


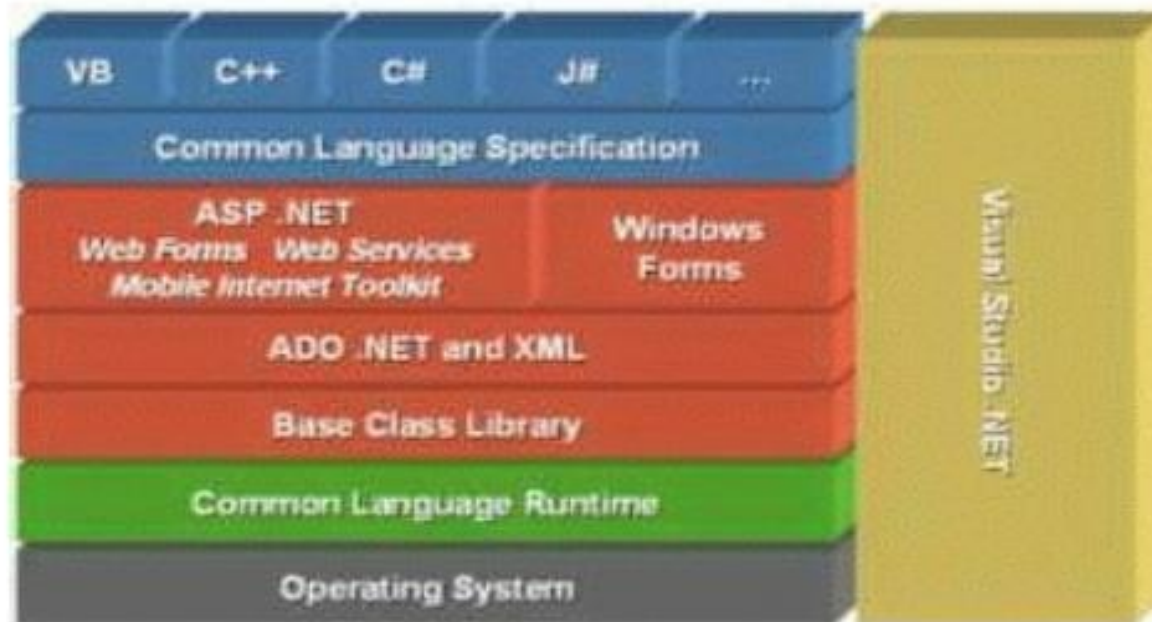
Unit-1

Windows Programming with VB.Net

.NET Architecture or Framework

- ▶ Microsoft Visual Basic .NET is a programming environment used to create graphical user interface (GUI) applications for the Microsoft Windows family of operating systems.
 - ▶ Visual Basic .NET helps you create solutions that run on the Microsoft Windows operating system.
 - ▶ Visual Basic .NET can be used to create applications for use over the Internet.
 - ▶ Visual Studio .NET is an integrated environment for building, testing, and debugging a variety of applications: Windows applications, Web applications, classes and custom controls, even console applications.
 - ▶ It provides numerous tools for automating the development process, visual tools to perform many common design and programming tasks.
- 

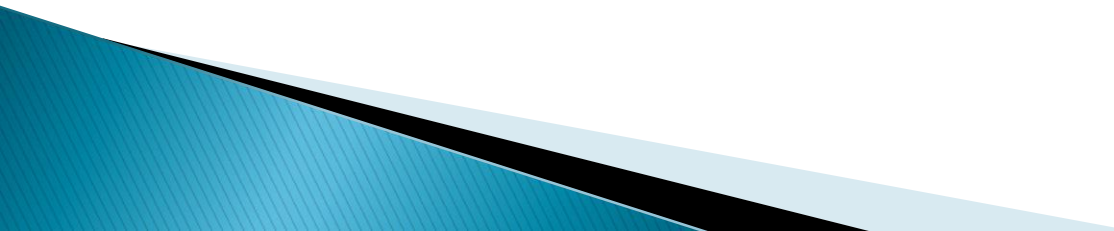
.NET Architecture



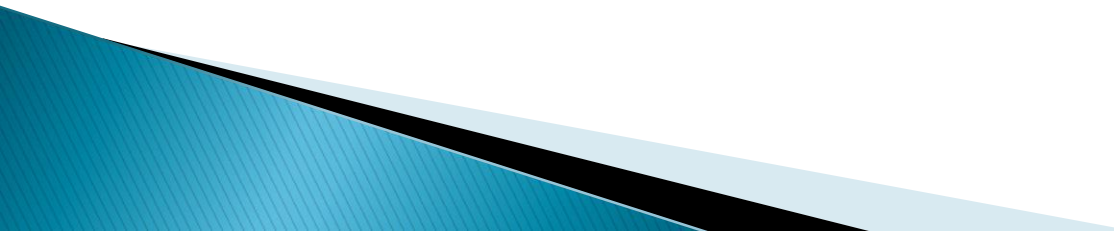
CLR–Common Language Runtime

- ▶ Central to the .NET framework is its run–time execution environment, known as the **Common Language Runtime (CLR)** or the **.NET runtime**.
- ▶ Code running under the control of the CLR is often termed **managed** code.
- ▶ However, before it can be executed by the CLR, any source code that we develop (in C# or some other language) needs to be compiled.
- ▶ Compilation occurs in two steps in .NET:
 - ❖ 1. Compilation of source code to **Microsoft Intermediate Language (MS–IL)**

2. Compilation of IL to platform-specific code by the CLR

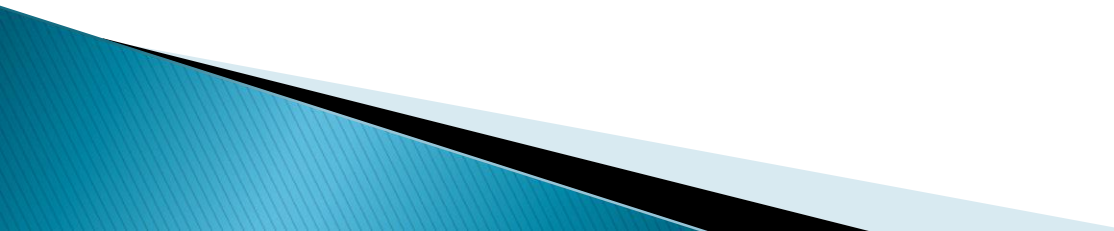
- ▶ The common language runtime manages memory, thread execution, code execution, code safety verification, compilation, and other system services.
 - ▶ The runtime enforces code access security. For example, users can trust that an executable embedded in a Web page can play an animation on screen or sing a song, but cannot access their personal data, file system, or network.
- 

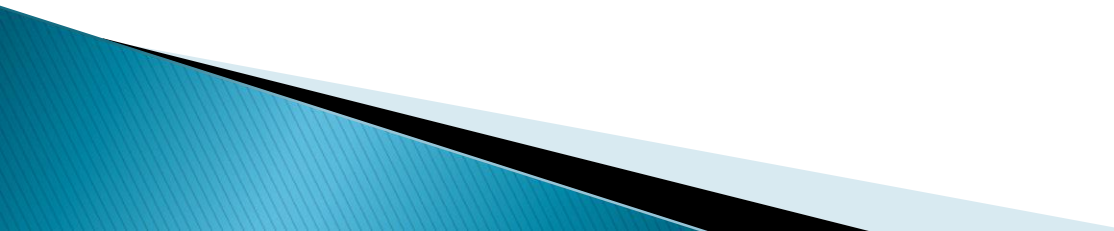
MSIL–Microsoft Intermediate Language

- ▶ Microsoft intermediate language shortly abbreviated as MSIL is a platform independent language meaning that it does not depend on the type of platform we are using to develop applications.
 - ▶ It simply gets compiled into an EXE (executable file) or DLL (Dynamic Link Library) which can be reused in different applications. .NET compiler can generate code written using any of the 23 programming languages supported by .NET like C++.NET, C#.NET, VB.NET, J#.NET etc and finally convert it into required machine code depending on the target machine.
- 


▶ Main advantages of Intermediate Language are:

1) Intermediate Language is independent of any language and hence there is a possibility to create applications with modules that were written in different programming languages supported by .NET. This means that we can develop different modules in an application in different programming languages and can compile the application.

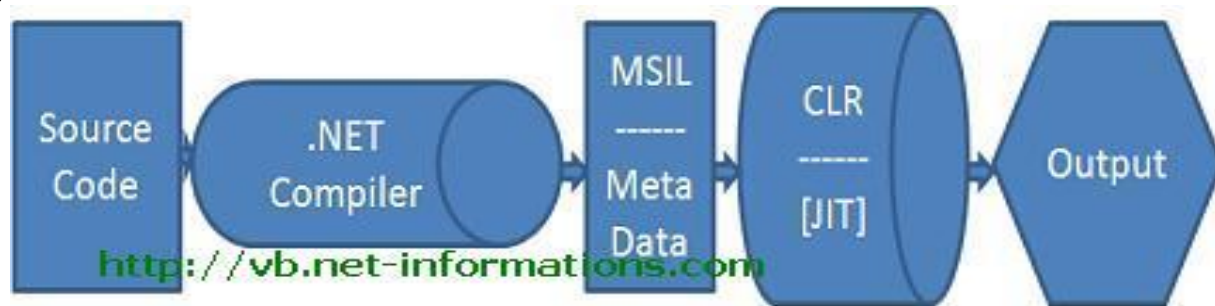


- ▶ Intermediate language is also platform independent which means that it can be compiled to different platforms or operating systems.
 - ▶ .NET is simply called platform independent and language independent language as all the code written in .NET will first be converted into intermediate language and then the .NET compiler compiles it and generates machine language code.
 - ▶ .NET is also referred to as highly interoperable language because of MSIL.
 - ▶ Interoperability is one of the main features in .NET which generally make users select .NET as the technology to develop applications.
- 

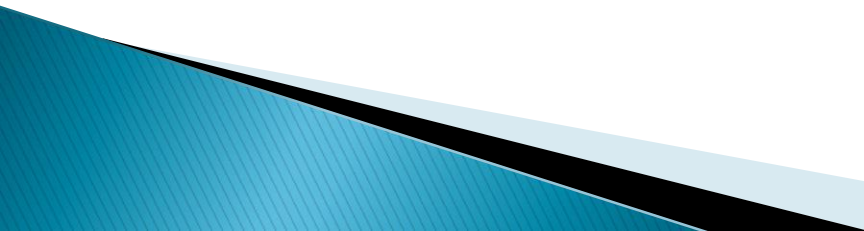
How Microsoft .NET Framework Works

- ▶ Microsoft .NET Languages Source Code are compiled into Microsoft Intermediate Language (MSIL) .
 - ▶ MSIL we can call it as Intermediate Language (IL) or Common Intermediate Language (CIL).
 - ▶ Microsoft Intermediate Language (MSIL) is a CPU independent set of instructions that can be converted to the native code.
 - ▶ Metadata also created in the course of compile time with Microsoft Intermediate Language (MSIL) and stored it with the compiled code .
- 

- ▶ Metadata is completely self-describing. Metadata is stored in a file called Manifest, and it contains information about the members, types, references and all the other data that the Common Language Runtime (CLR) needs for execution.



- ▶ The Common Language Runtime (CLR) uses metadata to locate and load classes, generate native code, provide security, and execute Managed Code.

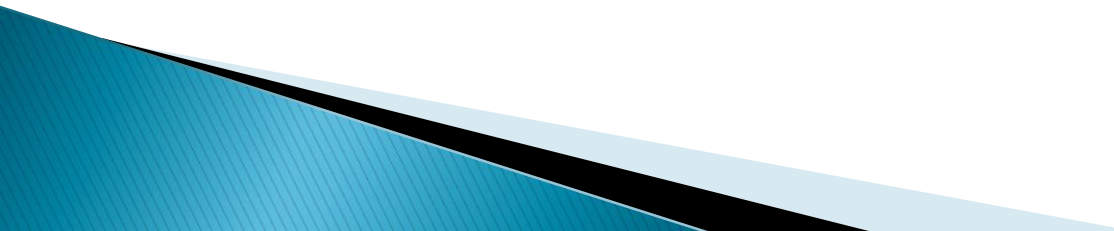
- ▶ During the runtime the Common Language Runtime (CLR)'s Just In Time (JIT) compiler converts the Microsoft Intermediate Language (MSIL) code into native code to the Operating System.
 - ▶ The native code is Operating System independent and this code is known as Managed Code , that is, the language's functionality is managed by the .NET Framework .
 - ▶ The Common Language Runtime (CLR) provides various Just In Time (JIT) compilers, and each works on a different architecture depends on Operating Systems, that means the same Microsoft Intermediate Language (MSIL) can be executed on different Operating Systems.
- 

Just In Time Compiler – JIT

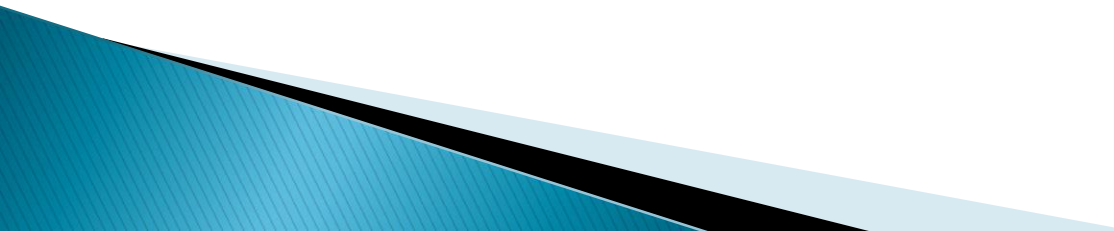
- ▶ The .Net languages uses its corresponding runtime to run the application on different Operating Systems .
- ▶ During the code execution time, the Managed Code compiled only when it is needed, that is it converts the appropriate instructions to the native code for execution just before when each function is called.
- ▶ This process is called Just In Time (JIT) compilation, also known as Dynamic Translation . With the help of Just In Time Compiler (JIT) the Common Language Runtime (CLR) doing these tasks.

Microsoft .Net Assembly


- ▶ Microsoft .Net Assembly is a logical unit of code, that contains code which the Common Language Runtime (CLR) executes. It is the smallest unit of deployment of a .net application and it can be a .dll or an .exe .
- ▶ It include both executable application files that you can run directly from Windows without the need for any other programs (.exe files), and libraries (.dll files) for use by other applications.
- ▶ Assemblies are the building blocks of .NET Framework applications. During the compile time Metadata is created with Microsoft Intermediate Language (MSIL) and stored in a file called Assembly Manifest .

- ▶ Every Assembly you create contains one or more program files and a Manifest. There are two types program files : Process Assemblies (EXE) and Library Assemblies (DLL).
 - ▶ We can create two types of Assembly:
 1. Private Assembly
 2. Shared Assembly
 - ▶ A private Assembly is used only by a single application
 - ▶ A shared Assembly is one that can be referenced by more than one application.
- 

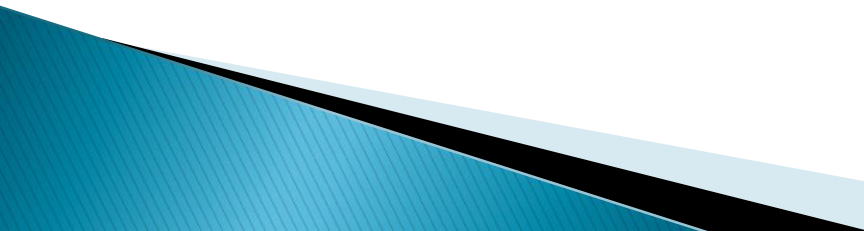
The .NET framework class library

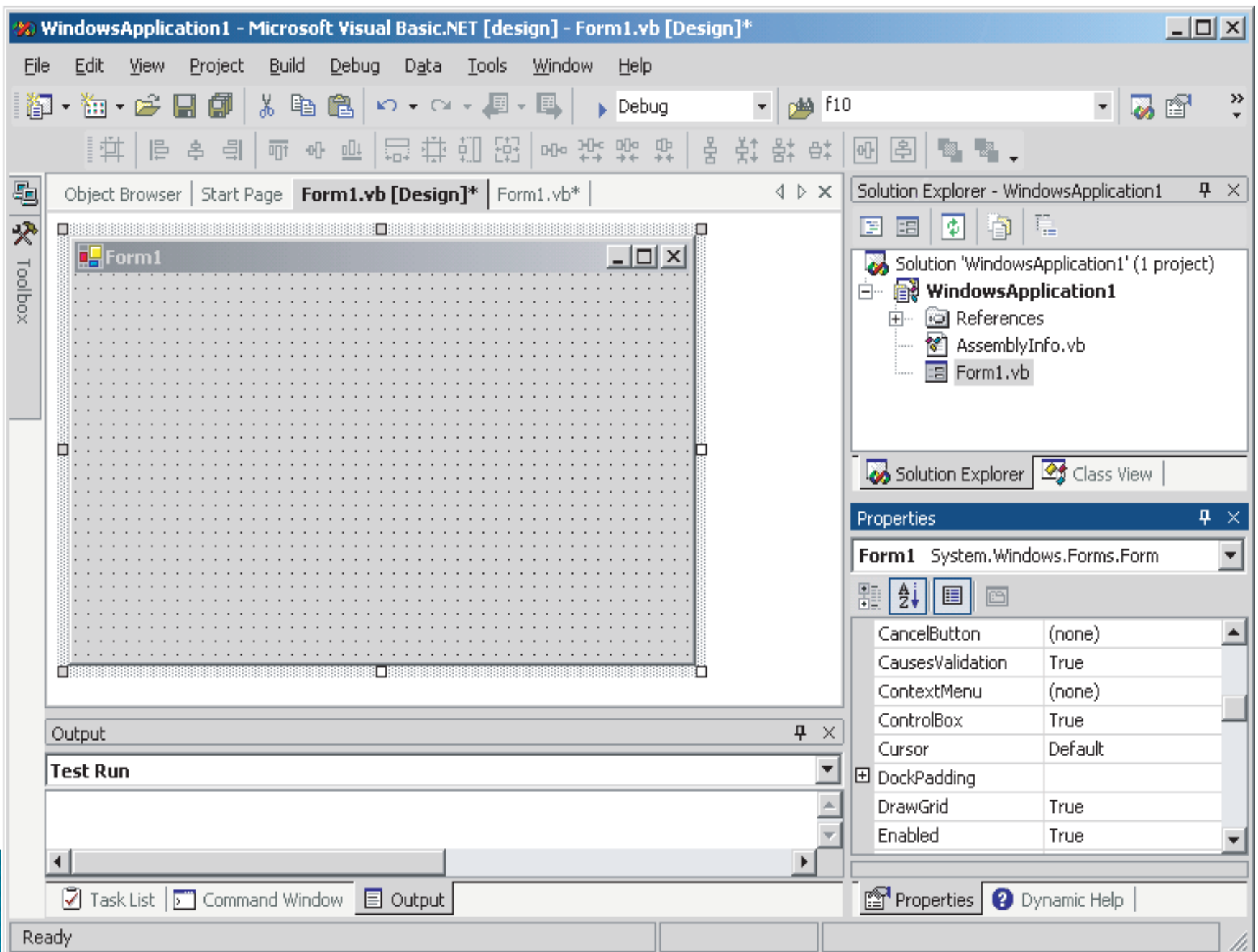
- ▶ The Framework class library (FCL) is a comprehensive collection of reusable types including classes, interfaces and data types included in the .NET Framework to provide access to system functionality.
 - ▶ The .NET FCL forms the base on which applications, controls and components are built in .NET.
 - ▶ It can be used for developing applications such as console applications, Windows GUI applications, ASP.NET applications, Windows and Web services, workflow-enabled applications, service oriented applications using Windows Communication, XML Web services, etc.
- 

VB.Net Applications

- ▶ The .NET framework is a revolutionary platform that helps you to write the following types of applications:
 - 1) Windows applications
 - 2) Web applications
 - 3) Web services
 - ▶ The .NET framework applications are multi-platform applications.
 - ▶ The framework has been designed in such a way that it can be used from any of the following languages: Visual Basic, C#, C++, Jscript, and COBOL, etc.
 - ▶ All these languages can access the framework as well as communicate with each other.
 - ▶ The .NET framework consists of an enormous library of codes used by the client languages like VB.Net.
 - ▶ These languages use object-oriented methodology.
- 

Introduction to .net IDE and its components

- ▶ To simplify the process of application development, Visual Studio .NET provides an environment that's common to all languages, which is known as *integrated development environment (IDE)*.
 - ▶ *The purpose* of the IDE is to enable the developer to do as much as possible with visual tools, before writing code.
 - ▶ The IDE provides tools for designing, executing, and debugging your applications.
- 



The IDE Components

- ▶ **The IDE Menu**

The IDE main menu provides the following commands, which lead to submenus.

- ▶ **File Menu**

The File menu contains commands for opening and saving projects, or project items, as well as the commands for adding new or existing items to the current project.

- ▶ **Edit Menu**

The Edit menu contains the usual editing commands. Among the commands of the Edit menu are the Advanced command and the IntelliSense command.

- ▶ **View Menu**

This menu contains commands to display any toolbar or window of the IDE.

▶ **Project Menu**

This menu contains commands for adding items to the current project (an item can be a form, a file, a component, even another project). The last option in this menu is the Set As StartUp Project command, which lets you specify which of the projects in a multiproject solution is the startup project (the one that will run when you press F5).

▶ **Build Menu**

The Build menu contains commands for building (compiling) your project. The two basic commands in this menu are the Build and Rebuild All commands. The Build command compiles (builds the executable) of the entire solution, but it doesn't compile any components of the project that haven't changed since the last build. The Rebuild All command does the same, but it clears any existing files and builds the solution from scratch.

▶ **Debug Menu**

This menu contains commands to start or end an application, as well as the basic debugging tools



- ▶ **Data Menu**

This menu contains commands you will use with projects that access data.

- ▶ **Format Menu**

The Format menu, which is visible only while you design a Windows or Web form, contains commands for aligning the controls on the form.

- ▶ **Tools Menu**

This menu contains a list of tools

- ▶ **Window Menu**

This is the typical Window menu of any Windows application. In addition to the list of open windows, it also contains the Hide command, which hides all Toolboxes and devotes the entire window of the IDE to the code editor or the Form Designer. The Toolboxes don't disappear completely.

They're all retracted, and you can see their tabs on the left and right edges of the IDE window. To expand a Toolbox, just hover the mouse pointer over the corresponding tab.

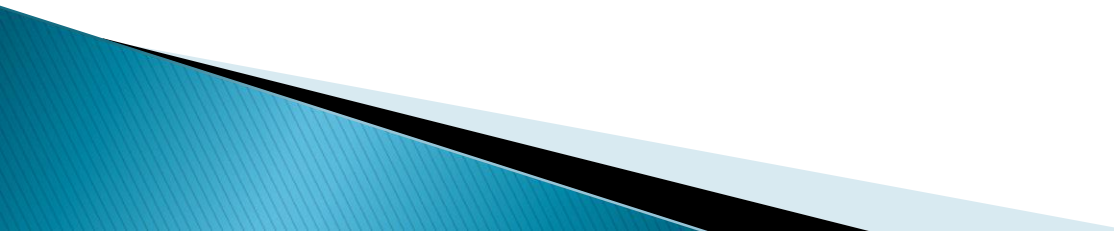
▶ **Help Menu**

This menu contains the various help options. The Dynamic Help command opens the Dynamic Help window, which is populated with topics that apply to the current operation. The Index command opens the Index window, where you can enter a topic and get help on the specific topic

▶ **The Toolbox Window**

Here you will find all the controls you can use to build your application's interface.

This window contains these tabs:

- ❖ Crystal Reports
 - ❖ Data
 - ❖ XML Schema
 - ❖ Dialog Editor
 - ❖ Web Forms
 - ❖ Components
 - ❖ Windows Forms
 - ❖ HTML
 - ❖ Clipboard Ring
 - ❖ General
- 


▶ **The Solution Explorer**

This window contains a list of the items in the current solution. A solution may contain multiple projects, and each project may contain multiple items. The Solution Explorer displays a hierarchical list of all the components, organized by project.

▶ **The Properties Window**

This window (also known as the Property Browser) displays all the properties of the selected component and their settings.

Every time you place a control on a form, you switch to this window to adjust the appearance of the control on the form, and you have already seen how to manipulate the properties of a control through the Properties window.



▶ The Output Window

The Output window is where many of the tools, including the compiler, send their output. Every time you start an application, a series of messages is displayed on the Output window. These messages are generated by the compiler, and you need not understand them at this point.

If the Output window is not visible, select the View ➤ Other Windows ➤ Output command from the menu.

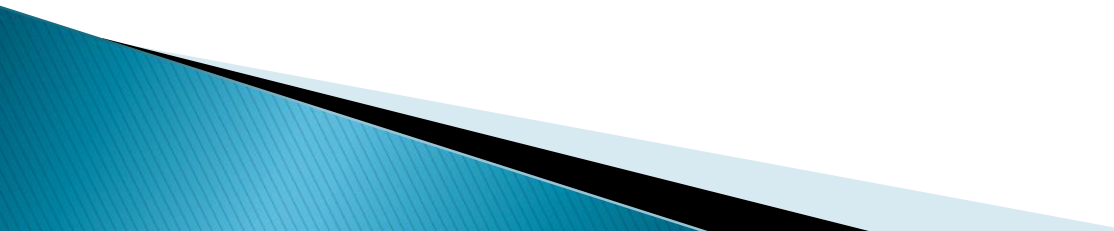
▶ The Command Window

While testing a program, you can interrupt its execution by inserting a *breakpoint*. When the *breakpoint* is reached, the program's execution is suspended and you can execute a statement in the Command window.

Any statement that can appear in your VB code can also be executed in the Command window.

▶ **The Task List Window**

This window is usually populated by the compiler with error messages, if the code can't be successfully compiled. You can double-click an error message in this window, and the IDE will take you to the line with the statement in error—which you should fix.



Variables

- ▶ In Visual Basic, as in any other programming language, variables store values during a program's execution.
- ▶ A variable has a name and a value.
- ▶ Syntax:
 - To declare a variable, use the Dim statement followed by the variable's name, the As keyword,
 - ▶ and its type, as follows:
 - ❖ Dim num As Integer
 - ❖ Dim expirydate As Date
 - ❖ Dim str As String
 - ▶ In VB.NET you can declare multiple variables of the same type without having to repeat each variable's type. The following statement, for instance, will create three Integer variables:
Dim width, depth, height As Integer

- ▶ You can declare multiple variables of the same or different type in the same line, as follows:
Dim Qty As Integer, Amount As Decimal, CardNum As String
- ▶ **Variable–Naming Conventions**
- ▶ When declaring variables, you should be aware of a few naming conventions.
- ▶ A variable's name: Must begin with a letter.
- ▶ Can't contain embedded periods. Except for certain characters used as data type identifiers, the only special character that can appear in a variable's name is the underscore character.
- ▶ Mustn't exceed 255 characters.
- ▶ Must be unique within its scope. This means that you can't have two identically named variables in the same subroutine, but you can have a variable named *counter* in many different subroutines.
- ▶ Variable names in VB.NET are case–insensitive: The variable names *myAge*, *myage*, and *MYAGE* all refer to the same variable in your code.

- ▶ **Variable Initialization**

- ▶ You can also initialize variables in the same line that declares them. The following line declares an Integer variable and initializes it to 3,045:

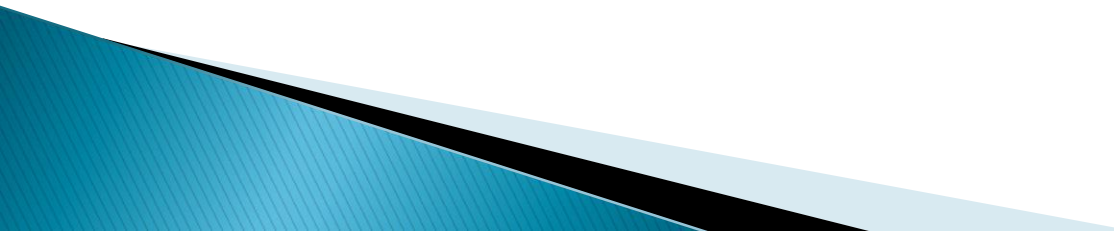
Dim distance As Integer = 3045

- ▶ This statement is equivalent to the following statements:

Dim distance As Integer
distance = 3045

- ▶ **Types of Variables**

- ▶ Visual Basic recognizes the following five categories of variables:

- ❖ Numeric
 - ❖ String
 - ❖ Boolean
 - ❖ Date
 - ❖ Object
- 

Constants

- ▶ Some variables don't change value during the execution of a program.
- ▶ **Constants are processed faster than variables. When the program is running, the values of constants don't have to be looked up. The compiler substitutes constant names with their values, and the program executes faster.**
- ▶ Syntax:
Const constantname As type = value
- ▶ Constants also have a scope and can be Public or Private. The constant *pi*, for instance, is usually declared in a module as Public so that every procedure can access it:
- ▶ Public Const pi As Double = 3.14159265358979

Operators

An operator performs a function on one or more operands. For example, we add two variables with the "+" addition operator and store the result in a third variable with the "=" assignment operator like this: `int x + int y = int z`. The two variables (x ,y) are called operands.

- ▶ **Arithmetic Operators**
- ▶ Arithmetic operators are used to perform arithmetic operations that involve calculation of numeric values. The table below summarizes them:

Operator	Use
\wedge	Exponentiation
$-$	Negation (used to reverse the sign of the given value, exp - intValue)
$*$	Multiplication
$/$	Division
\backslash	Integer Division
Mod	Modulus Arithmetic
$+$	Addition
$-$	Subtraction

- ▶ **Concatenation Operators**
- ▶ Concatenation operators join multiple strings into a single string. There are two concatenation operators, + and & as summarized below:

Operator	Use
+	String Concatenation
&	String Concatenation

Comparison Operators

- ▶ A comparison operator compares operands and returns a logical value based on whether the comparison is true or not. The table below summarizes them:

Operator	Use
=	Equality
<>	Inequality
<	Less than
>	Greater than
>=	Greater than or equal to
<=	Less than or equal to

VB.NET Type Conversions or Casting

- ▶ Casting refers to the process of converting an expression from one data type to another. Conversion is based on type compatibility and data compatibility.

There are two types of conversions:

- ▶ Implicit Conversion (Widening Conversion)
- ▶ Explicit Conversion (Narrowing Conversion)

Implicit Conversion

A Widening conversion is one that casts data from a data type with a narrower range of possible values to a data type with a wider range of possible values.

In implicit conversion the compiler will make conversion for us without asking.

Byte -> Short -> Integer -> Long -> Decimal -> Single -> Double

is an example of data compatibility.

Compiler checks for type compatibility compilation.

- ▶ Visual Basic uses implicit cast to do widening conversion automatically.
- ▶ When you do arithmetic expression, Visual Basic does widening conversions implicitly so all operands have the widest data type used in the expression.

Example:–

```
Dim num As Integer
    Dim marks As Decimal = 34.75
    num = CInt(marks)
    Console.WriteLine("Converted value is: " &
num)
```

Boxing And Unboxing In VB.NET

- ▶ With Boxing and Unboxing one can link between value-types and reference-types by allowing any value of a value-type to be converted to and from type object.

Boxing

- ▶ Boxing is a mechanism in which value type is converted into reference type.
- ▶ It is implicit conversion process in which object type (super type) is used.
- ▶ In this process type and value both are stored in object type

Unboxing

Unboxing is a mechanism in which reference type is converted into value.

- ▶ It is explicit conversion process.

Example:-

```
Dim i As Integer = 10
```

```
Dim j As Integer
```

```
' boxing
```

```
Dim o As Object
```

```
o = i
```

```
'unboxing
```

```
j = CInt(o)
```

```
Console.WriteLine("value of o object : " & o)
```

```
Console.WriteLine("Value of j : " & j)
```

