



NEAR EAST UNIVERSITY

Faculty of Engineering

Department of Computer Engineering

Stock Database Design: Java Application

**Graduation Project
COM- 400**

Student: Mohammad Elfawair (20020888)

Supervisor: Assist. Prof. Dr. Adil Amirijanov

Lefkoşa - 2006

ACKNOLEDGMENT

First I want to thank Dr.Adil Amirijanov .Under this guidance, I successfully overcome many difficulties and learn a bout programming, in each discussion he explained my questions patiently, and I felt progress from his advices. I asked him many questions in computer science and he always answered my questions in detail.

Special thanks to faculty of engineering for their helping me, to get good qualification that helps me in my life.

I also want to thanks my friends those helped me for the past 4 years , special thanks to Mohammad al Ramlawee, Imad al dahdouh for their guidance.

Finally, I want to thank my family, especially my parents. Without their endless support and love for me, I would never achieve my current position. I wish my mother lives happily with my father in the heaven be proud for me.

ABSTRACT

In this time (computer age) the need for computerizing becomes an important demand in many fields factories, hospitals, companies etc.....

According to this demand this program is written to satisfy that demand this is a computer program for automating stock sales, so this program helps stock owners to be up to date about their stock, see the sales and store them in a database management system, JAVA language is used because it is a high technology language that can work in any platform, also MSACCESS is used as a database management system to save data.

This program is recommended to all shops to have a computer based system for businesses, and to have accurate information to their customers, suppliers, and their products.

TABLE OF CONTENTS

ACKNOWLEDGMENT	i
ABSTRACT	ii
INTRODUCTION	iii
CHAPTER ONE: JAVA PROGRAMMING FUNDAMENTALS	1
1.1 About The Java Technology	1
1.1.1 The java programming language	1
1.1.2 The java platform	3
1.1.3 What can java technology do	4
1.2 Language Basics	5
1.2.1 Final Variables	5
1.2.2 Exception Handling Statements	6
1.3 Object Oriented Programming	6
1.3.1 Classes	6
1.3.2 Inheritance	7
1.3.3 Interface	9
1.4 Layout Managers	10
1.4.1 How Layout Managers works	10
1.4.2 How to use Border Layout	11
1.4.3 How to use Flow Layout	11
1.4.4 How to use Grid Layout	12
1.5 How To Use Action Listener	13
1.6 JComponents	13
1.6.1 JButton	13
1.6.2 JLabel	13

1.6.3	JMenu	14
1.6.4	JScrollPane	14
1.6.5	JDialog	14
1.6.5.1	Method ShowMessageDialog	15
1.6.6	JFrame	15
1.6.6.1	Responding to Window-Closing Events	16
1.6.7	JTable	16
1.6.8	JInternalFrame	17
CHAPTER TWO: MICROSOFT ACCESS DATABASE		18
2.1	Introduction	18
2.2	Objects	19
2.3	Tables	20
2.2.1	Data Types	21
2.2.2	Primary key	22
2.2.3	Entering Data	22
2.2.4	Relationships	23
2.4	Forms	24
2.5	Reports	24
CHAPTER THREE: OPEN DATABASE CONNECTIVITY AND JAVA DATABASE CONNECTIVITY		25
3.1	Open Database Connectivity (ODBC)	25
3.1.1	Why use ODBC	25
3.2	Java DataBase Connectivity (JDBC)	26
3.2.1	Creating JDBC Application	26
3.2.2	Establishing a Connection	26
3.2.3	Loading Drivers	27

3.2.4 Making the Connection	27
3.2.5 Creating JDBC Statements	28
3.2.6 Executing Statements	28
3.2.7 Closing the statements and the connection	29
3.3 Using The Statement Interface	29
3.3.1 Entering data into a table	29
3.3.2 Getting data from a table	29
3.3.3 Retrieving values from a Result Set	30
3.3.4 Using the method next	30
3.3.5 Using the getXXX method	30
3.3.6 Updating Tables	31
3.4 Using Prepared Statement Interface	31
3.4.1 When to use Prepared Statement object	31
3.4.2 Creating a Prepared Statement objects	32
3.4.3 Supplying prepared statements with parameters	32
 CHAPTER FOUR: STOK PROGRAM DESIGN	 34
4.1 Database Design Of The System	34
4.2Block Diagram Of The System	36
4.3Main Menu	36
4.3.1 Add Menu	37
4.3.1.1 Product	37
4.3.1.1 Supplier	38
4.3.1.1 Exit	39
4.3.2 Edit Menu	39
4.3.2.1 Product	39
4.3.2.2 Supplier	40
4.3.3 Delete Menu	40
4.3.3.1 Product	41
4.3.3.2 Supplier	41

4.3.4 Sales Menu	42
4.3.4.1 Show Sales	42
4.3.4.1 Sell	43
4.3.5 About Menu	45
4.3.5.1 About	45
 CONCLUSION	 46
REFERENCES	47
APPENDIX	48

INTRODUCTION

Computer automation becomes a public demand in these days because we are in the technology age so I have to satisfy this demand as a computer engineer.

This project helps stock managers to be up-to-date i.e. control efficiently their stock and their relatives, add, edit, delete suppliers and customers, control the sales and issue reports, graphical user interfaces is used to increase the interactivity between the user and the system.

This project contains four chapters

Chapter one talks about java programming language and java technology and introduce some of java fundamentals that is used in my project, such as OOP, Layout managers, and JComponents.

Chapter two talks about Microsoft Access Database Management System and introduce Database objects (i.e. Tables, reports, forms) used in designing the database part of the project.

Chapter three talks about open database connectivity (ODBC) which is an interface or bridge between the application programs and a database to link application program with database, also this chapter introduces java database connectivity (JDBC) that is an API for specification for connecting programs written in java to database.

Chapter four talks about the project design, the design of the project's database, block diagram of the project, and guidelines to use the program.

CHAPTER ONE

JAVA FUNDAMENTALS

1.1 About the Java Technology

Java technology is both a programming language and a platform.

1.1.1 The Java Programming Language

The Java programming language is a high-level language that can be characterized by all of the following buzzwords:

Simple	Architecture neutral
Object oriented	Portable
Distributed	High performance
Interpreted	Multithreaded
Robust	Dynamic
Secure	

With most programming languages, you either compile or interpret a program so that you can run it on your computer. The Java programming language is unusual in that a program is both compiled and interpreted. With the compiler, first you translate a program into an intermediate language called Java bytecodes —the platform-independent codes interpreted by the interpreter on the Java platform. The interpreter parses and runs each Java bytecode instruction on the computer. Compilation happens just once; interpretation occurs each time the program is executed. The following figure illustrates how this works.

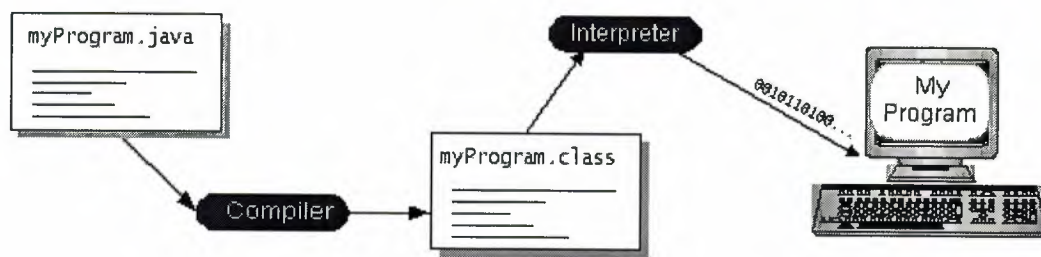


Figure1.1 MyProgram

You can think of Java bytecodes as the machine code instructions for the Java Virtual Machine (Java VM). Every Java interpreter, whether it's a development tool or a Web browser that can run applets, is an implementation of the Java VM

Java bytecodes help make "write once, run anywhere" possible. You can compile your program into bytecodes on any platform that has a Java compiler. The bytecodes can then be run on any implementation of the Java VM. That means that as long as a computer has a Java VM, the same program written in the Java programming language can run on Windows 2000, a Solaris workstation, or on an iMac.

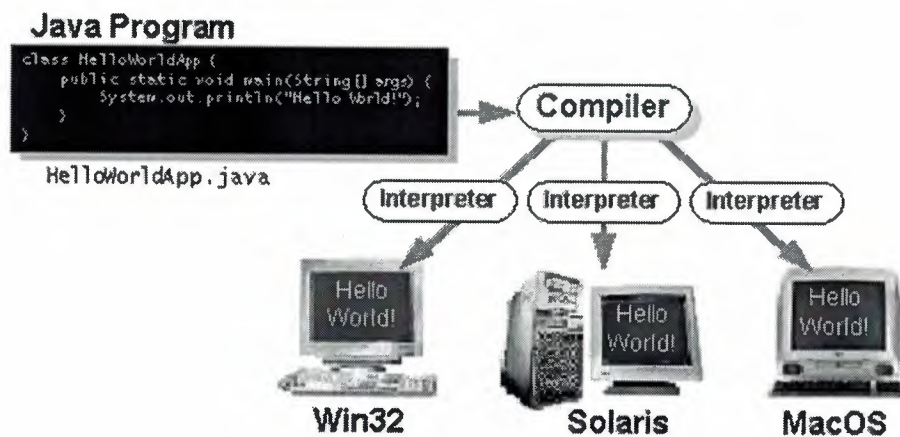


Figure 1.2 Java Program

1.1.2 The Java Platform

A platform is the hardware or software environment in which a program runs. I've already mentioned some of the most popular platforms like Windows 2000, Linux, Solaris, and MacOS. Most platforms can be described as a combination of the operating system and hardware. The Java platform differs from most other platforms in that it's a software-only platform that runs on top of other hardware-based platforms.

The Java platform has two components:

The Java Virtual Machine (Java VM)

The Java Application Programming Interface (Java API)

You've already been introduced to the Java VM. It's the base for the Java platform and is ported onto various hardware-based platforms.

The Java API is a large collection of ready-made software components that provide many useful capabilities, such as graphical user interface (GUI) widgets. The Java API is grouped into libraries of related classes and interfaces; these libraries are known as packages. The following figure depicts a program that's running on the Java platform. As the figure shows, the Java API and the virtual machine insulate the program from the hardware.

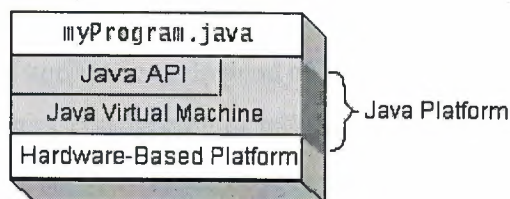


Figure 1.3 Java platform

Native code is code that after you compile it, the compiled code runs on a specific hardware platform. As a platform-independent environment, the Java platform can be a bit slower than native code. However, smart compilers, well-tuned interpreters, and just-in-time bytecode compilers can bring performance close to that of native code without threatening portability.

1.1.3 What Can Java Technology Do?

The most common types of programs written in the Java programming language are applets and applications. If you've surfed the Web, you're probably already familiar with applets. An applet is a program that adheres to certain conventions that allow it to run within a Java-enabled browser.

However, the Java programming language is not just for writing cute, entertaining applets for the Web. The general-purpose, high-level Java programming language is also a powerful software platform. Using the generous API, you can write many types of programs.

An application is a standalone program that runs directly on the Java platform. A special kind of application known as a server serves and supports clients on a network. Examples of servers are Web servers, proxy servers, mail servers, and print servers. Another specialized program is a servlet. A servlet can almost be thought of as an applet that runs on the server side. Java Servlets are a popular choice for building interactive web applications, replacing the use of CGI scripts. Servlets are similar to applets in that they are runtime extensions of applications. Instead of working in browsers, though, servlets run within Java Web servers, configuring or tailoring the server.

How does the API support all these kinds of programs? It does so with packages of software components that provide a wide range of functionality. Every full implementation of the Java platform gives you the following features:

The essentials: Objects, strings, threads, numbers, input and output, data structures, system properties, date and time, and so on.

Applets: The set of conventions used by applets.

Networking: URLs, TCP (Transmission Control Protocol), UDP (User Datagram Protocol) sockets, and IP (Internet Protocol) addresses.

Internationalization: Help for writing programs that can be localized for users worldwide. Programs can automatically adapt to specific locales and be displayed in the appropriate language.

Java Database Connectivity (JDBC™): Provides uniform access to a wide range of relational databases.

1.2 Language Basics

Here are some java programming basics that I used during desining of my projects

1.2.1 Final Variables

You can declare a variable in any scope to be final. The value of a final variable cannot change after it has been initialized. Such variables are similar to constants in other programming languages.

To declare a final variable, use the final keyword in the variable declaration before the type:

```
final int aFinalVar = 0;
```

1.2.2 Exception Handling Statements

The Java programming language provides a mechanism known as expression to help programs report and handle errors. When an error occurs, the program throws an exception. What does this mean? It means that the normal flow of the program is interrupted and that the runtime environment attempts to find an exception handler—a block of code that can handle a particular type of error. The exception handler can attempt to recover from the error or, if it determines that the error is unrecoverable, provide a gentle exit from the program.

Three statements play a part in handling exceptions:

The try statement identifies a block of statements within which an exception might be thrown.

The catch statement must be associated with a try statement and identifies a block of statements that can handle a particular type of exception. The statements are executed if an exception of a particular type occurs within the try block.

The finally statement must be associated with a try statement and identifies a block of statements that are executed regardless of whether or not an error occurs within the try block.

1.3 Object Oriented Programming

1.3.1 Classess

In the real world, you often have many objects of the same kind. For example, your bicycle is just one of many bicycles in the world. Using object-oriented terminology, we say that your bicycle object is an instance of the class of objects known as bicycles. Bicycles have some state (current gear, current cadence, two wheels) and behavior (change gears, brake) in common. However, each bicycle's state is independent of and can be different from that of other bicycles.

When building bicycles, manufacturers take advantage of the fact that bicycles share characteristics, building many bicycles from the same blueprint. It would be very inefficient to produce a new blueprint for every individual bicycle manufactured. In object-oriented software, it's also possible to have many objects of the same kind that share characteristics: rectangles, employee records, video clips, and so on. Like the bicycle manufacturers, you can take advantage of the fact that objects of the same kind are similar and you can create a blueprint for those objects. A software blueprint for objects is called a class

Definition: A class is a blueprint, or prototype, that defines the variables and the methods common to all objects of a certain kind.

1.3.2 Inheritance

Generally speaking, objects are defined in terms of classes. You know a lot about an object by knowing its class. Even if you don't know what a penny-farthing is, if I told you it was a bicycle, you would know that it had two wheels, handle bars, and pedals.

Object-oriented systems take this a step further and allow classes to be defined in terms of other classes. For example, mountain bikes, road bikes, and tandems are all kinds of bicycles. In object-oriented terminology, mountain bikes, road bikes, and tandems are all subclasses of the bicycle class. Similarly, the bicycle class is the super class of mountain bikes, racing bikes, and tandems. This relationship is shown in the following figure.

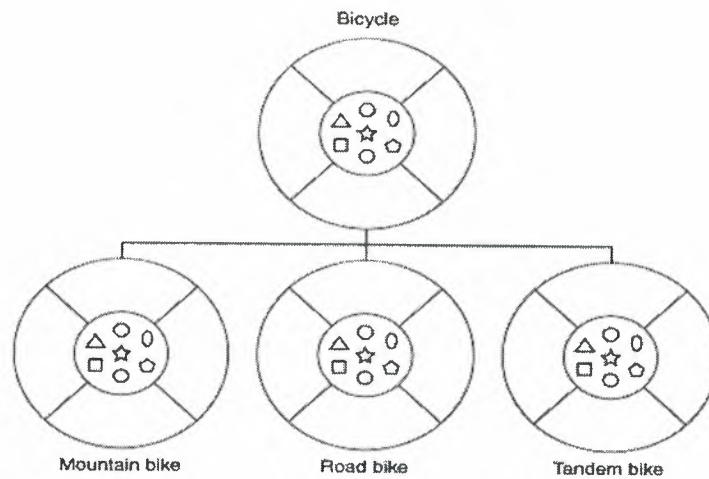


Figure 1.4 Bikes Classess

Each subclass inherits state (in the form of variable declarations) from the superclass. Mountain bikes, road bikes, and tandems share some states: cadence, speed, and the like. Also, each subclass inherits methods from the superclass. Mountain bikes, road bikes, and tandems share some behaviors: braking and changing pedaling speed, for example. However, subclasses are not limited to the state and behaviors provided to them by their superclass. Subclasses can add variables and methods to the ones they inherit from the superclass. Tandem bicycles have two seats and two sets of handle bars; some mountain bikes have an additional chain ring, giving them a lower gear ratio.

Subclasses can also override inherited methods and provide specialized implementations for those methods. For example, if you had a mountain bike with an additional chain ring, you would override the "change gears" method so that the rider could shift into those lower gears.

You are not limited to just one layer of inheritance. The inheritance tree, or class hierarchy, can be as deep as needed. Methods and variables are inherited down through the levels. In general, the farther down in the hierarchy a class appears, the more specialized its behavior.

The Object class is at the top of class hierarchy, and each class is its descendant (directly or indirectly). A variable of type Object can hold a reference to any object, such as an instance of a class or an array. Object provides behaviors that are required of all objects running in the Java Virtual Machine. For example, all classes inherit Object's toString method, which returns a string representation of the object.

Inheritance offers the following benefits:

Subclasses provide specialized behaviors from the basis of common elements provided by the superclass. Through the use of inheritance, programmers can reuse the code in the superclass many times.

Programmers can implement superclasses called abstract classes that define common behaviors. The abstract superclass defines and may partially implement the behavior, but much of the class is undefined and unimplemented. Other programmers fill in the details with specialized subclasses.

1.3.3 Interface

Definition: An interface is a device that unrelated objects — objects that are not related by class hierarchy — can use to interact with each other. An object can implement multiple interfaces.

The bicycle class and its class hierarchy define what a bicycle can and cannot do in terms of its “bicycleness.” But bicycles interact with the world on other terms. For example, a bicycle in a store could be managed by an inventory program. An inventory program doesn’t care what class of items it manages, as long as each item provides certain information, such as price and tracking number. Instead of forcing class relationships on otherwise unrelated items, the inventory program sets up a protocol of communication. This protocol comes in the form of a set of constant and method definitions contained within an interface. The inventory interface would define, but not implement, methods that set and get the retail price, assign a tracking number, and so on.

To work in the inventory program, the bicycle class must agree to this protocol by implementing the interface. When a class implements an interface, the class agrees to implement all the methods defined in the interface. Thus, the bicycle class would provide the implementations for the methods that set and get retail price, assign a tracking number, and so on.

1.4 Layout Managers

1.4.1 How Layout Management Works

Here's an example of a layout management sequence for a frame (JFrame).

After the GUI is constructed, the pack method is invoked on the JFrame. This specifies that the frame should be at its preferred size.

To find the frame's preferred size, the frame's layout manager adds the size of the frame's edges to the preferred size of the component directly contained by the frame. This is the sum of the preferred size of the frame's content pane, plus the size of the frame's menu bar, if any.

The content pane's layout manager is responsible for figuring out the content pane's preferred size. By default, this layout manager is a BorderLayout object. However, let's assume that we replace it with a GridLayout object that's set up to create two columns.

The interesting thing about grid layout is that it forces all components to be the same size, and it tries to make them as wide as the widest component's preferred width and as high as highest one's preferred height.

First, the grid layout manager queries the content pane for its insets — the size of the content pane's border, if any. Next, the grid layout manager queries each component in the content pane for its preferred size, noting the largest preferred width and largest preferred height. Then it calculates the content pane's preferred size.

When a component in the content pane is asked for its preferred size, the default implementation (used by most components) first checks whether the user specified a

preferred size. If so, it reports that size. If not, it queries its look and feel for the preferred size.

1.4.2 How to Use BorderLayout

Here's a snapshot of an application that uses a BorderLayout.

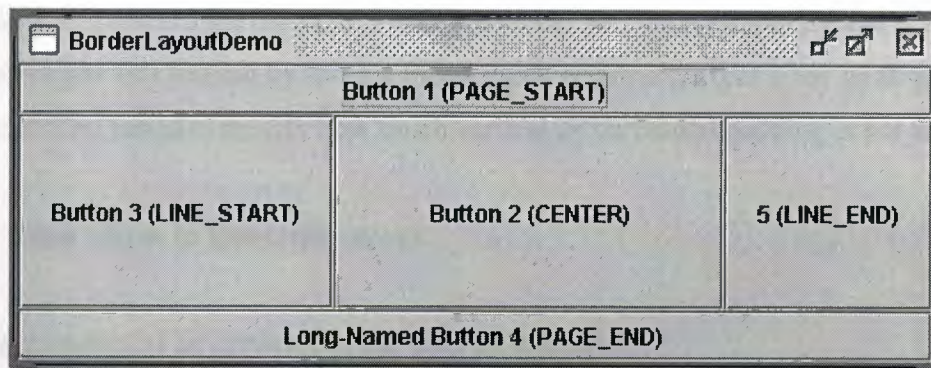


Figure 1.5 BorderLayout

If you enlarge the window, the center area gets as much of the available space as possible. The other areas expand only as much as necessary to fill all available space. Often, a container uses only one or two of the areas of the BorderLayout.

1.4.3 How to Use FlowLayout

The FlowLayout class provides a very simple layout manager that is used, by default, by JPanels. Here's a picture of an example that uses a flow layout:



Figure 1.6 FlowLayout

FlowLayout puts components in a row, sized at their preferred size. If the horizontal space in the container is too small to put all the components in one row, FlowLayout uses multiple rows. If the container is wider than necessary for a row of components, the row is, by default, centered horizontally within the container. You can specify that it stick to the left or right side instead by using a FlowLayout constructor that takes an alignment argument. You can also specify how much vertical or horizontal padding is put around

1.4.4 How to Use GridLayout

Here's a snapshot of an application that uses a GridLayout.

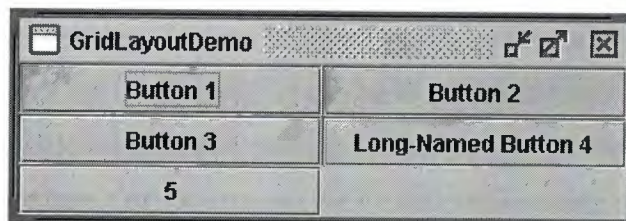


Figure 1.7 GridLayout

A GridLayout places components in a grid of cells. Each component takes all the available space within its cell, and each cell is exactly the same size. If you resize the GridLayoutDemo window, you'll see that the GridLayout changes the cell size so that the cells are as large as possible, given the space available to the container.

1.5 How to Write an Action Listener

Action listeners are probably the easiest — and most common — event handlers to implement. You implement an action listener to respond to the user's indication that some implementation-dependent action should occur.

When the user clicks a button, chooses a menu item or presses Enter in a text field, an action event occurs. The result is that an `actionPerformed` message is sent to all action listeners that are registered on the relevant component.

1.6 JComponent

1.6.1 JButton

Ordinary buttons — `JButton` objects — have just a bit more functionality than the `AbstractButton` class provides: You can make a `JButton` be the default button.

At most one button in a top-level container can be the default button. The default button typically has a highlighted appearance and acts clicked whenever the top-level container has the keyboard focus and the user presses the Return or Enter key. Here is a picture of a dialog, implemented in the `ListDialog` example, in which the Set button is the default button.

1.6.2 JLabel

With the `JLabel` class, you can display unselectable text and images. If you need to create a component that displays a string or an image (or both), you can do so by

using or extending JLabel. If the component is interactive and has state, consider using a button instead of a label.

1.6.3 JMenu

A menu provides a space-saving way to let the user choose one of several options. Other components with which the user can make a one-of-many choice include combo boxes, lists, radioButtons, spinner, and tool bar. If any of your menu items performs an action that is duplicated by another menu item or by a tool-bar button

Menus are unique in that, by convention, they aren't placed with the other components in the UI. Instead, a menu usually appears either in a menu bar or as a popup menu. A menu bar contains one or more menus and has a customary, platform-dependent location — usually along the top of a window. A popup menu is a menu that is invisible until the user makes a platform-specific mouse action, such as pressing the right mouse button, over a popup-enabled component. The popup menu then appears under the cursor.

1.6.4 JScrollPane

A JScrollPane provides a scrollable view of a component. When screen real estate is limited, use a scroll pane to display a component that is large or one whose size can change dynamically. Other containers used to save screen space include split panes and tabbed panes.

The code to create a scroll pane can be minimal

1.6.5 JDialog

Every dialog is dependent on a frame. When that frame is destroyed, so are its dependent dialogs. When the frame is iconified, its dependent dialogs disappear from the screen.

When the frame is deiconified, its dependent dialogs return to the screen. The AWT automatically provides this behavior.

A dialog can be modal. When a modal dialog is visible, it blocks user input to all other windows in the program. The JDialogs that JOptionPane creates are modal. To create a non-modal dialog, you must use the JDialog class directly.

Using JOptionPane, you can create and customize several different kinds of dialogs.

JOptionPane provides support for laying out standard dialogs, providing icons, specifying the dialog's title and text, and customizing the button text. Other features allow you to customize the components the dialog displays and specify where the dialog should appear onscreen

JOptionPane's icon support lets you easily specify which icon the dialog displays. You can use a custom icon, no icon at all, or any one of four standard JOptionPane icons (question, information, warning, and error). Each look and feel has its own versions of the four standard icons. The following figure shows the icons used in the Java look and feel.



Figure 1.8 JOptionPane icons

1.6.5.1 Method showMessageDialog

Displays a modal dialog with one button, which is labeled "OK" (or the localized equivalent). You can easily specify the message, icon, and title that the dialog displays.

1.6.6 JFrame

A frame, implemented as an instance of the `JFrame` class, is a window that typically has decorations such as a border, a title, and buttons for closing and iconifying the window. Applications with a GUI typically use at least one frame. Applets sometimes use frames, as well.

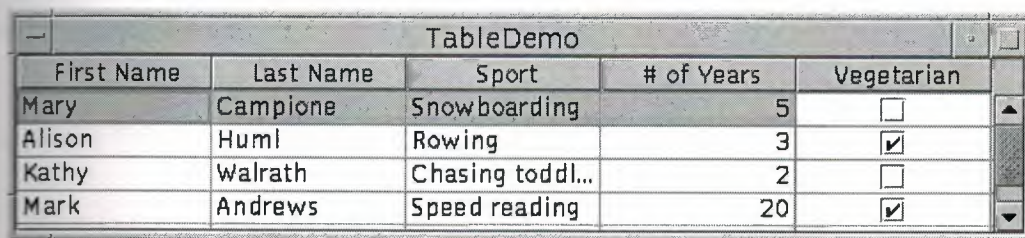
1.6.6.1 Responding to Window-Closing Events

By default, when the user closes a frame onscreen, the frame is hidden. Although invisible, the frame still exists and the program can make it visible again. If you want different behavior, then you need to either register a window listener that handles window-closing events, or you need to specify default close behavior using the `setDefaultCloseOperation` method. You can even do both.

The argument to `setDefaultCloseOperation` must be one of the following values, the first three of which are defined in the `WindowConstants` interface (implemented by `JFrame`, `JInternalPane`, and `JDialog`)

1.6.7 JTable

With the `JTable` class you can display tables of data, optionally allowing the user to edit the data. `JTable` doesn't contain or cache data; it's simply a view of your data. Here's a picture of a typical table displayed within a scroll pane:



First Name	Last Name	Sport	# of Years	Vegetarian
Mary	Campione	Snowboarding	5	<input type="checkbox"/>
Alison	Huml	Rowing	3	<input checked="" type="checkbox"/>
Kathy	Walrath	Chasing toddl...	2	<input type="checkbox"/>
Mark	Andrews	Speed reading	20	<input checked="" type="checkbox"/>

Figure 1.9 JTable

Table these parts:

- each cell displays an item of data
 - each column header describes its column
- maybe also have these labels:
- each column contains one type of data
- a table header displays the column headers]

1.6.8 JInternalFrame

With the `JInternalFrame` class you can display a `JFrame`-like window within another window. Usually, you add internal frames to a desktop pane. The desktop pane, in turn, might be used as the content pane of a `JFrame`. The desktop pane is an instance of `JDesktopPane`, which is a subclass of `JLayeredPane` that has added API for managing multiple overlapping internal frames.

You should consider carefully whether to base your program's GUI around frames or internal frames. Switching from internal frames to frames or vice versa isn't necessarily a simple task. By experimenting with both frames and internal frames, you can get an idea of the tradeoffs involved in choosing one over the other.

Here is a picture of an application that has two internal frames (one of which is iconified) inside a regular frame:

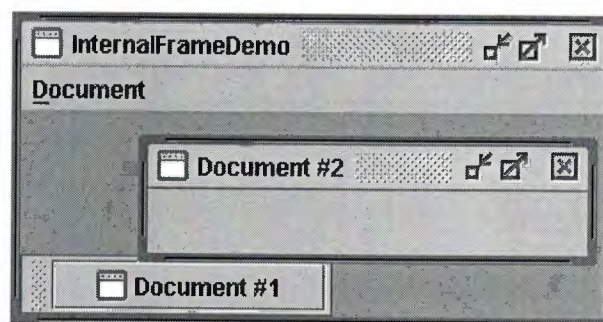


Figure 1.10 `JInternalFrame`

CHAPTER 2

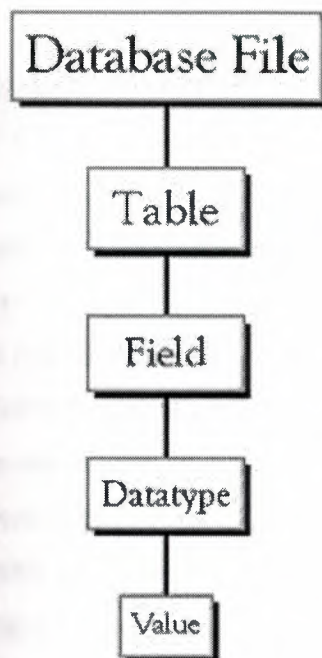
MICROSOFT ACCESS DATABASE

2.1 Introduction

Access is an interactive, relational database management system. A database is an organized collection of data stored in categories that are accessible in a logical or practical manner.

Relational databases enable data to be stored in multiple tables linked together via data indexes. This makes working with the data faster and easier. Once entered into the database, the data may be manipulated or viewed in various ways such as by sorting or by specially set-up queries and reports.

Microsoft Access is a powerful program to create and manage your databases. It has many built in features to assist you in constructing and viewing your information. Access is much more involved and is a more genuine database application than other programs such as Microsoft Works. First, you need to understand how Microsoft Access breaks down a database. Some keywords involved in this process are: *Database File*, *Table*, *Record*, *Field*, *Data-type*. Here is the Hierarchy that Microsoft Access uses in breaking down a database.



Database File: This is your main file that encompasses the entire database and that is saved to your hard-drive or floppy disk.

Table: A table is a collection of data about a specific topic. There can be multiple tables in a database.

Field: Fields are the different categories within a Table. Tables usually contain multiple fields.

Datatypes: Datatypes are the properties of each field. A field only has 1 datatype.

FieldName) Student LastName

Datatype) Text

2.2 Objects

Every database can contain several types of object. The data itself is contained in an object called a table. The data can be used in a wide variety of formats, for example, mailing lists, forms, reports and graphs. Each of these is a type of object. You create objects by clicking on the appropriate tab in the Database window, then clicking on the New button. Once objects such as tables have been created, they will be listed under the object tab whenever it is selected. To open an existing object, click on its name so that it becomes highlighted then click on Open. All objects have properties that can be set to determine how the object appears or operates

2.3 Tables

A table is the first type of object to create in a database. It is a way of defining how the data is to be stored. As with all databases, a table consists of 'records' (rows) and 'fields' (columns).

Each record represents one individual item in the database, such as a person in an address book, and each field represents a component of the record such as a surname or part of an address.

Access is a relational database management system. This means that instead of having the data stored in one large file or table, it can be divided into several smaller tables. This reduces the amount of duplication of data and makes it easier to manage. The tables can be related to each other by a common field such as a case identification number. You can work with several tables at the same time.

Creating a Table

To create a table, make sure that the Tables object tab is selected then click on the New button. This presents you with a dialogue box with five choices:

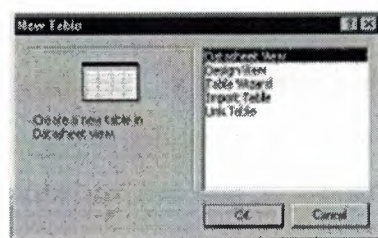


Figure 3.1 New Table

If you select Table Wizard, Access will help you to create a standard table by using pre-defined fields. Selecting Design View, on the other hand allows you to design your own table.

Select Design View and click OK to bring up the Table design window:

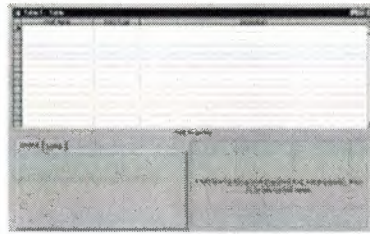


Figure 3.2 Table 1

In the lower right corner of this window is an area containing a brief explanation of the section containing the cursor? Field Names and

2.3.1 Data Types

Each field of the table has three components: the Field Name, which identifies the data stored in the field, the Data Type, which tells Access what kind of data will be stored in the field, such as text, numbers or dates, and the Description, which helps the user remember the purpose of the field. It is very important to choose the right data type for a field at this stage. Access will automatically insert a data type when you name a field but if you need to change it, click on the data type box, and then click on the downward arrow that appears to the right of the box. You can then choose a new data type from the drop-down list.

As you type in the name of each field, a Properties section appears in the lower left corner of the window where you can optionally define several properties for each field. Here you can specify the format of the data to be entered and specify criteria for validating the data as it is entered.

When you have defined the name, data type and properties of each field, save the table by choosing save from the File menu. At this point you will be informed that a Primary key has not been set and you will be asked if you wish Access to create one.

A table is a collection of data about a specific topic. Using a separate table for each topic means that you store that data only once, which makes your database more efficient, and reduces data-entry errors.

Tables organize data into columns (called **fields**) and rows (called **records**).

Each field in the Student Records table contains the same type of information for every student, such as student's Social Security Number. See Sec 4. This is an example of a **COLUMN**.

Student Records Table					
Soc Sec #	First Name	Last Name	BirthDate	Address	City
03456789	Todd	Jones	1/1/78	312 Wenona Rd	Bay City
015465866	Alan	Craig	2/8/80	123 N Union	Bay City
886585471	Stacy	Evans	3/8/81	RR 5 Box 880	Auburn
946131523	John	Anderson	4/5/80	83 Washington Dr.	Midland

Each record in a Student Records table contains all of the information about one student, such as their First Name, Last Name, Birthday, Address, and City, etc.... This is an example of a **ROW**.

Figure 3.3 Record

Primary Key

One or more fields (columns) whose value or values uniquely identify each record in a table. A primary key does not allow Null values and must always have a unique value. A primary key is used to relate a table to foreign keys in other tables.

A primary key field is a field that is used uniquely to identify each record. The field can be used by Access to manipulate data more efficiently. You do not have to specify a key. If you wish to use one, you can choose an existing field or, if you answer yes to the prompt for Access to create a Primary key, Access will create an extra field containing an identifier for each case.

Entering Data

Once you have defined the structure of a table, you can start to enter data into it. You created the table in Design view. To enter data, switch to Datasheet view by selecting Datasheet from the View menu. This produces a spreadsheet style window with the name of each field at the top of a column:



Figure 3.4 Column

The highlighted box or cell of the datasheet denotes the insertion point for data. If you requested Access to create a Primary key, the first cell is a counter called ID, which is automatically filled in by Access. Press TAB to move to the next cell containing your first Field

- **Saving Data:** You do not have to do anything special to save the data. When you leave a record to go to the next one or close the table, Access automatically saves any changes. To close the table choose Close from the File menu
- **Finding Data:** The simplest way to find a record in a table is: From the database window, open the table in Datasheet view. Select the field to search by clicking on its name box. Choose Find from the Edit menu. Enter the value you want to search for in the Find What? box and click on Find First.

You can also specify whether the search should match for case, in which direction to search and whether to search other fields. When you initiate a search, the cursor moves to the first occurrence of the search string in the table and the string is highlighted. To search for another occurrence of the same value, use the Find Next button.

2.3.4 Relationships

After you've set up multiple tables in your Microsoft Access database, you need a way of telling Access how to bring that information back together again. The first step in this process is to define relationships between your tables. After you have done that, you can create queries, forms, and reports to display information from several tables at once.

A relationship works by matching data in key fields - usually a field with the same name in both tables. In most cases, these matching fields are the primary key from one table, which provides a unique identifier for each record, and a foreign key in the other table.

For example, teachers can be associated with the students they are responsible for by creating a relationship between the teacher's table and the student's table using the TeacherID fields.

2.4 Forms

A form is nothing more than a graphical representation of a table. You can add, update, delete records in your table by using a form. NOTE: Although a form can be named different from a table, they both still manipulate the same information and the same exact data. Hence, if you change a record in a form, it will be changed in the table also.

A form is very good to use when you have numerous fields in a table. This way you can see all the fields in one screen, whereas if you were in the table view (datasheet) you would have to keep scrolling to get the field you desire.

2.5 Reports

A report is an effective way to present your data in a printed format. Because you have control over the size and appearance of everything on a report, you can display the information the way you want to see it.

CHAPTER 3

OPEN DATABASE CONNECTIVITY AND JAVA DATABASE CONNECTIVITY

3.1 Open DataBase Connectivity (ODBC)

Abbreviation of Open Database Connectivity, a standard database access method developed by Microsoft Corporation.

The aim of ODBC is to make it possible to access any data from any application, regardless of which database management system (DBMS) is handling the data. ODBC manages this by inserting a middle layer, called a database driver, between an application and the DBMS.

The purpose of this layer is to translate the application's data queries into commands that the DBMS understands. For this to work, both the application and the DBMS must be ODBC-compliant -- that is, the application must be capable of issuing ODBC commands and the DBMS must be capable of responding to them.

ODBC consists of two key components:

ODBC Driver Manager: an application binds to this generic library, which is responsible for loading the requested ODBC Driver.

ODBC Driver: dynamically loaded by the ODBC Driver manager for making connection to target Database.

3.1.1 Why use ODBC

ODBC enables maximum interoperability between the application and database as a single application can access any ODBC-enabled database by simply being configured to use its ODBC driver, and likewise an ODBC-enabled application can access a given

Database using its ODBC driver. The ODBC driver manager acts as the common interface enabling this dynamic switching to take place, thus giving application developers the database-independence they have always dreamed of.

3.2 JavaDataBaseConnectivity(JDBC)

Java Database Connectivity (JDBC) is an API specification for connecting programs written in Java to the data in popular databases. The API lets you encode access request statements in Structured Query Language (SQL) that are then passed to the program that manages the database. It returns the results through a similar interface. JDBC is very similar to ODBC and, with a small "bridge" program; you can use the JDBC interface to access databases through the ODBC interface.

3.2.1 Creating JDBC Application

The first thing you need to do is check that you are set up properly. This involves the following steps:

3.2.2 Establishing a Connection

The first thing you need to do is establish a connection with the DBMS you want to use.

This involves two steps:

Loading the driver.

Making the connection.

3.2.3 Loading Drivers

Loading the driver or drivers you want to use is very simple and involves just one line of code. If, for example, you want to use the JDBC-ODBC Bridge driver, the following code will load it:

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

Your driver documentation will give you the class name to use. For instance, if the class name is `jdbc.DriverXYZ`, you would load the driver with the following line of code:

```
Class.forName("jdbc.DriverXYZ");
```

You do not need to create an instance of a driver and register it with the `DriverManager` because calling `Class.forName` will do that for you automatically. If you were to create your own instance, you would be creating an unnecessary duplicate, but it would do no harm.

When you have loaded a driver, it is available for making a connection with a DBMS.

3.2.4 Making the Connection

The second step in establishing a connection is to have the appropriate driver connect to the DBMS. The following line of code illustrates the general idea:

```
Connection con = DriverManager.getConnection(url,  
    "myLogin", "myPassword");
```

This step is also simple, with the hardest thing being what to supply for URL. If you are using the JDBC-ODBC Bridge driver, the JDBC URL will start with `jdbc:odbc:`. The rest of the URL is generally your data source name or database system. So, if you are using ODBC to access an ODBC data source called "Fred," for example, your JDBC URL could be `jdbc:odbc:Fred`. In place of "myLogin" you put the name you use to log in to the DBMS; in place of "myPassword" you put your password for the DBMS.

3.2.5 Creating JDBC Statements

A Statement object is what sends your SQL statement to the DBMS. You simply create a Statement object and then execute it, supplying the appropriate execute method with the SQL statement you want to send. For a SELECT statement, the method to use is `executeQuery`. For statements that create or modify tables, the method to use is `executeUpdate`.

It takes an instance of an active connection to create a Statement object. In the following example, we use our Connection object `con` to create the Statement object `stmt`:

```
Statement stmt = con.createStatement();
```

At this point `stmt` exists, but it does not have an SQL statement to pass on to the DBMS. We need to supply that to the method we use to execute `stmt`. For example, in the following code fragment, we supply `executeUpdate` with the SQL statement.

3.2.6 Executing Statements

We used the method `executeUpdate` because the SQL statement is a DDL (data definition language) statement. Statements that create a table, alter a table, or drop a table are all examples of DDL statements and are executed with the method `executeUpdate`. As you might expect from its name, the method `executeUpdate` is also used to execute SQL statements that update a table. In practice, `executeUpdate` is used far more often to update tables than it is to create them because a table is created once but may be updated many times.

The method used most often for executing SQL statements is `executeQuery`. This method is used to execute SELECT statements, which comprise the vast majority of SQL statements. You will see how to use this method shortly.

3.2.7 Closing the statement and the connection

When finishing executing sql statement the statement and the connection were created should be close by close () method;

For example see the program outline below

```
conn = DriverManager.getConnection(url," "," "); // create a connection
stmt = conn.createStatement("any sql statement"); //create s statement
stmt.executeUpdate(); // execute the sql statement
stmt.close(); // close the statement
conn.close(); // close the connection
```

3.3 Using Statement Interface

3.3.1 Entering Data into a Table

I will enter our data into the table one row at a time, supplying the information to be stored in each column of that row. Note that the values to be inserted into the columns are listed in the same order that the columns were declared when the table was created, which is the default order.

If the SQL statement will not quite fit on one line on the page, we could split it into two strings concatenated by a plus sign (+) so that it will compile.

```
Statement stmt = con.createStatement();
stmt.executeUpdate(
    "INSERT INTO Tabele_Name +
    "VALUES (' ', , 0, 0)");
```

3.3.2 Getting Data from a Table

Now that the table has values in it, we can write a SELECT statement to access those values. The star (*) in the following SQL statement indicates that all columns should be selected. WHERE clause is used to narrow down the rows from which to select.

3.3.3 Retrieving Values from Result Sets

We now show how you send the above SELECT statements from a program written in the Java programming language and how you get the results we showed.

JDBC returns results in a ResultSet object, so we need to declare an instance of the class ResultSet to hold our results. The following code demonstrates declaring the ResultSet object rs and assigning the results of our earlier query to it:

```
ResultSet rs = stmt.executeQuery(  
    "SELECT * FROM Table_Name");
```

3.3.4 Using the Method next

The variable rs, which is an instance of ResultSet, in order to access the names and prices, we will go to each row and retrieve the values according to their types. The method next moves what is called a cursor to the next row and makes that row (called the current row) the one upon which we can operate. Since the cursor is initially positioned just above the first row of a ResultSet object, the first call to the method next moves the cursor to the first row and makes it the current row. Successive invocations of the method next move the cursor down one row at a time from top to bottom.

3.3.5 Using the getXXX Methods

We use the getXXX method of the appropriate type to retrieve the value in each column. For example, the method for retrieving a value of SQL type VARCHAR is getString, and the method for retrieving Float values is getFloat. The following code accesses the values stored in the current row of rs and prints a line with the name followed by three spaces and the price. Each time the method next is invoked, the next row becomes the current row, and the loop continues until there are no more rows in rs.

```
String query = "SELECT * FROM COFFEES";  
ResultSet rs = stmt.executeQuery(query);  
while (rs.next()) {
```

```
String s = rs.getString("String_value");
float n = rs.getFloat("Float_value");
System.out.println(s + " " + n);
}
```

JDBC offers two ways to identify the column from which a getXXX method gets a value. One-way is to give the column name, as was done in the example above. The second way is to give the column index (number of the column), with 1 signifying the first column, 2, the second, and so on. Using the column number instead of the column name looks like this:

```
String s = rs.getString(1);
float n = rs.getFloat(2);
```

3.3.6 Updating Tables

The SQL statement to update one row might look like this:

```
String updateString = "UPDATE Table_Name " +
"SET SALES = xyz " + "WHERE Condition";
```

Using the Statement object stmt, this JDBC code executes the SQL statement contained in updateString :

```
stmt.executeUpdate(updateString);
```

3.4 Using Prepared Statement Interface

Sometimes it is more convenient or more efficient to use a PreparedStatement object for sending SQL statements to the database. This special type of statement is derived from the more general class, Statement, which we already know.

3.4.1 When to Use a PreparedStatement Object

If you want to execute a Statement object many times, it will normally reduce execution time to use a PreparedStatement object instead.

The main feature of a PreparedStatement object is that, unlike a Statement object, it is given an SQL statement when it is created. The advantage to this is that in most cases, this SQL statement will be sent to the DBMS right away, where it will be compiled. As a result, the PreparedStatement object contains not just an SQL statement, but also an SQL statement that has been precompiled. This means that when the PreparedStatement is executed, the DBMS can just run the PreparedStatement's SQL statement without having to compile it first.

Although PreparedStatement objects can be used for SQL statements with no parameters, you will probably use them most often for SQL statements that take parameters. The advantage of using SQL statements that take parameters is that you can use the same statement and supply it with different values each time you execute it.

3.4.2 Creating a PreparedStatement Object

As with Statement objects, we create PreparedStatement objects with a Connection method. Using our open connection con from previous examples, you might write code such as the following to create a PreparedStatement object that takes two input parameters:

```
PreparedStatement deleteSales = con.prepareStatement(
```

```
    "UPDATE DELETE FROM Table_Name WHERE Condition =?");
```

The variable deleteSales now contains the SQL statement, "DELETE FROM Table_Name WHERE Condition =?", which has also, in most cases, been sent to the DBMS and been precompiled.

3.4.3 Supplying Values for PreparedStatement Parameters

You will need to supply values to be used in place of the question mark, if there are any, before you can execute a PreparedStatement object. You do this by calling one of the

setXXX methods defined in the class PreparedStatement . If the value you want to substitute for a question mark is a Java int , you call the method setInt. If the value you want to substitute for a question mark is a Java String, you call the method setString , and so on. In general, there is a setXXX method for each type in the Java programming language.

Using the PreparedStatement object deleteSales from the previous example, the following line of code sets the first question mark placeholder to a Java int with a value of 75:

```
updateSales.setInt(1, 75);
```

As you might surmise from the example, the first argument given to a setXXX method indicates which question mark is to be deleted.

```
PreparedStatement deleteSales = con.prepareStatement(  
    " UPDATE DELETE FROM Table_Name WHERE Condition =?");  
deleteSales.setInt(1, 75);  
deleteSales.executeUpdate();
```

We used the method executeUpdate to execute both the Statement stmt and the PreparedStatement updateSales. Notice, however, that no argument is supplied to executeUpdate when it is used to execute updateSales. This is true because updateSales already contains the SQL statement to be executed.

CHAPTER 4

STOCK DATABASE DESIGN

4.1 Data base design of the system

The stock database consists of three tables, supplier table, prouct table, and invoice table.

Supplier table contains four fields

- **Supplier code**
- **Supplier name**
- **Supplier city**
- **Supplier telephone**

Product table contains seven fields

- **Product code**
- **Product name**
- **Product price**
- **Product tax**
- **Product supplier**
- **Product date**
- **Product count**

Invoice table contains five fields

- **Invoice number**
- **Invoice date**
- **Invoice time**
- **Invoice product**

- **Invoice product count**

The relationships between tables will as follows

1-Product supplier of the product table is a foreign key to product supplier in the supplier table

2-Product code in invoice table is a foreign key to product code in product table.

The figure below shows the relationship.

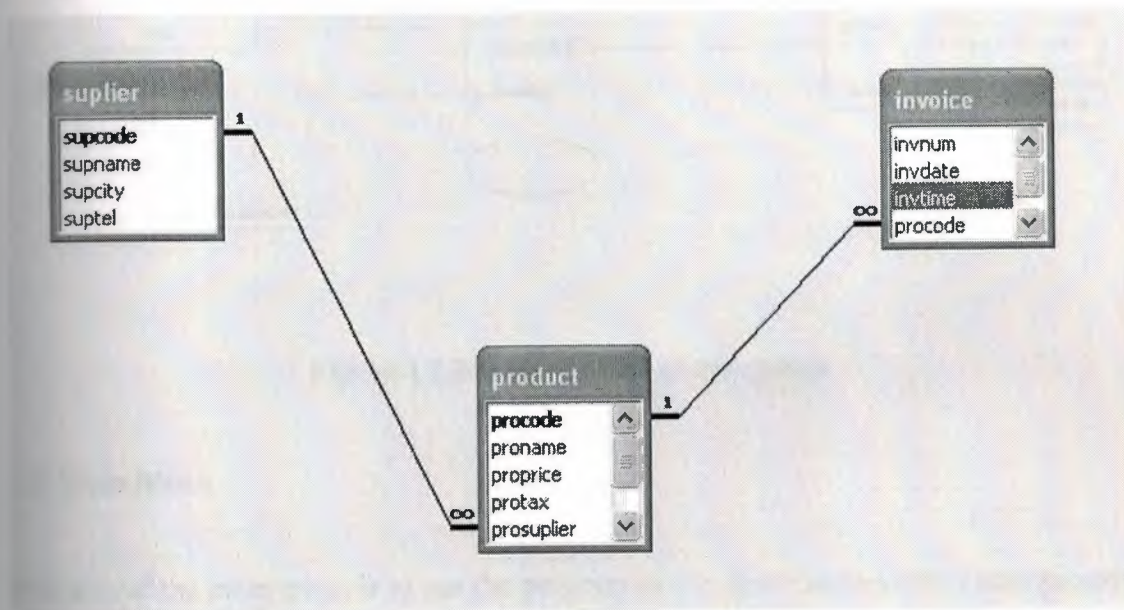


Figure 4.1 relationships between tables

4.2 Block diagram of the system

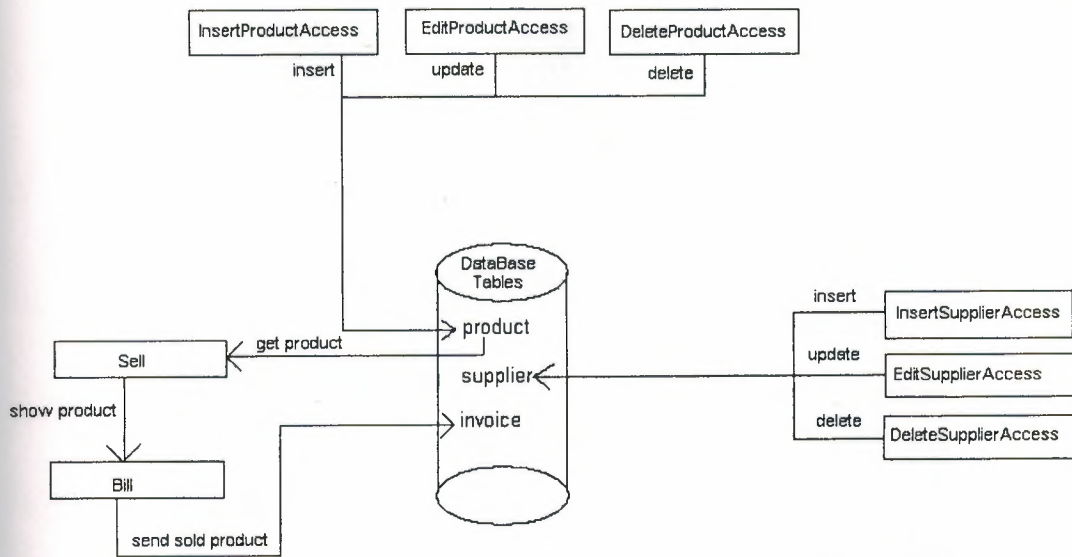


Figure 4.2 Block diagram of the system

4.3 Main Menu

The aim of the main menu is to use the program easily, faster and use the entire process screen. In main menu there is a menu bar contains four menus; each menu defines an operation on a particular object.



Fig 4.2 main menu

4.3.1 Add menu

Allows user to add product, supplier to the database or exit from the system.

4.3.1.1.Product

This menu item allows user to add item to the stock i.e. database

But, before adding any item to the database we have to add supplier for each item to be added.

Selecting this menu item will produce the following window.

Code	
Description	
Price	
Tax	
Supplier code	
Date	
Count	

Insert

Fig 4.4 insert product

Code: for entering a unique string for each item.

Description: for entering the supplier's code for this item.

Price: for entering the price.

Tax: for entering the tax.

Supplier Code: for entering the item's supplier code.

Date: for entering the date of purchasing.

Count: for entering the number of item sold by stock.

Insert: for insert supplier data to database.

4.3.1.2.Supplier

This menu item allows user to add supplier for product

Selecting this menu item will produce the following figure.

The image shows a graphical user interface window titled "insert supplier". The window has a standard title bar with a close button and two maximize/minimize buttons. Inside the window, there are four text input fields stacked vertically, each preceded by a label: "Supplier Code", "Supplier Name", "Supplier City", and "Supplier Tel". Below these input fields is a single button labeled "Insert".

Fig 4.4 insert supplier

Supplier Code: for entering unique supplier code.

Supplier Name: for entering supplier's name.

Supplier City: for entering supplier's city.

Supplier Tel: For entering supplier's telephone number.

Insert: for inserting this supplier to database.

4.3.1.3.Exit

This menu item allows user to exit from the program, by selecting this menu item the program terminates.

4.3.2. Edit menu

Allows user to edit product and supplier in database.

4.3.2.1.Product

This menu item allows user to modify product specification, first user should insert product code, press search button to find the particular product, product specifications will be appeared in text fields, and then user can edit what ever he/she wants by pressing edit button.

Edit product

Code

ID Number

Description

Price

Tax

Supplier code

Date

Count

Figure 4.5 edit product

4.3.2.2.Supplier

This menu item allows user to modify supplier's modifications, user should enter supplier's code, press search, and then change what ever he/she wants, finally press edit.

Edit supplier

Supplier Code

New Supplier Code

New Supplier Name

New Supplier City

New Supplier Tel

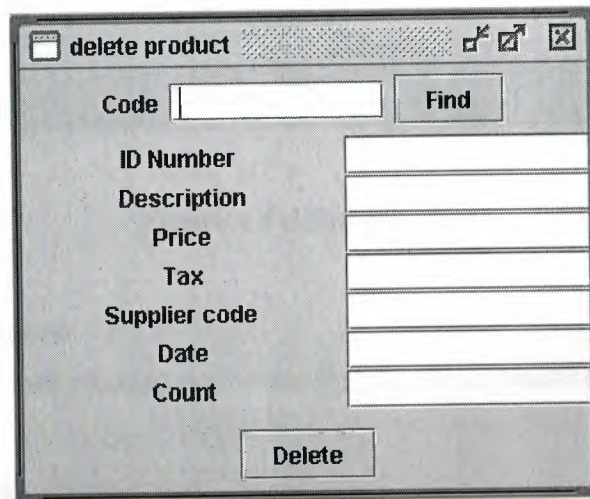
Figure 4.6 edit supplier

4.3.3. Delete menu

Allows user to delete product or supplier in the database.

4.3.3.1.Product

This menu item allows user to delete product from the database, user should enter the product's code, press find button, to show the product's details, then press delete button.



The screenshot shows a Windows-style dialog box titled "delete product". At the top, there is a "Code" text box and a "Find" button. Below these, the following fields are listed vertically: "ID Number", "Description", "Price", "Tax", "Supplier code", "Date", and "Count", each with its own input box. At the bottom center of the dialog is a "Delete" button. The dialog box has a standard title bar with minimize, maximize, and close buttons.

Figure 4.7 delete product

4.3.3.2.Supplier

This menu item allows user to delete particular product's supplier, first user should enter supplier's code, press enter, supplier's details will be appeared, and then by pressing delete button particular supplier will be deleted.

Delete supplier

Supplier Code **Find**

New Supplier Code

New Supplier Name

New Supplier City

New Supplier Tel

Delete

Figure 4.8 delete supplier

4.3.4. Sales menu

Allows user to sell products and to see the purchased products.

4.3.4.1.Show Sales

This menu item allows user to see all the sales of the stock, number of invoices, and the amount of sold products.

Reports

REPORTS

Invoice_Numb...	Date	Time	Product_Code	Product_Count
-----------------	------	------	--------------	---------------

FIND

TOT Sales:

TOT Invoices:

Sales Count:

Figure 4.9 reports

4.3.4.2.Sell

This menu item allows user to sell product to customer, by clicking this menu item two windows appear; one is sell ,the other is bill, user should enter the product code into (pro_code) field, with the amount of the product to be sold, then user should press add button ,automatically product name, amount tax, and vat will be added to the bill, user continue adding items till them finished, if the amount is not entered it is assumed to be 1, then user presses sum to find the sum of the product ,a customer give the user(cashier) the amount required ,user enter the amount received from the customer into Money Paid field then press print button ,if the money received by user exceeds the amount required then bill will show the remained money to be refunded to the customer, if the user does a

mistake he/she can create a new bill by pressing the new bill button, sold items will be added to invoice table.

Pro_Code	
Count	1
Money_Paid	

Add New Bill Sum Print

Figure 4.10 Sell

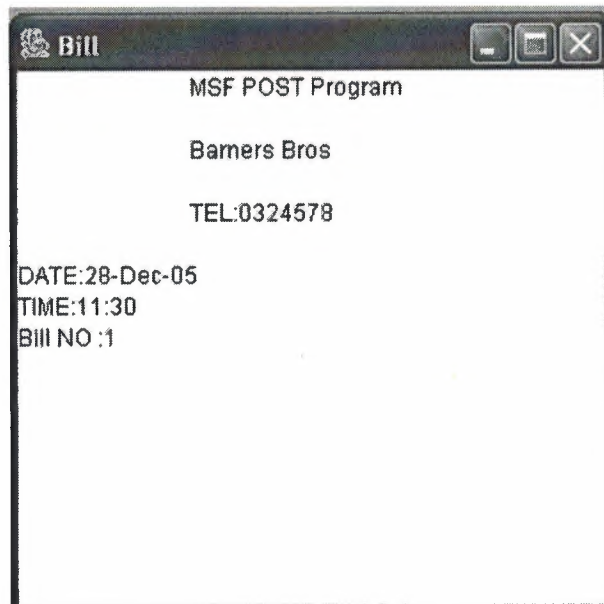


Figure 4.11 Bill

4.3.5. About menu

Contains only one menu item

4.3.5.1.About

Gives the user a brief description about the program, programmer, etc.....

CONCLUSION

Computers have entered to many fields, education, economy, factories, this program designed to satisfy user's demand for controlling stock sales.

This project helps stock managers to control easily efficiently all the transaction accruing in the stock

High technology programming language is used, with user friendly interfaces to communicate with the user, also a database system is used to save, add, edit delete elements in the database, and so each sale is recorded in the database for referencing later.

Some development will be added to this software, some refinements, and modifications will be added to increase the efficiency and performance activity.

REFERENCES

- [1] Deitel & Deitel "Java How to Program".
- [2] Java tutorials "<http://java.sun.com/docs/books/tutorial.com>".
- [3] What is ODBC "www.goradno.com".
- [4] Open Database Connectivity "www.openlinksoftware.com".

APPENDIX A

Main menu

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;

public class MainMenu extends JFrame implements ActionListener
{
    JDesktopPane desktop;
    MainMenu m;
    JMenuBar bar;
    JMenu menu1, menu2, menu3, menu4, menu5;
    JMenuItem mi1, mi2, mi3, mi4, mi5, mi6, mi7, mi8, mi9, mi10;
    Container c;
    InsertSupplierAccess insupplier;
    InsertProductAccess insproduct;
    EditSupplierAccess edsupplier;
    EditProductAccess edproduct;
    DeleteSupplierAccess delsupplier;
    DeleteProductAccess delproduct;
    Reports rp;
    Sell sel;
    Bill bil;

    public static void main(String args[])
    {
        MainMenu mm = new MainMenu();
        // mm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    public MainMenu()
```



```
{super("MainMenu");
c = getContentPane();
desktop = new JDesktopPane();
c.add(desktop);
bar = new JMenuBar();

menu1 = new JMenu("Add");
bar.add(menu1);

mi1 = new JMenuItem("Product");
menu1.add(mi1);
mi1.addActionListener(this);
menu1.addSeparator();

mi2 = new JMenuItem("Supplier");
menu1.add(mi2);
mi2.addActionListener(this);
menu1.addSeparator();

mi3 = new JMenuItem("Exit");
menu1.add(mi3);
mi3.addActionListener(this);

menu2 = new JMenu("Edit");
bar.add(menu2);

mi4 = new JMenuItem("Product");
menu2.add(mi4);
mi4.addActionListener(this);
menu2.addSeparator();
```

```
mi5 = new JMenuItem("Supplier");  
menu2.add(mi5);  
mi5.addActionListener(this);
```

```
menu3 = new JMenu("Delete");  
bar.add(menu3);
```

```
mi6 = new JMenuItem("Product");  
menu3.add(mi6);  
mi6.addActionListener(this);  
menu3.addSeparator();
```

```
mi7 = new JMenuItem("Supplier");  
mi7.addActionListener(this);  
menu3.add(mi7);
```

```
menu4 = new JMenu("Sales");  
bar.add(menu4);
```

```
mi8 = new JMenuItem("Show Sales");  
mi8.addActionListener(this);  
menu4.add(mi8);  
menu4.addSeparator();
```

```
mi10 = new JMenuItem("Sell");  
mi10.addActionListener(this);  
menu4.add(mi10);
```

```
menu5 = new JMenu("About");
```

```

bar.add(menu5);

mi9= new JMenuItem("About Me");
menu5.add(mi9);
mi9.addActionListener(this);

this.setJMenuBar(bar);
setVisible(true);
setSize(400,500);

this.addWindowListener(new WindowAdapter()
    {
        public void windowClosing(WindowEvent event)
        {System.exit(0);}
    }
);

}

public void actionPerformed(ActionEvent e)
{
    if(e.getSource()== mi1)
    {insproduct = new InsertProductAccess();
    insproduct.setLocation(200,200);
    insproduct.setSize(300,250);
    insproduct.setVisible(true);
    desktop.add(insproduct);

    }
}

```

```
if(e.getSource() == mi2)
{insupplier = new InsertSupplierAccess();
insupplier.setLocation(200,200);
insupplier.setSize(300,250);
insupplier.setVisible(true);
desktop.add(insupplier);
}
```

```
if(e.getSource() == mi3)
{System.exit(0);
}
```

```
if(e.getSource() ==mi4)
{edproduct = new EditProductAccess();
edproduct.setLocation(200,200);
edproduct.setSize(300,250);
edproduct.setVisible(true);
desktop.add(edproduct);
}
```

```
if(e.getSource() == mi5)
{
edsupplier = new EditSupplierAccess();
edsupplier.setSize(300,250);
edsupplier.setVisible(true);
edsupplier.setLocation(200,200);
desktop.add(edsupplier);
}
```

```
if(e.getSource() == mi6)
{
```



```

delproduct = new DeleteProductAccess();
delproduct.setSize(300,250);
delproduct.setVisible(true);
delproduct.setLocation(200,200);
desktop.add(delproduct);

}

if(e.getSource() == mi7)
{
delsupplier = new DeleteSupplierAccess();
delsupplier.setSize(300,250);
delsupplier.setVisible(true);
delsupplier.setLocation(200,200);
desktop.add(delsupplier);
}

if(e.getSource() == mi8)
{
rp = new Reports();
rp.setSize(600,600);
rp.setVisible(true);
rp.setLocation(200,200);
desktop.add(rp);
}

if(e.getSource() == mi9)
{
JOptionPane.showMessageDialog(m,"This program is Made Eng.Mohammad
Elfawair","About",JOptionPane.INFORMATION_MESSAGE);
}

if(e.getSource() == mi10)

```

```

{
    sel = new Sell();
    sel.setSize(400,300);
    sel.setVisible(true);
    sel.setLocation(200,200);
    desktop.add(sel);

}
}
}

```

Insert product access

```

import javax.swing.*.*;
import java.sql.*;
import java.awt.*.*;
import java.awt.event.*;

```

```

public class InsertProductAccess extends JFrame implements ActionListener

```

```

{
    JTextField text1,text2,text3,text4,text5,text6,text7;
    JLabel l1,l2,l3,l4,l5,l6,l7;
    JButton ins;
    JPanel panel1,panel2;
    Container c;
    Connection conn;
    PreparedStatement pstmt;
    String url = "jdbc:odbc:msf";
    InsertProductAccess application;

```

```

public InsertProductAccess()

```

```

{

super("insert product",true,true,true,true);
    c = getContentPane();
    panel1 = new JPanel();
    panel2 = new JPanel();

    panel2.setLayout(new GridLayout(7,2));

    l1 = new JLabel("Code",JLabel.CENTER);
    l2 = new JLabel("Description",JLabel.CENTER);
    l3 = new JLabel("Price",JLabel.CENTER);
    l4 = new JLabel("Tax",JLabel.CENTER);
    l5 = new JLabel("Supplier code ",JLabel.CENTER );
    l6 = new JLabel("Date",JLabel.CENTER);
    l7 = new JLabel("Count",JLabel.CENTER);

    text1 = new JTextField(10);
    text2 = new JTextField(15);
    text3 = new JTextField(6);
    text4 = new JTextField(5);
    text5 = new JTextField(10);
    text6 = new JTextField(11);
    text7 = new JTextField(3);

    ins = new JButton("Insert");

```



```
panel2.add(l1);
panel2.add(text1);
panel2.add(l2);
panel2.add(text2);
panel2.add(l3);
panel2.add(text3);
panel2.add(l4);
panel2.add(text4);
panel2.add(l5);
panel2.add(text5);
panel2.add(l6);
panel2.add(text6);
panel2.add(l7);
panel2.add(text7);

panel1.add(ins);

c.add(panel2,BorderLayout.NORTH);
c.add(panel1,BorderLayout.CENTER);

ins.addActionListener(this);

setSize(300,250);
setVisible(true);

} // end of Msf constructuer
```



```
public void actionPerformed( ActionEvent e)
{
```

```
    if( e.getSource() == ins )
    {
```

```
        try
        { Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

        }
    }
```

```
        catch(Exception exp)
        {
```

```
            JOptionPane.showMessageDialog(application,exp.getMessage(),"WARNING",JOptionPane.
            WARNING_MESSAGE);
        }
```

```
        try
        {
            conn = DriverManager.getConnection(url," "," ");
            pstmt = conn.prepareStatement("insert into product values( ? , ? , ? , ? , ? , ? , ?
        )");
```

```

        if(text1.getText().equals("") || text2.getText().equals("") ||
text3.getText().equals("") || text4.getText().equals("") || text5.getText().equals("") ||
text6.getText().equals("") || text7.getText().equals("") )
        { JOptionPane.showMessageDialog(application,"Fill all fields","Some Fields
are missing",JOptionPane.WARNING_MESSAGE);
        }

```

```

else
{

```

```

String code = text1.getText().trim();
String nam = text2.getText().trim();
double pric = Double.parseDouble(text3.getText().trim());
double tax = Double.parseDouble(text4.getText().trim());
String supco = text5.getText().trim();
String dat = text6.getText().trim();
int num = Integer.parseInt(text7.getText());

```

```

pstmt.setString(1,code);
pstmt.setString(2,nam);
pstmt.setDouble(3,pric);
pstmt.setDouble(4,tax);
pstmt.setString(5,supco);
pstmt.setString(6,dat);
pstmt.setInt(7,num);

```

```

pstmt.executeUpdate();
pstmt.close();
conn.close();

```

```
        JOptionPane.showMessageDialog(application," Insert done perfectly  
", "INFORMATION",JOptionPane.INFORMATION_MESSAGE);
```

```
        text1.setText("");
```

```
        text2.setText("");
```

```
        text3.setText("");
```

```
        text4.setText("");
```

```
        text5.setText("");
```

```
        text6.setText("");
```

```
        text7.setText("");
```

```
    }
```

```
}
```

```
        catch(SQLException ex )
```

```
{
```

```
        JOptionPane.showMessageDialog(application,"SQLERROR:" +  
ex.getMessage() ,"Data Base ERROR" ,JOptionPane.ERROR_MESSAGE);
```

```
}
```

```
    }// if cancel pressed
```

```
// end of action performed
```

```
}
```

Insert supplier access

```
import javax.swing.*;
```

```
import java.sql.*;
```

```
import java.awt.*;
```

```
import java.awt.event.*;
```

```
import javax.swing.event.*;
```

```
import java.beans.PropertyVetoException;
```

```
public class InsertSupplierAccess extends JFrame implements ActionListener
```

```
{
```

```
    JTextField text1,text2,text3,text4;
```

```
    JLabel l1,l2,l3,l4;
```

```
    JButton ins;
```

```
    JPanel panel1,panel2;
```

```
    Container c;
```

```
    Connection conn;
```

```
    PreparedStatement stmt;
```

```
    String url = "jdbc:odbc:msf";
```

```
    InsertSupplierAccess application;
```

```
public InsertSupplierAccess(){
```

```
    super("insert supplier",true,true,true,true);
```

```
    c = getContentPane();
```

```
    panel1 = new JPanel();
```

```
    panel2 = new JPanel();
```



```
panel2.setLayout(new GridLayout(4,2));
```

```
l1 = new JLabel("Supplier Code",JLabel.CENTER);
```

```
l2 = new JLabel("Supplier Name",JLabel.CENTER);
```

```
l3 = new JLabel("Supplier City",JLabel.CENTER);
```

```
l4 = new JLabel("Supplier Tel",JLabel.CENTER);
```

```
text1 = new JTextField(10);
```

```
text2 = new JTextField(15);
```

```
text3 = new JTextField(6);
```

```
text4 = new JTextField(5);
```

```
ins = new JButton("Insert");
```

```
panel2.add(l1);
```

```
panel2.add(text1);
```

```
panel2.add(l2);
```

```
panel2.add(text2);
```

```
panel2.add(l3);
```

```
panel2.add(text3);
```

```
panel2.add(l4);
```

```
panel2.add(text4);
```

```
panel1.add(ins);
```

```
c.add(panel2,BorderLayout.NORTH);  
c.add(panel1,BorderLayout.CENTER);
```

```
ins.addActionListener(this);
```

```
setSize(300,200);  
setVisible(true);
```

```
} // end of Msf constructuer
```

```
public void actionPerformed (ActionEvent e)
```

```
{String code,nam,city,tel;
```

```
if( e.getSource() == ins )
```

```
{
```

```
try
```

```
{ Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

```
}
```

```
catch(Exception exp)
```

```
{
```

```
JOptionPane.showMessageDialog(application,exp.getMessage(),"WARNING",JOptionPane.WARNING_MESSAGE);
```

```
}
```

```
try
```

```
{ conn = DriverManager.getConnection(url," "," ");
```

```
stmt = conn.prepareStatement("insert into supplier values( ?, ?, ?, ? )");
```

```
if(text1.getText().equals("") || text2.getText().equals("") || text3.getText().equals("") || text4.getText().equals("") )
```

```
JOptionPane.showMessageDialog(application,"Fill all fields","Some Fields are missing",JOptionPane.WARNING_MESSAGE);
```

```
else
```

```
{ code = (text1.getText().trim());
```

```
nam = (text2.getText().trim());
```

```
city = (text3.getText().trim());
```

```
tel = (text4.getText().trim());
```

```
stmt.setString(1,code);
```

```
stmt.setString(2,nam);
```

```
stmt.setString(3,city);
```

```
stmt.setString(4,tel);
```

```
stmt.executeUpdate();
```

```
stmt.close();
```

```
conn.close();
```



```

import javax.swing.*;
import java.sql.*;
import java.awt.*;
import java.awt.event.*;

public class EditSupplierAccess extends JFrame implements ActionListener
{
    JTextField text1,text2,text3,text4,text5;
    JLabel l1,l2,l3,l4,l5;
    JButton edit,srch;
    JPanel panel1,panel2,panel3;
    Container c;
    Connection conn;
    PreparedStatement pstmt;
    String url = "jdbc:odbc:msf";
    EditSupplierAccess application;
    Statement stmt;

    public EditSupplierAccess()
    {
        super("Edit supplier",true,true,true,true);
        c = getContentPane();
        panel1 = new JPanel();
        panel2 = new JPanel();
        panel3 = new JPanel();

        panel3.setLayout(new FlowLayout(FlowLayout.CENTER));
        panel2.setLayout(new GridLayout(4,2));
        panel1.setLayout(new FlowLayout(FlowLayout.CENTER));
    }

```

```
l1 = new JLabel("Supplier Code",JLabel.CENTER);
l2 = new JLabel("New Supplier Code",JLabel.CENTER);
l3 = new JLabel("New Supplier Name",JLabel.CENTER);
l4 = new JLabel("New Supplier City",JLabel.CENTER);
l5 = new JLabel("New Supplier Tel",JLabel.CENTER);
```

```
text1 = new JTextField("",10);
text2 = new JTextField("",10);
text3 = new JTextField("",15);
text4 = new JTextField("",6);
text5 = new JTextField("",5);
```

```
edit = new JButton("Edit");
srch = new JButton("Search");
```

```
panel3.add(l1);
panel3.add(text1);
panel3.add(srch);
```

```
panel2.add(l2);
panel2.add(text2);
panel2.add(l3);
panel2.add(text3);
panel2.add(l4);
panel2.add(text4);
panel2.add(l5);
panel2.add(text5);
```

```
panel1.add(edit);
```

```
c.add(panel3, BorderLayout.NORTH);
```

```
c.add(panel2, BorderLayout.WEST);
```

```
c.add(panel1, BorderLayout.SOUTH);
```

```
edit.addActionListener(this);
```

```
srch.addActionListener(this);
```

```
setSize(300,250);
```

```
setVisible(true);
```

```
} // end of constructuer
```

```
public void actionPerformed( ActionEvent e)
```

```
{ String nam,supcty,suptel,cd,id;
```

```
if (e.getSource() == srch )
```

```
{
```

```
    cd = text1.getText().trim();
```

```
    try
```

```
    { Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

```
    }
```

```
    catch(Exception exp)
```

```
    {
```

```
JOptionPane.showMessageDialog(application,exp.getMessage(),"WARNING",JOptionPane.WARNING_MESSAGE);
```

```
}
```

```
try
```

```
{ conn = DriverManager.getConnection(url," "," ");
```

```
stmt = conn.createStatement();
```

```
String s = "SELECT * FROM supplier " + "WHERE supcode = " + cd + "";
```

```
ResultSet rset = stmt.executeQuery(s);
```

```
boolean f = rset.next();
```

```
if(f)
```

```
{
```

```
text2.setText(rset.getString(1));
```

```
text3.setText(rset.getString(2));
```

```
text4.setText(rset.getString(3));
```

```
text5.setText(rset.getString(4));
```

```
}
```

```
else
```

```
{
```

```
JOptionPane.showMessageDialog(application,"Not Found....!\nTry Again","MESSAGE",JOptionPane.INFORMATION_MESSAGE);
```

```
}
```



```

    }

    catch(SQLException ex )
    {
        JOptionPane.showMessageDialog(application,"SQLERROR:" +
ex.getMessage() ,"Data Base ERROR" ,JOptionPane.ERROR_MESSAGE);

    }

}

if( e.getSource() == edit )
{

    try
    { Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

    }

    catch(Exception exp)
    {

JOptionPane.showMessageDialog(application,exp.getMessage(),"WARNING",JOptionP
ane.WARNING_MESSAGE);

    }

```

```

try
{

    conn = DriverManager.getConnection(url," "," ");
    pstmt = conn.prepareStatement("update supplier set supcode = ?, supname = ?
, supcity = ? , supitel = ? where supcode = ? ");

    if(text2.getText().equals("") || text3.getText().equals("") || text4.getText().equals("") ||
text5.getText().equals("") )
        { JOptionPane.showMessageDialog(application,"Fill all fields","Some Fields are
missing",JOptionPane.WARNING_MESSAGE);
        }
    else
        {

            cd = (text1.getText().trim());
            id = (text2.getText().trim());
            nam = (text3.getText().trim());
            supcity = (text4.getText().trim());
            supitel = (text5.getText().trim());

            pstmt.setString(1,id);
            pstmt.setString(2,nam);
            pstmt.setString(3,supcity);
            pstmt.setString(4,supitel);
            pstmt.setString(5,cd);
            pstmt.executeUpdate();
        }
    }
}

```

```
pstmt.close();  
conn.close();
```

```
JOptionPane.showMessageDialog(application," Edit done perfectly  
","INFORMATION",JOptionPane.INFORMATION_MESSAGE);
```

```
text1.setText("");  
text2.setText("");  
text3.setText("");  
text4.setText("");  
text5.setText("");
```

```
}
```

```
}
```

```
catch(SQLException ex )  
{  
    JOptionPane.showMessageDialog(application,"SQLERROR:" +  
ex.getMessage() ,"Data Base ERROR" ,JOptionPane.ERROR_MESSAGE);  
}
```

```
// if cancel pressed
```

```
}// end of action performed
```

```
}
```

Edit supplier access

```
import javax.swing.*;
```

```
import java.sql.*;
```

```
import java.awt.*;
```

```
import java.awt.event.*;
```

```
public class EditSupplierAccess extends JFrame implements ActionListener
```

```
{
```

```
    JTextField text1,text2,text3,text4,text5;
```

```
    JLabel l1,l2,l3,l4,l5;
```

```
    JButton edit,srch;
```

```
    JPanel panel1,panel2,panel3;
```

```
    Container c;
```

```
    Connection conn;
```

```
    PreparedStatement pstmt;
```

```
    String url = "jdbc:odbc:msf";
```

```
    EditSupplierAccess application;
```

```
    Statement stmt;
```

```
public EditSupplierAccess()
```

```
{
```

```
    super("Edit supplier",true,true,true,true);
```

```
    c = getContentPane();
```

```
    panel1 = new JPanel();
```

```
    panel2 = new JPanel();
```



```
panel3 = new JPanel();
```

```
panel3.setLayout(new FlowLayout(FlowLayout.CENTER));
```

```
panel2.setLayout(new GridLayout(4,2));
```

```
panel1.setLayout(new FlowLayout(FlowLayout.CENTER));
```

```
l1 = new JLabel("Supplier Code",JLabel.CENTER);
```

```
l2 = new JLabel("New Supplier Code",JLabel.CENTER);
```

```
l3 = new JLabel("New Supplier Name",JLabel.CENTER);
```

```
l4 = new JLabel("New Supplier City",JLabel.CENTER);
```

```
l5 = new JLabel("New Supplier Tel",JLabel.CENTER);
```

```
text1 = new JTextField("",10);
```

```
text2 = new JTextField("",10);
```

```
text3 = new JTextField("",15);
```

```
text4 = new JTextField("",6);
```

```
text5 = new JTextField("",5);
```

```
edit = new JButton("Edit");
```

```
srch = new JButton("Search");
```

```
panel3.add(l1);
```

```
panel3.add(text1);
```

```
panel3.add(srch);
```

```
panel2.add(l2);
```

```
panel2.add(text2);
```

```
panel2.add(l3);
panel2.add(text3);
panel2.add(l4);
panel2.add(text4);
panel2.add(l5);
panel2.add(text5);
```

```
panel1.add(edit);
```

```
c.add(panel3,BorderLayout.NORTH);
c.add(panel2,BorderLayout.WEST);
c.add(panel1,BorderLayout.SOUTH);
```

```
edit.addActionListener(this);
srch.addActionListener(this);
```

```
setSize(300,250);
setVisible(true);
```

```
} // end of constructuer
```

```
public void actionPerformed( ActionEvent e)
```

```
{ String nam,supcty,suptel,cd,id;
```

```
if (e.getSource() == srch )
```

```
{
```

```
    cd = text1.getText().trim();
```

```
    try
```

```

{ Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

}

catch(Exception exp)
{

JOptionPane.showMessageDialog(application,exp.getMessage(),"WARNING",JOptionPane.WARNING_MESSAGE);

}

try
{ conn = DriverManager.getConnection(url,"","");
  stmt = conn.createStatement();
  String s = "SELECT * FROM suplier " + "WHERE supcode = '" + cd + "'";
  ResultSet rset = stmt.executeQuery(s);

  boolean f=rset.next();

  if(f)
  {

      text2.setText(rset.getString(1));
      text3.setText(rset.getString(2));
      text4.setText(rset.getString(3));
      text5.setText(rset.getString(4));

  }
}

```

```

else
{
    JOptionPane.showMessageDialog(application,"Not Found....!\nTry
Again","MESSAGE" ,JOptionPane.INFORMATION_MESSAGE);
}

```

```

}

```

```

catch(SQLException ex )
{
    JOptionPane.showMessageDialog(application,"SQLERROR:" +
ex.getMessage() ,"Data Base ERROR" ,JOptionPane.ERROR_MESSAGE);
}

```

```

}

```

```

if( e.getSource() == edit )
{

```

```

    try
    { Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

    }

```

```

    catch(Exception exp)
    {

```



```
JOptionPane.showMessageDialog(application,exp.getMessage(),"WARNING",JOptionPane.WARNING_MESSAGE);
```

```
}
```

```
try
```

```
{
```

```
    conn = DriverManager.getConnection(url," "," ");
```

```
    pstmt = conn.prepareStatement("update supplier set suppliercode = ?, suppliername = ?, suppliercity = ?, supplierphone = ? where suppliercode = ? ");
```

```
    if(text2.getText().equals("") || text3.getText().equals("") || text4.getText().equals("") || text5.getText().equals(""))
```

```
    { JOptionPane.showMessageDialog(application,"Fill all fields","Some Fields are missing",JOptionPane.WARNING_MESSAGE);
```

```
    }
```

```
else
```

```
{
```

```
    cd = (text1.getText().trim());
```

```
    id = (text2.getText().trim());
```

```
    nam = (text3.getText().trim());
```

```
    suppliercity = (text4.getText().trim());
```

```
    supplierphone = (text5.getText().trim());
```

```
    pstmt.setString(1,id);
```

```
pstmt.setString(2,nam);  
pstmt.setString(3,supcty);  
pstmt.setString(4,suptel);  
pstmt.setString(5,cd);  
pstmt.executeUpdate();
```

```
pstmt.close();  
conn.close();
```

```
JOptionPane.showMessageDialog(application," Edit done perfectly  
","INFORMATION",JOptionPane.INFORMATION_MESSAGE);
```

```
text1.setText("");  
text2.setText("");  
text3.setText("");  
text4.setText("");  
text5.setText("");
```

```
}
```

```
}
```

```
catch(SQLException ex )  
{  
    JOptionPane.showMessageDialog(application,"SQLERROR:" +  
ex.getMessage() ,"Data Base ERROR" ,JOptionPane.ERROR_MESSAGE);  
}
```

```

    }// if cancel pressed

}

}

Delete product access
import javax.swing.*;
import java.sql.*;
import java.awt.*;
import java.awt.event.*;

public class DeleteProductAccess extends JFrame implements ActionListener
{
    JTextField text1,text2,text3,text4,text5,text6,text7,text8;
    JLabel l1,l2,l3,l4,l5,l6,l7,l8;
    JButton delete,find;
    JPanel panel1,panel2,panel3;
    Container c;
    Connection conn;
    PreparedStatement pstmt;
    String url = "jdbc:odbc:msf";
    DeleteProductAccess application;
    Statement stmt;

    public DeleteProductAccess()

```

```

{
super("delete product",true,true,true,true);
    c = getContentPane();
    panel1 = new JPanel();
    panel2 = new JPanel();
    panel3 = new JPanel();

    panel3.setLayout(new FlowLayout(FlowLayout.CENTER));
    panel2.setLayout(new GridLayout(7,2));
    panel1.setLayout(new FlowLayout(FlowLayout.CENTER));

    l1 = new JLabel("Code",JLabel.CENTER);
    l2 = new JLabel("Description",JLabel.CENTER);
    l3 = new JLabel("Price",JLabel.CENTER);
    l4 = new JLabel("Tax",JLabel.CENTER);
    l5 = new JLabel("Supplier code ",JLabel.CENTER );
    l6 = new JLabel("Date",JLabel.CENTER);
    l7 = new JLabel("Count",JLabel.CENTER);
    l8 = new JLabel("ID Number ",JLabel.CENTER);

    text1 = new JTextField(" ",10);
    text2 = new JTextField(" ",15);
    text3 = new JTextField(" ",6);
    text4 = new JTextField(" ",5);
    text5 = new JTextField(" ",10);
    text6 = new JTextField(" ",11);
    text7 = new JTextField(" ",5);
    text8 = new JTextField(" ",10);

    delete = new JButton("Delete");

```



```
find = new JButton("Find");
```

```
panel3.add(l1);  
panel3.add(text1);  
panel3.add(find);
```

```
panel2.add(l8);  
panel2.add(text8);  
panel2.add(l2);  
panel2.add(text2);  
panel2.add(l3);  
panel2.add(text3);  
panel2.add(l4);  
panel2.add(text4);  
panel2.add(l5);  
panel2.add(text5);  
panel2.add(l6);  
panel2.add(text6);  
panel2.add(l7);  
panel2.add(text7);
```

```
panel1.add(delete);
```

```
c.add(panel3,BorderLayout.NORTH);  
c.add(panel2,BorderLayout.WEST);  
c.add(panel1,BorderLayout.SOUTH);
```

```
delete.addActionListener(this);
```

```
find.addActionListener(this);
```

```
setSize(300,250);
```

```
setVisible(true);
```

```
} // end of Msf constructuer
```

```
public void actionPerformed( ActionEvent e)
```

```
{
```

```
    String cd = text1.getText().trim();
```

```
    if (e.getSource() == find )
```

```
    {
```

```
        try
```

```
        { Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

```
        }
```

```
        catch(Exception exp)
```

```
        {
```

```
            JOptionPane.showMessageDialog(application,exp.getMessage(),"WARNING",JOptionPane.WARNING_MESSAGE);
```

```
        }
```

```

try
{
    conn = DriverManager.getConnection(url, " ", " ");
    stmt = conn.createStatement();

    String s = "SELECT * FROM product " + "WHERE procode = '" + cd + "'";

    ResultSet rset = stmt.executeQuery(s);

    boolean f = rset.next();

    if(f)
    {
        text1.setText(rset.getString("procode"));
        text8.setText(text1.getText());
        text2.setText(rset.getString("praname"));
        text3.setText(String.valueOf(rset.getDouble("proprice")));
        text4.setText(String.valueOf(rset.getDouble("protax")));
        text5.setText(rset.getString("prosupplier"));
        text6.setText(String.valueOf(rset.getDate("prodate")));
        text7.setText(String.valueOf(rset.getInt("procount")));

    }

    else
    {
        JOptionPane.showMessageDialog(application, "Not Found....!\nTry Again", "MESSAGE" ,JOptionPane.INFORMATION_MESSAGE);
    }
}

```

```

    }

}

catch(SQLException ex )
{
    JOptionPane.showMessageDialog(application,"SQLException:" +
ex.getMessage() ,"Data Base ERROR" ,JOptionPane.ERROR_MESSAGE);

}

}

if( e.getSource() == delete )
{

    try
    { Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

    }

    catch(Exception exp)
    {

JOptionPane.showMessageDialog(application,exp.getMessage(),"WARNING",JOptionPane.WARNING_MESSAGE);

    }
}

```



```

try
{

    conn = DriverManager.getConnection(url, " ", " ");
    pstmt = conn.prepareStatement("DELETE FROM product WHERE procode
= ? ");

    pstmt.setString(1,cd);
    pstmt.executeUpdate();

    pstmt.close();
    conn.close();

    JOptionPane.showMessageDialog(application," Delete done perfectly
", "INFORMATION",JOptionPane.INFORMATION_MESSAGE);
    text1.setText("");
    text2.setText("");
    text3.setText("");
    text4.setText("");
    text5.setText("");
    text6.setText("");
    text7.setText("");
    text8.setText("");

}

catch(SQLException ex )
{

```

```
JOptionPane.showMessageDialog(application,"SQLERROR:" +  
ex.getMessage() ,"Data Base ERROR" ,JOptionPane.ERROR_MESSAGE);
```

```
}
```

```
}// if cancel pressed
```

```
}// end of action performed
```

```
}
```

Delete supplier access

```
import javax.swing.*;
```

```
import java.sql.*;
```

```
import java.awt.*;
```

```
import java.awt.event.*;
```

```
public class DeleteSupplierAccess extends JFrame implements ActionListener  
{
```

```
    JTextField text1,text2,text3,text4,text5;
```

```
    JLabel l1,l2,l3,l4,l5;
```

```
    JButton delete,find;
```

```
    JPanel panel1,panel2,panel3;
```

```
Container c;  
Connection conn;  
PreparedStatement pstmt;  
String url = "jdbc:odbc:msf";  
DeleteSupplierAccess application;  
Statement stmt;
```

```
public DeleteSupplierAccess()  
{  
    super("Delete supplier",true,true,true,true);  
    c = getContentPane();  
    panel1 = new JPanel();  
    panel2 = new JPanel();  
    panel3 = new JPanel();  
  
    panel3.setLayout(new FlowLayout(FlowLayout.CENTER));  
    panel2.setLayout(new GridLayout(4,2));  
    panel1.setLayout(new FlowLayout(FlowLayout.CENTER));  
  
    l1 = new JLabel("Supplier Code",JLabel.CENTER);  
    l2 = new JLabel("New Supplier Code",JLabel.CENTER);  
    l3 = new JLabel("New Supplier Name",JLabel.CENTER);  
    l4 = new JLabel("New Supplier City",JLabel.CENTER);  
    l5 = new JLabel("New Supplier Tel",JLabel.CENTER);  
  
    text1 = new JTextField("",10);  
    text2 = new JTextField("",10);  
    text3 = new JTextField("",15);  
    text4 = new JTextField("",6);  
    text5 = new JTextField("",5);
```

```
delete = new JButton("Delete");  
find = new JButton("Find");
```

```
panel3.add(l1);  
panel3.add(text1);  
panel3.add(find);
```

```
panel2.add(l2);  
panel2.add(text2);  
panel2.add(l3);  
panel2.add(text3);  
panel2.add(l4);  
panel2.add(text4);  
panel2.add(l5);  
panel2.add(text5);
```

```
panel1.add(delete);
```

```
c.add(panel3,BorderLayout.NORTH);  
c.add(panel2,BorderLayout.WEST);  
c.add(panel1,BorderLayout.SOUTH);
```

```
delete.addActionListener(this);
```



```
find.addActionListener(this);
```

```
setSize(300,250);
```

```
setVisible(true);
```

```
} // end of constructuer
```

```
public void actionPerformed( ActionEvent e)
```

```
{ String nam,supcty,supcnty,id;
```

```
String cd = text1.getText().trim();
```

```
if (e.getSource() == find )
```

```
{
```

```
try
```

```
{ Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

```
}
```

```
catch(Exception exp)
```

```
{
```

```
JOptionPane.showMessageDialog(application,exp.getMessage(),"WARNING",JOptionP  
ane.WARNING_MESSAGE);
```

```
}
```

```
try
```

```
{ conn = DriverManager.getConnection(url," "," ");
```

```
stmt = conn.createStatement();
```

```

String s = "SELECT * FROM supplier " + "WHERE supcode = " + cd +"";
ResultSet rset = stmt.executeQuery(s);

boolean f = rset.next();

if(f)
{

    text2.setText(rset.getString(1));
    text3.setText(rset.getString(2));
    text4.setText(rset.getString(3));
    text5.setText(rset.getString(4));

}

else
{
    JOptionPane.showMessageDialog(application,"Not Found....!\nTry
Again","MESSAGE" ,JOptionPane.INFORMATION_MESSAGE);
}

}

catch(SQLException ex )
{
    JOptionPane.showMessageDialog(application,"SQLERROR:" +
ex.getMessage() ,"Data Base ERROR" ,JOptionPane.ERROR_MESSAGE);
}

```

```

    }

}

if( e.getSource() == delete )
{

    try
    { Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

    }

    catch(Exception exp)
    {

OptionPane.showMessageDialog(application,exp.getMessage(),"WARNING",JOptionPane.WARNING_MESSAGE);

    }

    try
    {

        conn = DriverManager.getConnection(url," "," ");
        pstmt = conn.prepareStatement("Delete FROM supplier WHERE supcode = ?
");
        pstmt.setString(1,cd);

```

```

if(text2.getText().equals("") || text3.getText().equals("") || text4.getText().equals("") ||
text5.getText().equals("")) )
    { JOptionPane.showMessageDialog(application,"Fill all fields","Some Fields are
missing",JOptionPane.WARNING_MESSAGE);
    }
else
    {

        pstmt.executeUpdate();
        pstmt.close();
        conn.close();

        JOptionPane.showMessageDialog(application," Delete done perfectly
","INFORMATION",JOptionPane.INFORMATION_MESSAGE);
        text1.setText("");
        text2.setText("");
        text3.setText("");
        text4.setText("");
        text5.setText("");

    }
}

catch(SQLException ex )
{
    JOptionPane.showMessageDialog(application,"SQLERROR:" +
ex.getMessage() ,"Data Base ERROR" ,JOptionPane.ERROR_MESSAGE);

}

```



```
}// if cancel pressed
```

```
}// end of action performed
```

```
}
```

Sell

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import java.sql.*;
import java.util.Vector;
import java.util.*;
import java.text.*;
```

```
public class Sell extends JFrame implements ActionListener
```

```
{Bill b;
    Container c;
    JLabel l1,l2,l3;
    JTextField t1,t2,t3;
    JButton b1,b2,b3,b4;
    JPanel p1,p2;
    Statement stmt;
    PreparedStatement pstmt;
    Sell application;
    Connection conn;
    String url = "jdbc:odbc:msf";
```

```

Vector vect = new Vector(100,200);
Vector vect2 = new Vector(100,200);
Sell s;
int bin = 1;
Format formatter1,formatter2;
java.util.Date datef;
java.util.Date timef;
String date,time;
JDesktopPane desk;

public Sell()
{ super("Sell",true,true,true,true);
  c = getContentPane();

  datef = new java.util.Date();
  timef = new java.util.Date();

  formatter1 = new SimpleDateFormat("dd-MMM-yy");
  date = formatter1.format(datef);

  formatter2 = new SimpleDateFormat("hh:mm");
  time = formatter2.format(timef);

  b = new Bill(date,time,bin);

  p1 = new JPanel();
  p1.setLayout(new GridLayout(3,3) );
  p2 = new JPanel();
  p2.setLayout (new FlowLayout(FlowLayout.CENTER) );

  l1 = new JLabel("Pro_Code",JLabel.CENTER);

```

```
l2 = new JLabel("Count",JLabel.CENTER);  
l3 = new JLabel("Money_Paid",JLabel.CENTER);
```

```
t1 = new JTextField(" ",10);  
t2 = new JTextField(" ",10);  
t2.setText("1");  
t3 = new JTextField(" ",10);
```

```
b1 = new JButton("Add");  
b2 = new JButton("New Bill");  
b3 = new JButton("Sum");  
b4 = new JButton("Print");
```

```
p1.add(l1);  
p1.add(t1);  
p1.add(l2);  
p1.add(t2);  
p1.add(l3);  
p1.add(t3);
```

```
p2.add(b1);  
p2.add(b2);  
p2.add(b3);  
p2.add(b4);
```

```
c.add(p1,BorderLayout.NORTH);  
c.add(p2,BorderLayout.SOUTH);
```

```
setSize(400,300);  
setVisible(true);
```

```
b1.addActionListener(this);
b2.addActionListener(this);
b3.addActionListener(this);
b4.addActionListener(this);
```

```
}
```

```
public void actionPerformed(ActionEvent e)
{String cnt,mm,pcod,em,pcode;
double mn,mo;
String id = t1.getText().trim();
int n = Integer.parseInt(t2.getText().trim() );
int day,mon,yer,min,hor,pent,pcont;
```

```
if(e.getSource() == b1 )
{
```

```
try
```

```
{
```

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

```
}
```

```
catch(Exception ex)
```

```
{
```

```
JOptionPane.showMessageDialog(application,ex.getMessage(),"Loader class
not Found",JOptionPane.ERROR_MESSAGE);
```

```
}
```



```

try
{
    conn = DriverManager.getConnection(url," "," ");
    stmt = conn.createStatement();
    String s ="select pronom,proprice,protax from product where procode =
""+id+"" ";

    ResultSet r = stmt.executeQuery(s);
    boolean f=r.next();
    if(f)
    {
        mm = r.getString("pronom");
        mn = r.getDouble("proprice");
        mo = r.getDouble("protax");

        b.fill(mm,mo,mn,n);
        vect.add(id);
        vect2.add(t2.getText().trim());

        stmt.close();
        conn.close();

        t1.setText("");
        t2.setText("1");
    }

}
catch(SQLException ex )
{

```

```
JOptionPane.showMessageDialog(application,"SQLException: " +  
ex.getMessage() ,"Data Base ERROR" ,JOptionPane.ERROR_MESSAGE);
```

```
}
```

```
}
```

```
if(e.getSource() == b2 )
```

```
{ vect.clear();
```

```
 vect2.clear();
```

```
 b = new Bill(date,time,bin);
```

```
}
```

```
if(e.getSource() == b3 )
```

```
{
```

```
 b.sum();          /* dont forget to quit after sum is calculated */
```

```
}
```

```
if(e.getSource() == b4 )
```

```
{
```

```
 double g;
```

```
 g =Double.parseDouble( t3.getText().trim() );
```

```
 b.money_paid(g);
```

```
 int i = 0;
```

```
 int j = 0;
```

```
 while(i < vect.size() && j < vect2.size() )
```

```

{
    pcode =(String)(vect.get(i));
    em = String.valueOf(vect2.get(j));
    pcont = Integer.parseInt(em);

    try
    {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    }

    catch(Exception exp)
    {
        JOptionPane.showMessageDialog(null,exp.getMessage(),"Loader Class Not
Found",JOptionPane.ERROR_MESSAGE);
    }

    try
    {
        conn = DriverManager.getConnection(url,"","");
        pstmt = conn.prepareStatement("insert into invoice values(?,?,?,?)");

        pstmt.setInt(1,bin);
        pstmt.setString(2,date);
        pstmt.setString(3,time);
        pstmt.setString(4,pcode);
        pstmt.setInt(5,pcont);

        pstmt.executeUpdate();

        pstmt.close();
    }
}

```

```
conn.close();
```

```
}
```

```
catch(SQLException ex)
```

```
{
```

```
    JOptionPane.showMessageDialog(null,ex.getMessage(),"DATABASE  
ERROR",JOptionPane.ERROR_MESSAGE);
```

```
}
```

```
i = i + 1;
```

```
j = j + 1;
```

```
}
```

```
i = 0;
```

```
j = 0;
```

```
while(i < vect.size() && j < vect2.size() )
```

```
{
```

```
    pcod =(String)(vect.get(i));
```

```
    em = String.valueOf(vect2.get(j));
```

```
    pcnt = Integer.parseInt(em);
```

```
for(int k = 0;k < pcnt ;k++ )
```

```
{
```

```
    try
```



```

    {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    }

    catch(Exception exp)
    {
        JOptionPane.showMessageDialog(application,exp.getMessage(),"Loader Class
Not Found",JOptionPane.ERROR_MESSAGE);
    }

    try
    {
        conn = DriverManager.getConnection(url,"","");
        stmt = conn.createStatement();
        String s = "UPDATE product SET procount = procount - 1 WHERE procode =
"+"pcod+" " ";
        stmt.executeUpdate(s);

        stmt.close();
        conn.close();
    }

    catch(SQLException ex)
    {
        JOptionPane.showMessageDialog(application,ex.getMessage(),"DATABASE
ERROR",JOptionPane.ERROR_MESSAGE);
    }

}

i = i + 1;
j = j + 1;

```

```

    }
    vect.clear();
    vect2.clear();
    t3.setText("");
    bin++;
}

} // end action performed

```

```

}

```

Bill

```

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.sql.*;
import java.util.*;

```

```

public class Bill extends JFrame

```

```

{
    JTextArea a = new JTextArea(30,50);
    Container c;
    double tottax;
    double totsum;
    Bill application;
    Statement stmt;
    Connection conn;
    String url = "jdbc:odbc:msf";

```

```

public Bill(String date,String time,int bb)
{
    super("Bill");
    c = getContentPane();
    tottax = 0;
    totdsum = 0;

    a.setText("\tMSF POST Program\n\n\tBarners Bros\n\n\tTEL:0324578\n\n");
    a.append("DATE:"+date +"\n" );
    a.append("TIME:" +time +"\n");
    a.append("Bill NO :"+bb+"\n\n");
    a.setEditable(false);
    c.add(a);

    setSize(310,500);
    setVisible(true);

}

public void fill(String nm, double tx, double pc,int n)
{
    double mc ;
    mc = pc * tx + pc ;
    if(n == 1)
        a.append(nm+"\t"+tx+"\t\t" + mc + "\n");
    if(n > 1)
        a.append(n + " x" + nm+"\t"+n * tx+"\t\t"+n * mc+"\n");
    tottax+=tx;
    totdsum+=mc;
}

public void sum()
{
    a.append("-----\n");
    a.append("TOT TAX "+" \t\t\t" +tottax + "\n");
}

```

```
a.append("TOT PRICE" + "\t\t" + totsum + "\n");
```

```
}
```

```
public void money_paid(double t)
```

```
{
```

```
    a.append("-----\n");
```

```
    if ( t == totsum )
```

```
        a.append("Money Paid" + "\t\t" + t + "\n");
```

```
    if (t > totsum )
```

```
    {
```

```
        a.append("Money Paid" + "\t\t" + t + "\n");
```

```
        a.append("Money Remained" + "\t\t" + (t - totsum) + "\n\n");
```

```
    }
```

```
    a.append("\t" + "THANK YOU");
```

```
}
```

```
}
```

Show sales

```
import javax.swing.*;
```

```
import java.awt.event.*;
```

```
import java.sql.*;
```

```
import java.awt.*;
```

```
import java.util.*;
```

```
import javax.swing.table.*;
```

```
import java.text.*;
```



```
public class Reports extends JFrame implements ActionListener
```

```
{Reports mp;
```

```
DefaultTableModel model;
```

```
Container c;
```

```
JTable table;
```

```
JScrollPane scrol;
```

```
JPanel panel1,panel2,panel3,panel4,panel5,panel6;
```

```
JLabel label1,label2,label3,label4;
```

```
TextField text1,text2,text3;
```

```
Button button;
```

```
Statement stmt;
```

```
Connection conn;
```

```
String url = "jdbc:odbc:msf";
```

```
Vector vector;
```

```
public Reports()
```

```
{ super("Reports",true,true,true,true);
```

```
c = getContentPane();
```

```
model = new DefaultTableModel();
```

```
JTable table = new JTable(model);
```

```
model.addColumn("Invoice_Number");
```

```
model.addColumn("Date");
```

```
model.addColumn("Time");
```

```
model.addColumn("Product_Code");
```

```
model.addColumn("Product_Count");
```

```
scrol = new JScrollPane(table);
```

```
panel1 = new JPanel();

label1 = new JLabel("REPORTS");

panel1.add(label1);

panel3 = new JPanel();
panel3.add(scrol);

button = new JButton("FIND");
panel4 = new JPanel();
panel4.add(button);

panel2 = new JPanel();
panel2.setLayout(new GridLayout(3,2) );

label2 = new JLabel("TOT Sales:",JLabel.RIGHT);
label3 = new JLabel("TOT Invoices:",JLabel.RIGHT);
label4 = new JLabel("Sales Count:",JLabel.RIGHT);

text1 = new JTextField(10);
text2 = new JTextField(10);
text3 = new JTextField(10);

panel2.add(label2);
panel2.add(text1);
panel2.add(label3);
panel2.add(text2);
panel2.add(label4);
panel2.add(text3);
```

```
panel5 = new JPanel();
panel5.setLayout(new BorderLayout() );
panel5.add(panel4,BorderLayout.NORTH );
panel5.add(panel2,BorderLayout.CENTER );
```

```
c.add(panel1,BorderLayout.NORTH);
c.add(panel3,BorderLayout.CENTER);
c.add(panel5,BorderLayout.SOUTH);
```

```
button.addActionListener(this);
```

```
setSize(600,600);
setVisible(true);
```

```
}
```

```
public void actionPerformed(ActionEvent e)
```

```
{ int num;
  java.sql.Date t1;
  java.sql.Time t2;
  String dt1,dt2,tm1,tm2,s;
  int n;
```

```
if(e.getSource()== button)
```

```
{
  try
  {
```

```

        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    }
    catch(Exception ex )
    {

JOptionPane.showMessageDialog(mp,ex.getMessage(),"ERROR",JOptionPane.ERROR_
MESSAGE);
    }
    try
    {
        conn = DriverManager.getConnection(url,"","");
        stmt =
conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UP
DATEABLE);
        String m = "select * from invoice ORDER BY invdate,invtime";

        ResultSet r = stmt.executeQuery(m);

while(r.next() )
    {
        vector = new Vector(5);
        num = r.getInt("invnum");
        t1 = r.getDate("invdate");
        t2 = r.getTime("invtime");
        s = r.getString("procode");
        n = r.getInt("procount");

        vector.add(new Integer(num));

        vector.add(t1);

```



```

vector.add(t2);

vector.add(s);

vector.add(new Integer(n));

model.addRow(vector);

}

stmt.close();
conn.close();

}
catch(SQLException exp)
{
    JOptionPane.showMessageDialog(mp,exp.getMessage(),"SQL
ERROR",JOptionPane.ERROR_MESSAGE);

}

try
{
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
}

catch(Exception ex )
{

```

```
JOptionPane.showMessageDialog(mp,ex.getMessage(),"ERROR",JOptionPane.ERROR_
MESSAGE);
```

```
}
```

```
try
```

```
{
```

```
    conn = DriverManager.getConnection(url,"","");
```

```
    stmt = conn.createStatement();
```

```
    String d = "SELECT sum((invoice.procount)*((product.proprice *
product.protax)+product.proprice)) FROM invoice,product WHERE invoice.procode =
product.procode";
```

```
    ResultSet r = stmt.executeQuery(d);
```

```
    if(r.next())
```

```
        text1.setText(String.valueOf(r.getDouble(1)));
```

```
    stmt.close();
```

```
    conn.close();
```

```
}
```

```
catch(SQLException exp)
```

```
{
```

```
    JOptionPane.showMessageDialog(mp,exp.getMessage(),"SQL
ERROR",JOptionPane.ERROR_MESSAGE);
```

```
}
```

```
try
```

```
{
```

```
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

```
}
```

```
catch(Exception ex )
```

```

    {

JOptionPane.showMessageDialog(mp,ex.getMessage(),"ERROR",JOptionPane.ERROR_
MESSAGE);

    }
    try
    {
        conn = DriverManager.getConnection(url,"","");
        stmt = conn.createStatement();
        String p = "SELECT sum(invoice.procount) FROM invoice, product WHERE
invoice.procode = product.procode";
        ResultSet r = stmt.executeQuery(p);
        if(r.next())
            text2.setText(String.valueOf(r.getDouble(1)));
        stmt.close();
        conn.close();
    }

    catch(SQLException exp)
    {
        JOptionPane.showMessageDialog(mp,exp.getMessage(),"SQL
ERROR",JOptionPane.ERROR_MESSAGE);

    }
    try
    {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    }

    catch(Exception ex )

```

```

    {

OptionPane.showMessageDialog(mp,ex.getMessage(),"ERROR",OptionPane.ERROR_
MESSAGE);

    }

try
    {
        conn = DriverManager.getConnection(url,"","");
        stmt = conn.createStatement();
        String g = "SELECT DISTINCT count(invnum) FROM invoice";
        ResultSet r = stmt.executeQuery(g);
        if(r.next())
            text3.setText(String.valueOf(r.getDouble(1)));
        stmt.close();
        conn.close();

    }

catch(SQLException exp)
    {
        JOptionPane.showMessageDialog(mp,exp.getMessage(),"SQL
ERROR",OptionPane.ERROR_MESSAGE);

    }

}

```