in code. This property can't be an empty string, and you can't have two or more controls on a form with the same name. The special nature of this pr erty is indirectly confirmed by the fact that it appears as (Name) in the Properties ndow, where the initial parenthesis serves to move it to the beginning of the property st.

When you create a control, Visual Basic assigns it a default name. For exan le, the first TextBox control that you place on the form is named *Text1*, the secon one is named *Text2*, and so forth. Similarly, the first Label control is named *Label1*, nd the first CommandButton control is named *Command1*. This default naming sche e frees you from having to invent a new, unique name each time you create a contro Notice that the *Caption* property of Label and CommandButton controls, as well as e *Text* property of TextBox controls, initially reflect the control's *Name* property, but he two properties are independent of each other. In fact, you have just modified the *aption*

and *Text* properties of the controls in the Rectangle Demo form without affect g their *Name* properties.

Because the *Name* property identifies the control in code, it's a good habit to n dify it so that it conveys the meaning of the control itself. This is as important as s lecting meaningful names for your variables. In a sense, most controls on a form are special variables whose contents are entered directly by the user.

Microsoft suggests that you always use the same three-letter prefix for all the ontrols of a given class. The control classes and their recommended prefixes are show in

Table 1-1.Table 1-1. Standard three-letter prefixes for forms and all i trinsic controls.

Control Class	*Prefix*	*Control Class*	*Prefix*
CommandButton	cmd	Data	dat
TextBox	txt	HScrollBar	hsb
Label	lbl	VScrollBar	vsb
PictureBox	pic	DriveListBox	drv
OptionButton	opt	DirListBox	dir
CheckBox	chk	FileListBox	fil
ComboBox	cbo	Line	lin
ListBox	lst	Shape	shp
Timer	tmr	OLE	ole
Frame	fra	Form	Frm

For instance, you should prefix the name of a TextBox control with txt, the n ne of a Label control with lbl, and the name of a CommandButton control with cmd Forms

should also follow this convention, and the name of a form should be prefi› d with the *frm* string. This convention makes a lot of sense because it lets you ded e both the control's type and meaning from its name. This book sticks to this aming convention, especially for more complex examples when code readability is at take.

In our example, we will rename the Text1 through Text4 controls as t› Width, txtHeight, txtPerimeter, and txtArea respectively. The Command1 control vill be renamed cmdEvaluate, and the four Label1 through Label4 controls will be named lblWidth, lblHeight, lblPerimeter, and lblArea, respectively. However, ple e note that Label controls are seldom referred to in code, so in most cases you can le e their

1.7.Adding Code

Up to this point, you have created and refined the user interface of your prog m and created an application that in principle can be run. (Press F5 and run it to c nvince yourself that it indeed works.) But you don't have a useful application yet. o turn your pretty but useless program into your first working application, you nee to add some code. More precisely, you have to add some code in the *Click* even of the cmdEvaluate control. This event fires when the user clicks on the Evaluate b tton or presses its associated hot key (the Alt+E key combination, in this case).

To write code within the *Click* event, you just select the cmdEvaluate control d then press the F7 key, or right-click on it and then invoke the View Code comma d from the pop-up menu. Or you simply double-click on the control using the lef mouse button. In all cases, the code editor window appears, with the flashing curso ocated between the following two lines of code:

```
Private Sub cmdEvaluate_Click()
End Sub
```

Visual Basic has prepared the template of the *Click* event procedure for you, d you have to add one or more lines of code between the *Sub* and *End Sub* staten nts. In this simple program, you need to extract the values stored in the txtWi th and

txtHeight controls, use them to compute the rectangle's perimeter and area, an assign the results to the txtPerimeter and txtArea controls respectively:

```
Private Sub cmdEvaluate_Click()
    ' Declare two floating point variables.
    Dim reWidth As Double, reHeight As Double
    ' Extract values from input TextBox controls.
    reWidth = CDbl(txtWidth.Text)
    reHeight = CDbl(txtHeight.Text)
    ' Evaluate results and assign to output text boxes.
    txtPerimeter.Text = CStr((reWidth + reHeight) * 2)
    txtArea.Text = CStr(reWidth * reHeight)
End Sub
```

1.8.Running and Debugging the Program

You're finally ready to run this sample program. You can start its execution ir several ways: By invoking the Start command from the Run menu, by click ıg the corresponding icon on the toolbar, or by pressing the F5 key. In all cases, y ı'll see the form designer disappear and be replaced (but not necessarily in the same osition on the screen) by the real form. You can enter any value in the leftmost xtBox controls and then click on the Evaluate button (or press the Alt+E key combin ion) to see the calculated perimeter and area in the rightmost controls. When you're ı ıished, end the program by closing its main (and only) form.

CAUTION

You can also stop any Visual Basic program running in the environment b invoking the End command from the Run menu, but in general this isn't good approach because it prevents a few form-related events—namely th

QueryUnload and the *Unload* events—from firing. In some cases, these even procedures contain the so-called *clean-up code,* for example, statements tha close a database or delete a temporary file. If you abruptly stop the executio of a program, you're actually preventing the execution of this code. As general rule, use the End command only if strictly necessary.This program i so simple that you hardly need to test and debug it. Of course, this wouldn't b true for any real-world application. Virtually all programs need to betested an debugged, which is probably the most delicate (and often tedious) part of programmer's job. Visual Basic can't save you from this nuisance, but at leas it offers so many tools that you can often complete it very quickly. To se some Visual Basic debugging tools in action, place a breakpoint on the firs line of the *Click* event procedure while the program is in design mode. Yo can set a breakpoint by moving the text cursor to the appropriate line and the invoking the

Toggle Breakpoint command from the Debug menu or pressing the F9 short it key. You can also set and delete breakpoints by left-clicking on the gray vertical s ip that runs near the left border of the code editor window. In all cases, the line on w ich the breakpoint is set will be highlighted in red.

After setting the breakpoint at the beginning of the *Click* event procedure, pre s F5 to run the program once again, enter some values in the Width and Height fie ls, and then click on the Evaluate button. You'll see the Visual Basic environment ent break mode, and you are free to perform several actions that let you better understan what's actually going on:

- Press F8 to execute the program one statement at a time. The Visu Basic instruction that's going to be executed next—that is, the current statem nt —is highlighted in yellow.
- Show the value of an expression by highlighting it in the code win w and then pressing F9 (or selecting the Quick Watch command from th Debug menu). You can also add the selected expression to the list of values d played in the Watch window, as you can see in Figure 1-10.

- An alternative way to show the value of a variable or a property is to n ›ve the mouse cursor over it in the code window; after a couple of seconds, ; yellow *data tip* containing the corresponding value appears.
- Evaluate any expression by clicking on the Immediate window and typ ıg ? or *Print* followed by the expression. This is necessary when you need to 'aluate the value of an expression that doesn't appear in the code window.
- You can view the values of all the local variables (but not express ns) by selecting the Locals command from the View menu. This com ınd is particularly useful when you need to monitor the value of many local ı riables and you don't want to set up a watching expression for each one.
- You can affect the execution flow by placing the text cursor on the s tement that you want to execute next and then selecting the Set Next S tement command from the Debug menu. Or you can press the Ctrl+ 9 key combination. You need this technique to skip over a piece of code at you don't want to execute or to reexecute a given block of lines without r tarting the program.

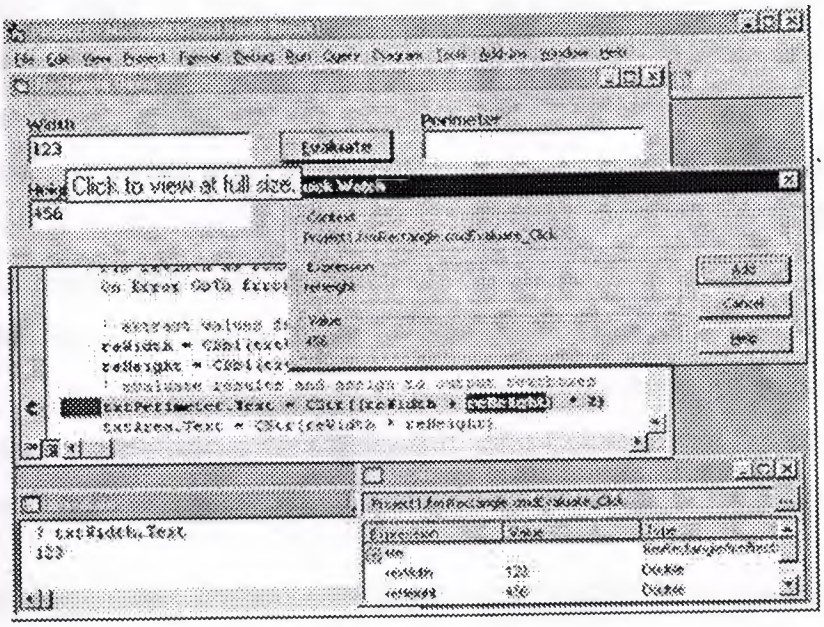


Figure 1-10. The Rectangle Demo program in break mode, with severa debug tools activated.

1.9.Refining the Sample Program

Our first Visual Basic project, Rectangle.vbp, is just a sample program, but t s is no excuse not to refine it and turn it into a complete and robust, albeit trivial, app ation.

The first type of refinement is very simple. Because the txtPerimeter and xtArea controls are used to show the results of the computation, it doesn't make nse to make their contents editable by the user. You can make them read-only f lds by setting their *Locked* property to True. (A suggestion: select the two controls, p ss F4, and modify the property just once.) Some programmers prefer to use Label co rols to display result values on a form, but using read-only TextBox controls as an advantage: The end user can copy their contents to the clipboard and pas those contents into another application.

A second refinement is geared toward increasing the application's consiste y and usability. Let's suppose that your user uses the Rectangle program to deter ne the perimeter and area of a rectangle, takes note of the results, and then enter a new width or a new height (or both). Unfortunately, an instant before your user c cks on the Evaluate button the phone rings, engaging the user in a long conversatio When he or she hangs up, the form shows a plausible, though incorrect, result. How an you be sure that those values won't be mistaken for good ones? The solution is imple, indeed: as soon as the user modifies either the txtWidth or the txtHeight xtBox controls, the result fields must be cleared. In Visual Basic, you can accomp sh this task by trapping each source control's *Change* event and writing a cc ple of statements in the corresponding event procedure. Since *Change* is the dèfa event for TextBox controls—just as the *Click* event is for CommandButtons contr —you only have to double-click the txtWidth and txtHeight controls on the form desi ner to

have Visual Basic create the template for the corresponding event procedures This is the code that you have to add to the procedures:

```
Private Sub txtWidth_Change()
    txtPerimeter.Text = ""
    txtArea.Text = ""
End Sub
```

```
Private Sub txtHeight_Change()
    txtPerimeter.Text = ""
    txtArea.Text = ""
End Sub
```

Note that you don't have to retype the statements in the *txtHeight*'s *Chan* event procedure: just double-click the control to create the *Sub ... End Sub* templ e, and then copy and paste the code from the *txtWidth_Click* procedure. Whe you're finished, press F5 to run the program to check that it now behaves as expected

The purpose of the next refinement that I am proposing is to increase the p gram's robustness. To see what I mean, run the Rectangle project and press the l aluate button without entering width or height values: the program raises a Type M match error when trying to extract a numeric value from the txtWidth control. If thi were a real-world, compiled application, such an *untrapped* error would cause the ap ication to end abruptly, which is, of course, unacceptable. All errors should be trap ed and dealt with in a convenient way. For example, you should show the user w re the problem is and how to fix it. The easiest way to achieve this is by setting up n error handler in the *cmdEvaluate_Click* procedure, as follows. (The lines you would dd are in boldface.)

```
Private Sub cmdEvaluate_Click()
    ' Declare two floating point variables.
    Dim reWidth As Double, reHeight As Double
    On Error GoTo WrongValues
    ' Extract values from input textbox controls.
    reWidth = CDbl(txtWidth.Text)
    reHeight = CDbl(txtHeight.Text)
    Ensure that they are positive values.
    If reWidth <= 0 Or reHeight <= 0 Then GoTo WrongValues
    ' Evaluate results and assign to output text boxes.
    txtPerimeter.Text = CStr((reWidth + reHeight) * 2)
    txtArea.Text = CStr(reWidth * reHeight)
    Exit Sub
WrongValues:
    MsgBox "Please enter valid Width and Height values",
vbExclamation
```

```
End Sub
```

Note that we have to add an *Exit Sub* statement to prevent the *MsgBox* statem ıt from being erroneously executed during the normal execution flow. To see how he *On Error* statement works, set a breakpoint on the first line of this procedure, un the application, and press the F8 key to see what happens when either of the ·xtBox controls contains an empty or invalid string.

1.10.Ready, Compile, Run!

Visual Basic is a very productive programming language because it allow: you to build and test your applications in a controlled environment, without first pro ıcing a compiled executable program. This is possible because Visual Basic conve s your source code into *p-code* and then interprets it. P-code is a sort of inte ıediate language, which, because it's not executed directly by the CPU, is slower t ın real natively compiled code. On the other hand, the conversion from source code t p-code takes only a fraction of the time needed to deliver a compiled application. ıis is a great productivity bonus unknown to many other languages. Another bene of p-code is that you can execute it step-by-step while the program is runnin in the environment, investigate the values of the variables, and—to some exter —even modify the code itself. This is a capability that many other languages don't ave or have acquired only recently; for example, the latest version of Microsoft Vis ıl C++ has it. By comparison, Visual Basic has always offered this feature which undoubtedly contributed to making it a successful language.At some time du ng the program development, you might want to create an executable (EXE) prograr There are several reasons to do this: compiled programs are often (much) fas r than interpreted ones, users don't need to install Visual Basic to run your applicat ın, and you usually don't want to let other people peek at your source code. Visu Basic makes the compilation process a breeze: when you're sure that your appli tion is completed, you just have to run the Make *projectname* command from the Fil nenu.

It takes a few seconds to create the Rectangle.exe file. This executabl file is independent of the Visual Basic environment and can be executed in the sam way as any other Windows application—for example, from the Run command of t e Start menu. But this doesn't mean that you can pass this EXE file to another ı er and

expect that it works. All Visual Basic programs, in fact, depend on a nu ber of ancillary files—most notably the MSVBVM60.DLL file, a part of the Visu Basic runtime—and won't execute accurately unless all such files are correctly inst led on the target system.,

For this reason, you should never assume that a Visual Basic program will ex ute on every Windows system because it's working on your computer or on other co puters in your office. (If your business is software development, it's highly probable hat the Visual Basic environment is installed on all the computers around you.) stead, prepare a standard installation using the Package and Deployment Wizard, nd try running your application on a clean system. If you develop software profes onally, you should always have such a clean system at hand, if possible with st the operating system installed. If you're an independent developer, you probably on't be inclined to buy a complete system just to test your software. I found a very sin le and relatively inexpensive solution to this dilemma: I use one computer with re ovable hard disks, so I can easily test my applications under different system config ations. And since a clean system requires only hundreds of megabytes of disk spa . I can recycle all of my old hard disks that aren't large enough for any other use.

Before I conclude this chapter, you should be aware of one more det l. The compilation process doesn't necessarily mean that you aren't using p-code In the Visual Basic jargon, *compiling* merely means *creating an executable file*. In f ct, you can compile to p-code, even if this sounds like an oxymoron to a developer oming from another language. (See Figure 1-11.) In this case, Visual Basic creates an EXE

file that embeds the same p-code that was used inside the development envir nment. That's why you can often hear Visual Basic developers talking about *p-c le* and *native-code* compilations to better specify which type of compilation they're r ferring to.

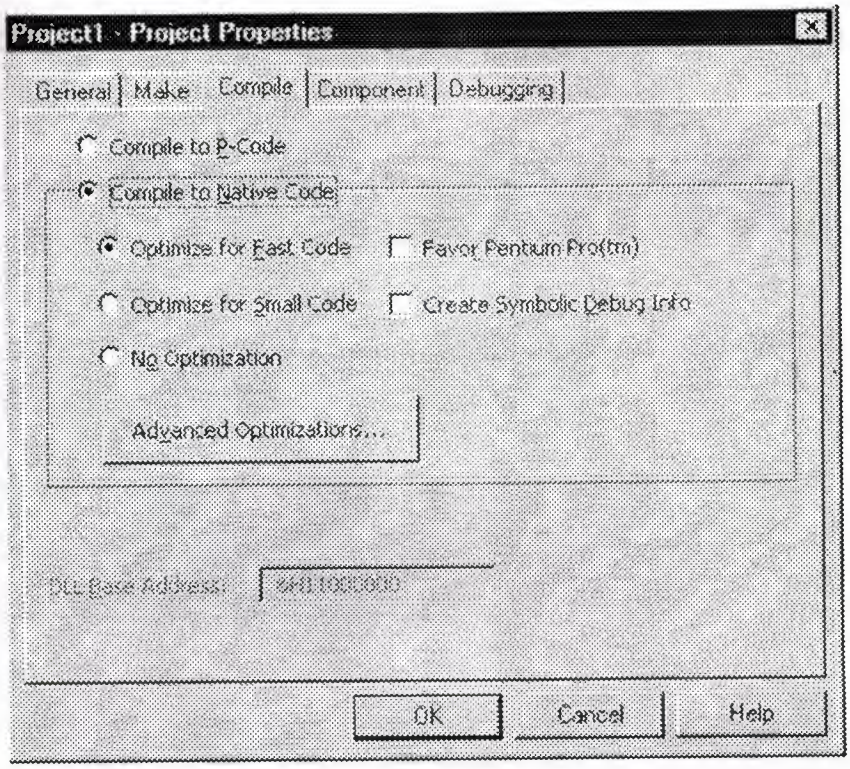


Figure 1-11. You can opt to compile to p-code or native code in the Com ile tab of the Project Properties dialog.

In general, such p-code-compiled programs run at the same speed as int preted programs within the IDE, so you're missing one of the biggest benefit of the compilation process. But here are a few reasons why you might decide to cr te a p-code executable:

- P-code-compiled executables are often smaller than programs com led to native code. This point can be important if you're going to distrib e your application over the Internet or when you're creating ActiveX controls hat are embedded in an HTML page.
- P-code compilation is often faster than native code compilation, so y might prefer to stick to p-code when you compile the program in the test p se. (A few types of applications can't be tested within the IDE, most otably multithreaded components.)
- If your application spends most of its time accessing databases or re awing windows, compilation to native code doesn't significantly imp ve its

performance because the time spent executing Visual Basic code is only a fraction of the total execution time.

We've come to the end of this *tour de force* in the Visual Basic IDE. In this chapter, I've illustrated the basics of Visual Basic development, and I hope I've given you a taste of how productive this language can be. Now you're ready to move to the next chapters, where you can learn more about forms and controls and about how to make the best of their properties, methods, and events.

CHAPTER 2

DATABASE AND ACCESS

2.1.Why is the computer necessary in our life

Computer software has become a driving force; it is a powerful force that s Decision-making and serves as a basis for modern investigation and problem solving.Com uters have become a key factor that gives products and services that modern look ,its e bedded in systems of all kinds;medical,industrial,military,entertainment,even office-based pr ducts.

A Computer system in a service management record can promise better speed and efficiency with almost no change of effors.

2.2.How to develop a database application

The steps involved in database application development any relational data base pplication there are always the same basic steps to follow.Microsoft Access is a relation data base management system because all data is stored in an Access data base in the for of simple tables.Another name for a table is relation.

The steps of Access database design like this

- Database design
- Tables design
- Forms design
- Query design
- Report design
- Macro design
- Modüles design

2.3.Relational database

DBMS has established themselves as one of the primary means for data torage for information based systems ranging from large business applications to simpl pc based programs.However a relational database management system (RDBMS) is the sy em used to work with data management operations more than 15 years,and still improvir ,providing more sophisticated storage,retriaval systems.Relational database manageme t systems

provides organisations with ability to handle huge ammount of data and chan ing it into meaningful information.

2.4.The facilities of access

Microsoft Access is relational DBMS(Database Management System) with all ıe features necessary to develop and use a data base application.The facilities it offers can l ɔ found on most modern relational DBMSs and all versions of Access.

- Tables are where all the data is stored.They are usually linked by relati ıships.
- Queries are the way you extract data from the database
- Forms are the method used for input and display of database data.
- Reports are used to display nicely formatted data on paper.
- Macros are sets of simple commands that execute sequences (' database operations.
- Modules are used to stare general-purpose VB database program code.

2.5.Visual basic and Access

Microsoft Acess is the DBMS(Database Management System) VB and Access in developing data base applications is that for non-trivial database applications,VB ffers more flexibility to the developper then the VB comes with Access.Access database using VB program code and setting properties.

First method of linking VB forms to Access databases called the data conti l.The data control is a simple VB control that you drag on to a VB form to link it to y(ır choosen database.The data can be displayed and updated using tiedtext boxes,list b xes,combo boxes,and grids.

2.5.1.DAO(Data Access objects)

The DAO approach to database programming often requires more code ,bu like SQL compared to the Qury Design View,offers greater control to the database progr nmer over what's going on his/her application.

Data Access Objects are things like databases,recordsets,table and query defi itions, and fields,Rather than tying a recor set to a data control when we use DAO we sha allow our programs to create and manipulate recordsets.

2.5.2.ADO(Active X Data Objects)

The ADO programming is in principle very similar to DAO programming b t cointains some new commands.ADO is Microsoft's new approach to database program ing which aims to give the programmer a more consistent way of connecting to a bro l range of different types of data source.

2.6.The application of Access

MS Access is begin used as the development tool,and the application is going to b a single user application,which means its going to be instolled on one machine,this applica on however may be used by more than one user on many computers sharing the same ables by using simple advancements.

For a new database,after having specified the database name and path as above,yo will be confronted with the following window.

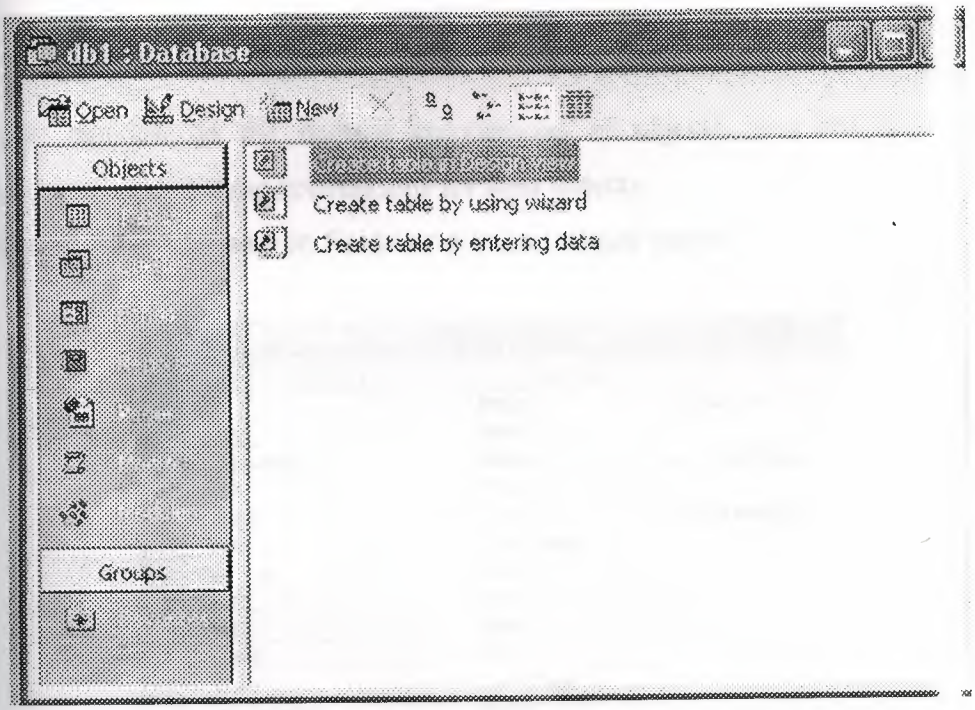


Figure1.1. The window of database

This window shows that there are notables in database yet.Click new button.

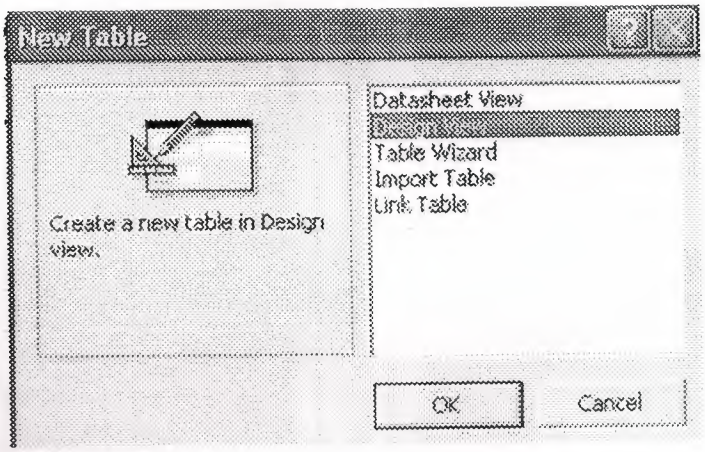


Figure 1.2. The window is type of table design

sellect the Design View by clicking on the listbox and then the OK button.Design View gives us more control over the design of our database than either the Table Wizard or the Datasheet view.Import Table is used to bring in data from an existing database and Link Table is database to an external table.

2.6.1. Tables Design

In my project's table designing with primary key. Guide Lines for making a database project.

The database consists of one tables;

Table1

Please pay attention on the naming conventions of objects, you are required to use appropriate names using these conventions for your objects.

TheTable one have got eleven fields one is its unique name

Table1 : Tablo

Alan Adı	Veri Türü	
id	Metin	numaralar
name	Metin	
explanation	Metin	urun aciklamasi
piece	Metin	
a_item	Metin	active madde
l_date	Tarih/Saat	
shelf_no	Metin	
price	Metin	
f_tel	Metin	
f_name	Metin	
b_date	Tarih/Saat	
p_explenation	Metin	

Figure 1.3 The Table1

CHAPTER3

MAIN PROGRAM

3.1. MAIN MENU

This is the main menu of the program. There is also some sub menus on tl top of the main menu. From the main menu we can go sub programs by using this sub menu There are also some buttons. They are used to go to the sub programs. They are providing cilities for users of the program. We can see all sub programs on the main menu.

Record button is used to go record part of the program. In the part we ent r madicine rrecord information.

Information button is used to go to Iformation part. Here we make rapor stock and son in the stock .

Seel button is used to show information such as number of madicine d piece of madicine and code of madicine

Esdeger button is used to keep information about the madicine and their accompaniment .

The form and codes of the main menu is following down.

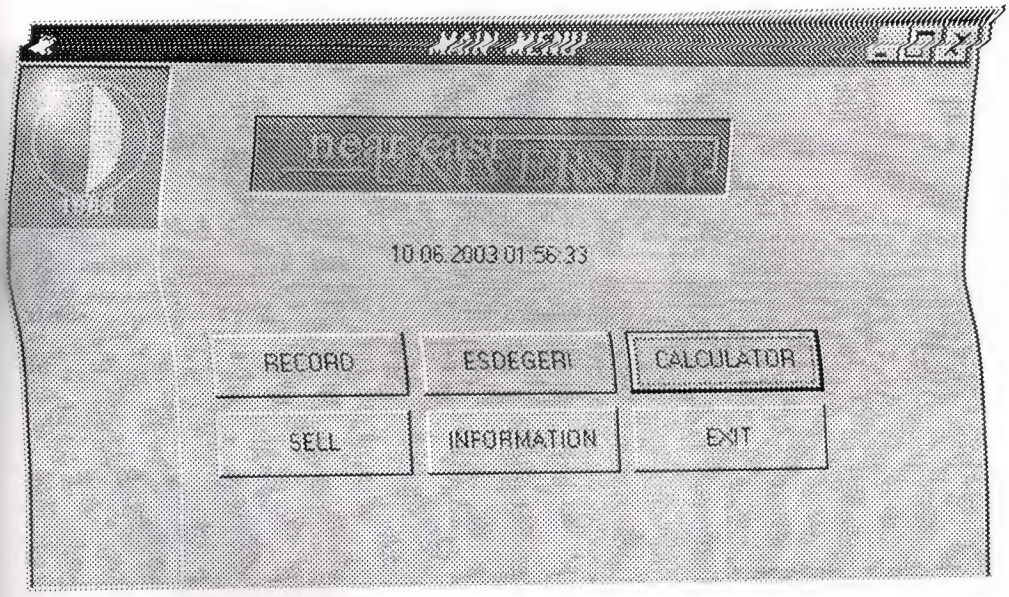


Figure 3.1. Main Menu

```
Private Sub Command1_Click()
Load Form2
```

```
Form2.Show
End Sub
Private Sub Command2_Click()
Load Form4
Form4.Show
Form1.Visible = False
End Sub
Private Sub Command3_Click()
Load Form3
Form3.Show
End Sub
Private Sub Command4_Click()
DataReport1.Show
Form1.Visible = True
End Sub
Private Sub Command5_Click(Index As Integer)
End
End Sub
Private Sub Command6_Click()
Load Form5
Form5.Show
Form1.Visible = False
End Sub
Private Sub Form_Initialize()
Form8.Show
Form9.Visible = True
Form1.Visible = False
Load Form8
Form8.Show
End Sub
Private Sub Form_Load()
Picture1.Align = center
End Sub
Private Sub Label1_Click()
```

```
Label1.Caption = Now()
End Sub
Private Sub Timer1_Timer()
Form1.Caption = Right(Form1.Caption, (Len(Form1.Caption) - 1)) + Left(Form1.  aption, 1)
Label1.Caption = Now()
End Sub
```

.2. THE PASSWORD SECREEN

built In this secreen Who can enter the program. The progaram user have a pswo l and user ıame. With pasword and user name the user can use the program. The pasword se een is on ıctive when the program start to run.

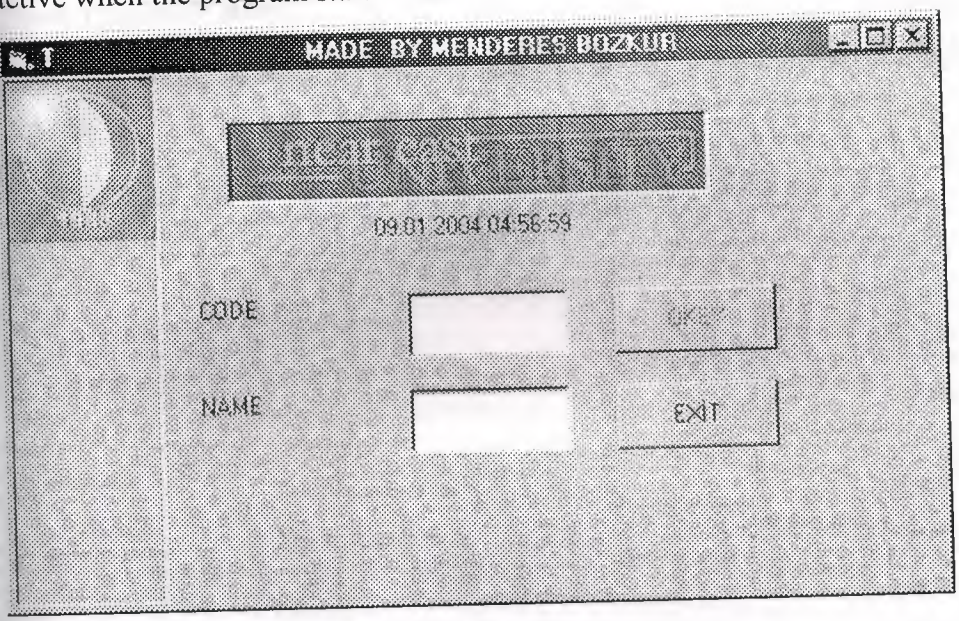


Figure 3.2 Pasword Writing

```
Private Sub Command1_Click()
Form9.Visible = False
Form1.Visible = True
End Sub
Private Sub Command2_Click()
End
End Sub
Private Sub Text1_Change()
If Text1.Text <> "" And Text1.Text = "yasin" And Text2.Text <> "" Then
Command1.Enabled = True
Command1.SetFocus
End If
End Sub
Private Sub Text2_Change()
If Text1.Text <> "" And Text2.Text <> "" And Text1.Text = "991267" Then
```

```
Command1.Enabled = True
Command1.SetFocus
End If
End Sub
Private Sub Timer1_Timer()
Label3.Caption = Now()
Form9.Caption = Right(Form9.Caption, (Len(Form9.Caption) - 1)) + Left(Form9.  aption, 1)
End Sub
```

3.3. UNIT EXPIRE DATE

The secreen About Product which is expire date. When main program run expire d te secreen will be on active

directly. Al the information about madicine are provided with this form and there : e 90 day for exipire date which are shown product.

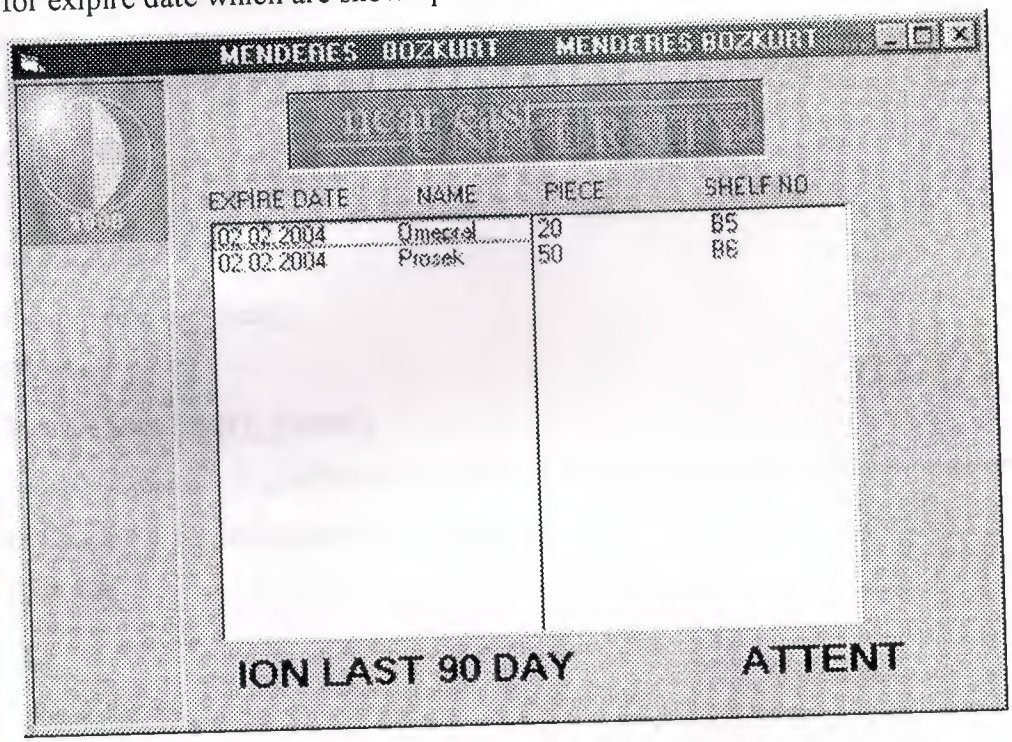


Figure 3.2. Product Selection

```
Private Sub Form_Initialize()
Dim db As Database
Dim tb As Recordset
Set db = OpenDatabase("C:\WINDOWS\Desktop\vt1.mdb")
Set tb = db.OpenRecordset("tablo1")
While Not tb.EOF
x = tb.Fields("l_date")
m1 = Val(Month(x))
m2 = Val(Month(Date))
Y1 = Val(Year(x))
Y2 = Val(Year(Date))
```

```
If Y1 = Y2 And m1 >= m2 Then
m = (m1 - m2) * 31
If m < 90 Then
List1.AddItem Str(tb.Fields("l_date")) + "          " + tb.Fields("name")
List2.AddItem tb.Fields("piece") + "                    " + tb.Fields("shelf_no")
End If
End If
tb.MoveNext
Wend
End Sub
Private Sub ta_Click()
End Sub
Private Sub Timer1_Timer()
Form8.Caption = Right(Form8.Caption, (Len(Form8.Caption) - 1)) + Left(Form8.  aption, 1)
ta.Caption = Right(ta.Caption, (Len(ta.Caption) - 1)) + Left(ta.Caption, 1)
End Sub
```

3.4. UNIT RECORD OF MADICINE

Section of showing the type of record. You can select to type of record with using ı cord of madicine secreen :the type of record are searching, deleting, adding, finding, editi ıng and also you can see the report of stock with this screen.

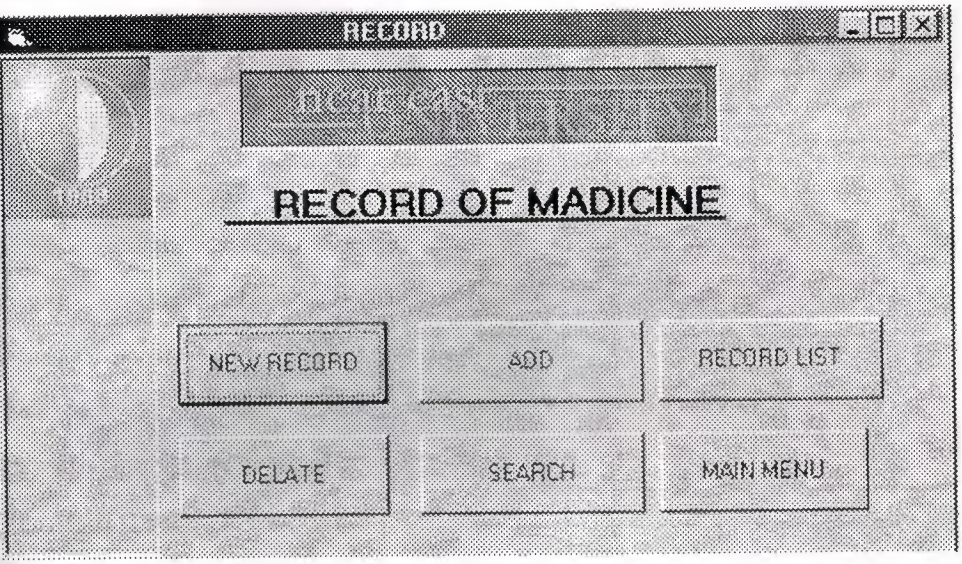


Figure 3.4. Record Of Madicine

```
Private Sub Command1_Click(Index As Integer)
Load Form6
Form6.Show
Form2.Visible = False
End Sub
Private Sub Command2_Click()
Load Form6
Form6.Show
Form2.Visible = False
End Sub
Private Sub Command3_Click()
Load Form6
Form6.Show
Form2.Visible = False
End Sub
Private Sub Command4_Click()
```

```
Load Form6
Form6.Show
Form2.Visible = False
End Sub
Private Sub Command5_Click()
DataReport1.Show
End Sub
Private Sub Command6_Click()
Form2.Visible = False
Form1.Visible = True
Form1.Show
End Sub
Private Sub Form_Load()
Form1.Visible = False
End Sub
Private Sub RE_Click()
End Sub
Private Sub Timer1_Timer()
Form2.Caption = Right(Form2.Caption, (Len(Form2.Caption) - 1)) + Left(Form2.  aption, 1)
End Sub
```

.5. UNITE RECORDE

'he recording will accour with code. If oyu want to write data to database You mu t enter the :ode of product. May be you forget to enter product code the software will situmu te to user vith message box. Also when you record some data to database some of the recor ng name s vary important. Forexample code and expire date this are vary important object the program. Because of searching accoure with code and expire date. Also the name accompaniment is vary important because of if the product there is not on the stoc the accompaniment will came to screen. That is the reson you must enter the accompa ment of product.

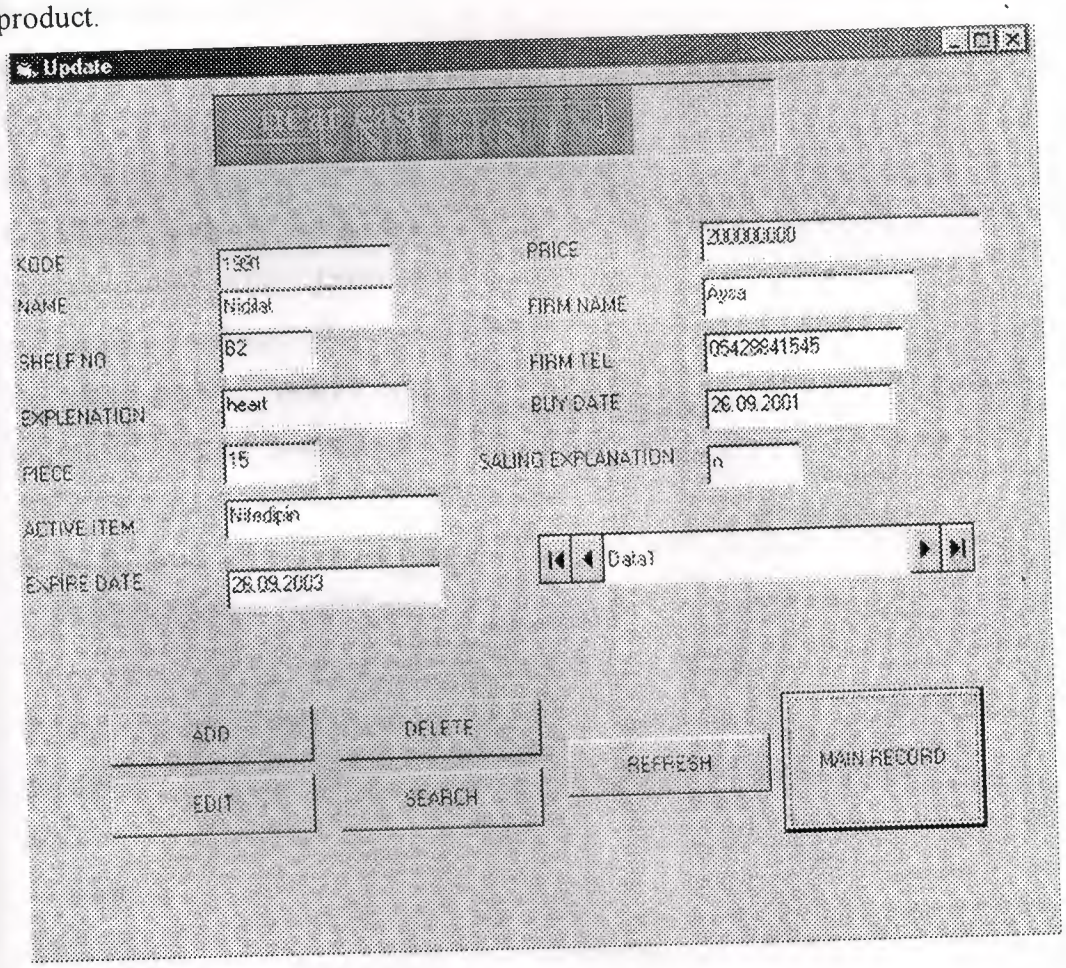


Figure 3.5. Record

```
Private Sub Command1_Click()
Dim db As Database
Dim tb As Recordset
Set db = OpenDatabase("C:\WINDOWS\Desktop\vt1.mdb")
Set tb = db.OpenRecordset("tablo1")
yasin:
```

```
MsgBox "YOU MUST ENTER ALL COMPONENT"
a = MsgBox("DO YOU WANT TO RECORD", vbYesNoCancel, "READ CREFU LY")
If a = 6 Then GoTo kayit
If a = 7 Then GoTo fin
If a = 2 Then GoTo fin2
kayit:
tb.AddNew
If Text5.Text = "" Then GoTo yasin
tb.Fields("no") = Text1.Text
tb.Fields("explanation") = Text2.Text
tb.Fields("piece") = Text3.Text
tb.Fields("a_item") = Text4.Text
tb.Fields("l_date") = Text5.Text
tb.Fields("shelf_no") = Text6.Text
tb.Fields("p_explenation") = Text11.Text
tb.Fields("name") = Text12.Text
tb.Fields("f_tel") = Text8.Text
tb.Fields("f_name") = Text9.Text
tb.Fields("b_date") = Text10.Text
tb.Fields("price") = Text7.Text
Move Last
If Text1.Text = "" Then GoTo yasin
tb.Update
tb.Close
db.Close
fin:
GoTo son
fin2:
Text1.Text = ""
Text2.Text = ""
Text3.Text = ""
Text4.Text = ""
Text5.Text = ""
Text6.Text = ""
```

```
Text7.Text = ""
Text8.Text = ""
Text9.Text = ""
Text10.Text = ""
Text11.Text = ""
Text12.Text = ""
son:
End Sub
Private Sub Command2_Click()
Dim db As Database
Dim tb As Recordset
Dim s  As String
Dim c As Integer
Set db = OpenDatabase("C:\WINDOWS\Desktop\vt1.mdb")
Set tb = db.OpenRecordset("tablo1")
s = Text1.Text
tb.Index = "primarykey"
tb.Seek "=", s
While Not tb.EOF
If s = tb.Fields("no") Then
a = MsgBox("do you want to delate", vbYesNo, "delate screen")
If a = vbYes Then GoTo sil
If a = vbNo Then GoTo atla
sil:
tb.Delete
Text1.Text = ""
Text2.Text = ""
Text3.Text = ""
Text4.Text = ""
Text5.Text = ""
Text6.Text = ""
Text7.Text = ""
Text8.Text = ""
Text9.Text = ""
```

```
Text10.Text = ""
Text11.Text = ""
Text12.Text = ""
c = c + 1
atla:
End If
tb.MoveNext
Wend
If c <> 0 Then
MsgBox "is deleted by you"
End If
End Sub
Private Sub Command3_Click()
Form6.Visible = False
Form2.Visible = True
Form2.Show
End Sub
Private Sub Command4_Click()
Text1.Text = ""
Text2.Text = ""
Text3.Text = ""
Text4.Text = ""
Text5.Text = ""
Text6.Text = ""
Text7.Text = ""
Text8.Text = ""
Text9.Text = ""
Text10.Text = ""
Text11.Text = ""
Text12.Text = ""
End Sub
Private Sub Command5_Click()
Dim db As Database
Dim tb As Recordset
```

```
Dim s  As String
Dim c As Integer
Set db = OpenDatabase("C:\WINDOWS\Desktop\vt1.mdb")
Set tb = db.OpenRecordset("tablo1")
s = Text1.Text
tb.Index = "primarykey"
tb.Seek "=", s
While Not tb.EOF
If s = tb.Fields("no") Then
a = MsgBox("ıs ıt thıs record", vbYesNo, "search screen")
If a = vbYes Then GoTo sil
If a = vbNo Then GoTo atla
sil:
Text1.Text = tb.Fields("no")
 Text2.Text = tb.Fields("explanation")
 Text3.Text = tb.Fields("piece")
 Text4.Text = tb.Fields("a_item")
 Text5.Text = tb.Fields("l_date")
 Text6.Text = tb.Fields("shelf_no")
 Text11.Text = tb.Fields("p_explenation")
 Text12.Text = tb.Fields("name")
 Text8.Text = tb.Fields("f_tel")
 Text9.Text = tb.Fields("f_name")
 Text10.Text = tb.Fields("b_date")
 Text7.Text = tb.Fields("price")
c = c + 1
atla:
End If
tb.MoveNext
 Wend
 If c <> 0 Then
 MsgBox "ıs found by searcher"
 End If
 End Sub
```

```
Private Sub Command6_Click()
Dim db As Database
Dim tb As Recordset
Dim s  As String
Dim c As Integer
Set db = OpenDatabase("C:\WINDOWS\Desktop\vt1.mdb")
Set tb = db.OpenRecordset("tablo1")
s = Text1.Text
tb.Index = "primarykey"
tb.Seek "=", s
While Not tb.EOF
If s = tb.Fields("no") Then
a = MsgBox("do you want to change", vbYesNo, "changing screen")
If a = vbYes Then GoTo sil
If a = vbNo Then GoTo atla
sil:
tb.Edit
tb.Fields("no") = Text1.Text
tb.Fields("explanation") = Text2.Text
tb.Fields("piece") = Text3.Text
tb.Fields("a_item") = Text4.Text
tb.Fields("l_date") = Text5.Text
tb.Fields("shelf_no") = Text6.Text
tb.Fields("p_explenation") = Text11.Text
tb.Fields("name") = Text12.Text
tb.Fields("f_tel") = Text8.Text
tb.Fields("f_name") = Text9.Text
tb.Fields("b_date") = Text10.Text
tb.Fields("price") = Text7.Text
tb.Update
c = c + 1
atla:
End If
tb.MoveNext
```

```
end
c <> 0 Then
sgBox "is changed by you"
d If
d Sub
vate Sub Form_Load()
d Sub
```

3.6. UNITE ACCOMPANIMENT

In this screen you can learn any madicine that you want thay are name, shelf no, accompaniment, how many piece so on . If your madicine is there the software wil stimulate to you like there is your searching madicine in the database with messagebox. İf th re is not any pruduct that you want also the soft ware will stimulate to you like, there is not ny item like aspirin please buy on the list. Also the madicine on the list are the same madic ıe that you want but just the company name is different. İf you interest the chamical line the p ıduct are the same product on list and you want.

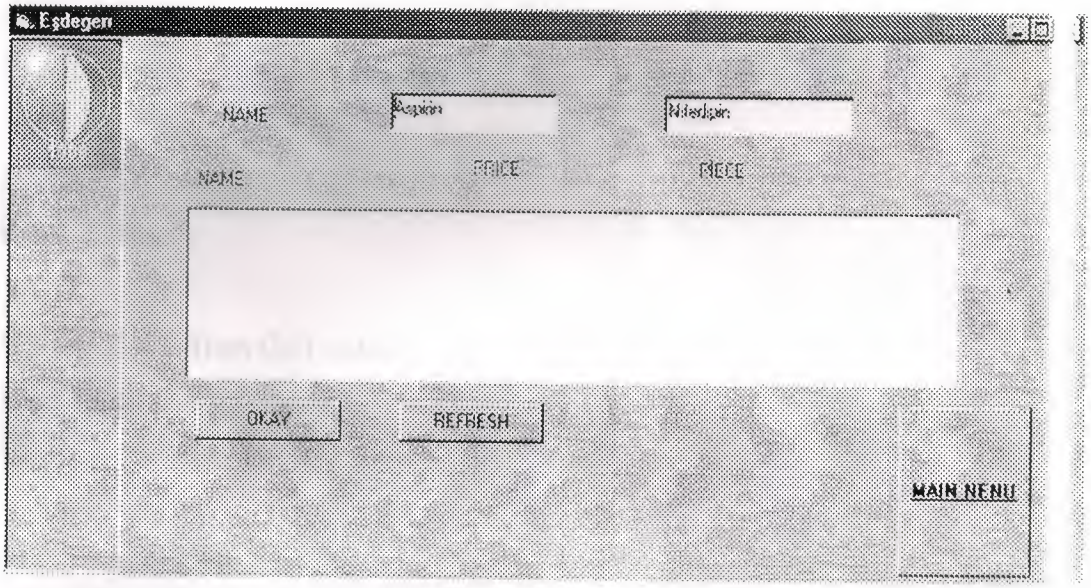


Figure 3.6. Equvalence

```
Private Sub Command1_Click()
Form3.Visible = False
Form1.Visible = True
Form1.Show
End Sub
Private Sub Command2_Click()
Dim db As Database
Dim tb As Recordset
Dim s, sd, a1, a2, a3, a4, a5 As String
Dim a11, a21, a31, a41, a51 As String
Dim c As Integer
Set db = OpenDatabase("C:\WINDOWS\Desktop\vt1.mdb")
Set tb = db.OpenRecordset("Tablo1")
```

```
s = Text1.Text
While Not tb.EOF
If s = tb.Fields("name") Then
 a5 = tb.Fields("no")
 a3 = tb.Fields("piece")
 a4 = tb.Fields("a_item")
 a1 = tb.Fields("name")
 a2 = tb.Fields("price")
take = a4
List1.AddItem tb.Fields("name") + "    " + "   " + a2 + "    " + a3 + "   " + a4 + "   "  a5
c = c + 1
End If
tb.MoveNext
Wend
If Val(a3) > 0 Then GoTo atla
List1.Clear
MsgBox "THERE IS NOT " + s
tb.MoveFirst
While Not tb.EOF
If take = tb.Fields("a_item") Then
 If s = tb.Fields("name") Then GoTo unwrite
 a51 = tb.Fields("no")
 a31 = tb.Fields("piece")
 a41 = tb.Fields("a_item")
 a11 = tb.Fields("name")
 a21 = tb.Fields("price")
List1.AddItem tb.Fields("name") + "   " + a21 + "   " + a31 + "    " + a41 + "    " +  51
unwrite:
End If
tb.MoveNext
Wend
atla:
If c = 0 Then MsgBox "aaaaaaa"
End Sub
```

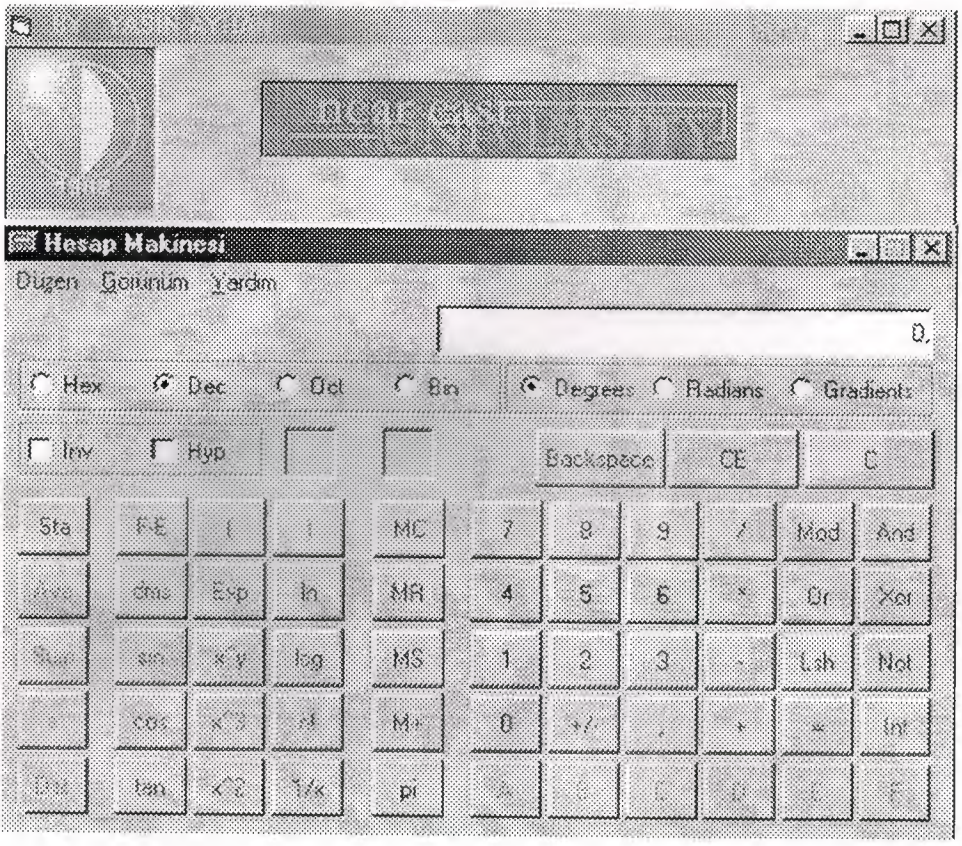
```
Private Sub Command3_Click()
List1.Clear
Text1.SetFocus
Text1.Text = ""
Text2.Text = ""
End Sub
Private Sub Form_Load()
Form1.Visible = False
End Sub
Private Sub Command1_Click()
Form3.Visible = False
Form1.Visible = True
Form1.Show
End Sub
Private Sub Command2_Click()
Dim db As Database
Dim tb As Recordset
Dim s, sd, a1, a2, a3, a4, a5 As String
Dim a11, a21, a31, a41, a51 As String
Dim c As Integer
Set db = OpenDatabase("C:\WINDOWS\Desktop\vt1.mdb")
Set tb = db.OpenRecordset("Tablo1")
s = Text1.Text
While Not tb.EOF
If s = tb.Fields("name") Then
 a5 = tb.Fields("no")
 a3 = tb.Fields("piece")
 a4 = tb.Fields("a_item")
 a1 = tb.Fields("name")
 a2 = tb.Fields("price")
take = a4
List1.AddItem tb.Fields("name") + "    " + "   " + a2 + "     " + a3 + "   " + a4 + "   " + a5
c = c + 1
End If
```

```
tb.MoveNext
Wend
If Val(a3) > 0 Then GoTo atla
List1.Clear

MsgBox "THERE IS NOT " + s
tb.MoveFirst
While Not tb.EOF
If take = tb.Fields("a_item") Then
 If s = tb.Fields("name") Then GoTo unwrite
  a51 = tb.Fields("no")
  a31 = tb.Fields("piece")
  a41 = tb.Fields("a_item")
  a11 = tb.Fields("name")
  a21 = tb.Fields("price")
List1.AddItem tb.Fields("name") + "   " + a21 + "   " + a31 + "    " + a41 + "    " +  51
unwrite:
End If
tb.MoveNext
Wend
atla:
If c = 0 Then MsgBox "aaaaaaa"
End Sub
Private Sub Command3_Click()
List1.Clear
Text1.SetFocus
Text1.Text = ""
Text2.Text = ""
End Sub
Private Sub Form_Load()
Form1.Visible = False
End Sub
```

3.7. UNIT CALCULATOR

In this screen there is clasic calculator and sciectific calculator. I made a map for s entific calculator to windows.




```
Dim a As Integer
Dim c As Integer
Private Sub Command1_Click()
c = 1
a = Val(Text1.Text)
Text1.Text = ""
Text1.SetFocus
End Sub
Private Sub Command2_Click()
c = 2
a = Val(Text1.Text)
Text1.Text = ""
Text1.SetFocus
End Sub
```

```
Text1.Text = ""
End Sub
Private Sub Timer1_Timer()
Form5.Caption = Right(Form5.Caption, (Len(Form5.Caption) - 1)) + Left(Form5.  aption, 1)
End Sub
```

3.8. UNIT SALLING

I think this screen is so important screen because of you sale your madicine from y ur stock. Esy to use this scren. After salling the madicine the madicine will dicrease from th stock.

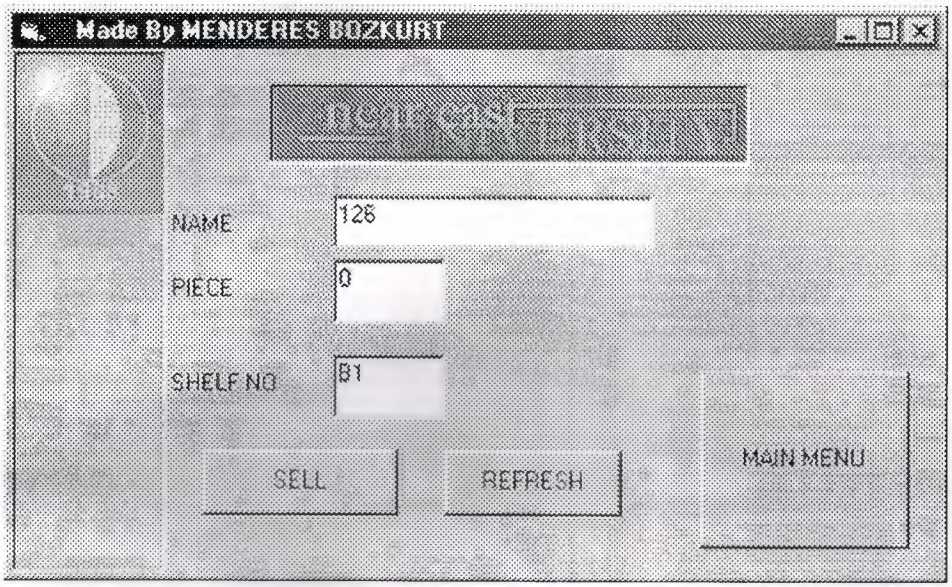


Figure 3.8. salling

```
Private Sub Command1_Click()
Form4.Visible = False
Form1.Visible = True
Form1.Show
End Sub
Private Sub Command2_Click()
Dim db As Database
Dim tb As Recordset
Dim s  As String
Dim c As Integer
Set db = OpenDatabase("C:\WINDOWS\Desktop\vt1.mdb")
Set tb = db.OpenRecordset("tablo1")
s = Text1.Text
tb.Index = "primarykey"
tb.Seek "=", s
If Val(tb.Fields("piece")) = 0 Or Val(tb.Fields("piece")) < Val(Text2.Text) Then (  )To atla
While Not tb.EOF
If s = tb.Fields("no") Then
```

```
k = Int(tb.Fields("piece")) - Int(Text2.Text)
a = MsgBox("did you sell", vbYesNo, "changing screen")
If a = vbYes Then GoTo sil
If a = vbNo Then GoTo atla
sil:
tb.Edit
tb.Fields("piece") = Str$(k)
tb.Update
c = c + 1
End If
tb.MoveNext
Wend
If c <> 0 Then
MsgBox "It is salled "
Else
atla:
MsgBox "DEMAND IS GRATER THEN STOCK "
MsgBox "THERE ARE" + "" + tb.Fields("piece")
Text3.Text = tb.Fields("shelf_no")
End If
End Sub
Private Sub Command3_Click()
Text1.Text = ""
Text2.Text = ""
Text3.Text = ""
End Sub
Private Sub Form_Load()
Form1.Visible = False
End Sub
Private Sub Timer1_Timer()
Form4.Caption = Right(Form4.Caption, (Len(Form4.Caption) - 1)) + Left(Form4.  aption, 1)
End Sub
```

3.9. UNIT REPORT

The report of all stock in the firm. We can get any extra information obout madicin

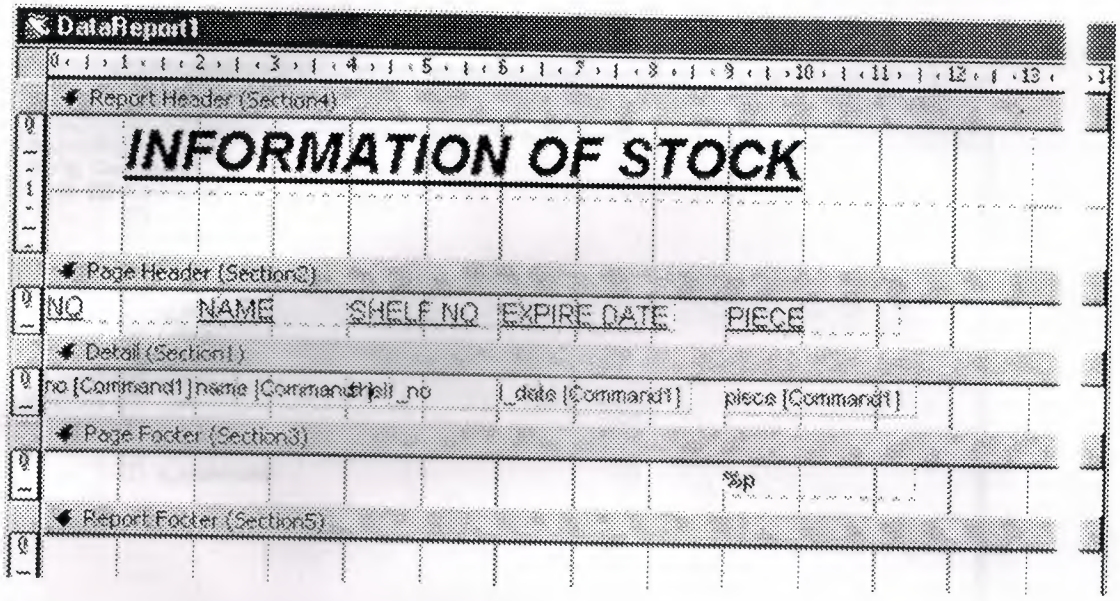


Figure 3.8. Information of Stock

3.10. UNIT DATA ENVIRONMENT

Also we use data environment for creating the report

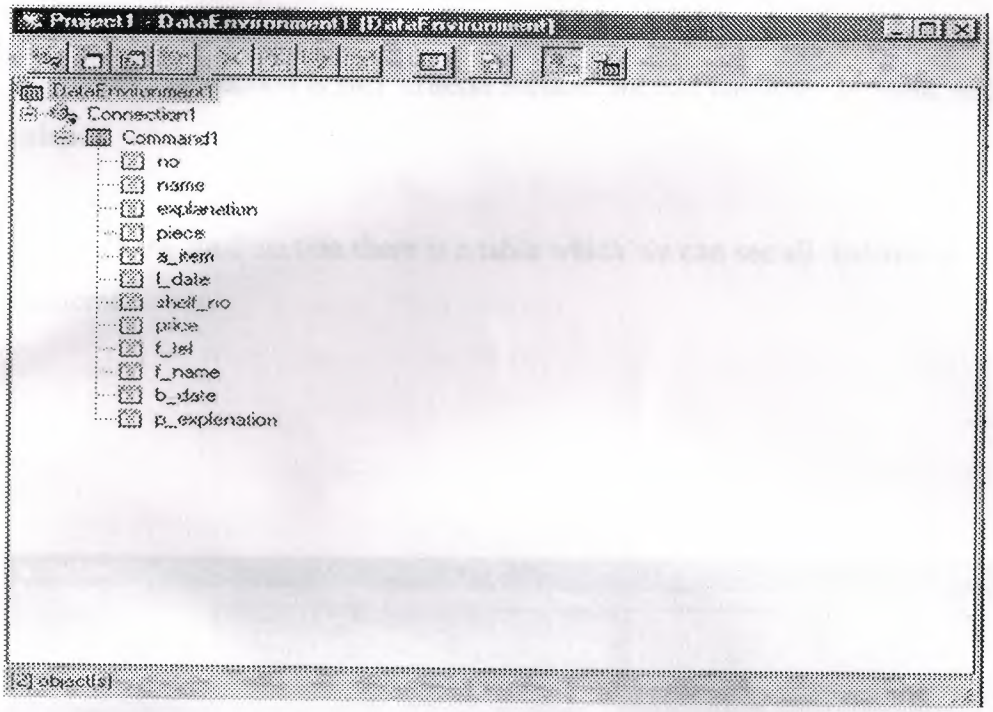


Figure 3.9. Data environment

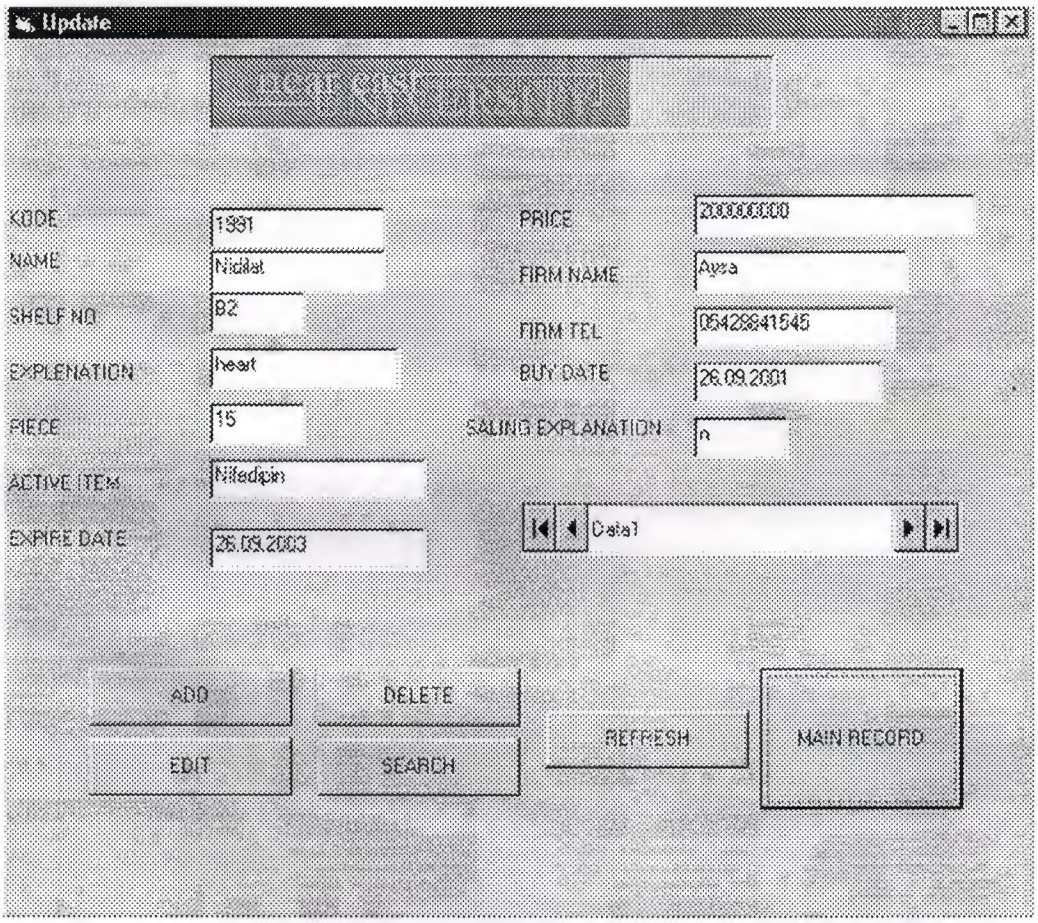
BALANCES

In this part I prepared code balances accompaniment balances and xpire date balance. In the balances there are three section.

First section is search criteria. Here we can search code and madicine t ıt we want to find it. We search madicine by use the mad_ no ,m_name , balance, and balan ; type. We search madicine by use the code, accompaniment, and expire date

Second section is sort criteria section .we sort madicine by code, accompa iment, and balance.

In the final section there is a table which we can see all balance informati ı about the madicine.



CONCLUSSION

Visual Basic is an easy program to grasp. This cause is why I have decide to use this program.

Visual Basic is a Microsoft Windows programming Language. Visual Basic is a distinctly different language providing powerfull features such as graphical use interfaces, even handling, access to the Win32 API, object-oriented features, error handling structured programming, and much more.

In this project I built madicine database program. It is easy to use and It can be use most kind of drugstore. I used Visual Basic for write this program and I used Microsoft Access database for keep all my databases.

In this study our main aim to put accross is that this program can be perated by some one who has never used it before.

In this program there is also menus to make your writting much simpler, I containing windows menus and also afacility to prepare reports.

REFERANCES

1-) Ihsan Karagülle ; Zeydin Pala(1999). Microsoft Visual Basic 6 . Istanbul. Türkmen press.

2-) Prof. Dr. Mithat Uysal (1999). Development Of The Software with \ sual Basic 6.0. Istanbul. Beta Press.

3-) Ihsan Karagülle ; Zeydin Pala (1999). Microsoft Visual Basic 6.0 Pro. Istanbul. Türkmen Press.

4-) Hilal Drugstore, Girne

5-) Macit Pharmacy, Lefkosa