NEAR EAST UNIVERSITY

Faculty of Engineering

Department of Computer Engineering

AUTOMATION OF PHARMACY

Graduation Project
COM - 400

Student: Murat Everekli

Supervisor: Assist. Professor Dr. Elbrus Imanov

Nicosia – 2008

ACKNOWLEDGEMENTS

"First I would like to thank my supervisor Yard.Doç.Dr Imanov Elbrus for his invaluable experience and motivation in my work and myself over the course of this Graduation Project. Without his encouragement and direction, this work would not have been completed and I am really thankful to my teacher.

Second, I would like to express my gratitude to Near East University for the scholarship that made the work possible.

Third, I thank my family for their constant encouragement and support during the preparation of this Project. I am nothing without their support. I shall never forget their sacrifices for my education so that I can enjoy my successful life as they are expecting. Especially My father Dt. Faruk Everekli thanks so much to give me all of his life and health area experience and knowledge. I will never forget my father, my mother Fatma Everekli, and my sisters Sezin Everekli.

I want to thank to my friends Mehmet Kale, Cevdet Dikiciler, Berna Küçük, and Esra Okuyan for their help, they get tired with me, and they helped me everytime whenever I want. I thank them with my all

Finally, My best friend Ecz. Ayşen Aydın thanks so much to her invaluable affection and giving me all information about pharmacy system and medicines. She may get peaceful in all her life."

ABSTRACT

The Health is very important in our life because that is the every thing when think logical. So those Health sectors always develop with using technology. Medicine sector is developed parallel with Health sector. Everyday, companies produce new medicines. Those medicines to transport patients is using mediator, Pharmacy. Pharmacy also includes more modern services related to patient and providing drug information.

Pharmacists are the experts in drug discovery, development, preparation and usage of medicines but some problems about administer, control and automation of pharmacy. Because pharmacists have so much medicines so much formula and so much patients in pharmacy.

The aim of this project is the easy and practical controlling, administering to sell and stock. The main problem's for pharmacist is control the selling and stock speedly. My main goal is wanted to design my Project to solving the problem in this project.

TABLE OF CONTENTS

ACKNOWLEDGMENT i
ABSTRACT ii
TABLE OF CONTENTS iii
LIST OF ABBREVIATIONS iv
INTRODUCTION 1
CHAPTER ONE: DELPHI 2
1.1. Introduction to Delphi 2
1.2. What is Delphi? 4
1.3. What kind of programming can you do with Delphi? 5
1.4. Versions are there and how do they differ? 6
1.5. What's new in Delphi 7 8
1.5.1. Code Insight changes 9
1.5.2. Debugger changes 9
1.5.3. Miscellaneous improvements 10
1.5.4. Attachments 10
1.5.5. Type support 11
1.5.6. Other enhancements 11
1.5.7. Database technology changes (Enterprise editions) 12
1.5.8. Component library changes 12
1.5.9. New unit and components 12
1.6. Some Knowledge About Delphi 13
1.6.1. Example: Try first Delphi Program 14
1.6.2. Delphi Style 16
1.7. How Delphi Helps You Define Patterns 17
1.7.1. Delphi examples of design patterns 18
1.7.2. Pattern: Singleton 19
1.7.2.1. Definition 19
1.7.2.2. Applications in Delphi 19
1.7.2.3. Implementation example 19
1.7.3. Pattern: Adapter 19
1.7.3.1. Definition 19
1.7.3.2. Applications in Delphi 20
1.7.3.3. Implementation example 20
1.7.4. Pattern: Template Method 20
1.7.4.1. Definition 20
1.7.4.2. Applications in Delphi 20
1.7.4.3. A typical example of abstraction is the Tgraphic 21
1.7.4.4. Implementation example 21
1.7.5. Pattern: Builder 21
1.7.5.1. Definition 21
1.7.5.2. Applications in Delphi 22
1.7.5.3. Implementation example 22
1.5.6. Pattern: Abstract Factory 22
1.7.6.1. Definition 22
1.7.6.2. Applications in Delphi 23
1.7.6.3. Implementation example 23

1.7.7. Pattern: Factor Method 23
1.7.7.1. Definition 23
1.7.7.2. Application in Delphi 23
1.7.7.3. Implementation example 24
1.8. Key Elements of Delphi Class Definitions 24
1.8.1. Unit Structure 24
1.8.2. Class Interfaces 24
1.8.3. Properties 24
1.8.4. Inheritance 25
1.8.5. Abstract Methods 27
1.8.6. Messages 27
1.8.7. Events 27
1.8.8. Constructors and Destructors 28
1.9. The VCL to Applications Developers 28
1.9.1. The VCL to Component Writers 29
1.9.2. The VCL is made up of Components 29
1.9.3. Component Types, Structure and VCL hierarchy 30
1.9.4. Component Types 30
1.9.4.1. Standard components 30
1.9.4.2. Custom components 31
1.9.4.3. Graphical components 32
1.9.4.4. Non-visual components 32
1.9.4.5. Structure of a component 32
1.9.4.6. Component properties 32
1.10. Properties Provide Access to Internal Storage Fields 33
1.10.1. Property-access Method 34
1.10.2. Types of Properties 35
1.10.3. Methods 36
1.10.4. Events 36
1.10.5. Containership 37
1.10.6. Ownership 38
1.10.7. Parenthood 38
CHAPTER TWO: DATABASE **40**
2.1. Demerits to Absence of Database 40
2.2. Merits of Database 41
2.3. Database Design 41
2.4. Database Models 42
2.4.1. Flat Model 43
2.4.2. Network Model 43
2.4.3. Relational Model 43
2.4.3.1. Why we use a relational database design 44
2.5. Relationship between Tables 45
2.5.1. One-to-one Relationships 45
2.5.2. One-to-many Relationship 45
2.6. Data Modeling 46
2.6.1. Database Normalization 46
2.6.2. Primary Key 46
2.6.3. Foreign Key 47
2.6.4. Compound Key 48

CHAPTER TREE: MY PROJECT **49**
3.1. My Database and Tables 49
3.2. My Automation of Pharmacy Program 51
3.2.1. Login Form 51
3.2.2. Main Form 52
3.2.3. New Patient Form 53
3.2.4. New Doctor Form 54
3.2.5. New Drug Form 55
3.2.6. Stock Entry and Stock Exit Forms 56
3.2.7. Selling with SSK Formulas and with Formulas Forms 57
3.2.8. Selling without Formulas Forms 58
3.2.9. Account Control Forms 59
3.2.10. Stock Controlling System Forms 60
3.2.11. Selling Controlling System Forms 61
3.2.12. Tools Forms 63
3.2.13. About Forms 64
CONCLUSION **65**
REFERENCES **66**
APPENDIX **67**

LIST OF ABBREVIATIONS

SSK	Sosyal Sigortalar Kurumu
UNFO	Unformal Prescription
FORM	Formal Prescription
Bağ-Kur	Employer

INTRODUCTION

In the near future, the technology is developed a lot and started to use by anyone in the world no matter who he/she is. Because of the technology is entered to every platform of our life person needed to combine both software and hardware. Without software the machines are nothing. They need software to operate.

The automation is also became a part of our lives. The people operate with automation systems in everywhere. This Automation is used to keep the information about the receiving, coming and going documents. We will control the stock and operate it with using automation system. I used Borland Delphi 7 in my project, because I find it easy and I liked its coding system.

In this project pharmacists can keep patient information, doctor information and medicines information. He /she can control stock information and selling information in specific time period who can be print out this information in my project. The software can be used at every pharmacy easily. Firstly Patient must enter the information about the medicines. Barcode system is the best primary keys for all things in database. Pharmacists can control everything with using this system. I adapted barcode system in this project.

The objective of this project is to develop pharmacy automation. The project consists of introduction, three chapters and conclusion.

Chapter One describes the information about the Delphi 7. In this chapter have the Delphi 7's properties, components and some examples.

Chapter Two describes the Database. In this chapter I described the Databasing system. How to create table, how to controlling this table and how to do database normalization.

The last chapter I explained my project Database. Than I presents my program. I advertised the parts of program. How to using this unit easily and explanation of the program followed by the Appendices.

CHAPTER 1

DELPHI

1.1 INTRODUCTION TO DELPHI

The name "Delphi" was never a term with which either Olaf Helmer or Norman Dalkey (the founders of the method) were particular happy. Since many of the early Delphi studies focused on utilizing the technique to make forecasts of future occurrences, the name was first applied by some others at Rand as a joke. However, the name stuck. The resulting image of a priestess, sitting on a stool over a crack in the earth, inhaling sulfur fumes, and making vague and jumbled statements that could be interpreted in many different ways, did not exactly inspire confidence in the method.

The straightforward nature of utilizing an iterative survey to gather information "sounds" so easy to do that many people have done "one" Delphi, but never a second. Since the name gives no obvious insight into the method and since the number of unsuccessful Delphi studies probably exceeds the successful ones, there has been a long history of diverse definitions and opinions about the method. Some of these misconceptions are expressed in statements such as the following that one finds in the literature:

It is a method for predicting future events.

It is a method for generating a quick consensus by a group.

It is the use of a survey to collect information.

It is the use of anonymity on the part of the participants.

It is the use of voting to reduce the need for long discussions.

It is a method for quantifying human judgement in a group setting.

Some of these statements are sometimes true; a few (e.g. consensus) are actually contrary to the purpose of a Delphi. Delphi is a communication structure aimed at producing detailed critical examination and discussion, not at forcing a quick compromise. Certainly quantification is a property, but only to serve the goal of quickly identifying agreement and disagreement in order to focus attention. It is often very common, even today, for people to come to a view of the Delphi method that reflects a particular application with which they are familiar. In 1975 Linstone and Turoff proposed a view of the Delphi method that they felt best summarized both the technique and its objective:

"Delphi may be characterized as a method for structuring a group communication process, so that the process is effective in allowing a group of individuals, as a whole, to deal with complex problems." The essence of Delphi is structuring of the group communication process. Given that there had been much earlier work on how to facilitate and structure face-to-face meetings, the other important distinction was that Delphi was commonly applied utilizing a paper and pencil communication process among groups in which the members were dispersed in space and time. Also, Delphis were commonly applied to groups of a size (30 to 100 individuals) that could not function well in a face-to-face environment, even if they could find a time when they all could get together.

Additional opportunity has been added by the introduction of Computer Mediated Communication Systems (Hiltz and Turoff, 1978; Rice and Associates, 1984; Turoff, 1989; Turoff, 1991). These are computer systems that support group communications in either a synchronous (Group Decision Support Systems, Desanctis et. al., 1987) or an asynchronous manner (Computer Conferencing). Techniques that were developed and refined in the evolution of the Delphi Method (e.g. anonymity, voting) have been incorporated as basic facilities or tools in many of these computer based systems. As a result, any of these systems can be used to carry out some form of a Delphi process or Nominal Group Technique (Delbecq, et. al., 1975).

The result, however, is not merely confusion due to different names to describe the same things; but a basic lack of knowledge by many people working in these areas as to what was learned in the studies of the Delphi Method about how to properly employ these techniques and their impact on the communication process. There seems to be a great deal of "rediscovery" and repeating of earlier misconceptions and difficulties.

Given this situation, the primary objective of this chapter is to review the specific properties and methods employed in the design and execution of Delphi Exercises and to examine how they may best be translated into a computer based environment.

1.2 WHAT IS DELPHI?

Delphi is an object oriented, component based, visual, rapid development environment for event driven Windows applications, based on the Pascal language. Unlike other popular competing Rapid Application Development (RAD) tools, Delphi compiles the code you write and produces really tight, natively executable code for the target platform. In fact the most recent versions of Delphi optimise the compiled code and the resulting executables are as efficient as those compiled with any other compiler currently on the market.The term "visual" describes Delphi very well. All of the user interface development is conducted in a What You See Is What You Get environment (WYSIWYG), which means you can create polished, user friendly interfaces in a very short time, or prototype whole applications in a few hours.

Delphi is, in effect, the latest in a long and distinguished line of Pascal compilers (the previous versions of which went by the name "Turbo Pascal") from the company formerly known as Borland, now known as Inprise. In common with the Turbo Pascal compilers that preceded it, Delphi is not just a compiler, but a complete development environment. Some of the facilities that are included in the "Integrated Development Environment" (IDE) are listed below:

- A syntax sensitive program file editor
- A rapid optimising compiler
- Built in debugging /tracing facilities
- A visual interface developer
- Syntax sensitive help files
- Database creation and editing tools
- Image/Icon/Cursor creation / editing tools

• Version Control CASE tools What's more, the development environment itself is extensible, and there are a number of add ins available to perform functions such as memory leak detection and profiling.

In short, Delphi includes just about everything you need to write applications that will run on an Intel platform under Windows, but if your target platform is a Silicon Graphics running IRIX, or a Sun Sparc running SOLARIS, or even a PC running LINUX, then you will need to look elsewhere for your development tools.

This specialisation on one platform and one operating system, makes Delphi a very strong tool. The code it generates runs very rapidly, and is very stable, once your own bugs have been ironed out!

1.3 WHAT KIND OF PROGRAMMING CAN YOU DO WITH DELPHI?

The simple answer is "more or less anything". Because the code is compiled, it runs quickly, and is therefore suitable for writing more or less any program that you would consider a candidate for the Windows operating system.

You probably won't be using it to write embedded systems for washing machines, toasters or fuel injection systems, but for more or less anything else, it can be used (and the chances are that probably someone somewhere has!)

Some projects to which Delphi is suited:

- Simple, single user database applications
- Intermediate multi-user database applications
- Large scale multi-tier, multi-user database applications
- Internet applications
- Graphics Applications
- Multimedia Applications
- Image processing/Image recognition
- Data analysis
- System tools

This is not intended to be an exhaustive list, more an indication of the depth and breadth of Delphi's applicability. Because it is possible to access any and all of the Windows API, and because if all else fails, Delphi will allow you to drop a few lines of assembler code directly into your ordinary Pascal instructions, it is possible to do more or less anything. Delphi can also be used to write Dynamically Linked Libraries (DLLs) and can call out to DLLs written in other programming languages without difficulty.

Because Delphi is based on the concept of self contained Components (elements of code that can be dropped directly on to a form in your application, and exist in object form, performing their function until they are no longer required), it is possible to build applications very rapidly. Because Delphi has been available for quite some time, the number of pre-written components has been increasing to the point that now there is a component to do more or less anything you can imagine. The job of the programmer has become one of gluing together appropriate components with code that operates them as required.

1.4 VERSIONS ARE THERE AND HOW DO THEY DIFFER?

Borland (as they were then) has a long tradition in the creation of high speed compilers. One of their best known products was Turbo Pascal - a tool that many programmers cut their teeth on. With the rise in importance of the Windows environment, it was only a matter of time before development tools started to appear that were specific to this new environment.

In the very beginning, Windows produced SDKs (software development kits) that were totally non-visual (user interface development was totally separated from the development of the actual application), and required great patience and some genius to get anything working with. Whilst these tools slowly improved, they still required a really good understanding of the inner workings of Windows.

To a great extent these criticisms were dispatched by the release of Microsoft's Visual Basic product, which attempted to bring Windows development to the masses. It achieved this to a great extent too, and remains a popular product today. However,it suffered from several drawbacks:

1) It wasn't as stable as it might have been

2) It was an interpreted language and hence was slow to run

3) It had as its underlying language BASIC, and most "real" programmers weren't so keen!

Into this environment arrived the eye opening Delphi I product, and in many ways the standard for visual development tools for Windows was set. This first version was a 16 bit compiler, and produced executable code that would run on Windows 3.1 and Windows 3.11. Of course, Microsoft have ensured (up to now) that their 32 bit operating systems (Win95, Win98, and Win NT) will all run 16 bit applications, however, many of the features that were introduced in these newer operating systems are not accessible to the 16 bit applications developed with Delphi I.

Delphi 2 was released quite soon after Delphi I, and in fact included a full distribution of Delphi I on the same CD. Delphi 2, (and all subsequent versions) have been 32 bit compilers, producing code that runs exclusively on 32bit Windows platforms. (We ignore for simplicity the WIN32S DLLs which allow Win 3.1x to run some 32 bit applications).

Delphi is currently standing at Version 4.0, with a new release (version 5.0) expected shortly. In its latest version, Delphi has become somewhat feature loaded, and as a result, we would argue, less stable than the earlier versions. However, in its defence, Delphi (and Borland products in general) have always been more stable than their competitors products, and the majority of Delphi 4's glitches are minor and forgivable - just don't try and copy/paste a selection of your code, midway through a debugging session!

The reasons for the version progression include the addition of new components, improvements in the development environment, the inclusion of more internet related support and improvements in the documentation. Delphi at version 4 is a very mature product, and Inprise has always been responsive in developing the product in the direction that the market requires it to go. Predominantly this means right now, the inclusion of more and more Internet, Web and CORBA related tools and components - a trend we are assured continues with the release of version 5.0

For each version of Delphi there are several sub-versions, varying in cost and features, from the most basic "Developer" version to the most complete (and expensive) "Client Server" version. The variation in price is substantial, and if you are contemplating a purchase, you should study the feature list carefully to ensure you are not paying for features you will never use. Even the most basic "Developer" version contains the vast majority of the features you are likely to need on a day to day basis. Don't assume that you will need Client Server, simply because you are intending to write a large database application - The developer edition is quitcapable ofthis.

1.5 What's New in Delphi 7

Delphi 7 includes new features and enhancements in the following areas:

- "IDE changes"
- "Web technology changes (Professional and Enterprise editions)"
- "COM changes (Professional and Enterprise editions)"
- "Database technology changes (Professional and Enterprise editions)"
- "Component library changes"
- "Runtime library changes"
- "Compiler changes" on page 8
- "Rave Reports support (Professional and Enterprise editions)"
- "ModelMaker support (Professional and Enterprise editions)"
- "Documentation changes"

If you are upgrading from a previous version of Delphi, see "Upgrade and compatibility issues"

IDE changes The IDE has new features in the following areas:

Compiler messages

- The new View|Additional Message Info command displays a Message Hints window from which you can download and view information about compiler messages from Borland's Web site.

• The new Project|Options|Compiler Messages page gives you greater control over which compiler warnings are generated.

Component palette changes

• There is a new CLX-only version of the System page displayed when you open a CLX application in Delphi. It includes several directory and file components. In previous releases, the System page was displayed only for VCL applications and included components for system-level access.

• The new Indy Intercepts and Indy I/O Handlers pages provide open source Internet protocol components. (Professional and Enterprise editions)

• The new IW Standard, IW Data, IW Client Side, and IW Control pages provide IntraWeb components for developing Web-based applications.

• The new Rave page provides components for adding report generation to your applications.

• If a component page can be scrolled horizontally to display additional icons, a new drop-down menu button can also be used to list the additional icons.

1.5.1 Code Insight changes

• Code completion is now faster and lets you browse to the declaration of items in the code completion list by using Ctrl+click on any identifier in the list.

• New HTML code completion automatically displays valid HTML elements and attributes in the Code editor. (Professional & Enterprise editions only)

• You can create customized code completion managers by using the OpenTools API. See "Extending the IDE" in the Delphi online help for details.

• The Tools|Editor Options|Code Insight page lets you set colors for the symbols displayed in the Code Insight tools.

1.5.2 Debugger changes

• The Watch List now has:

• Multiple tabs, allowing you to organize watches into distinct watch groups for easier debugging. To add a watch group, right-click the Watch List and select Add Group.

• A Watch Name column and a Value column. To show/hide the column headers, right-click the Watch List and select Show Column Headers.

• A checkbox to enable or disable individual watches.

as part of a multipart form. When an application receives the attachment, it saves it to a temporary file, which is then available to your application.

1.5.5 Type support

• You can now customize the conversion between remotable classes and their SOAP representation by overriding two new virtual methods that were added to TRemotable: ObjectToSOAP and SOAPToObject.

• Exception objects for exceptions that occur when responding to a Web Service request (ERemotableException instances) now contain more information from the SOAP fault packet.

• Type definitions are automatically registered with the remotable type registry when you register an invokable interface.

• TXSDecimal has a new AsBcd property for easier conversion between XML and native types. Similarly, TXSHexBinary has a new AsByteArray property. Remotable classes that represent time values now let you work with fractional seconds rather than milliseconds.

1.5.6 Other enhancements

• New events on THTTPReqResp let you to intercept the HTTP message before it is sent, and to monitor progress while sending or receiving long messages.

• THTTPSoapPascalInvoker now publish events that let you write code to execute before or after the invoker executes a requested method call.

• You now have more control over the mapping between invokable interfaces and WSDL documents. TWSDLHTMLPublish now publishes several events to let you control the generated WSDL. You can also identify the mapping between function return values and parameter names, the use of namespaces, and default SOAP actions. On the client side, literal encodings are now supported as well as RPCstyle encoding.

• A new interface, IRIOAccess lets you access the remote interfaced object that implements an invokable interface.

• The IOPConvert interface has a new property: Encoding. This allows you to specify the character set to use for encoded messages that are passed between the client and Web Service provider.

• There are changes to Web Services that affect DataSnap applications. For more information, see "Database technology changes (Professional and Enterprise editions)" .

• The TLinkedRIO constructor now automatically generates separate file names for each method you call, making debugging easier.

• TOPToSoapDomConvert now has two new events that you can use when debugging the deserialization of SOAP packets.

• You can now use overloaded methods on invokable interfaces that you define.

1.5.7 Database technology changes (Professional and Enterprise editions)

• The dbExpress drivers have been updated for Informix SE, Oracle 9i, DB2 7.2, InterBase 6.5, and MySQL 3.23.49. A new driver is available for MSSQL 2000.

• There are several new and changed database components. See "Component library changes" on page 6 for details.

• Borland has deprecated SQL Links; no further enhancements will be made to SQL Links and it will not be included with Delphi after 2002. Borland recommends using dbExpress for SQL server database access in Delphi.

1.5.7 Component library changes

• VCL applications now include components that enable support for Windows common controls version 6. Your application will automatically use the new Windows controls on Windows XP systems if it finds a suitable manifest file. For more information, see "Common controls and XP themes" in the *Developer's Guide* or online Help.

1.5.8 New unit and components

• The new DBClientActns unit contains three new action components for working with client datasets: TClientDataSetApply, TClientDataSetUndo, and TClientDataSetRevert.

• The dbExpress page of the Component palette includes TSimpleDataSet for use with simple, two-tier database applications (TSimpleDataSet replaces TSQLClientDataSet).

• The Dialogs page of the Component palette includes TPageSetupDialog for providing a Windows standard page setup dialog box.

• The Additional page of the Component palette includes TXPColorMap, TStandardColorMap, and TTwilightColorMap for colorizing menus and toolbars.

• The new CLX version of the System page of the Component palette includes new directory and file components.

• The new Indy Intercepts and Indy I/O Handlers pages on Component palette provide Internet protocols. (Professional and Enterprise editions)

Changed components;

• The CLX versions of TOpenDialog and TSaveDialog have been expanded to support additional features such as file previewing.

• The VCL version of TCustomForm has two new properties, ScreenSnap and SnapBuffer, which control whether a form snaps to the edge of the screen when the form is moved.

• TCustomComboBoxEx has a new AutoCompleteOptions property that enables a combo box to respond to user keystrokes.

• CLX dialog objects that descend from TOpenDialog and TQtDialog can now use Windows Common Dialogs in place of Qt Dialogs. This behavior is controlled by the UseNativeDialog property, which defaults to true.

Deprecated components;

• Information about deprecated components can be found in the readme.txt file in the Delphi7 directory.

1.6 SOME KNOWLEDGE ABOUT DELPHI

Delphi is a Rapid Application Development (RAD) environment. It allows you to drag and drop components on to a blank canvas to create a program. Delphi will also allow you to use write console based DOS like programs.

Delphi is based around the Pascal language but is more developed object orientated derivative. Unlike Visual Basic, Delphi uses punctuation in its basic syntax to make the program easily readable and to help the compiler sort the code. Although Delphi code is not case sensitive there is a generally accepted way of writing Delphi code. The main reason for this is so that any programmer can read your code and easily understand what you are doing, because they write their code like you write yours.

For the purposes of this series I will be using Delphi 7. There are more recent versions available (2005 and 2006) however Delphi 7 should be available inexpensively compared to the new versions which will set you back a lot of money. Delphi 7 will more than likely be available in a magazine for free.

1.6.1 Example: Try First Delphi Program

First thing is first, fire up your copy of Delphi and open the Project > Options menu. To compile a console application you need to change a setting on the Linker tab called 'Generate console application', check the box and click OK. Now select File > Close All if anything is already loaded. Then select File > New > Other > Console Application.

Notice the first line refers to the keyword program. You can rename this to HelloWorld. You can also remove the commented portion enclosed in curly brackets. The uses keyword allows you to list all units that you want to use in the program. At the moment just leave it as it is, SysUtils is all we need.

Your unit should now look like this:

Delphi Code:

```
program HelloWorld;

  {$APPTYPE CONSOLE}

 uses

  SysUtils;

 begin

 end.
```

Now what we have just done is written a program, it currently doesn't do a thing however. Hit the run button and see the result. Now wasn't that completely worthless.

Luckily this isn't the end of the article so we'll actually have a worthwhile program at the end of it. All we need to do is insert some code in the main procedure we have just made.

Every good programmer's first program was 'Hello World' and you'll be no exception. All we need to do is use the WriteLn procedure to write 'Hello World!' to the console, simple.Notice the semicolon at the end of the line, at the end of any statement you need to add a semicolon. Run the program and see the results...

Now I don't know about you but I saw hello world flash up and go away in a second, if you didn't write the program you wouldn't even know what it said. To solve this problem we need to tell the program to leave the console open until the user is ready to close it. We can use ReadLn for this which reads the users input from the console.

Delphi Code:

```
program HelloWorld;
 {$APPTYPE CONSOLE}
 uses
   SysUtils;
 begin
   WriteLn('Hello World!' + #13#10 + #13#10 +
     'Press RETURN to end...');
   ReadLn;
 end.
```

I have added a few extra things into the 'Hello World' string so the user knows what to do to end the program as it could be a bit confusing. '#13#10' is to insert a carriage return as 13 and 10 are the ASCII codes for a carriage return followed by a new line feed. ASCII can be inserted in this way into strings.

1.6.2 Delphi Style

Coding style, the way you format your code and the way in which you present it on the page.At the end of the day who cares about my style, I can read it, and Delphi strips all the spaces out of it and doesn't care if I indent. Why waste my time?

Neatly present code which conforms to the accepted standards not only makes your code much easier for you to read and debug but also but any one else who might read your code to help you, or learn from you can do so with ease. After all which code is easier to follow, example 1 or 2?

Delphi Code:

```
// Example 1

procedure xyz();

var

x,y,z,a:integer;

begin

x:=1;y:=2;

for z:=x to y do begin

a:=power(z,y);

showmessage(inttostr(a));

end;

end;
```

Delphi Code:

```
// Example 2

procedure XYZ();

var

X,Y,Z,A: Integer;
```

```
begin
 X := 1;
 Y := 2;
 for Z := X to Y do
 begin
      A := Power(Z, Y);
      ShowMessage(IntToStr(A));
 end; //for end
end; //procedure end
```

Design patterns are frequently recurring structures and relationships in object-oriented design. Getting to know them can help you design better, more reusable code and also help you learn to design more complex systems.

Much of the ground-breaking work on design patterns was presented in the book Design Patterns: Elements of Reusable Object-Oriented Software by Gamma, Helm, Johnson and Vlissides. You might also have heard of the authors referred to as "the Gang of Four". If you haven't read this book before and you're designing objects, it's an excellent primer to help structure your design. To get the most out of these examples, I recommend reading the book as well.

Another good source of pattern concepts is the book Object Models: Strategies, Patterns and Applications by Peter Coad. Coad's examples are more business oriented and he emphasises learning strategies to identify patterns in your own work.

1.7 HOW DELPHI HELPS YOU DEFINE PATTERNS

Delphi implements a fully object-oriented language with many practical refinements that simplify development.

The most important class attributes from a pattern perspective are the basic inheritance of classes; virtual and abstract methods; and use of protected and public scope. These give you the tools to create patterns that can be reused and extended, and let you isolate varying functionality from base attributes that are unchanging.

Delphi is a great example of an extensible application, through its component architecture, IDE interfaces and tool interfaces. These interfaces define many virtual and abstract constructors and operations.

1.7.1 Delphi Examples of Design Patterns

I should note from the outset, there may be alternative or better ways to implement these patterns and I welcome your suggestions on ways to improve the design. The following patterns from the book *Design Patterns* are discussed and illustrated in Delphi to give you a starting point for implementing your own Delphi patterns.

Pattern Name	*Definition*
Singleton	"Ensure a class has only one instance, and provide a global point of access to it."
Adapter	"Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces."
Template Method	"Define the skeleton of an algorithm in an operation, deferring some steps to subclasses. Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure."
Builder	"Separate the construction of a complex object from its representation so that the same construction process can create different representations."
Abstract Factory	"Provide an interface for creating families of related or dependant objects without specifying their concrete classes."
Factory Method	"Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory method lets a class defer instantiation to subclasses."

Note: These definitions are taken from *Design Patterns*.

1.7.2 Pattern: Singleton

1.7.2.1 Definition

"Ensure a class has only one instance, and provide a global point of access to it."

This is one of the easiest patterns to implement.

1.7.2.2 Applications in Delphi

There are several examples of this sort of class in the Delphi VCL, such as TApplication, TScreen or TClipboard. The pattern is useful whenever you want a single global object in your application. Other uses might include a global exception handler, application security, or a single point of interface to another application.

1.7.2.3 Implementation Example

To implement a class of this type, override the constructor and destructor of the class to refer to a global (interface) variable of the class.

Abort the constructor if the variable is assigned, otherwise create the instance and assign the variable.

In the destructor, clear the variable if it refers to the instance being destroyed.

Note: To make the creation and destruction of the single instance automatic, include its creation in the initialization section of the unit. To destroy the instance, include its destruction in an ExitProc (Delphi 1) or in the finalization section of the unit (Delphi 2).

1.7.3 Pattern: Adapter

1.7.3.1 Definition

"Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces."

1.7.3.2 Applications in Delphi

A typical example of this is the wrapper Delphi generates when you import a VBX or OCX. Delphi generates a new class which translates the interface of the external control into a Pascal compatible interface. Another typical case is when you want to build a single interface to old and new systems.

Note Delphi does not allow class adaption through multiple inheritance in the way described in Design Patterns. Instead, the adapter needs to refer to a specific instance of the old class.

1.7.3.3 Implementation Example

The following example is a simple (read only) case of a new customer class, an adapter class and an old customer class. The adapter illustrates handling the year 2000 problem, translating an old customer record containing two digit years into a new date format. The client using this wrapper only knows about the new customer class. Translation between classes is handled by the use of virtual access methods for the properties. The old customer class and adapter class are hidden in the implementation of the unit.

1.7.4 Pattern: Template Method

1.7.4.1 Definition

"Define the skeleton of an algorithm in an operation, deferring some steps to subclasses. Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure."

This pattern is essentially an extension of abstract methods to more complex algorithms.

1.7.4.2 Applications in Delphi

Abstraction is implemented in Delphi by abstract virtual methods. Abstract methods differ from virtual methods by the base class not providing any implementation. The descendant class is completely responsible for implementing an abstract method. Calling an abstract method that has not been overridden will result in a runtime error.

1.7.4.3 A typical example of abstraction is the TGraphic class.

TGraphic is an abstract class used to implement TBitmap, TIcon and TMetafile. Other developers have frequently used TGraphic as the basis for other graphics objects such as PCX, GIF, JPG representations. TGraphic defines abstract methods such as Draw, LoadFromFile and SaveToFile which are then overridden in the concrete classes. Other objects that use TGraphic, such as a TCanvas only know about the abstract Draw method, yet are used with the concrete class at runtime.

Many classes that use complex algorithms are likely to benefit from abstraction using the template method approach. Typical examples include data compression, encryption and advanced graphics processing.

1.7.4.4 Implementation Example

To implement template methods you need an abstract class and concrete classes for each alternate implementation. Define a public interface to an algorithm in an abstract base class. In that public method, implement the steps of the algorithm in calls to protected abstract methods of the class. In concrete classes derived from the base class, override each step of the algorithm with a concrete implementation specific to that class.

1.7.5 Pattern: Builder

1.7.5.1 Definition

"Separate the construction of a complex object from its representation so that the same construction process can create different representations."

A Builder seems similar in concept to the Abstract Factory. The difference as I see it is the Builder refers to single complex objects of different concrete classes but containing multiple parts, whereas the abstract factory lets you create whole families of concrete classes. For example, a builder might construct a house, cottage or office. You might employ a different builder for a brick house or a timber house, though you would give them both similar instructions about the size and shape of the house. On the other hand the factory generates parts and not the whole. It might produce a range of windows for buildings, or it might produce a quite different range of windows for cars.

1.7.5.2 Applications in Delphi

The functionality used in Delphi's VCL to create forms and components is similar in concept to the builder. Delphi creates forms using a common interface, through Application.CreateForm and through the TForm class constructor. TForm implements a common constructor using the resource information (DFM file) to instantiate the components owned by the form. Many descendant classes reuse this same construction process to create different representations. Delphi also makes developer extensions easy. TForm's OnCreate event also adds a hook into the builder process to make the functionality easy to extend.

1.7.5.3 Implementation Example

The following example includes a class TAbstractFormBuilder and two concrete classes TRedFormBuilder and TBlueFormBuilder. For ease of development some common functionality of the concrete classes has been moved into the shared TAbstractFormBuilder class.

1.7.6 Pattern: Abstract Factory

1.7.6.1 Definition

"Provide an interface for creating families of related or dependant objects without specifying their concrete classes."

The Factory Method pattern below is commonly used in this pattern.

1.7.6.2 Applications in Delphi

This pattern is ideal where you want to isolate your application from the implementation of the concrete classes. For example if you wanted to overlay Delphi's VCL with a common VCL layer for both 16 and 32 bit applications, you might start with the abstract factory as a base.

1.7.6.3 Implementation Example

The following example uses an abstract factory and two concrete factory classes to implement different styles of user interface components. TOAbstractFactory is a singleton class, since we usually want one factory to be used for the whole application.

At runtime, our client application instantiates the abstract factory with a concrete class and then uses the abstract interface. Parts of the client application that use the factory don't need to know which concrete class is actually in use.

1.7.7 Pattern: Factory Method

1.7.7.1 Definition

"Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory method lets a class defer instantiation to subclasses."

The Abstact Factory pattern can be viewed as a collection of Factory Methods.

1.7.7.2 Applications in Delphi

This pattern is useful when you want to encapsulate the construction of a class and isolate knowledge of the concrete class from the client application through an abstract interface.

One example of this might arise if you had an object oriented business application potentially interfacing to multiple target DBMS. The client application only wants to know about the business classes, not about their implementation-specific storage and retrieval.

1.7.7.3 Implementation Example

In the Abstract Factory example, each of the virtual widget constructor functions is a Factory Method. In their implementation we define a specific widget class to return.

1.8 KEY ELEMENTS OF DELPHI CLASS DEFINITIONS

1.8.1 Unit Structure

Delphi units (.PAS files) allow declaration of interface and implementation sections. The interface defines the part that is visible to other units using that unit. The keyword *uses* can be added to a unit's interface or implementation section to list the other units that your unit uses. This indicates to the compiler that your unit refers to parts of the used unit's interface. Parts of a unit declared in the implementation section are all private to that unit, i.e. never visible to any other unit. Types, functions and procedures declared in the interface of a unit must have a corresponding implementation, or be declared as external (e.g. a call to a function in a DLL).

1.8.2 Class Interfaces

Classes are defined as types in Delphi and may contain fields of standard data types or other objects, methods declared as functions or procedures, and properties. The type declaration of a class defines its interface and the scope of access to fields, methods and properties of the class. Class interfaces are usually defined in the interface of a unit to make them accessible to other modules using that unit. However they don't need to be. Sometimes a type declaration of a class may be used only within the implementation part of a unit.

1.8.3 Properties

Properties are a specialised interface to a field of a defined type, allowing access control through read and write methods. Properties are not virtual, you can replace a property with another property of the same name, but the parent class doesn't know about the new property. It is however possible to make the access methods of a property virtual.

1.8.4 Inheritance

Delphi's inheritance model is based on a single hierarchy. Every class inherits from TObject and can have only one parent.

A descendant class inherits all of the interface and functionality of its parent class, subject to the scope described below.

Multiple inheritance from more than one parent is not allowed directly. It can be implemented by using a container class to create instances one or more other classes and selectively expose parts of the contained classes.

Private, Protected, Public and Published ScopeScope refers to the visibility of methods and data defined in the interface of a class, i.e. what parts of the class are accessible to the rest of the application or to descendant classes.

The default scope is public, for instance the component instances you add to a form at design time. Public says "come and get me"; it makes the data or method visible to everything at runtime.

Published parts of a class are a specialized form of Public scope. They indicate special behaviour for classes derived from TPersistent. A persistent class can save and restore its published properties to persistent storage using Delphi's standard streaming methods. Published properties also interact with Delphi Object Inspector in the IDE. A class must descend from TPersistent in order to use Published. There's also not much point in publishing methods, since you can't store them, although Delphi's compiler doesn't stop you. Published also lets another application access details of the class through Delphi's runtime type information. This would be rarely used, except in Delphi's design time interaction with its VCL.

Encapsulation or information hiding is essential to object orientation, so Protected and Private scope let you narrow the access to parts of a class.

Protected parts are visible only to descendant classes, or to other classes defined in the same unit.

Private parts are visible only to the defining class, or to other classes defined in the same unit.

It's important to note that once something is given public or published scope, it cannot be hidden in descendant classes.

Static, Virtual and Dynamic Methods; Override and Inherited

Methods declared as virtual or dynamic let you change their behaviour using override in a descendant class. You're unlikely to see a virtual method in the private part of a class, since it could only be overridden in the same unit, although Delphi's compiler doesn't stop you from doing this.

Override indicates that your new method replaces the method of the same name from the parent class. The override must be declared with the same name and parameters as the original method.

When a method is overridden, a call to the parent class's method actually executes the override method in the real class of the object.

Static methods on the other hand have no virtual or override declaration. You can replace a method of a class in a descendant class by redeclaring another method, however this is not object oriented. If you reference your descendant class as the parent type and try to call the replaced method, the static method of the parent class is executed. So in most cases, it's a bad idea to replace a static method.

Virtual and dynamic methods can be used interchangeably. They differ only in their treatment by the compiler and runtime library. Delphi's help explains that dynamic methods have their implementation resolved at compile time and run slightly faster, whereas virtual methods are resolved at runtime, resulting in slightly slower access but a smaller compiled program. Virtual is usually the preferred declaration. Delphi's help suggests using dynamic when you have a base class with many descendants that may not override the method.

The inherited directive lets you refer back to a property or method as it was declared in the parent class. This is most often used in the implementation of an override method, to call the inherited method of the parent class and then supplement its behaviour.

1.8.5 Abstract Methods

Abstract is used in base classes to declare a method in the interface and defer its implementation to a descendant class. I.e. it defines an interface, but not the underlying operation. Abstract must be used with the virtual or dynamic directive. Abstract methods are never implemented in the base class and must be implemented in descendant classes to be used. A runtime error occurs if you try to execute an abstract method that is not overridden. Calling inherited within the override implementation of an abstract method will also result in a runtime error, since there is no inherited behaviour.

1.8.6 Messages

Delphi's handling of Windows messages is a special case of virtual methods. Message handlers are implemented in classes that descend from TControl. I.e classes that have a handle and can receive messages. Message handlers are always virtual and can be declared in the private part of a class interface, yet still allow the inherited method to be called. Inherited in a message handler just uses the keyword inherited, there is no need to supply the name of the method to call.

1.7.7 Events

Events are also an important characteristic of Delphi, since they let you delegate extensible behaviour to instances of a class. Events are properties that refer to a method of another object. Events are not inherited in Delphi 1; Delphi 2 extends this behaviour to let you use inherited in an event. . Inherited in an event handler just uses the keyword inherited, there is no need to supply the name of the method to call.

Events are particularly important to component developers, since they provide a hook for the user of the component to modify its behaviour in a way that may not be foreseen at the time the component is written.

1.8.8 Constructors and Destructors

The constructor and destructor are two special types of methods. The constructor initializes a class instance (allocates memory initialized to 0) and returns a reference (pointer) to the object. The destructor deallocates memory used by the object (but not the memory of other objects created by the object).

Classes descended from TObject have a static constructor, Create, and a virtual destructor Destroy.

TComponent introduces a new public property, the Owner of the component and this must be initialized in the constructor. TComponent's constructor is declared virtual, i.e. it can be overridden in descendant classes.It is essential when you override a virtual constructor or destructor in a TComponent descendant to include a call to the inherited method.

1.9 THE VCL TO APPLICATIONS DEVELOPERS

Applications Developers create complete applications by interacting with the Delphi visual environment (as mentioned earlier, this is a concept nonexistent in many other frameworks). These people use the VCL to create their user-interface and the other elements of their application: database connectivity, data validation, business rules, etc..

Applications Developers should know which properties, events, and methods each component makes available. Additionally, by understanding the VCL architecture, Applications Developers will be able to easily identify where they can improve their applications by extending components or creating new ones. Then they can maximize the capabilities of these components, and create better applications.

1.9.1 The VCL to Component Writers

Component Writers expand on the existing VCL, either by developing new components, or by increasing the functionality of existing ones. Many component writers make their components available for Applications Developers to use.

A Component Writer must take their knowledge of the VCL a step further than that of the Application Developer. For example, they must know whether to write a new component or to extend an existing one when the need for a certain characteristic arises. This requires a greater knowledge of the VCL's inner workings.

1.9.2 The VCL is made up of components

Components are the building blocks that developers use to design the user-interface and to provide some non-visual capabilities to their applications. To an Application Developer, a component is an object most commonly dragged from the Component palette and placed onto a form. Once on the form, one can manipulate the component's properties and add code to the component's various events to give the component a specific behavior. To a Component Writer, components are objects in Object Pascal code. Some components encapsulate the behavior of elements provided by the system, such as the standard Windows 95 controls. Other objects introduce entirely new visual or non-visual elements, in which case the component's code makes up the entire behavior of the component.

The complexity of different components varies widely. Some might be simple while others might encapsulate a elaborate task. There is no limit to what a component can do or be made up of. You can have a very simple component like a TLabel, or a much more complex component which encapsulates the complete functionality of a spreadsheet.

1.9.3 Component Types, Structure and VCL hierarchy

Components are really just special types of objects. In fact, a component's structure is based on the rules that apply to Object Pascal. There are three fundamental keys to understanding the VCL.

First, you should know the special characteristics of the four basic component types: standard controls, custom controls, graphical controls and non-visual components.

Second, you must understand the VCL structure with which components are built. This really ties into your understanding of Object Pascal's implementation. Third, you should be familiar with the VCL hierarchy and you should also know where the four component types previously mentioned fit into the VCL hierarchy. The following paragraphs will discuss each of these keys to understanding the VCL.

1.9.4 Component Types

As a component writer, there four primary types of components that you will work with in Delphi: standard controls, custom controls, graphical controls, and non-visual components. Although these component types are primarily of interest to component writers, it's not a bad idea for applications developers to be familiar with them. They are the foundations on which applications are built.

1.9.4.1 Standard Components

Some of the components provided by Delphi 2.0 encapsulate the behavior of the standard Windows controls: TButton, TListbox and Tedit, for example. You will find these components on the *Standard* page of the Component Palette. These components are Windows' common controls with Object Pascal wrappers around them.

Each standard component looks and works like the Windows' common control which it encapsulates. The VCL wrapper's simply makes the control available to you in the form of a Delphi component-it doesn't define the common control's appearance or functionality, but rather, surfaces the ability to modify a control's appearance/functionality in the form of methods and properties. If you have the VCL source code, you can examine how the VCL wraps these controls in the file STDCTRLS.PAS.

If you want to use these standard components unchanged, there is no need to understand how the VCL wraps them. If, however, you want to extend or change one of these components, then you must understand how the Window's common control is wrapped by the VCL into a Delphi component.

For example, the Windows class LISTBOX can display the list box items in multiple columns. This capability, however, isn't surfaced by Delphi's TListBox component (which encapsulates the Windows LISTBOX class). (TListBox only displays items in a single column.) Surfacing this capability requires that you override the default creation of the TListBox component.

This example also serves to illustrate why it is important for Applications Developers to understand the VCL. Just knowing this tidbit of information helps you to identify where enhancements to the existing library of components can help make your life easier and more productive.

1.9.4.2 Custom components

Unlike standard components, custom components are controls that don't already have a method for displaying themselves, nor do they have a defined behavior. The Component Writer must provide to code that tells the component how to draw itself and determines how the component behaves when the user interacts with it. Examples of existing custom components are the TPanel and TStringGrid components.

It should be mentioned here that both standard and custom components are *windowed* controls. A "windowed control" has a window associated with it and, therefore, has a window handle. Windowed controls have three characteristics: they can receive the input focus, they use system resources, and they can be parents to other controls. (Parents is related to containership, discussed later in this paper.) An example of a component which can be a container is the TPanel component.

1.9.4.3 Graphical components

Graphical components are visual controls which cannot receive the input focus from the user. They are non-windowed controls. Graphical components allow you to display something to the user without using up any system resources; they have less "overhead" than standard or custom components. Graphical components don't require a window handle-thus, they cannot can't get focus. Some examples of graphical components are the TLabel and TShape components.

Graphical components cannot be containers of other components. This means that they cannot own other components which are placed on top of them.

1.9.4.4 Non-visual components

Non-visual components are components that do not appear on the form as controls at run-time. These components allow you to encapsulate some functionality of an entity within an object. You can manipulate how the component will behave, at design-time, through the Object Inspector. Using the Object Inspector, you can modify a non-visual component's properties and provide event handlers for its events. Examples of such components are the TOpenDialog, TTable, and TTimer components.

1.9.4.5 Structure of a component

All components share a similar structure. Each component consists of common elements that allow developers to manipulate its appearance and function via properties, methods and events. The following sections in this paper will discuss these common elements as well as talk about a few other characteristics of components which don't apply to all components.

1.9.4.6 Component properties

Properties provide an extension of an object's fields. Unlike fields, properties do not store data: they provide other capabilities. For example, properties may use methods to read or write data to an object field to which the user has no access. This adds a certain level of protection as to how a given field is assigned data. Properties also cause "side effects" to occur when the user makes a particular assignment to the property.

Thus what appears as a simple field assignment to the component user could trigger a complex operation to occur behind the scenes.

1.10 PROPERTIES PROVIDE ACCESS TO INTERNAL STORAGE FIELDS

There are two ways that properties provide access to internal storage fields of components: directly or through access methods. Examine the code below which illustrates this process.

```
TCustomEdit = class(TWinControl)
private
  FMaxLength: Integer;
protected
  procedure SetMaxLength(Value: Integer);
...
published
  property MaxLength: Integer read
     FMaxLength write SetMaxLength default 0;
...
end;
```

The code above is snippet of the TCustomEdit component class. TCustomEdit is the base class for edit boxes and memo components such as TEdit, and TMemo.

TCustomEdit has an internal field FMaxLength of type Integer which specifies the maximum length of characters which the user can enter into the control. The user doesn't directly access the FMaxLength field to specify this value. Instead, a value is added to this field by making an assignment to the MaxLength property.

The property MaxLength provides the access to the storage field FMaxLength. The property definition is comprised of the property name, the property type, a read declaration, a write declaration and optional default value.

The read declaration specifies how the property is used to read the value of an internal storage field. For instance, the MaxLength property has direct read access to FMaxLength. The write declaration for MaxLength shows that assignments made to the MaxLength property result in a call to an *access method* which is responsible for assigning a value to the FMaxLength storage field. This access method is SetMaxLength.

1.10.1 Property-access methods

Access methods take a single parameter of the same type as the property. One of the primary reasons for write access methods is to cause some side-effect to occur as a result of an assignment to a property. Write access methods also provide a method layer over assignments made to a component's fields. Instead of the component user making the assignment to the field directly, the property's write access method will assign the value to the storage field if the property refers to a particular storage field. For example, examine the implementation of the SetMaxLength method below.

```
procedure TCustomEdit.SetMaxLength(Value: Integer);
begin
 if FMaxLength <> Value then
 begin
  FMaxLength := Value;
  if HandleAllocated then
     SendMessage(Handle, EM_LIMITTEXT, Value, 0);
 end;
end;
```

The code in the SetMaxLength method checks if the user is assigning the same value as that which the property already holds. This is done as a simple optimization. The method then assigns the new value to the internal storage field, FMaxLength. Additionally, the method then sends an EM_LIMITTEXT Windows message to the window which the TCustomEdit encapsulates. The EM_LIMITTEXT message places a limit on the amount of text that a user can enter into an edit control. This last step is what is referred to as a *side-effect* when assigning property values. Side effects are any additional actions that occur when assigning a value to a property and can be quite sophisticated.

Providing access to internal storage fields through property access methods offers the advantage that the Component Writer can modify the implementation of a class without modifying the interface. It is also possible to have access methods for the read access of a property. The read access method might, for example, return a type which is different that that of a properties storage field. For instance, it could return the string representation of an integer storage field.

Another fundamental reason for properties is that properties are accessible for modification at run-time through Delphi's Object Inspector. This occurs whenever the declaration of the property appears in the published section of a component's declaration.

1.10.2 Types of properties

Property type	Object Inspector treatment
Simple	Numeric, character, and string properties appear in the Object Inspector as numbers, characters, and strings, respectively. The user can type and edit the value of the property directly.
Enumerated	Properties of enumerated types (including Boolean) display the value as defined in the source code. The user can cycle through the possible values by double-clicking the value column. There is also a drop-down list that shows all possible values of the enumerated type.
Set	Properties of set types appear in the Object Inspector looking like a set. By expanding the set, the user can treat each element of the set as a Boolean value: True if the element is included in the set or False if

	it's not included.
Object	Properties that are themselves objects often have their own property editors. However, if the object that is a property also has published properties, the Object Inspector allows the user to expand the list of object properties and edit them individually. Object properties must descend from TPersistent.
Array	Array properties must have their own property editors. The Object Inspector has no built-in support for editing array properties.

Properties can be of the standard data types defined by the Object Pascal rules. Property types also determine how they are edited in Delphi's Object Inspector. The table below shows the different property types as they are defined in Delphi's online help.

For more information on properties, refer to the "Component Writers Guide" which ships with Delphi.

1.10.3 Methods

Since components are really just objects, they can have methods. We will discuss some of the more commonly used methods later in this paper when we discuss the different levels of the VCL hierarchy.

1.10.4 Events

Events provide a means for a component to notify the user of some pre-defined occurrence within the component. Such an occurrence might be a button click or the pressing of a key on a keyboard.

Components contain special properties called events to which the component user assigns code. This code will be executed whenever a certain event occurs. For instance, if you look at the events page of a TEdit component, you'll see such events as OnChange, OnClick and OnDblClick. These events are nothing more than pointers to methods.

When the user of a component assigns code to one of those events, the user's code is referred to as an event handler. For example, by double clicking on the events page for a particular event causes Delphi to generate a method and places you in the Code Editor where you can add your code for that method. An example of this is shown in the code below, which is an OnClick event for a TButton component.

It becomes clearer that events are method pointers when you assign an event handler to an event programmatically. The above example was Delphi generated code. To link your own an event handler to a TButton's OnClick event at run time you must first create a method that you will assign to this event. Since this is a method, it must belong to an existing object. This object can be the form which owns the TButton component although it doesn't have to be. In fact, the event handlers which Delphi creates belong to the form on which the component resides. The code below illustrates how you would create an event handler method.

When you define methods for event handlers, these methods must be defined as the same type as the event property and the field to which the event property refers. For instance, the OnClick event refers to an internal data field, FOnClick. Both the property OnClick, and field FOnClick are of the type TNotifyEvent. TNotifyEvent is a procedural type as shown below:

```
TNotifyEvent = procedure (Sender: TObject) of object;
```

Note the use of the of object specification. This tells the compiler that the procedure definition is actually a method and performs some additional logic like ensuring that an implicit Self parameter is also passed to this method when called. Self is just a pointer reference to the class to which a method belongs.

1.10.5 Containership

Some components in the VCL can own other components as well as be parents to other components. These two concepts have a different meaning as will be discussed in the section to follow.

1.10.6 Ownership

All components may be owned by other components but not all components can own other components. A component's Owner property contains a reference to the component which owns it.

The basic responsibility of the owner is one of resource management. The owner is responsible for freeing those components which it owns whenever it is destroyed. Typically, the form owns all components which appear on it, even if those components are placed on another component such as a TPanel. At design-time, the form automatically becomes the owner for components which you place on it. At run-time, when you create a component, you pass the owner as a parameter to the component's constructor. For instance, the code below shows how to create a TButton component at run-time and passes the form's implicit Self variable to the TButton's Create constructor. TButton.Create will then assign whatever is passed to it, in this case Self or rather the form, and assign it to the button's Owner property.

```
MyButton := TButton.Create(self);
```

When the form that now owns this TButton component gets freed, MyButton will also be freed.

You can create a component without an owner by passing nil to the component's Create constructor, however, you must ensure that the component is freed when it is no longer needed. The code below shows you how to do this for a TTable component.

1.10.7 Parenthood

Parenthood is a much different concept from ownership. It applies only to windowed components, which can be parents to other components. Later, when we discuss the VCL hierarchy, you will see the level in the hierarchy which introduces windowed controls.

Parent components are responsible for the display of other components. They call the appropriate methods internally that cause the children components to draw themselves. The Parent property of a component refers to the component which is its parent. Also, a component's parent does not have to be it's owner. Although the parent component is mainly responsible for the display of components, it also frees children components when it is destroyed.

Windowed components are controls which are visible user interface elements such as edit controls, list boxes and memo controls. In order for a windowed component to be displayed, it must be assigned a parent on which to display itself. This task is done automatically by Delphi's design-time environment when you drop a component from the Component Palette onto your form.

CHAPTER 2

DATABASE

Every thing around us has a particular identity. To identify anything system, actor or person in words we need a data or information. So this information is valuable and in this advanced era we can store it in database and access this data by the blink of eye.

For an instant if we go through the definitions of database we may find following definitions.

A database is a collection of related information.

A database is an organized body of related information.

2.1 DEMERITS TO ABSENCE OF DATABASE

A glance on the past will may help us to reveal the drawbacks in case of absence of database.

In the past when there wasn't proper system of database, Much paper work was need to do and to handle great deal of written paper documentation was giant among the problems itself.

In the huge networks to deal with equally bulky data, more workers are needed which affidavit cost much labor expanses.

The old criteria for saving data and making identification was much time consuming such as if we want to search the particular data of a person.

Before the Development of Computer database it was a great problem to search for some thing. Efforts to avoid the headache of search often results in new establishments of data.

Before the development of database it seemed very unsafe to keep the worthy information. In Some situation some big organization had to employee the special persons in order to secure the data.

Before the implementation of database any firm had to face the plenty of difficulties in order to maintain their Management. To hold the check on the expenses of the firm, the manager faced difficulties.

2.2 MERITS OF DATABASE

The modern era is known as the golden age computer sciences and technology. In a simple phrase we can express that the modern age is built on the foundation of database.

If we carefully watch our daily life we can examine that some how our daily life is being connected with database.

There are several benefits of database developments.

Now with the help of computerized database we can access data in a second.

By the development of the database we can make data more secure.

By the development of database we can reduce the cost.

2.3 DATABASE DESIGN

The design of a database has to do with the way data is stored and how that data is related. The design process is performed after you determine exactly what information needs to be stored and how it is to be retrieved.

A collection of programs that enables you to store, modify, and extract information from a database. There are many different types of DBMS ranging from small systems that run on personal computers to huge systems that run on mainframes. The following are examples of database applications:

Computerized library systems

Automated teller machines

Flight reservation systems

Computerized parts inventory systems

From a technical standpoint, DBMS can differ widely. The terms relational, network, flat, and hierarchical all refer to the way a DBMS organizes information internally. The internal organization can affect how quickly and flexibly you can extract information.

Requests for information from a database are made in the form of a query.

Database design is a complex subject. A properly designed database is a model of a business, Country Database or some other in the real world. Like their physical model counterparts, data models enable you to get answers about the facts that make up the objects being modeled. It's the questions that need answers that determine which facts need to be stored in the data model.

In the relational model, data is organized in tables that have the following characteristics: every record has the same number of facts, every field contains the same type of facts (Data) in each record, and there is only one entry for each fact. N*o* two records are exactly the same.

The more carefully you design, the better the physical database meets users' needs. In the process of designing a complete system, you must consider user needs from a variety of viewpoints.

2.4 DATABASE MODELS

Various techniques are used to model data structures. Certain models are more easily implemented by some types of database management systems than others. For any one logical model various physical implementation may be possible. An example of this is the relational model: in larger systems the physical implementation often has indexes which point to the data; this is similar to some aspects of common implementations of the network model. But in small relational database the data is often stored in a set of files, one per table, in a flat, un-indexed structure. There is some confusion below and elsewhere in this article as to logical data model vs. its physical implementation.

2.4.1 Flat Model

The flat (or table) model consists of a single, two dimensional array of data elements, where all members of a given column are assumed to be similar values, and all members of a row are assumed to be related to one another. For instance, columns for name and password might be used as a part of a system security database. Each row would have the specific password associated with a specific user. Columns of the table often have a type associated with them, defining them as character data, date or time information, integers, or floating point numbers. This model is the basis of the spreadsheet.

2.4.2 Network Model

The network model allows multiple datasets to be used together through the use of pointers (or references). Some columns contain pointers to different tables instead of data. Thus, the tables are related by references, which can be viewed as a network structure. A particular subset of the network model, the hierarchical model, limits the relationships to a tree structure, instead of the more general directed graph structure implied by the full network model.

2.4.3 Relational Model

The relational data model was introduced in an academic paper by E.F. Cod in 1970 as a way to make database management systems more independent of any particular application. It is a mathematical model defined in terms of predicate logic and set theory.

Although the basic idea of a relational database has been very popular, relatively few people understand the mathematical definition and only a few obscure DBMSs implement it completely and without extension. Oracle, for example, can be used in a purely relational way, but it also allow tables to be defined that allow duplicate rows an extension (or violation) of the relational model. In common English usage, a DBMS is called relational if it supports relational operational operations, regardless of whether it enforces strict adherence to the relational model. The following is an informal, not-technical explanation of how "relational" database management systems commonly work.

A relational database contains multiple tables, each similar to the one in the "flat" database model. However, unlike network databases, the tables are not linked by pointers. Instead, keys are used to match up rows of data in different tables. A key is just one or more columns in one table that correspond to columns in other tables. Any column can be a key, or multiple columns can be grouped together into a single key. Unlike pointers, it's not necessary to define all the keys in advance; a column can be used as a key even if it wasn't originally intended to be one.

A key that can be used to uniquely identify a row in a table is called a unique key. Typically one of the unique keys is the preferred way to refer to row; this is defined as the table's primary key.

When a key consists of data that has an external, real-world meaning (such as a person's name, a book's ISBN, or a car's serial number), it's called a "natural" key. If no nature key is suitable, an arbitrary key can be assigned (such as by given employees ID numbers). In practice, most databases have both generated and natural keys, because generated keys can be used internally to create links between rows that can't break, while natural keys can be used, less reliably, for searches and for integration with other databases. (For example, records in two independently developed databases could be matched up by social security number, except when the social security numbers are incorrect, missing, or have changed).

2.4.3.1 Why we use a Relational Database Design

Maintaining a simple, so-called flat database consisting of a single table doesn't require much knowledge of database theory. On the other hand, most database worth maintaining are quite a bit more complicated than that. Real life databases often have hundreds of thousands or even millions of records, with data that are very intricately related. This is where using a full-fledged relational database program becomes essential. Consider, for example, the Library of Congress, which has over 16 million books in its collection. For reasons that will become apparent soon, a single table simply will not do for this database.

2.5 RELATIONSHIPS BETWEEN TABLES

When you create tables for an application, you should also consider the relationships between them. These relationships give a relational database much of its power. There are three types of relationships between tables: one-to-one, one-to-many and many-to-many relationships.

2.5.1 One-To-One Relationships

In a one-to-one relationship, each record in one table corresponds to a single record in a second table. This relationship is not very common, but it can offer several benefits. First, you can put the fields from both tables into a single, combined table. One reason for using two tables is that each field is a property of a separate entity, such as owner operators and their tracks. Each operator can operate just one truck at a time, but the fields for the operator and truck tables refer to different entities.

A one-to-one relationship can also reduce the time needed to open a large table by placing some of the table's columns in a second, separate table. This approach makes particular sense when a table has some fields that are used infrequently. Finally, a one-to-one relationship can support in a table requires security, placing them in a separate table lets your application restrict to certain fields. Your application can link the restricted table back to the main table via a one-to-one relationship so that people with proper permissions can edit, delete, and add new records to these fields.

2.5.2 One-To-Many Relationships

A one-to-many relationship, in which a row from one table corresponds to one or more rows from a second table, is more common. This kind of relationship can form the basis for a Many-To-Many relationship as well.

2.6 DATA MODELING

In information system design, data modeling is the analysis and design of the information in the system, concentrating on the logical entities and the logical dependencies between these entities. Data modeling is an abstraction activity in that the details of the values of individual data observations are ignored in favor of the structure, relationships, names and formats of the data of interest, although a list of valid values is frequently recorded. It is by the data model that definitions of what the data means is related to the data structures.

While a common term for this activity is "Data Analysis" the activity actually has more in common with the ideas and methods of synthesis (putting things together), than it does in the original meaning of the term analysis (taking things apart). This is because the activity strives to bring the data structures of interest together in a cohesive, inseparable, whole by eliminating unnecessary data redundancies and relating data structures by relationships.

2.6.1 Database Normalization

Database normalization is a series of steps followed to obtain a database design that allows for consistent storage and efficient access of data in a relational database. These steps reduce data redundancy and the risk of data becoming inconsistent.

However, many relational DBMS lack sufficient separation between the logical database design and the physical implementation of the data store, such that queries against a fully normalized database often perform poorly. In this case de-normalizations are sometimes used to improve performance, at the cost of reduced consistency.

2.6.2 Primary Key

In database design, a primary key is a value that can be used to identify a particular row in a table. Attributes are associated with it. Examples are names in a telephone book (to look up telephone numbers), words in a dictionary (to look up definitions) and Dewey Decimal Numbers (to look up books in a library).

In the relational model of data, a primary key is a candidate key chosen as the main method of uniquely identifying a relation. Practical telephone books, dictionaries and libraries can not use names, words or Dewey Decimal System Numbers as candidate keys because they do not uniquely identify telephone numbers, word definitions or books. In some design situations it is impossible to find a natural key that uniquely identifies a relation. A surrogate key can be used as the primary key. In other situations there may be more than one candidate key for a relation, and no candidate key is obviously preferred. A surrogate key may be used as the primary key to avoid giving one candidate key artificial primacy over the others. In addition to the requirement that the primary key be a candidate key, there are several other factors which may make a particular choice of key better than others for a given relation.

The primary key should generally be short to minimize the amount of data that needs to be stored by other relations that reference it. A compound key is usually not appropriate. (However, this is a design consideration, and some database management systems may be better than others in this regard.)

The primary key should be immutable, meaning its value should not be changed during the course of normal operations of the database. (Recall that a primary key is the means of uniquely identifying a tuple, and that identity by definition, never changes.) This avoids the problem of dangling references or orphan records created by other relations referring to a tuple whose primary key has changed. If the primary key is immutable, this can never happen.

2.6.3 Foreign Key

A foreign key (FK) is a field in a database record under one primary key that points to a key field of another database record in another table where the foreign key of one table refers to the primary key of the other table. This way references can be made to link information together and it is an essential part of database normalization.

For example, a person sending an e-mail needs not to include the entire text of a book in the e-mail. Instead, they can include the ISBN of the book, and interested persons can then use the number to get information about the book, or even the book itself. The ISBN is the primary key of the book, and it is used as a foreign key in the e-mail.

Note that using a foreign key often assumes its existence as a primary key somewhere else. Improper foreign key/primary key relationships are the source of many database problems.

2.6.4 Compound Key

In database design, a compound key (also called a composite key) is a key that consists on 2 or more attributes.

No restriction is applied to the attribute regarding their (initial) ownership within the data model. This means that any one, none or all, of the multiple attributes within the compound key can be foreign keys. Indeed, a foreign key may, itself, be a compound key.

Compound keys almost always originate from attributive or associative entities (tables) within the model, but this is not an absolute value.

CHAPTER 3

Automation Of Pharmacy

3.1 Database and Tables

Paradox is a relational database management system currently published by Corel Corporation. It was originally released for DOS by Ansa Software, but a Windows version was released by Borland in 1992.

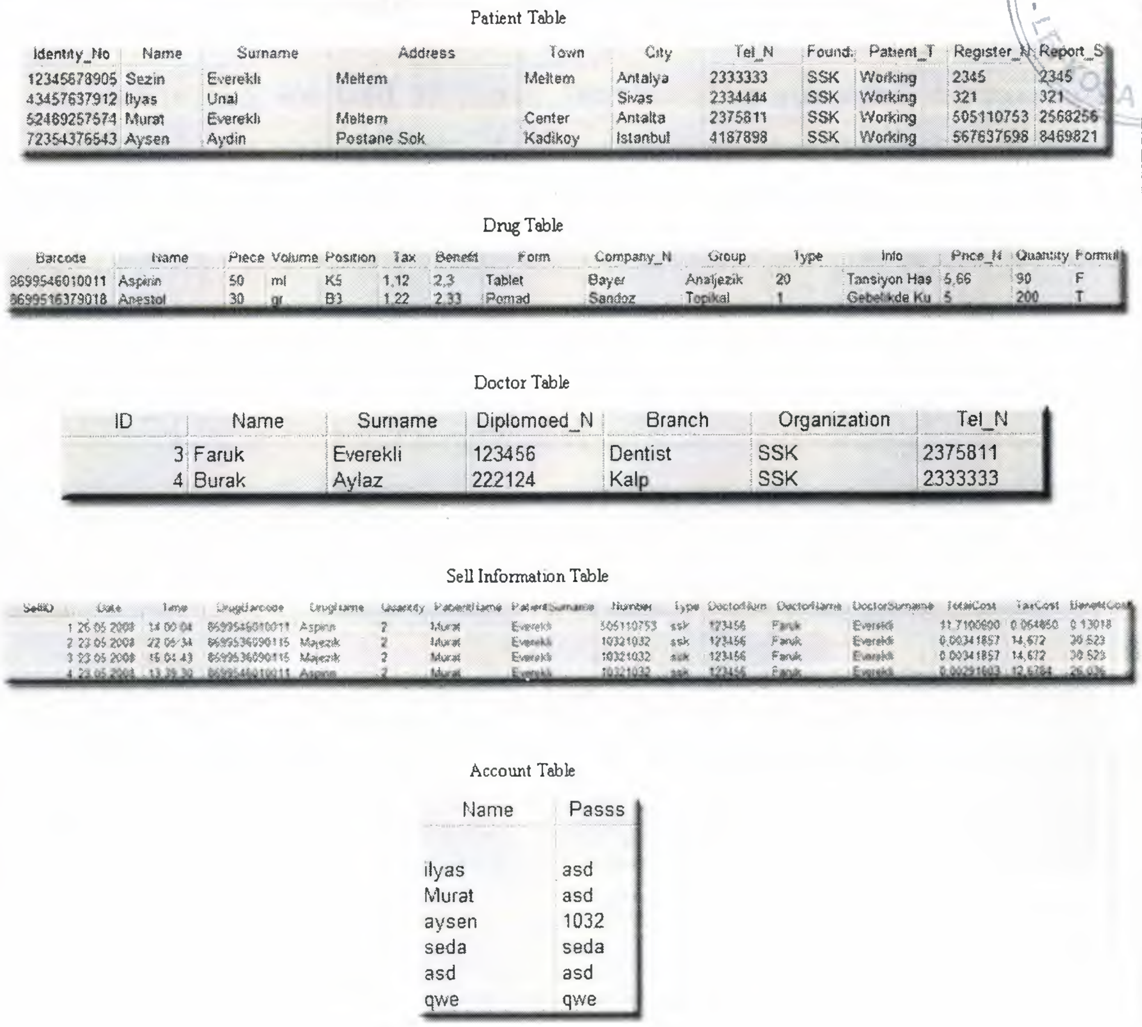
The Paradox database driver works in one of two modes. One mode is when either of the following is installed. Paradox recovery is a data recovery program for paradox databases [**.db**].

We prefer to use Paradox database system in this project. Because this system's usage is easy and very useful and maximum record number is enough to pharmacy system storage.

We used 5 tables; Patient table, Doctor Table, Drug Table, Selling Table and Account table. Patient table, doctor table, drug table and selling table are related with each other. Patient table's, doctor table's and drug table's primary keys use in selling table. See my table design in Figure 3.1.1 .

First table name is Patient Table. Patient table's primary key is Identitiy_No. This table connected to Sell Information table with using Identity number. When the users went to sell drug and enter the patient's identity number, patient's name and surname automatically fill the edit with using this primary key.

Second table is the Drug Table. Drug table's primary key is barcode. Barcode system is the international unique. Every drug has a unique barcode number. Using this primary key I relate the sell information table. The users enter the barcode number with using barcode reader, primary key catch the drug's name, quality, tax proportion, benefit proportion and net cost. Than using this information and write this data to sell information table.



Patient Table

Identity_No	Name	Surname	Address	Town	City	Tel_N	Found.	Patient_T	Register_N	Report_S
12345678905	Sezin	Everekli	Meltem	Meltem	Antalya	2333333	SSK	Working	2345	2345
43457637912	Ilyas	Unal			Sivas	2334444	SSK	Working	321	321
52469257574	Murat	Everekli	Meltem	Center	Antalta	2375811	SSK	Working	505110753	2568256
72354376543	Aysen	Aydin	Postane Sok	Kadikoy	Istanbul	4187898	SSK	Working	567637698	8469821

Drug Table

Barcode	Name	Piece	Volume	Position	Tax	Benefit	Form	Company_N	Group	Type	Info	Price_N	Quantity	Formul
8699546010011	Aspirin	50	ml	K5	1,12	2,3	Tablet	Bayer	Analjezik	20	Tansiyon Has	5,66	90	F
8699516379018	Anestol	30	gr	B3	1,22	2,33	Pomad	Sandoz	Topikal	1	Gebelikde Ku	5	200	T

Doctor Table

ID	Name	Surname	Diplomoed_N	Branch	Organization	Tel_N
3	Faruk	Everekli	123456	Dentist	SSK	2375811
4	Burak	Aylaz	222124	Kalp	SSK	2333333

Sell Information Table

SellID	Date	Time	DrugBarcode	DrugName	Quantity	PatientName	PatientSurname	Number	Type	DoctorNum	DoctorName	DoctorSurname	TotalCost	TaxCost	BenefitCos
1	26.05.2008	14:00:04	8699546010011	Aspirin	2	Murat	Everekli	505110753	ssk	123456	Faruk	Everekli	11.7100800	0.064850	0.13018
2	23.05.2008	22:05:34	8699536090115	Majezik	2	Murat	Everekli	10321032	ssk	123456	Faruk	Everekli	0,00341857	14,672	30.523
3	23.05.2008	15:04:43	8699536090115	Majezik	2	Murat	Everekli	10321032	ssk	123456	Faruk	Everekli	0.00341857	14,672	30.523
4	23.05.2008	13:39:30	8699546010011	Aspirin	2	Murat	Everekli	10321032	ssk	123456	Faruk	Everekli	0.00291603	12,6784	26.036

Account Table

Name	Passs
ilyas	asd
Murat	asd
aysen	1032
seda	seda
asd	asd
qwe	qwe

Figure 3.1.1 Project's tables

Third table is the Doctor Table. We use a Diplomoed Number as a primary key. The users enter the doctor's diplomoed number, doctor name and surname enters the sell information table automatically.

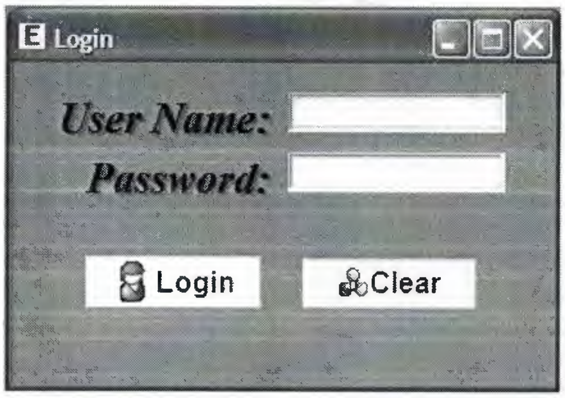
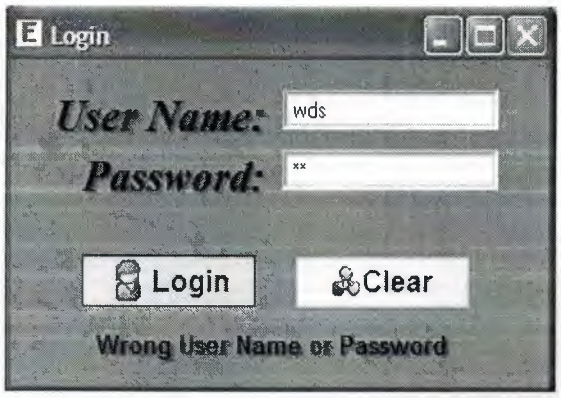
Sell Information Table is the most important table. Table's primary key is SellID. First three table related with this table.

Last table name is Account. Account table primary key is Name but this table is not related with other table.

3.2 Automation of Pharmacy Program's Forms

In our project We used 19 forms. We wanted to connected database and program; We used datasource, query and table components. And We used much more than 60 buttons, 120 editboxs, 100 labels, 30 dbeditboxs, graphs and calendars.

3.2.1 Login Form

 Figure 3.2.1.1 Login area	 Login User Name: wds Password: ** Login Clear Wrong User Name or Password Figure 3.2.1.2 When enter wrong password

When the program is started, this window opens. This is the login area. (Figure 3.2.1) Users must be having Username and Password. If he/she has not got, who applies to get account. The users can control his/her account in account control area. When users enter username and password true, the main menu opens.

If users enter wrong username or password, the program shows this window. (Figure 3.2.2)

3.2.2 Main Form

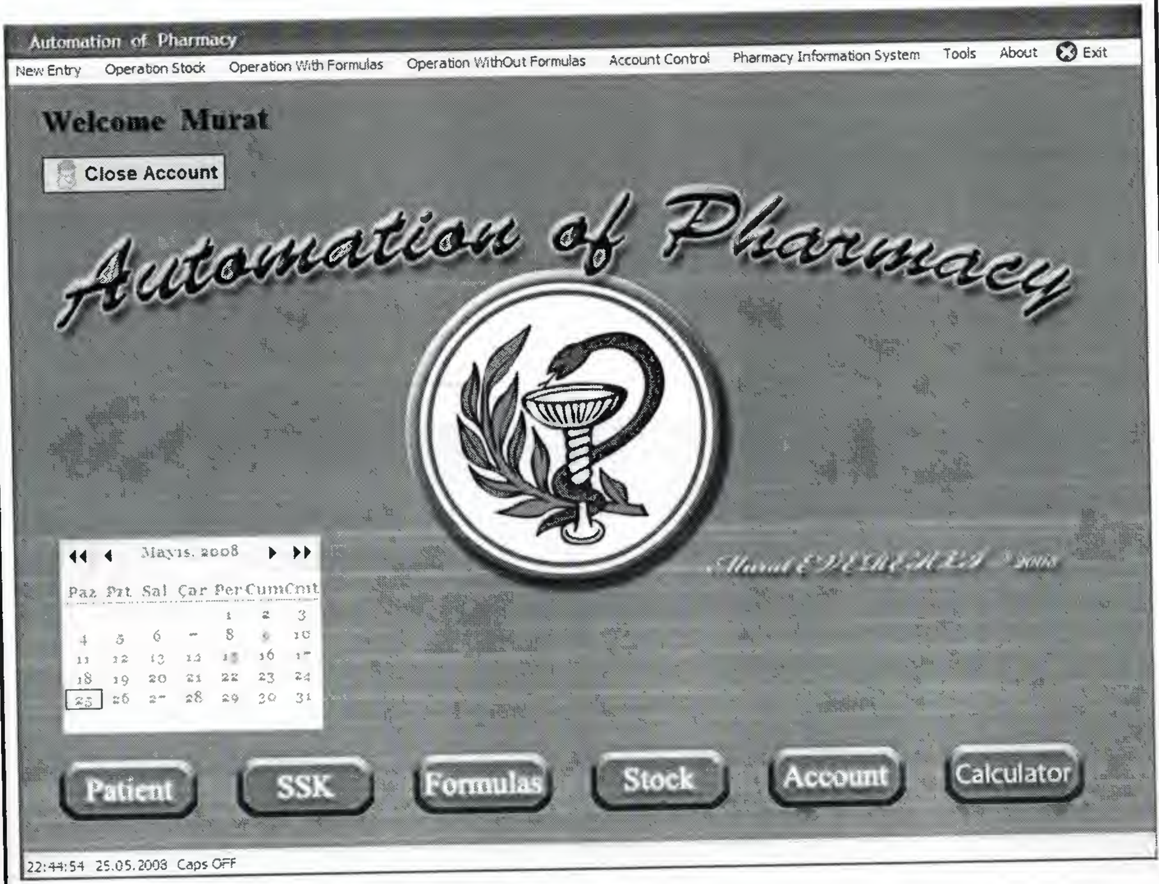


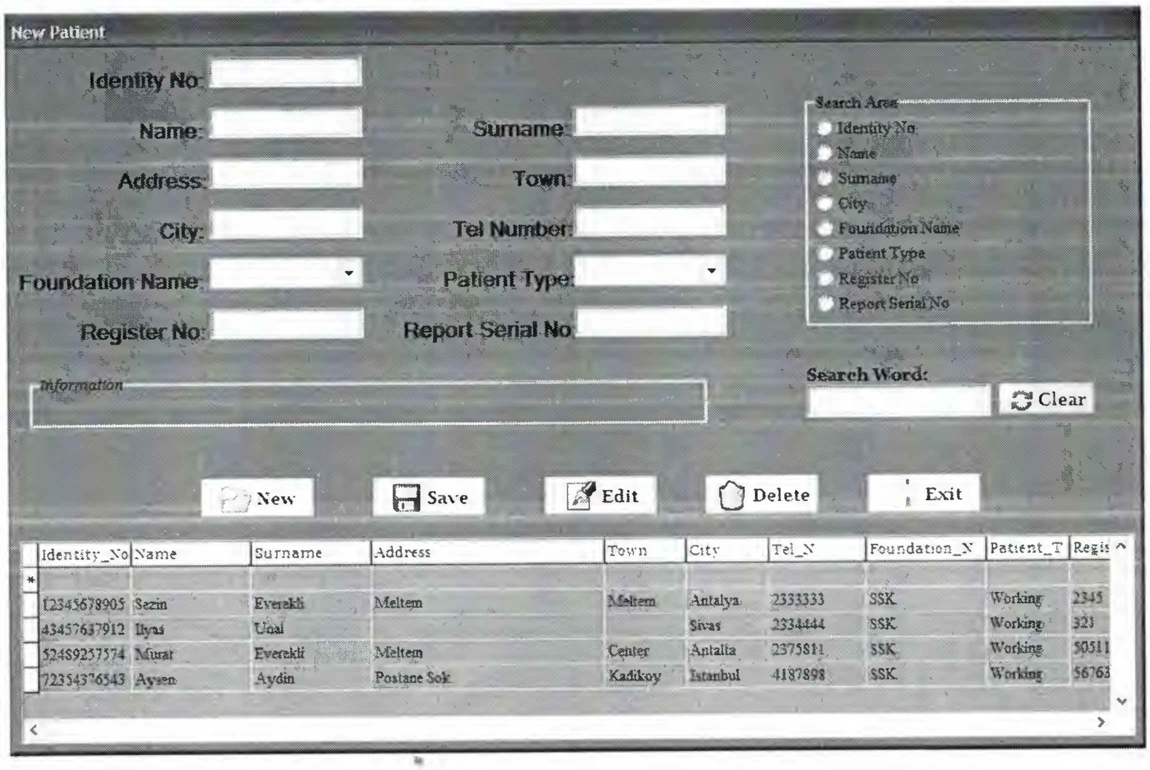
Figure 3.2.2.1 Main Menu

When user passes the login page; the main menu opens. (Figure 3.2.3) Main menu have 9 top main menus;

- New entry uses to enter new patient, doctor and drugs.
- Operation stock menu has stock entry and stock exit.
- Operation with formulas has two areas. SSK formulas sales and formulas sales.
- Operation without formulas area use to sell without formulas
- Account control menu has new account area and delete account area.
- Pharmacy information system has two areas. Stock controlling system and selling controlling area.
- Tools; calculator, internet browser and Microsoft word.
- In the About area; help and about.
- Last main menu is the Exit button.

6 quick buttons, calendars and close account button. The users can control the date and time. 6 quick buttons are the most use area; New Patient, selling with SSK formulas, selling with formulas, stock entry, new account and calculator. If the users want to close your account; he/she will use the Close Account button in main form.

3.2.3 New Patient Form

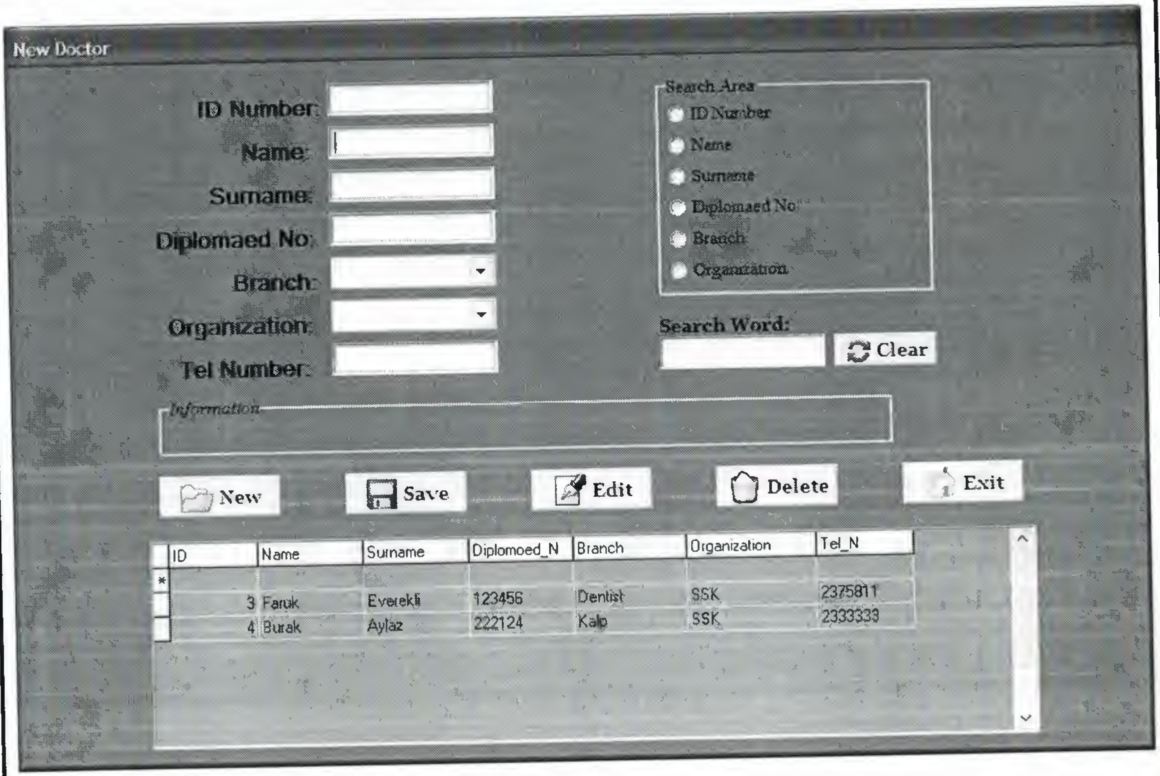


Identity_No	Name	Surname	Address	Town	City	Tel_N	Foundation_N	Patient_T	Regis
12345678905	Sezin	Everekli	Meltem	Meltem	Antalya	2333333	SSK	Working	2345
43457637912	Ilyas	Unal			Sivas	2334444	SSK	Working	321
52489257574	Murat	Everekli	Meltem	Center	Antalia	2375811	SSK	Working	50511
72354376543	Aysen	Aydin	Postane Sok	Kadikoy	Istanbul	4187898	SSK	Working	56763

Figure 3.2.3.1 New Patient

This form is related with the Patient Table. We used this form to insert new record, edit or delete the record into patient table. Identity Number must be entering. Because this is the primary key. Users can search with using Search Area who search with using identity number, name, surname, city, foundation name, patient type, register no or report serial number. Information area uses to information about the active editbox. We used dbgrid to see patient table in there.

3.2.4 New Doctor Form

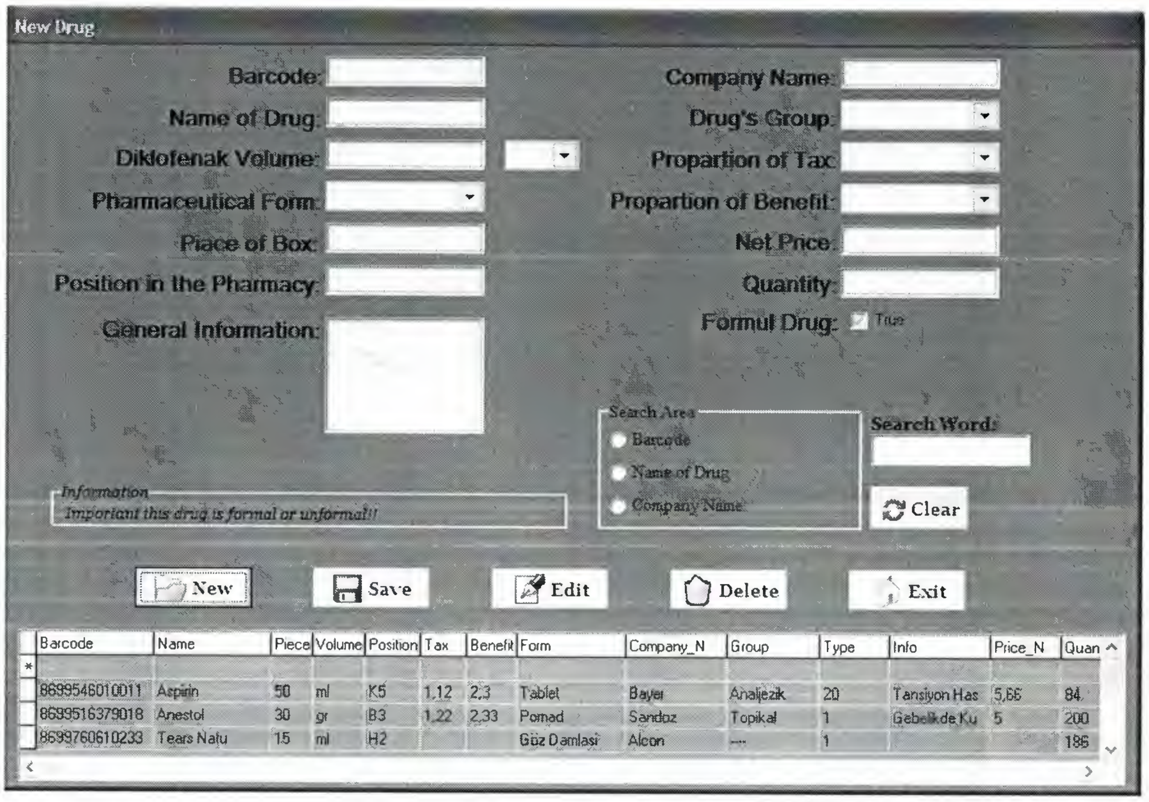


ID	Name	Surname	Diplomoed_N	Branch	Organization	Tel_N
3	Faruk	Everekli	123456	Dentist	SSK	2375811
4	Burak	Aylaz	222124	Kalp	SSK	2333333

Figure 3.2.4.1 New Doctor

This form is related with the Doctor Table. We used this form to insert new record, edit or delete the record into doctor table. Diplomoed Number must be entering. Because this is the primary key. ID number is the autoincrement which the system automatically gives. Users can search with using Search Area who searches with using id number, name, surname, diplomaed number, foundation name, branch or organization. Information area uses to information about the active editbox. We used dbgrid to see doctor table in there.

3.2.5 New Drug Form

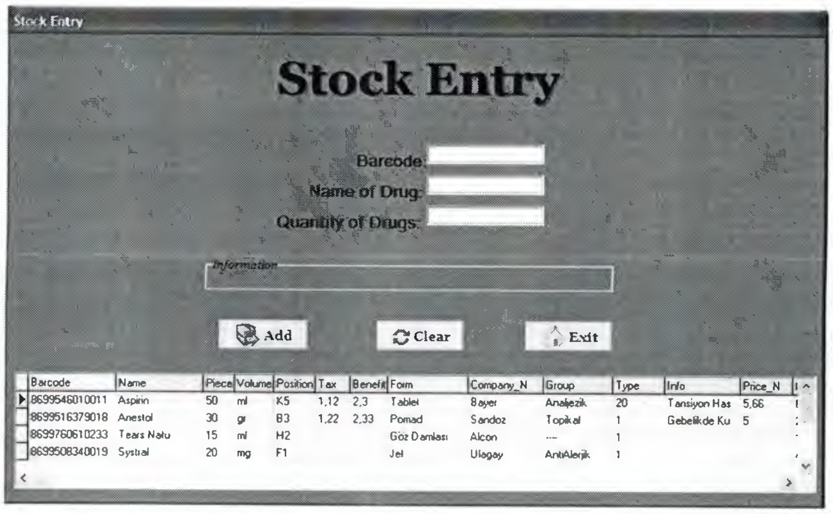


	Barcode	Name	Piece	Volume	Position	Tax	Benefit	Form	Company_N	Group	Type	Info	Price_N	Quan
*														
	8699546010011	Aspirin	50	ml	K5	1,12	2,3	Tablet	Bayer	Analjezik	20	Tansiyon Has	5,66	84
	8699516379018	Anestol	30	gr	B3	1,22	2,33	Pomad	Sandoz	Topikal	1	Gebelikde Ku	5	200
	8699760610233	Tears Nalu	15	ml	H2			Göz Damlasi	Alcon	---	1			186

Figure 3.2.5.1 New Drug

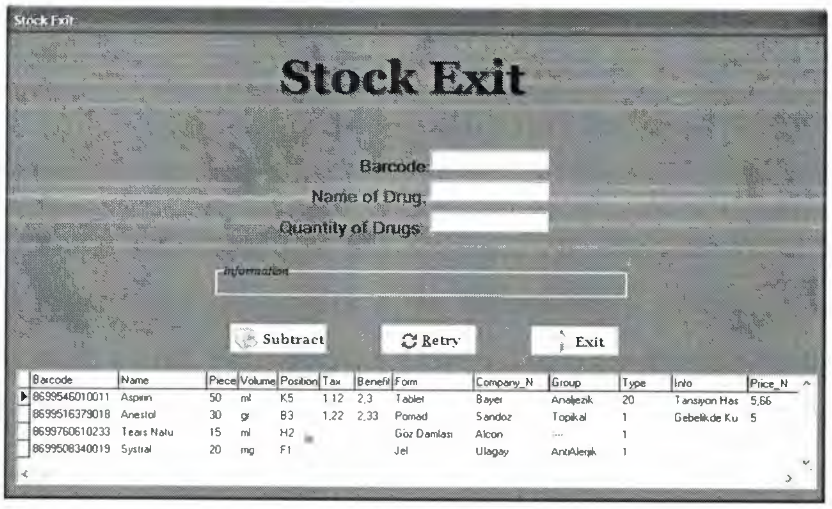
This form is related with the Drug Table. We used this form to insert new record, edit or delete the record into drug table. Barcode must be entering. Because this is the primary key. When the user enters the barcode number, he/she can use barcode machine. Users can search with using Search Area who searches with using barcode number, name of drug or company name. Information area uses to information about the active editbox. We used dbgrid to see drug table in there. This form's important part is formula drug is true or false. This is important because when the users want to sell the formula drug to buyer who is not having formula, system will be error and the buyer can not buy.

3.2.6 Stock Entry and Stock Exit Forms



Barcode	Name	Piece	Volume	Position	Tax	Benefit	Form	Company_N	Group	Type	Info	Price_N
8699546010011	Aspirin	50	ml	K5	1,12	2,3	Tablet	Bayer	Analjezik	20	Tansiyon Has	5,66
8699516379018	Anestol	30	gr	B3	1,22	2,33	Pomad	Sandoz	Topikal	1	Gebelikde Ku	5
8699760610233	Tears Natu	15	ml	H2			Göz Damlası	Alcon	---	1		
8699508340019	Systral	20	mg	F1			Jel	Ulagay	AntiAlerjik	1		

Figure 3.2.6.1 Stock Entry

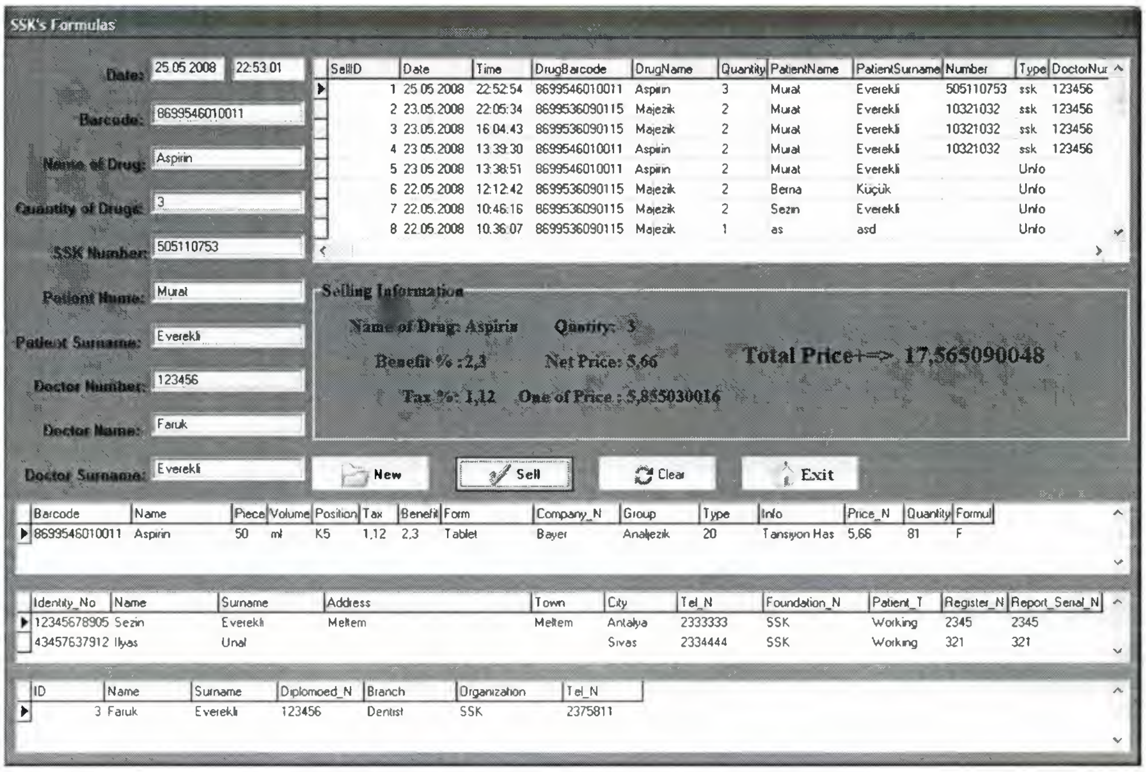


Barcode	Name	Piece	Volume	Position	Tax	Benefit	Form	Company_N	Group	Type	Info	Price_N
8699546010011	Aspirin	50	ml	K5	1,12	2,3	Tablet	Bayer	Analjezik	20	Tansiyon Has	5,66
8699516379018	Anestol	30	gr	B3	1,22	2,33	Pomad	Sandoz	Topikal	1	Gebelikde Ku	5
8699760610233	Tears Natu	15	ml	H2			Göz Damlası	Alcon	---	1		
8699508340019	Systral	20	mg	F1			Jel	Ulagay	AntiAlerjik	1		

Figure 3.2.6.2 Stock Exit

These forms are related with the Drug Table too. We used these forms to add quantity or subtract quantity from the stock. Barcode must be entering to change quantity which is the primary key. When the barcode machine read the barcode number, name of drug automatically fill. Than user enter quantity to add or subtract. Information area uses to information about the active editbox. We used dbgrid to see drug table in there.

3.2.7 Selling with SSK Formulas and Selling with Formulas Forms



Date:	25.05.2008 22:53:01
Barcode:	8699546010011
Name of Drug:	Aspirin
Quantity of Drugs:	3
SSK Number:	505110753
Patient Name:	Murat
Patient Surname:	Everekli
Doctor Number:	123456
Doctor Name:	Faruk
Doctor Surname:	Everekli

SellID	Date	Time	DrugBarcode	DrugName	Quantity	PatientName	PatientSurname	Number	Type	DoctorNur
1	25.05.2008	22:52:54	8699546010011	Aspirin	3	Murat	Everekli	505110753	ssk	123456
2	23.05.2008	22:05:34	8699536090115	Majezik	2	Murat	Everekli	10321032	ssk	123456
3	23.05.2008	16:04:43	8699536090115	Majezik	2	Murat	Everekli	10321032	ssk	123456
4	23.05.2008	13:39:30	8699546010011	Aspirin	2	Murat	Everekli	10321032	ssk	123456
5	23.05.2008	13:38:51	8699546010011	Aspirin	2	Murat	Everekli		Unfo	
6	22.05.2008	12:12:42	8699536090115	Majezik	2	Berna	Küçük		Unfo	
7	22.05.2008	10:46:16	8699536090115	Majezik	2	Sezin	Everekli		Unfo	
8	22.05.2008	10:36:07	8699536090115	Majezik	1	as	asd		Unfo	

Selling Information

Name of Drug: Aspiris Qantity: 3

Benefit % :2,3 Net Price: 5,66 Total Price+=> 17,565090048

Tax %: 1,12 One of Price : 5,855030016

Barcode	Name	Piece	Volume	Position	Tax	Benefit	Form	Company_N	Group	Type	Info	Price_N	Quantity	Formul
8699546010011	Aspirin	50	ml	K5	1,12	2,3	Tablet	Bayer	Analjezik	20	Tansiyon Has	5,66	81	F

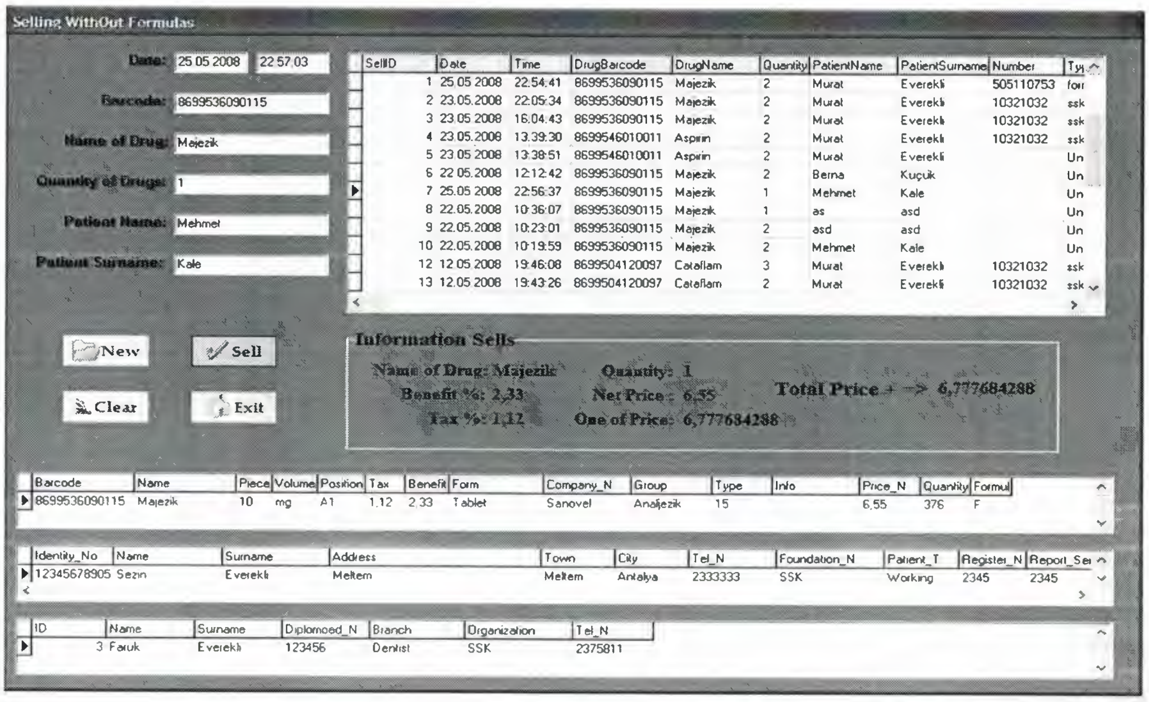
Identity_No	Name	Surname	Address	Town	City	Tel_N	Foundation_N	Patient_T	Register_N	Report_Serial_N
12345678905	Sezin	Everekli	Meltem	Meltem	Antalya	2333333	SSK	Working	2345	2345
43457637912	Ilyas	Unal			Sivas	2334444	SSK	Working	321	321

ID	Name	Surname	Diplomoed_N	Branch	Organization	Tel_N
3	Faruk	Everekli	123456	Dentist	SSK	2375811

Figure 3.2.7.1 Selling with SSK's Formulas

These forms are related with the Selling Table. We used these forms to sell drugs. SellID is the primary key which is the autoincrement. System takes the number automatically when the sell will complete. Date and time are the system times which are automatically fill too. The pharmacist enters the barcode number with using barcode reader, automatically catch the name of drug, tax proportion, benefit proportion, net cost and quantity from drug table after he/she enter quantity. Than entered the patient number by using barcode reader, automatically catch the name and surname of patient. Finally the user enters doctor number and automatically doctor's name and surname entered. After that the users enter the sell button, the system calculate the benefit cost, tax cost, one of price and total cost than send this data to selling table. The user controls this data from the Selling Information area. Selling with formulas form look like Figure 3.2.7.1 . Only one difference is type of data in selling table. Selling with using SSK form fills the type SSK or if selling with using Formulas form fills the type FORM.

3.2.8 Selling without Formulas Forms



Date:	25 05 2008 22 57 03
Barcode:	8699536090115
Name of Drug:	Majezik
Quantity of Drugs:	1
Patient Name:	Mehmet
Patient Surname:	Kale

SellID	Date	Time	DrugBarcode	DrugName	Quantity	PatientName	PatientSurname	Number	Ty
1	25.05 2008	22:54:41	8699536090115	Majezik	2	Murat	Evereklı	505110753	for
2	23.05.2008	22:05:34	8699536090115	Majezik	2	Murat	Evereklı	10321032	ssk
3	23.05 2008	16:04:43	8699536090115	Majezik	2	Murat	Evereklı	10321032	ssk
4	23.05.2008	13:39:30	8699546010011	Asprin	2	Murat	Evereklı	10321032	ssk
5	23.05 2008	13:38:51	8699546010011	Asprin	2	Murat	Evereklı		Un
6	22 05.2008	12:12:42	8699536090115	Majezik	2	Berna	Kuçuk		Un
7	25.05 2008	22:56:37	8699536090115	Majezik	1	Mehmet	Kale		Un
8	22.05.2008	10:36:07	8699536090115	Majezik	1	as	asd		Un
9	22.05.2008	10:23:01	8699536090115	Majezik	2	asd	asd		Un
10	22.05.2008	10:19:59	8699536090115	Majezik	2	Mehmet	Kale		Un
12	12.05 2008	19:46:08	8699504120097	Cataflam	3	Murat	Evereklı	10321032	ssk
13	12.05 2008	19:43:26	8699504120097	Cataflam	2	Murat	Evereklı	10321032	ssk

Information Sells

Name of Drug: Majezik Quantity: 1
Benefit %: 2,33 Net Price : 6,55 Total Price -> 6,777684288
Tax %: 1,12 One of Price: 6,777684288

Barcode	Name	Piece	Volume	Position	Tax	Benefit	Form	Company_N	Group	Type	Info	Price_N	Quantity	Formul
8699536090115	Majezik	10	mg	A1	1,12	2,33	Tablet	Sanovel	Anafjezik	15		6,55	376	F

Identity_No	Name	Surname	Address	Town	City	Tel_N	Foundation_N	Patient_T	Register_N	Report_Se
12345678905	Sezın	Evereklı	Meltem	Meltem	Antalya	2333333	SSK	Working	2345	2345

ID	Name	Surname	Diplomoed_N	Branch	Organization	Tel_N
3	Faruk	Evereklı	123456	Dentist	SSK	2375811

Figure 3.2.8.1 Selling Without Formulas

This form is related with the Selling Table. We used this form to sell without formulas drugs. SellID is the primary key which is the autoincrement. System takes the number automatically when the sell will complete. Date and time are the system times which are automatically fill too. The pharmacist enters the barcode number with using barcode reader, automatically catch the name of drug, tax proportion, benefit proportion, net cost and quantity from drug table after program control this drug is formulas or unformulas. If the drug is formulas, program doesn't permission to sell. (Figure 3.2.8.2) If drug is unformulas than the user can enter the patient name and surname. Finally the users enter the sell button, the system calculate the benefit cost, tax cost, one of price and total cost than send this data to selling table. The user controls this data from the Selling Information area. (Figure 3.2.8.1)

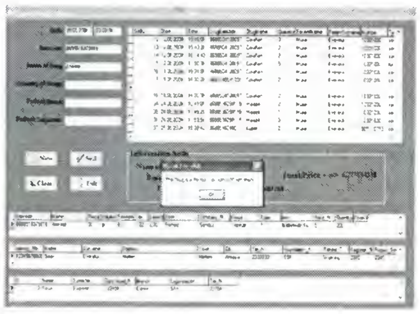


Figure 3.2.8.2 Selling Warning

3.2.9 Account Control Forms

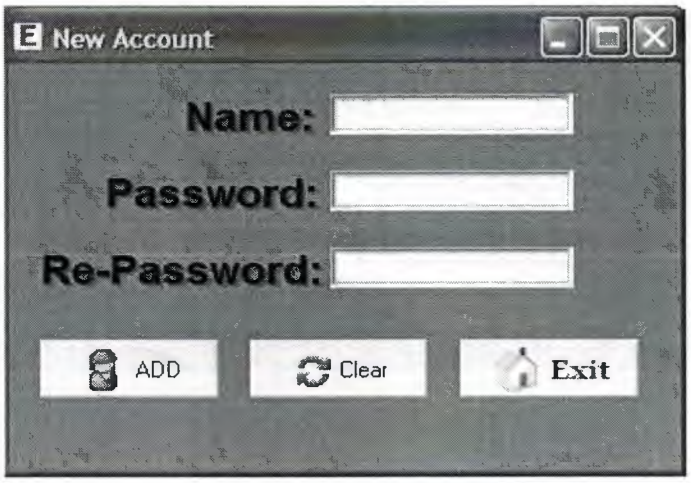
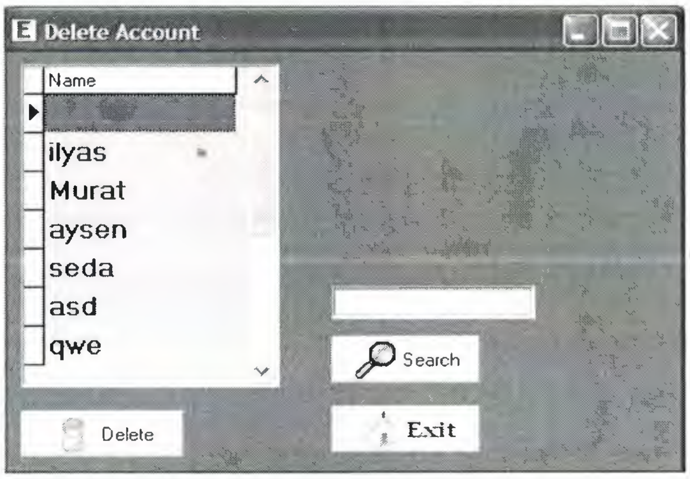


Figure 3.2.9.1 New Account

These forms are related with the Account Table. We used these forms to create a new user or delete the user. To create new account firstly enters username and password. (Figure 3.2.9.1) This form has a password control. Users must be same password to create new account. When ready to use new account, user can logout and other new user enter new username and password to pass the main form.

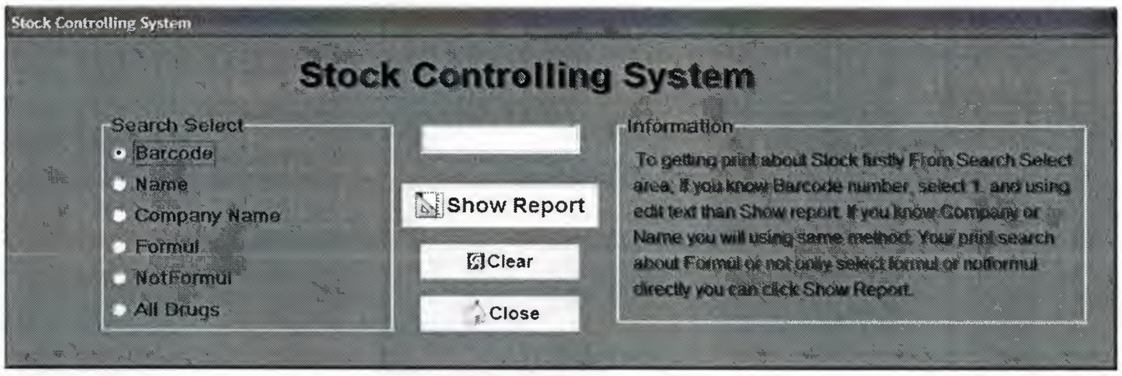


Name
ilyas
Murat
aysen
seda
asd
qwe

Figure 3.2.9.1 Delete Account

Delete account form is used account table. When the users want to delete whoever one, firstly search and find the user and click the delete.

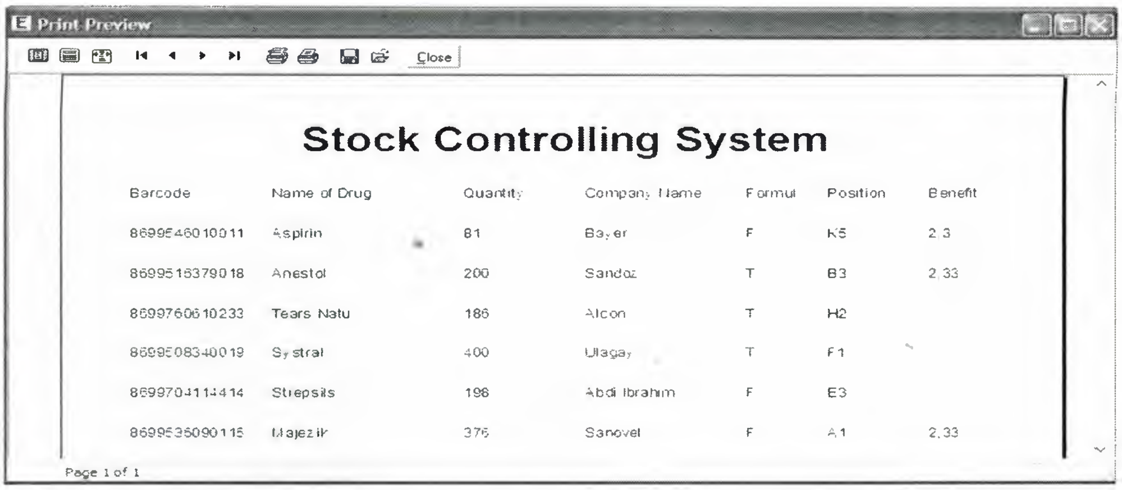
3.2.10 Stock Controlling System Forms



To getting print about Stock firstly From Search Select area; If you know Barcode number, select 1. and using edit text than Show report. If you know Company or Name you will using same method. Your print search about Formul or not only select formul or notformul directly you can click Show Report.

Figure 3.2.10.1 Stock Controlling System

We created this form to controlling the stock. (Figure 3.2.10.1) This form related with the drug tables. Users firstly select the search groups than enter the search parameters. Finally he/she click the Show Report to see report. Users can search with using barcode, name, company name, formulas or all drugs groups. In the Information Area program gives the short information about the how to show report.



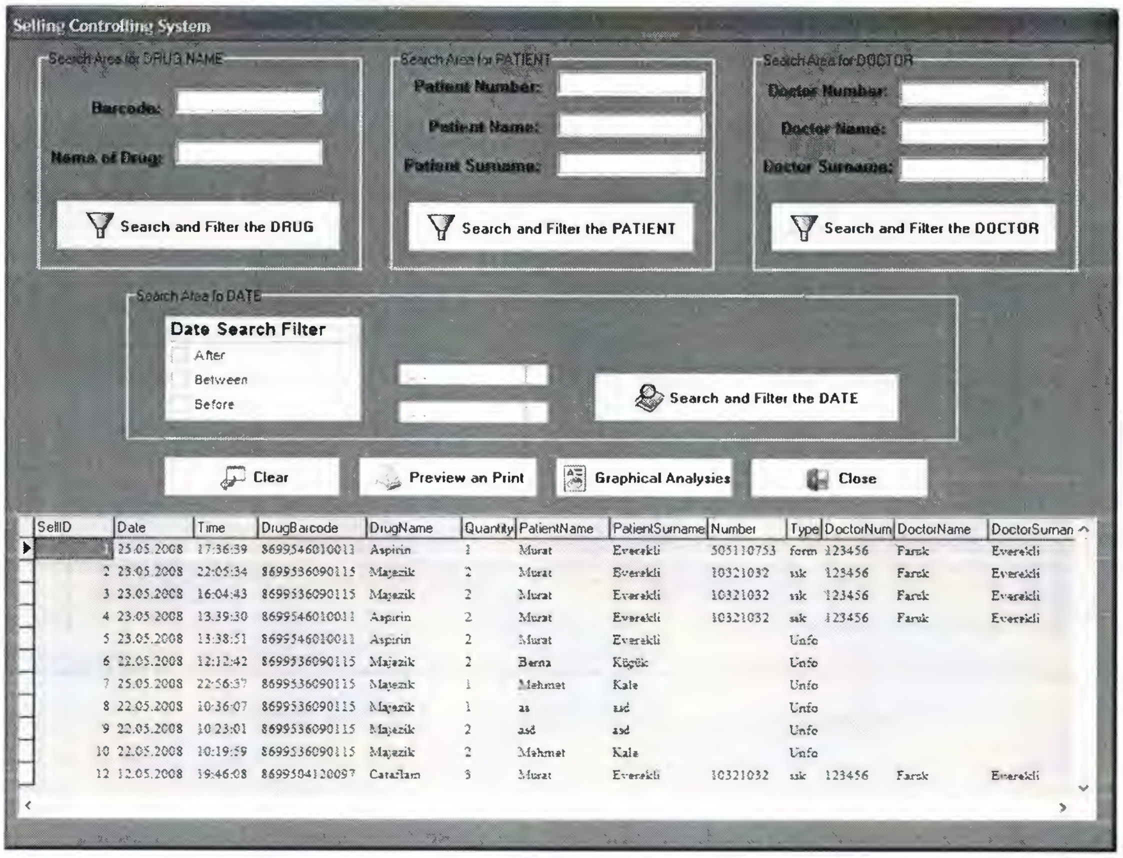
Stock Controlling System

Barcode	Name of Drug	Quantity	Company Name	Formul	Position	Benefit
8699546010011	Aspirin	81	Bayer	F	K5	2,3
8699516379018	Anestol	200	Sandoz	T	B3	2,33
8699760610233	Tears Natu	186	Alcon	T	H2	
8699508340019	Systral	400	Ulagay	T	F1	
8699704114414	Strepsils	196	Abdi Ibrahim	F	E3	
8699535090115	Majezik	376	Sanovel	F	A1	2,33

Figure 3.2.10.2 Stock Controlling System's Report

When the user selects the search groups than click the show report, this form will show (Figure 3.2.10.2) who take the print out this report.

3.2.11 Selling Controlling System Forms



SellID	Date	Time	DrugBarcode	DrugName	Quantity	PatientName	PatientSurname	Number	Type	DoctorNum	DoctorName	DoctorSurnan
1	25.05.2008	17:36:39	8699546010013	Aspirin	1	Murat	Everekli	505110753	form	123456	Faruk	Everekli
2	23.05.2008	22:05:34	8699536090115	Majezik	2	Murat	Everekli	10321032	ssk	123456	Faruk	Everekli
3	23.05.2008	16:04:43	8699536090115	Majezik	2	Murat	Everekli	10321032	ssk	123456	Faruk	Everekli
4	23.05.2008	13:39:30	8699546010011	Aspirin	2	Murat	Everekli	10321032	ssk	123456	Faruk	Everekli
5	23.05.2008	13:38:51	8699546010011	Aspirin	2	Murat	Everekli		Unfo			
6	22.05.2008	12:12:42	8699536090115	Majezik	2	Berna	Küçük		Unfo			
7	25.05.2008	22:56:37	8699536090115	Majezik	1	Mehmet	Kala		Unfo			
8	22.05.2008	10:36:07	8699536090115	Majezik	1	as	asd		Unfo			
9	22.05.2008	10:23:01	8699536090115	Majezik	2	asd	asd		Unfo			
10	22.05.2008	10:19:59	8699536090115	Majezik	2	Mehmet	Kala		Unfo			
12	12.05.2008	19:46:08	8699504120097	Cataflam	3	Murat	Everekli	10321032	ssk	123456	Faruk	Everekli

Figure 3.2.11.1 Selling Controlling System

This form is used to control the sells. User can choose the drug name with using barcode, patient with using patient number or doctor with using doctor number to filter. Than user can date filtered. After date, Between dates or Before date or not select the date and only using first filtered. If the user clicks Graphical Analysis, Graphical View form will show with using filtered conditional. This graph shows the name of drug and how many sells in the filtered date. We used the 3D graphic to give information about the sells.

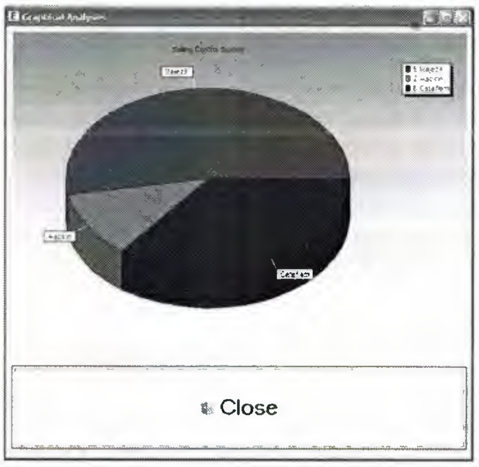
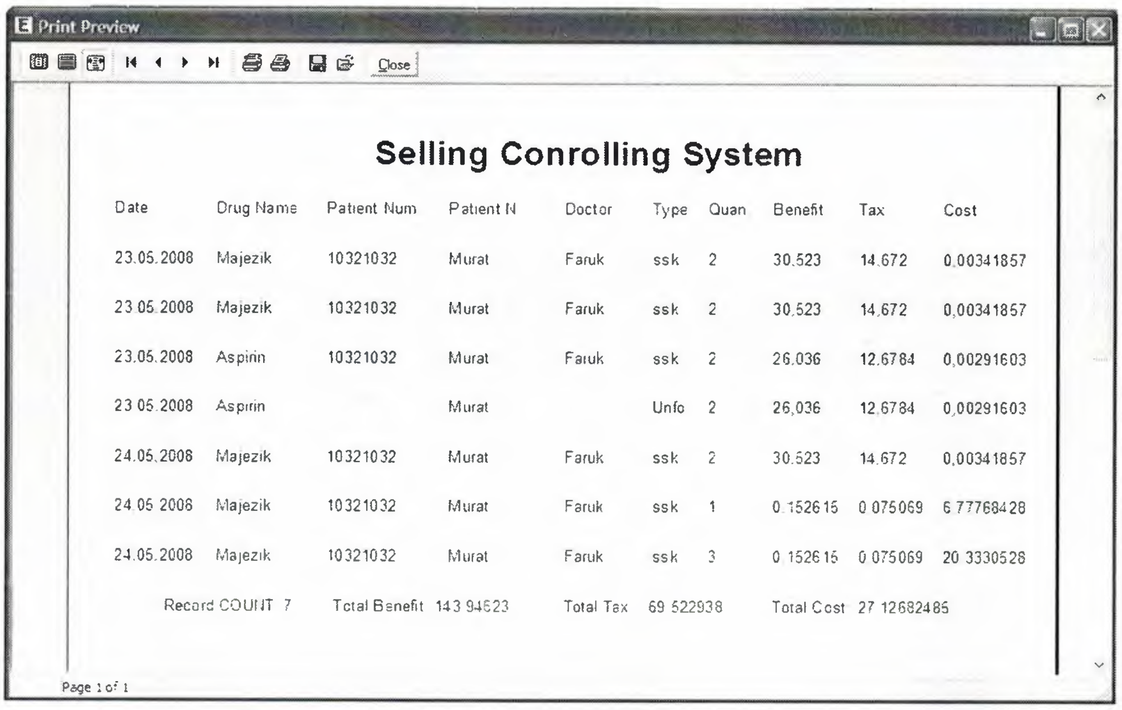


Figure 3.2.11.2 Graphical View


Selling Conrolling System

Date	Drug Name	Patient Num	Patient N	Doctor	Type	Quan	Benefit	Tax	Cost
23.05.2008	Majezik	10321032	Murat	Faruk	ssk	2	30.523	14.672	0,00341857
23 05 2008	Majezik	10321032	Murat	Faruk	ssk	2	30,523	14,672	0,00341857
23.05.2008	Aspirin	10321032	Murat	Faruk	ssk	2	26.036	12.6784	0,00291603
23 05 2008	Aspirin		Murat		Unfo	2	26,036	12,6784	0,00291603
24.05.2008	Majezik	10321032	Murat	Faruk	ssk	2	30.523	14.672	0,00341857
24 05 2008	Majezik	10321032	Murat	Faruk	ssk	1	0 152615	0 075069	6 77768428
24.05.2008	Majezik	10321032	Murat	Faruk	ssk	3	0 152615	0 075069	20 3330528

Record COUNT 7 Total Benefit 143 94623 Total Tax 69 522938 Total Cost 27 12682485

Page 1 of 1

Figure 3.2.11.3 Selling Controlling System Report

If the user clicks the Print Preview, this report will show. (Figure 3.2.11.3) User can print out. Report has Record Count, Total Benefit, Total Tax and Total Cost. This information controlled and filtered by the user.

3.2.12 Tools Forms

These forms are Calculator, Internet Browser and Microsoft Word.

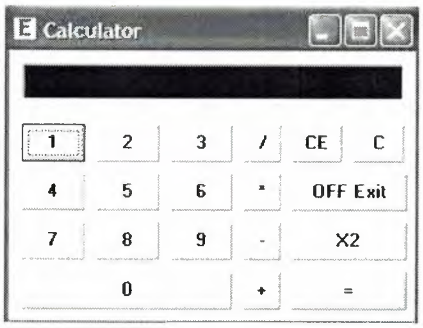


Figure 3.2.12.1 Calculator

Calculator form is simple calculator. (Figure 3.2.12.1) Second form is an Internet Browser. The user can connect to internet. (Figure 3.2.12.2) If the user clicks the Duty Pharmacy button, the program connects the web page about the duty pharmacy. The user controls the duty pharmacy. (Figure 3.2.12.3)

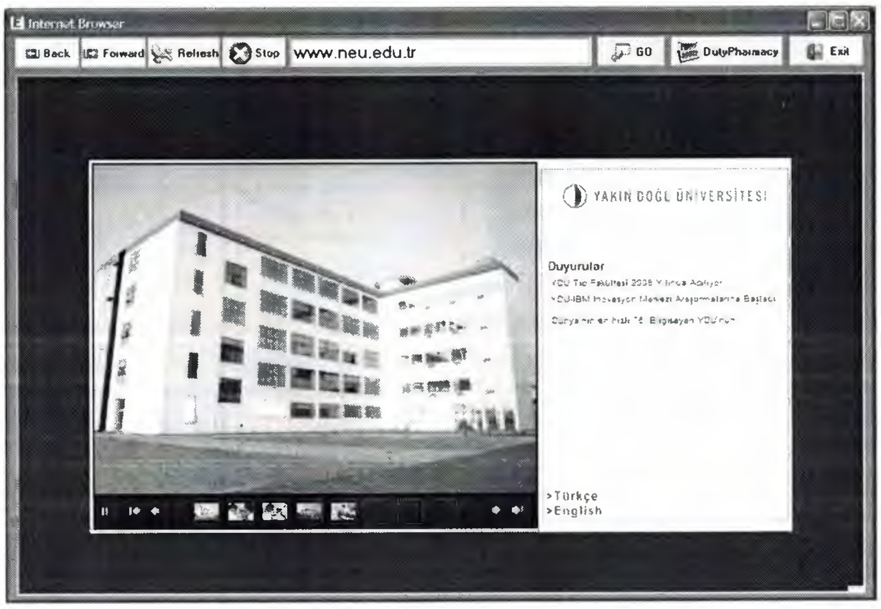
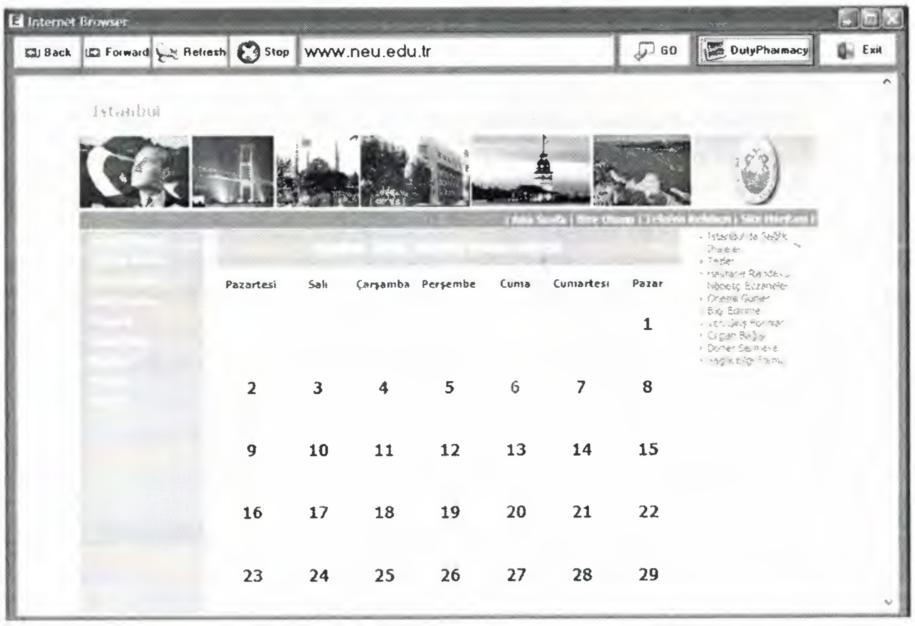


Figure 3.2.12.2 Internet Browser

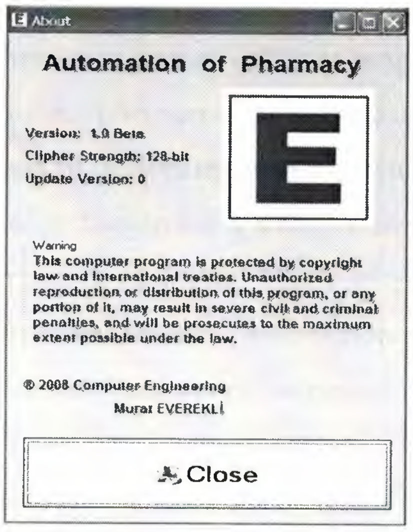


Pazartesi	Salı	Çarşamba	Perşembe	Cuma	Cumartesi	Pazar
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29

Figure 3.2.12.3

3.2.13 About Forms

This area has 2 forms; Help and Version. Version form shows in Figure 3.2.13.1. Second form is about the short information about the program and how to use for helping the users. (Figure 3.2.13.2)

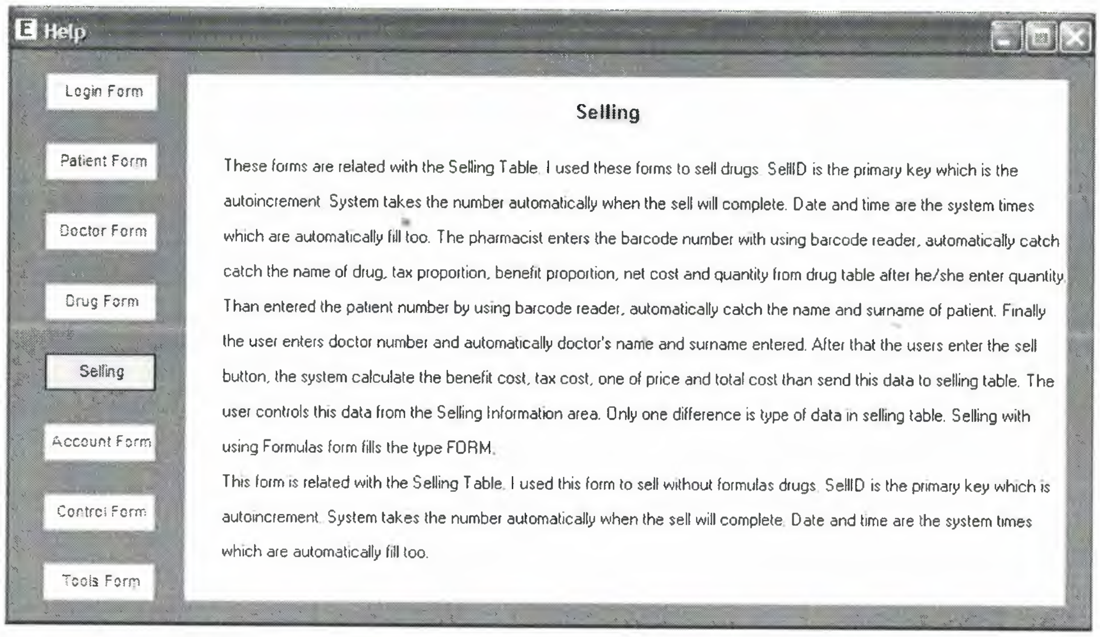


Version: 1.0 Beta
Clipher Strength: 128-bit
Update Version: 0

Warning
This computer program is protected by copyright law and international treaties. Unauthorized reproduction or distribution of this program, or any portion of it, may result in severe civil and criminal penalties, and will be prosecutes to the maximum extent possible under the law.

® 2008 Computer Engineering
Murat EVEREKLİ

Figure 3.2.13.1 Version



Selling

These forms are related with the Selling Table. I used these forms to sell drugs. SellID is the primary key which is the autoincrement. System takes the number automatically when the sell will complete. Date and time are the system times which are automatically fill too. The pharmacist enters the barcode number with using barcode reader, automatically catch catch the name of drug, tax proportion, benefit proportion, net cost and quantity from drug table after he/she enter quantity. Than entered the patient number by using barcode reader, automatically catch the name and surname of patient. Finally the user enters doctor number and automatically doctor's name and surname entered. After that the users enter the sell button, the system calculate the benefit cost, tax cost, one of price and total cost than send this data to selling table. The user controls this data from the Selling Information area. Only one difference is type of data in selling table. Selling with using Formulas form fills the type FORM.

This form is related with the Selling Table. I used this form to sell without formulas drugs. SellID is the primary key which is autoincrement. System takes the number automatically when the sell will complete. Date and time are the system times which are automatically fill too.

Figure 3.2.13.2 Help

CONCLUSION

The main aim of this program was control the stock and selling easily so that I completed this program under the light of my aim.

Delphi has many components that are allows us to write windows programs more than quickly and more easily. I have used businessskin components to have better vision in customer relationship management software program which I have designed. customer relationship management program will facilitate the service of employees who are face to face with customers in businesses such as customer support and marketing. The aim of customer relationship management software to obtain facility in working of user and to give the best service to customer. That's why the program allows user to check all done works with more detailed information about customers and their products. Using the details of customers registered in customer relationship management program, company can contact with customer directly in emergency situations and you can see any changing in stocks quickly. I prefered to use Barcode reader to control to selling and stock more fast.

This program designed for smaller size business companies but it can be developed for large business areas on demand

REFERENCES

[1] Delphi 2007 Handbook by Alister Christie (09 October 2007)

[2] Inside Delphi 2006 by Ivan Hladni (25 November 2005)

[3] Introducing Delphi Programming: Theory through Practice by John Barrow, Linda Miller, Katherine Malan and Helene Gelderblom

[4] Delphi 7 Edition Book by Ihsan Karagülle

[5] http://groups.google.com 'group of Delphi 2007'

[6] http://www.delphiturk.com

[7] http://www.1keydata.com/sql

APPENDIX

Program Code

Form -1- Main Menu:

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, jpeg, ExtCtrls, Menus, ComCtrls, StdCtrls, SkinCtrls, spcalendar,
  SkinBoxCtrls;

type
  TForm1 = class(TForm)
    Image9: TImage;
    MainMenu1: TMainMenu;
    NewEntry1: TMenuItem;
    NewPatient1: TMenuItem;
    NewDoctor1: TMenuItem;
    NewDrog1: TMenuItem;
    OperationStock1: TMenuItem;
    StockEntry1: TMenuItem;
    StockExit1: TMenuItem;
    OperationwithFormula1: TMenuItem;
    OperationWithOutFormula1: TMenuItem;
    OperationWithOutFormula2: TMenuItem;
    SellignWithFormulas1: TMenuItem;
    SellingWithOutFormulas1: TMenuItem;
    Pharmacy1: TMenuItem;
    StockControllingSystem1: TMenuItem;
    SellingControllingSystem1: TMenuItem;
    About1: TMenuItem;
    Help1: TMenuItem;
    Version2: TMenuItem;
    Exit1: TMenuItem;
    StatusBar1: TStatusBar;
    Timer1: TTimer;
    ools1: TMenuItem;
    Calculater1: TMenuItem;
    Calculater2: TMenuItem;
    AccountControl1: TMenuItem;
    NewAccount1: TMenuItem;
    DeleteAccount1: TMenuItem;
    Image1: TImage;
    Image2: TImage;
```

```
 Image3: TImage;
 Image4: TImage;
 Image5: TImage;
 Image6: TImage;
 Image7: TImage;
 Image8: TImage;
 Image10: TImage;
 Image11: TImage;
 Image12: TImage;
 Image13: TImage;
 Image14: TImage;
 Image15: TImage;
 Image16: TImage;
 Image17: TImage;
 Image18: TImage;
 Image19: TImage;
 spSkinShadowLabel1: TspSkinShadowLabel;
 spSkinMonthCalendar1: TspSkinMonthCalendar;
 spSkinButton1: TspSkinButton;
 Word1: TMenuItem;
 procedure Image1MouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
 procedure Image1MouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
 procedure NewPatient1Click(Sender: TObject);
 procedure NewDoctor1Click(Sender: TObject);
 procedure Timer1Timer(Sender: TObject);
 procedure FormActivate(Sender: TObject);
 procedure NewDrog1Click(Sender: TObject);
 procedure StockEntry1Click(Sender: TObject);
 procedure StockExit1Click(Sender: TObject);
 procedure Exit1Click(Sender: TObject);
 procedure NewAccount1Click(Sender: TObject);
 procedure DeleteAccount1Click(Sender: TObject);
 procedure Image4MouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
 procedure Image4MouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
 procedure Image3MouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
 procedure Image3MouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
 procedure Image2MouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
 procedure Image2MouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
 procedure Image5MouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
 procedure Image5MouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
```

```
    procedure Image6MouseDown(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    procedure Image6MouseUp(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    procedure OperationWithOutFormula1Click(Sender: TObject);
    procedure SellignWithFormulas1Click(Sender: TObject);
    procedure Calculater2Click(Sender: TObject);
    procedure Image6Click(Sender: TObject);
    procedure Image4Click(Sender: TObject);
    procedure Image1Click(Sender: TObject);
    procedure Image2Click(Sender: TObject);
    procedure Image3Click(Sender: TObject);
    procedure Image5Click(Sender: TObject);
    procedure SellingWithOutFormulas1Click(Sender: TObject);
    procedure StockControllingSystem1Click(Sender: TObject);
    procedure spSkinButton1Click(Sender: TObject);
    procedure SellingControllingSystem1Click(Sender: TObject);
    procedure Calculater1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure Version1Click(Sender: TObject);
    procedure Word1Click(Sender: TObject);
    procedure Version2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

uses Unit2, Unit3, Unit4, Unit5, Unit6, Unit7, Unit8, Unit9, Unit10,
  Unit11, Unit12, Unit13, Unit14, Unit15, Unit16, Unit17, Unit19;

{$R *.dfm}

procedure TForm1.Image1MouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
image1.Picture:=image14.Picture;
end;

procedure TForm1.Image1MouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
image1.Picture:=image15.Picture;
form2.show;
form1.Hide;
```

```
end;

procedure TForm1.NewPatient1Click(Sender: TObject);
begin
form2.show;
form1.Hide;
end;

procedure TForm1.NewDoctor1Click(Sender: TObject);
begin
form3.show;
form1.Hide;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
function IsCapsLockOn : boolean;
begin Result := 0 <> (GetKeyState(VK_CAPITAL) and $01);
end;
begin
statusbar1.Panels[0].Text:=timetostr(time);
statusbar1.Panels[1].Text:=datetostr(date);
form1.Caption:=copy(caption,2,length(caption)-1)+caption[1];
if(IsCapsLockOn=true)then begin
statusbar1.Panels[2].Text:='Caps ON';
end
else begin
statusbar1.Panels[2].Text:='Caps OFF';
end;
end;

procedure TForm1.FormActivate(Sender: TObject);
begin
statusbar1.Panels[0].Width:=50;
statusbar1.Panels[1].Width:=63;
spskinmonthcalendar1.Date:=(date);
end;

procedure TForm1.NewDrog1Click(Sender: TObject);
begin
form4.show;
form1.Hide;
end;

procedure TForm1.StockEntry1Click(Sender: TObject);
begin
form5.show;
form1.Hide;
end;

procedure TForm1.StockExit1Click(Sender: TObject);
```

```
begin
form6.show;
form1.Hide;
end;

procedure TForm1.Exit1Click(Sender: TObject);
var
x:word;
begin
x:=application.MessageBox('Do you want to EXIT?','EXIT',36);
if(x=IDYES)then
begin
form1.Close;
form7.Close;
end;
end;

procedure TForm1.NewAccount1Click(Sender: TObject);
begin
form8.show;
end;

procedure TForm1.DeleteAccount1Click(Sender: TObject);
begin
form9.show;
end;

procedure TForm1.Image4MouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
image4.Picture:=image7.Picture;
end;

procedure TForm1.Image4MouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
image4.Picture:=image8.Picture;
form5.show;
form1.Hide;
end;

procedure TForm1.Image3MouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
image3.Picture:=image10.Picture;
end;

procedure TForm1.Image3MouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
```

```
image3.Picture:=image11.Picture;
end;

procedure TForm1.Image2MouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
image2.Picture:=image12.Picture;
end;

procedure TForm1.Image2MouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
image2.Picture:=image13.Picture;
end;

procedure TForm1.Image5MouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
image5.Picture:=image16.Picture;
end;

procedure TForm1.Image5MouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
image5.Picture:=image17.Picture;
end;

procedure TForm1.Image6MouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
image6.Picture:=image18.Picture;
end;

procedure TForm1.Image6MouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
image6.Picture:=image19.Picture;
end;

procedure TForm1.OperationWithOutFormula1Click(Sender: TObject);
begin
form10.show;
form1.Hide;
end;

procedure TForm1.SellignWithFormulas1Click(Sender: TObject);
begin
form11.show;
form1.Hide;
end;
```

```
procedure TForm1.Calculater2Click(Sender: TObject);
begin
form12.show;
end;

procedure TForm1.Image6Click(Sender: TObject);
begin
form12.Show;
end;

procedure TForm1.Image4Click(Sender: TObject);
begin
form5.show;
form1.Hide;
end;

procedure TForm1.Image1Click(Sender: TObject);
begin
form2.show;
form1.Hide;
end;

procedure TForm1.Image2Click(Sender: TObject);
begin
form10.show;
form1.Hide;
end;

procedure TForm1.Image3Click(Sender: TObject);
begin
form11.show;
form1.Hide;
end;

procedure TForm1.Image5Click(Sender: TObject);
begin
form8.show;
form1.Hide;
end;

procedure TForm1.SellingWithOutFormulas1Click(Sender: TObject);
begin
form13.show;
form1.Hide;
end;

procedure TForm1.StockControllingSystem1Click(Sender: TObject);
begin
form14.show;
```

```
end;

procedure TForm1.spSkinButton1Click(Sender: TObject);
begin
form1.Close;
form7.Show;
end;

procedure TForm1.SellingControllingSystem1Click(Sender: TObject);
begin
form15.show;
form1.Hide;
end;

procedure TForm1.Calculater1Click(Sender: TObject);
begin
form16.show;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
BorderIcons := BorderIcons - [BiSystemMenu];
end;

procedure TForm1.Version1Click(Sender: TObject);
begin
form17.show;
end;

procedure TForm1.Word1Click(Sender: TObject);
begin
winExec('c:\windows\notepad.exe',SW_SHOW);
end;

procedure TForm1.Version2Click(Sender: TObject);
begin
form19.show;
end;

end.
```

Form -2-   New Patient:

```
unit Unit2;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, DBCtrls, Mask, Buttons, ExtCtrls, Grids, DBGrids, DB,
  DBTables, SkinCtrls;

type
  TForm2 = class(TForm)
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    Label7: TLabel;
    Label8: TLabel;
    Label9: TLabel;
    Label10: TLabel;
    Label11: TLabel;
    DBEdit1: TDBEdit;
    DBEdit2: TDBEdit;
    DBEdit3: TDBEdit;
    DBEdit4: TDBEdit;
    DBEdit5: TDBEdit;
    DBEdit6: TDBEdit;
    DBEdit7: TDBEdit;
    DBComboBox1: TDBComboBox;
    DBComboBox2: TDBComboBox;
    DBEdit8: TDBEdit;
    DBEdit9: TDBEdit;
    RadioGroup1: TRadioGroup;
    Edit1: TEdit;
    Label12: TLabel;
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    BitBtn3: TBitBtn;
    BitBtn4: TBitBtn;
    BitBtn5: TBitBtn;
    DBGrid1: TDBGrid;
    GroupBox1: TGroupBox;
    Label13: TLabel;
    DataSource1: TDataSource;
    Query1: TQuery;
    BitBtn6: TBitBtn;
    procedure DBEdit1Click(Sender: TObject);
    procedure DBEdit2Click(Sender: TObject);
    procedure DBEdit3Click(Sender: TObject);
    procedure DBEdit4Click(Sender: TObject);
    procedure DBEdit5Click(Sender: TObject);
    procedure DBEdit6Click(Sender: TObject);
    procedure DBEdit7Click(Sender: TObject);
    procedure DBComboBox1Click(Sender: TObject);
    procedure DBComboBox2Click(Sender: TObject);
    procedure DBEdit8Click(Sender: TObject);
    procedure DBEdit9Click(Sender: TObject);
    procedure Edit1Click(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
    procedure BitBtn3Click(Sender: TObject);
    procedure BitBtn4Click(Sender: TObject);
    procedure BitBtn5Click(Sender: TObject);
    procedure BitBtn6Click(Sender: TObject);
    procedure DBEdit1KeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit2KeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit3KeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit4KeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit5KeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit6KeyPress(Sender: TObject; var Key: Char);
    procedure DBComboBox1KeyPress(Sender: TObject; var Key: Char);
    procedure DBComboBox2KeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit8KeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit9KeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit7KeyPress(Sender: TObject; var Key: Char);
    procedure Edit1Change(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
    procedure FormKeyPress(Sender: TObject; var Key: Char);
    procedure FormActivate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form2: TForm2;

implementation

uses Unit1;

{$R *.dfm}

procedure TForm2.DBEdit1Click(Sender: TObject);
begin
label13.Caption:='Enter the National number. Ex=> T.C. Identity Number';
end;
```

```
procedure TForm2.DBEdit2Click(Sender: TObject);
begin
label13.Caption:='Enter the Name of Patient.';
end;

procedure TForm2.DBEdit3Click(Sender: TObject);
begin
label13.Caption:='Enter the Surname of Patient.';
end;

procedure TForm2.DBEdit4Click(Sender: TObject);
begin
label13.Caption:='Enter the Address'#39's of Patient';
end;

procedure TForm2.DBEdit5Click(Sender: TObject);
begin
label13.Caption:='Enter the Town'#39's of Patient';
end;

procedure TForm2.DBEdit6Click(Sender: TObject);
begin
label13.Caption:='Enter the City'#39's of Patient';
end;

procedure TForm2.DBEdit7Click(Sender: TObject);
begin
label13.Caption:='Enter the Telefon Number.';
end;

procedure TForm2.DBComboBox1Click(Sender: TObject);
begin
label13.Caption:='Enter the Fundation Name of Patient. 3 alternative to choose';
end;

procedure TForm2.DBComboBox2Click(Sender: TObject);
begin
label13.Caption:='Enter the Patient Type';
end;

procedure TForm2.DBEdit8Click(Sender: TObject);
begin
label13.Caption:='Enter the Patient'#36's Register Number';
end;

procedure TForm2.DBEdit9Click(Sender: TObject);
begin
label13.Caption:='Enter the Report Serial Number';
end;
```

```
procedure TForm2.Edit1Click(Sender: TObject);
begin
label13.Caption:='Select the data than enter the search word!';
end;

procedure TForm2.BitBtn2Click(Sender: TObject);
begin
dbedit1.Text:='';
dbedit2.Text:='';
dbedit3.Text:='';
dbedit4.Text:='';
dbedit5.Text:='';
dbedit6.Text:='';
dbedit7.Text:='';
dbedit8.Text:='';
dbedit9.Text:='';
dbcombobox1.Text:='';
dbcombobox2.Text:='';
dbedit1.SetFocus;
query1.Insert;
end;

procedure TForm2.BitBtn3Click(Sender: TObject);
begin
query1.Post;
end;

procedure TForm2.BitBtn4Click(Sender: TObject);
begin
query1.Edit;
dbedit1.SetFocus;
end;

procedure TForm2.BitBtn5Click(Sender: TObject);
var
x:word;
begin
x:=application.MessageBox('Do you want to DELETE?','DELETE',36);
if(x=IDYES)then
begin
query1.Delete;
end;
end;
procedure TForm2.BitBtn6Click(Sender: TObject);
begin
form1.Show;
form2.Close;
end;
```

```
procedure TForm2.DBEdit1KeyPress(Sender: TObject; var Key: Char);
begin
if(key=#13)then dbedit2.SetFocus;
end;

procedure TForm2.DBEdit2KeyPress(Sender: TObject; var Key: Char);
begin
if(key=#13)then dbedit3.SetFocus;
end;

procedure TForm2.DBEdit3KeyPress(Sender: TObject; var Key: Char);
begin
if(key=#13)then dbedit4.SetFocus;
end;

procedure TForm2.DBEdit4KeyPress(Sender: TObject; var Key: Char);
begin
if(key=#13)then dbedit5.SetFocus;
end;

procedure TForm2.DBEdit5KeyPress(Sender: TObject; var Key: Char);
begin
if(key=#13)then dbedit6.SetFocus;
end;

procedure TForm2.DBEdit6KeyPress(Sender: TObject; var Key: Char);
begin
if(key=#13)then dbedit7.SetFocus;
end;

procedure TForm2.DBComboBox1KeyPress(Sender: TObject; var Key: Char);
begin
if(key=#13)then dbcombobox2.SetFocus;
end;

procedure TForm2.DBComboBox2KeyPress(Sender: TObject; var Key: Char);
begin
if(key=#13)then dbedit8.SetFocus;
end;

procedure TForm2.DBEdit8KeyPress(Sender: TObject; var Key: Char);
begin
if(key=#13)then dbedit9.SetFocus;
end;

procedure TForm2.DBEdit9KeyPress(Sender: TObject; var Key: Char);
begin
if(key=#13)then bitbtn3.SetFocus;
end;
```

```
procedure TForm2.DBEdit7KeyPress(Sender: TObject; var Key: Char);
begin
if(key=#13)then dbcombobox1.SetFocus;
end;

procedure TForm2.Edit1Change(Sender: TObject);
begin
if(radiogroup1.ItemIndex=0)then
begin
query1.Close;
query1.SQL.Clear;
query1.SQL.Add('select * from patient_tb where identity_no
like'+#39+(edit1.Text)+'%'+#39);
query1.Open;
end;
if(radiogroup1.ItemIndex=1)then
begin
query1.Close;
query1.SQL.Clear;
query1.SQL.Add('select * from patient_tb where name
like'+#39+(edit1.Text)+'%'+#39);
query1.Open;
end;
if(radiogroup1.ItemIndex=2)then
begin
query1.Close;
query1.SQL.Clear;
query1.SQL.Add('select * from patient_tb where surname
like'+#39+(edit1.Text)+'%'+#39);
query1.Open;
end;
if(radiogroup1.ItemIndex=3)then
begin
query1.Close;
query1.SQL.Clear;
query1.SQL.Add('select * from patient_tb where city like'+#39+(edit1.Text)+'%'+#39);
query1.Open;
end;
if(radiogroup1.ItemIndex=4)then
begin
query1.Close;
query1.SQL.Clear;
query1.SQL.Add('select * from patient_tb where foundation_n
like'+#39+(edit1.Text)+'%'+#39);
query1.Open;
end;
if(radiogroup1.ItemIndex=5)then
begin
query1.Close;
query1.SQL.Clear;
```

```
query1.SQL.Add('select * from patient_tb where patient_T
like'+#39+(edit1.Text)+'%'+#39);
query1.Open;
end;
if(radiogroup1.ItemIndex=6)then
begin
query1.Close;
query1.SQL.Clear;
query1.SQL.Add('select * from patient_tb where Register_N
like'+#39+(edit1.Text)+'%'+#39);
query1.Open;
end;
if(radiogroup1.ItemIndex=7)then
begin
query1.Close;
query1.SQL.Clear;
query1.SQL.Add('select * from patient_tb where Report_Serial_N
like'+#39+(edit1.Text)+'%'+#39);
query1.Open;
end;
end;

procedure TForm2.BitBtn1Click(Sender: TObject);
begin
edit1.Text:='';
query1.Refresh;
radiogroup1.ItemIndex:=-1;
end;

procedure TForm2.FormKeyPress(Sender: TObject; var Key: Char);
begin
If (Key = #13) then
 begin
   key := #0;
   Perform(WM_NEXTDLGCTL, 0, 0);
  end;
end;
procedure TForm2.FormActivate(Sender: TObject);
begin
query1.Insert;
end;

end.
```

Form -3- New Doctor

```
unit Unit3;

interface

uses
 Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
 Dialogs, StdCtrls, Grids, DBGrids, Buttons, ExtCtrls, DBCtrls, Mask, DB,
 DBTables, SkinCtrls;

type
 TForm3 = class(TForm)
  Label1: TLabel;
  Label2: TLabel;
  Label3: TLabel;
  Label4: TLabel;
  Label5: TLabel;
  Label6: TLabel;
  DBEdit1: TDBEdit;
  DBEdit2: TDBEdit;
  DBEdit3: TDBEdit;
  DBEdit4: TDBEdit;
  DBComboBox1: TDBComboBox;
  DBComboBox2: TDBComboBox;
  RadioGroup1: TRadioGroup;
  Edit1: TEdit;
  Label12: TLabel;
  BitBtn1: TBitBtn;
  BitBtn2: TBitBtn;
  BitBtn3: TBitBtn;
  BitBtn4: TBitBtn;
  BitBtn5: TBitBtn;
  DBGrid1: TDBGrid;
  GroupBox1: TGroupBox;
  Label13: TLabel;
  Query1: TQuery;
  DataSource1: TDataSource;
  Label7: TLabel;
  DBEdit5: TDBEdit;
  BitBtn6: TBitBtn;
  procedure DBEdit1Click(Sender: TObject);
  procedure DBEdit2Click(Sender: TObject);
  procedure DBEdit3Click(Sender: TObject);
  procedure DBComboBox1Click(Sender: TObject);
  procedure DBComboBox2Click(Sender: TObject);
  procedure DBEdit4Click(Sender: TObject);
  procedure Edit1Change(Sender: TObject);
  procedure BitBtn2Click(Sender: TObject);
  procedure BitBtn3Click(Sender: TObject);
```

```
    procedure BitBtn5Click(Sender: TObject);
    procedure BitBtn4Click(Sender: TObject);
    procedure BitBtn6Click(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
    procedure spSkinButton1Click(Sender: TObject);
    procedure FormActivate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form3: TForm3;

implementation

uses Unit1, Unit2;

{$R *.dfm}

procedure TForm3.DBEdit1Click(Sender: TObject);
begin
label13.Caption:='Enter the Name'#39's of Doctor.';
end;

procedure TForm3.DBEdit2Click(Sender: TObject);
begin
label13.Caption:='Enter the Surname'#39's of Doctor.';
end;

procedure TForm3.DBEdit3Click(Sender: TObject);
begin
label13.Caption:='Enter the Diplomaed numbers'#39's of Doctor. Contact the Doctor!';
end;

procedure TForm3.DBComboBox1Click(Sender: TObject);
begin
label13.Caption:='Select the Brach'#39's of Doctor.';
end;

procedure TForm3.DBComboBox2Click(Sender: TObject);
begin
label13.Caption:='Select the Organization'#39's of Doctor.';
end;

procedure TForm3.DBEdit4Click(Sender: TObject);
begin
label13.Caption:='Enter the Tel Number'#39's of Doctor.';
end;
```

```
procedure TForm3.Edit1Change(Sender: TObject);
begin
if(radiogroup1.ItemIndex=0)then
begin
query1.Close;
query1.SQL.Clear;
query1.SQL.Add('select * from Doctor_tb where ID like'+#39+(edit1.Text)+'%'+#39);
query1.Open;
end;

if(radiogroup1.ItemIndex=1)then
begin
query1.Close;
query1.SQL.Clear;
query1.SQL.Add('select * from Doctor_tb where Name
like'+#39+(edit1.Text)+'%'+#39);
query1.Open;
end;

if(radiogroup1.ItemIndex=2)then
begin
query1.Close;
query1.SQL.Clear;
query1.SQL.Add('select * from Doctor_tb where Surname
like'+#39+(edit1.Text)+'%'+#39);
query1.Open;
end;

if(radiogroup1.ItemIndex=3)then
begin
query1.Close;
query1.SQL.Clear;
query1.SQL.Add('select * from Doctor_tb where Diplomoed_N
like'+#39+(edit1.Text)+'%'+#39);
query1.Open;
end;

if(radiogroup1.ItemIndex=0)then
begin
query1.Close;
query1.SQL.Clear;
query1.SQL.Add('select * from Doctor_tb where Branch
like'+#39+(edit1.Text)+'%'+#39);
query1.Open;
end;

if(radiogroup1.ItemIndex=0)then
begin
query1.Close;
```

```
query1.SQL.Clear;
query1.SQL.Add('select * from Doctor_tb where Organization
like'+#39+(edit1.Text)+'%'+#39);
query1.Open;
end;

end;

procedure TForm3.BitBtn2Click(Sender: TObject);
begin
dbedit1.Text:='';
dbedit2.Text:='';
dbedit3.Text:='';
dbedit4.Text:='';
dbedit5.Text:='';
dbcombobox1.Text:='';
dbcombobox2.Text:='';
dbedit1.SetFocus;
query1.Insert;
end;

procedure TForm3.BitBtn3Click(Sender: TObject);
begin
query1.Post;
end;

procedure TForm3.BitBtn5Click(Sender: TObject);
var
x:word;
begin
x:=application.MessageBox('Do you want to DELETE?','DELETE',36);
if(x=IDYES)then
begin
query1.Delete;
end;

end;
procedure TForm3.BitBtn4Click(Sender: TObject);
begin
query1.Edit;
dbedit1.SetFocus;
end;

procedure TForm3.BitBtn6Click(Sender: TObject);
begin
form3.Close;
form1.show;
end;

procedure TForm3.BitBtn1Click(Sender: TObject);
```

```
begin
edit1.Text:='';
query1.Refresh;
radiogroup1.ItemIndex:=-1;
end;

procedure TForm3.spSkinButton1Click(Sender: TObject);
begin
form3.close;
form1.Show;
end;

procedure TForm3.FormActivate(Sender: TObject);
begin
query1.Insert;
end;

end.
```

Form -4- New Drug

```
unit Unit4;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Grids, DBGrids, StdCtrls, DBCtrls, Mask, Buttons, ExtCtrls, DB,
  DBTables, SkinCtrls;

type
  TForm4 = class(TForm)
    Label1: TLabel;
    GroupBox1: TGroupBox;
    Label13: TLabel;
    RadioGroup1: TRadioGroup;
    BitBtn2: TBitBtn;
    BitBtn3: TBitBtn;
    BitBtn4: TBitBtn;
    BitBtn5: TBitBtn;
    BitBtn1: TBitBtn;
    Edit1: TEdit;
    DBEdit1: TDBEdit;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    Label8: TLabel;
    Label9: TLabel;
    Label10: TLabel;
    Label11: TLabel;
    Label12: TLabel;
    DBEdit2: TDBEdit;
    DBEdit3: TDBEdit;
    DBComboBox1: TDBComboBox;
    DBEdit4: TDBEdit;
    DBComboBox2: TDBComboBox;
    DBComboBox3: TDBComboBox;
    DBComboBox4: TDBComboBox;
    DBEdit5: TDBEdit;
    DBComboBox5: TDBComboBox;
    DBMemo1: TDBMemo;
    Label14: TLabel;
    DBGrid1: TDBGrid;
    Query1: TQuery;
    DataSource1: TDataSource;
    Label7: TLabel;
    Label15: TLabel;
    DBEdit6: TDBEdit;
    DBEdit7: TDBEdit;
    Label16: TLabel;
    DBCheckBox1: TDBCheckBox;
    DBEdit8: TDBEdit;
    BitBtn6: TBitBtn;
    procedure BitBtn2Click(Sender: TObject);
    procedure BitBtn3Click(Sender: TObject);
    procedure BitBtn4Click(Sender: TObject);
    procedure BitBtn5Click(Sender: TObject);
    procedure BitBtn6Click(Sender: TObject);
    procedure Edit1Change(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
    procedure RadioGroup1Click(Sender: TObject);
    procedure DBEdit1Click(Sender: TObject);
    procedure DBEdit2Click(Sender: TObject);
    procedure DBEdit3Click(Sender: TObject);
    procedure DBComboBox1Click(Sender: TObject);
    procedure DBEdit4Click(Sender: TObject);
    procedure DBComboBox2Click(Sender: TObject);
    procedure DBComboBox3Click(Sender: TObject);
    procedure DBComboBox4Click(Sender: TObject);
    procedure DBEdit5Click(Sender: TObject);
    procedure DBComboBox5Click(Sender: TObject);
    procedure DBMemo1Click(Sender: TObject);
    procedure spSkinButton1Click(Sender: TObject);
    procedure FormActivate(Sender: TObject);
    procedure DBEdit8Click(Sender: TObject);
    procedure DBEdit6Click(Sender: TObject);
    procedure DBEdit7Click(Sender: TObject);
    procedure DBCheckBox1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form4: TForm4;

implementation

uses Unit1;

{$R *.dfm}

procedure TForm4.BitBtn2Click(Sender: TObject);
begin
dbedit1.Text:='';
dbedit2.Text:='';
```

```
dbedit3.Text:='';
dbedit4.Text:='';
dbedit5.Text:='';
dbcombobox1.Text:='';
dbcombobox2.Text:='';
dbcombobox3.Text:='';
dbcombobox4.Text:='';
dbcombobox5.Text:='';
dbedit6.Text:='';
dbmemo1.Text:='';
dbedit1.SetFocus;
query1.Insert;
end;

procedure TForm4.BitBtn3Click(Sender: TObject);
begin
if dbedit7.Text=''then begin
dbedit7.Text:='0';
end;
query1.Post;
end;

procedure TForm4.BitBtn4Click(Sender: TObject);
begin
query1.Edit;
dbedit1.SetFocus;
end;

procedure TForm4.BitBtn5Click(Sender: TObject);
var
x:word;
begin
x:=application.MessageBox('Do you want to DELETE?','DELETE',36);
if(x=IDYES)then
begin
query1.Delete;
end;
end;
procedure TForm4.BitBtn6Click(Sender: TObject);
begin
form4.Close;
form1.show;
end;

procedure TForm4.Edit1Change(Sender: TObject);
begin
if(radiogroup1.ItemIndex=0)then
begin
query1.Close;
query1.SQL.Clear;
```

```
query1.SQL.Add('select * from Drug_tb where Barcode
like'+#39+(edit1.Text)+'%'+#39);
query1.Open;
end;

if(radiogroup1.ItemIndex=1)then
begin
query1.Close;
query1.SQL.Clear;
query1.SQL.Add('select * from Drug_tb where Name like'+#39+(edit1.Text)+'%'+#39);
query1.Open;
end;

if(radiogroup1.ItemIndex=2)then
begin
query1.Close;
query1.SQL.Clear;
query1.SQL.Add('select * from Drug_tb where Company_Na
like'+#39+(edit1.Text)+'%'+#39);
query1.Open;
end;

end;
procedure TForm4.BitBtn1Click(Sender: TObject);
begin
edit1.Text:='';
query1.Refresh;
radiogroup1.ItemIndex:=-1;
end;

procedure TForm4.RadioGroup1Click(Sender: TObject);
begin
edit1.SetFocus;
end;

procedure TForm4.DBEdit1Click(Sender: TObject);
begin
label13.Caption:='Enter the Barcode Number to use Barcode Reader';
end;

procedure TForm4.DBEdit2Click(Sender: TObject);
begin
label13.Caption:='Enter the Name of Drug';
end;

procedure TForm4.DBEdit3Click(Sender: TObject);
begin
label13.Caption:='Enter the Diklokenak Volume';
end;
```

```
procedure TForm4.DBComboBox1Click(Sender: TObject);
begin
label13.Caption:='Choose the Volume';
end;

procedure TForm4.DBEdit4Click(Sender: TObject);
begin
label13.Caption:='Enter the position in your Pharmacy';
end;

procedure TForm4.DBComboBox2Click(Sender: TObject);
begin
label13.Caption:='Choose the Tax Propartion';
end;

procedure TForm4.DBComboBox3Click(Sender: TObject);
begin
label13.Caption:='Choose the Propartion of Benefit';
end;

procedure TForm4.DBComboBox4Click(Sender: TObject);
begin
label13.Caption:='Choose the Pharmaceutical Form';
end;

procedure TForm4.DBEdit5Click(Sender: TObject);
begin
label13.Caption:='Enter the Company Name';
end;

procedure TForm4.DBComboBox5Click(Sender: TObject);
begin
label13.Caption:='Choose the Drog#39s Group';
end;

procedure TForm4.DBMemo1Click(Sender: TObject);
begin
label13.Caption:='Enter the General Information';
end;

procedure TForm4.spSkinButton1Click(Sender: TObject);
begin
form4.Close;
form1.Show;
end;

procedure TForm4.FormActivate(Sender: TObject);
begin
query1.Insert;
end;
```

```
procedure TForm4.DBEdit8Click(Sender: TObject);
begin
label13.Caption:='Number of piace in box';
end;

procedure TForm4.DBEdit6Click(Sender: TObject);
begin
label13.Caption:='Enter the net Price of the Drog#39s';
end;

procedure TForm4.DBEdit7Click(Sender: TObject);
begin
label13.Caption:='Enter the quantity of the Drog#39s';
end;

procedure TForm4.DBCheckBox1Click(Sender: TObject);
begin
label13.Caption:='Important this drug is formal or unformal!!';
end;

end.
```

Form -5- Stock Entry

```
unit Unit5;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, DB, DBTables, StdCtrls, Grids, DBGrids, Buttons, Mask, DBCtrls,
  SkinCtrls;

type
  TForm5 = class(TForm)
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    BitBtn1: TBitBtn;
    BitBtn3: TBitBtn;
    GroupBox1: TGroupBox;
    Label13: TLabel;
    Edit1: TEdit;
    Query2: TQuery;
    DataSource2: TDataSource;
    DataSource1: TDataSource;
    DBGrid2: TDBGrid;
    DBEdit3: TDBEdit;
    Table1: TTable;
    Label4: TLabel;
    Edit2: TEdit;
    DBGrid1: TDBGrid;
    spSkinShadowLabel1: TspSkinShadowLabel;
    Edit3: TEdit;
    BitBtn6: TBitBtn;
    procedure BitBtn3Click(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
    procedure BitBtn6Click(Sender: TObject);
    procedure Edit2Change(Sender: TObject);
    procedure FormActivate(Sender: TObject);
    procedure Edit2KeyPress(Sender: TObject; var Key: Char);
    procedure spSkinButton1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form5: TForm5;

implementation
```

```
uses Unit4, Unit1;

{$R *.dfm}


procedure TForm5.BitBtn3Click(Sender: TObject);
var
x,y,z: integer;
begin
if edit1.Text=''then
showmessage ('Please Enter Quantity')
else
begin
table1.Locate('barcode',edit2.Text,[loPartialKey,loCaseInsensitive]);
x:= strtoint(edit1.Text);
y:= strtoint(dbedit3.Text);
z:=x+y;
label4.Caption:=inttostr(z);
table1.Edit;
table1.FieldValues['Quantity']:=label4.Caption;
table1.Refresh;
table1.Next;
end;
end;

procedure TForm5.BitBtn1Click(Sender: TObject);
begin
Query2.Refresh;
edit3.Clear;
edit1.Clear;
edit2.Clear;
end;

procedure TForm5.BitBtn6Click(Sender: TObject);
begin
form5.Close;
form1.Show;
end;

procedure TForm5.Edit2Change(Sender: TObject);
var
num:integer;
begin
Query2.Close;
Query2.SQL.Clear;
Query2.SQL.Add('select Barcode,Name from drug_tb1 where barcode
like'#39+(edit2.Text)+'%'+#39);
Query2.Open;
edit3.Text:=DBGrid1.Fields[1].AsString;
```

```
if(Query2.RecordCount=0)then
begin
num:=Application.MessageBox('Do you want to turn New Drug?','This Drug Not
Found',
MB_YESNOCANCEL+MB_ICONQUESTION);
if num=mrYes then begin
form5.Close;
form4.Show;
end;
if num=mrNO then begin
edit2.Text:='';
edit1.Text:='';
edit3.Text:='';
edit2.SetFocus;
end;
end;
if edit2.Text=''then
edit3.Text:=''
else begin
table1.Filtered:=false;
table1.Filter:='Barcode='+edit2.Text;
table1.Filtered:=true;
end;
end;

procedure TForm5.FormActivate(Sender: TObject);
begin
edit1.Text:='';
edit2.Text:='';
edit3.Text:='';
edit2.SetFocus;
end;

procedure TForm5.Edit2KeyPress(Sender: TObject; var Key: Char);
begin
if(key=#13)then edit1.SetFocus;
end;

procedure TForm5.spSkinButton1Click(Sender: TObject);
begin
form5.Close;
form1.Show;
end;

end.
```

Form -6- Stock Exit

```
unit Unit6;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Buttons, Mask, DBCtrls, ExtCtrls, Grids, DBGrids, DB,
  DBTables, SkinCtrls;

type
  TForm6 = class(TForm)
    DataSource1: TDataSource;
    DBGrid1: TDBGrid;
    BitBtn1: TBitBtn;
    BitBtn3: TBitBtn;
    Label4: TLabel;
    Label6: TLabel;
    Label7: TLabel;
    Label8: TLabel;
    GroupBox2: TGroupBox;
    Label9: TLabel;
    DBEdit5: TDBEdit;
    Table1: TTable;
    Edit1: TEdit;
    Edit2: TEdit;
    spSkinShadowLabel1: TspSkinShadowLabel;
    Edit3: TEdit;
    BitBtn6: TBitBtn;
    procedure BitBtn3Click(Sender: TObject);
    procedure Edit2Change(Sender: TObject);
    procedure FormActivate(Sender: TObject);
    procedure BitBtn6Click(Sender: TObject);
    procedure spSkinButton1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form6: TForm6;

implementation

uses Unit5, Unit1;

{$R *.dfm}
```

```
procedure TForm6.BitBtn3Click(Sender: TObject);
var
x,y,z: integer;
begin
if edit1.Text=''then
showmessage ('Please Enter Quantity')
else
begin
table1.Locate('barcode',edit2.Text,[loPartialKey,loCaseInsensitive]);
x:= strtoint(edit1.Text);
y:= strtoint(dbedit5.Text);
z:=Y-X;
label8.Caption:=inttostr(z);
table1.Edit;
table1.FieldValues['Quantity']:=label8.Caption;
table1.Refresh;
table1.Next;
end;
end;

procedure TForm6.Edit2Change(Sender: TObject);
begin
form5.Query2.Close;
form5.Query2.SQL.Clear;
form5.Query2.SQL.Add('select Barcode,Name from drug_tb1 where barcode
like'#39+(edit2.Text)+'%'+#39);
form5.Query2.Open;
edit3.Text:=form5.DBGrid1.Fields[1].AsString;
if(form5.Query2.RecordCount=0)then
begin
ShowMessage('Not Found this Drug, turn new drug and add it!!');
end;
if edit2.Text=''then
else begin
table1.Filtered:=false;
table1.Filter:='Barcode='+edit2.Text;
table1.Filtered:=true;
end;
end;

procedure TForm6.FormActivate(Sender: TObject);
begin
edit2.Text:='';
edit3.Text:='';
edit1.Text:='';
edit2.SetFocus;

end;

procedure TForm6.BitBtn6Click(Sender: TObject);
```

```
begin
form6.close;
form1.show;
end;

procedure TForm6.spSkinButton1Click(Sender: TObject);
begin
form6.Close;
form1.Show;
end;

end.
```

Form -7- Login

```
unit Unit7;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, IdBaseComponent, IdComponent, IdTCPConnection, IdTCPClient,
  IdHTTP, Grids, StdCtrls, ExtCtrls, DB, DBTables, Buttons, Mask,
  SkinBoxCtrls, SkinCtrls;

type
  TForm7 = class(TForm)
    Edit1: TEdit;
    Edit2: TEdit;
    DataSource1: TDataSource;
    Query1: TQuery;
    spSkinShadowLabel1: TspSkinShadowLabel;
    spSkinShadowLabel3: TspSkinShadowLabel;
    spSkinShadowLabel2: TspSkinShadowLabel;
    spSkinButton1: TspSkinButton;
    spSkinButton2: TspSkinButton;
    procedure BitBtn1Click(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
    procedure Edit1KeyPress(Sender: TObject; var Key: Char);
    procedure Edit2KeyPress(Sender: TObject; var Key: Char);
    procedure FormActivate(Sender: TObject);
    procedure spSkinButton1Click(Sender: TObject);
    procedure spSkinButton2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form7: TForm7;

implementation

uses Unit1, Unit8;

{$R *.dfm}

procedure TForm7.BitBtn1Click(Sender: TObject);
begin
query1.Close;
query1.SQL.Clear;
```

```
query1.SQL.Add('Select * from enter where Name='+#39+edit1.Text+#39+' and
Passs='+#39+edit2.Text+#39);
query1.Open;
if query1.RecordCount=0 then
spskinshadowlabel2.Visible:=true
else begin
form7.Hide;
form1.Show;
form1.spSkinShadowLabel1.Caption:='Welcome  '+edit1.Text;
end;
end;
procedure TForm7.BitBtn2Click(Sender: TObject);
begin
edit1.Text:='';
edit2.Text:='';
spskinshadowlabel2.Visible:=false;
end;

procedure TForm7.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13)then edit2.SetFocus;
end;

procedure TForm7.Edit2KeyPress(Sender: TObject; var Key: Char);
begin
if(key=#13) then spskinbutton1.SetFocus;
end;

procedure TForm7.FormActivate(Sender: TObject);
begin
edit1.SetFocus;
edit1.Text:='';
edit2.Text:='';
end;

procedure TForm7.spSkinButton1Click(Sender: TObject);
begin
query1.Close;
query1.SQL.Clear;
query1.SQL.Add('Select * from enter where Name='+#39+edit1.Text+#39+' and
Passs='+#39+edit2.Text+#39);
query1.Open;
if query1.RecordCount=0 then
spskinshadowlabel2.Visible:=true
else begin
form7.Hide;
form1.Show;
form1.spSkinShadowLabel1.Caption:='Welcome  '+edit1.Text;
end;
end;
```

```
procedure TForm7.spSkinButton2Click(Sender: TObject);
begin
edit1.Text:='';
edit2.Text:='';
spskinshadowlabel2.Visible:=false;
edit1.SetFocus;
end;

end.
```

Form -8- New Account

```
unit Unit8;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Buttons, Mask, DBCtrls, DB, DBTables, SkinCtrls;

type
  TForm8 = class(TForm)
    DBEdit1: TDBEdit;
    DBEdit2: TDBEdit;
    Edit1: TEdit;
    BitBtn1: TBitBtn;
    BitBtn3: TBitBtn;
    DataSource1: TDataSource;
    Query1: TQuery;
    spSkinShadowLabel1: TspSkinShadowLabel;
    spSkinShadowLabel2: TspSkinShadowLabel;
    spSkinShadowLabel3: TspSkinShadowLabel;
    spSkinShadowLabel4: TspSkinShadowLabel;
    BitBtn6: TBitBtn;
    procedure BitBtn1Click(Sender: TObject);
    procedure BitBtn3Click(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
    procedure BitBtn6Click(Sender: TObject);
    procedure FormActivate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form8: TForm8;

implementation

uses Unit1;

{$R *.dfm}

procedure TForm8.BitBtn1Click(Sender: TObject);
var
a:word;
begin
if (dbedit2.Text=edit1.Text) then
  a:=application.MessageBox('Are you sure add this user?','Are you sure',36);
```

```
   if(a=IDYES)then begin
     query1.Insert;
     query1.Post;
     dbedit1.Clear;
     dbedit2.Clear;
     edit1.Text:='';
     end
   else begin
     spskinshadowlabel4.Visible:=true;
     dbedit1.Clear;
     dbedit2.Clear;
     edit1.Text:='';
   end
  end;
procedure TForm8.BitBtn3Click(Sender: TObject);
begin
dbedit1.Clear;
dbedit2.Clear;
edit1.Text:='';
spskinshadowlabel4.Caption:='';
dbedit1.SetFocus;
end;

procedure TForm8.BitBtn2Click(Sender: TObject);
begin
form8.Close;
form1.show;
end;

procedure TForm8.BitBtn6Click(Sender: TObject);
begin
form8.Close;
form1.Show;
end;

procedure TForm8.FormActivate(Sender: TObject);
begin
dbedit1.SetFocus;
end;

end.
```

Form -9- Delete Account

```
unit Unit9;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Buttons, Grids, DBGrids, DB, DBTables, SkinCtrls;

type
  TForm9 = class(TForm)
    Query1: TQuery;
    DataSource1: TDataSource;
    DBGrid1: TDBGrid;
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    Edit1: TEdit;
    BitBtn6: TBitBtn;
    procedure BitBtn2Click(Sender: TObject);
    procedure FormActivate(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
    procedure spSkinButton1Click(Sender: TObject);
    procedure BitBtn6Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form9: TForm9;

implementation

uses Unit1;

{$R *.dfm}

procedure TForm9.BitBtn2Click(Sender: TObject);
var
a:word;
begin
a:=application.MessageBox('Are you sure?','Warning',36);
if(a=IDYES)then begin
query1.Delete;
end;
end;

procedure TForm9.FormActivate(Sender: TObject);
```

```
begin
query1.Refresh;
end;

procedure TForm9.BitBtn1Click(Sender: TObject);
begin
query1.Close;
query1.SQL.Clear;
query1.SQL.Add('select * from enter where name like'+#39+(edit1.Text)+'%'+#39);
query1.Open;
end;

procedure TForm9.spSkinButton1Click(Sender: TObject);
begin
form9.Close;
form1.show;
end;

procedure TForm9.BitBtn6Click(Sender: TObject);
begin
form9.Close;
form1.Show;
end;

end.
```

Form -10- SSK's Formulas

```
unit Unit10;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, SkinCtrls, Grids, DBGrids, DB, DBTables, StdCtrls, Buttons,
  ExtCtrls;

type
  TForm10 = class(TForm)
    spSkinShadowLabel1: TspSkinShadowLabel;
    spSkinShadowLabel2: TspSkinShadowLabel;
    spSkinShadowLabel3: TspSkinShadowLabel;
    spSkinShadowLabel4: TspSkinShadowLabel;
    spSkinShadowLabel5: TspSkinShadowLabel;
    spSkinShadowLabel6: TspSkinShadowLabel;
    Edit1: TEdit;
    Edit2: TEdit;
    Edit3: TEdit;
    Edit4: TEdit;
    Edit5: TEdit;
    Edit6: TEdit;
    Edit7: TEdit;
    spSkinShadowLabel7: TspSkinShadowLabel;
    Edit8: TEdit;
    DataSource1: TDataSource;
    DBGrid1: TDBGrid;
    DBGrid2: TDBGrid;
    Table2: TTable;
    DataSource2: TDataSource;
    DataSource3: TDataSource;
    Table3: TTable;
    DBGrid3: TDBGrid;
    BitBtn2: TBitBtn;
    BitBtn3: TBitBtn;
    Timer1: TTimer;
    Table1: TTable;
    BitBtn1: TBitBtn;
    Label1: TLabel;
    DataSource4: TDataSource;
    Table4: TTable;
    DBGrid4: TDBGrid;
    spSkinShadowLabel8: TspSkinShadowLabel;
    spSkinShadowLabel9: TspSkinShadowLabel;
    spSkinShadowLabel10: TspSkinShadowLabel;
    Edit9: TEdit;
    Edit10: TEdit;
    Edit11: TEdit;
    GroupBox1: TGroupBox;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    Label7: TLabel;
    Label8: TLabel;
    Label9: TLabel;
    Label10: TLabel;
    Label11: TLabel;
    Label12: TLabel;
    Label13: TLabel;
    Label14: TLabel;
    Label15: TLabel;
    BitBtn6: TBitBtn;
    procedure Timer1Timer(Sender: TObject);
    procedure FormActivate(Sender: TObject);
    procedure Edit8Change(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
    procedure BitBtn3Click(Sender: TObject);
    procedure BitBtn4Click(Sender: TObject);
    procedure Edit9Change(Sender: TObject);
    procedure Edit3Change(Sender: TObject);
    procedure Edit3KeyPress(Sender: TObject; var Key: Char);
    procedure Edit8KeyPress(Sender: TObject; var Key: Char);
    procedure Edit5KeyPress(Sender: TObject; var Key: Char);
    procedure Edit9KeyPress(Sender: TObject; var Key: Char);
    procedure BitBtn6Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form10: TForm10;

implementation

uses Unit5, Unit2, Unit3, Unit1;

{$R *.dfm}

procedure TForm10.Timer1Timer(Sender: TObject);
begin
edit1.Text:=datetostr(date);
edit2.Text:=timetostr(time);
```

```
end;

procedure TForm10.FormActivate(Sender: TObject);
begin
edit3.Text:='';
edit4.Text:='';
edit5.Text:='';
edit6.Text:='';
edit7.Text:='';
edit8.Text:='';
edit9.Text:='';
edit10.Text:='';
edit11.Text:='';
table3.Insert;
edit3.SetFocus;
end;

procedure TForm10.Edit8Change(Sender: TObject);
begin
if edit8.Text='' then
else begin
form2.Query1.Close;
form2.Query1.SQL.Clear;
form2.Query1.SQL.Add('select * from Patient_tb where Register_N
like'#39+(edit8.Text)+'%'+#39);
form2.Query1.Open;
edit6.Text:=form2.DBGrid1.Fields[1].AsString;
edit7.Text:=form2.DBGrid1.Fields[2].AsString;
table2.Close;
table2.Open;
table2.Filtered:=false;
table2.Filter:='register_n='+edit8.Text;
table1.Filtered:=true;
if(form2.Query1.RecordCount=0)then
begin
ShowMessage('Not Found this Patient, turn new drug and add it!!');
end;
end;
end;

procedure TForm10.BitBtn2Click(Sender: TObject);
var
a,b,c,d,sum1,sum2:real;
num:integer;
begin
num:=Application.MessageBox('Are you sure to do SELL?','!!SELL!!',
MB_YESNOCANCEL+MB_ICONQUESTION);
if num=mrYes then begin
if edit5.Text=''then
showmessage('Please Enter Quantity')
```

```
else begin
table3.Edit;
table3.FieldByName('Date').AsString:=edit1.Text;
table3.FieldByName('Time').AsString:=edit2.Text;
table3.FieldByName('DrugBarcode').AsString:=edit3.Text;
table3.FieldByName('DrugName').AsString:=edit4.Text;
table3.FieldByName('Quantity').AsString:=edit5.Text;
table3.FieldByName('PatientName').AsString:=edit6.Text;
table3.FieldByName('PatientSurname').AsString:=edit7.Text;
table3.FieldByName('number').AsString:=edit8.Text;
table3.FieldByName('type').AsString:='ssk';
table3.FieldByName('DoctorNum').AsString:=edit9.Text;
table3.FieldByName('DoctorName').AsString:=edit10.Text;
table3.FieldByName('DoctorSurname').AsString:=edit11.Text;
label1.Caption:=table1.Fields[13].AsString;
table1.Edit;
table1.FieldByName('Quantity').AsString:=inttostr(strtoint(label1.Caption)-
strtoint(edit5.Text));
table1.Post;
label5.Caption:=table3.Fields[4].AsString;
label8.Caption:=table3.Fields[5].AsString;
label7.Caption:=table1.Fields[5].AsString;
label6.Caption:=table1.Fields[6].AsString;
label9.Caption:=table1.Fields[12].AsString;
a:=strtofloat(label6.Caption);//benefit 2.3
b:=strtofloat(label7.Caption);//tax 1,12
c:=strtofloat(label8.Caption);//quantity
d:=strtofloat(label9.Caption);//net price 5,66
table3.FieldByName('BenefitCost').AsString:=floattostr(d*(a*0.01));
sum1:=(d*(a*0.01));//benefit cost
sum1:=sum1+d;//benefit+netPrice
table3.FieldByName('TaxCost').AsString:=floattostr(sum1*(b*0.01));//tax cost
sum2:=(sum1*(b*0.01));
table3.FieldByName('TotalCost').AsString:=floattostr((sum1+sum2)*c);
label10.Caption:=floattostr(sum1+sum2);
label11.Caption:=floattostr((sum1+sum2)*c);
table3.Post;
end;
end;
if num=mrCancel then begin
form10.Close;
form1.show;
end;
end;

procedure TForm10.BitBtn1Click(Sender: TObject);
begin
edit3.Text:='';
edit4.Text:='';
edit5.Text:='';
```

```
edit6.Text:='';
edit7.Text:='';
edit8.Text:='';
edit9.Text:='';
edit10.Text:='';
edit11.Text:='';
edit3.SetFocus;
table3.Insert;
end;

procedure TForm10.BitBtn3Click(Sender: TObject);
begin
edit3.Text:='';
edit4.Text:='';
edit5.Text:='';
edit6.Text:='';
edit7.Text:='';
edit8.Text:='';
edit9.Text:='';
edit10.Text:='';
edit11.Text:='';
edit3.SetFocus;
end;

procedure TForm10.BitBtn4Click(Sender: TObject);
begin
form10.Close;
form1.Show;
end;

procedure TForm10.Edit9Change(Sender: TObject);
begin
if edit9.Text='' then
else begin
form3.Query1.Close;
form3.Query1.SQL.Clear;
form3.Query1.SQL.Add('select * from Doctor_tb where Diplomoed_N
like'#39+(edit9.Text)+'%'+#39);
form3.Query1.Open;
edit10.Text:=form3.DBGrid1.Fields[1].AsString;
edit11.Text:=form3.DBGrid1.Fields[2].AsString;
table4.Close;
table4.Open;
table4.Filtered:=false;
table4.Filter:='Diplomoed_N='+edit9.Text;
table4.Filtered:=true;
if(form3.Query1.RecordCount=0)then
begin
ShowMessage('Not Found this Doctor, turn New Doctor and add it!!');
end;
```

```
end;
end;

procedure TForm10.Edit3Change(Sender: TObject);
begin
if edit3.Text='' then
else begin
table1.Close;
table1.Open;
table1.Filtered:=false;
table1.Filter:='barcode='+edit3.Text;
table1.Filtered:=true;
edit4.Text:=dbgrid2.Fields[1].AsString;
end;
end;

procedure TForm10.Edit3KeyPress(Sender: TObject; var Key: Char);
begin
if(key=#13)then edit5.SetFocus;
end;

procedure TForm10.Edit8KeyPress(Sender: TObject; var Key: Char);
begin
if(key=#13)then edit9.SetFocus;
end;

procedure TForm10.Edit5KeyPress(Sender: TObject; var Key: Char);
begin
if(key=#13)then edit8.SetFocus;
end;

procedure TForm10.Edit9KeyPress(Sender: TObject; var Key: Char);
begin
if(key=#13)then bitbtn2.SetFocus;
end;

procedure TForm10.BitBtn6Click(Sender: TObject);
begin
form10.Close;
form1.Show;
end;

end.
```

Form -11- Sellign With Formulas

```
unit Unit11;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, ExtCtrls, StdCtrls, DB, DBTables, Grids, DBGrids, Buttons,
  SkinCtrls;

type
  TForm11 = class(TForm)
    spSkinShadowLabel1: TspSkinShadowLabel;
    spSkinShadowLabel2: TspSkinShadowLabel;
    spSkinShadowLabel3: TspSkinShadowLabel;
    spSkinShadowLabel4: TspSkinShadowLabel;
    spSkinShadowLabel5: TspSkinShadowLabel;
    spSkinShadowLabel6: TspSkinShadowLabel;
    spSkinShadowLabel7: TspSkinShadowLabel;
    Edit1: TEdit;
    Edit2: TEdit;
    Edit3: TEdit;
    Edit4: TEdit;
    Edit5: TEdit;
    Edit6: TEdit;
    Edit7: TEdit;
    Edit8: TEdit;
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    BitBtn3: TBitBtn;
    BitBtn4: TBitBtn;
    DataSource1: TDataSource;
    DataSource2: TDataSource;
    DataSource3: TDataSource;
    DBGrid1: TDBGrid;
    DBGrid2: TDBGrid;
    DBGrid3: TDBGrid;
    Table1: TTable;
    Table2: TTable;
    Table3: TTable;
    Label1: TLabel;
    Timer1: TTimer;
    Table4: TTable;
    DataSource4: TDataSource;
    DBGrid4: TDBGrid;
    spSkinShadowLabel8: TspSkinShadowLabel;
    spSkinShadowLabel9: TspSkinShadowLabel;
    spSkinShadowLabel10: TspSkinShadowLabel;
    Edit9: TEdit;
    Edit10: TEdit;
    Edit11: TEdit;
    GroupBox1: TGroupBox;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    Label7: TLabel;
    Label8: TLabel;
    Label9: TLabel;
    Label10: TLabel;
    Label11: TLabel;
    Label12: TLabel;
    Label13: TLabel;
    Label14: TLabel;
    Label15: TLabel;
    procedure Timer1Timer(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
    procedure BitBtn3Click(Sender: TObject);
    procedure BitBtn4Click(Sender: TObject);
    procedure Edit3Change(Sender: TObject);
    procedure Edit6Change(Sender: TObject);
    procedure FormActivate(Sender: TObject);
    procedure Edit9Change(Sender: TObject);
    procedure Edit3KeyPress(Sender: TObject; var Key: Char);
    procedure Edit5KeyPress(Sender: TObject; var Key: Char);
    procedure Edit6KeyPress(Sender: TObject; var Key: Char);
    procedure Edit9KeyPress(Sender: TObject; var Key: Char);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form11: TForm11;

implementation

uses Unit5, Unit2, Unit3, Unit10, Unit1;

{$R *.dfm}

procedure TForm11.Timer1Timer(Sender: TObject);
begin
edit1.Text:=datetostr(date);
edit2.Text:=timetostr(time);
end;
```

```
procedure TForm11.BitBtn1Click(Sender: TObject);
begin
edit3.Text:='';
edit4.Text:='';
edit5.Text:='';
edit6.Text:='';
edit7.Text:='';
edit8.Text:='';
edit9.Text:='';
edit10.Text:='';
edit11.Text:='';
edit3.SetFocus;
table3.Insert;
end;

procedure TForm11.BitBtn2Click(Sender: TObject);
var
a,b,c,d,sum1,sum2:real;
num:integer;
begin
num:=Application.MessageBox('Are you sure to do SELL?','!!SELL!!',
MB_YESNOCANCEL+MB_ICONQUESTION);
if num=mrYes then begin
if edit5.Text=''then
showmessage('Please Enter Quantity')
else begin
table3.Edit;
table3.FieldByName('Date').AsString:=edit1.Text;
table3.FieldByName('Time').AsString:=edit2.Text;
table3.FieldByName('DrugBarcode').AsString:=edit3.Text;
table3.FieldByName('DrugName').AsString:=edit4.Text;
table3.FieldByName('Quantity').AsString:=edit5.Text;
table3.FieldByName('PatientName').AsString:=edit7.Text;
table3.FieldByName('PatientSurname').AsString:=edit8.Text;
table3.FieldByName('number').AsString:=edit6.Text;
table3.FieldByName('type').AsString:='formulas';
table3.FieldByName('DoctorNum').AsString:=edit9.Text;
table3.FieldByName('DoctorName').AsString:=edit10.Text;
table3.FieldByName('DoctorSurname').AsString:=edit11.Text;
label1.Caption:=table1.Fields[13].AsString;
table1.Edit;
table1.FieldByName('Quantity').AsString:=inttostr(strtoint(label1.Caption)-
strtoint(edit5.Text));
table1.Post;
label6.Caption:=table3.Fields[4].AsString;
label7.Caption:=table3.Fields[5].AsString;
label8.Caption:=table1.Fields[5].AsString;
label9.Caption:=table1.Fields[6].AsString;
label13.Caption:=table1.Fields[12].AsString;
```

```
a:=strtofloat(label9.Caption);//benefit
b:=strtofloat(label8.Caption);//tax
c:=strtofloat(label7.Caption);//quantity
d:=strtofloat(label13.Caption);//net price
table3.FieldByName('BenefitCost').AsString:=floattostr(d*(a*0.01));
sum1:=(d*(a*0.01));//benefit cost
sum1:=sum1+d;//benefit+netPrice
table3.FieldByName('TaxCost').AsString:=floattostr(sum1*(b*0.01));//tax cost
sum2:=(sum1*(b*0.01));
table3.FieldByName('TotalCost').AsString:=floattostr((sum1+sum2)*c);
label14.Caption:=floattostr(sum1+sum2);
label15.Caption:=floattostr((sum1+sum2)*c);
table3.Post;
end;
end;
if num=mrCancel then begin
form10.Close;
form1.show;
end;
end;

procedure TForm11.BitBtn3Click(Sender: TObject);
begin
edit3.Text:='';
edit4.Text:='';
edit5.Text:='';
edit6.Text:='';
edit7.Text:='';
edit8.Text:='';
edit3.SetFocus;
end;

procedure TForm11.BitBtn4Click(Sender: TObject);
begin
form11.Close;
form1.Show;
end;

procedure TForm11.Edit3Change(Sender: TObject);
begin
if edit3.Text='' then
else begin
table1.Close;
table1.Open;
table1.Filtered:=false;
table1.Filter:='barcode='+edit3.Text;
table1.Filtered:=true;
edit4.Text:=dbgrid2.Fields[1].AsString;
end;
end;
```

```
procedure TForm11.Edit6Change(Sender: TObject);
begin
if edit6.Text='' then
else begin
form2.Query1.Close;
form2.Query1.SQL.Clear;
form2.Query1.SQL.Add('select * from Patient_tb where Register_N
like'#39+(edit6.Text)+'%'+#39);
form2.Query1.Open;
edit7.Text:=form2.DBGrid1.Fields[1].AsString;
edit8.Text:=form2.DBGrid1.Fields[2].AsString;
table2.Close;
table2.Open;
table2.Filtered:=false;
table2.Filter:='register_n='+edit6.Text;
table1.Filtered:=true;
if(form2.Query1.RecordCount=0)then
begin
ShowMessage('Not Found this Patient, turn new drug and add it!!');
end;
end;
end;

procedure TForm11.FormActivate(Sender: TObject);
begin
edit3.Text:='';
edit4.Text:='';
edit5.Text:='';
edit6.Text:='';
edit7.Text:='';
edit8.Text:='';
edit9.Text:='';
edit10.Text:='';
edit11.Text:='';
edit3.SetFocus;
end;

procedure TForm11.Edit9Change(Sender: TObject);
begin
if edit9.Text='' then
else begin
form3.Query1.Close;
form3.Query1.SQL.Clear;
form3.Query1.SQL.Add('select * from Doctor_tb where Diplomoed_N
like'#39+(edit9.Text)+'%'+#39);
form3.Query1.Open;
edit10.Text:=form3.DBGrid1.Fields[1].AsString;
edit11.Text:=form3.DBGrid1.Fields[2].AsString;
table4.Close;
```

```
table4.Open;
table4.Filtered:=false;
table4.Filter:='Diplomoed_N='+edit9.Text;
table4.Filtered:=true;
if(form3.Query1.RecordCount=0)then
begin
ShowMessage('Not Found this Doctor, turn new doctor!!');
end;
end;
end;

procedure TForm11.Edit3KeyPress(Sender: TObject; var Key: Char);
begin
if(key=#13) then edit5.SetFocus;
end;

procedure TForm11.Edit5KeyPress(Sender: TObject; var Key: Char);
begin
if(key=#13) then edit6.SetFocus;
end;

procedure TForm11.Edit6KeyPress(Sender: TObject; var Key: Char);
begin
if(key=#13) then edit9.SetFocus;
end;

procedure TForm11.Edit9KeyPress(Sender: TObject; var Key: Char);
begin
if(key=#13) then bitbtn2.SetFocus;
end;

end.
```

Form -12- Calculator

```
unit Unit12;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ComCtrls;

type
  TForm12 = class(TForm)
    Button1: TButton;
    RichEdit1: TRichEdit;
    Button2: TButton;
    Button3: TButton;
    Button4: TButton;
    Button5: TButton;
    Button6: TButton;
    Button7: TButton;
    Button8: TButton;
    Button9: TButton;
    Button10: TButton;
    Button11: TButton;
    Button12: TButton;
    Button13: TButton;
    Button14: TButton;
    Button15: TButton;
    Button16: TButton;
    Button17: TButton;
    Button18: TButton;
    Button19: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure Button5Click(Sender: TObject);
    procedure Button6Click(Sender: TObject);
    procedure Button7Click(Sender: TObject);
    procedure Button8Click(Sender: TObject);
    procedure Button9Click(Sender: TObject);
    procedure Button10Click(Sender: TObject);
    procedure FormActivate(Sender: TObject);
    procedure Button11Click(Sender: TObject);
    procedure Button12Click(Sender: TObject);
    procedure Button13Click(Sender: TObject);
    procedure Button14Click(Sender: TObject);
    procedure Button15Click(Sender: TObject);
    procedure Button16Click(Sender: TObject);
    procedure Button17Click(Sender: TObject);
    procedure Button18Click(Sender: TObject);
    procedure Button19Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form12: TForm12;
  num1:integer;
  num2:integer;
  operation:integer;
  result1:integer;
  result:real;

implementation

uses Unit1, Unit14;

{$R *.dfm}

procedure TForm12.Button1Click(Sender: TObject);
begin
richedit1.Text:=richedit1.Text+'1';
end;

procedure TForm12.Button2Click(Sender: TObject);
begin
richedit1.Text:=richedit1.Text+'2';
end;

procedure TForm12.Button3Click(Sender: TObject);
begin
richedit1.Text:=richedit1.Text+'3';
end;

procedure TForm12.Button4Click(Sender: TObject);
begin
richedit1.Text:=richedit1.Text+'4';
end;

procedure TForm12.Button5Click(Sender: TObject);
begin
richedit1.Text:=richedit1.Text+'5';
end;

procedure TForm12.Button6Click(Sender: TObject);
begin
richedit1.Text:=richedit1.Text+'6';
```

```
end;

procedure TForm12.Button7Click(Sender: TObject);
begin
richedit1.Text:=richedit1.Text+'7';
end;

procedure TForm12.Button8Click(Sender: TObject);
begin
richedit1.Text:=richedit1.Text+'8';
end;

procedure TForm12.Button9Click(Sender: TObject);
begin
richedit1.Text:=richedit1.Text+'9';
end;

procedure TForm12.Button10Click(Sender: TObject);
begin
richedit1.Text:=richedit1.Text+'0';
end;

procedure TForm12.FormActivate(Sender: TObject);
begin
richedit1.Text:='';
end;

procedure TForm12.Button11Click(Sender: TObject);
begin
num1:=StrToInt(richedit1.Text);
operation:=4;
richedit1.Text:='';
button18.enabled:=true;
end;

procedure TForm12.Button12Click(Sender: TObject);
begin
num1:=StrToInt(richedit1.Text);
operation:=3;
richedit1.Text:='';
button18.enabled:=true;
end;

procedure TForm12.Button13Click(Sender: TObject);
begin
num1:=StrToInt(richedit1.Text);
operation:=2;
richedit1.Text:='';
button18.enabled:=true;
end;
```

```
procedure TForm12.Button14Click(Sender: TObject);
begin
num1:=StrToInt(richedit1.Text);
operation:=1;
richedit1.Text:='';
button18.enabled:=true;
end;

procedure TForm12.Button15Click(Sender: TObject);
begin
richedit1.Text:='';
num1:=0;
num2:=0;
richedit1.Text:='';
richedit1.SetFocus;
end;

procedure TForm12.Button16Click(Sender: TObject);
begin
richedit1.Text:='';
button18.enabled:=false;
richedit1.SetFocus;
end;

procedure TForm12.Button17Click(Sender: TObject);
begin
num1:=StrToInt(richedit1.Text);
result1:=num1*num1;
richedit1.Text:=inttostr(result1);
end;

procedure TForm12.Button18Click(Sender: TObject);
begin
num2:=StrToInt(richedit1.Text);
if operation=1 then result:=num1+num2;
if operation=2 then result:=num1-num2;
if operation=3 then result:=num1*num2;
if operation=4 then result:=num1/num2;
richedit1.Text:=floattostr(result);
end;

procedure TForm12.Button19Click(Sender: TObject);
begin
richedit1.Clear;
form14.close;
form1.show;
end;

end.
```

Form -13- Selling WithOut Formulas

```
unit Unit13;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Grids, DBGrids, DB, DBTables, StdCtrls, ExtCtrls, SkinCtrls;

type
  TForm13 = class(TForm)
    Table1: TTable;
    Table2: TTable;
    Table3: TTable;
    Table4: TTable;
    DataSource1: TDataSource;
    DataSource2: TDataSource;
    DataSource3: TDataSource;
    DataSource4: TDataSource;
    DBGrid1: TDBGrid;
    DBGrid2: TDBGrid;
    DBGrid3: TDBGrid;
    DBGrid4: TDBGrid;
    Edit1: TEdit;
    Edit2: TEdit;
    Edit3: TEdit;
    Edit4: TEdit;
    Edit5: TEdit;
    Edit7: TEdit;
    Edit8: TEdit;
    Label1: TLabel;
    spSkinShadowLabel1: TspSkinShadowLabel;
    spSkinShadowLabel2: TspSkinShadowLabel;
    spSkinShadowLabel3: TspSkinShadowLabel;
    spSkinShadowLabel6: TspSkinShadowLabel;
    spSkinShadowLabel4: TspSkinShadowLabel;
    spSkinShadowLabel5: TspSkinShadowLabel;
    Timer1: TTimer;
    GroupBox1: TGroupBox;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    Label7: TLabel;
    Label8: TLabel;
    Label9: TLabel;
    Label10: TLabel;
    Label11: TLabel;
    Label12: TLabel;
    Label13: TLabel;
    Label14: TLabel;
    Label15: TLabel;
    spSkinButton1: TspSkinButton;
    spSkinButton2: TspSkinButton;
    spSkinButton3: TspSkinButton;
    spSkinButton4: TspSkinButton;
    procedure spSkinButton4Click(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);
    procedure FormActivate(Sender: TObject);
    procedure Edit3Change(Sender: TObject);
    procedure spSkinButton1Click(Sender: TObject);
    procedure spSkinButton2Click(Sender: TObject);
    procedure spSkinButton3Click(Sender: TObject);
    procedure Edit3KeyPress(Sender: TObject; var Key: Char);
    procedure Edit5KeyPress(Sender: TObject; var Key: Char);
    procedure Edit7KeyPress(Sender: TObject; var Key: Char);
    procedure Edit8KeyPress(Sender: TObject; var Key: Char);
    private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form13: TForm13;

implementation

uses Unit1;

{$R *.dfm}

procedure TForm13.spSkinButton4Click(Sender: TObject);
begin
form13.Close;
form1.show;
end;

procedure TForm13.Timer1Timer(Sender: TObject);
begin
edit1.Text:=datetostr(date);
edit2.Text:=timetostr(time);
end;

procedure TForm13.FormActivate(Sender: TObject);
begin
edit3.Text:='';
edit4.Text:='';
```

```
edit5.Text:='';
edit7.Text:='';
edit8.Text:='';
table1.Insert;
edit3.SetFocus;
end;

procedure TForm13.Edit3Change(Sender: TObject);
begin
if edit3.Text='' then
else begin
table2.Close;
table2.Open;
table2.Filtered:=false;
table2.Filter:='barcode='+edit3.Text;
table2.Filtered:=true;
edit4.Text:=dbgrid2.Fields[1].AsString;
end;
if table2.FieldByName('Formul').AsString='T'then begin
showmessage('This Drug is a Formul; You can NOT sell this!!');
edit4.Text:='';
edit3.Text:='';
edit3.SetFocus;
end;
end;
procedure TForm13.spSkinButton1Click(Sender: TObject);
begin
edit3.Text:='';
edit4.Text:='';
edit5.Text:='';
edit7.Text:='';
edit8.Text:='';
edit3.SetFocus;
table1.Insert;
end;

procedure TForm13.spSkinButton2Click(Sender: TObject);
var
a,b,c,d,sum1,sum2:real;
num:integer;
begin
num:=Application.MessageBox('Are you sure to do SELL?','!!SELL!!',
MB_YESNOCANCEL+MB_ICONQUESTION);
if num=mrYes then begin
if edit5.Text=''then
showmessage('Please Enter Quantity')
else begin
table1.Edit;
table1.FieldByName('Date').AsString:=edit1.Text;
table1.FieldByName('Time').AsString:=edit2.Text;
```

```
table1.FieldByName('DrugBarcode').AsString:=edit3.Text;
table1.FieldByName('DrugName').AsString:=edit4.Text;
table1.FieldByName('Quantity').AsString:=edit5.Text;
table1.FieldByName('PatientName').AsString:=edit7.Text;
table1.FieldByName('PatientSurname').AsString:=edit8.Text;
table1.FieldByName('type').AsString:='Unformula';
label1.Caption:=table2.Fields[13].AsString;
table2.Edit;
table2.FieldByName('Quantity').AsString:=inttostr(strtoint(label1.Caption)-
strtoint(edit5.Text));
table2.Post;
label6.Caption:=table1.Fields[4].AsString;
label7.Caption:=table1.Fields[5].AsString;
label8.Caption:=table2.Fields[5].AsString;
label9.Caption:=table2.Fields[6].AsString;
label13.Caption:=table2.Fields[12].AsString;
a:=strtofloat(label9.Caption);//benefit
b:=strtofloat(label8.Caption);//tax
c:=strtofloat(label7.Caption);//quantity
d:=strtofloat(label13.Caption);//net price
table1.FieldByName('BenefitCost').AsString:=floattostr(d*(a*0.01));
sum1:=(d*(a*0.01));//benefit cost
sum1:=sum1+d;//benefit+netPrice
table1.FieldByName('TaxCost').AsString:=floattostr(sum1*(b*0.01));//tax cost
sum2:=(sum1*(b*0.01));
table1.FieldByName('TotalCost').AsString:=floattostr((sum1+sum2)*c);
label14.Caption:=floattostr(sum1+sum2);
label15.Caption:=floattostr((sum1+sum2)*c);
table1.Post;
end;
end;
if num=mrCancel then begin
form13.Close;
form1.show;
end;
end;

procedure TForm13.spSkinButton3Click(Sender: TObject);
begin
edit3.Text:='';
edit4.Text:='';
edit5.Text:='';
edit7.Text:='';
edit8.Text:='';
edit3.SetFocus;
end;

procedure TForm13.Edit3KeyPress(Sender: TObject; var Key: Char);
begin
if(key=#13) then edit5.SetFocus;
```

```
end;

procedure TForm13.Edit5KeyPress(Sender: TObject; var Key: Char);
begin
if(key=#13) then edit7.SetFocus;
end;

procedure TForm13.Edit7KeyPress(Sender: TObject; var Key: Char);
begin
if(key=#13) then edit8.SetFocus;
end;

procedure TForm13.Edit8KeyPress(Sender: TObject; var Key: Char);
begin
if(key=#13) then spskinbutton2.SetFocus;
end;

end.
```

Form -14- Stock Controlling System

```
unit Unit14;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, ExtCtrls, QuickRpt, DB, DBTables, StdCtrls, QRCtrls, SkinCtrls;

type
  TForm14 = class(TForm)
    QuickRep1: TQuickRep;
    ColumnHeaderBand1: TQRBand;
    DetailBand1: TQRBand;
    PageFooterBand1: TQRBand;
    PageHeaderBand1: TQRBand;
    SummaryBand1: TQRBand;
    QRLabel1: TQRLabel;
    QRLabel2: TQRLabel;
    QRLabel3: TQRLabel;
    QRLabel4: TQRLabel;
    QRLabel5: TQRLabel;
    QRDBText1: TQRDBText;
    QRDBText2: TQRDBText;
    QRDBText3: TQRDBText;
    QRDBText4: TQRDBText;
    QRDBText5: TQRDBText;
    QRLabel6: TQRLabel;
    RadioGroup1: TRadioGroup;
    spSkinButton1: TspSkinButton;
    Edit1: TEdit;
    Query1: TQuery;
    DataSource2: TDataSource;
    spSkinButton2: TspSkinButton;
    QRLabel7: TQRLabel;
    QRLabel8: TQRLabel;
    QRDBText6: TQRDBText;
    QRDBText7: TQRDBText;
    spSkinShadowLabel1: TspSkinShadowLabel;
    GroupBox1: TGroupBox;
    spSkinButton3: TspSkinButton;
    spSkinShadowLabel2: TspSkinShadowLabel;
    spSkinShadowLabel3: TspSkinShadowLabel;
    spSkinShadowLabel4: TspSkinShadowLabel;
    spSkinShadowLabel5: TspSkinShadowLabel;
    spSkinShadowLabel6: TspSkinShadowLabel;
    spSkinShadowLabel7: TspSkinShadowLabel;
    procedure Button1Click(Sender: TObject);
    procedure FormActivate(Sender: TObject);
    procedure spSkinButton1Click(Sender: TObject);
    procedure RadioGroup1Click(Sender: TObject);
    procedure spSkinButton2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form14: TForm14;

implementation

uses Unit1;

{$R *.dfm}

procedure TForm14.Button1Click(Sender: TObject);
begin
quickrep1.Preview;
end;

procedure TForm14.FormActivate(Sender: TObject);
begin
quickrep1.Visible:=false;
end;

procedure TForm14.spSkinButton1Click(Sender: TObject);
begin
if(radiogroup1.ItemIndex=0)then begin
query1.Close;
query1.SQL.Clear;
query1.SQL.Add('select * from drug_tb1 where barcode
like'+#39+(edit1.Text)+'%'+#39);
query1.Open;
if(query1.RecordCount=0)then begin
qrlabel6.Caption:='Drug Not FOUND!';
end else
qrlabel6.Caption:='Stock Controlling System';
end;

if(radiogroup1.ItemIndex=1)then begin
query1.Close;
query1.SQL.Clear;
query1.SQL.Add('select * from drug_tb1 where name like'+#39+(edit1.Text)+'%'+#39);
query1.Open;
if(query1.RecordCount=0)then begin
qrlabel6.Caption:='Drug Not FOUND!';
end else
```

```
qrlabel6.Caption:='Stock Controlling System';
end;

if(radiogroup1.ItemIndex=2)then begin
query1.Close;
query1.SQL.Clear;
query1.SQL.Add('select * from drug_tb1 where Company_N
like'+#39+(edit1.Text)+'%'+#39);
query1.Open;
if(query1.RecordCount=0)then begin
qrlabel6.Caption:='Company Name Not FOUND!';
end else
qrlabel6.Caption:='Stock Controlling System';
end;

if(radiogroup1.ItemIndex=3)then begin
query1.Close;
query1.SQL.Clear;
query1.SQL.Add('select * from drug_tb1 where Formul like'+#39+('T')+'%'+#39);
query1.Open;
if(query1.RecordCount=0)then begin
qrlabel6.Caption:='Not Having Formul Drug!';
end else
qrlabel6.Caption:='Stock Controlling System';
end;

if(radiogroup1.ItemIndex=4)then begin
query1.Close;
query1.SQL.Clear;
query1.SQL.Add('select * from drug_tb1 where Formul like'+#39+('F')+'%'+#39);
query1.Open;
if(query1.RecordCount=0)then begin
qrlabel6.Caption:='Not Having NonFormul Drug!';
end else
qrlabel6.Caption:='Stock Controlling System';
end;

if(radiogroup1.ItemIndex=5)then begin
query1.Close;
query1.SQL.Clear;
query1.SQL.Add('select * from drug_tb1');
query1.Open;
end;


quickrep1.Preview;
end;


procedure TForm14.RadioGroup1Click(Sender: TObject);
```

```
begin

if(radiogroup1.ItemIndex=3) or (radiogroup1.ItemIndex=4) or
(radiogroup1.ItemIndex=5)then begin
edit1.Visible:=false;
end
else begin
edit1.Visible:=true;
end;

end;

procedure TForm14.spSkinButton2Click(Sender: TObject);
begin
form14.Close;
form1.Show;
end;

end.
```

Form -15- Selling Controlling System

```
unit Unit15;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ExtCtrls, Grids, DBGrids, DB, DBTables, Mask,
  SkinBoxCtrls, Buttons, SkinCtrls, QRCtrls, QuickRpt;

type
  TForm15 = class(TForm)
    Table1: TTable;
    DataSource1: TDataSource;
    DBGrid1: TDBGrid;
    RadioGroup1: TRadioGroup;
    BitBtn1: TBitBtn;
    spSkinDateEdit1: TspSkinDateEdit;
    spSkinDateEdit2: TspSkinDateEdit;
    spSkinRadioGroup1: TspSkinRadioGroup;
    BitBtn2: TBitBtn;
    GroupBox1: TGroupBox;
    spSkinShadowLabel1: TspSkinShadowLabel;
    Edit1: TEdit;
    BitBtn3: TBitBtn;
    Edit2: TEdit;
    spSkinShadowLabel2: TspSkinShadowLabel;
    GroupBox2: TGroupBox;
    Edit3: TEdit;
    Edit4: TEdit;
    spSkinShadowLabel3: TspSkinShadowLabel;
    spSkinShadowLabel4: TspSkinShadowLabel;
    spSkinShadowLabel5: TspSkinShadowLabel;
    Edit5: TEdit;
    BitBtn4: TBitBtn;
    BitBtn5: TBitBtn;
    BitBtn6: TBitBtn;
    GroupBox4: TGroupBox;
    Edit6: TEdit;
    Edit7: TEdit;
    Edit8: TEdit;
    BitBtn8: TBitBtn;
    spSkinShadowLabel8: TspSkinShadowLabel;
    spSkinShadowLabel9: TspSkinShadowLabel;
    spSkinShadowLabel10: TspSkinShadowLabel;
    BitBtn9: TBitBtn;
    QuickRep1: TQuickRep;
    ColumnHeaderBand1: TQRBand;
    DetailBand1: TQRBand;
```

```
var
  Form15: TForm15;

implementation

uses Unit1, Unit18;

{$R *.dfm}


procedure TForm15.BitBtn2Click(Sender: TObject);
begin
table1.Close;
table1.Open;
table1.Filtered:=false;
table1.Refresh;
end;

procedure TForm15.spSkinRadioGroup1Click(Sender: TObject);
begin
if spskinradiogroup1.ItemIndex=0 then
begin
spskindateedit1.Visible:=true;
spskindateedit1.Visible:=false;
end;
if spskinradiogroup1.ItemIndex=1 then
begin
spskindateedit1.Visible:=true;
spskindateedit1.Visible:=true;
end;
if spskinradiogroup1.ItemIndex=2 then
begin
spskindateedit1.Visible:=true;
spskindateedit1.Visible:=false;
end;
end;

procedure TForm15.BitBtn1Click(Sender: TObject);
begin

if spskinradiogroup1.ItemIndex=1 then begin
table1.IndexName:='datee';
table1.SetRange([spskindateedit1.Text],[spskindateedit2.Text]);
table1.ApplyRange;
end;

if spskinradiogroup1.ItemIndex=0 then begin
table1.IndexName:='datee';
table1.SetRangeStart;
table1.FieldByName('date').AsDateTime:=strtodate(spskindateedit2.Text);
```

```
table1.ApplyRange;
end;

if spskinradiogroup1.ItemIndex=2 then begin
table1.IndexName:='datee';
table1.SetRangeEnd;
table1.FieldByName('date').AsDateTime:=strtodate(spskindateedit2.Text);
table1.ApplyRange;
end;

end;
procedure TForm15.BitBtn3Click(Sender: TObject);
begin
if edit1.Text='' then
else begin
table1.Close;
table1.Open;
table1.Filtered:=false;
table1.Filter:='Drugbarcode='+edit1.Text;
table1.Filtered:=true;
edit2.Text:=dbgrid1.Fields[4].AsString;
end;
end;

procedure TForm15.BitBtn4Click(Sender: TObject);
begin
if edit3.Text='' then
else begin
table1.Close;
table1.Open;
table1.Filtered:=false;
table1.Filter:='Number='+edit3.Text;
table1.Filtered:=true;
edit4.Text:=dbgrid1.Fields[6].AsString;
edit5.Text:=dbgrid1.Fields[7].AsString;
end;
end;

procedure TForm15.BitBtn8Click(Sender: TObject);
begin
if edit6.Text='' then
else begin
table1.Close;
table1.Open;
table1.Filtered:=false;
table1.Filter:='DoctorNum='+edit6.Text;
table1.Filtered:=true;
edit7.Text:=dbgrid1.Fields[11].AsString;
edit8.Text:=dbgrid1.Fields[12].AsString;
end;
```

```
end;

procedure TForm15.BitBtn5Click(Sender: TObject);
var
count:integer;
benefit:real;
tax:real;
cost:real;
begin
benefit:=0.0;
tax:=0.0;
cost:=0.0;
count:=table1.RecordCount;
qrlabel16.Caption:=inttostr(count);
table1.First;
while not table1.Eof do begin
benefit:=benefit+table1.Fields[15].AsFloat;
tax:=tax+table1.Fields[14].AsFloat;
cost:=cost+table1.Fields[13].AsFloat;
table1.Next;
end;
qrlabel17.Caption:=floattostr(benefit);
qrlabel18.Caption:=floattostr(tax);
qrlabel19.Caption:=floattostr(cost);
quickrep1.Preview;
end;

procedure TForm15.FormActivate(Sender: TObject);
begin
quickrep1.Visible:=false;
end;

procedure TForm15.BitBtn9Click(Sender: TObject);
begin
form15.Close;
form1.show;

end;

procedure TForm15.BitBtn6Click(Sender: TObject);
begin
form18.show;
end;

end.
```

Form -16- Internet Browser

```
unit Unit16;

interface

uses
 Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
 Dialogs, StdCtrls, Buttons, OleCtrls, SHDocVw;

type
 TForm16 = class(TForm)
  WebBrowser1: TWebBrowser;
  Edit1: TEdit;
  GO: TBitBtn;
  BitBtn2: TBitBtn;
  Back: TBitBtn;
  Forward: TBitBtn;
  DutyPharmacy: TBitBtn;
  Exit: TBitBtn;
  Label1: TLabel;
  BitBtn1: TBitBtn;
  procedure ExitClick(Sender: TObject);
  procedure GOClick(Sender: TObject);
  procedure DutyPharmacyClick(Sender: TObject);
  procedure BitBtn2Click(Sender: TObject);
  procedure ForwardClick(Sender: TObject);
  procedure BackClick(Sender: TObject);
  procedure Edit1KeyPress(Sender: TObject; var Key: Char);
  procedure FormActivate(Sender: TObject);
  procedure BitBtn1Click(Sender: TObject);
 private
  { Private declarations }
 public
  { Public declarations }
 end;

var
 Form16: TForm16;

implementation

uses Unit1;

{$R *.dfm}

procedure TForm16.ExitClick(Sender: TObject);
begin
form16.Close;
form1.show;
```

```
end;

procedure TForm16.GOClick(Sender: TObject);
begin
webbrowser1.Navigate(edit1.Text);
end;

procedure TForm16.DutyPharmacyClick(Sender: TObject);
begin
webbrowser1.Navigate(label1.Caption);
end;

procedure TForm16.BitBtn2Click(Sender: TObject);
begin
webbrowser1.Refresh;
end;

procedure TForm16.ForwardClick(Sender: TObject);
begin
webbrowser1.GoForward;
end;

procedure TForm16.BackClick(Sender: TObject);
begin
webbrowser1.GoBack;
end;

procedure TForm16.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13)then
webbrowser1.Navigate(edit1.Text);
end;

procedure TForm16.FormActivate(Sender: TObject);
begin
edit1.SetFocus;
end;

procedure TForm16.BitBtn1Click(Sender: TObject);
begin
webbrowser1.Stop;
end;

end.
```

Form -17- Help

```
unit Unit17;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, ExtCtrls, SkinCtrls, StdCtrls;

type
  TForm17 = class(TForm)
    spSkinButton1: TspSkinButton;
    spSkinButton2: TspSkinButton;
    spSkinButton3: TspSkinButton;
    spSkinButton4: TspSkinButton;
    spSkinButton5: TspSkinButton;
    spSkinButton6: TspSkinButton;
    spSkinButton7: TspSkinButton;
    spSkinButton8: TspSkinButton;
    Panel1: TPanel;
    Panel2: TPanel;
    Panel3: TPanel;
    Panel4: TPanel;
    Panel5: TPanel;
    Panel6: TPanel;
    Panel7: TPanel;
    Panel8: TPanel;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    spSkinShadowLabel1: TspSkinShadowLabel;
    spSkinShadowLabel2: TspSkinShadowLabel;
    Label5: TLabel;
    Label6: TLabel;
    Label7: TLabel;
    Label8: TLabel;
    Label9: TLabel;
    spSkinShadowLabel3: TspSkinShadowLabel;
    Label10: TLabel;
    Label11: TLabel;
    Label12: TLabel;
    Label13: TLabel;
    spSkinShadowLabel4: TspSkinShadowLabel;
    Label14: TLabel;
    Label15: TLabel;
    Label16: TLabel;
    Label17: TLabel;
    Label18: TLabel;
    Label19: TLabel;
    spSkinShadowLabel5: TspSkinShadowLabel;
    Label20: TLabel;
    Label21: TLabel;
    Label22: TLabel;
    Label23: TLabel;
    Label24: TLabel;
    Label25: TLabel;
    Label26: TLabel;
    Label27: TLabel;
    Label28: TLabel;
    Label30: TLabel;
    Label29: TLabel;
    Label31: TLabel;
    spSkinShadowLabel6: TspSkinShadowLabel;
    Label32: TLabel;
    Label33: TLabel;
    Label34: TLabel;
    Label35: TLabel;
    Label36: TLabel;
    spSkinShadowLabel7: TspSkinShadowLabel;
    Label37: TLabel;
    Label38: TLabel;
    Label39: TLabel;
    Label40: TLabel;
    Label41: TLabel;
    Label42: TLabel;
    Label43: TLabel;
    Label44: TLabel;
    Label45: TLabel;
    Label46: TLabel;
    Label47: TLabel;
    spSkinShadowLabel8: TspSkinShadowLabel;
    Label48: TLabel;
    Label49: TLabel;
    Label50: TLabel;
    Label51: TLabel;
    procedure spSkinButton1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure spSkinButton2Click(Sender: TObject);
    procedure spSkinButton3Click(Sender: TObject);
    procedure spSkinButton4Click(Sender: TObject);
    procedure spSkinButton5Click(Sender: TObject);
    procedure spSkinButton6Click(Sender: TObject);
    procedure spSkinButton7Click(Sender: TObject);
    procedure spSkinButton8Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

```
var
  Form17: TForm17;

implementation

{$R *.dfm}

procedure TForm17.spSkinButton1Click(Sender: TObject);
begin
panel1.Visible:=true;
panel2.Visible:=false;
panel3.Visible:=false;
panel4.Visible:=false;
panel5.Visible:=false;
panel6.Visible:=false;
panel7.Visible:=false;
panel8.Visible:=false;
end;

procedure TForm17.FormCreate(Sender: TObject);
begin
panel1.Visible:=true;
panel2.Visible:=false;
panel3.Visible:=false;
panel4.Visible:=false;
panel5.Visible:=false;
panel6.Visible:=false;
panel7.Visible:=false;
panel8.Visible:=false;
end;

procedure TForm17.spSkinButton2Click(Sender: TObject);
begin
panel1.Visible:=false;
panel2.Visible:=true;
panel3.Visible:=false;
panel4.Visible:=false;
panel5.Visible:=false;
panel6.Visible:=false;
panel7.Visible:=false;
panel8.Visible:=false;
end;

procedure TForm17.spSkinButton3Click(Sender: TObject);
begin
panel1.Visible:=false;
panel2.Visible:=false;
panel3.Visible:=true;
panel4.Visible:=false;
```

```
panel5.Visible:=false;
panel6.Visible:=false;
panel7.Visible:=false;
panel8.Visible:=false;
end;

procedure TForm17.spSkinButton4Click(Sender: TObject);
begin
panel1.Visible:=false;
panel2.Visible:=false;
panel3.Visible:=false;
panel4.Visible:=true;
panel5.Visible:=false;
panel6.Visible:=false;
panel7.Visible:=false;
panel8.Visible:=false;
end;

procedure TForm17.spSkinButton5Click(Sender: TObject);
begin
panel1.Visible:=false;
panel2.Visible:=false;
panel3.Visible:=false;
panel4.Visible:=false;
panel5.Visible:=true;
panel6.Visible:=false;
panel7.Visible:=false;
panel8.Visible:=false;
end;

procedure TForm17.spSkinButton6Click(Sender: TObject);
begin
panel1.Visible:=false;
panel2.Visible:=false;
panel3.Visible:=false;
panel4.Visible:=false;
panel5.Visible:=false;
panel6.Visible:=true;
panel7.Visible:=false;
panel8.Visible:=false;
end;

procedure TForm17.spSkinButton7Click(Sender: TObject);
begin
panel1.Visible:=false;
panel2.Visible:=false;
panel3.Visible:=false;
panel4.Visible:=false;
panel5.Visible:=false;
panel6.Visible:=false;
```

```
panel7.Visible:=true;
panel8.Visible:=false;
end;

procedure TForm17.spSkinButton8Click(Sender: TObject);
begin
panel1.Visible:=false;
panel2.Visible:=false;
panel3.Visible:=false;
panel4.Visible:=false;
panel5.Visible:=false;
panel6.Visible:=false;
panel7.Visible:=false;
panel8.Visible:=true;
end;

end.
```

Form -18- Graphical Analysies

```
unit Unit18;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Buttons, TeEngine, Series, ExtCtrls, TeeProcs, Chart;

type
  TForm18 = class(TForm)
    Chart1: TChart;
    Series1: TPieSeries;
    BitBtn1: TBitBtn;
    procedure FormKeyPress(Sender: TObject; var Key: Char);
    procedure BitBtn1Click(Sender: TObject);
    procedure FormActivate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form18: TForm18;

implementation

uses Unit15;

{$R *.dfm}

procedure TForm18.FormKeyPress(Sender: TObject; var Key: Char);
begin
If (Key = #13) then
  begin
    key := #0;
    Perform(WM_NEXTDLGCTL, 0, 0);
   end;
end;

procedure TForm18.BitBtn1Click(Sender: TObject);
begin
form18.Close;
end;

procedure TForm18.FormActivate(Sender: TObject);
var
c1,c2,c3:integer;
```

```
begin
c1:=0;
c2:=0;
c3:=0;
form15.Table1.First;
while not form15.Table1.Eof do begin
 if form15.Table1.FieldByName('DrugBarcode').AsString='8699536090115' then
 c1:=c1+1;
 if form15.Table1.FieldByName('DrugBarcode').AsString='8699546010011' then
 c2:=c2+1;
 if form15.Table1.FieldByName('DrugBarcode').AsString='8699504120097' then
 c3:=c3+1;
form15.Table1.Next;
end;
series1.Clear;
series1.Add(c1,'Majezik',clred);
series1.Add(c2,'Aspirin',clgreen);
series1.Add(c3,'Cataflam',clblue);
end;

end.
```

Form -19- About

```
unit Unit19;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, jpeg, ExtCtrls, StdCtrls, Buttons, SkinCtrls;

type
  TForm19 = class(TForm)
    spSkinShadowLabel1: TspSkinShadowLabel;
    spSkinShadowLabel2: TspSkinShadowLabel;
    spSkinShadowLabel3: TspSkinShadowLabel;
    spSkinShadowLabel4: TspSkinShadowLabel;
    spSkinShadowLabel5: TspSkinShadowLabel;
    spSkinShadowLabel6: TspSkinShadowLabel;
    GroupBox1: TGroupBox;
    spSkinShadowLabel11: TspSkinShadowLabel;
    spSkinShadowLabel12: TspSkinShadowLabel;
    spSkinShadowLabel13: TspSkinShadowLabel;
    spSkinShadowLabel14: TspSkinShadowLabel;
    spSkinShadowLabel15: TspSkinShadowLabel;
    spSkinShadowLabel16: TspSkinShadowLabel;
    BitBtn1: TBitBtn;
    Image1: TImage;
    procedure BitBtn1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form19: TForm19;

implementation

{$R *.dfm}

procedure TForm19.BitBtn1Click(Sender: TObject);
begin
form19.Close;
end;

end.
```