MAR GREGORIOS COLLEGE OF ARTS & SCIENCE

Block No.8, College Road, Mogappair West, Chennai – 37

Affiliated to the University of Madras
Approved by the Government of Tamil Nadu
An ISO 9001:2015 Certified Institution

DEPARTMENT OF COMMERCE COMPUTER APPLICATION

SUBJECT NAME: VISUAL BASIC & RELATIONAL DATABASE MANAGEMENT SYSTEM

SUBJECT CODE: CPC51

SEMESTER: V

PREPARED BY: PROF. P. UMAESWARI

SYLLABUS

UNIT – I

Form – Form property- variables- data types- string –numbers- writing simple programs-toolbox-creating controls-name property- command button-access keys-image controls-text boxes-labels-check box – Frame – Message boxes.

UNIT – II

Displaying information – Determinate loops – Indeterminate loops- Conditionals – built- in suctions (String, Numeric) – functions and procedures. – Lists – arrays –control arrays- combo boxes- projects with multiple forms- Menus- MDI forms.

UNIT – III

Database Management System – Advantages – Components – class diagram – Events-Normalization – 1 NF – 2 NF – 3 NF.
`

UNIT IV

Oracle – an introduction – SQL *plus Environment – SQL – Logging into SQL * plus- SQL *plus commands – Errors – Oracle Tables: DDL – Naming rules and conventions – Data types-Constraints – Creating oracle Table- Displaying table information- Altering an existing table – Dropping, Renaming, truncating table.

UNIT – V

DML – Insert and select commands – Data access techniques: ADO – Connection object – Recordset object. Forms and Reports : Design of Form and report – Form layout – data reports.

UNIT - I

HISTORY OF VISUAL BASIC

Alan Cooper is considered the father of Visual Basic. Alan Cooper created the ***drag-and-drop design*** for the user interface of Visual Basic.

The History of Visual Basic dates back to 1991 when VB 1.0 was introduced. The core of Visual Basic was built on the older BASIC language. Visual Basic 1.0 for Windows was released in May 1991 at a trade show in Atlanta, Georgia.

Visual Basic 2.0 was released in November 1992. The programming environment was easier to use, and its speed was improved. Notably, forms became core objects, thus laying the foundational concepts of class modules.

Visual Basic 3.0 was released in 1993 and came in Standard and Professional versions. VB3 included version 1.1 of the Microsoft Jet Database Engine that could read and write Jet (or Access) 1.x databases.

Visual Basic 4.0 was released in August 1995. It was the first version that could create 32-bit as well as 16-bit Windows programs. It also introduced the ability to write non-GUI classes in Visual Basic. While previous versions of Visual Basic had used VBX controls, Visual Basic now used OLE controls (with files names ending in .ocx). These were later to be named ActiveX controls.

Visual Basic 5.0 was released in February 1997. Visual Basic 5.0 also introduced the ability to create custom user controls, as well as the ability to compile to native Windows executable code, speeding up calculation-intensive code execution. A free, downloadable Control Creation Edition was also released for creation of ActiveX controls.

Visual Basic 6.0 released in mid 1998 improved in a number of areas including the ability to create web-based applications. VB6 has been the most successful version in the history of Visual Basic. Visual Basic can be used in a number of different areas, for example, Education, Engineering, Research, Medicine, Business, Commerce, Marketing and Sales, Accounting, Consulting, Law, and Science.

The evolution of Visual Basic can be summarized by the following table.

VERSION	YEAR
Version 1 (for Windows)	March 20, 1991
Version 1 (for MS-DOS)	September 1992
Version 2	November 1992
Version 3, VBA (VB for Applications)	June 1993

Version 4 (16- and 32- bit Support)	October 1996
Version 5 (No 16-bit support)	April 1997
Version 6 (Part of Visual Studio)	October 1998
Version 7 (.Net)	February 2002
Version 7.1 (.Net 2003)	April 2003
VBA.Net for Office 2003	October 2003
Version 8.0 (.Net 2.0, Visual Studio 2005)	November 2005
Version 9 (.Net 3.5, Visual Studio 2008)	November 2007
Version 10 (Visual Studio 2010 and .Net Framework 4.0)	October 2008

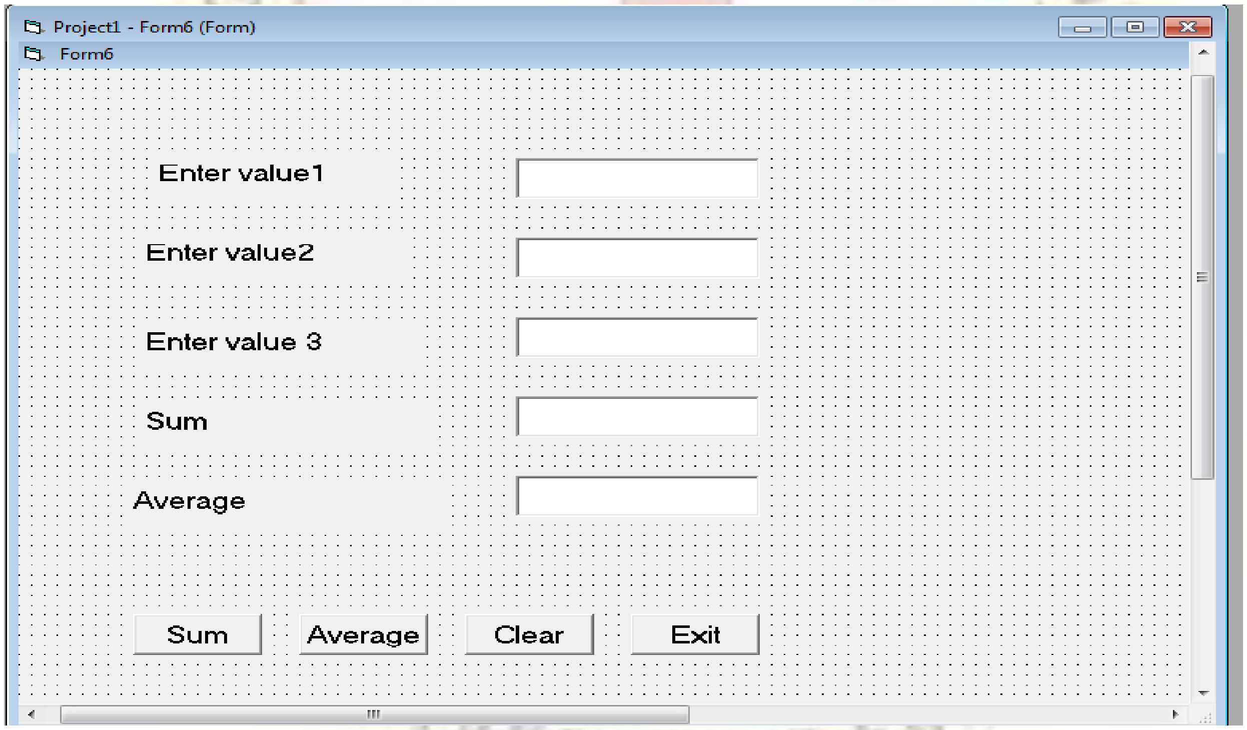
FEATURES OF VISUAL BASIC

- Visual Basic is a **third-generation high level programming language** which evolved from the earlier DOS version called BASIC.
- In V B, program is done in a **graphical environment**.
- Visual Basic is an **event-driven programming** language. VB programs are built around events. All the activities in a VB program are triggered by one event or another.
- It is **object-oriented**. It revolves around ready-made objects.
- It provides a **Graphical User Interface (GUI)** with ***icons***, and ***buttons***, that help users to simply '**Point and Click**'.
- It provides ***prebuilt objects*** called **controls** which can be placed on the screen using simple **'drag and drop'** facility.
- It enables **modular program development**.
- It provides a complete **set of tools** to the users for **Rapid Application Development** (RAD).
- It is the fastest and easiest way to develop **Windows based application**.
- With Visual Basic, building **prototypes** become easy.
- It provides several **functions**, **keywords**, and **statements** which directly relate to GUI.
- It allows users to create their own **ActiveX controls**.
- It eliminates the need for writing numerous lines of code to create controls.

INTRODUCTION TO FORM

- Form is a ***window***, initially blank, on which other controls can be placed.
- It is a ***container of other controls***.
- Forms are the ***basic building blocks*** of a VB application.
- It is the window or the screen with which users ***interact*** when they run an application.
- It is saved with **.frm** extension.
- Form has **properties**, **events**, and **methods** which are used to control its ***appearance*** and ***behavior***.
- The form can be ***resized*** and ***moved around*** the screen.

Example



Source Code

```
 Private Sub Command1_Click()
     Text4.Text = Val(Text1.Text) + Val(Text2.Text) + Val(Text3.Text)
End Sub
 Private Sub Command2_Click()
    Text5.Text = Val(Text4.Text) / 3
```

```
End Sub
Private Sub Command3_Click()
       Text1.Text = ""

       Text2.Text = ""

       Text3.Text = ""

       Text4.Text = ""

       Text5.Text = ""
End Sub
Private Sub Command4_Click()
End
End Sub
```

FORM PROPERTIES

- A form in Visual Basic has as many as 50 properties.
- Each of these properties will affect the ***appearance*** and ***behavior*** of a form.
- Form properties can be set at ***design time*** in the ***Properties Window*** or at ***run time*** by
  Writing Code.
- Design time is when the user is building the interface, setting properties, and writing code. Runtime is when the application is being executed.
- The properties window can be accessed using the shortcut key **F4** or by right clicking mouse button and choosing **properties option**.
- To access a property of a control, press **Ctrl+Shift+First letter** of the property.

PROPERTY	DESCRIPTION
Name	It is the name by which the form is ***referred in the code***.
Caption	It is the ***meaningful name*** that appears on the ***Title Bar*** of a form.
BackColor	It determines the ***background color*** of a form. It can be set by using the ***color palette*** which appears beside the property.

Picture	It specifies the ***name of the file that contains the picture*** to be displayed on the form. This property can be set at design time or at runtime.
Font Properties	It includes the Font, FontName, FontSize, FontBold, FontItalic, FontStrikethru, and FontUnderline.
Visible	It is a Boolean property. If the property is set to true, the form is visible. Otherwise, the form is not visible.
Window State	The value of this property specifies the ***window state of the form at runtime***. The values include 0-normal, 1-minimized, and 2-maximized.
Position Properties	Properties like ***Left, Top, Height, and Width*** can be set at design time or at runtime to locate the form at a place of the user's choice.
MDI Child	It is a Boolean property. If its value is set to true, the form will behave as an MDI child.
Control Box	It is a Boolean property. If it is set to true, the ***control box (minimize, maximize, close buttons)*** is visible on the top left hand corner of the form.
Min Button and Max Button	These properties have Boolean values true or false. If the properties are set to false, the ***user cannot minimize or maximize the form at run time***. The default values are true.
Moveable	It is Boolean property. If its value is set to false, the form cannot be moved during runtime. The default value is true.
Border Style	This property determines the ***type of window*** that the user will see during runtime. This property determines ***whether a form is movable or can be resized***.

Example

```
Private Sub Form_Load() Form1.Cls
    Form1.WindowState = vbMaximized
    End Sub
```

FORM EVENTS

- A form comes to life when the user clicks to start a VB application.
- The lifecycle of a form has many events.

EVENT	DESCRIPTION
Load	During the load event, the ***form with all its properties and variables is loaded in memory***. The load event occurs whenever the '**show**' method is executed or a form property is referenced.
Initialize	During the initialize event, ***all variables*** associated with a form are ***initialized***.
Activate	The activate event occurs ***when a form gets user input***. This event also occurs when the '**show**' method or '**setFocus**' method of the form is called.
Deactivate	The deactivate event occurs ***when another form gets the focus***.
Paint	The paint event fires automatically ***when a form is refreshed*** (when areas of a form are uncovered or when a form is resized).
Unload	The unload event occurs ***when the user closes a form***. During this event, the ***form is unloaded from memory***.
QueryUnload	The QueryUnload event occurs before the Unload event is fired. The QueryUnload event allows the option to abort the Unload event.
Terminate	The terminate event is the ***final event*** in the lifecycle of a form. All ***memory held by the form variables are released*** during this event.

Example

```
 Private Sub Form_Paint()
      DrawWidth = 5
      Circle (Rnd * 3000, Rnd * 7000), Rnd * 800, vbYellow
      Line(1000,1000) – (5000,5000), RGB(0,255,0)
      Form1.Circle(500,500), 300, RGB(255,0,0)
End Sub
```

FORM METHODS

METHOD	DESCRIPTION
Cls	The Cls method ***clears the object***, generally the Form and Picturebox object.

Hide	The Hide method ***makes a Form invisible to the user***. It will not unload a form.
Show	The Show method displays a Form to the user. It internally sets the visible property to true.
Refresh	The Refresh method ***repaints or redraws an object completely***.
SetFocus	The SetFocus method ***moves focus to the specified Form***, the specified control on the active form, or the specified field on the active datasheet.
Move	The Move method allows a programmer to ***position a Form at a desired location***.
Line	The Line method is used to draw a line, rectangle, or box.
Circle	The Circle method is used to draw a circle, ellipse, or an arc.
Pset	The Pset method is used to ***draw a point with a given color at a given location***.
Point	The point method ***returns the color of the screen at a given location***.

Example

```
Private Sub UserForm_Click()
UserForm1.PrintForm
End Sub
```

VARIABLES

- **A variable** is an ***identifier*** that ***points to a memory location*** in which data used during computation is stored.
- A variable can be used to hold ***different types of data*** at different times during program execution.
- **Variable declaration** is the process which is used to inform a program that a particular word is a variable.
- A variable has to have a **name** and **data type**.
- Variables can be declared in the **declaration section** of a form module, standard module, or class module or within a procedure.

- Variables in VB can be declared using **Dim, Static, Private,** or **public** keywords.
- If a variable is not declared before use, VB automatically creates a variable with that name. This is called **implicit declaration**.
- The keyword **option explicit** in the declaration section of a form module, standard module, or class module makes variable declaration compulsory. Go to **Tools -> Options -> Editor** and check the **Require Variable Declaration** option. This will automatically insert the **Option Explicit** statement in any new module.

Syntax

variablename [As Type]

Example

Dim a As Integer

Dim flag As Boolean

RULES FOR NAMING VARIABLES

- Must begin with an alphabet.
- Must not have an embedded period or a special character.
- Must not exceed 255 characters.
- Must be unique within the same scope.
  - A ***module-level variable*** **cannot** have the same name as any ***procedures*** or ***types*** defined in the module.
  - A ***local variable*** can have the same name as ***public procedures, types, or variables defined in other modules***.
  - It is safer to have names of variables different from procedure names or controls in other modules.

Example

```
Dim a As Double
Dim b As Double
Dim res As Double
        Private Sub Command1_Click()
```

```
   a = Val(Text1.Text)
    b = Val(Text2.Text)
    res = a + b
         Text3.Text = res
 End Sub
```

SCOPE OF A VARIABLE

- The ***place of creation*** and ***use*** of a variable is called its **scope**.
- In VB, the scope of a variable can be a procedure, a form module, a standard module, a class module and so on.
- **Lifetime** of a variable is defined as the ***period between creating a variable and destroying it***.

TYPES

- Procedure-level Variables – Dim, Static keywords
- Module-level Variables – Private, Public keywords

PROCEDURE-LEVEL VARIABLE

- A procedure-level local variable is ***declared inside a procedure*** using **Dim** keyword.
- A procedure-level variable can be ***accessed only inside the procedure*** where it is declared.
- The value of a procedure-level variable is **local** to that procedure.
- The value of a variable in one procedure cannot be accessed from another procedure.
- A Procedure-level variable exists only as long as the procedure is executing.
- When the procedure finishes, the value of a procedure-level variable disappears.

Example

```
Dim cnt As Integer
Dim flag As  Boolean
```

- Procedure-level static variable is declared with **static** keyword. It is ***local*** to the procedure in which it is declared but it exists the ***entire time the application is running***.
- A static variable will ***preserve its value*** even when its procedure ends.
- The value of a static variable will ***remain in memory*** even if its procedure terminates.

Example

```
Static counter As Integer
```

Example - 1

```
 Private Sub Command1_Click()
    Dim cnt As Integer
     cnt = cnt + 1
     MsgBox "cnt = " & cnt
 End Sub
```

Output

```
cnt = 1

cnt = 1

cnt = 1
```

Example - 2

```
 Private Sub Command2_Click()
    Static count As Integer
    count = count + 1
     MsgBox "count = " & count
  End Sub
```

Output

```
count = 1

count = 2

count = 3
```

MODULE-LEVEL VARIABLE

- A module-level private variable is declared in the **declaration section of a module** using the **private keyword.**
- A module-level private variable is available to ***all procedures within a module***.
- A module-level private variable will not be available to procedures in other modules.

Example

Private temp As Integer

- A module-level public variable is declared in the **declaration section of a module** using the **public keyword.**
- A module-level public variable is available ***to all modules in an application***.
- Public variables cannot be declared inside a procedure.
- Local variables will have preference over public variables. This is called **shadowing**.

Example

Public val As Integer

Scope	Private	Public
Procedure-level	Variables are private to the ***procedure*** in which they are declared.	Not applicable. Procedure-level variables cannot be public.
Module-level	Variables are private to the ***module*** in which they are declared.	Variables are available to ***all modules*** in an application.

Example

Public x As Integer
x = 15
Dim x As Integer
x = 10

Accessing the value of ‘x’ **inside** the procedure will give the value **10**. Accessing the value of ‘x’ **outside** the procedure will give the value **15**.

To access the module-level public variable inside the procedure use the following

Debug.print Module1.x

The above statement will give the value **15.**

DATA TYPES

- A **variable** should have a **name** and **data type**.
- The data type of a variable ***what type of values*** can be stored in the variable and ***how*** these variables are ***stored*** in the computer’s memory.
- **Arrays** also have data types.
- The ***arguments*** and ***return type*** of **functions** also have data types.

S.No.	DATA TYPE	SIZE	PURPOSE
1	Integer	2 bytes	It is used to hold **whole numbers**. Its range is **-2^{15} to -2^{15}-1**.

2	Long	4 bytes	It is used to hold **long integers**. It is much slower than integer.
3	Single	4 bytes	It is used to hold **number with decimal point**.
4	Double	8 bytes	It is used to hold **number with decimal point**. It has a larger range than a single data type.
5	String	1. Variable length - 0 to 2 billion 10 bytes + string length 2. Fixed length – 1 to 65,400	It is used to store **text or string** values. It is the ***most commonly used*** data type. By **default**, a string variable or argument is a **variable-length string.** A variable-length string can grow or shrink as data is assigned to it. Fixed length string variables will be **padded** with enough trailing spaces or will be ***truncated*** if assigned string is long.
6	Byte	1 byte	It is used to hold **one byte of data**. It can store values from 0 to 255. It cannot hold negative numbers.
7	Boolean	2 Bytes	It is used for holding **True** or **False** values. Zero represents false and non-zero represents true.
8	Currency	8 bytes	It is used to store **monetary values**.
9	Date	8 bytes	It is used to hold **date and time data**. It can store dates from January 1, 100 and December 31, 9999.
11	Object	4 bytes	It is used to hold **references to objects** within an application or other applications. **Example:** Form, Controls, Procedure, record set
12	User-defined		Number required by elements. The range of each element is the same as the range of its data type.

EXAMPLE

```
Dim    x        As    Integer
Dim    db       As    Database              'object data type
Dim    rs       As    Recordset             ' object data type
Dim    ItemName as    String                ' a variable-length string
Dim    ItemCode as    String * 10           ' a fixed-length string
```

- ItemCode will be padded with enough trailing spaces to make it to 10 characters.
- If the value assigned to ItemCode has more than 10 characters, VB truncates the characters.
- If both of the variants contain numbers, the + operator performs addition. If both the variants contain strings, then the + operator performs string concatenation.
- If one of the variants contains a number and the other a string, you have a problem. VB first attempts to convert the string into a number. If the conversion is successful, the + operator adds the two values; if unsuccessful, it generates a Type mismatch error.
- It is better to use '**&**' operator for string concatenation rather than '+' operator.

STRING FUNCTIONS

STRING FUNCTION	**DESCRIPTION**	**SYNTAX**
Trim	Removes leading and trailing spaces in a string.	**Trim**(*string*)
Ltrim	Removes leading spaces in a string.	**LTrim**(*string*)
Rtrim	Removes trailing spaces in a string.	**RTrim**(*string*)
Len	Returns an integer value which is the length of a string including empty spaces.	**Len**(*string*)
LCase	Converts all the characters of a string to small letters	**LCase**(*string*)
UCase	Converts all the characters of a string to capital letters	**UCase**(*string*)
Left	Extracts a specified number of characters from the beginning of a string.	**Left**(*string*, *length*)

Right	Extracts a specified number of characters from the end of a string.	**Right**(*string*, *length*)
Mid	Extracts a substring from the original string.	**Mid**(*string*, *start*[, *length*])
StrComp	Compares string1 and string2 and returns a value that represents the result of the comparison. It returns -1 if string1 < string2. It returns 0 if both strings are equal. It returns 1 if string1 > string 2.	**StrComp**(*string1*, *string2*[, *compare*])
StrReverse	Reverses a string.	**StrReverse**(*string1*)
InStr	Returns the position of the first occurrence of one string within another.	**InStr**([*start*,]*string1*, *string2*[, *compare*])
InStrRev	Returns the position of the last occurrence of one string within another string.	**InstrRev**(*string1*, *string2*[, *start*[, *compare*]])
Replace	Finds a string in an expression and replaces with another string.	**Replace**(*expression*, *find*, *replacewith* [, *start* [, *count*[, *compare*]]])
Str()	Returns the string equivalent of a number.	**Str**(number)
Val( )	Converts a string to a number.	**Val**(string)
String()	Returns "character" n times.	**String(n, "Character")**
Split	Used to breakup a string at specified places.	**Split**(*expression*[, *delimiter*[, *count*[, *compare*]]])

NUMBERS (MATH FUNCTIONS)

MATH FUNCTION	**DESCRIPTION**	**SYNTAX**
Abs	Returns the absolute (unsigned) value of the argument.	**Abs**(*num*)
Sqr	Returns the square root value of the argument	**Sqr**(*num*)
Rnd	Returns a random value between 0 and 1	**Rnd**[(*num*)]
Randomize	It uses number to initialize the Rnd function's random-number generator, giving it a new seed value.	**Randomize**

Sgn	Returns sign of a number	**Sgn(**Num**)**
Round	Rounds a number n to m decimal places	**Round** (n, m)
Log	Returns the natural logarithm of the argument	**Log**(*num*)
Exp	Exp of a number x is the value of e^x	**Exp**()
Sin	Returns the sine value of the argument in *radians*	**Sin**(*num*)

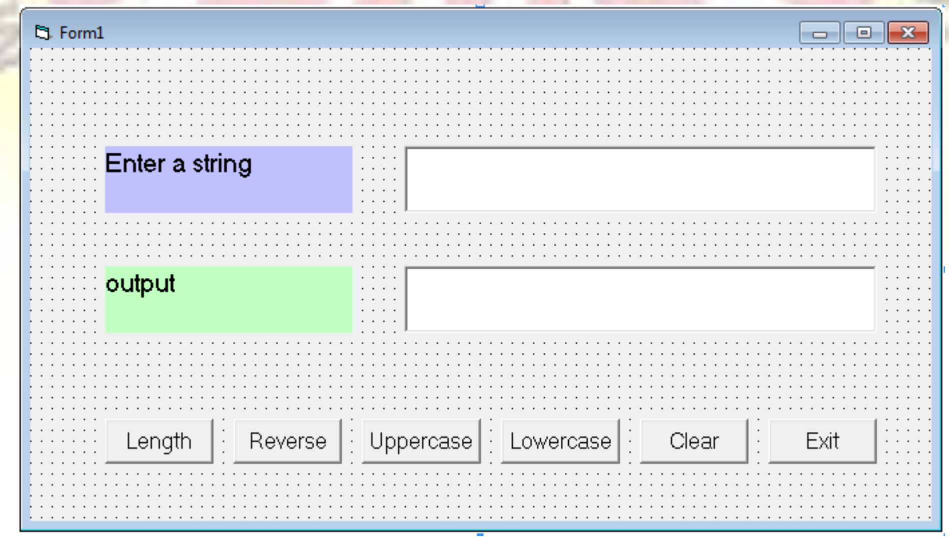
WRITING SIMPLE PROGRAMS

How to Concat two string in Visual basic?

```
Sub Main()
Dim str1, str2, str3 As String
str1 = "Visual"
str2 = "Basic"
str3 = "Program"
Console.WriteLine(str1 + " " + str2 + " " + str3)
Console.ReadLine()
End Sub
End Module
```

Output: Visual basic program

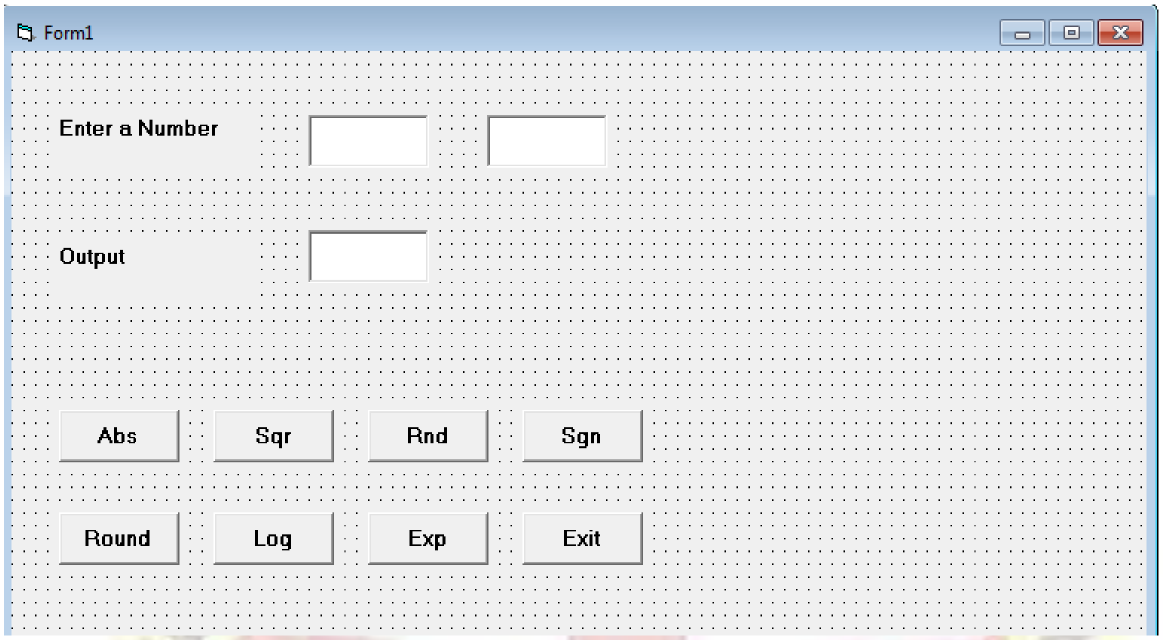
Create a VB application using string functions.



```
Private Sub Command1_Click()
Text2.Text = Len(Text1.Text)
End Sub
```

```
Private Sub Command2_Click()
        Text2.Text = StrReverse(Text1.Text)
        If Text1.Text = Text2.Text Then
                MsgBox "Given string is palindrome"
        Else
                 MsgBox "Given string is not palindrome"
         End If
End Sub
Private Sub Command3_Click()
Text2.Text = UCase(Text1.Text)
End Sub
  Private Sub Command4_Click()
  Text2.Text = LCase(Text1.Text)
  End Sub
Private Sub Command5_Click()
Text1.Text = ""
Text2.Text = ""
End Sub
Private Sub Command6_Click()
End
End Sub
```

Create a VB application using math functions.



```
Private Sub Form_Load() Text2.Visible
            = False
End Sub

Private Sub Command1_Click()
            Text2.Visible = False
            Text3.Text = Abs(Val(Text1.Text)) End Sub

Private Sub Command2_Click()
            Text2.Visible = False
            Text3.Text = Sqr(Val(Text1.Text)) End Sub

Private Sub Command3_Click()
           Text2.Visible = False
           Text3.Text = Rnd(Val(Text1.Text)) End Sub

Private Sub Command4_Click()
           Text2.Visible = False
           Text3.Text = Sgn(Val(Text1.Text)) End Sub
```

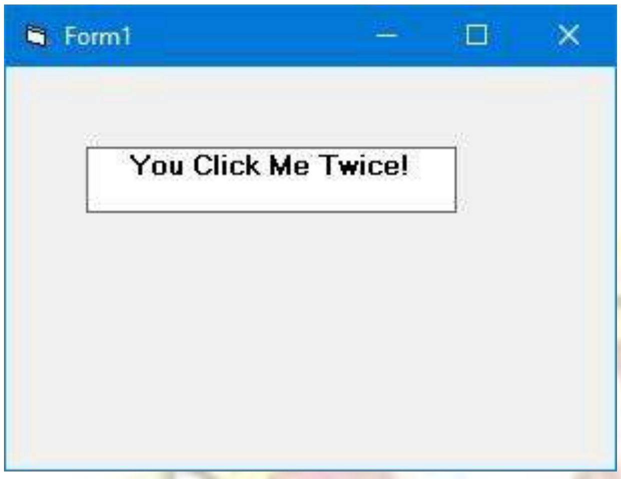
```
 Private Sub Command5_Click()
        Text2.Visible = True Text2.SetFocus
        Text3.Text = Round(Val(Text1.Text), Val(Text2.Text)) End Sub

Private Sub Command6_Click()
             Text2.Visible = False
             Text3.Text = Log(Val(Text1.Text))
End Sub


 Private Sub Command7_Click()
             Text2.Visible = False
             Text3.Text = Exp(Val(Text1.Text))
 End Sub


Private Sub Command8_Click()

End
End Sub
```

This program display a message whether the label is being click once or click twice.

In this program, insert a label and rename it as MyLabel and change its caption to "CLICK ME". Next, key in the following codes:

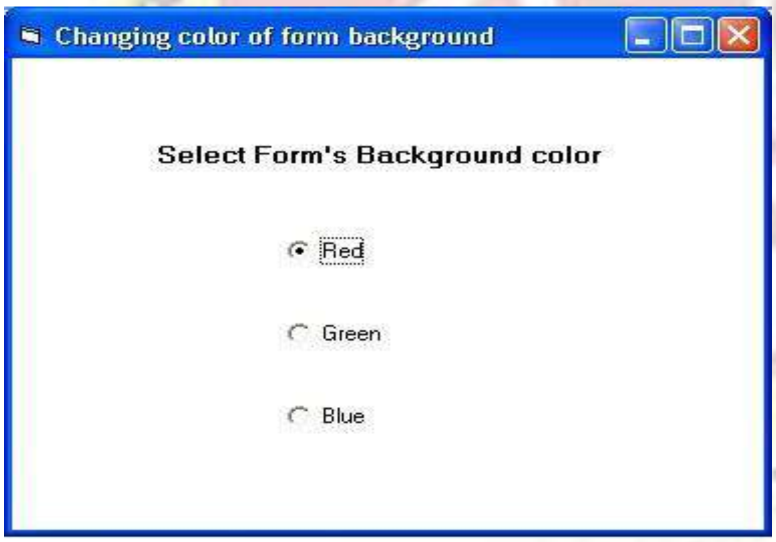
```
Private Sub MyLabel_Click()
MyLabel.Caption = "You Click Me Once"
End Sub


Private Sub MyLabel_DblClick()
MyLabel.Caption = "You Click Me Twice!"
End Sub
```

The Output



Running the program and click the label once, the "CLICK ME" caption will change to "You Click Me Once". If you click the label twice, the "CLICK ME" caption will change to "You Click Me Twice!".

Change the background color of the form.


Changing color of form background
Select Form's Background color
Red
Green
Blue

Private Sub Option1_Click(Index As Integer)

Select Case Index

Case 0

Form1. BackColor = vbRed

Case 1

Form1. BackColor = vbGreen

Case 2

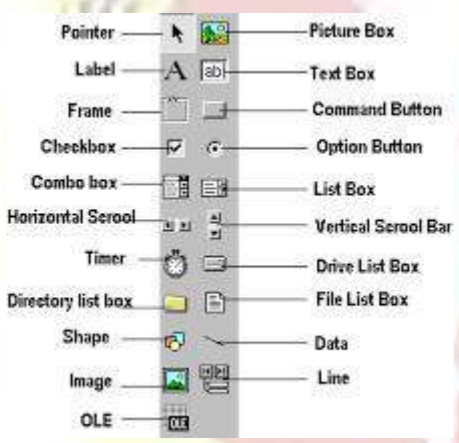
Form1. BackColor = vbBlue

End Select

End Sub

TOOLBOX

The tool box in Visual Basic contains many ready-made objects called controls. The toolbox contains 21 standard controls by default.



The important controls are discussed below

1. Label Box A

• Label control allows users to ***display text*** which cannot be changed.

2. Text Box / Edit Field / Edit Control ab|

• Text control is used to ***accept user input*** or to ***display text*** to the user.

• It allows users to ***edit data*** in a text control.

3. Command Button

• Command button is used to ***carry out a command***.

• When a user clicks a command button, the computer will perform the task associated with that button.

4. Option Button

• Option button allows users to ***display multiple choices***.

• It allows users to ***select only one option*** among the available **mutually exclusive** options.

5. Check Box

• Check box acts as ***toggle switches***, turning options on or off.

• Check box allows users to ***display multiple choices***.

• It allows users to ***select one or more options*** among the available options.

6. Frame

• Frame control allows users to ***group controls together***.

• To group controls, ***draw the Frame first***, and then ***draw the controls*** inside the frame.

7. Picture Box

• A picture box control is used to ***display a picture*** in bitmap, icon, metafile, JPEG or GIF formats.

• It **clips** the graphic if the control isn't large enough to display the entire image.

• It can be used as a **container**.

8. Image

• An image control is used to ***display an image.***

• It uses ***fewer resources*** than a picture box.

• It cannot be used as a container.

9. Timer

• Timer control is used to ***control actions*** that must take place ***at or after set intervals***.

• It is ***invisible*** while the program is running.

10. List Box

• The list box control is used to ***display a list of items***.

• It allows the user to ***choose any one item*** from the list.

• It can display only a set of items that are available.

• Items can only be added to the list during ***run-time***, not at design time.

11. Combo Box

• Combo box is a ***combination*** of list box and text box.

• It is a ***drop-down list box***.

• It is used to ***display a list of items*** and it allows the user to ***choose any one item*** from it.

• It allows the users to ***enter new items*** to the combo box when the program is running.

12. HScrollBar

• Horizontal Scroll Bar control helps users to ***navigate*** through a long list of items, to ***indicate the current position*** on a scale, or as an input device or indicator of ***speed or quantity***.

• The ***position of the slider*** determines the ***value*** returned by a scroll bar.

13. VScrollBar

• Vertical Scroll Bar control helps users to ***navigate*** through a long list of items, to ***indicate the current position*** on a scale, or as an input device or indicator of ***speed or quantity***.

• The ***position of the slider*** determines the ***value*** returned by a scroll bar.

14. Line and Shape Controls

• Line and Shape Controls are used to draw lines and various shapes such as squares, circles, etc.

15. Pointer Control

• Pointer control is used to ***select objects*** that are placed on a form.

16. Common Dialog Box

• The common dialog box control is used to perform ***open, save, print***, and similar operations on ***files***.

17. Drive, Directory, and File List Controls

• Drive, Directory, and File controls can be used to ***display available drives, directories and files***.

• They are used to provide ***file management facilities***.

• The user can select a valid drive on the system.

• The user can see a **hierarchical structure** of drives, directories and files in the system.

18. Data Control

• Data control is used to ***access data and manipulate data*** in a **database**.

19. DBGrid Control

• DBGrid control is used to ***create tables and spreadsheet***.

20. OLE (Object Linking and Embedding)

• The OLE control allows users to ***link*** a ***VB program to another object or program***.

• For example, from within a VB program, users can call ***Microsoft Excel***, or ***MS Paint***, complete the work and get back to the VB application

1.7.1 CREATING CONTROLS

Toolbox provides a set of tools that you use at design time to place controls on a form. In addition to the default toolbox layout, you can create your own custom layouts by selecting Add Tab from the context menu and adding controls to the resulting tab. We discuss the nine controls indicated on the Toolbox in the Figure.

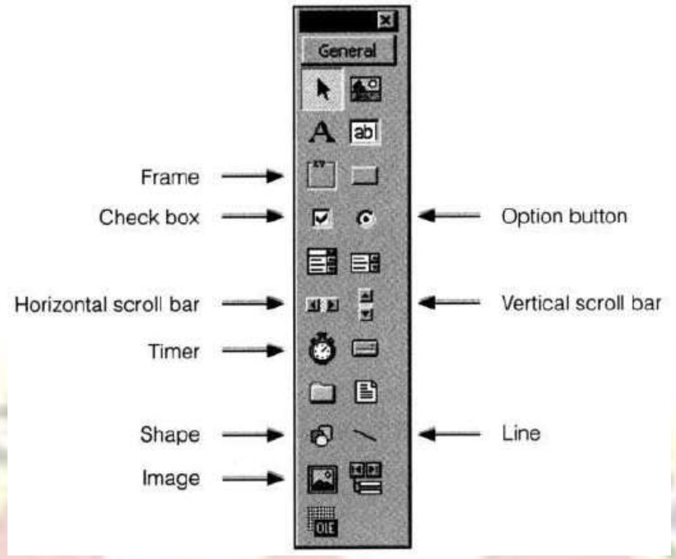
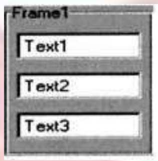


FIGURE 1 *Nine Elementary Controls.*

1.7.1.1 THE FRAME CONTROL



Frames are passive objects used to group related sets of controls for visual effect. You rarely write event procedures for frames. The preceding frame has a group of three text boxes attached to it. When you drag the frame, the attached controls follow as a unit. If you hide the frame, the attached controls will be hidden as well.

1.7.1.2 THE CHECK BOX CONTROL

A check box, which consists of a small square and a caption, presents the user with a yes/no choice. The Value property of a check box is 0 when the square is empty and is 1 when the square is checked. At run time, the user clicks on the square to toggle between the unchecked and checked states. So doing also trig-gers the Click event.

1.7.1.3 THE OPTION BUTTON CONTROL

Option buttons are used to give the user a single choice from several options. Normally, a group of several option buttons is attached to a frame or picture box with the single-click- draw technique. Each button consists of a small circle accompanied by text that is set with the Caption property. When a circle or its accompanying text is clicked, a solid dot appears in the circle and the button is said to be “on.” At most one option button in a group can be on at the same time. Therefore, if one button is on and another button in the group is clicked, the first button will turn off. By convention, the names of option buttons have the prefix opt.

1.7.1.4 THE HORIZONTAL AND VERTICAL SCROLL BAR CONTROLS

There are two types of scroll bars. When the user clicks on one of the arrow buttons, the thumb moves a small amount toward that arrow. When the user clicks between the thumb and one of the arrow buttons, the thumb moves a large amount toward that arrow. The user can also move the thumb by dragging it. The main properties of a scroll bar control are Min, Max, Value, SmallChange, and LargeChange, which are set to integers. At any time, hsbBar. Value is a number between hsbBar.Min and hsbBar. Max determined by the position of the thumb. If the thumb is halfway between the two arrows, then hsbBar. Value is a number halfway between hsbBar.Min and hsbBar.Max. If the thumb is near the left arrow button, then hsbBar. Value is an appropriately proportioned value near hsbBar.Min. When an arrow button is clicked, hsbBar.Value changes by hsbBar.SmallChange and the thumb moves accordingly. When the bar between the thumb and one of the arrows is clicked, hsbBar.Value changes by hsbBar.LargeChange and the thumb moves accordingly. When the thumb is dragged, hsbBar.Value changes accordingly. The default values of Min, Max, SmallChange, and LargeChange are 0, 32767, 1, and 1, respectively. However, these values are usually reset at design time.

1.7.1.5 THE TIMER CONTROL

The timer control, which is invisible during run time, triggers an event after a specified amount of time. The length of time, measured in milliseconds, is set with the Interval property to be any number from 0 to 65,535 (about 1 minute and 5 seconds). The event triggered each time Timer1.Interval milliseconds elapses is called Timer1_Timer( ). In order to begin timing, a timer must first be turned on by setting its Enabled property to True. A timer is turned off either by setting its Enabled property to False or by setting its Interval property to 0. The standard pre- fix for the name of a timer control is tmr.

1.7. THE SHAPE CONTROL

The shape control assumes one of six possible predefined shapes depending on the value of its Shape property. Figure 10-5 shows the six shapes and the values of their corresponding Shape properties. Shapes are usually placed on a form at design time for decoration or to highlight certain parts of the form. By convention, names of shape controls have the prefix shp.

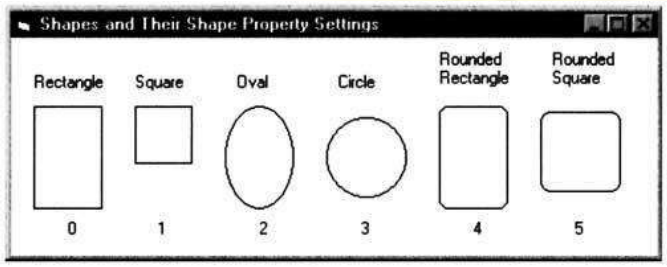


FIGURE 10-5 *The Six Possible Shapes for a Shape Control*

1.7.1.7 THE LINE CONTROL

The Line control, which produces lines of various thickness, styles, and colors, is primarily used to enhance the visual appearance of forms. The most useful properties of lines are BorderColor (color of the line), BorderWidth (thickness of the line), BorderStyle (solid, dashed, dotted, etc.), and Visible. Figure 10-8 shows several effects that can be achieved with lines. By con- vention, names of line controls have the prefix lin.

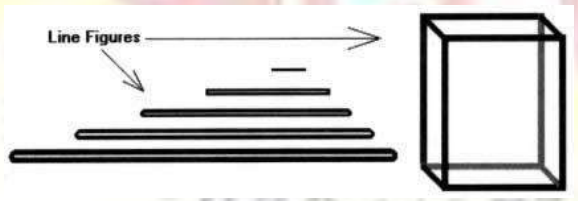


FIGURE 10-8 *Several Effects Achieved with Line Controls*

1.7.1.8 THE IMAGE CONTROL

The image control is designed to hold pictures stored in graphics files such as .BMP files cre- ated with Windows' Paint, .ICO files of icons that come with Visual Basic, or .GIF and JPEG images used on the World Wide Web. Pictures are placed in image controls with the Picture property. If you double-click on the Picture property during design time, a file-selection dia- log box appears and assists you in selecting an appropriate file. However, prior to setting the Picture property, you should set the Stretch property. If the Stretch property is set to False (the default value), the image control will be resized to fit the picture. If the Stretch property is set to True, the picture will be resized to fit the image control. Therefore, with Stretch property True, pictures can be reduced (by placing them into a small image control) or enlarged (by plac- ing them into an image control bigger than the picture). Figure 10-9 shows a picture created with Paint and reduced to several different sizes. By convention, names of image controls have the prefix img.

A picture can be assigned to an image control at run time. However, a statement such as

imgBox.Picture = "*filespec*"

will not do the job. Instead, we must use the LoadPicture function in a statement such as

imgBox.Picture = LoadPicture("*filespec*")

Image controls enhance the visual appeal of programs. Also, because image controls respond to the Click event and can receive the focus, they can serve as pictorial command buttons.

The Naming property

The Name property is a string used by clients to identify, find, or announce an object for the user. All objects support the Name property. For example, the text on a button control is its name, while the name for a list box or edit control is the static text that immediately precedes the control in the tabbing order.

Every control has it's properties one of the common properties of controls is name property. Name property is the one by which the control is identified and it is referred as the object name of that control.

COMMAND BUTTON

- The command button control is used to ***begin***, ***interrupt***, or ***end*** a process.
- When clicked, it ***invokes a command*** that has been written on its Click event Procedure.

- Most Visual Basic applications have command buttons that allow the user to simply ***click them to perform actions***.

COMMAND BUTTON PROPERTIES

PROPERTY	DESCRIPTION
Name	It is the ***name by which the Command Button is referred in the code***.
Caption	It is the ***meaningful name*** that appears on a command button.
Appearance	The Appearance property sets whether or not an object is painted at runtime with ***3-D effects***. 0 – Flat, 1 – 3-D
Style	The Style property determines ***if the command button will display text only or graphics also***. 0 - Standard, 1- Graphical
Visible	It is a Boolean property. If the property is set to true, the command button is visible. Otherwise, it is hidden.
Enabled	It is a Boolean property. By default, this property's value is True, which means that the control can respond to user-generated events.
Cancel	It is a Boolean property. It indicates whether the command button is the cancel button on a form. Pressing Esc key will activate this button. The default value is False.
BackColor	It determines the ***background color*** of a command button. It can be set by using the ***color palette.***
Font Properties	The font properties of a command button include the Font, FontName, FontSize, FontBold, FontItalic, FontStrikethru, and FontUnderline
Position Properties	Properties like ***Left, Top, Height, and Width*** can be set at design time and runtime to locate the form at a place of the user's choice.
Picture	The Picture property specifies the picture that will be displayed on the command button when it is down or selected.
DisabledPicture	The DisabledPicture property ***specifies the picture that will be displayed when the command button is disabled***.

COMMAND BUTTON EVENTS

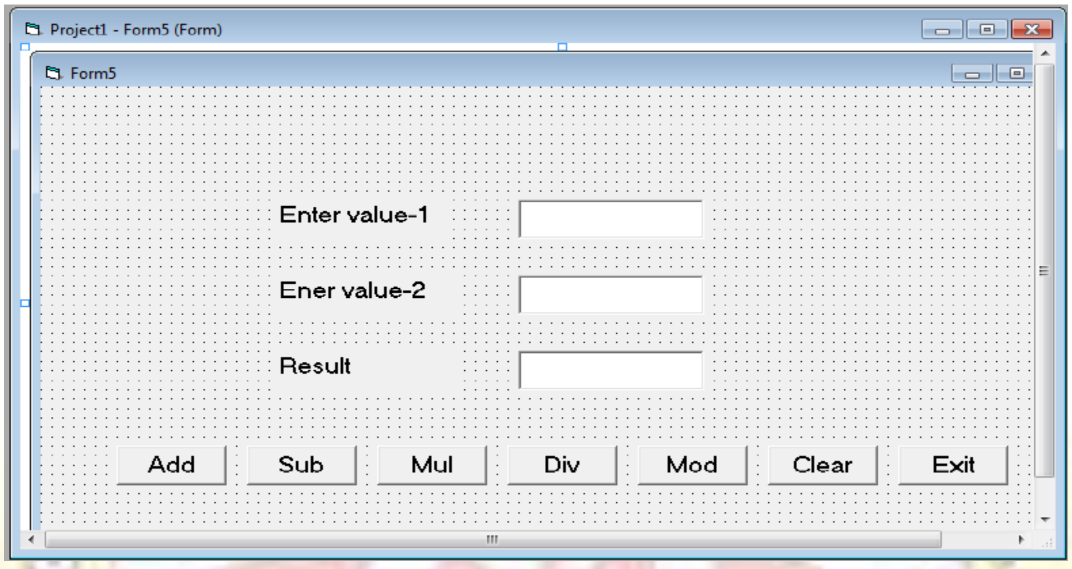
EVENT	DESCRIPTION
Click	The Click event occurs when a command button is clicked. The code written in the Click event procedure is invoked. Clicking a command button control also generates the MouseDown and MouseUp events.
DragDrop	The DragDrop event occurs when a user clicks on an object and drags it to a different location on the screen.
DragOver	The DragOver event occurs when the user drags an object over another control.
KeyDown	The **KeyDown** event occurs when the user presses a key while a form or control has the focus.
KeyPress	The **KeyPress** event occurs when the user presses and releases a key or key combination while a form or control has the focus.
KeyUp	The **KeyUp** event occurs when the user releases a key while a form or control has the focus.
MouseDown	The MouseDown event occurs when the user presses any mouse button.
MouseMove	The MouseMove event occurs when the user moves the mouse across the screen.
MouseUp	The MouseUp event occurs when the user releases any mouse button.
GotFocus	The **GotFocus** event occurs when the specified object receives the focus.
LostFocus	The **LostFocus e**vent occurs when the specified object loses the focus.

COMMAND BUTTON METHODS

METHOD	DESCRIPTION
Drag	The Drag method is triggered when an object is dragged.
Move	The Move method is used to reposition an object across the screen.
Refresh	The Refresh method ***repaints or redraws an object completely***.
ZOrder	It places a specified form or control at the front or back of the z-order within its graphical level.

EXAMPLE

Create a VB application to perform arithmetic operations using textbox control and command button control.



```
Private Sub Command1_Click()
    Text3.Text = Val(Text1.Text) + Val(Text2.Text)

    End Sub

Private Sub Command2_Click()
    Text3.Text = Val(Text1.Text) - Val(Text2.Text)

    End Sub

Private Sub Command3_Click()
    Text3.Text = Val(Text1.Text) * Val(Text2.Text)

    End Sub

Private Sub Command4_Click()
    Text3.Text = Val(Text1.Text) / Val(Text2.Text)

    End Sub

Private Sub Command5_Click()
    Text3.Text = Val(Text1.Text) Mod Val(Text2.Text)
    End Sub

Private Sub Command6_Click() Text1.Text = ""
    Text2.Text = "" Text3.Text = ""
End Sub

Private Sub Command7_Click()

End
End Sub
```

Access Keys

An *access key* is an underlined character in the text of a menu, menu item, or the label of a control such as a button. With an access key, the user can "click" a button by pressing the Alt key in combination with the predefined access key. For example, if a button runs a procedure to print a form, and therefore its Text property is set to "Print," adding an ampersand before the letter "P" causes the letter "P" to be underlined in the button text at run time. The user can run the command associated with the button by pressing Alt+P.

Controls that cannot receive focus can't have access keys.

Set the Text property to a string that includes an ampersand (&) before the letter that will be the shortcut.

Example:

```
// Set the letter "P" as an access key.
button1.Text = "&Print";
```

To use an ampersand in a caption without creating an access key, include two ampersands (&&).

A single ampersand is displayed in the caption and no characters are underlined.

In the **Properties** window of Visual Studio, set the **Text** property to a string that includes an ampersand ('&') before the letter that will be the access key. For example, to set the letter "P" as the access key, enter **&Print**.

IMAGE CONTROLS

- An image control is used to ***display an image.***
- It uses ***fewer resources*** than a picture box.
- It cannot be used as a container of other controls.

IMAGE PROPERTIES

PROPERTY	DESCRIPTION
Name	It is the ***name by which the image control is referred in the code***.
Picture	It specifies the graphic to be displayed in a control.
Stretch	It is a Boolean property. It specifies a value that determines whether a graphic resizes to fit the size of an image control.
Appearance	It determines whether or not an object is painted with 3-D effects at runtime. 0 – Flat, or 1 - 3-D.
BorderStyle	It specifies the border style of a control. 0 - None, 1 - Fixed Single
Visible	It is a Boolean property. If it is set to true, the image control will be visible to the user.
Enabled	It is a Boolean property. If it is set to true, the user can access the control.
Position Properties	Properties like ***Left, Top, Height, and Width*** can be set at design time or runtime to locate the control at a place of the user's choice.
ToolTipText	It is the text that pops up when the mouse pointer is paused over a control.
Tag	It stores any extra data needed for your program.
DataFormat	It returns a DataFormat object for use against a bindable property of the component.
DataSource	It specifies the data control through which the current control is bound to a database.
DataField	It binds a control to a field in the current record.

IMAGE EVENTS

EVENT	DESCRIPTION
Click	The Click event occurs when a control is clicked.

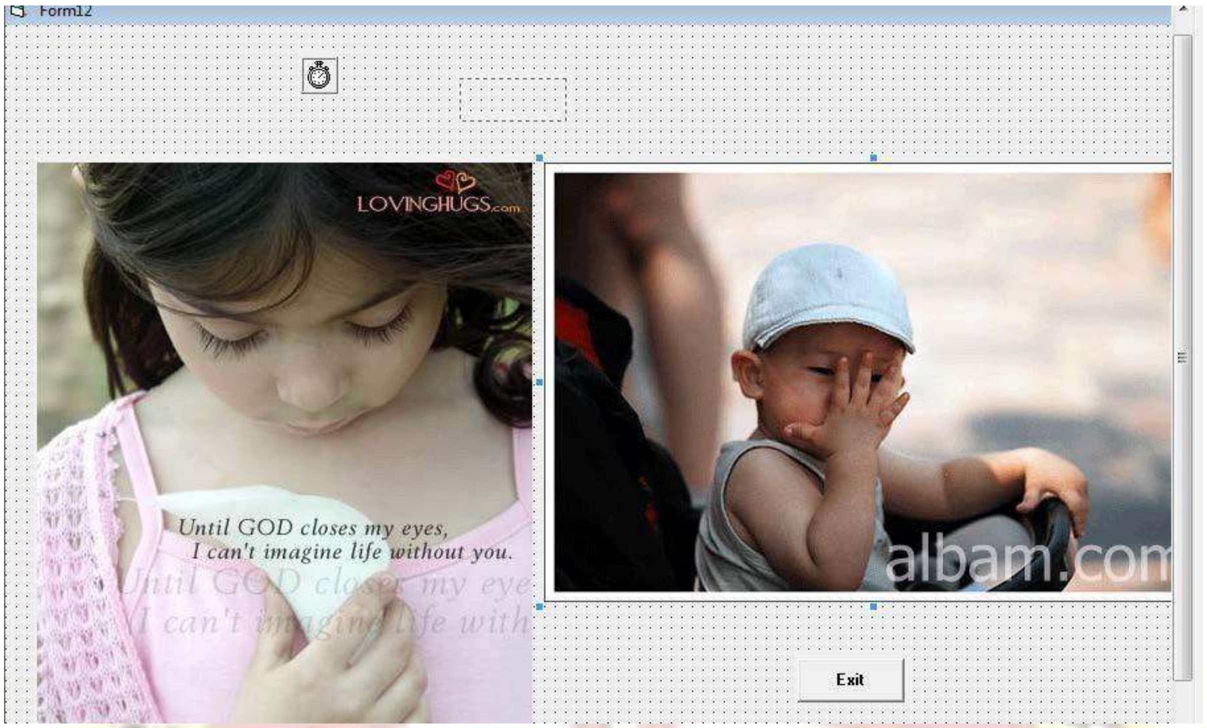
DblClick	The DblClick event occurs when a control is double clicked.
DragDrop	The DragDrop event occurs when a user clicks on an object and drags it to a different location on the screen.
DragOver	The DragOver event occurs when the user drags an object over another control.
MouseDown	The MouseDown event occurs when the user presses any mouse button.
MouseMove	The MouseMove event occurs when the user moves the mouse across the screen.

IMAGE METHODS

METHOD	**DESCRIPTION**
Drag	The Drag method is triggered when an object is dragged.
Move	The Move method is used to reposition an object across the screen.
Refresh	The Refresh method ***repaints or redraws an object completely***.
ZOrder	It places a specified form or control at the front or back of the z-order within its graphical level.

Example

Create a VB application to toggle images using image controls and timer control.



General Declarations

```
Dim flag As Integer Private Sub
Form_Load()
    Image1.Visible = False
    Image2.Visible = False
    Image3.Picture = Image1.Picture
    Timer1.Interval = 3000
    flag = 1 End
Sub
Private Sub Timer1_Timer()
If flag = 1 Then
        Image3.Picture = Image1.Picture
        flag = 2
    Else
        Image3.Picture = Image2.Picture
        flag = 1
End If End
Sub

Private Sub Command1_Click()
End
```

End Sub

TEXT BOX

- Text control is used to ***accept user input*** or to ***display text*** to the user.
- It allows users to ***edit data*** in a text control.

TEXTBOX PROPERTIES

PROPERTY	DESCRIPTION
Name	It is the name by which the textbox is ***referred in the code***.
Text	It specifies the text which is contained in the textbox.
Appearance	It determines whether or not an object is painted with 3-D effects at runtime. 0 - Flat look, 1 - 3-D look.
BorderStyle	It specifies the border style of a control. 0-None, 1-Fixed Single
Alignment	It refers to alignment of text in a textbox. 0 - Left justify, 1- Right justify, 2 - Center
MultiLine	It is a Boolean property. It specifies whether the textbox can display multiple lines of text.
MaxLength	It specifies the maximum number of characters that can be displayed in a text box. A value of 0 indicates text of unlimited length.
PasswordChar	It hides the actual text entered in a textbox with the specified character.
BackColor	It determines the ***background color*** of a textbox. It can be set by using the ***color palette*** which appears beside the property.
ForeColor	It determines the ***text color*** of a textbox. It can be set by using the ***color palette*** which appears beside the property.
DataFormat	It specifies the data format as either number, currency, date, time, percentage, Boolean etc.
DataSource	It specifies the data control through which the current control is bound to a database.
DataField	It binds a control to a field in the current record.
ScrollBars	It specifies a value indicating whether an object has horizontal or vertical scroll bars. 0 - None, 1- Horizontal, 2 - Vertical, 3 - Both

TEXTBOX EVENTS

EVENT	DESCRIPTION
Change	It is triggered when the text property changes.
Click	The Click event occurs when a control is clicked.
DblClick	The DblClick event occurs when a control is double clicked.
GotFocus	The GotFocus event occurs when the specified object receives the focus.
LostFocus	It is triggered when the user leaves the textbox.
KeyDown	The KeyDown event occurs when the user presses a key while a form or control has the focus.
KeyUp	The KeyUp event occurs when the user releases a key while a form or control has the focus.
KeyPress	It is triggered whenever a key is pressed.
MouseDown	The MouseDown event occurs when the user presses any mouse button.
MouseMove	The MouseMove event occurs when the user moves the mouse across the screen.
MouseUp	The MouseUp event occurs when the user releases any mouse button.
DragDrop	The DragDrop event occurs when a user clicks on an object and drags it to a different location on the screen.
DragOver	The DragOver event occurs when the user drags an object over another control.

TEXTBOX METHODS

METHOD	DESCRIPTION
Refresh	The Refresh method ***repaints or redraws an object completely***.

SetFocus	The SetFocus method ***moves focus to the specified Form***, the specified control on the active form, or the specified field on the active datasheet.
Move	The Move method allows a programmer to ***position a control at a desired location***.
Drag	The Drag method is triggered when an object is dragged.
ZOrder	The z-order method places a specified form or control at the front or back of the z-order within its graphical level.

LABEL

Label control allows users to ***display text*** which cannot be edited.

LABEL PROPERTIES

PROPERTY	**DESCRIPTION**
Name	It is the name by which the label is ***referred in the code***.
Caption	It is the text that is displayed in the label.
Appearance	It determines whether or not an object is painted with 3-D effects at runtime. 0 - Flat look, 1 - 3-D look.
Border Style	It specifies the border style of a control. 0-None, 1-Fixed Single
Alignment	It refers to alignment of text in a label. 0 - Left justify, 1- Right justify, 2 - Center
AutoSize	It is a Boolean property. It gets or sets a val`ue specifying if the control should be automatically resized to display all its contents.
WordWrap	If set to true, it automatically wraps text in a label. Default value is false.
Visible	It is a Boolean property. If the property is set to true, the label is visible. Otherwise, it is hidden.
Enabled	It is a Boolean property. By default, this property’s value is True, which means that the control can respond to user-generated events.
BackColor	It determines the ***background color*** of a label. It can be set by using the ***color palette*** which appears beside the property.

ForeColor	It determines the ***text color*** of a label. It can be set by using the ***color palette*** which appears beside the property.
DataField	It binds a control to a field in the current record.
DataMember	It specifies the data member for a data connection.
ToolTipText	It is the text that pops up when the mouse pointer is paused over a control.

LABEL EVENTS

EVENT	**DESCRIPTION**
Change	It is triggered when the text in the label changes.
Click	It is triggered when user clicks on a label.
DblClick	It is triggered when user double clicks on a label.
DragDrop	The DragDrop event occurs when a user clicks on an object and drags it to a different location on the screen.
DragOver	The DragOver event occurs when the user drags an object over another control.
MouseDown	The MouseDown event occurs when the user presses any mouse button.
MouseMove	The MouseMove event occurs when the user moves the mouse across the screen.
MouseUp	The MouseUp event occurs when the user releases any mouse button.

LABEL METHODS

METHOD	**DESCRIPTION**
Drag	The Drag method is triggered when an object is dragged.
Move	The Move method allows a programmer to ***position an object at a desired location***.
Refresh	The Refresh method ***repaints or redraws an object completely***.
ZOrder	It places a specified form or control at the front or back of the z-order within its graphical level.

CHECK BOX

- Check box acts as ***toggle switches***, turning options on or off.
- Check box allows users to ***display multiple choices***.
- It allows users to ***select one or more options*** among the available options.

CHECK BOX PROPERTIES

PROPERTY	DESCRIPTION
Name	It is the ***name by which the Checkbox Box is referred in the code***.
Caption	It is the ***meaningful name*** that appears on a checkbox.
Value	It specifies the value of an object. 0 - Unchecked, 1 - Checked, 2- Grayed. Default value is 0.
BackColor	The BackColor property sets the ***background color*** used to display text and graphics in an object.
ForeColor	It specifies the foreground color used to display text and graphics in an object.
Font Properties	The font properties of a checkbox include the Font, FontName, FontSize, FontBold, FontItalic, FontStrikethru, and FontUnderline.
Postion Properties	Properties like ***Left, Top, Height, and Width*** can be set at design time or at runtime to locate the control at a place of the user's choice.
Picture	The Picture property specifies the picture that will be displayed in the checkbox.
DataFormat	It specifies the data format. The data format can be either number, currency, date, time, percentage, Boolean etc.
DataSource	It specifies the data control through which the current control is bound to a database.
DataField	It binds a control to a field in the current record.
DataMember	It specifies the data member for a data connection.

CHECK BOX EVENTS

EVENT	DESCRIPTION

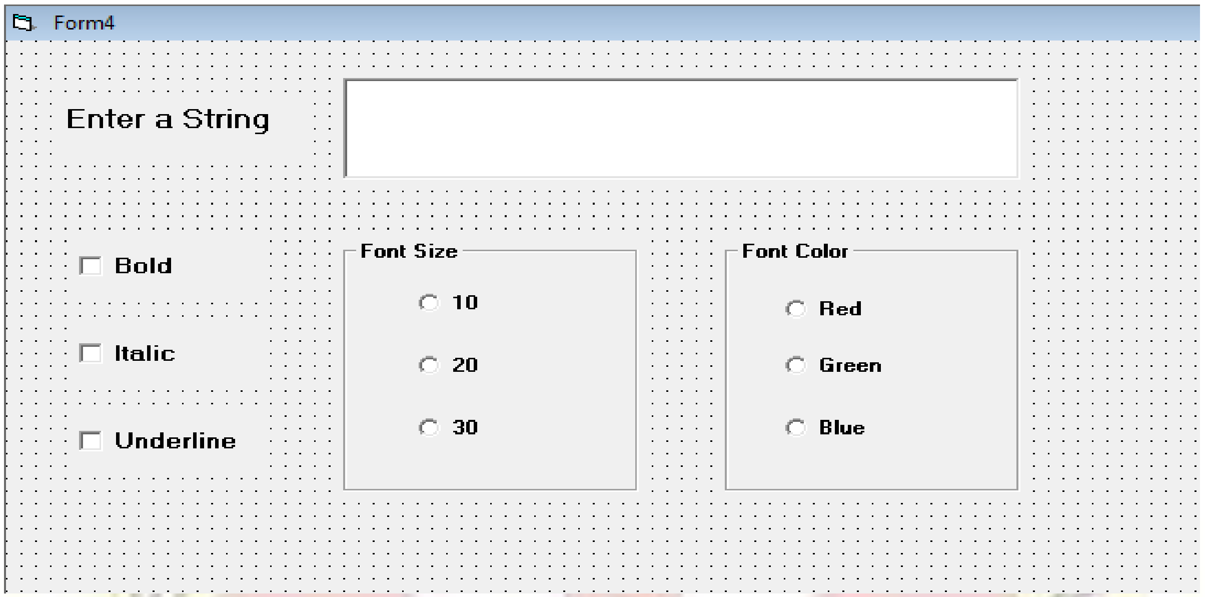
Click	The Click event occurs when a checkbox is clicked.
DragDrop	The DragDrop event occurs when a user clicks on an object and drags it to a different location on the screen.
DragOver	The DragOver event occurs when the user drags an object over another control.
KeyDown	The **KeyDown** event occurs when the user presses a key while a form or control has the focus.
KeyPress	The **KeyPress** event occurs when the user presses and releases a key or key combination while a form or control has the focus.
KeyUp	The **KeyUp** event occurs when the user releases a key while a form or control has the focus.
MouseDown	The MouseDown event occurs when the user presses any mouse button.
MouseMove	The MouseMove event occurs when the user moves the mouse across the screen.
MouseUp	The MouseUp event occurs when the user releases any mouse button.

CHECK BOX METHODS

METHOD	**DESCRIPTION**
Drag	The Drag method is triggered when an object is dragged.
Move	The Move method is used to reposition an object across the screen.
Refresh	The Refresh method ***repaints or redraws an object completely***.
SetFocus	The SetFocus method is used to transfer focus to the control.
ZOrder	It places a specified form or control at the front or back of the z-order within its graphical level.

Example

Create a VB application for text formatting using textbox, checkbox, option button, and frame controls.



```
Private Sub Check1_Click()
If Check1.Value = 1 Then
Text1.FontBold  = True
Else
Text1.FontBold = False
End If
End Sub

Private Sub Check2_Click()

If Check2.Value = 1 Then
Text1.FontItalic = True
Else
Text1.FontItalic = False
End If
End Sub

Private Sub Check3_Click()
If Check3.Value = 1 Then
Text1.FontUnderline = True
Else
Text1.FontUnderline = False
```

```
End If
End Sub
Private Sub Option1_Click()
If Option1.Value = True Then
Text1.FontSize = 10
End If End Sub

Private Sub Option2_Click()
If Option2.Value = True Then
Text1.FontSize = 20
End If End Sub

Private Sub Option3_Click()
If Option3.Value = True Then
Text1.FontSize = 30
End If End Sub

Private Sub Option4_Click()
Text1.ForeColor = RGB(255, 0, 0)
End Sub

Private Sub Option5_Click()
Text1.ForeColor = RGB(0, 255, 0)
End Sub
Private Sub Option6_Click()
Text1.ForeColor = RGB(0, 0, 255)
End Sub
```

FRAME

- Frame control allows users to ***group controls together***.
- To group controls, ***draw the Frame first***, and then ***draw the controls*** inside the frame.

FRAME PROPERTIES

PROPERTY	DESCRIPTION
Name	It is the ***name by which the frame is referred in the code***.
Caption	It is the ***meaningful name*** that appears on a frame.
Appearance	The Appearance property sets whether or not an object is painted at runtime with ***3-D effects***.
BorderStyle	It specifies the border style of a control. 0 - None, 1- Fixed Single
Visible	It is a Boolean property. If the property is set to true, the frame is visible. Otherwise, it is hidden.
Enabled	It is a Boolean property. By default, this property's value is True, which means that the control can respond to user-generated events.
BackColor	It determines the ***background color*** of a frame. It can be set by using the ***color palette*** which appears beside the property.
ForeColor	It determines the ***text color*** of a frame. It can be set by using the ***color palette*** which appears beside the property.

FRAME EVENTS

EVENT	DESCRIPTION
Click	The Click event occurs when a frame is clicked.
DblClick	The DblClick event occurs when a frame is double clicked.
DragDrop	The DragDrop event occurs when a user clicks on an object and drags it to a different location on the screen.
DragOver	The DragOver event occurs when the user drags an object over another control.
MouseDown	The MouseDown event occurs when the user presses any mouse button.
MouseMove	The MouseMove event occurs when the user moves the mouse across the screen.
MouseUp	The MouseUp event occurs when the user releases any mouse button.

FRAME METHODS

METHOD	DESCRIPTION
Drag	The Drag method is triggered when an object is dragged.
Move	The Move method is used to reposition an object across the screen.
Refresh	The Refresh method ***repaints or redraws an object completely***.
ZOrder	It places a specified form or control at the front or back of the z-order within its graphical level.

MESSAGE BOXES: MSGBOX AND INPUTBOX FUNCTIONS

- MsgBox statement can be used to display outputs and messages to the user.
- MsgBox function can be used for inputs.

Syntax

MsgBox message, typecode [, *title*]

Result = MsgBox(*prompt, typecode* [, *title*])

Both message and title are strings, and can be straight text (in quotes), variables, string functions, or combination of these, joined by ampersands (&). Typecode is a number formed by adding together the codes that control which buttons are to appear, which symbol is to be displayed, and which button is to be highlighted when the box opens. Result is an integer.

VALUE	BUTTON SET	CONSTANT
0	OK	vbOKOnly
1	OK, Cancel	vbOKCancel
2	Abort, Retry, Ignore	vBAbortRetryIgnore
3	Yes, No, Cancel	vbYesNoCancel
4	Yes, No	vbYesNo
5	Retry, Cancel	vbRetryCancel

VALUE	SYMBOL	DESCRIPTION
16	✖	vbCritical

32	❓	vbQuestion
48	⚠	vbExclamation
64	ℹ	vbInformation

The value returned by the MsgBox function shows which button was clicked.

VALUE	**BUTTON**
1	OK
2	Cancel
3	Abort
4	Retry
5	Ignore
6	Yes
7	No

The **InputBox Function** is used to get inputs from the users. An InputBox will always display OK and Cancel, and cannot hold symbols.

Syntax

Result = InputBox(prompt[, title] [, default_value])

The default_value is a string that can be displayed in the entry slot of the box, and will be returned if the user presses OK. Clicking Cancel, or pressing the [Esc] key, produces a Null value, which can cause problems. The return value is of Variant type.

UNIT II

DISPLAYING INFORMATION

Visual basic displays the information in text on a form using the Print method. The general format of the Print method applied to a form is

Formname.Print expression

Where, expression is any visual basic expression. An empty Print statement can be used to add a blank line. To suppress the automatic carriage return and line feed, place a semicolon at the end of a Print statement.

The Me keyword can be used to identify the current form. If the form name is left out, the expression will be printed on the current form. To use Print statements in the Form_Load event to display information when the form starts up, include Show statement before any print statements. When the following code is written in the Form_Load and the program is run the text will be displayed on the form.

```
Private Sub Form_Load()
Show
Print “Welcome to Visual Basic”
End Sub
```

When the form is minimized and restored, the text will disappear if the Auto Redraw is set to False. Once the Auto Redraw property is set to True, Visual basic saves a copy of the object in memory. Whenever Visual basic processes on Object, refresh statement, will redraw the object immediately and generate the Paint event if the object supports this.

FONTS

Proportionally spaced fonts are those fonts in which character may be of different widths. For Example: Arial. Non-proportionally spaced fonts are those fonts in which characters are of the same width. For example: Courier New. The Print statement can be made to display its information in a particular position on the form by setting the currentX and currentY.

```
FormName . CurrentX = Value
FormName . CurrentY = Value
```

Where, the value may be any numeric expression. Whenever the Cls method is used to clear a form, Visual Basic reset the CurrentX and CurrentY values to zero. To display information at the beginning of seventh line space set CurrentX and CurrentY as follows:

```
CurrentY = Me . TextHieght (“X”) * 6
CurrentX = 0
```

Where, the TextHeight returns the amount of vertical space need to display the string.

TAB AND SPC COMMANDS

The Tab function lets to move to a specific column and start printing there. Its syntax is

Print Tab (ColumnNumber%)

Where, ColumnNumber% is an integral expression. If the current column position is greater than its value, Tab skips to this column on the next line.

The Spc function inserts the specified number of spaces into a line starting at the current print position. Its syntax is

Spc(Integer%)

FORMAT FUNCTION

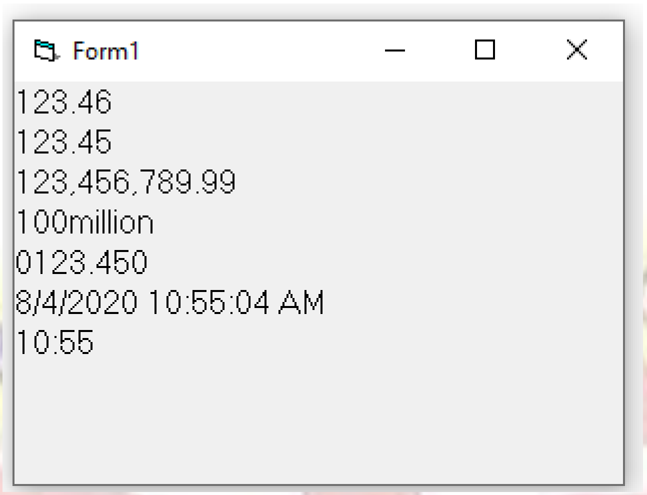
The format function works with a number and a template called a format string. The syntax is

Format (NumericExpression, FormatString$)

For example, the following Form_Load event procedure is run.

```
Private Sub Form_Load()
    Show
    Me.Font.Size = 12
    Me.Print Format(123.456789, "###.##")
    Me.Print Format(123.45, "###.###")
    Me.Print Format(123456789.991, "#,#.##")
    Me.Print Format(100000000, "#00,,") & ("million")
    Me.Print Format(123.45, "0000.000")
    Me.Print Format(Now, "General Date")
    Me.Print Format(Now, "Short Time")
End Sub
```

Output



```
123.46
123.45
123,456,789.99
100million
0123.450
8/4/2020 10:55:04 AM
10:55
```

LOOPING STATEMENTS

Looping statements are used to ***repeat a group of statements*** until certain specified conditions are met. This can be achieved by a repetition structure which allows the programmer to repeat an action until given condition is true. This repeated structure forms a loop. Two types of loops are available, determinate loop and indeterminate loop.

DETERMINATE LOOPS

Repeating the loop for a certain number of fixed times is called a determinate loop.

FOR … NEXT STATEMENT

It is an ***entry-controlled*** loop. The test condition is evaluated and if the condition is true, then the body of the loop is executed. When the test condition becomes false, control is transferred out of the loop and execution continues with the statement immediately after the body of the loop. It allows positive and negative increments.

Syntax

```
For counter = start To end [Step stepValue]
    [statements]
    [Exit For]
    [statements]
Next [counter]
```

EXAMPLE

Create a VB application to calculate the sum and average of 'n' numbers using For … Next statement.

```
  sum = 0
  avg = 0
  For i = 1 To n
     arr(i) = InputBox("Enter a number") sum = sum + arr(i)
  Next i
  avg = sum / n
  Text2.Text = sum Text3.Text = avg
End Sub
```

DO … LOOP STATEMENT

- The Do … Loop statement can be either ***entry-controlled*** or ***exit-controlled***.
- In an exit-controlled loop, the body of the loop is always ***executed at least once***.
- In a **Do … While** loop, the body of the loop is executed while the test condition is true. When the test condition becomes false, control is transferred out of the loop and execution continues with the statement immediately after the body of the loop.
- In a **Do … Until** loop, the body of the loop is executed while the test condition is false. When the test condition becomes true, control is transferred out of the loop and execution continues with the statement immediately after the body of the loop.

Syntax

ENTRY-CONTROLLED	EXIT-CONTROLLED

Do [{**While | Until**} *condition*]
    [statements]
[**Exit Do**]
    [statements]
Loop
	Do
	    [statements] [**Exit Do**]
	    [statements]
	Loop [{**While | Until**}*condition*]

Example

Create a VB application to calculate the sum and average of ‘n’ numbers using Do … While statement.

```
Private Sub Command2_Click()
Dim i As Integer
    Dim sum As Integer
    Dim avg As Integer
    Dim cnt  As Integer
    n = Val(Text1.Text)
    sum = 0
    avg = 0
    i = 1
    Do While i <= n
        arr(i) = InputBox("Enter a number")
        sum = sum + arr(i)
        i = i + 1
    Loop
    avg = sum / n
    Text2.Text = sum
    Text3.Text = avg
End Sub
```

Example

Create a VB application to calculate the sum and average of ‘n’ numbers using Do … Until statement.

```
Private Sub Command3_Click() Dim i
    As Integer
    Dim sum As Integer Dim
    avg As Integer Dim cnt As
    Integer n =
    Val(Text1.Text) sum = 0
    avg = 0
    i = 1
    Do Until i > n
        arr(i) = InputBox("Enter a number") sum = sum
        + arr(i)
        i = i + 1 Loop
    avg = sum / n
    Text2.Text = sum
    Text3.Text = avg
End Sub
```

WHILE … WEND STATEMENT

- It is an ***entry-controlled*** loop. The test condition is evaluated and if the condition is true, then the body of the loop is executed. When the test condition becomes false, control is transferred out of the loop and execution continues with the statement immediately after the body of the loop.

Syntax

```
While condition
    [statements]
Wend
```

Example

Create a VB application to calculate the sum and average of ‘n’ numbers using while statement.

```
Private Sub Command1_Click() Dim i
    As Integer
    Dim sum As Integer Dim
    avg As Integer Dim cnt As
    Integer n =
    Val(Text1.Text) sum = 0
    avg = 0

    i = 1
    While i <= n
        arr(i) = InputBox("Enter a number")
        sum = sum + arr(i)
        i = i + 1
    Wend
    avg = sum / n
    Text2.Text = sum
    Text3.Text = avg
End Sub
```

INDETERMINATE LOOPS

Indeterminate loop, which repeats the action of execution until certain condition is to be satisfied.

CONDITIONALS

Branching statements are used to ***change the order of execution*** of statements based on certain conditions. Branching statements include if statement and select … case statements.

IF STATEMENT

If ... Else statement is a ***two-way decision*** statement. It is used to control the flow of execution of statements based on the outcome of a condition.

Syntax

```
If condition Then [ifstatements] [Else elsestatements]
```

If the test condition is true, then if statements is executed. Otherwise, else statements is executed.

Example

Create a VB application to check whether a number is odd or even.

```
Private Sub Command1_Click()
```

```
    Dim n As Integer
    n = Val(Text1.Text) If n Mod 2 = 0 Then
        MsgBox n & " is even"
         Else
        MsgBox n & " is odd"
    End If
End Sub
```

IF … ELSEIF STATEMENT

If ... Else if statement is used when ***multipath decisions*** are involved.

Syntax

The conditions will be ***tested in sequence***. As soon as a condition is found to be true, the statement-block associated with it will be executed and the rest of the ladder is skipped. When all n conditions are false, then the else statements-block will be executed.

Example

Create a VB application to print the biggest of three numbers.

If *condition* **Then**

    [ifstatements]

    [**ElseIf** *condition-n* **Then**

    [elseifstatements]

[Else

    [elsestatements]]

End If

```
Private Sub Command1_Click() Dim a As
    Integer
    Dim b As Integer Dim c As
    Integer Dim big As Integer
    big = 0
    a = Val(Text1.Text)
    b = Val(Text2.Text)
    c = Val(Text3.Text)
    If ((a > b) And (a > c)) Then
```

```
     big = a
    ElseIf (b > c) Then
    big = b
    Else
        big = c
    End If
    MsgBox "Biggest Number is " & big
    End Sub
```

NESTED IF STATEMENT

Nested If ... Else statement is used when a ***series of decisions*** are involved.

Syntax

```
If condition  Then
If condition Then
        [ifstatements]
    Else
        [elsestatements]]
    End If
    Else
    [elsestatements]]
End If
```

Example

Create a VB application to calculate bonus of a person based on his age, gender, and balance.

```
Private Sub Command1_Click()
Dim age As Integer
    Dim gender As String
    Dim balance As Integer
    Dim bonus As Double
    age = Val(Text1.Text)
    gender = Trim(Text2.Text)
```

```
balance = Val(Text3.Text) If (age > 60)
Then
    If (gender = "f" Or gender = "F") Then
    bonus = 0.05 * balance
    ElseIf (gender = "m" Or gender = "M") Then
    bonus = 0.04 * balance
    Else
         bonus = 0.03 * balance
         End If
Else
       bonus = 0.02 * balance
       End If
   MsgBox ("Bonus = " & bonus)
   End Sub
```

SELECT STATEMENT

Select … Case statement is used when there are many alternate paths all based on the value of a variable.

Syntax

```
Select Case testexpression
[Case expressionlist-n
     [statements-n]]
          ...
[Case Else
     [elsestatements]]
End Select
```

Example

Create a VB application to print the day of a week using select … case statement.

```
Private Sub Command1_Click()
    Dim n As Integer
    n = Val(Text1.Text)
    Select Case n
```

```
        Case 1:
            MsgBox "Day is Sunday"
        Case 2:
            MsgBox "Day is Monday"
        Case 3:
            MsgBox "Day is Tuesday"
        Case 4:
            MsgBox "Day is Wednesday"
        Case 5:
            MsgBox "Day is Thursday"
        Case 6:
            MsgBox "Day is Friday"
        Case 7:
            MsgBox "Day is Saturday"
        Case Else:
          MsgBox "Enter a number between 1 and 7"
        End Select
End Sub
```

BUILT-IN-FUNCTIONS

Visual Basic provides many built-in functions which (usually) accept one or more arguments, and return a value based on the argument(s).

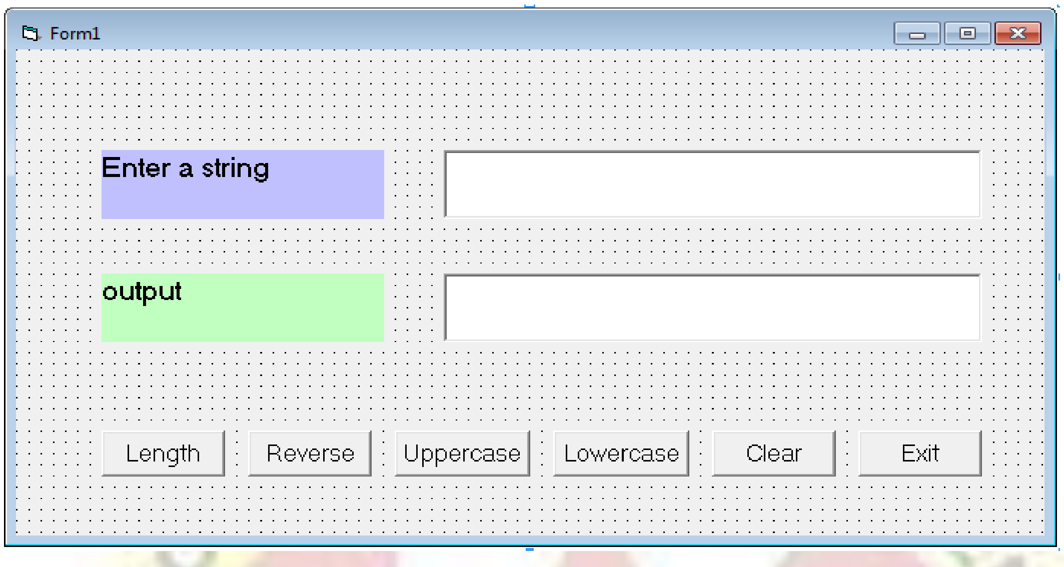
STRING FUNCTIONS

STRING FUNCTION	**DESCRIPTION**	**SYNTAX**
Trim	Removes leading and trailing spaces in a string.	**Trim**(*string*)
Ltrim	Removes leading spaces in a string.	**LTrim**(*string*)
Rtrim	Removes trailing spaces in a string.	**RTrim**(*string*)
Len	Returns an integer value which is the length of a string including empty spaces.	**Len**(*string*)
LCase	Converts all the characters of a string to small letters	**LCase**(*string*)

UCase	Converts all the characters of a string to capital letters	**UCase**(*string*)
Left	Extracts a specified number of characters from the beginning of a string.	**Left**(*string*, *length*)
Right	Extracts a specified number of characters from the end of a string.	**Right**(*string*, *length*)
Mid	Extracts a substring from the original string.	**Mid**(*string*, *start*[, *length*])
StrComp	Compares string1 and string2 and returns a value that represents the result of the comparison. It returns -1 if string1 < string2. It returns 0 if both strings are equal. It returns 1 if string1 > string 2.	**StrComp**(*string1*, *string2*[, *compare*])
StrReverse	Reverses a string.	**StrReverse**(*string1*)
InStr	Returns the position of the first occurrence of one string within another.	**InStr**([*start*,]*string1*, *string2*[, *compare*])
InStrRev	Returns the position of the last occurrence of one string within another string.	**InstrRev**(*string1*, *string2*[, *start*[, *compare*]])
Replace	Finds a string in an expression and replaces with another string.	**Replace**(*expression*, *find*, *replacewith* [, *start* [, *count*[, *compare*]]])
Str()	Returns the string equivalent of a number.	**Str**(number)
String()	Returns “character” n times.	**String(n, "Character")**
Split	Used to breakup a string at specified places.	**Split**(*expression*[, *delimiter*[, *count*[, *compare*]]])
Join	Used to build a larger string out of smaller strings.	**Join**(*list*[, *delimiter*])

EXAMPLE

Create a VB application using string functions.



```
Private Sub Command1_Click()
Text2.Text = Len(Text1.Text)
End Sub

Private Sub Command2_Click()
     Text2.Text = StrReverse(Text1.Text)
     If Text1.Text = Text2.Text Then
          MsgBox "Given string is palindrome"
     Else
           MsgBox "Given string is not palindrome"
      End If
End Sub

Private Sub Command3_Click()
Text2.Text = UCase(Text1.Text)
End Sub

 Private Sub Command4_Click()
 Text2.Text = LCase(Text1.Text)
 End Sub
```

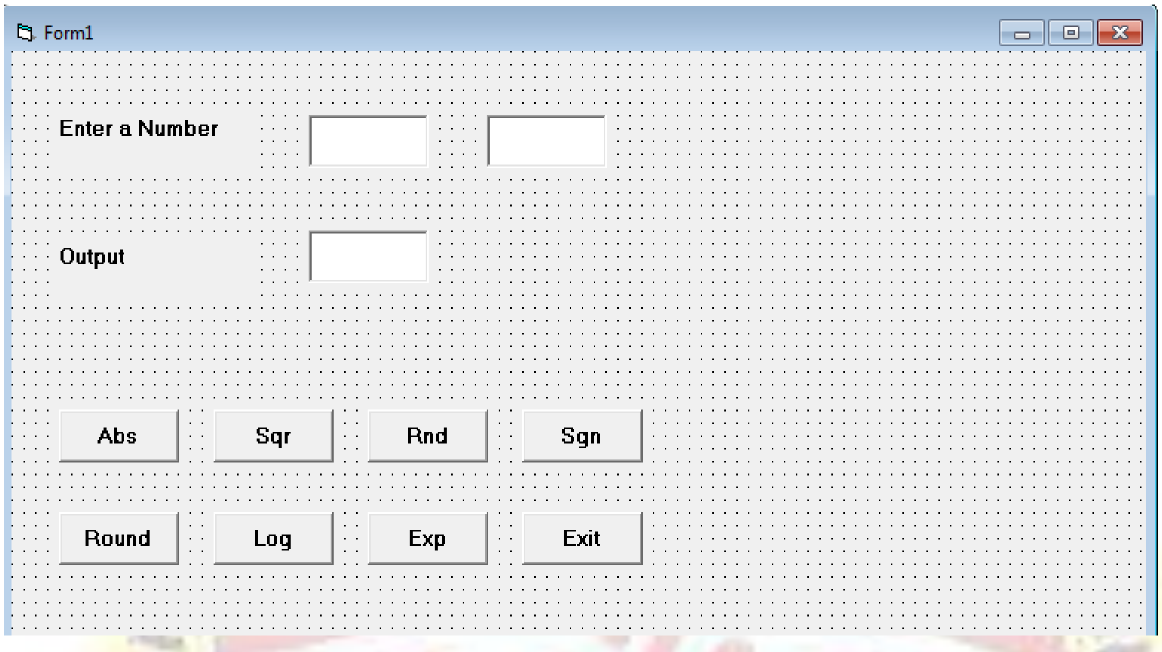
```
    Private Sub Command5_Click()
    Text1.Text = ""
    Text2.Text = ""
End Sub

Private Sub Command6_Click()
End
End Sub
```

MATH FUNCTIONS (Numeric)

MATH FUNCTION	DESCRIPTION	SYNTAX
Abs	Returns the absolute (unsigned) value of the argument.	**Abs**(*num*)
Sqr	Returns the square root value of the argument	**Sqr**(*num*)
Rnd	Returns a random value between 0 and 1	**Rnd**[(*num*)]
Randomize	It uses number to initialize the Rnd function's random-number generator, giving it a new seed value.	**Randomize**
Sgn	Returns sign of a number	**Sgn(**Num**)**
Round	Rounds a number n to m decimal places	**Round** (n, m)
Log	Returns the natural logarithm of the argument	**Log**(*num*)
Exp	Exp of a number x is the value of e^x	**Exp**()
Sin	Returns the sine value of the argument in *Radians*	**Sin**(*num*)
Cos	Returns the cosine value of the argument in *Radians*	**Cos**(*num*)
Tan	Returns the tangent value of the argument in *Radians*	**Tan**(*num*)

EXAMPLE

Create a VB application using math functions.



```
Private Sub Form_Load()
Text2.Visible = False
End Sub

Private Sub Command1_Click()
        Text2.Visible = False
        Text3.Text = Abs(Val(Text1.Text))
End Sub

Private Sub Command2_Click()
        Text2.Visible = False
        Text3.Text = Sqr(Val(Text1.Text))
End Sub

Private Sub Command3_Click()
       Text2.Visible = False
       Text3.Text = Rnd(Val(Text1.Text))
End Sub
```

```
Private Sub Command4_Click()
        Text2.Visible = False
        Text3.Text = Sgn(Val(Text1.Text))
End Sub

Private Sub Command5_Click()
    Text2.Visible = True Text2.SetFocus
    Text3.Text = Round(Val(Text1.Text), Val(Text2.Text))
End Sub

Private Sub Command6_Click()
        Text2.Visible = False
        Text3.Text = Log(Val(Text1.Text))
End Sub

Private Sub Command7_Click()
        Text2.Visible = False
        Text3.Text = Exp(Val(Text1.Text))
End Sub
```

OPERATORS

An operator is a ***symbol*** that is used to manipulate data and variables called ***operands*** to produce a resultant value. Visual Basic supports the following types of operators:

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Assignment Operator
- Concatenation Operators

ARITHMETIC OPERATORS

Arithmetic operators are used to perform arithmetic operations.

OPERATOR	DESCRIPTION	EXAMPLE	RESULT
+	Addition	5+6	11
-	Subtraction, Unary minus	14-6	8
*	Multiplication	5*4	20
/	Division	25/5	5
\	Integer Division	20/3	6
Mod	Modulo Division	20/3	2
^	Exponentiation	3^3	27

Example

```
Private Sub Command1_Click()
    Text3.Text = Val(Text1.Text) + Val(Text2.Text)
End Sub
```

RELATIONAL OPERATORS

Relational operators are used to ***compare two quantities***. An expression containing a relational operator is called a ***relational expression***. The value of a relational expression is either ***true* or *false***.

OPERATOR	DESCRIPTION	EXAMPLE	RESULT
=	Equal to	10 > 8	True
<>	Not equal to	5<>4	True
<	Less than	20<5	False
>	Greater than	25 > 10	True
<=	Less than or equal to	21 <=21	True
>=	Greater than or equal to	5>=5	True
Is	Compares references		

LOGICAL OPERATORS

Logical operators are used to ***combine two or more conditions*** and make decisions. An expression containing a logical operator is called a ***logical expression***. A logical expression returns either ***true*** or ***false***.

OPERATOR	DESCRIPTION	EXAMPLE
And	True, if both the operands are true	(age > 21) And (salary > 50000)
Or	True, if one of the operands is true	(num == 0) Or (num == 1)
Not	Negation of Operand	Not a

ASSIGNMENT OPERATOR

Assignment operator is used to assign the result of an expression to a variable.

OPERATOR	DESCRIPTION	EXAMPLE	RESULT
=	Assigns result of an expression to a variable	a = 3+5*2	a=13

CONCATENATION OPERATORS

Concatenation operator is used for joining strings.

OPERATOR	DESCRIPTION	EXAMPLE	RESULT
+	Joins a number with a string or joins two strings	“abc” + “def”	abcdef
&	Joins two strings	“Jenefa” & ” “ & “Joy”	Jenefa Joy

FUNCTIONS AND PROCEDURES

We use procedures and functions to create modular programs. Visual Basic statements are grouped in a block enclosed by Sub, Function and matching End statements. The difference between the two is that functions return values, procedures do not.A procedure and function is a piece of code in a larger program. They perform a specific task. The advantages of using procedures and functions are:

- Reducing duplication of code
- Decomposing complex problems into simpler pieces
- Improving clarity of the code
- Reuse of code
- Information hiding

FUNCTIONS

A function is a block of Visual Basic statements inside Function, End Function statements. Functions return values.There are two basic types of functions. Built-in functions and user defined ones. The built-in functions are part of the Visual Basic language. There are various mathematical, string or conversion functions.

```
Module Example

   Sub Main()

      Console.WriteLine(Math.Abs(-23))
      Console.WriteLine(Math.Round(34.56))
      Console.WriteLine("ZetCode has {0} characters", _Len("ZetCode"))

   End Sub

End Module
```

In the preceding example, we use two math functions and one string function. **Built-in functions** help programmers do some common tasks. In the following example, we have a user defined function.

```
Module Example

   Dim x As Integer = 55
   Dim y As Integer = 32

   Dim result As Integer

   Sub Main()

      result = Addition(x, y)
      Console.WriteLine(Addition(x, y))

   End Sub

   Function Addition(ByVal k As Integer, _ByVal l As Integer) As Integer
```

```
        Return k+l
    End Function

End Module
```

Two values are passed to the function. We add these two values and return the result to the Main() function.

```
result = Addition(x, y)
```

Addition function is called. The function returns a result and this result is assigned to the result variable.

```
Function Addition(ByVal k As Integer, _ByVal l As Integer) As Integer
    Return k+l
End Function
```

This is the Addition function signature and its body. It also includes a return data type, for the returned value. In our case is an Integer. Values are returned to the caller with the Return keyword.

PROCEDURES

A procedure is a block of Visual Basic statements inside Sub, End Sub statements. Procedures do not return values.

```
Module Example

    Sub Main()

        SimpleProcedure()

    End Sub

    Sub SimpleProcedure()
        Console.WriteLine("Simple procedure")
    End Sub

End Module
```

This example shows basic usage of procedures. In our program, we have two procedures. The Main() procedure and the user defined SimpleProcedure(). As we already know, the Main() procedure is the entry point of a Visual Basic program.

```
SimpleProcedure()
```

Each procedure has a name. Inside the Main() procedure, we call our user defined SimpleProcedure() procedure.

```
Sub SimpleProcedure()
   Console.WriteLine("Simple procedure")
End Sub
```

Procedures are defined outside the Main() procedure. Procedure name follows the Sub statement. When we call a procedure inside the Visual Basic program, the control is given to that procedure. Statements inside the block of the procedure are executed. Procedures can take optional parameters.

```
Module Example

   Sub Main()
      Dim x As Integer = 55
      Dim y As Integer = 32
      Addition(x, y)
   End Sub

   Sub Addition(ByVal k As Integer, ByVal l As Integer)
      Console.WriteLine(k+l)
   End Sub

End Module
```

In the above example, we pass some values to the Addition() procedure.

```
Addition(x, y)
```

Here we call the Addition() procedure and pass two parameters to it. These parameters are two Integer values.

```
Sub Addition(ByVal k As Integer, ByVal l As Integer)
   Console.WriteLine(k+l)
End Sub
```

We define a *procedure signature*. A procedure signature is a way of describing the parameters and parameter types with which a legal call to the function can be made. It contains the name of the procedure, its parameters and their type, and in case of functions also the return value. The ByVal keyword specifies how we pass the values to the procedure. In our case, the procedure obtains two numerical values, 55 and 32. These numbers are added and the result is printed to the console.

LISTS

- The list box control is used to ***display a list of items***.
- It allows the user to ***choose any one item*** from the list.
- It can display only a set of items that are available.
- Items can only be added to the list during ***run-time***, not at design time.

LIST BOX PROPERTIES

PROPERTY	**DESCRIPTION**
Name	It is the ***name by which the ListBox is referred in the code***.
List	It specifies the items contained in a list box.
Columns	It specifies a value that determines whether a list box scrolls vertically in a single column or horizontally in snaking columns.
MultiSelect	It specifies a value that determines whether a user can make multiple selections in a control.
Sorted	It is a Boolean property. It indicates whether the elements of a control are automatically sorted alphabetically. The default value is False.
Visible	It is a Boolean property. If the property is set to true, the list box is visible. Otherwise, it is hidden.
Enabled	It is a Boolean property. By default, this property's value is True, which means that the control can respond to user-generated events.
BackColor	It determines the ***background color*** of a list box. It can be set by using the ***color palette.***
ForeColor	It determines the ***forecolor*** of a list box. It can be set by using the ***color palette.***
DataFormat	It specifies the data format as either number, currency, date, time, percentage, Boolean etc.
DataSource	It specifies the data control through which the current control is bound to a database.
DataField	It binds a control to a field in the current record.

DataMember	It specifies the data member for a data connection.
ItemData	It denotes a specific number for each item in a listbox control.

LIST BOX EVENTS

EVENT	DESCRIPTION
Click	The Click event occurs when an item in a listbox is clicked. The code written in the Click event procedure is invoked.
DblClick	The Click event occurs when an item in a listbox is double clicked.
ItemCheck	The ItemCheck event occurs when an item in the listbox is checked. It will work when the style of the listbox is set to 1- Checkbox
DragDrop	The DragDrop event occurs when a user clicks on an object and drags it to a different location on the screen.
DragOver	The DragOver event occurs when the user drags an object over another control.
KeyDown	The KeyDown event occurs when the user presses a key while a form or control has the focus.
KeyPress	The KeyPress event occurs when the user presses and releases a key or key combination while a form or control has the focus.
KeyUp	The KeyUp event occurs when the user releases a key while a form or control has the focus.
MouseDown	The MouseDown event occurs when the user presses any mouse button while the mouse pointer is over an object.
MouseMove	The MouseMove event occurs when the user moves the mouse across an object.
MouseUp	The MouseDown event occurs when the user releases any mouse button while the mouse pointer is over an object.

LIST BOX METHODS

METHOD	DESCRIPTION

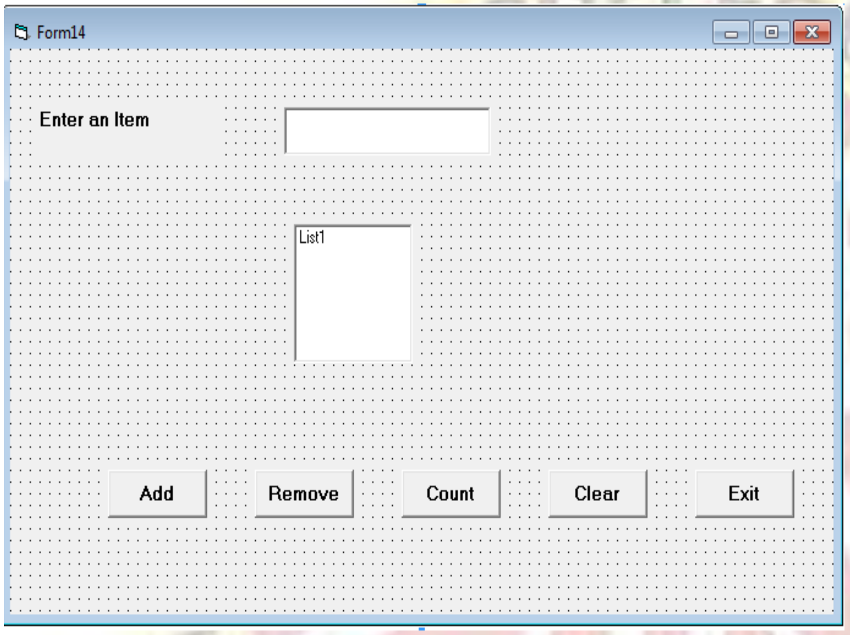
AddItem	It is used to insert an item into a listbox at runtime.
RemoveItem	It removes an item from a listbox at runtime as identified by the listindex.
Clear	It removes all items from a listbox.
Refresh	The Refresh method refreshes a listbox when items are added to or removed from a listbox.
Drag	The Drag method is triggered when an object is dragged.
Move	The Move method is used to reposition an object across the screen.
SetFocus	The SetFocus method is used to transfer focus to the listbox.

Example

Create a VB application to perform various operations on a listbox.

```
Private Sub Command1_Click()
    List1.AddItem (Text1.Text)
End Sub

Private Sub  Command2_Click()

If List1.ListIndex = -1 Then
        MsgBox "Please select an item in the list box and then click remove button"
    Else
        List1.RemoveItem (List1.ListIndex)

     End If
End Sub

Private Sub Command3_Click()
    MsgBox "Number of items in listbox is " & List1.ListCount

End Sub

Private Sub Command4_Click()
List1.Clear Text1.Text = ""
```

```
End Sub

Private Sub Command5_Click()
End
End Sub
```



ARRAYS

This array discusses about the following topics.

1. Fixed Size Array
   - 1-D Array
   - 2-D Array
2. Dynamic Array
   - Declaration
   - Redim Keyword
   - Preserve Keyword
3. LBound and UBound Functions
4. Control Array

Array and its types are discussed below

- Array is a ***set of similar items*** having the ***same name*** and the elements of an array are identified by their ***index (subscript)*** values.
- In VB, subscript numbering starts from 0 (zero) unless specified otherwise.
- The **scope of array** depends on the ***place and method of declaration***.
- The values for the ***subscripts*** of an array can be passed using variables.
- Arrays are of ***three Categories***:
  a) Fixed-size Array
  b) Multidimensional Array
  c) Dynamic Array

FIXED-SIZE ARRAY

- When the total number of elements the array will hold is known in advance, the ***size of the array can be specified at the time of its declaration***. Such types of arrays are called **fixed-size arrays**.
- For fixed-size arrays, specifying the **array size** (**upper bound)** is **compulsory**.

ONE DIMENSIONAL ARRAY

Declaration of 1-D Array *Syntax :*

Dim | Private | Public ArrayName(subscript) As Data Type

To create a **local array**, declare the array ***inside a procedure*** using the ***Private*** keyword / **Dim** keyword.

Dim arr(10) As Integer

1. To create a **module-level array**, declare the array in the ***declaration section of a module*** using the ***Private*** keyword.

   Private arr(10) As Integer
2. To create a **public array**, declare the array in the ***declaration section of a module*** using ***Public*** keyword.

   Public arr(10) As Integer

The following declaration statement creates an array with name 'arr' which can hold a maximum of **11 elements**.

Dim arr(10) As Integer

The ***first element*** is denoted by **arr(0)** and the ***last element*** is denoted by **arr(10)**.

```
Dim a(1 to 10) As  Integer
```

The above statement creates an array ‘a’ whose ***first element*** is denoted by **a(1)** and the ***last element*** is denoted by **a(10)**.

```
Dim x(10 to 20) As Integer
```

The above statement creates an array ‘x’ whose ***first element*** is denoted by **x(10)** and the ***last element*** is denoted by **x(10)**.

Example

```
   Dim arr(5) As Integer
   Dim n As Integer
   Dim key As Integer
Private Sub Command1_Click()
          Dim i As Integer
          n = Val(Text1.Text)
          For i = 1 To n
                arr(i) = InputBox("Enter element")
                Next i
End Sub
Private Sub Command2_Click()
             Dim i As Integer
             Dim flag As Boolean
             Dim pos As Integer
             key = Val(Text2.Text)
             flag = False
             For i = 1 To n
                 If (arr(i) = key) Then
                 flag = True
                       pos = i
                       Exit For
                 End If
                  Next i
```

```
            If (flag = True) Then
                MsgBox key & " is found is postion " & pos
                Else
                MsgBox "Key not found"
                End If
End Sub
```

MULTI-DIMENSIONAL ARRAY

- An array having **more than one dimension** is called a multidimensional array.
- A **table** of data can be represented by a **two dimensional array**.
- An array having three dimensions is called a three dimensional array.
- A multidimensional array can have up to ***60 dimensions***.
- A multidimensional array takes up a ***lot of space***.

Example

Dim stud(2, 1) As Integer

The above statement declares a 2-D array having ***three rows and two columns***.

Dim sales(11, 2, 4) As Integer

The above statement declares a 3-D array which can be used to store sales details of a company for **12 months, three departments and five products**.

EXAMPLE 2-D ARRAY

DEPARTMENT	**BOYS**	**GIRLS**
BCA	130	80
BCS	110	50
BIT	90	30

General Declarations

```
    Dim m As Integer
    Dim n As Integer
    Dim stud(5, 5) As Integer
Private Sub Command1_Click()
```

```
Dim i As Integer
        Dim j As Integer
            m = Val(Text1.Text)
            n  =  Val(Text2.Text)
            For i = 1 To m
                For j = 1 To n
                    stud(i, j) = InputBox("Enter value")
                    Next j
            Next i
End Sub
Private Sub Command2_Click()
            Dim rowtot As Integer
            Dim coltot As Integer
            Dim grandtot As Integer
            grandtot = 0
            For i = 1 To m
            rowtot = 0
                For j = 1 To n
                    rowtot = rowtot + stud(i, j)
                    Next j
                   MsgBox "Row total= " & rowtot
                   grandtot = grandtot + rowtot
            Next i
            For j = 1 To n
            coltot = 0
                For i = 1 To m
                    coltot = coltot + stud(i, j)
                    Next i
                   MsgBox "Column total= " & coltot
                   Next j
            MsgBox "Grand Total = " & grandtot
```

End Sub

DYNAMIC ARRAY

- Dynamic arrays are used when the total number of elements the array will hold is not known in advance.
- A dynamic array can be ***resized*** at any time and it helps to ***manage memory*** efficiently.
- The **ReDim** is used in conjunction with the Dim statement while declaring dynamic arrays.
- **ReDim** can be used to ***change the upper bound or lower bound of a dimension*** of a dynamic array.
- **ReDim** cannot be used to change the number of dimensions of a dynamic array.
- **ReDim cannot** be used to **change the data type of an array,** unless the array is previously declared as Variant.
- ReDim **is an** executable statement that can appear only inside a procedure**.**

Declaration of dynamic array syntax

Dim | Private | Public   ArrayName[(subscript)]   As   Data Type

Example

```
Dim DynSales( ) As  Integer
```

The above statement creates an **open-ended array** called DynSales.

```
ReDim DynSales(11, 5 )
```

General   Declarations

```
Dim arr( ) 'open-ended array
Private Sub Command2_Click()
        ReDim arr(5)

        arr(0) = 10
        arr(1) = 20
        arr(2) = 30
        ReDim Preserve arr(3)
```

```
            MsgBox  arr(0)
            MsgBox  arr(1)
            MsgBox arr(2)
           ReDim Preserve arr(UBound(arr) + 1) lb = LBound(arr)
            ub = UBound(arr)
            MsgBox "Lower Bound = " & lb
            MsgBox "Upper Bound = " & ub
End Sub
```

Output

```
            10 20 30
            Lower Bound = 0
            Upper Bound = 4
```

LBOUND AND UBOUND FUNCTIONS

- The function LBound( ) returns the ***lower bound*** of an array.
- The function UBound( ) returns the ***upper bound*** of an array.
- In case of a ***single dimensional dynamic array***, the array can be increased by one element without losing the values of the existing elements using the UBound( ) function.

Example

```
Private Sub Command1_Click()
Dim counter(25)
     lb = LBound(counter)
     ub = UBound(counter)
     MsgBox "Lower Bound = " & lb
     MsgBox "Upper Bound = " & ub
End Sub
```

Output

Lower Bound = 0

Upper Bound = 25

CONTROL ARRAY

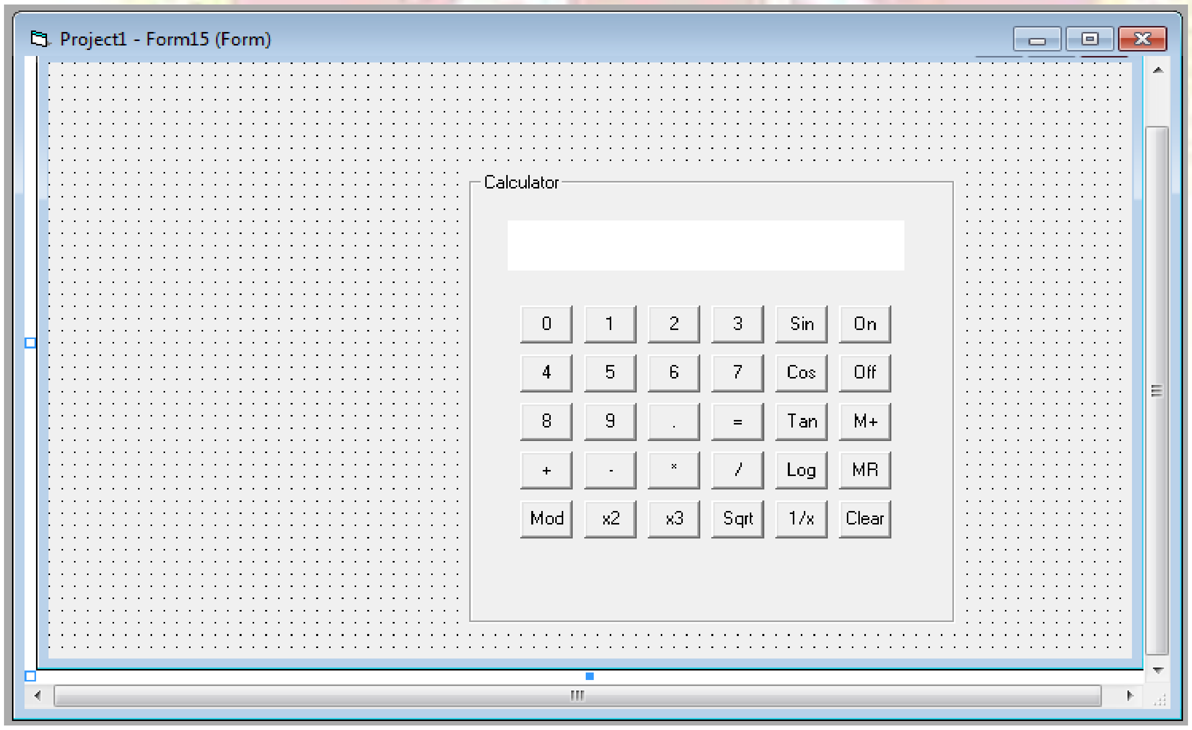
Control Array is a ***set of identical controls*** that have a ***common name*** and identifying index numbers.

Example

Scientific Calculator using Command Button Control Array

Steps

- Draw a Frame and change its caption to ‘Calculator’.
- Draw a Label inside the frame.
- Create a control array named ‘button’ using command buttons. The control array consists of the elements 0 to 9, . (decimal point), =, +, -, *, /, Mod, x2, x3, sqrt, Sin, Cos, Tan, Log, and 1/x with index from 0 to 24 respectively.
- Create command buttons On, Off, M+, MR and Clear.



Note:

Initially, draw a command button inside the frame (Do not double click command button to create it, and drag and drop inside frame. Instead draw it inside the frame). Change the name of the command button as ‘button’ and change its caption to ‘0’. Copy the button and paste it. A dialog box containing “You already have a control named ‘button’. Do you want to create a control array?“ will appear. Click ‘Yes’ button. Create the control array using the above procedure and change the caption of each button as appropriate.

```
Dim n As Double
Dim opr As Integer
Dim memory As Double
    Private Sub Form_Load()
    Label1.Caption = ""
End Sub

Private Sub button_Click(Index As Integer)
Select Case Index
Case 0 To 9
Label1.Caption = Label1.Caption & Index
Case 10
Label1.Caption = Label1.Caption & button(Index).Caption
Case 12 To 24
n = CDbl(Label1.Caption)
Label1.Caption = ""
opr = Index
Case 11
Select Case opr
Case 12
Label1.Caption = n + CDbl(Label1.Caption)
 Case 13
Label1.Caption = n - CDbl(Label1.Caption)
Case 14
Label1.Caption = n * CDbl(Label1.Caption)
Case 15
Label1.Caption = n / CDbl(Label1.Caption)
```

```
Case 16
Label1.Caption = n Mod CDbl(Label1.Caption)
Case 17
Label1.Caption = CDbl(n * n)
Case 18
Label1.Caption = CDbl(n * n * n)
Case 19
Label1.Caption = n ^ (1 / 2)
Case 20
Label1.Caption = Sin(n * 3.14 / 180)
Case 21
Label1.Caption = Cos(n * 3.14 / 180)
Case 22
Label1.Caption = Tan(n * 3.14 / 180)
Case 23
Label1.Caption = Log(n)
Case 24
Label1.Caption = 1 / n
End Select
End Select
 End Sub
Private Sub Clear_Click()
Label1.Caption = ""
End Sub

Private Sub On_Click()
Dim i As Integer
For i = 1 To 19
button(i).Enabled = True
Next i
Off.Enabled = True
```

```
Mem.Enabled = True
Mr.Enabled = True
Label1.Caption = ""
End Sub
Private Sub Off_Click()
Dim i As Integer
For i = 1 To 19
button(i).Enabled = False
Next i
Off.Enabled = False
Mem.Enabled = False
Mr.Enabled = False
Label1.Caption = ""
End Sub

Private Sub Mem_Click()
memory = CDbl(Label1.Caption)
End Sub

Private Sub Mr_Click()
Label1.Caption = memory
End Sub
```

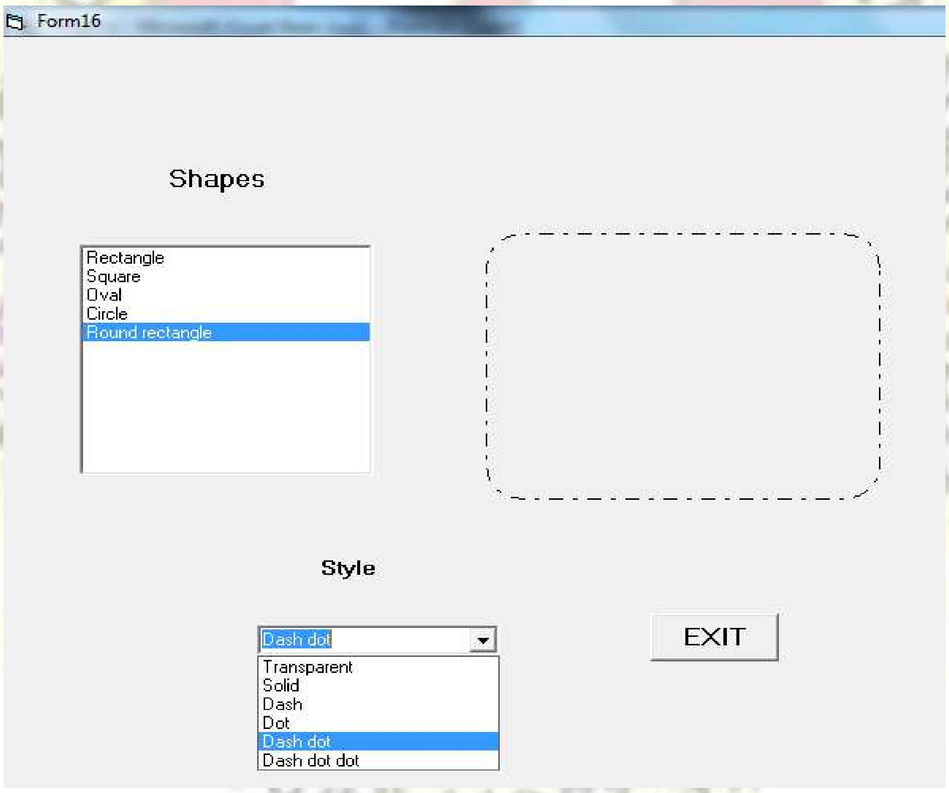
COMBO BOX METHODS

METHOD	DESCRIPTION
AddItem	It is used to insert an item into a combobox at runtime.
RemoveItem	It is used to remove an item from a combobox, as identified by its listindex at runtime.
Clear	It removes all items from a combobox.

Refresh	The Refresh method refreshes a combobox when items are added to or removed from a combobox.
Drag	The Drag method is triggered when an object is dragged.
Move	The Move method is used to reposition an object across the screen.

EXAMPLE

Create a VB application to draw various shapes using listbox, combobox, and shape controls.



Private Sub Form_Load()

List1.AddItem "Rectangle" List1.AddItem "Square"

List1.AddItem "Oval" List1.AddItem "Circle"

List1.AddItem "Round rectangle" Combo1.AddItem

```
        "Transparent" Combo1.AddItem "Solid"
        Combo1.AddItem "Dash" Combo1.AddItem "Dot"
        Combo1.AddItem "Dash dot" Combo1.AddItem "Dash
        dot not"
End Sub

Private Sub List1_Click()
Select Case List1.ListIndex
        Case 0
                Shape1.Shape = 0
        Case 1
                Shape1.Shape = 1
        Case 2
                Shape1.Shape = 2
        Case 3
                Shape1.Shape = 3
        Case 4
                Shape1.Shape = 4
        Case 5
                Shape1.Shape = 5 End Select
End Sub

Private Sub Combo1_Change()

Select Case Combo1.ListIndex
        Case 0
                Shape1.BorderStyle = 0
        Case 1
                Shape1.BorderStyle = 1
        Case 2
                Shape1.BorderStyle = 2
        Case 3
                Shape1.BorderStyle = 3
        Case 4
                Shape1.BorderStyle = 4
        Case 5
```

```
            Shape1.BorderStyle = 5 End Select
    End Sub

    Private Sub Combo1_Click()
     Select Case Combo1.ListIndex
        Case 0
            Shape1.BorderStyle = 0
        Case 1
            Shape1.BorderStyle = 1
        Case 2
            Shape1.BorderStyle = 2
        Case 3
            Shape1.BorderStyle = 3
        Case 4
            Shape1.BorderStyle = 4
        Case 5
            Shape1.BorderStyle = 5
             End Select
    End Sub
```

PROJECTS WITH MULTIPLE FORMS

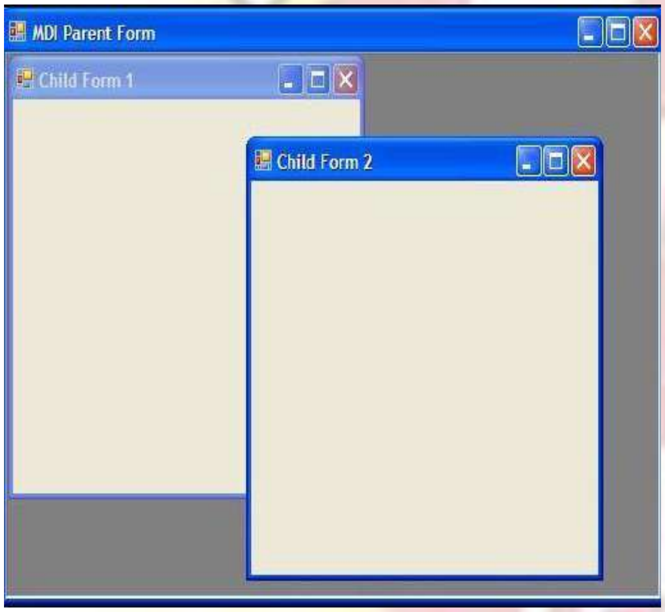
Select File, New, Project from the main menu in Visual Studio. Then pick a Visual Basic Windows Application to create. A form will be created with a default name of Form1. Add a second form by right-clicking the project and selecting Add. Add Windows Form from the menu that appears.

WRITING THE VISUAL BASIC CODE TO ADD THE CHILDREN TO THE MDI PARENT

The next step is to add the two new forms Multiple Document Interface (MDI) (*MDIchild1* and *MDIchild2*) to the parent form (*MDIparent*). To do this click on the tab for the parent form in Visual Studio and double click on the form to display the event procedures. We will now write Visual Basic code to add the two child forms to the container parent form. To do this we will set the *MdiParent* property of each child to reference the *MDIparent* form. Note that because this is the *Load* event of the actual parent form, we refer to it with the keyword *Me* rather than by the form name. Having set the *Mdiparent* of each child we then need to display the form using the form *Show()* method:

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
        MDIchild1.MdiParent = Me
        MDIchild1.Show()
        MDIchild2.MdiParent = Me
        MDIchild2.Show()
End Sub
```

Output

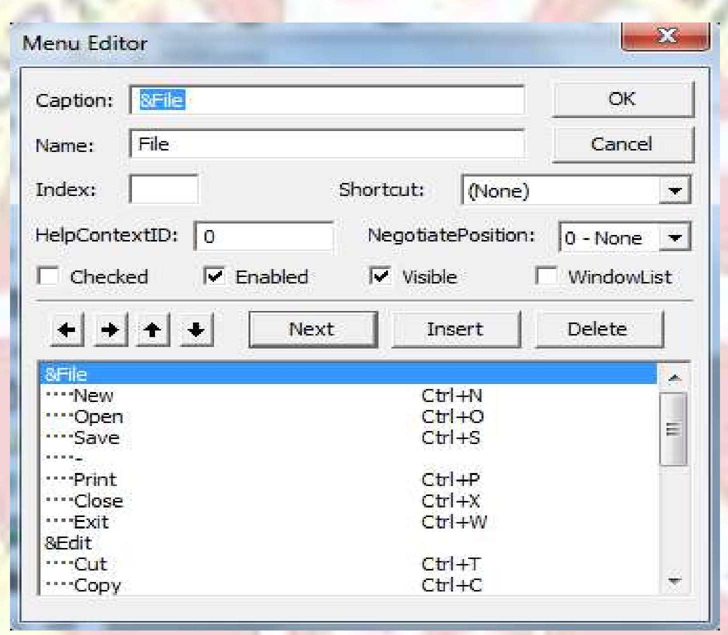


MENUS

- All Windows-compliant applications will have a menu system.
- A menu system contains a ***number of options***, logically organized and easily accessible by the user.
- Each menu item has a ***name***, ***caption***, and optionally a ***shortcut key***.
- When a user clicks a menu option, a list of options is displayed. Clicking on any menu item will generate a **Click event**. User can write code to respond to the click event.
- A menu system is tied to a form, it cannot exist independent of a form.

Creating a Menu System

- Select **Tools -> Menu Editor** or click the Menu Editor icon on the tool bar or press the keys **Ctrl+E**.
- This opens the Menu Editor. Type values for caption, name and shortcut (optional).
- If you want to enable selection by an **Alt-keystroke** combination, then type an
  - **ampersand (&)** in front of the key letter in caption property.
- To insert a **separator bar**, select the menu item before which a separator bar should be inserted and enter – (hyphen) in the caption box and type a name in the name box.



Menu Editor

Caption: &File
Name: File
Index:
Shortcut: (None)
HelpContextID: 0
NegotiatePosition: 0 - None
☐ Checked ☑ Enabled ☑ Visible ☐ WindowList

```
&File
····New                          Ctrl+N
····Open                         Ctrl+O
····Save                         Ctrl+S
····-
····Print                        Ctrl+P
····Close                        Ctrl+X
····Exit                         Ctrl+W
&Edit
····Cut                          Ctrl+T
····Copy                         Ctrl+C
```

MENU PROPERTIES AND EVENTS

PROPERTY	DESCRIPTION
Name	Specifies the name of the menu item. It helps to access the menu item through code.
Caption	Allows you to enter the menu or command name that you want to appear on your menu bar or in a menu.
Checked	Allows you to have a check mark appear initially at the left of a menu item.

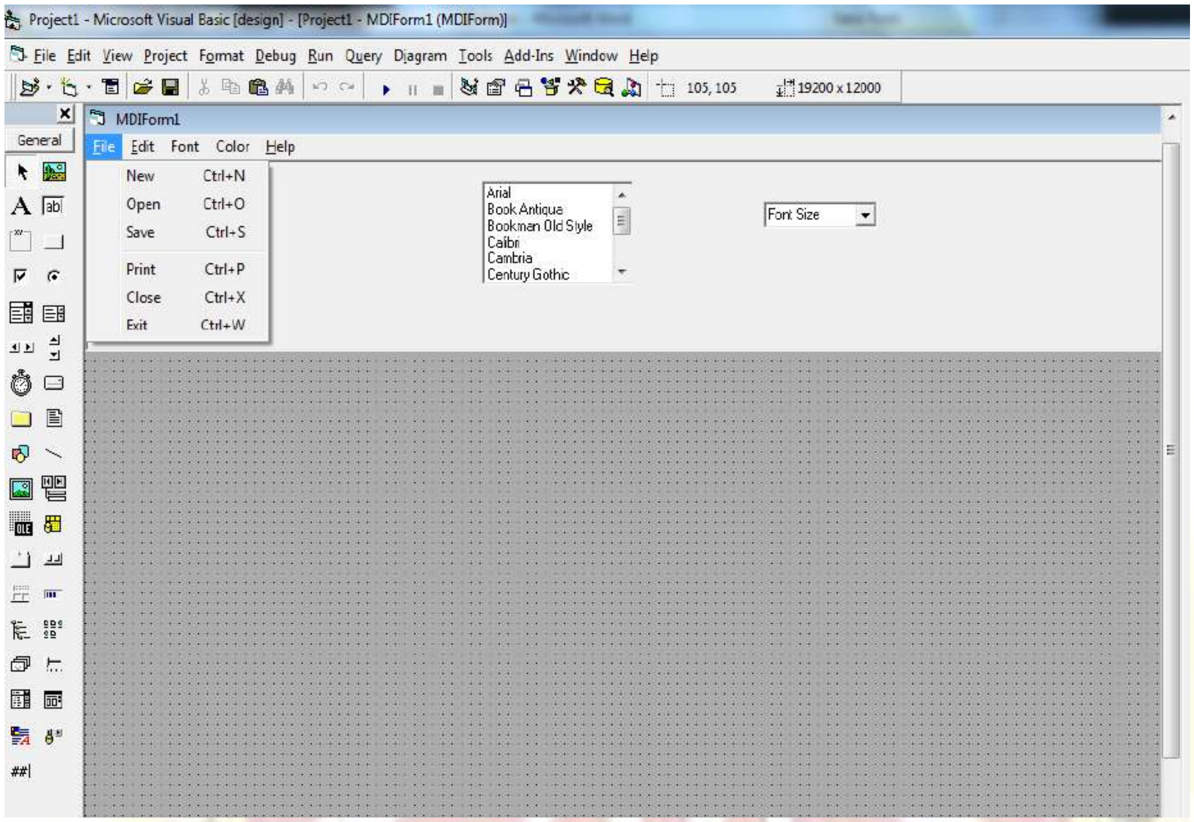
Enabled	Allows you to select whether you want the menu item to respond to events, or clear if you want the item to be unavailable and appear dimmed.
Shortcut	Allows you to select a shortcut key for each command.
Visible	Allows you to have the menu item appear on the menu.
WindowList	Determines if the menu control contains a list of open MDI child forms in an MDI application.

EVENT	DESCRIPTION
Click	Specifies code to be executed when a menu item is clicked.

MDI FORM

- An MDI form is a ***container*** for other forms in an application.
- Multiple Document Interface (MDI) Form allows an application to have multiple windows within the main window**.**
- It is used to display more than one document within the main window.
- Most of the control buttons and code for various forms can be shared.
- If a VB project contains an MDI form, any ordinary forms with **MDIChild property** set to True will appear as windows within the MDI form at runtime.
- An MDI form can have **menus**, **Picture box** control, **Toolbar** and **Data control**. Other types of controls can be placed within a Picture box.
- A VB project can contain ***only one MDI form***.

- As a general rule, only one Child form is set up at design time, and additional forms can be created at run time either by using an ***array*** or by ***loading new instances*** of the original form.


Project1 - Microsoft Visual Basic [design] - [Project1 - MDIForm1 (MDIForm)]
File Edit View Project Format Debug Run Query Diagram Tools Add-Ins Window Help
105, 105
19200 x 12000
General
MDIForm1
File Edit Font Color Help
New Ctrl+N
Open Ctrl+O
Save Ctrl+S
Print Ctrl+P
Close Ctrl+X
Exit Ctrl+W
Arial
Book Antiqua
Bookman Old Style
Calibri
Cambria
Century Gothic
Font Size

Example

```
        Dim a() As New Childform1
        Dim i As Integer
Private Sub MDIForm_Load()
i = 0
End Sub

Private Sub New_Click()
ReDim a(i + 1)
        a(i).RichTextBox1.Font = "ms sans serif"
```

```
            a(i).Caption = "Child" & Val(i + 1)
             a(i).Show
            a(i).SetFocus i = i + 1
End Sub

Private Sub Open_Click()
            Dim fname As String
            a(i).CommonDialog1.ShowOpen
            fname = a(i).CommonDialog1.FileName
            a(i).RichTextBox1.LoadFile (fname)
End Sub

Private Sub Save_Click()
            Dim fname As String
            a(i).CommonDialog1.ShowSave
              fname = a(i).CommonDialog1.FileName
                a(i).RichTextBox1.SaveFile (fname)
End Sub

Private Sub Print_Click()
            Dim fname As String a(i).CommonDialog1.ShowPrinter
            fname = a(i).CommonDialog1.FileName Printer.Print
            fname
End Sub

Private Sub Close_Click()
Unload Screen.ActiveForm
End Sub

Private Sub Exit_Click()
End
End Sub

Private Sub Cut_Click()
```

```
            Clipboard.SetText (a(i).RichTextBox1.SelText)
             a(i).RichTextBox1.SelText = ""
End Sub

Private Sub Copy_Click()
    Clipboard.SetText (a(i).RichTextBox1.SelText)

    End Sub

Private Sub Paste_Click() a(i).RichTextBox1.SelText = Clipboard.GetText
End Sub

Private Sub Clear_Click() a(i).RichTextBox1.Text = ""
End Sub

Private Sub List1_Click()

a(i).RichTextBox1.SelFontName = List1.List(List1.ListIndex)

End Sub

Private Sub FontName_Click()
a(i).RichTextBox1.SelFontName = List1.List(List1.ListIndex)

End Sub

Private Sub Combo1_Change()

a(i).RichTextBox1.SelFontSize = Val(Combo1.Text)
End Sub

Private Sub Combo1_Click()

a(i).RichTextBox1.SelFontSize = Val(Combo1.Text)
End Sub

Private Sub FontSize_Click()

a(i).RichTextBox1.SelFontSize = Val(Combo1.Text)
End Sub
```

```
Private Sub Bold_Click()

a(i).RichTextBox1.SelBold = True
End Sub

Private Sub Italic_Click()

a(i).RichTextBox1.SelItalic = True
End Sub

Private Sub Underline_Click()

a(i).RichTextBox1.SelUnderline = True
End Sub

Private Sub Backcolor_Click()

a(i).CommonDialog1.ShowColor

a(i).RichTextBox1.Backcolor = a(i).CommonDialog1.Color

End Sub

Private Sub Forecolor_Click()
a(i).CommonDialog1.ShowColor
a(i).RichTextBox1.SelColor = a(i).CommonDialog1.Color

End Sub

Private Sub Contents_Click()
        a(i).CommonDialog1.ShowHelp
End Sub
```

UNIT III

INTRODUCTION TO DATABASE MANAGEMENT SYSTEM

A **database-management system** (DBMS) is a collection of interrelated data and a set of programs to access those data. This is a collection of related data with an implicit meaning and hence is a database.

A **datum** – a unit of data – is a symbol or a set of symbols which is used to represent something. This relationship between symbols and what they represent is the essence of what we mean by information.

As the name suggests, the database management system consists of two parts. They are:

- Database and
- Management System

WHAT IS A DATABASE?

To find out what database is, we have to start from data, which is the basic building block of any DBMS.

Data: Facts, figures, statistics etc. having no particular meaning (e.g. 1, ABC, 19 etc).

Record: Collection of related data items, e.g. in the above example the three data items had no meaning. But if we organize them in the following way, then they collectively represent meaningful information.

Roll	Name	Age
1	ABC	19

Table or Relation: Collection of related records.

Roll	Name	Age
1	ABC	19

2	DEF	22
3	XYZ	28

The columns of this relation are called Fields, Attributes or Domains. The rows are called Tuples or Records.

Database: Collection of related relations. Consider the following collection of tables:

T1

Roll	**Name**	**Age**
1	ABC	19
2	DEF	22
3	XYZ	28

T2

Roll	**Address**
1	KOL
2	DEL
3	MUM

T3

Roll	**Year**
1	I
2	II
3	I

T4

Year	**Hostel**
I	H1

II	H2

We now have a collection of 4 tables. They can be called a "related collection" because we can clearly find out that there are some common attributes existing in a selected pair of tables. Because of these common attributes we may combine the data of two or more tables together to find out the complete details of a student. Questions like "Which hostel does the youngest student live in?" can be answered now, although Age and Hostel attributes are in different tables. A database in a DBMS could be viewed by lots of different people with different responsibilities

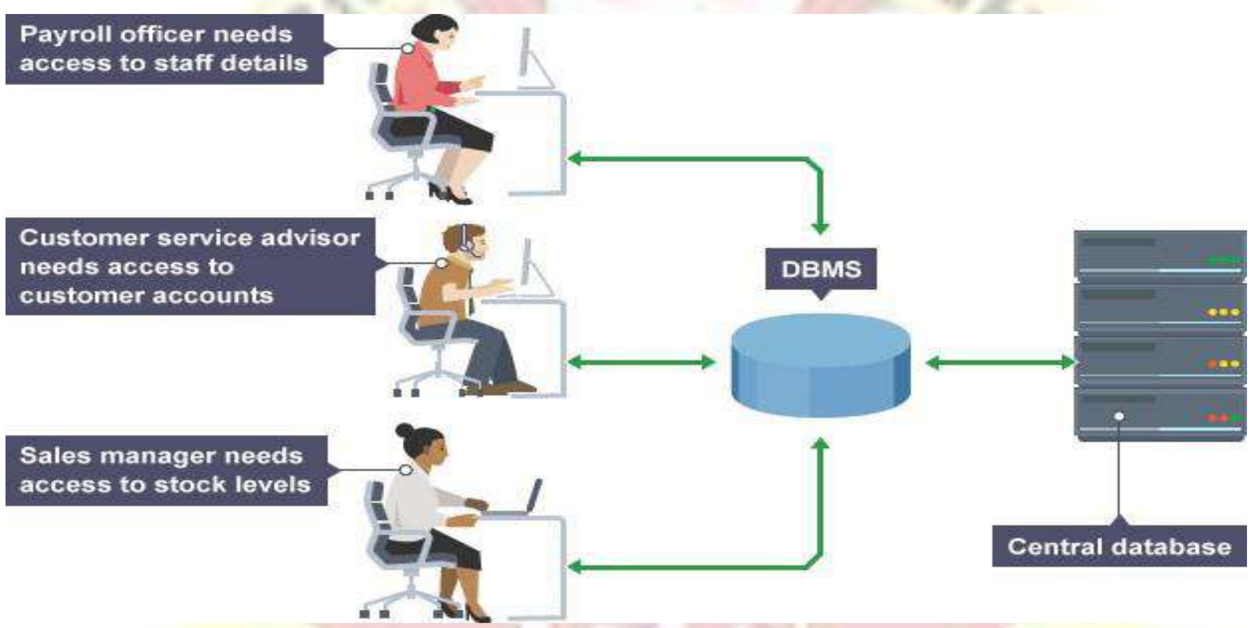

Payroll officer needs access to staff details
Customer service advisor needs access to customer accounts
Sales manager needs access to stock levels
DBMS
Central database

Figure 1 Empolyees are accessing Data through DBMS

For example, within a company there are different departments, as well as customers, who each need to see different kinds of data. Each employee in the company will have different levels of access to the database with their own customized **front-end** application. In a database, data is organized strictly in row and column format. The rows are called **Tuple** or **Record**. The data items within one row may belong to different data types. On the other hand, the columns are often called **Domain** or **Attribute**. All the data items within a single attribute are of the same data type.

WHAT IS MANAGEMENT SYSTEM?

A database-management system (DBMS) is a collection of interrelated data and a set of programs to access those data. This is a collection of related data with an implicit meaning and hence is a database. The collection of data, usually referred to as the database, contains information relevant to an enterprise. The primary goal of a DBMS is to provide a way to store and retrieve database information that is both convenient and efficient. By data, we mean known facts that can be recorded and that have implicit meaning.

The management system is important because without the existence of some kind of rules and regulations it is not possible to maintain the database. We have to select the particular attributes which should be included in a particular table; the common attributes to create relationship between two tables; if a new record has to be inserted or deleted then which tables should have to be handled etc. These issues must be resolved by having some kind of rules to follow in order to maintain the integrity of the database.

Database systems are designed to manage large bodies of information. Management of data involves both defining structures for storage of information and providing mechanisms for the manipulation of information. In addition, the database system must ensure the safety of the information stored, despite system crashes or attempts at unauthorized access. If data are to be shared among several users, the system must avoid possible anomalous results.

Because information is so important in most organizations, computer scientists have developed a large body of concepts and techniques for managing data.

DATABASE MANAGEMENT SYSTEM (DBMS) AND ITS APPLICATIONS

A Database management system is a computerized record-keeping system. It is a repository or a container for collection of computerized data files. The overall purpose of DBMS is to allow the users to define, store, retrieve and update the information contained in the database on demand. Information can be anything that is of significance to an individual or organization.

Databases touch all aspects of our lives. Some of the major areas of application are as follows:

- Banking
- Airlines
- Universities
- Manufacturing and selling

- Human resources

ENTERPRISE INFORMATION

- Sales: For customer, product, and purchase information.
- Accounting: For payments, receipts, account balances, assets and other accounting information.
- Human resources: For information about employees, salaries, payroll taxes, and benefits, and for generation of paychecks.
- Manufacturing: For management of the supply chain and for tracking production of items in factories, inventories of items in warehouses and stores, and orders for items.

BANKING AND FINANCE

- Banking: For customer information, accounts, loans, and banking transactions.
- Credit card transactions: For purchases on credit cards and generation of monthly statements.
- Finance: For storing information about holdings, sales, and purchases of financial instruments such as stocks and bonds; also for storing real-time market data to enable online trading by customers and automated trading by the firm.
- Universities: For student information, course registrations, and grades (in addition to standard enterprise information such as human resources and accounting.
- Airlines: For reservations and schedule information. Airlines were among the first to use databases in a geographically distributed manner.
- Telecommunication: For keeping records of calls made, generating monthly bills, maintaining balances on prepaid calling cards, and storing information about the communication networks.

ADVANTAGES OF DBMS

Controlling of Redundancy: Data redundancy refers to the duplication of data (i.e storing same data multiple times). In a database system, by having a centralized database and centralized control of data by the DBA the unnecessary duplication of data is avoided. It also eliminates the extra time for processing the large volume of data. It results in saving the storage space.

Improved Data Sharing: DBMS allows a user to share the data in any number of application programs.

Data Integrity: Integrity means that the data in the database is accurate. Centralized control of the data helps in permitting the administrator to define integrity constraints to the data in the database. For example: in customer database we can enforce integrity that it must accept the customer only from Noida and Meerut city.

Security: Having complete authority over the operational data, enables the DBA in ensuring that the only mean of access to the database is through proper channels. The DBA can define authorization checks to be carried out whenever access to sensitive data is attempted.

Data Consistency: By eliminating data redundancy, we greatly reduce the opportunities for inconsistency. For example: is a customer address is stored only once, we cannot have disagreement on the stored values. Also updating data values is greatly simplified when each value is stored in one place only. Finally, we avoid the wasted storage that results from redundant data storage.

Efficient Data Access: In a database system, the data is managed by the DBMS and all access to the data is through the DBMS providing a key to effective data processing.

Enforcements of Standards: With the centralized of data, Database Administrators (DBA) can establish and enforce the data standards which may include the naming conventions, data quality standards etc.

Data Independence: Ina database system, the database management system provides the interface between the application programs and the data. When changes are made to the data representation, the data obtained by the DBMS is changed but the DBMS is continues to provide the data to application program in the previously used way. The DBMs handles the task of transformation of data wherever necessary.

Reduced Application Development and Maintenance Time: DBMS supports many important functions that are common to many applications, accessing data stored in the DBMS, which facilitates the quick development of application.

DISADVANTAGES OF DBMS

- It is bit complex. Since it supports multiple functionality to give the user the best, the underlying software has become complex. The designers and developers should have thorough knowledge about the software to get the most out of it.
- Because of its complexity and functionality, it uses large amount of memory. It also needs large memory to run efficiently.
- DBMS system works on the centralized system, i.e.; all the users from all over the world access this database. Hence any failure of the DBMS, will impact all the users.
- DBMS is generalized software, i.e.; it is written work on the entire systems rather specific one. Hence some of the application will run slow.

COMPONENTS OF DBMS

There are the following components of DBMS:

- Software
- Hardware
- Procedures
- Data
- Users

SOFTWARE

- The main component of a Database management system is the software. It is the set of programs which is used to manage the database and to control the overall computerized database.
- The DBMS software provides an easy-to-use interface to store, retrieve, and update data in the database.
- This software component is capable of understanding the Database Access Language and converts it into actual database commands to execute or run them on the database.

HARDWARE

- This component of DBMS consists of a set of physical electronic devices such as computers, I/O channels, storage devices, etc that create an interface between computers and the users.
- This DBMS component is used for keeping and storing the data in the database.

PROCEDURES

- Procedures refer to general rules and instructions that help to design the database and to use a database management system.
- Procedures are used to setup and install a new database management system (DBMS), to login and logout of DBMS software, to manage DBMS or application programs, to take backup of the database, and to change the structure of the database, etc.

DATA

- It is the most important component of the database management system.
- The main task of DBMS is to process the data. Here, databases are defined, constructed, and then data is stored, retrieved, and updated to and from the databases.
- The database contains both the metadata (description about data or data about data) and the actual (or operational) data.

USERS

- The users are the people who control and manage the databases and perform different types of operations on the databases in the database management system. There are three types of user who play different roles in DBMS:
  - ❖ Application Programmers
  - ❖ Database Administrators
  - ❖ End-Users

APPLICATION PROGRAMMERS

The users who write the application programs in programming languages (such as Java, C++, or Visual Basic) to interact with databases are called Application Programmer.

DATABASE ADMINISTRATORS (DBA)

A person who manages the overall DBMS is called a database administrator or simply DBA.

END-USERS

The end-users are those who interact with the database management system to perform different operations by using the different database commands such as insert, update, retrieve, and delete on the data, etc.

CLASS DIAGRAM

Class diagram describes the attributes and operations of a class and also the constraints imposed on the system. The class diagrams are widely used in the modeling of object oriented systems because they are the only Unified Modeling Language (UML) diagrams, which can be mapped directly with object-oriented languages. A class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects. A UML class diagram is made up of a set of classes and a set of relationships between classes.

Class diagram is a static diagram. It represents the static view of an application. Class diagram is not only used for visualizing, describing, and documenting different aspects of a system but also for constructing executable code of the software application.

Class diagram shows a collection of classes, interfaces, associations, collaborations, and constraints. It is also known as a structural diagram.

PURPOSE OF CLASS DIAGRAMS

The purpose of class diagram is to model the static view of an application. Class diagrams are the only diagrams which can be directly mapped with object-oriented languages and thus widely used at the time of construction. UML (Unified Modeling Language) is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems.

UML diagrams like activity diagram, sequence diagram can only give the sequence flow of the application, however class diagram is a bit different. It is the most popular UML diagram in the coder community.

The purpose of the class diagram can be summarized as

- Analysis and design of the static view of an application.
- Describe responsibilities of a system.
- Base for component and deployment diagrams.
- Forward and reverse engineering.

UNIFIED MODELING LANGUAGE (UML) - BASIC NOTATIONS

UML is popular for its diagrammatic notations. We all know that UML is for visualizing, specifying, constructing and documenting the components of software and non-software systems. Hence, visualization is the most important part which needs to be understood and remembered.

UML notations are the most important elements in modeling. Efficient and appropriate use of notations is very important for making a complete and meaningful model. The model is useless, unless its purpose is depicted properly.Hence, learning notations should be emphasized from the very beginning. Different notations are available for things and relationships. UML diagrams are made using the notations of things and relationships. Extensibility is another important feature which makes UML more powerful and flexible.

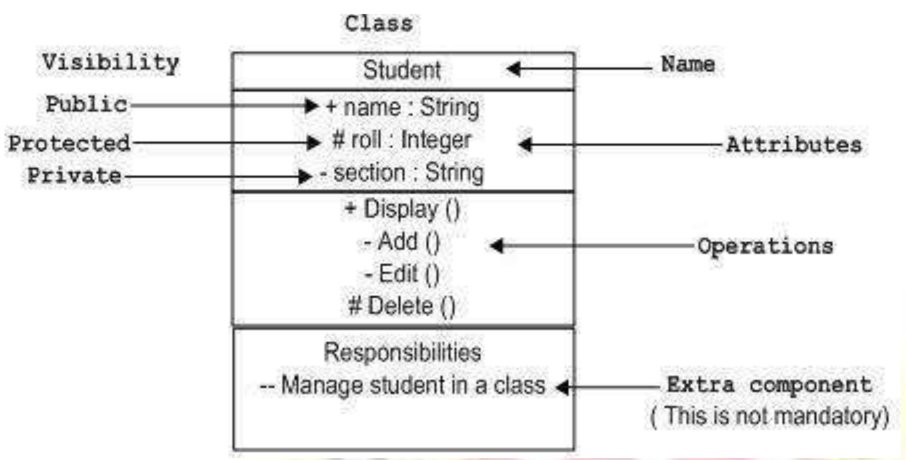
Structural Things: Graphical notations used in structural things are most widely used in UML. These are considered as the nouns of UML models. Following are the list of structural things.

- Classes
- Object
- Interface
- Collaboration
- Use case
- Active classes
- Components
- Nodes

Class Notation

UML *class* is represented by the following figure. The diagram is divided into four parts.

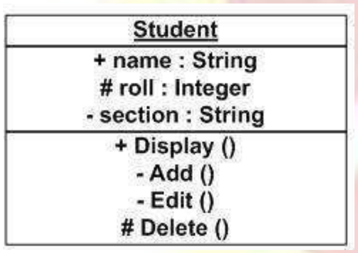
- The top section is used to name the class.
- The second one is used to show the attributes of the class.
- The third section is used to describe the operations performed by the class.
- The fourth section is optional to show any additional components.



Classes are used to represent objects. Objects can be anything having properties and responsibility.

Object Notation

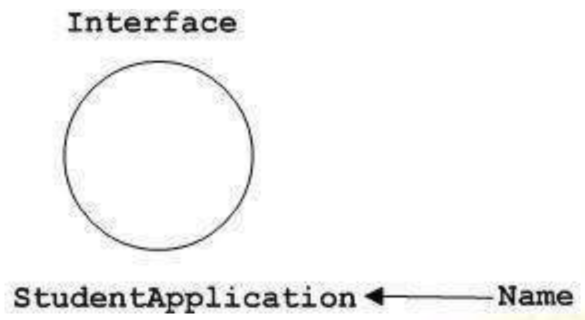
The object is represented in the same way as the class. The only difference is the name which is underlined as shown in the following figure.



As the object is an actual implementation of a class which is known as the instance of a class. Hence, it has the same usage as the class.

Interface Notation

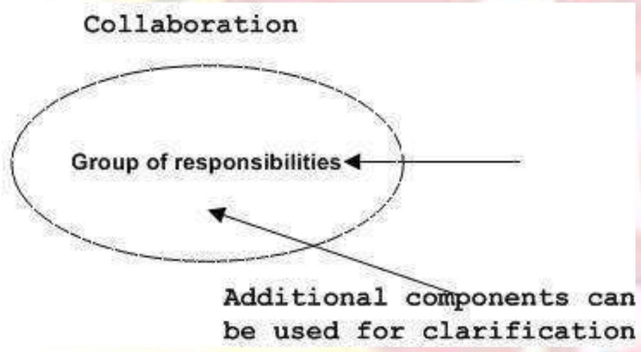
Interface is represented by a circle as shown in the following figure. It has a name which is generally written below the circle.



Interface is used to describe the functionality without implementation. Interface is just like a template to define different functions, not the implementation. When a class implements the interface, it also implements the functionality as per requirement.

Collaboration Notation

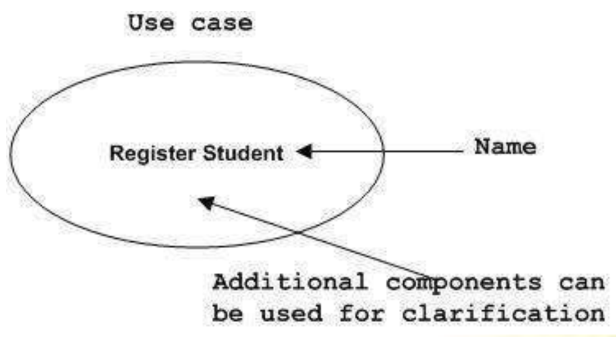
Collaboration is represented by a dotted eclipse as shown in the following figure. It has a name written inside the eclipse.


Collaboration
Group of responsibilities
Additional components can be used for clarification

Collaboration represents responsibilities. Generally, responsibilities are in a group.

Use Case Notation

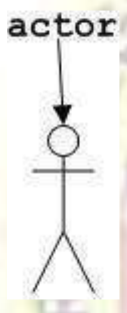
Use case is represented as an eclipse with a name inside it. It may contain additional responsibilities.



Use case is used to capture high level functionalities of a system.

Actor Notation

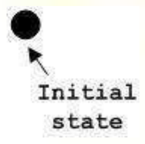
An actor can be defined as some internal or external entity that interacts with the system.



An actor is used in a use case diagram to describe the internal or external entities.

Initial State Notation

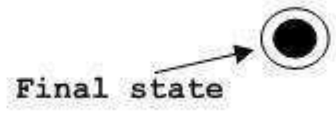
Initial state is defined to show the start of a process. This notation is used in almost all diagrams.



The usage of Initial State Notation is to show the starting point of a process.

Final State Notation

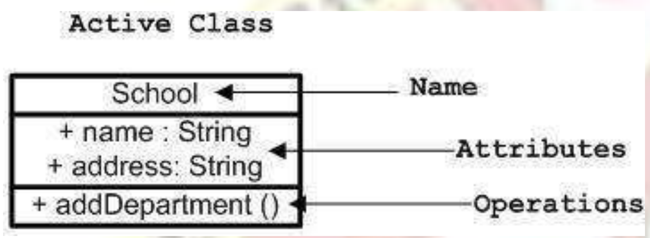
Final state is used to show the end of a process. This notation is also used in almost all diagrams to describe the end.



The usage of Final State Notation is to show the termination point of a process.

Active Class Notation

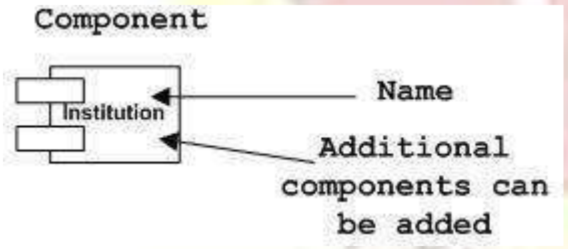
Active class looks similar to a class with a solid border. Active class is generally used to describe the concurrent behavior of a system.



Active class is used to represent the concurrency in a system.

Component Notation

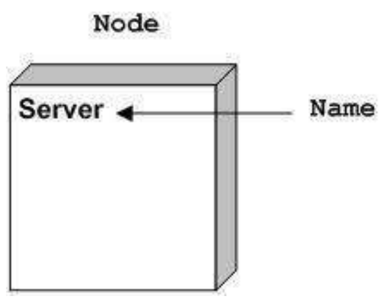
A component in UML is shown in the following figure with a name inside. Additional elements can be added wherever required.



Component is used to represent any part of a system for which UML diagrams are made.

Node Notation

A node in UML is represented by a square box as shown in the following figure with a name. A node represents the physical component of the system.



Node is used to represent the physical part of a system such as the server, network, etc.

BEHAVIORAL THINGS

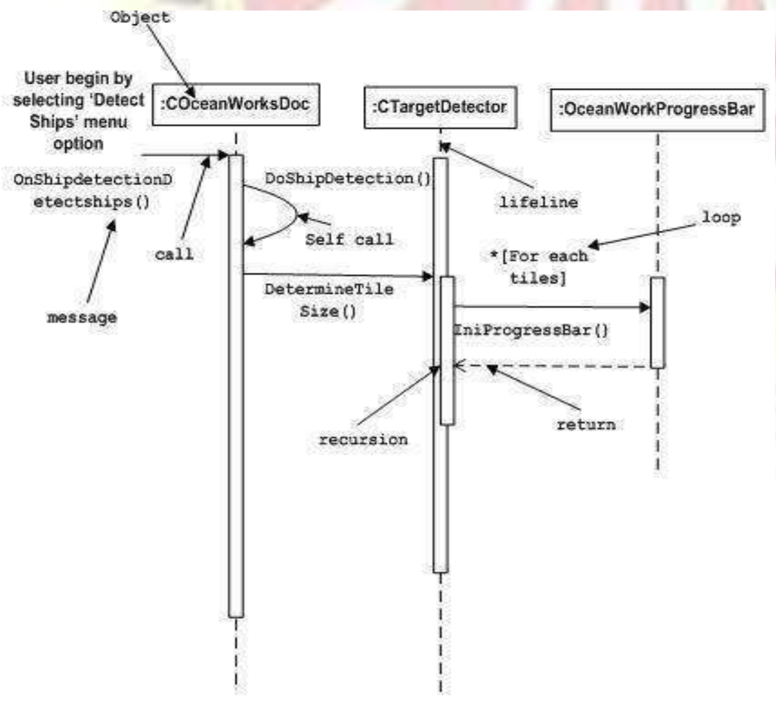
Dynamic parts are one of the most important elements in UML. UML has a set of powerful features to represent the dynamic part of software and non-software systems. These features include interactions and state machines.

Interactions can be of two types −

- Sequential (Represented by sequence diagram)
- Collaborative (Represented by collaboration diagram)

INTERACTION NOTATION

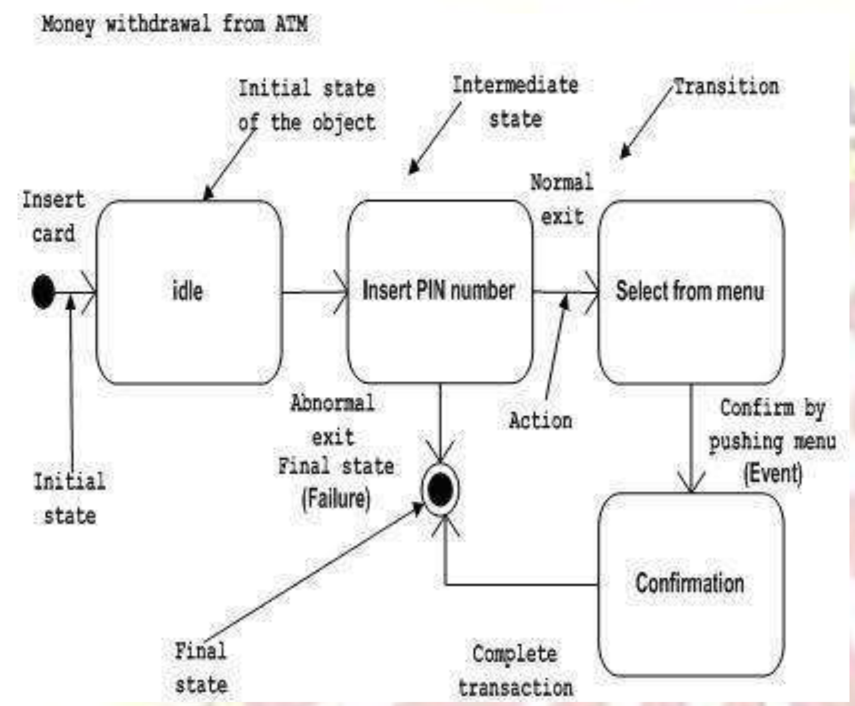
Interaction is basically a message exchange between two UML components. The following diagram represents different notations used in an interaction.



Interaction is used to represent the communication among the components of a system.

STATE MACHINE NOTATION

State machine describes the different states of a component in its life cycle. The notations are described in the following diagram.



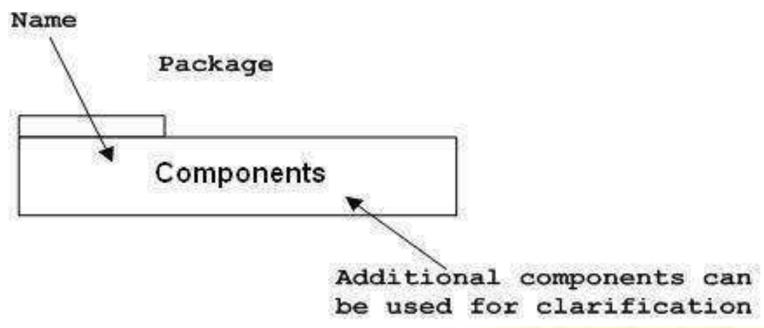
State machine is used to describe different states of a system component. The state can be active, idle, or any other depending upon the situation.

GROUPING THINGS

Organizing the UML models is one of the most important aspects of the design. In UML, there is only one element available for grouping and that is package.

Package Notation

Package notation is shown in the following figure and is used to wrap the components of a system.

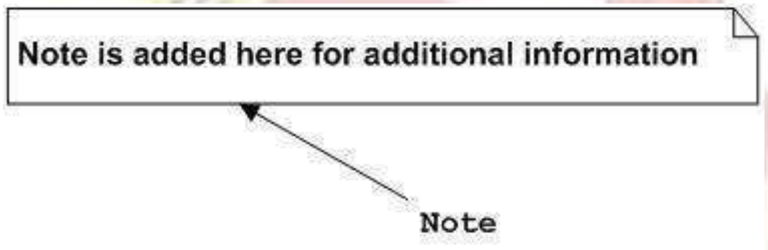


Annotational Things

In any diagram, explanation of different elements and their functionalities are very important. Hence, UML has *notes* notation to support this requirement.

Note Notation

This notation is shown in the following figure. These notations are used to provide necessary information of a system.



RELATIONSHIPS

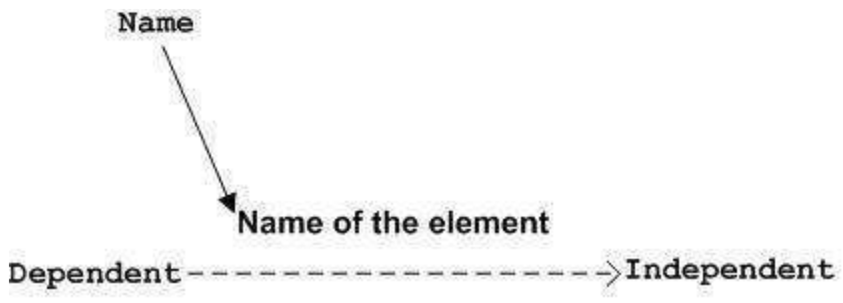
A model is not complete unless the relationships between elements are described properly. The *Relationship* gives a proper meaning to a UML model. Following are the different types of relationships available in UML.

- Dependency
- Association
- Generalization
- Extensibility

Dependency Notation

Dependency is an important aspect in UML elements. It describes the dependent elements and the direction of dependency.

Dependency is represented by a dotted arrow as shown in the following figure. The arrow head represents the independent element and the other end represents the dependent element.

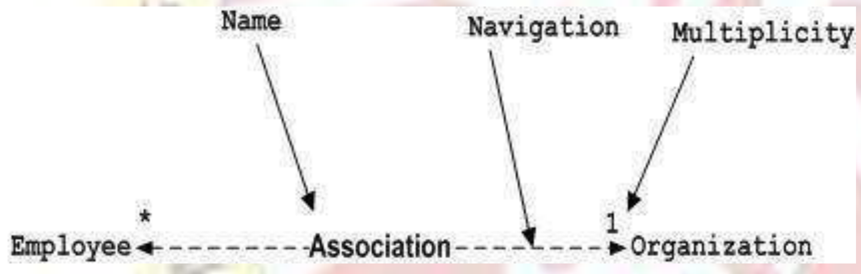


Dependency is used to represent the dependency between two elements of a system

Association Notation

Association describes how the elements in a UML diagram are associated. In simple words, it describes how many elements are taking part in an interaction.

Association is represented by a dotted line with (without) arrows on both sides. The two ends represent two associated elements as shown in the following figure. The multiplicity is also mentioned at the ends (1, *, etc.) to show how many objects are associated.

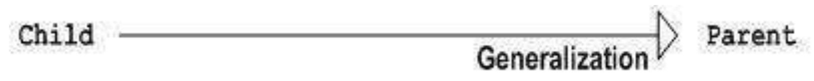


Association is used to represent the relationship between two elements of a system.

Generalization Notation

Generalization describes the inheritance relationship of the object-oriented world. It is a parent and child relationship.

Generalization is represented by an arrow with a hollow arrow head as shown in the following figure. One end represents the parent element and the other end represents the child element.

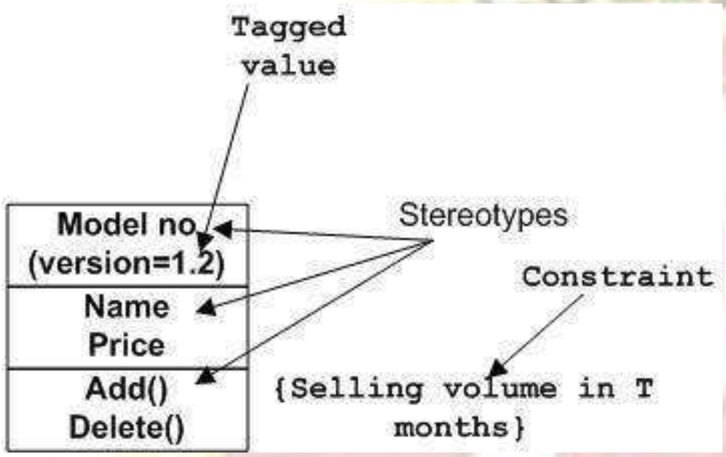


Generalization is used to describe parent-child relationship of two elements of a system.

Extensibility Notation

All the languages (programming or modeling) have some mechanism to extend its capabilities such as syntax, semantics, etc. UML also has the following mechanisms to provide extensibility features.

- Stereotypes (Represents new elements)
- Tagged values (Represents new attributes)
- Constraints (Represents the boundaries)



Extensibility notations are used to enhance the power of the language. It is basically additional elements used to represent some extra behavior of the system. These extra behaviors are not covered by the standard available notations.

HOW TO DRAW A CLASS DIAGRAM?

Class diagrams are the most popular UML diagrams used for construction of software applications. It is very important to learn the drawing procedure of class diagram.

Class diagrams have a lot of properties to consider while drawing but here the diagram will be considered from a top level view.

Class diagram is basically a graphical representation of the static view of the system and represents different aspects of the application. A collection of class diagrams represent the whole system.

The following points should be remembered while drawing a class diagram −

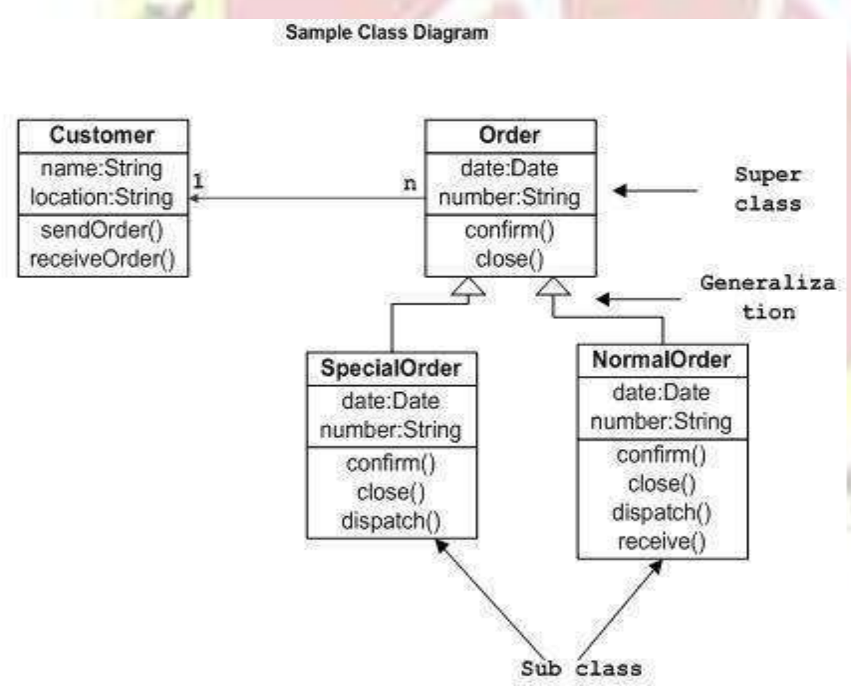
- The name of the class diagram should be meaningful to describe the aspect of the system.
- Each element and their relationships should be identified in advance.

- Responsibility (attributes and methods) of each class should be clearly identified
- For each class, minimum number of properties should be specified, as unnecessary properties will make the diagram complicated.
- Use notes whenever required to describe some aspect of the diagram. At the end of the drawing it should be understandable to the developer/coder.
- Finally, before making the final version, the diagram should be drawn on plain paper and reworked as many times as possible to make it correct.

The following diagram is an example of an Order System of an application. It describes a particular aspect of the entire application.

- First of all, Order and Customer are identified as the two elements of the system. They have a one-to-many relationship because a customer can have multiple orders.
- Order class is an abstract class and it has two concrete classes (inheritance relationship) SpecialOrder and NormalOrder.
- The two inherited classes have all the properties as the Order class. In addition, they have additional functions like dispatch () and receive ().

The following class diagram has been drawn considering all the points mentioned above.



WHERE TO USE CLASS DIAGRAMS?

Class diagram is a static diagram and it is used to model the static view of a system. The static view describes the vocabulary of the system.

Class diagram is also considered as the foundation for component and deployment diagrams. Class diagrams are not only used to visualize the static view of the system but they are also used to construct the executable code for forward and reverse engineering of any system.

Generally, UML diagrams are not directly mapped with any object-oriented programming languages but the class diagram is an exception.

Class diagram clearly shows the mapping with object-oriented languages such as Java, C++, etc. From practical experience, class diagram is generally used for construction purpose.

In a nutshell it can be said, class diagrams are used for

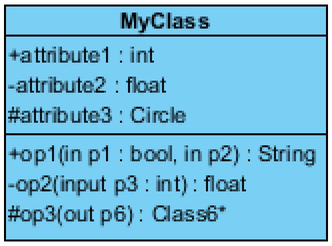
- Describing the static view of the system.
- Showing the collaboration among the elements of the static view.
- Describing the functionalities performed by the system.
- Construction of software applications using object oriented languages.

BENEFITS OF CLASS DIAGRAM

- Class Diagram illustrates data models for even very complex information systems
- It provides an overview of how the application is structured before studying the actual code. This can easily reduce the maintenance time
- It helps for better understanding of general schematics of an application.
- Allows drawing detailed charts which highlights code required to be programmed
- Helpful for developers and other stakeholders.

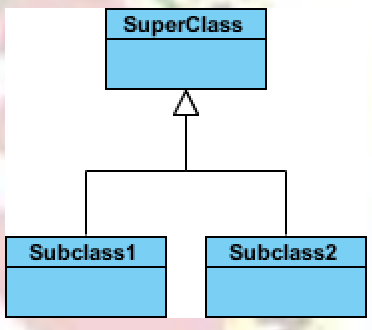
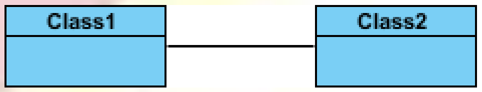
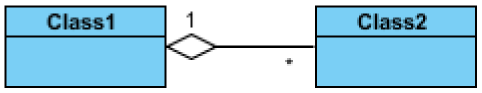
EVENTS

The events can be modeled in terms of UML. The event as change agents that have consequences and as information objects that represent information. To create object oriented structures that represent events in terms of attributes, associations, operations, state charts, and messages. An outline gives a run-time environment for the processing of events with multiple participants.



The graphical representation of the class - MyClass as shown above:

- MyClass has 3 attributes and 3 operations
- Parameter p3 of op2 is of type int
- op2 returns a float
- op3 returns a pointer (denoted by a *) to Class6

Relationship Type	**Graphical Representation**
Inheritance (or Generalization): • Represents an "is-a" relationship. • An abstract class name is shown in italics. • SubClass1 and SubClass2 are specializations of Super Class. • A solid line with a hollow arrowhead that point from the child to the parent class	
Simple Association: • A structural link between two peer classes. • There is an association between Class1 and Class2 • A solid line connecting two classes	
Aggregation: A special type of association. It represents a "part of" relationship. • Class2 is part of Class1. • Many instances (denoted by the *) of Class2 can be	

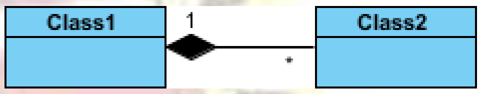
associated with Class1.

- Objects of Class1 and Class2 have separate lifetimes.
- A solid line with an unfilled diamond at the association end connected to the class of composite

Composition:

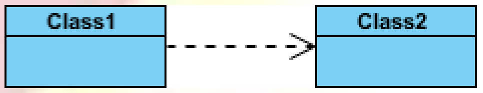
A special type of aggregation where parts are destroyed when the whole is destroyed.

- Objects of Class2 live and die with Class1.
- Class2 cannot stand by itself.
- A solid line with a filled diamond at the association connected to the class of composite



Dependency:

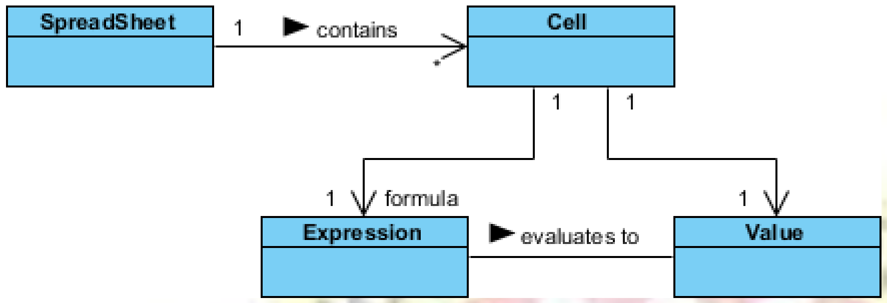
- Exists between two classes if the changes to the definition of one may cause changes to the other (but not the other way around).
- Class1 depends on Class2
- A dashed line with an open arrow



RELATIONSHIP NAMES

- Names of relationships are written in the middle of the association line.
- Good relation names make sense when the person read them out loud:
    - "Every spreadsheet **contains** some number of cells",
    - "an expression **evaluates to** a value"

- They often have a **small arrowhead to show the direction** in which direction to read the relationship, e.g., expressions evaluate to values, but values do not evaluate to expressions.



Relationship - Roles

- A role is a directional purpose of an association.
- Roles are written at the ends of an association line and describe the purpose played by that class in the relationship.
  - E.g., A cell is related to an expression. The nature of the relationship is that the expression is the **formula** of the cell.

NAVIGABILITY

The arrows indicate whether, given one instance participating in a relationship, it is possible to determine the instances of the other class that are related to it.

The diagram above suggests that,

- Given a spreadsheet, we can locate all of the cells that it contains, but that
  we cannot determine from a cell in what spreadsheet it is contained.
- Given a cell, we can obtain the related expression and value, but
  given a value (or expression) we cannot find the cell of which those are attributes.

VISIBILITY OF CLASS ATTRIBUTES AND OPERATIONS

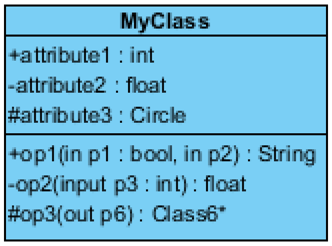
In object-oriented design, there is a notation of visibility for attributes and operations. UML identifies four types of visibility: **public**, **protected**, **private**, and **package**.

The +, -, # and ~ symbols before an attribute and operation name in a class denote the visibility of the attribute and operation.

- + denotes public attributes or operations
- - denotes private attributes or operations
- # denotes protected attributes or operations

- ~ denotes package attributes or operations

Class Visibility Example



In the example above:

- attribute1 and op1 of MyClassName are public
- attribute3 and op3 are protected.
- attribute2 and op2 are private.

Access for each of these visibility types is shown below for members of different classes.

Access Right	public (+)	private (-)	protected (#)	Package (~)
Members of the same class	yes	yes	yes	yes
Members of derived classes	yes	no	yes	yes
Members of any other class	yes	no	no	in same package

MULTIPLICITY

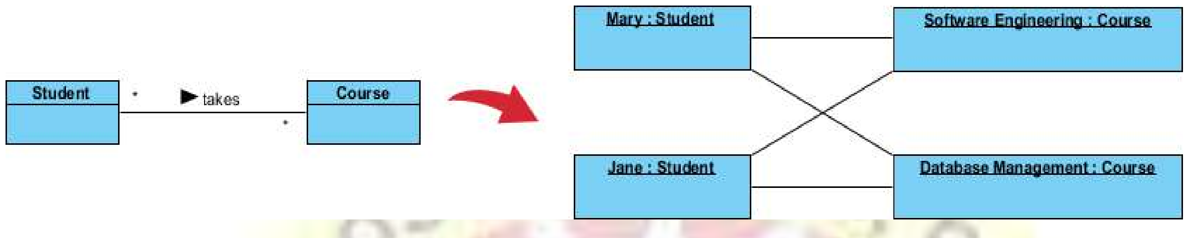
How many objects of each class take part in the relationships and multiplicity can be expressed as:

- Exactly one - 1
- Zero or one - 0..1
- Many - 0..* or *
- One or more - 1..*
- Exact Number - e.g. 3..4 or 6
- Or a complex relationship - e.g. 0..1, 3..4, 6.* would mean any number of objects other than 2 or 5

Multiplicity Example

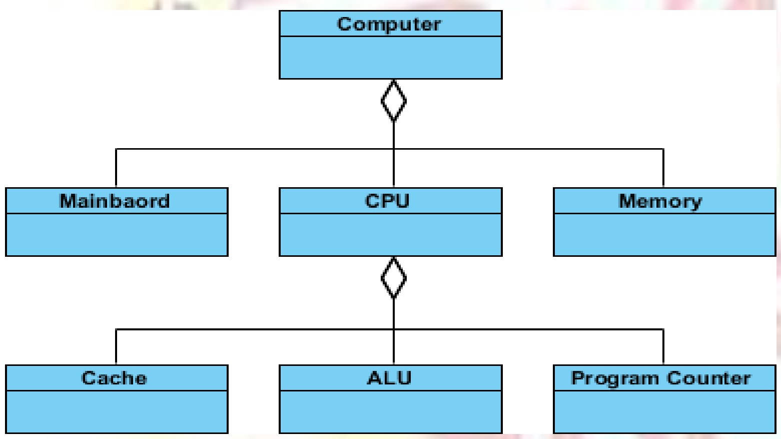
- Requirement: A Student can take many Courses and many Students can be enrolled in one Course.

- In the example below, the **class diagram** (on the left), describes the statement of the requirement above for the static model while the object diagram (on the right) shows the snapshot (an instance of the class diagram) of the course enrollment for the courses Software Engineering and Database Management respectively)



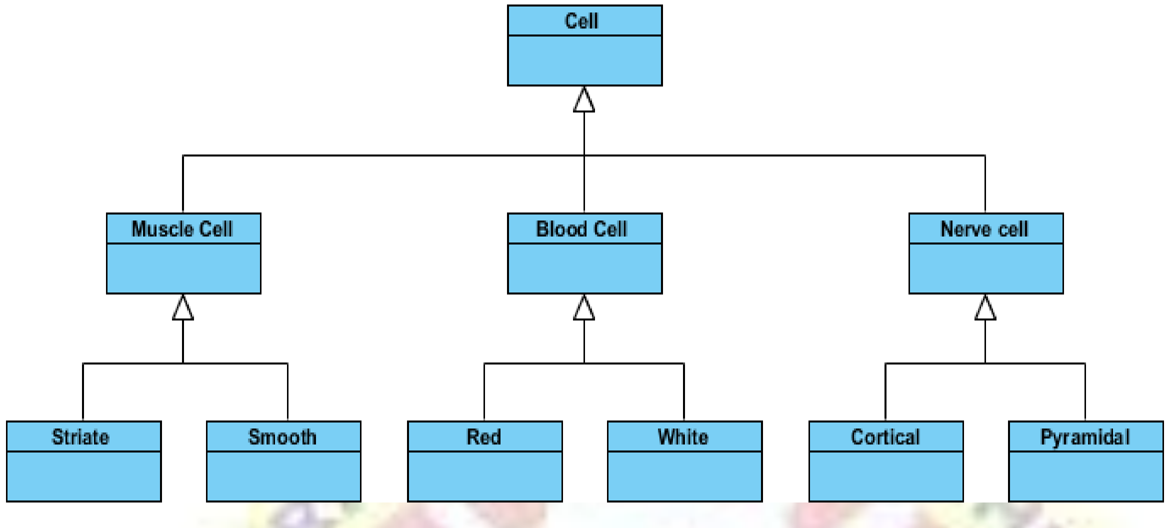
Aggregation Example - Computer and parts

- An aggregation is a special case of association denoting a "consists-of" hierarchy
- The aggregate is the parent class, the components are the children classes



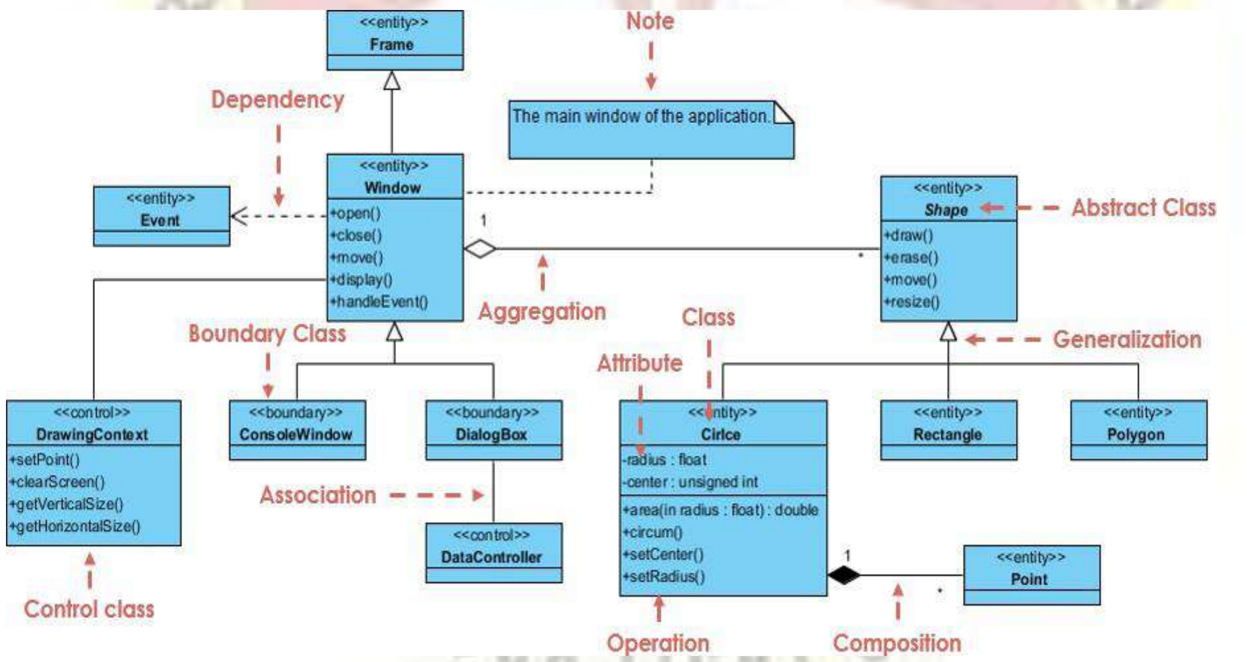
Inheritance Example - Cell Taxonomy

- Inheritance is another special case of an association denoting a "kind-of" hierarchy
- Inheritance simplifies the analysis model by introducing a taxonomy
- The child classes inherit the attributes and operations of the parent class.



Class Diagram - Diagram Tool Example

A class diagram may also have notes attached to classes or relationships. Notes are shown in closed doted lines.



In the example above:

We can interpret the meaning of the above class diagram by reading through the points as following.

1. Shape is an abstract class. It is shown in Italics.
2. Shape is a superclass. Circle, Rectangle and Polygon are derived from Shape. In other words, a Circle is-a Shape. This is a generalization / inheritance relationship.

3. There is an association between DialogBox and DataController.
4. Shape is part-of Window. This is an aggregation relationship. Shape can exist without Window.
5. Point is part-of Circle. This is a composition relationship. Point cannot exist without a Circle.
6. Window is dependent on Event. However, Event is not dependent on Window.
7. The attributes of Circle are radius and center. This is an entity class.
8. The method names of Circle are area(), circum(), setCenter() and setRadius().
9. The parameter radius in Circle is an in parameter of type float.
10. The method area() of class Circle returns a value of type double.
11. The attributes and method names of Rectangle are hidden. Some other classes in the diagram also have their attributes and method names hidden.

NORMALIZATION

Normalization should be part of the database design process. However, it is difficult to separate the normalization process from the ER modelling process so the two techniques should be used concurrently.

Use an Entity Relation Diagram (ERD) to provide the big picture, or macro view, of an organization's data requirements and operations. This is created through an iterative process that involves identifying relevant entities, their attributes and their relationships.

Normalization procedure focuses on characteristics of specific entities and represents the micro view of entities within the ERD.

Normalization is the branch of relational theory that provides design insights. It is the process of determining how much redundancy exists in a table. The goals of normalization are to:

- Be able to characterize the level of redundancy in a relational schema
- Provide mechanisms for transforming schemas in order to remove redundancy

Normalization theory draws heavily on the theory of functional dependencies. Normalization theory defines six normal forms (NF). Each normal form involves a set of dependency properties that a schema must satisfy and each normal form gives guarantees about the presence and/or absence of update anomalies. This means that higher normal forms have less redundancy, and as a result, fewer update problems.

- Normalization is the process of organizing the data in the database.
- Normalization is used to minimize the redundancy from a relation or set of relations. It is also used to eliminate the undesirable characteristics like Insertion, Update and Deletion Anomalies.
- Normalization divides the larger table into the smaller table and links them using relationship.
- The normal form is used to reduce redundancy from the database table.

NORMAL FORMS

All the tables in any database can be in one of the normal forms. Ideally we only want minimal redundancy for PK to FK. Everything else should be derived from other tables. There are six normal forms, but we will only look at the first three, which are:

- First normal form (1NF)
- Second normal form (2NF)
- Third normal form (3NF)

Normal Form	**Description**
1NF	A relation is in 1NF if it contains an atomic value.
2NF	A relation will be in 2NF if it is in 1NF and all non-key attributes are fully functional dependent on the primary key.
3NF	A relation will be in 3NF if it is in 2NF and no transition dependency exists.

First Normal Form (1NF)

In the *first normal form*, only single values are permitted at the intersection of each row and column; hence, there are no repeating groups.

As per the rule of first normal form, an attribute (column) of a table cannot hold multiple values. It should hold only atomic values.

Example: Suppose a company wants to store the names and contact details of its employees. It creates a table that looks like this:

emp_id	emp_name	emp_address	emp_mobile
101	Herschel	New Delhi	8912312390

102	Jon	Kanpur	8812121212 9900012222
103	Ron	Chennai	7778881212
104	Lester	Bangalore	9990000123 8123450987

Two employees (Jon & Lester) are having two mobile numbers so the company stored them in the same field.

This table is **not in 1NF** as the rule says “each attribute of a table must have atomic (single) values”, the emp_mobile values for employees Jon & Lester violates that rule.
To make the table complies with 1NF we should have the data like this:

emp_id	emp_name	emp_address	emp_mobile
101	Herschel	New Delhi	8912312390
102	Jon	Kanpur	8812121212
102	Jon	Kanpur	9900012222
103	Ron	Chennai	7778881212
104	Lester	Bangalore	9990000123
104	Lester	Bangalore	8123450987

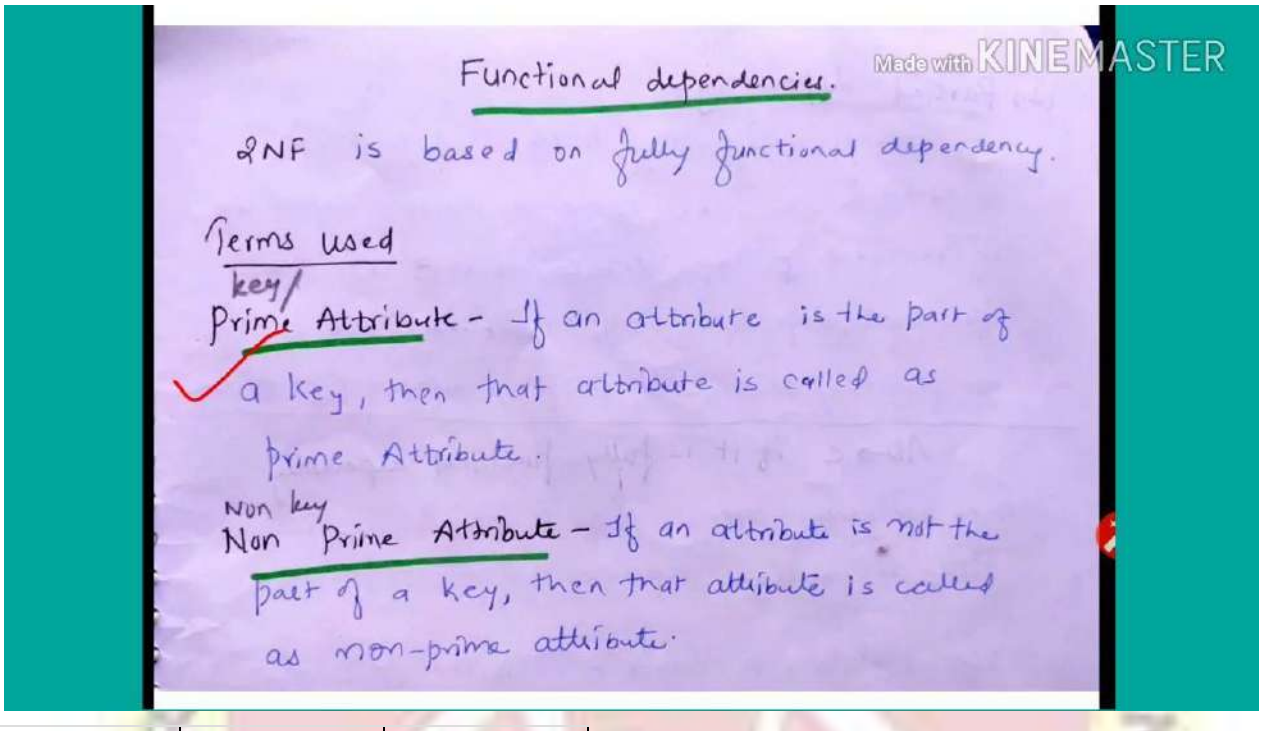
Second Normal Form (2NF)

A table is said to be in 2NF if both the following conditions hold:

- Table is in 1NF (First normal form)
- No non-prime attribute is dependent on the proper subset of any candidate key of table.

An attribute that is not part of any candidate key is known as non-prime attribute.

Example: Suppose a school wants to store the data of teachers and the subjects they teach. They create a table that looks like this: Since a teacher can teach more than one subjects, the table can have multiple rows for a same teacher.


Functional dependencies.

2NF is based on fully functional dependency.

Terms used

key/
Prime Attribute - If an attribute is the part of a key, then that attribute is called as prime Attribute.

Non key
Non Prime Attribute - If an attribute is not the part of a key, then that attribute is called as non-prime attribute.

Made with KINEMASTER

teacher_id	subject	teacher_age
111	Maths	38
111	Physics	38
222	Biology	38
333	Physics	40
333	Chemistry	40

Candidate Keys: {teacher_id, subject}

Non prime attribute: teacher_age

The table is in 1 NF because each attribute has atomic values. However, it is not in 2NF because non prime attribute teacher_age is dependent on teacher_id alone which is a proper subset of candidate key. This violates the rule for 2NF as the rule says "**no** non-prime attribute is dependent on the proper subset of any candidate key of the table".

To make the table complies with 2NF we can break it in two tables like this:

teacher_details table:

teacher_id	teacher_age
111	38
222	38
333	40

teacher_subject table:

teacher_id	subject
111	Maths
111	Physics
222	Biology
333	Physics
333	Chemistry

Now the tables comply with Second normal form (2NF).

Third Normal Form (3NF)

A table design is said to be in 3NF if both the following conditions hold:

- Table must be in 2NF
- Transitive functional dependency of non-prime attribute on any super key should be removed.

An attribute that is not part of any candidate key is known as non-prime attribute.

In other words 3NF can be explained like this: A table is in 3NF if it is in 2NF and for each functional dependency X-> Y at least one of the following conditions hold:

- X is a super key of table
- Y is a prime attribute of table

An attribute that is a part of one of the candidate keys is known as prime attribute.

Example: Suppose a company wants to store the complete address of each employee, they create a table named employee_details that looks like this:

emp_id	emp_name	emp_zip	emp_state	emp_city	emp_district
1001	John	282005	UP	Agra	Dayal Bagh
1002	Ajeet	222008	TN	Chennai	M-City
1006	Lora	282007	TN	Chennai	Urrapakkam
1101	Lilly	292008	UK	Pauri	Bhagwan
1201	Steve	222999	MP	Gwalior	Ratan

Super keys: {emp_id}, {emp_id, emp_name}, {emp_id, emp_name, emp_zip}…so on

Candidate Keys: {emp_id}

Non-prime attributes: all attributes except emp_id are non-prime as they are not part of any candidate keys.

Here, emp_state, emp_city & emp_district dependent on emp_zip. And, emp_zip is dependent on emp_id that makes non-prime attributes (emp_state, emp_city & emp_district) transitively dependent on super key (emp_id). This violates the rule of 3NF.

To make this table complies with 3NF we have to break the table into two tables to remove the transitive dependency:

employee table:

emp_id	emp_name	emp_zip
1001	John	282005

1002	Ajeet	222008
1006	Lora	282007
1101	Lilly	292008
1201	Steve	222999

employee_zip table:

emp_zip	emp_state	emp_city	emp_district
282005	UP	Agra	Dayal Bagh
222008	TN	Chennai	M-City
282007	TN	Chennai	Urrapakkam
292008	UK	Pauri	Bhagwan
222999	MP	Gwalior	Ratan

UNIT IV

INTRODUCTION TO ORACLE SQL

Structured Query Language (SQL) is the set of statements with which all programs and users access data in an Oracle database. Application programs and Oracle tools often allow users access to the database without using SQL directly, but these applications in turn must use SQL when executing the user's request. Oracle SQL is a superset of the American National Standards Institute (ANSI) and the International Organization for Standardization (ISO) SQL:1999 standard.

SQL Developer is the graphical user interface (GUI) tool that Oracle supplies to query the database, explore objects, run reports, and run scripts. It runs on Windows, Linux, and Mac OSX. It can be used to access Oracle databases 9*i*, 10*g*, 11*g* and 12*c*, as well as other databases such as Times Ten, Microsoft Access, MySQL, and SQL Server.

An Oracle database is a collection of data treated as a unit. The purpose of a database is to store and retrieve related information. A database server is the key to solving the problems of information management. In general, a server reliably manages a large amount of data in a multiuser environment so that many users can concurrently access the same data. All this is accomplished while delivering high performance. A database server also prevents unauthorized access and provides efficient solutions for failure recovery.

Oracle Database is the first database designed for enterprise grid computing, the most flexible and cost effective way to manage information and applications. Enterprise grid computing creates large pools of industry-standard, modular storage and servers. With this architecture, each new system can be rapidly provisioned from the pool of components. There is no need for peak workloads, because capacity can be easily added or reallocated from the resource pools as needed.

The database has **logical structures** and **physical structures**. Because the physical and logical structures are separate, the physical storage of data can be managed without affecting the access to logical storage structures.

Oracle provides a number of utilities to facilitate the SQL development process:

- Oracle SQL Developer is a graphical tool that lets you browse, create, edit, and delete (drop) database objects, edit and debug PL/SQL code, run SQL statements and scripts, manipulate and export data, and create and view reports. With SQL Developer, anyone can connect to any target Oracle Database schema using standard Oracle Database authentication.

- Once connected, the person can perform operations on objects in the database. The person can also connect to schemas for selected third-party (non-Oracle) databases, such as MySQL, Microsoft SQL Server, and Microsoft Access, view metadata and data in these databases, and migrate these databases to Oracle.

- SQL*Plus is an interactive and batch query tool that is installed with every Oracle Database server or client installation. It has a command-line user interface and a Web-based user interface called *i*SQL*Plus.

- Oracle JDeveloper is a multiple-platform integrated development environment supporting the complete lifecycle of development for Java, Web services, and SQL.

- It provides a graphical interface for executing and tuning SQL statements and a visual schema diagrammer (database modeler). It also supports editing, compiling, and debugging PL/SQL applications.

- Oracle Application Express is a hosted environment for developing and deploying database-related Web applications. SQL Workshop is a component of Oracle Application Express that lets view and manages database objects from a Web browser. SQL Workshop offers quick access to a SQL command processor and a SQL script repository.

SQL *PLUS ENVIRONMENT

SQL Queries are sent to the Oracle RDBMS using the tool called SQL *Plus. This is the principal CLIENT tool for ORACLE. It is an environment through which any interaction with the database is done using SQL commands.

SQL * Plus program can be used in conjunction with the SQL database language and its procedural language extension PL/SQL. SQL * Plus enables the user to manipulate SQL commands and PL/SQL statements and to perform additional tasks such as to

- Enter, edit, store, retrieve and run SQL commands
- Format and print calculations, query results in the form of reports.
- List column definitions for any table
- Access and copy data between SQL databases.

STRUCTURED QUERY LANGUAGE (SQL)

SQL is a Structured Query Language and is the industry standard language to define and manipulate the data in Relational Database Management" System. In a database environment, the interactions between the Client and the Server are only through SQL. This one-point communication language in Client-Server architecture facilitates the data base, connectivity and processing.

Structured Query Language is a simple English-like language SQL is also pronounced as sequel and consists of layers of increasing complexity and capability. End-users with little or no experience in data processing can learn SQL features very quickly. It is a Fourth Generation Language.

SQL was first introduced by IBM Research and was introduced into the commercial market first by Oracle Corporation in 1979. A committee at the American National, Standards Institute has endorsed SQL as the standard language for RDBMS.

SQL provides the following functionalities:

- Creation of tables.
- Querying the exact data.

- Change the data structure and the data.
- Combine and calculate the data to get required information.

NON PROCEDURAL LANGUAGE

SQL is a non-procedural language and is free of logic and procedural constructs. In SQL, all we need to say is what we want and not how to go about it. It access not require, the user to specify the methodology for accessing the data. SQL processes of records rather than one at a time. SQL language can be used by Database Administrators application programmers decision support personnel and management.

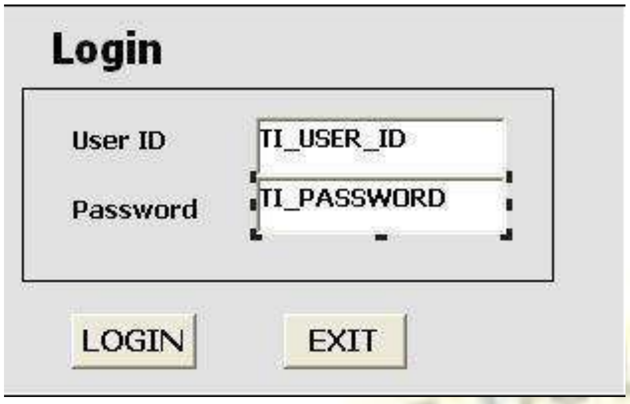
SQL facilitates interaction by embeddingSQL Standard programming languages such as COBOL, FORTRAN, C etc. through a variety of RDBMS tools like SQL * Plus, Report Generators, Duplication Generators, Form Generators.

DATABASE ACCESS THROUGH SQL

SOL operates over database tables. Tables constitute tabular representation of data with data residing in the form of a spreadsheet or rows and columns. Each row has a set of data items and the kerns are called fields.

LOGGING INTO SQL *PLUS

To enter into Oracle and interact with the database using SQL *Plus, a user name and a password must be given. Logging to oracle can be done by using either the menu option or by entering Plus 80w (From Oracle 8) in the Start - Run Option.



The figure prompts the user to enter a username and a password. If the system is connected to multi-user environment, the database s name must be provided in the "Host String".

SHORTCUTS TO STARTING SQL *PLUS

While starting SQL *Plus, the username along with the password and the database (if required) can be given. The username and the password must be separated by a slash (/). For example consider a user called SCOTT and the password called TIGER. If the user is connected to the personal database, SQL *Plus can be started by giving.

PLUS SQL SCOTT/TIGER (or) SQL PLUS SCOTT / TIGER

If the user is connected to a network, SQL *Plus can be started by giving PLUS SOW SCOTT/TIGER @ ORACLE After this, the SQL prompt appears from where the commands can be entered and executed.

SQL BUFFER

The area where SQL * Plus stores the most recently typed SQL commands or PL/SQL commands is called SQL Buffer. The command remains in the buffer until another command is entered. Thus, if the same command or block has to be re executed or edited, it can be done without retyping the same.

Note : SQL *Plus commands are not stored in the SQL Buffer and hence may not be rerun.

FEATURES OF SQL

The following are some of the features of SQL:

- Easy to learn.
- Flexible language.
- Individual statements are used to make simple queries.
- Portable language.
- Not only a query language, but also used to create database tables, insert, delete and granting access to users and many more.
- Independent for the internal structure. Same result will be returned whether any indexes have been done or not.

SQL *PLUS COMMANDS

Based on the type of action that each command performs, SQL commands can be broadly classified as follows:

Classifications	Description	Commands
DDL (Date Definition Language)	Is used to define the structure of a table, or modify the structure Is used to manipulate with the data	CREATE, ALTER DROP, TRUNCATE, RENAME
DML (Data Manipulation Language)	Is used to restrict or grant access to tables	INSERT, UPDATE, DELETE
DCL (Data Control Language)	Is used to restrict or grant access to tables	GRANT, REVOKE
TCL (Transacton Control Language)	Is used to complete fully or undo the transactions	COMMIT, SAVEPOINT, ROLLBACK

Queries	Is used to select records from the tables or other objects	SELECT

SQL*PLUS ERROR MESSAGES

SQL keyword errors occur when one of the words that the SQL query language reserves for its commands and clauses is misspelled. For example, writing “UPDTE” instead of “UPDATE” will produce SQL keyword error

SP2-0002 ACCEPT statement must specify a variable name

Cause: Required variable name was missing after the ACCEPT command.

Action: Re-enter the ACCEPT command with a variable argument to store the input value.

SP2-0004 Nothing to append

Cause: There was no specified text entered after the APPEND command.

Action: Re-enter the APPEND command with the specified text.

SP2-0499 Misplaced APPEND keyword

Cause: The APPEND keyword was in the wrong position in the COPY command.

Action: Check the syntax of the COPY command for the correct options.

SP2-0501 Error in SELECT statement: *Oracle_database_error_message*

Cause: Invalid SELECT statement found in the COPY command.

Action: Check the syntax of the COPY command for the correct options.

SP2-0513 Misplaced CREATE keyword

Cause: The CREATE keyword was in the wrong position in the COPY command.

Action: Check the syntax of the COPY command for the correct options.

SP2-0514 Misplaced REPLACE keyword

Cause: The REPLACE keyword was in the wrong position in the COPY command.

Action: Check the syntax of the COPY command for the correct options.

SP2-0515 Maximum number of columns (*max_num_columns*) exceeded

Cause: The maximum number of columns was exceeded in the COPY command.

Action: Reduce the number of columns and try again.

SP2-0516 Invalid *command_name* name NULL encountered

Cause: An invalid or null column name was specified in either the COLUMN or the ATTRIBUTE command.

Action: Retry the operation with a valid column name.

SP2-0517 Missing comma or right parenthesis

Cause: A missing right parenthesis was identified in the COPY command.

Action: Retry the operation with a comma or right parenthesis.

SP2-0518 Missing USING clause

Cause: USING keyword is missing in the USING clause of the COPY command.

Action: Specify the USING keyword before the USING clause of the COPY command.

SP2-0519 FROM string missing Oracle Net @database specification

Cause: Missing connect string for the database that contains the data to be copied from in the COPY command.

Action: Include a FROM clause to specify a source database other than the default.

SP2-0645 Operating System error occurred
Unable to complete EDIT command

Cause: An operating system error occurred with the EDIT command.

Action: Check that the file was created successfully, and verify that the device you are writing to is still available.

SP2-0650 New passwords do not match

Cause: The new passwords entered did not match.

Action: Re-issue the PASSWORD command and make sure that the new passwords are entered correctly.

SP2-0659 Password unchanged

Cause: The PASSWORD command failed to change passwords because:

- No passwords were given.
- The new passwords did not match.

Action: Re-issue the PASSWORD command and make sure that the new passwords are entered correctly.

ORACLE TABLES - DATA DEFINITION LANGUAGE (DDL)

Data Definition Language (DDL) actually consists of the SQL commands that can be used to define the database schema. It simply deals with descriptions of the database schema and is used to create and modify the structure of database objects in the database.

Examples of DDL commands:

- CREATE – is used to create the database or its objects (like table, index, function, views, store procedure and triggers).
- ALTER-is used to alter the structure of the database.
- DROP – is used to delete objects from the database.
- RENAME –is used to rename an object existing in the database.
- TRUNCATE–is used to remove all records from a table, including all spaces allocated for the records are removed.
- COMMENT –is used to add comments to the data dictionary.

Data Creation through SQL section deals with creation of tables, altering its structure, inserting and retrieving records and querying complex data using SQL

NAMING RULES AND CONVENTIONS

Naming conventions are an important part of coding conventions. Naming conventions are the rules for how “things” are named. In the case of a database, the “things” being named are schema objects such as tables, sequences, indexes, views, triggers as well as constraints. In a database it is essential to choose names with particular care.

If one thinks of a database with several data providers and consumers (usually including applications and interfaces to and from other databases), it is easy to imagine that objects cannot be easily be renamed. They should be given suitable names from the very beginning.

Compared to the naming of classes in object-oriented programming languages, the database developer also has to struggle with the restricted hierarchy of namespaces. While Oracle is only familiar with the schema for all object types and also the packages for PL/SQL code, in Java a freely-definable hierarchy of Java packages can be generated to which classes can be added.

The length of an identifier may also be restricted. Fortunately, in Oracle 12cR2, the 30-byte restriction for an identifier has been increased to 128 bytes.

CHOICE OF SUITABLE NAMES

The domain knowledge should always be taken into account when choosing names. As a rule, the domain knowledge is reflected in the name of the relation (table name). Not least due of the length restriction for identifiers, it is recommended to introduce a human readable name abbreviation (i.e. a mnemonic) for each table. Triggers, indexes, sequences and constraints are, for example, assigned to tables. The way in which they are named should follow certain rules. The affiliation of such an object to a table should be obvious; as well as the object type and ultimately the technical execution.

Example:

All employees are included in the employee table (domain knowledge). The employee table is given the abbreviation emp. There is a sequence named emp_seq. There is a trigger named emp_seq_tg.

Using our naming convention, we now know the following:

- The emp_seq sequence and the emp_seq_tg trigger belong to the employee table, since they bear the abbreviation emp.
- Because the sequence bears a name in accordance with our convention:
- The employee.id technical key column is populated by the sequence.
- The sequence starts with 1 and ends with 10^{18}-1, it as well as employee.id, will fit in a 64-bit signed integer.
- The sequence does not repeat.
- Since the trigger also bears a name in accordance with our convention, we know that
- This trigger populates the employee.id key column.
- The trigger does nothing else. (Which is good!)

There is already a lot of implicit knowledge being conveyed by the naming conventions. We have also incorporated a few “best practices” into our naming conventions such as largely eliminating triggers and the data types for sequences and id columns.

DATA TYPES

Each literal or column value manipulated by Oracle has a data type. A value's data type associates a fixed set of properties with the Value. Broadly classifying the data types, they can be of two types

- BUILT-IN
- USER — DEFINED

Built-in data types are predefined set of data types set in Oracle. Based on the type of data that can be stored, built-in data types pan be classified as

- Character Data types
- Numeric Data type
- Date Data type
- Raw Data type
- Long Raw Data type
- Lob Data type

CHARACTER DATA TYPE

Char (n)

Char datatype is a fixed length character data of length n bytes.

Default size is 1 byte and it can hold a maximum of 2000 bytes. Character data types had blank spaces to the fixed length if the user enters a value lesser than the specified length.

Syntax

Char (n)

Example :

X char (4) stores upto 4 characters of data in the column X.

Varchar 2 (size)

Varchar 2 datatype are variable length character strings. They can store alpha-numeric values and the size must be specified. The maximum length of varchar 2 datatype is 4000 bytes. Unlike char datatype, blank spaces are not padded to the length of the string. So, this is more preferred than &erecter datatypes since it does not store the maximum length.

Syntax

Varchar 2 (size)

Example :

X varchar2 (10) stores upto 10 characters of data in the column X.

NUMERIC DATA TYPES (NUMBER)

The number data types can store numeric values where p stands for the precision and stands for the scale. The precision can range between 1 to 38 and the scale ranges from - 84 to 127.

Syntax

Number (p, s)

Example :

Sal number — Here the scale is 0 and the precision is 38.

Sal number(7) — Here the scale is. 0 and the number is a fixed point number of 7 digits

Sal number (7,3) — Stores 5 digits followed by 2 decimal points.

DATE DATA TYPE

Date datatype is used to store date and time values. The default format is DD-MON-YY. The valid data for a data ranges from January 1,4712 BC to December 31,4712 AD. Date Data type stores 7 bytes one each for century, year, month, day, hour, minute and second.

RAW DATA TYPE

RAW (n)
RAW datatype stores binary data of length n bytes. The maximum size is 255 bytes. Specifying the size is a must for this datatype.
Syntax
Raw (n)
LONG Datatype
Stores character data of variable length upto 2 Gigabytes (GB) or 231-1

LONG RAW DATA TYPE

Long Raw Data type stores into 2 Gigabytes (GB) of raw binary data. The use of long **v**alues are restricted. The restrictions are:

- A Table cannot contain more than one LONG column.
- LONG columns cannot appear in Integrity constraints
- They cannot appear in WHERE, ORDER BY clauses of SELECT statements
- Cannot be a part of expressions or **conditions.**
- Cannot appear in the SELECT list of CREATE TABLE as SELECT.

LOB DATA TYPES

In addition to the above data types, Oracle8 supports LOB data types. LOB is the acronym for LARGE OBJECTS. The LOB data types stores up to 4 GB of data. This data type is used for storing video clippings, large images, history documents etc.

CONSTRAINTS

Data security and Data Integrity are the most important factors in deciding the success of a system. Constraints are a mechanism, used by Oracle to restrict invalid data from being entered into the table and thereby maintain the integrity of` the data. They are otherwise called a Business Rules. These constraints can be broadly classified into 3 types:

- Entity Integrity Constraints
- Domain Integrity Constraints
- Referential Integrity Constraint's

ENTITY INTEGRITY CONSTRAINT

Entity Integrity constraints can be classified as

- PRIMARY KEY
- UNIQUE KEY

Choosing a table‘s **Primary Key**

A primary key allows each row in a table to be uniquely identified and ensures that the duplicate rows exist and no null values are entered. Selecting a primary key needs the following guideline:

- Choose a column whose data values are unique
- Choose a column whose data values never change.

A primary key value is used to identify a row in the table. Therefore, primary key values must not contain any data that is used for any other purpose. Primary key can contain one to more columns of the same table. Together they form a composite Primary Key.

Using **Unique Key**

Unique Key constraint is used to prevent the duplication of key values within the tows of a table. If values are entered into a column defined with a unique key, repeating the same data for that column is not possible but it can contain any number of null values. According to Oracle one null is not equal to another null.

DOMAIN INTEGRITY CONSTRAINTS

Domain Integrity constraints are based on the column values and any deviations or violations are prevented. The two types of Domain Integrity

Constraints are

- Not Null Constraints
- Check Constraints

Choosing NULL Constraints

By Default all columns can contain null values. NOT NULL constraints are used for columns that absolutely require values at all times. NOT NULL constraints are often combined with other types of constraints to further restrict the values that can exist in specific columns of a table.

Choosing Check Constraints

Check Constraints are used to check whether the values in the table satisfy the criteria is specified for that column. They contain conditions. The conditions have the following limitations.

Conditions must be a Boolean expression that can be evaluated using the values in the record being inserted or updated Conditions must not contain sub-queries. Conditions cannot contain any SQL functions. Conditions cannot contain pseudo columns.

REFERENTIAL INTEGRITY CONSTRAINT

This constraint establishes the relationship between tables. A single or combination of columns, which can be related to the other tables, is used to perform this operation. Foreign key is used to establish the relationship. This kind of relationship can be referred to as a Parent-Child relationship.

The table containing the referenced Key from wile-re other tables refer for value is called the Parent table and the table containing the foreign key is called the child table.

Adding Constraints

Constraints can be added in two different ways. There are

- Adding at the time of creating tables

- Adding after creating tables

The next section deals with the adding constraints at the time of creating tables. Every constraints contains a name followed by the type of the constraint.

Adding Entity Integrity Constraints

Both Primary Key and Unique Key constraints can be added at the time of creation of tables. Let us consider creating a table called item master, which contain item code, item_name and unit_price.

Example

Create table Item master (item _code number primary key, Item name varchar2 (20) unique,
Unit Price number (9,2));

The table is created along with the constraints. If a constraint is given without specifying thename of the constraint, Oracle by default assigns a name to the constraint that is unique. The constraint starts with _SYS_C‘ followed by some numbers.

After creation, records are inserted into the table as follows:
INSERT INTO item_master VALUES (I, 'Pencils', 2.50);
If the same command is executed again, it raises an error.
INSERT INTO item master VALUES (l, 'Pencils', 2.50);
ERROR at line 1.
ORA-00001: unique constraint (HEMA.SYS_C00769) violated
By looking at this error message, the user may not understand which value is violated in order to avoid this situation, a name has to be provided for every constraint that is easy to read. Refining the above example
Create table Item_master (item_code number CONSTRAINT pkit_code primary key,
Item name varchar2(20) CONSTRAINT unque name unique, Unit_Price number (9,2));

Naming the constraints always provide better readability.

Adding Domain Integrity Constraints

The user has to necessarily provide values for the columns containing NOT NULL values. NOT NULL constraint is ideal in cases where the value for the column must exist-Consider an organization that needs to store all the employee information. In this case, the employee name column cannot be left blank.

CREATING ORACLE TABLE (USING THE CREATE – COMMAND)

Oracle Database is made up of tables that contain rows (horizontal) and contains (vertical). Each column contains a data value at the intersection of A row and a column the table definition contains the name of the attribute (property of field).**and the type** of the data that the column **contains** To create a table, use CREATE TABLE command. CREATE Command is used to define the structure of a table or any object.

Syntax:

CREATE TABLE <table name.> (column 1 datatype, column 2 datatype ... c.);

Here, table name refers to the name of the table or entity, column Hi the name the-first column, column2 the name of the second column and so on. For each column there must be an appropriate data type which describes the type of data it can hold: The statement terminated by a semi-colon.

The following example illustratesthe Creation of a table:

Example

Create table EMPLOYEE (Empno NUMBER Empname CHAR (10), Doj DATE);

This would display

Table created

In the above example, an entity called EMPLOYEE is **created.** It contains columns Emp to that can hold numeric data, *Empname* that contains character data-and *Doj* that contains the type of data. Table names are case-insensitive.

The structure of the data would look like.

Name	Type
EMPNO	NUMBER

EMPNAME	CHAR(10)
DOJ	DATE

While creating tables, consider the following points

- Table name must start with alphabet
- Table name length must not exceed 30 characters
- No two tables can have the**,** same name
- Reserved words of Oracle are not allowed.

4.11.1 DISPLAYING TABLE INFORMATION (VIEWING THE TABLE STRUCTURE)

After creating the table, viewing the structure can be done using **DESCRIBE** followed by the name of the table.

Syntax:

DESC [file] <tablename>

Example:

DESC EMPLOYEE

The output would look like

Name	Type
EMPNO	NUMBER
EMPNAME	CHAR(10)
DOJ	DATE

USING THE ALTER - COMMAND

A table's structure can be altered using the **ALTER** Command. The Command allows the structure of the existing table to be altered by adding new columns or fields dynamically and modifying the existing fields data types. Using this command, one or more columns can be added.

Syntax:

Alter table <tablename> add (columnl datatype, Column2 datatype);

Alter table<tablename>modify (column1 datatype, column2 datatype);

Example :

Consider the previous example where a new column called Salary is to be added.

ALTER TABLE employee ADD (salary NUMBER);

USING THE DROP - COMMAND

DROP command completely removes a table from the database. This command will also destroy the table structure and the data stored in it. Following is its syntax,

DROP TABLE table_name

Here is an example explaining it,

DROP TABLE student;

The above query will delete the Student table completely. It can also be used on Databases, to delete the complete database. For example, to drop a database,

DROP DATABASE Test;

The above query will drop the database with name Test from the system.

USING RENAME – COMMAND

RENAME command is used to set a new name for any existing table. Following is the syntax,

RENAME TABLE old_table_name to new_table_name;

Here is an example explaining it.

RENAME TABLE student to students_info;

The above query will rename the table student to students_info.

USING TRUNCATE – COMMAND

TRUNCATE command removes all the records from a table. But this command will not destroy the table's structure. When we use TRUNCATE command on a table its (auto-increment) primary key is also initialized. Following is its syntax,

TRUNCATE TABLE table_name

Here is an example explaining it,

TRUNCATE TABLE student;

The above query will delete all the records from the table student.

UNIT V

DATA MANIPULATION LANGUAGE (DML)

Data Manipulation Language (DML) statements are used for managing data in database. DML commands are not auto-committed. It means changes made by DML command are not permanent to database, it can be rolled back.

USING INSERT - COMMAND

Insert command is used to add one or more rows to a table. The values are separated commas and the values are entered in the same ORDER as specified by the structure of the table.

Inserting records into tables can be done in different ways:

- Inserting records into all fields
- Inserting records into selective fields
- Continuous insertions.
- Inserting records using SELECT statement.

Use the INSERT command to enter data into a table. Insert data one row at a time, or select several rows from an existing table and insert them all at once.

Following is its general syntax,

INSERT INTO table_name VALUES(data1, data2, ...)

Lets see an example,

Consider a table **student** with the following fields.

s_id	name	age

INSERT INTO student VALUES(101, 'Adam', 15);

The above command will insert a new record into **student** table.

s_id	name	age
101	Adam	15

Insert value into only specific columns

We can use the INSERT command to insert values for only some specific columns of a row. We can specify the column names along with the values to be inserted like this,

INSERT INTO student (id, name) values(102, 'Alex');

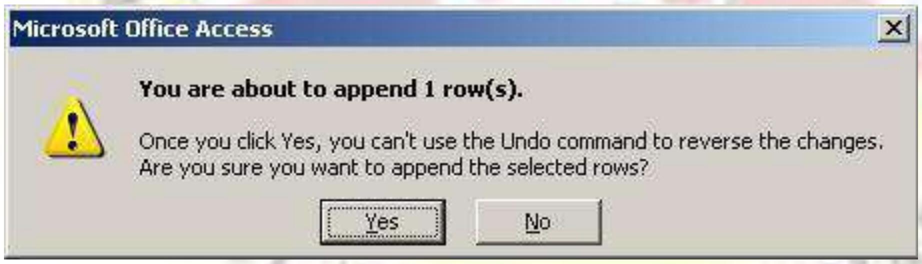
The above SQL query will only insert id and name values in the newly inserted record.

Exercise: Add two rows to the Personnel table, one with all the data filled in, the other with required columns only (two separate queries). Use these SQL statements or similar:

```
INSERT INTO Personnel
VALUES("7777777", "Smith", "John", #7/17/1950#, "Chemistry");

INSERT INTO Personnel (StaffID, LastName, FirstName)
VALUES ("5555555", "Jones", "Jane");
```

NOTE: The date/time data type uses a # delimiter in Access; quotes in other databases.

When you execute the statements, Access will give you a warning (other database systems do not!):



You are about to append 1 row(s).

Once you click Yes, you can't use the Undo command to reverse the changes.
Are you sure you want to append the selected rows?

Click Yes to complete the execution step.

Case 1:

Consider inserting values into Selective fields.

Syntax:

Insert into <tablename> (Selective column1, selective column2) values (value 1, value 2)

Example:

INSERT INTO Employee (empno, empname) VALUES (1330, _Saravanan‘);

Displays the feedback as 1 row created

Case 2:

Consider continuous insertion of records. In order to insert continuously use "&" (ampersand).

Syntax :

Insert into <tablename> Values (&Coll, &Col2, &Co13...);

Oracle prompts the user to insert values onto all the columns of the table. The following example illustrates this.

Example

INSERT INTO employee VALUES (&eno, &name, &doj, &sal);

Output will be:

Enter value: for eno: 1247

Enter valuefor name: Mena

Enter value for doj name:10-jan-2000

Enter value for sal : 5000

old 1: insert into employee values (&eno, _&name‘, &doj, &sal);

new 1: insert into employee values (1247, 'meena', 10-jan-2000', 5000);

1 row created.

Now, consider inserting_ records continuously for selective fields. This is similar to case 2

Insert into <tablename> (selective column l, selective column2) Values (Coll, &co12);

The following example inserts records into the *empno and empname* columns.

Example :

INSERT INTO Employee(empno, empname) VALUES (&eno, '&name‘);

Output

Enter value for eno : 1440

Enter value for name : Diana

old 1: insert into employee values (&eno, _&name‘);

new 1: insert into employee values (1440,‗Diana‘);

1 row created.

Case 3:

Multiple Records can be inserted using a single Insert command along with Select statement. This case is dealt alter the section on Select Statement.

Note : Using Insert and Values combination, only one record can be inserted at a time.

USING THE SELECT – COMMAND (RETRIEVING RECORDS)

Retrieving data from the database is the most common SQL operation. Database retrieval is called a *query* and is performed using SELECT statement. A basic SELECT statement contains two clauses or parts Select some data (columnname(s)) FROM a table or More tables (table name(s)) Retrieval of records can be done in various ways:

- Selecting all records from a table
- Retrieving selective columns for all records from a table
- Selecting records based on conditions
- Selecting records in a sorted order

Consider the first case of selecting all the records from the table.

Example

SELECT empno, empname, doj, salary FROM employee;

Here, all the column names are given in the SELECT clause. This can be further simplified by giving * as follows:

Example

SELECT * FROM employee;

This would display:

EMPNO	EMPNAME	DOJ	SALARY
1237	Kalai	10-MAR-2000	5000
1330	Saravanan		
1247	Meena	10-JAN-2000	5000

1440	Diana		

* INDICATED ALL THE COLUMN NAMES.

Note in the above display, there are no values entered in DOJ and Salary Column for the employee 1001 and 1004. Here the values in these columns are considered to have NULL values. Anytime it can be updated using the Update Command. In the second case, the column names must be specified in the SELECT statement.

Syntax :

SELECT Col1, Col2;

FROM<tablename>;

Example:

SELECT empname, salary FROM employee;

The records would be displayed as follows

EMPNAME	SALARY
Kalai	5000
Saravanan	
Meena	5000
Diana	

This statement retrieves the column values of empname and salary.

5.1.2.1 Conditional Retrieval

Conditional retrieval enables selective rows to be selected. While selecting rows, restriction can be applied through a condition that governs the selection. An additional clause called WHERE must be given along with the SELECT statement to apply the condition to select a specific set of rows! The order of precedence first goes to the WHERE clause and the records that match the condition are alone selected.

Syntax:

Select (column name (s)) FROM (table name(s)) WHERE condition(s) consider selecting employee records whose salary is equal to or greater than 3000. The query can be written as

Example:

SELECT empno, empname, salary FROM employee where salary >=3000;

The records selected will be,

EMPNO	EMPNAME	SALARY
1237	Kalai	5000
1247	Meena	5000

Example :

SELECT * FROM employee WHERE salary = 1000;

The display would be no rows selected since there are no records matching the condition specified in the WHERE clause.

CREATE TABLE employee2 AS SELECT empno, empname FROM EMPLOYEE;

The statements given above create new tables called employee 1 andemployees2 respectively. In the case of employee' table, the structure, which exists in the employee table, is copied and the records are inserted. In the case of employee2 table, two columns are copied from the employee name with the records and me structure. The above Statements can alternately written as.

CREATE TABLE <tablename>

and

INSERT INTO <tablename>SELECT <columnlist> FROM <tablename>

as Inserting records using SELECT Statement.

Copying the Structure of one table can be copied on to another table without the records being copied. In order to do this, along with the SELECT statement, add a WHERE clause which yields to any FALSE condition. The following example explain this

CREATE TABLE employee3 AS SELECT * FROM employee WHERE 1=2;

This statement creates a table called Employee3 whose structure is the same as Employee but the records are not copied since WHERE clause evaluates to FALSE.

DATA ACCESS TECHNIQUES

Data access refers to a user's ability to access or retrieve data stored within a database or other repository. Users who have data access can store, retrieve, move or manipulate stored data, which can be stored on a wide range of hard drives and external devices.

ODBC (Open Database Connectivity, Open Database interconnection) It is an integral part of the database in Microsoft's WOSA (Windows Open Services Architecture). It establishes a set of specifications. It also provides a set of standard APIs for database access application programming interfaces (API). These APIs use SQL to complete most of their tasks. ODBC also provides support for the SQL language can directly send SQL statements to ODBC. It is an early database interface technology introduced by Microsoft. It is actually the predecessor of ADO.

DAO (Data Access Objects): The data Access object is used to expose the Microsoft Jet Database Engine (which was first used for Microsoft Access and now supports other databases) and allows developers to directly connect to other databases through ODBC, directly connect to the Access table. DAO is most suitable for single-system applications or local distribution in a small range. Its internal access to the Jet Database has been accelerated and optimized, and it is also very convenient to use. Therefore, if the database is an Access database and is used locally, we recommend that you use this Access method-application uniqueness.

RDO (Remote Data Objects) The remote data object is an ODBC-oriented data access interface. It is combined with the easy-to-use DAO style and provides an interface, shows the underlying functions and flexibility of all ODBC databases. Although RDO is restricted in its access to Jet or Indexed sequential access method (ISAM) databases, it can only access relational databases through the existing ODBC driver. However, RDO has proved the best interface that have SQL Server, Oracle, and other large relational database developers often choose. RDO provides more complex objects, attributes, and methods used to access stored procedures and complex result sets. It is undoubtedly based on ODBC.

OLE DB Is a strategic system-level programming interface of Microsoft used to manage data within the entire organization. OLE DB is an open specification built on the ODBC function. ODBC is specially developed to access relational databases. OLE DB is used to access relational and non-relational information sources, such as the host Indexed sequential access method (ISAM) / virtual storage access method (VSAM) and hierarchical database, email and file system storage, text, graphics, and geographic data, as well as custom business objects. OLE DB defines a set of COM interfaces, encapsulates various database management system services, and allows you to create software components to implement these services. OLE DB components include data providers (including and presenting data), data users (using data), and service components (processing and transmitting data, such as query processors and cursor engines).
OLE DB interfaces help to smoothly integrate components, so that OLE DB Component vendors can quickly provide high-quality OLE DB components to the market. In addition, OLE DB contains a "bridge" Connecting ODBC, which provides consistent support for various ODBC relational database drivers. Claim to replace ODBC, but also compatible with ODBC.

ADO (ActiveX Data Object) is the successor of DAO/RDO. ADO 2.0 is more functionally similar to RDO, and generally there is a similar between the two models. ADO "extends" the object model used by DAO and RDO, which means it contains fewer objects, more attributes, methods (and parameters), and events. As the latest database access mode, ADO is also easy to use, so Microsoft has clearly stated that it will focus on ADO in the future and will not upgrade DAO/RDO, therefore, ADO has become the mainstream of database development.

ADO (ACTIVEX DATA OBJECT)

ADO involves three Data Storage Methods: DSN (Data Source Name), ODBC (open data connection), and OLE DB. A Data Source Name (**DSN**) is a data structure that contains the information about a specific database that an Open Database Connectivity (ODBC) driver needs in order to connect to it. The following routine will explain in detail the specific access implementation of these three methods. It can be said that it is the integration of system-level programming interfaces such as ODBC and OLEDB, and the upgrade of application-level programming interfaces such as DAO and RDO.

- ADO is a Microsoft technology
- ADO stands for **A**ctiveX **D**ata **O**bjects
- ADO is a Microsoft Active-X component
- ADO is automatically installed with Microsoft Internet Information Services (IIS)
- ADO is a programming interface to access data in a database

Accessing a Database from an Active Server Pages (ASP) Page

The common way to access a database from inside an ASP page is to:

1. Create an ADO connection to a database
2. Open the database connection
3. Create an ADO recordset
4. Open the recordset
5. Extract the data you need from the recordset
6. Close the recordset
7. Close the connection

ADO DATABASE CONNECTION

Before a database can be accessed from a web page, a database connection has to be established.

CREATE AN ODBC DATABASE CONNECTION

If you have an ODBC database called "northwind" you can connect to the database with the following ASP code:

```
<%
set conn=Server.CreateObject("ADODB.Connection")
conn.Open "northwind"
%>
```

With an ODBC connection, you can connect to any database, on any computer in your network, as long as an ODBC connection is available.

An ODBC Connection to an MS Access Database

Here is how to create a connection to a MS Access Database:

1. Open the **ODBC** icon in your Control Panel.
2. Choose the **System DSN** tab.
3. Click on **Add** in the System DSN tab.
4. **Select** the Microsoft Access Driver. Click **Finish.**

5. In the next screen, click **Select** to locate the database.
6. Give the database a **D**ata **S**ource **N**ame (DSN).
7. Click **OK**.

Note that this configuration has to be done on the computer where the web site is located.

ADO CONNECTION OBJECT

The ADO Connection Object is used to create an open connection to a data source. Through this connection, data access and manipulate a database.

To access a database multiple times, establish a connection using the Connection object. Make a connection to a database by passing a connection string via a Command or Recordset object. However, this type of connection is only good for one specific, single query.

ProgID

set objConnection=Server.CreateObject("ADODB.connection")

Properties

Property	**Description**
Attributes	Sets or returns the attributes of a Connection object
CommandTimeout	Sets or returns the number of seconds to wait while attempting to execute a command
ConnectionString	Sets or returns the details used to create a connection to a data source
ConnectionTimeout	Sets or returns the number of seconds to wait for a connection to open
Mode	Sets or returns the provider access permission
Provider	Sets or returns the provider name
State	Returns a value describing if the connection is open or closed
Version	Returns the ADO version number

Methods

Method	Description
BeginTrans	Begins a new transaction
Cancel	Cancels an execution
Close	Closes a connection
CommitTrans	Saves any changes and ends the current transaction
Execute	Executes a query, statement, procedure or provider specific text
Open	Opens a connection
RollbackTrans	Cancels any changes in the current transaction and ends the transaction

Events

Note: You cannot handle events using VBScript or JScript (only Visual Basic, Visual C++, and Visual J++ languages can handle events).

Event	Description
BeginTransComplete	Triggered after the BeginTrans operation
CommitTransComplete	Triggered after the CommitTrans operation
Disconnect	Triggered after a connection ends
ExecuteComplete	Triggered after a command has finished executing
RollbackTransComplete	Triggered after the RollbackTrans operation
WillExecute	Triggered before a command is executed

Collections

Collection	Description
Errors	Contains all the Error objects of the Connection object
Properties	Contains all the Property objects of the Connection object

ADO RECORDSET OBJECT

The ADO Recordset object is used to hold a set of records from a database table. Recordset object consist of records and columns (fields).

In ADO, this object is the most important and the one used most often to manipulate data from a database.

Examples

GetRows

This example demonstrates how to use the GetRows method.

ProgID

```
set objRecordset=Server.CreateObject("ADODB.recordset")
```

When you first open a Recordset, the current record pointer will point to the first record and the BOF and EOF properties are False. If there are no records, the BOF and EOF property are True.

Recordset objects can support two types of updating:

- **Immediate updating** - all changes are written immediately to the database once you call the Update method.
- **Batch updating** - the provider will cache multiple changes and then send them to the database with the UpdateBatch method.

In ADO there are 4 different cursor types defined:

- **Dynamic cursor** - Allows you to see additions, changes, and deletions by other users.
- **Keyset cursor -** Like a dynamic cursor, except that you cannot see additions by other users, and it prevents access to records that other users have deleted. Data changes by other users will still be visible.
- **Static cursor** - Provides a static copy of a recordset for you to use to find data or generate reports. Additions, changes, or deletions by other users will not be visible. This is the only type of cursor allowed when you open a client-side Recordset object.
- **Forward-only cursor** - Allows you to only scroll forward through the Recordset. Additions, changes, or deletions by other users will not be visible.

The cursor type can be set by the CursorType property or by the CursorType parameter in the Open method.

Note: Not all providers support all methods or properties of the Recordset object.

PROPERTIES

Property	**Description**
AbsolutePage	Sets or returns a value that specifies the page number in the Recordset object
AbsolutePosition	Sets or returns a value that specifies the ordinal position of the current record in the Recordset object
CacheSize	Sets or returns the number of records that can be cached
CursorLocation	Sets or returns the location of the cursor service
CursorType	Sets or returns the cursor type of a Recordset object
DataMember	Sets or returns the name of the data member that will be retrieved from the object referenced by the DataSource property
DataSource	Specifies an object containing data to be represented as a Recordset object
EditMode	Returns the editing status of the current record
EOF	Returns true if the current record position is after the last record, otherwise false
PageSize	Sets or returns the maximum number of records allowed on a single page of a Recordset object
RecordCount	Returns the number of records in a Recordset object
Sort	Sets or returns the field names in the Recordset to sort on
Source	Sets a string value or a Command object reference, or returns a String value that indicates the data source of the Recordset object
State	Returns a value that describes if the Recordset object is open, closed, connecting, executing or retrieving data
Status	Returns the status of the current record with regard to batch updates or other bulk operations
StayInSync	Sets or returns whether the reference to the child records will change when the parent record position changes

METHODS

Method	**Description**
AddNew	Creates a new record
Cancel	Cancels an execution
Clone	Creates a duplicate of an existing Recordset
Close	Closes a Recordset
CompareBookmarks	Compares two bookmarks
Delete	Deletes a record or a group of records
Find	Searches for a record in a Recordset that satisfies a specified criteria
GetRows	Copies multiple records from a Recordset object into a two-dimensional array
GetString	Returns a Recordset as a string
Move	Moves the record pointer in a Recordset object
MoveFirst	Moves the record pointer to the first record
MoveLast	Moves the record pointer to the last record
MoveNext	Moves the record pointer to the next record
MovePrevious	Moves the record pointer to the previous record
Requery	Updates the data in a Recordset by re-executing the query that made the original Recordset
Resync	Refreshes the data in the current Recordset from the original database
Save	Saves a Recordset object to a file or a Stream object
Update	Saves all changes made to a single record in a Recordset object

EVENTS

Note: You cannot handle events using VBScript or JScript (only Visual Basic, Visual C++, and Visual J++ languages can handle events).

Event	**Description**
EndOfRecordset	Triggered when you try to move to a record after the last record
FetchComplete	Triggered after all records in an asynchronous operation have been fetched
FetchProgress	Triggered periodically in an asynchronous operation, to state how many more records that have been fetched
FieldChangeComplete	Triggered after the value of a Field object change
MoveComplete	Triggered after the current position in the Recordset has changed
WillMove	Triggered before the current position in the Recordset changes

Collections

Collection	**Description**
Fields	Indicates the number of Field objects in the Recordset object
Properties	Contains all the Property objects in the Recordset object

FORMS AND REPORTS

Forms and reports are an important part of the database application. Designer use them to create an integrated application, making it easier for user to perform their task. Decision maker and clerical workers use from and report on a daily basis. Normally forms were used as input and report were used to display result. Now a day forms are also used to display result, Basic use of forms are:

- Collect data
  Display query data
  Display analysis and computation result

Switch board
Direct manipulation of object like Graphics

Reports are typically printed on paper, but they are increasingly begin created for direct display on the screen. Report can be used to format the data and present results from complex analysis. Forms and reports have several common features.

DESIGN OF FORMS AND REPORT

The most important concept to remember when designing forms and reports is to understand that they are the primary concert of the user. The key of effective design is to determine the needs of the user. As a designer, you talk with to learn what they want to accomplish. Then you use your experience to provide features that make the form more useful. Researchers in human factors have developed several guidelines to help you design forms. Some factors are:
a. User Control: match user task, Respond to user control and event User customization.
b. Consistency: Layout, design, and color, action.
c. Feedback: Method(Visual, text, Audio, Graphics)
d. Forgiveness: Correction of errors, Confirmation on delete and updates.

FORM LAYOUT

Individual forms or windows are your primary means of communication with people who use your application. Forms are used to collect data, display results and organize the overall system. Several standard layouts are provided by the DBMS to simplify the development of many common forms. Normally we will be working with four basic types of form.

a. Tabular Forms:-
One of the simplest forms is the tabular forms, which displays the columns and rows from a table or query. It can be used as a sub form and is rarely used as a stand-alone form. Microsoft Access provides an even simpler version of form called a datasheet.

b. Single-Row or Columnar Forms:-
A single-row form displays data for one row at a time. The goal is to display every column. Its greatest feature is that the designer can display the data at any location on the form. It is useful for designing a form like a traditional paper form.

c. Subform Forms:-
A subform is usually a datasheet (or tabular form) embedded on the main form. A subform generally shows a one-to-many relationship.

d. Switchboard:-
It provides overall structure of an application. It directs the user to other form & report in the application. It often contains image and reflect the style of the company.

DATA REPORTS

Several issue are involve in designing report as in the development of form, you and users need to determine the content and layout of reports. Issues in designing report are:

a. Report usage and user need.
b. Report layout choice like tabular, subgroup, chart, etc.
c. Paper size
d. How often it is generated.
e. Even that triggers report.
f. How large is the report.
g. Colors
h. Security control, etc.

TYPES OF REPORT

a. Tabular and level report
It is basically means printing column of data like output of query in tabular reports, data are presented in tabular forms .Example grade sheets of all students.

b. Group and subtotal Report
The most common types of report is based on groups and compute, subtotals common example may be printing receipt or a bills.

Form, Report, and Control objects are Microsoft Access objects. You can set properties for these objects from within a Sub, Function, or event procedure. Set properties for form and report sections also possible.

SET A PROPERTY OF A FORM OR REPORT

Refer to the individual form or report within the Forms or Reports collection, followed by the name of the property and its value. For example, to set the Visible property of the Customers form to True (-1), use the following line of code:

VBCopy

```
Forms!Customers.Visible = True
```

You can also set a property of a form or report from within the object's module by using the object's **Me** property. Code that uses the **Me** property executes faster than code that uses a fully qualified object name. For example, to set the **RecordSource** property of the Customers form to an SQL statement that returns all records with a CompanyName field entry beginning with "A" from within the Customers form module, use the following line of code:

VBCopy

```
Me.RecordSource = "SELECT * FROM Customers " _& "WHERE CompanyName Like 'A*'"
```

SET A PROPERTY OF A CONTROL

Refer to the control in the **Controls** collection of the **Form** or **Report** object on which it resides. You can refer to the **Controls** collection either implicitly or explicitly, but the code executes faster if you use an implicit reference. The following examples set the **Visible** property of a text box called CustomerID on the Customers form:

VBCopy

```
' Faster method.
Me!CustomerID.Visible = True
```

VBCopy

```
' Slower method.
Forms!Customers.Controls!CustomerID.Visible = True
```

The fastest way to set a property of a control is from within an object's module by using the object's **Me** property. For example, you can use the following code to toggle the **Visible** property of a text box called CustomerID on the Customers form:

VBCopy

```
With Me!CustomerID
   .Visible = Not .Visible
End With
```

SET A PROPERTY OF A FORM OR REPORT SECTION

Refer to the form or report within the Forms or Reports collection, followed by the Section property and the integer or constant that identifies the section. The following examples set the Visible property of the page header section of the Customers form to False**:**

VBCopy

```
Forms!Customers.Section(3).Visible = False
```

VBCopy

Me!Section(acPageHeader).Visible = False

CREATING A DATA REPORT IN VISUAL BASIC 6 (VB6)

Once you have gone to all the trouble of developing and managing a database, it is nice to have the ability to obtain printed or displayed information from your data. The process of obtaining such information is known as creating a data report.

There are two steps to creating a data report. First, we need to create a Data Environment. This is designed within Visual Basic and is used to tell the data report what is in the database. Second, we create the Data Report itself. This, too, is done within Visual Basic. The Data Environment and Data Report files then become part of the Visual Basic project developed as a database management system.

The Visual Basic 6.0 data report capabilities are vast and using them is a detailed process. The use of these capabilities is best demonstrated by example. We will look at the rudiments of report creation by building a tabular report for our phone database.

EXAMPLE - PHONE DIRECTORY - BUILDING A DATA REPORT

We will build a data report that lists all the names and phone numbers in our phone database. We will do this by first creating a Data Environment, then a Data Report. We will then reopen the phone database management project and add data reporting capabilities.

CREATING A DATA ENVIRONMENT

1. Start a new **Standard EXE** project.

2. On the Project menu, click **Add Data Environment**. If this item is not on the menu, click **Components**. Click the **Designers** tab, and choose **Data Environment** and click **OK** to add the designer to your menu.

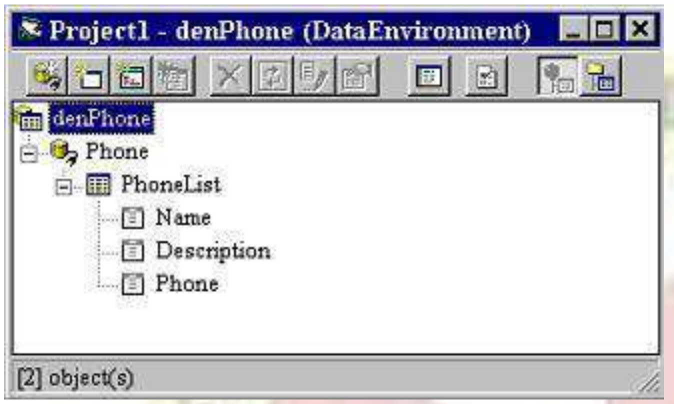
3. We need to point to our database. In the **Data Environment** window, right-click the **Connection1** tab and select **Properties**. In the **Data Link Properties** dialog box, choose **Microsoft Jet 3.51 OLE DB Provider**. Click **Next** to get to the **Connection** tab. Click the **ellipsis** button. Find your phone database (mdb) file. Click **OK** to close the dialog box.

4. We now tell the **Data Environment** what is in our database. Right-click the **Connection1** tab and click **Rename**. Change the name of the tab to Phone. Right-click this newly named tab and click **Add Command** to create a **Command1** tab. Right-click this tab and choose **Properties**. Assign the following properties:

Command Name - PhoneList
Connection - Phone
DataBase Object - Table
ObjectName - PhoneList

5. Click **OK**. All this was needed just to connect the environment to our database.

6. Display the properties window and give the data environment a name property of denPhone. Click **File** and **Save** denPhone As. Save the environment in an appropriate folder. We will eventually add this file to our phone database management system. At this point, my data environment window looks like this (I expanded the PhoneList tab by clicking the + sign):



CREATING A DATA REPORT

Once the Data Environment has been created, we can create a Data Report. We will drag things out of the Data Environment onto a form created for the Data Report, so make sure your Data Environment window is still available.

1. On the Project menu, click Add Data Report and one will be added to your project. If this item is not on the menu, click Components. Click the Designers tab, and choose Data Report and click OK to add the designer to your menu.

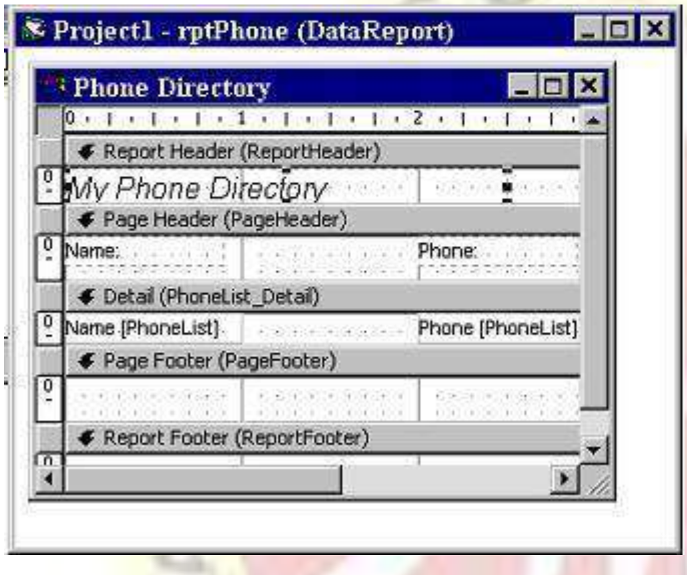
2. Set the following properties for the report:

Name - rptPhone
Caption - Phone Directory
DataSource - denPhone (your phone data environment - choose, don't type)
DataMember - PhoneList (the table name - choose don't type)

3. Right-click the **Data Report** and click **Retrieve Structure**. This establishes a report format based on the **Data Environment.**

4. Note there are five sections to the data report: a Report Header, a Page Header, a Detail section, a Page Footer, and a Report Footer. The headers and footers contain information you want printed in the report and on each page. To place information in one of these regions, right-click the selected region, click Add Control, then choose the control you wish to place. These controls are called data report controls and properties are established just like you do for usual controls. Try adding some headers.

5. The Detail section is used to layout the information you want printed for each record in your database. We will place two field listings (Name, Phone) there. Click on the Name tab in the Data Environment window and drag it to the Detail section of the Data Report. Two items should appear: a text box Name and a text box Name (PhoneList). The first text box is heading information. Move this text box into the Page Header section. The second text box is the actual value for Name from the PhoneList table. Line this text box up under the Name header. Now, drag the Phone tab from the Data Environment to the Data Report. Adjust the text boxes in the same manner. Our data report will have page headers Name and Phone. Under these headers, these fields for each record in our database will be displayed. When done, the form should look something like this:



In this form, I've resized the labels a bit and added a Report Header. Also, make sure you close up the Detail section to a single line. Any space left in this section will be inserted after each entry.

6. Click File and Save rptPhone As. Save the environment in an appropriate folder. We will now reopen our phone database manager and attach this and the data environment to that project and add capabilities to display the report.

ACCESSING THE DATA REPORT

1. Reopen the phone directory project. Add a command button named cmdReport and give it a Caption of Show Report. (There may be two tabs in your toolbox, one named General and one named DataReport. Make sure you select from the General tools.)

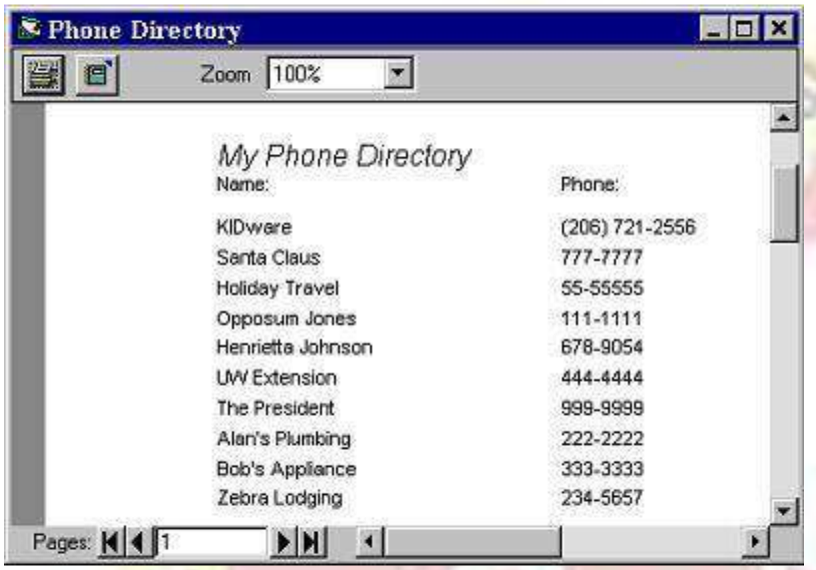
2. We will now add the data environment and data report files to the project. Click the Project menu item, then click Add File. Choose denPhone and click OK. Also add rptPhone. Look at your Project Window. Those files should be listed under Designers.

3. Use this code in cmdReport_Click:

```
Private Sub cmdReport_Click()
rptPhone.Show
End Sub
```

4. This uses the Show method to display the data report.

5. Save the application and run it. Click the Show Report button and this should appear:



My Phone Directory

Name:	Phone:
KIDware	(206) 721-2556
Santa Claus	777-7777
Holiday Travel	55-55555
Opposum Jones	111-1111
Henrietta Johnson	678-9054
UW Extension	444-4444
The President	999-9999
Alan's Plumbing	222-2222
Bob's Appliance	333-3333
Zebra Lodging	234-5657