Explain SDI & MDI Applications.

The SDI and MDI forms are the interface design for document handling within a single Windows application. The MDI stands for Multiple Document Interface whereas SDI stands for Single Document Interface.

MDI: A multiple document Interface is one that allows viewing multiple windows within a large window.

SDI: A single Document Interface is one where all Windows appear independently of one another without the unification of a single parent window.

The Visual Basic IDE can be viewed in two ways:

1. With the Multiple Document Interface (MDI)
2. Single Document Interface (SDI)

MDI view shows all the distinct windows of Visual Bassic IDE as child windows within on large IDE Window.

In the SDI view, distinct windows of the Visual Basic IDE exist independently of each other.

MDI Forms

- This is the main form or parent form which is not duplicated, but acts like a container for all the Windows which is also called the primary window.
- The windows in which the individual documents are displayed are called Child Windows.
- An MDI application must have atleast two form, the primary parent form and one or more child forms.
- The parent form may not contain any controls. While the parent Form is open in design mode, the icon on the tool box are not displayed, but you can't place any control on the form.
- The parent form usually have a menu.

Compare SDI & MDI

SDI (Single Document Interface) applications allow only one open document frame window at a time. WordPad, Paint etc are examples of SDI applications.

MDI (Multi Document Interface) applications allow multiple document frame windows to be open in the same instance of an application. An MDI application has a window within which multiple MDI child windows, which are frame windows themselves, can be opened, each containing a separate document. In some applications, the child windows can be of different types, such as chart windows and spreadsheet windows. In that case, the menu bar can change as MDI child windows of different types are activated.
Word,Excel etc are examples of MDI applications.

Image list

The Windows Forms ImageList component is used to store images, which can then be displayed by controls. An image list allows you to write code for a single, consistent catalog of images. You can also associate the same image list with multiple controls

Properties:

Properties	Description
ColorDepth	Property gets the color depth for this image list. determines the number of colors that the images are rendered with
Images	Property gets an ImageCollection object for this image list.
ImageSize	Property used to get or set the image size for the images in the list.
TransparentColor	Property used to get or set the transparent color for this list.

Methods:

Method	Description
Draw	Method used to draw the given image.

Helpprovider

This control provides pop-up or on line help for controls. HelpProvider control Provides help with your application is very useful as it allows your users to understand it more easily, thereby increasing productivity and saving some money. Support for help in Visual Basic exists and you can display HTML files that can contain a set of linked topics.

Help class:

The Help class allows us to display HTML help to users. The Help class provides two methods: ShowHelp and ShowHelpIndex.

ShowHelp - ShowHelp is used to display a help file for a particular control and requires that control to be displayed along with the help file. The URL you specify for the help file can be in the form of F:\myHelp (local machine) or http:\\www.abc.com\help.htm (Web).

ShowHelpIndex -ShowHelpIndex method is used to display the index of a specified help file. You call the ShowHelpIndex method just like you call the ShowHelp method.

Properties :

HelpString - Determines the help string associated with a control.

HelpKeyWord - Determines the help keyword associated with a control.

ShowHelp - Determines if help should be displayed for this control.

Error provider

The errorprovider control provides a user interface to indicate to the user that a control on a form has an error associated with it, or The ErrorProvider component provides an easy way to set validation errors. It allows us to set an error message for any control on the form when the input is not valid. When an error message is set, an icon indicating the error will appear next to the control and the error message is displayed as Tool Tip when the mouse is over the control.

Method

SetError(control,message) –To specify the control and message. If the string length is greater than zero, then the error icon is displayed, and the ToolTip for the error icon is the message description text. If the string length is zero, the error icon is hidden.

Properties:

Icon - Icon property are used to indicate an error.

ContainerControl - ContainerControl property are used to parent control, usually the form, that contains the data-bound controls on which the errorprovider can display error icon.

BlinkStyle - Controls whether the error icon blinks when an error is set.

BlinkRate The rate at which the error icon should flash. The rate is expressed in milliseconds. The default is 250 milliseconds.

DataGrid View

DataGridView provides a visual interface to data. It is an excellent way to display and allow editing for your data. It is accessed with VB.NET code. Data edited in the DataGridView can then be persisted in the database.

The DataGridView control looks like the Excel spreadsheet, which can be added onto the form so that the end-user can use it for specific purposes.

The DataGridView control is basically made of rows and columns. Additionally, every column has its header, where the header text can be changed. The header text can be changed in the designer. It can also be changed programmatically. Vertical and horizontal scrollbars appear automatically whenever they are needed. The scrollbars are used to scroll through the pages to see more content. So a DataGridView control may contain information of multiple pages.

Show Data From Database into DataGridView

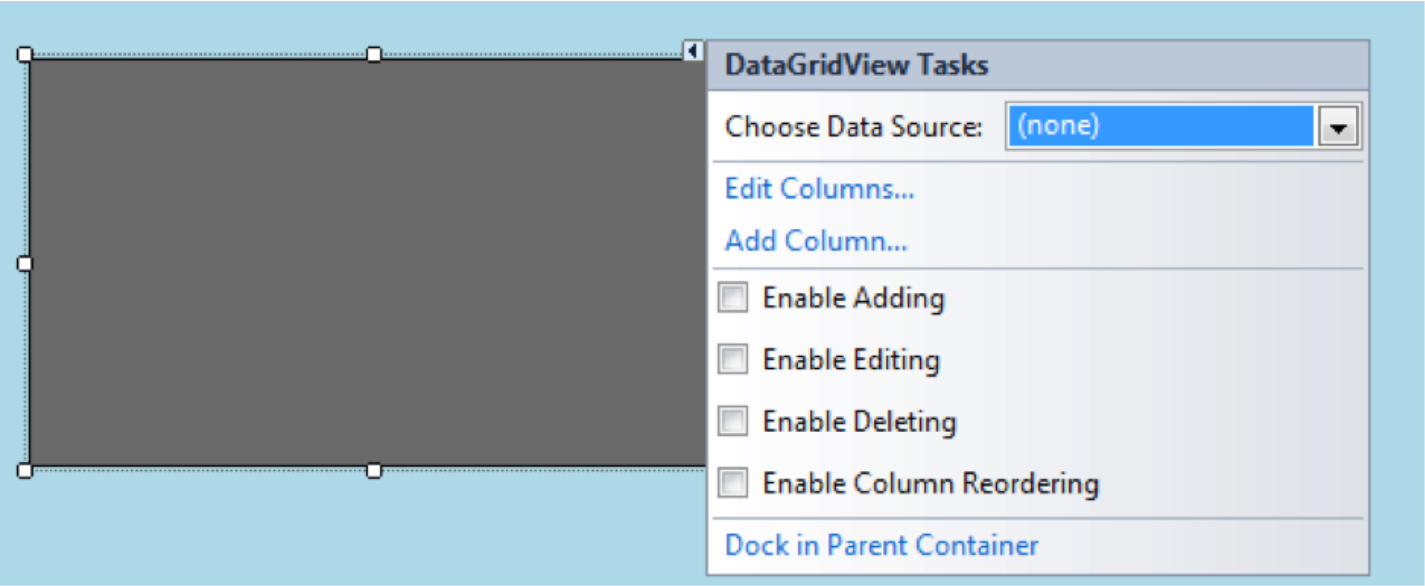
Although you can add data manually, the DataGridView control is generally used for showing data from the database. For example, if you have an Access database connected to the Visual Basic project, you can show

data from it. The data from the database will be shown in rows and columns of the DataGridView control. Besides Access databases, you can also show data on the DataGridView control from other database management systems like Oracle, MS SQL Server, MySQL, dBASE, etc.

Choosing Data Source for the DataGridView control

After adding the DataGridView control on the form, the first thing that you need to do is to choose the data source, either from the designer or in the code, so that the data is automatically shown from the database.

Choosing data source from the designer: The following screenshot demonstrates how to choose data source from the designer.

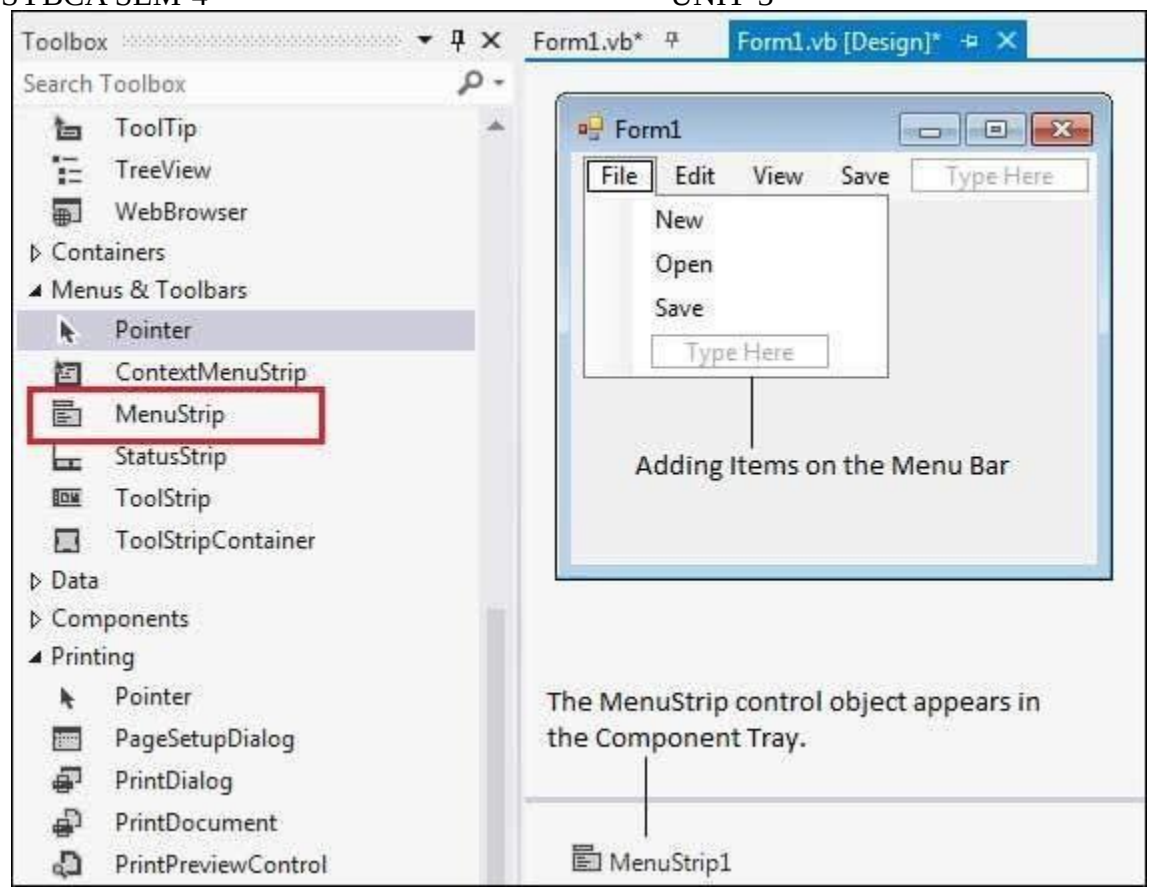


Menus

The **MenuStrip** control represents the container for the menu structure.

The MenuStrip control works as the top-level container for the menu structure. The ToolStripMenuItem class and the ToolStripDropDownMenu class provide the functionalities to create menu items, sub menus and drop-down menus.

If you need a separator bar, right click on your menu then go to insert->Separator.



Sr.No.	Property & Description
1	**CanOverflow** Gets or sets a value indicating whether the MenuStrip supports overflow functionality.
2	**GripStyle** Gets or sets the visibility of the grip used to reposition the control.
3	**MdiWindowListItem** Gets or sets the ToolStripMenuItem that is used to display a list of Multiple-document interface (MDI) child forms.

4	**ShowItemToolTips** Gets or sets a value indicating whether ToolTips are shown for the MenuStrip.

5	**Stretch** Gets or sets a value indicating whether the MenuStrip stretches from end to end in its container.

Sr.No.	Event & Description
1	**MenuActivate** Occurs when the user accesses the menu with the keyboard or mouse.
2	**MenuDeactivate** Occurs when the MenuStrip is deactivated.

Dialogue

There are many built-in dialog boxes to be used in Windows forms for various tasks like opening and saving files, printing a page, providing choices for colors, fonts, page setup, etc., to the user of an application. These built-in dialog boxes reduce the developer's time and workload.

All of these dialog box control classes inherit from the **CommonDialog** class and override the *RunDialog()* function of the base class to create the specific dialog box.

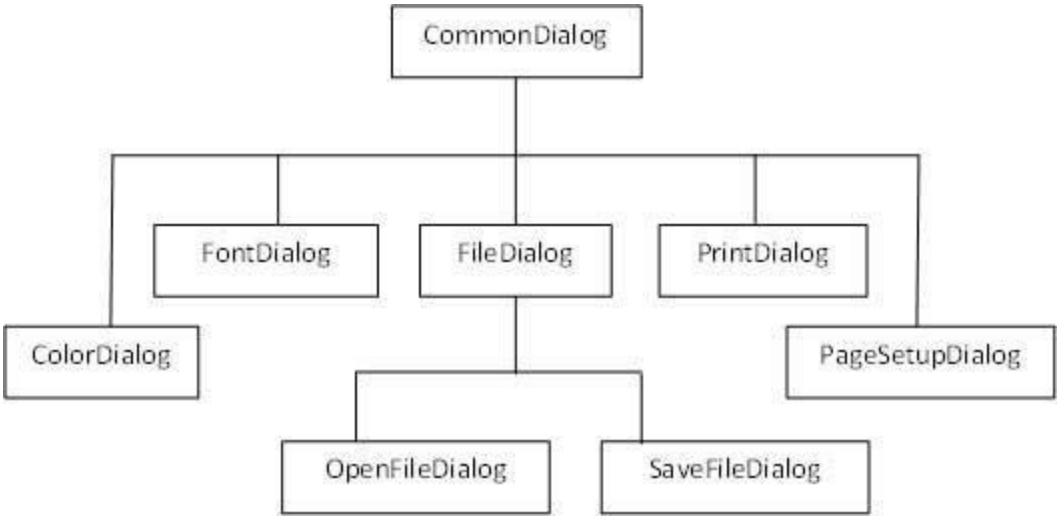
The RunDialog() function is automatically invoked when a user of a dialog box calls its *ShowDialog()* function.

The **ShowDialog** method is used to display all the dialog box controls at run-time. It returns a value of the type of **DialogResult** enumeration. The values of DialogResult enumeration are −

- **Abort** − returns DialogResult.Abort value, when user clicks an Abort button.
- **Cancel** − returns DialogResult.Cancel, when user clicks a Cancel button.
- **Ignore** − returns DialogResult.Ignore, when user clicks an Ignore button.
- **No** − returns DialogResult.No, when user clicks a No button.
- **None** − returns nothing and the dialog box continues running.
- **OK** − returns DialogResult.OK, when user clicks an OK button

- **Retry** − returns DialogResult.Retry , when user clicks an Retry button
- **Yes** − returns DialogResult.Yes, when user clicks an Yes button

The following diagram shows the common dialog class inheritance −



All these above-mentioned classes have corresponding controls that could be added from the Toolbox during design time. You can include relevant functionality of these classes to your application, either by instantiating the class programmatically or by using relevant controls.

When you double click any of the dialog controls in the toolbox or drag the control onto the form, it appears in the Component tray at the bottom of the Windows Forms Designer, they do not directly show up on the form.

The following table lists the commonly used dialog box controls. Click the following links to check their detail

Sr.No.	Control & Description
1	**ColorDialog** It represents a common dialog box that displays available colors along with controls that enable the user to define custom colors.
2	**FontDialog** It prompts the user to choose a font from among those installed on the local computer and lets the user select the font, font size, and color.
3	**OpenFileDialog** It prompts the user to open a file and allows the user to select a file to open.

4	**SaveFileDialog** It prompts the user to select a location for saving a file and allows the user to specify the name of the file to save data.
5	**PrintDialog** It lets the user to print documents by selecting a printer and choosing which sections of the document to print from a Windows Forms application.

Modal And Modeless

Model dialog boxes forces the user to acknowledge the dialog before moving before moving onto the application. **Modeless** dialog boxes enable the user to interact with the dialog and the application interchangeably.

A modal dialog box doesn't allow the user to access the parent window while the dialog is open – it must be dealt with and closed before continuing. A modeless dialog can be open in the background.

Example for Model Dialog is Save, Save As Dialog in MS – Word. while it is opening you can't do anything in the application until you close that window. Example for Modeless Dialog is Find, Replace dialogs. You can use Find Dialog, same time you can also work in that word application.

A modeless dialog box has the following characteristics

- It has a thin border
- It can be neither minimized nor maximized. This means that it is not equipped with the Minimize or the Maximize buttons
- It is not represented on the taskbar with a button
- It must provide a way for the user to close it

What is an Exception? Explain types of exception

An exception is a problem that arises during the execution of a program. An exception is a response to an exceptional circumstance that arises while a program is running, such as an attempt to divide by zero.Exceptions provide a way to transfer control from one part of a program to another.

1. Structured Exception Handling

In structured exception handling, blocks of code are encapsulated, with each block having one or more associated handlers. Each handler specifies some form of filter condition on the type of exception it handles. When an exception is raised by code in a protected block, the set of corresponding handlers is searched in order, and the first one with a matching filter condition is executed. A single method can have multiple structured exception handling blocks, and the blocks can also be nested within each other.

The **Try...Catch...Finally** statement is used specifically for structured exception handling

- **Try** − A Try block identifies a block of code for which particular exceptions will be activated. It's followed by one or more Catch blocks.
- **Catch** − A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The Catch keyword indicates the catching of an exception.
- **Finally** − The Finally block is used to execute a given set of statements, whether an exception is thrown or not thrown. For example, if you open a file, it must be closed whether an exception is raised or not.
- **Throw** − A program throws an exception when a problem shows up. This is done using a Throw keyword.

Try Dim i As Integer Dim resultValue As Integer i = 100 resultValue = i / 0 MsgBox("The result is " & resultValue) Catch ex As Exception MsgBox("Exception catch here ..") Finally MsgBox("Finally block executed ") End Try	When you execute the program you will get the message box “Exception catch here ..." Because there is a divide by 0 exceptions occurs.

2. Unstructured Exception Handling

The **On Error** statement is used specifically for unstructured exception handling. In unstructured exception handling, **On Error** is placed at the beginning of a block of code. It then has "scope" over that block; it handles any errors occurring within the block. If the program encounters another **On Error** statement, that statement becomes valid and the first statement becomes invalid.

Button1_Click On Error GoTo nextstep Dim result As Integer Dim num As Integer num = 100 result = num / 0 nextstep: MsgBox("Control Here") End Sub	When you execute the program you will get the message box "Control Here" Because the On Error statement redirect the exception to the Label statement.

List of Standard Exceptions

Exception	Description
OutOfMemoryException	The exception that is thrown when there is not enough memory to continue the execution of a program.
OverflowException	The exception that is thrown when an arithmetic, casting, or conversion operation in a checked context results in an overflow.
StackOverflowException	The exception that is thrown when the execution stack overflows by having too many pending method calls. This class cannot be inherited.
NullReferenceException	The exception that is thrown when there is an attempt to dereference a null object reference.
IndexOutOfRangeException	The exception that is thrown when an attempt is made to access an element of an array with an index that is outside the bounds of the array. This class cannot be inherited.
FormatException	The exception that is thrown when the format of an argument does not meet the parameter specifications of the invoked method.
DivideByZeroException	The exception that is thrown when there is an attempt to divide an integral or decimal value by zero.
DllNotFoundException	The exception that is thrown when a DLL specified in a DLL import cannot be found.
ArgumentNullException	The exception that is thrown when a null reference (**Nothing** in Visual Basic) is passed to a method that does not accept it as a valid argument.
ArgumentOutOfRangeException	The exception that is thrown when the value of an argument is outside the allowable range of values as defined by the invoked method.
ArithmeticException	The exception that is thrown for errors in an arithmetic, casting, or conversion operation.
ArgumentException	The exception that is thrown when one of the arguments provided to a method is not valid.

Define Events

An event is a callback mechanism. With it, objects can notify users that something interesting has happened. If desired, data can be passed from the object to the client as part of the notification.