



Unit 1 :Introduction to .Net Framework

Introduction to .NET Framework

.NET is a software framework which is designed and developed by Microsoft. The first version of the .Net framework was 1.0 which came in the year 2002. In easy words, it is a virtual machine for compiling and executing programs written in different languages like C#, VB.Net etc.

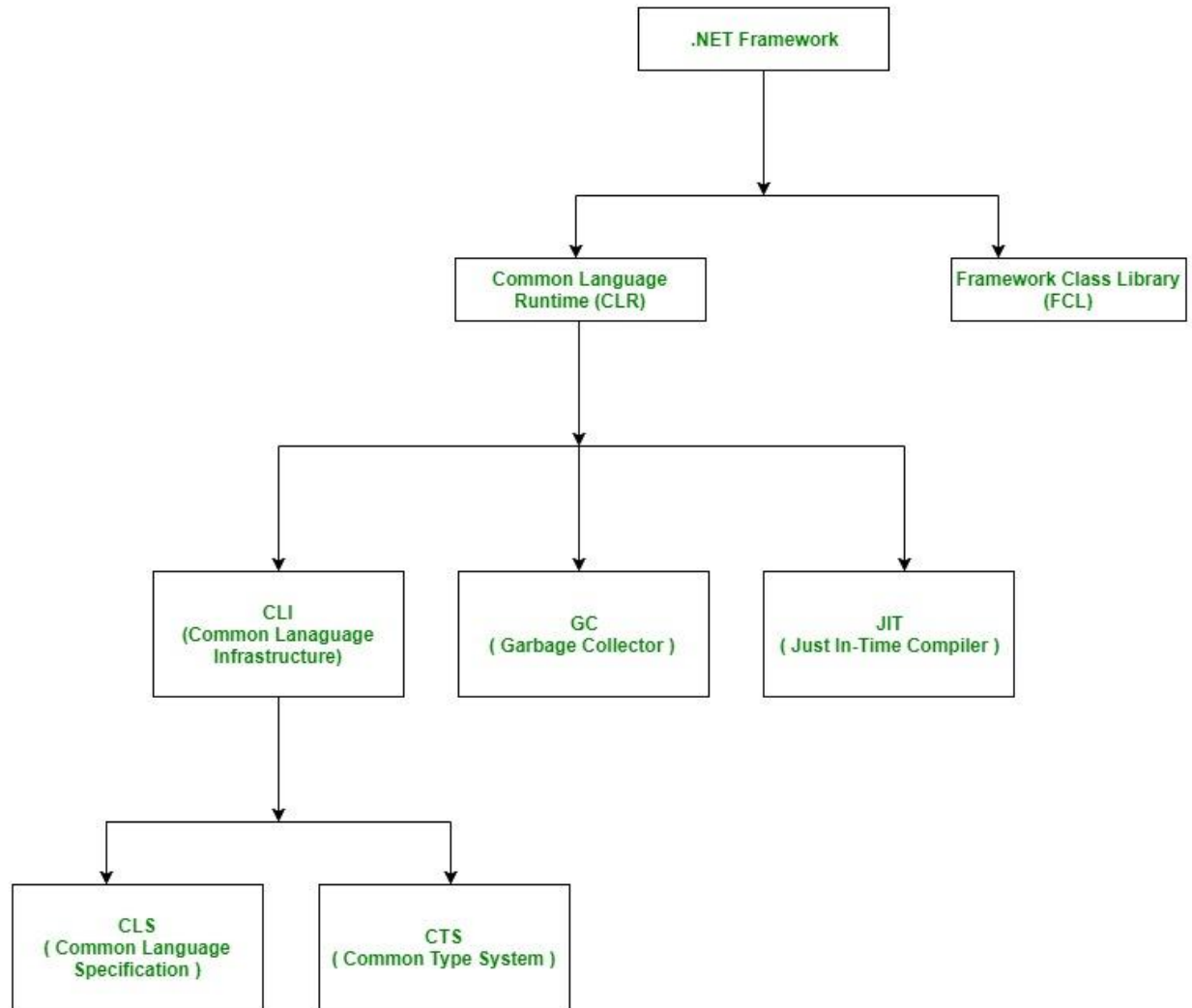
It is used to develop Form-based applications, Web-based applications, and Web services. There is a variety of programming languages available on the .Net platform, VB.Net and C# being the most common ones. It is used to build applications for Windows, phone, web, etc. It provides a lot of functionalities and also supports industry standards.

.NET Framework supports more than 60 programming languages in which 11 programming languages are designed and developed by Microsoft. The remaining Non-Microsoft Languages which are supported by .NET Framework but not designed and developed by Microsoft.

Common Language Runtime(CLR): CLR is the basic and Virtual Machine component of the .NET Framework. It is the run-time environment in the .NET Framework that runs the codes and helps in making the development process easier by providing the various services such as remoting, thread management, type-safety, memory management, robustness, etc.. Basically, it is responsible for managing the execution of .NET programs regardless of any .NET programming language. It also helps in the management of code, as code that targets the runtime is known as the Managed Code and code doesn't target to runtime is known as Unmanaged code.

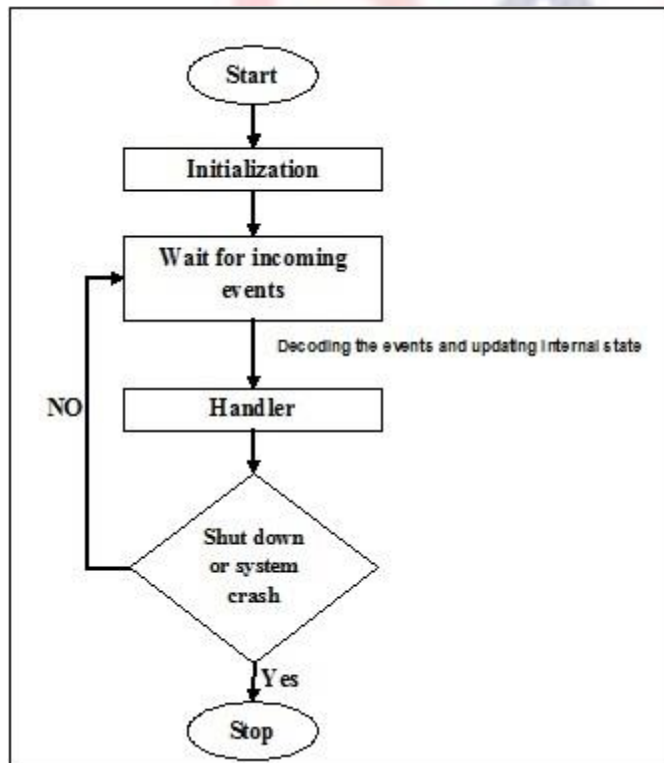
Framework Class Library(FCL): It is the collection of reusable, object-oriented class libraries and methods, etc that can be integrated with CLR. Also called the Assemblies. It is just like the header files in C/C++ and packages in the java. Installing .NET framework basically is the installation of CLR and FCL into the system. Below is the overview of .NET Framework

Is .NET application platform dependent or platform independent?



The combination of Operating System Architecture and CPU Architecture is known as the platform. Platform dependent means the programming language code will run only on particular Operating System. A .NET application is platform dependent because of the .NET framework which is only able to run on the Windows-based operating system. The .Net application is platform independent also because of Mono framework. Using Mono framework the .Net application can run on any Operating System including windows. Mono framework is a third party software developed by Novell Company which is now a part of Micro Focus Company. It is a paid framework

Event-driven programming focuses on events. Eventually, the flow of program depends upon events. Until now, we were dealing with either sequential or parallel execution model but the model having the concept of event-driven programming is called asynchronous model. Event-driven programming depends upon an event loop that is always listening for the new incoming events. The working of event-driven programming is dependent upon events. Once an event loops, then events decide what to execute and in what order. Following flowchart will help you understand how this works –



.Net Framework Architecture

The basic architecture of the .Net framework is as shown below.



.NET Components

The architecture of the .Net framework is based on the following key components;

1. Common Language Runtime

The "Common Language Infrastructure" or CLI is a platform on which the .Net programs are executed.

The CLR has the following key features:

Exception Handling - Exceptions are errors which occur when the application is executed.

Examples of exceptions are:

If an application tries to open a file on the local machine, but the file is not present.

If the application tries to fetch some records from a database, but the connection to the database is not valid.

Garbage Collection - Garbage collection is the process of removing unwanted resources when they are no longer required.

Examples of garbage collection are

A File handle which is no longer required. If the application has finished all operations on a file, then the file handle may no longer be required.

The database connection is no longer required. If the application has finished all operations on a database, then the database connection may no longer be required.

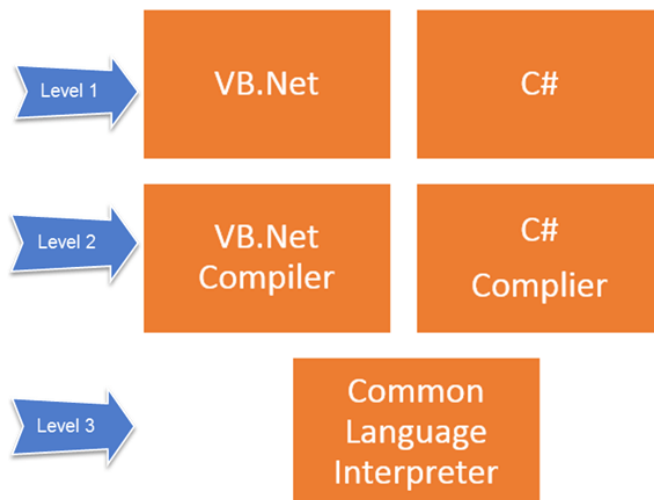
Working with Various programming languages –

As noted in an earlier section, a developer can develop an application in a variety of .Net programming languages.

Language - The first level is the programming language itself, the most common ones are VB.Net and C#.

Compiler – There is a compiler which will be separate for each programming language. So underlying the VB.Net language, there will be a separate VB.Net compiler. Similarly, for C#, you will have another compiler.

Common Language Interpreter – This is the final layer in .Net which would be used to run a .net program developed in any programming language. So the subsequent compiler will send the program to the CLI layer to run the .Net application.





2. Class Library

The .NET Framework includes a set of standard class libraries. A class library is a collection of methods and functions that can be used for the core purpose.

For example, there is a class library with methods to handle all file-level operations. So there is a method which can be used to read the text from a file. Similarly, there is a method to write text to a file.

Most of the methods are split into either the System.* or Microsoft.* namespaces. (The asterisk * just means a reference to all of the methods that fall under the System or Microsoft namespace)

A namespace is a logical separation of methods. We will learn these namespaces more in detail in the subsequent chapters.

3. Languages

The types of applications that can be built in the .Net framework is classified broadly into the following categories.

WinForms – This is used for developing Forms-based applications, which would run on an end user machine. Notepad is an example of a client-based application.

ASP.Net – This is used for developing web-based applications, which are made to run on any browser such as Internet Explorer, Chrome or Firefox.

The Web application would be processed on a server, which would have Internet Information Services Installed.

Internet Information Services or IIS is a Microsoft component which is used to execute an Asp.Net application.

The result of the execution is then sent to the client machines, and the output is shown in the browser.



ADO.Net – This technology is used to develop applications to interact with Databases such as Oracle or Microsoft SQL Server.

Microsoft always ensures that .Net frameworks are in compliance with all the supported Windows operating systems.

.Net Framework Design Principle

The following design principles of the .Net framework are what make it very relevant to create .Net based applications.

Interoperability - The .Net framework provides a lot of backward support. Suppose if you had an application built on an older version of the .Net framework, say 2.0. And if you tried to run the same application on a machine which had the higher version of the .Net framework, say 3.5. The application would still work. This is because with every release, Microsoft ensures that older framework versions gel well with the latest version.

Portability- Applications built on the .Net framework can be made to work on any Windows platform. And now in recent times, Microsoft is also envisioning to make Microsoft products work on other platforms, such as iOS and Linux.

Security - The .NET Framework has a good security mechanism. The inbuilt security mechanism helps in both validation and verification of applications. Every application can explicitly define their security mechanism. Each security mechanism is used to grant the user access to the code or to the running program.

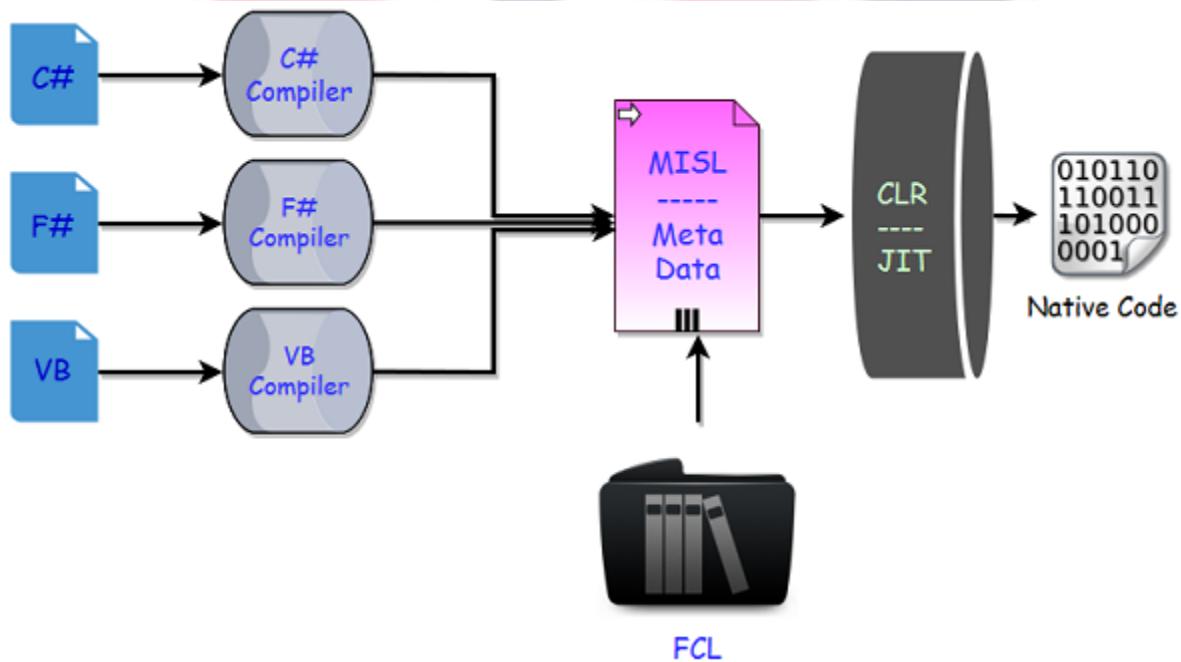
Memory management - The Common Language runtime does all the work or memory management. The .Net framework has all the capability to see those resources, which are not used by a running program. It would then release those resources accordingly. This is done via a program called the "Garbage Collector" which runs as part of the .Net framework.

The garbage collector runs at regular intervals and keeps on checking which system resources are not utilized, and frees them accordingly.

Simplified deployment - The .Net framework also have tools, which can be used to package applications built on the .Net framework. These packages can then be distributed to client machines. The packages would then automatically install the application.

In this chapter, we will understand the execution process of .NET Core and compare it with the .NET Framework. The managed execution process includes the following steps.

- Choosing a compiler
- Compiling your code to MSIL
- Compiling MSIL to native code
- Running code



Choosing a Compiler

- It is a multi-language execution environment, the runtime supports a wide variety of data types and language features.
- To obtain the benefits provided by the common language runtime, you must use one or more language compilers that target the runtime.

Compiling your code to MSIL

- Compiling translates your source code into Microsoft Intermediate Language (MSIL) and generates the required metadata.
- Metadata describes the types in your code, including the definition of each type, the signatures of each type's members, the members that your code references, and other data that the runtime uses at execution time.
- The runtime locates and extracts the metadata from the file as well as from framework class libraries (FCL) as needed during execution.



Compiling MSIL to Native Code

- At execution time, a just-in-time (JIT) compiler translates the MSIL into native code.
- During this compilation, code must pass a verification process that examines the MSIL and metadata to find out whether the code can be determined to be type safe.

Running Code

- The common language runtime provides the infrastructure that enables the execution to take place and services that can be used during execution.
- During execution, managed code receives services such as garbage collection, security, interoperability with unmanaged code, cross-language debugging support, and enhanced deployment and

Features of .NET Framework

- Common Language Runtime (CLR)
- .NET Framework Class Library (FCL)
- Interoperability
- Common Type System (CTS)
- Asynchronous Programming
- Portability
- High Performance
- Memory Management
- Security

IDE (integrated Development Environment)

- It is a software application that provides comprehensive facilities to computer programmers for software development.

In general IDE is a graphical user interface based workbench which helps the developer in building software applications by providing tools like a source code editor; build automation tools, a debugger etc

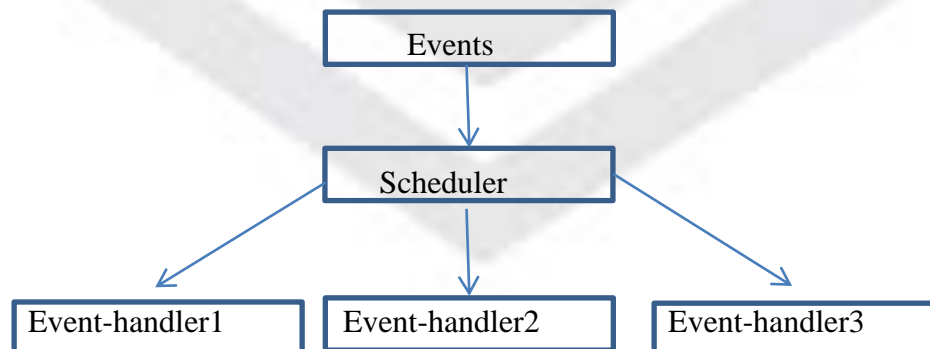
In the context of .NET based applications, visual studio is the most commonly used IDE.

VS.Net is just an editor, provided by Microsoft to help developers to write .NET programs easily. VS.Net provides:

1. Editor which automatically generate lot of code.
2. Allow developers to drag and drop control to a form.
3. Provide shortcuts to compile and build the application etc.

Event driven programming

- Changing the state of an object is known as an event. For example click on button, dragging mouse etc. Event handling is the mechanism that controls the event and decides what should happen if an event occurs.
- Event driven programming is a paradigm in which the flow of the program is determined by events (mouse clicks, key presses), sensor outputs, or messages from other programs/threads.
- In fig a simple event driven programming paradigm where each and every event go through a scheduler. Scheduler passes the code written inside the events to a particular event handler.





VB.net support event driven programming by following ways:

- A Visual studio IDE provides toolbox which is a window that gives us access to all controls, components, etc. User can easily drag and drop controls or tools on forms.
- A Visual studio IDE also provides editor which automatically generates codes for different events like click or press events. The programmer therefore concentrate on issues such as interface design, which involves adding controls such as command buttons, text boxes, and labels to standard forms
- Once the user interface is substantially complete, the programmer can add event-handling code to each control as required.
- Each event-handler is usually bound to a specific object or control on a form. Any additional subroutines, methods, or function procedures required are usually placed in a separate code module, and can be called from other parts of the program as and when needed.

NET Framework:

- Assemblies and namespaces are basic building blocks of the .net framework, They form the fundamental unit of deployment, version control, reuse , security permissions and more.

Assemblies:

- An assembly is a collection of types and resources that are built to work together and form a logical unit of functionality.

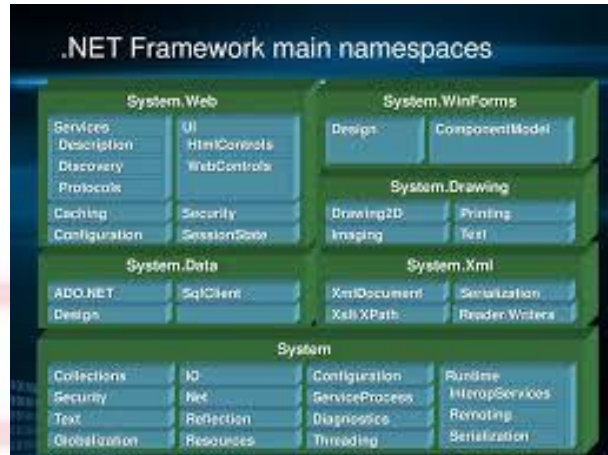


Manifest:

Manifest is the most important part of an Assembly. Manifest contains useful information, about an assembly which makes an assembly: self-describing,

The specifications in an assembly are collectively referred to as the assembly's the manifest contains:

1. The name of the assembly.
2. Version information for the assembly.
3. Security information for the assembly.
4. A list of all files that are part of the assembly.
5. Type reference information for the types specified in the assembly.
6. A list of other assemblies that are referenced by the assembly.
7. Custom information, such as user-friendly assembly title, description and product information,



2. System Namespaces:

- Namespace plays a fundamental role in the .Net framework. You cannot build a vb.net application without using classes from the .net system namespace.
- The .net framework class library consists of several thousand classes and other types, (such as interfaces, structures and enumerations) that are divided into several namespaces
- Most commonly used namespaces are as follows:

1. System.Data:

- System.Data and its nested namespaces, notably system.Data.oledb and system.data.sqlclient provide data access using ADO.NET
- The oledb and sqlclient namespaces are responsible for defining **data providers** that can connect to **a data source** retrieve data from a data source, write data back to a data source and execute commands against the data source.

ADO.net is an entirely new model for data access that is based on the disconnected dataset.

2. System.IO:

The system.IO namespace contains classes that provide a variety of input/output functionality, such as

- Manipulating directories and files
- Monitoring changes in directories and files
- Reading and writing single bytes, multi byte blocks or characters to and from streams.
- Reading and writing characters to and from strings
- Writing and reading data types and objects to and from streams
- Providing random access to files(file stream)



3. System.Text.RegularExpressions:

- The system.Text.RegularExpressions namespace contains classes that provide access to the .NET framework's regular expression engine.
- In its simplest form a regular expression is a text string that represents a pattern that other strings may or may not match.

4. System.Windows.Forms

- This namespace is the mother of all namespaces and is used for creating windows applications.
- The system.Windows.Forms namespace contains classes for creating windows based applications that take full advantage of the rich user interface features available in the Microsoft Windows operating system.
- In this namespace you will find the form class and many other controls that can be added to forms to create user interfaces.
- In fact each new form added to a vb.net Project contains the following line:
Imports System.Windows.Forms
- Fortunately Visual Studio provides the functionality of the system.Windows.Forms namespace to us as VB programmers, so we do not need to program directly against this namespace.

Architecture of .NET Framework

.NET Framework

- * Microsoft .NET (pronounced “dot net”) is a software component that runs on the Windows operating system.
- * .NET provides tools and libraries that enable developers to create Windows software much faster and easier.
- * The .NET Framework must be installed on a user's PC to run .NET applications.



NET Framework

.Net Framework

.NET is a framework to develop software applications. It is designed and developed by Microsoft and the first beta version released in 2000.

It is used to develop applications for web, Windows, phone. Moreover, it provides a broad range of functionalities and support.

This framework contains a large number of class libraries known as Framework Class Library (FCL). The software programs written in .NET are executed in the execution environment, which is called CLR (Common Language Runtime). These are the core and essential parts of the .NET framework.

This framework provides various services like memory management, networking, security, memory management, and type-safety.

The .Net Framework supports more than 60 programming languages such as C#, F#, VB.NET, J#, VC++, JScript.NET, APL, COBOL, Perl, Oberon, ML, Pascal, Eiffel, Smalltalk, Python, Cobra, ADA, etc.

Following is the .NET framework Stack that shows the modules and components of the Framework.

The .NET Framework is composed of four main components:

Common Language Runtime (CLR)

Framework Class Library (FCL),

Core Languages (WinForms, ASP.NET, and ADO.NET), and



Other Modules (WCF, WPF, WF, Card Space, LINQ, Entity Framework, Parallel LINQ, Task Parallel Library, etc.)

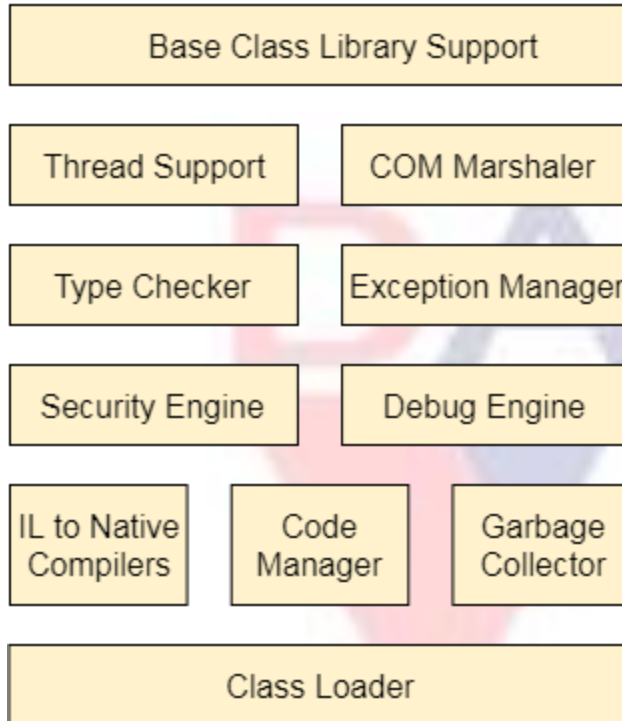
.NET APIs for Store/UWP apps		Task-Based Async Model		4.5 2012
Parallel LINQ		Task Parallel Library		4.0 2010
LINQ		Entity Framework		3.5 2007
WPF	WCF	WF	Card Space	3.0 2006
WinForms	ASP.NET	ADO.NET		.NET Framework 2.0 2005
Framework Class Library				
Common Language Runtime				

CLR (Common Language Runtime)

It is a program execution engine that loads and executes the program. It converts the program into native code. It acts as an interface between the framework and operating system. It does exception handling, memory management, and garbage collection. Moreover, it provides security, type-safety, interoperability, and portability. A list of CLR components are given below:

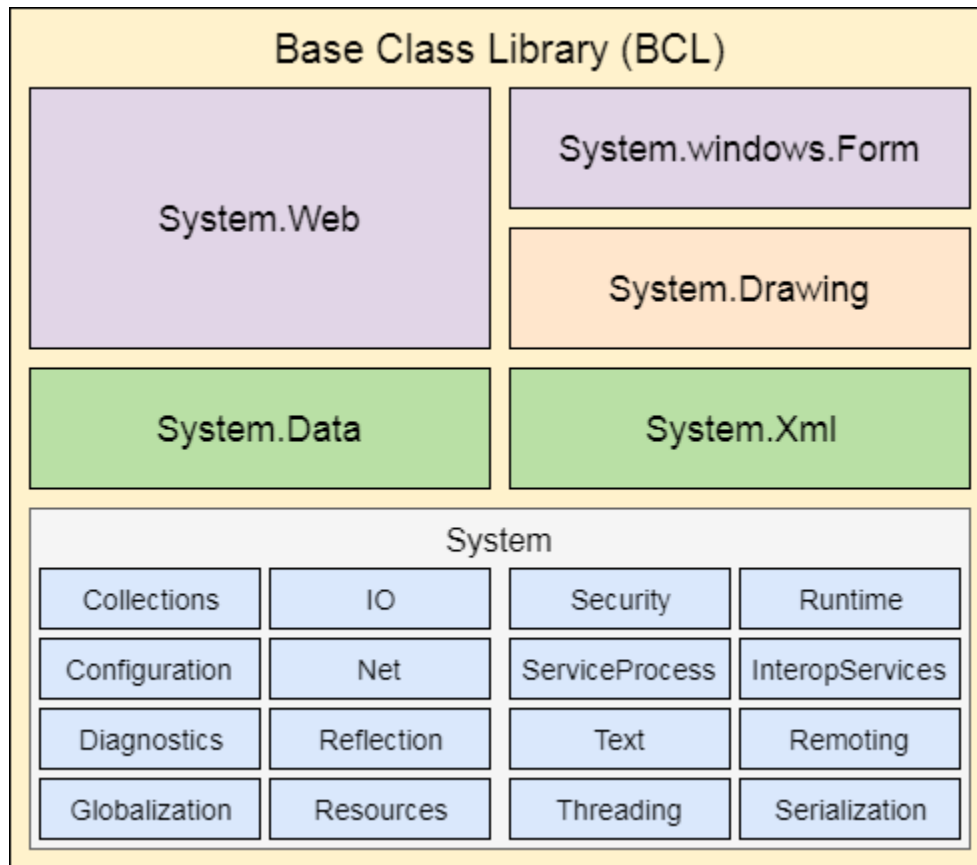


Common Language Runtime



FCL (Framework Class Library)

It is a standard library that is a collection of thousands of classes and used to build an application. The BCL (Base Class Library) is the core of the FCL and provides basic functionalities.



WinForms

Windows Forms is a smart client technology for the .NET Framework, a set of managed libraries that simplify common application tasks such as reading and writing to the file system.

ASP.NET

ASP.NET is a web framework designed and developed by Microsoft. It is used to develop websites, web applications, and web services. It provides a fantastic integration of HTML, CSS, and JavaScript. It was first released in January 2002.

ADO.NET



ADO.NET is a module of .Net Framework, which is used to establish a connection between application and data sources. Data sources can be such as SQL Server and XML. ADO .NET consists of classes that can be used to connect, retrieve, insert, and delete data.

WPF (Windows Presentation Foundation)

Windows Presentation Foundation (WPF) is a graphical subsystem by Microsoft for rendering user interfaces in Windows-based applications. WPF, previously known as "Avalon", was initially released as part of .NET Framework 3.0 in 2006. WPF uses DirectX.

WCF (Windows Communication Foundation)

It is a framework for building service-oriented applications. Using WCF, you can send data as asynchronous messages from one service endpoint to another.

WF (Workflow Foundation)

Windows Workflow Foundation (WF) is a Microsoft technology that provides an API, an in-process workflow engine, and a rehostable designer to implement long-running processes as workflows within .NET applications.

LINQ (Language Integrated Query)

It is a query language, introduced in .NET 3.5 framework. It is used to make the query for data sources with C# or Visual Basics programming languages.

Entity Framework

It is an ORM based open source framework which is used to work with a database using .NET objects. It eliminates a lot of developers effort to handle the database. It is Microsoft's recommended technology to deal with the database.



Parallel LINQ

Parallel LINQ or PLINQ is a parallel implementation of LINQ to objects. It combines the simplicity and readability of LINQ and provides the power of parallel programming.

It can improve and provide fast speed to execute the LINQ query by using all available computer capabilities.

Apart from the above features and libraries, .NET includes other APIs and Model to improve and enhance the .NET framework.

In 2015, Task parallel and Task parallel libraries were added. In .NET 4.5, a task-based asynchronous model was added.



Unit :2 Introduction to VB.Net

What is VB.Net?

VB.NET stands for Visual Basic.NET, and it is a computer programming language developed by Microsoft. It was first released in 2002 to replace Visual Basic 6. VB.NET is an object-oriented programming language. This means that it supports the features of object-oriented programming which include encapsulation, polymorphism, abstraction, and inheritance.

VB.NET Features

VB.NET comes loaded with numerous features that have made it a popular programming language amongst programmers worldwide. These features include the following:

- VB.NET is not case sensitive like other languages such as C++ and Java.
- It is an object-oriented programming language. It treats everything as an object.
- Automatic code formatting, XML designer, improved object browser etc.
- Garbage collection is automated.
- Support for Boolean conditions for decision making.
- Simple multithreading, allowing your apps to deal with multiple tasks simultaneously.
- Simple generics.
- A standard library.
- Events management.
- References. You should reference an external object that is to be used in a VB.NET application.
- Attributes, which are tags for providing additional information regarding elements that have been defined within a program.
- Windows Forms- you can inherit your form from an already existing form.

Advantages of VB.NET

The following are the pros/benefits you will enjoy for coding in VB.NET:

- Your code will be formatted automatically.
- You will use object-oriented constructs to create an enterprise-class code.
- You can create web applications with modern features like performance counters, event logs, and file system.
- You can create your web forms with much ease through the visual forms designer. You will also enjoy drag and drop capability to replace any elements that you may need.



- You can connect your applications to other applications created in languages that run on the .NET framework.
- You will enjoy features like docking, automatic control anchoring, and in-place menu editor all good for developing web applications.

Disadvantages of VB.NET

Below are some of the drawbacks/cons associated with VB.NET:

- VB.NET cannot handle pointers directly. This is a significant disadvantage since pointers are much necessary for programming. Any additional coding will lead to many CPU cycles, requiring more processing time. Your application will become slow.
- VB.NET is easy to learn. This has led to a large talent pool. Hence, it may be challenging to secure a job as a VB.NET programmer.

Data types

Data types refer to an extensive system used for declaring variables or functions of different types. The type of a variable determines how much space it occupies in storage and how the bit pattern stored is interpreted.

Data Types Available in VB.Net

VB.Net provides a wide range of data types. The following table shows all the data types available –

Data Type	Storage Allocation	Value Range
Boolean	Depends on implementing platform	True or False
Byte	1 byte	0 through 255 (unsigned)
Char	2 bytes	0 through 65535 (unsigned)
Date	8 bytes	0:00:00 (midnight) on January 1, 0001 through 11:59:59 PM on December 31, 9999



Decimal	16 bytes	0 through +/- 79,228,162,514,264,337,593,543,950,335 (+/- 7.9...E+28) with no decimal point; 0 through +/- 7.9228162514264337593543950335 with 28 places to the right of the decimal
Double	8 bytes	-1.79769313486231570E+308 through - 4.94065645841246544E-324, for negative values 4.94065645841246544E-324 through 1.79769313486231570E+308, for positive values
Integer	4 bytes	-2,147,483,648 through 2,147,483,647 (signed)
Long	8 bytes	-9,223,372,036,854,775,808 through 9,223,372,036,854,775,807(signed)
Object	4 bytes on 32-bit platform 8 bytes on 64-bit platform	Any type can be stored in a variable of type Object
SByte	1 byte	-128 through 127 (signed)
Short	2 bytes	-32,768 through 32,767 (signed)
Single	4 bytes	-3.4028235E+38 through -1.401298E-45 for negative values; 1.401298E-45 through 3.4028235E+38 for positive values
String	Depends on implementing platform	0 to approximately 2 billion Unicode characters
UInteger	4 bytes	0 through 4,294,967,295 (unsigned)
ULong	8 bytes	0 through 18,446,744,073,709,551,615 (unsigned)
User-Defined	Depends on implementing platform	Each member of the structure has a range determined by its data type and independent of the ranges of the other members



UShort	2 bytes	0 through 65,535 (unsigned)
--------	---------	-----------------------------

VB.Net is an object-oriented programming language. In Object-Oriented Programming methodology, a program consists of various objects that interact with each other by means of actions. The actions that an object may take are called methods. Objects of the same kind are said to have the same type or, more often, are said to be in the same class.

When we consider a VB.Net program, it can be defined as a collection of objects that communicate via invoking each other's methods. Let us now briefly look into what do class, object, methods and instance variables mean.

- **Object** – Objects have states and behaviors. Example: A dog has states - color, name, breed as well as behaviors - wagging, barking, eating, etc. An object is an instance of a class.
- **Class** – A class can be defined as a template/blueprint that describes the behaviors/states that objects of its type support.
- **Methods** – A method is basically a behavior. A class can contain many methods. It is in methods where the logics are written, data is manipulated and all the actions are executed.
- **Instance Variables** – Each object has its unique set of instance variables. An object's state is created by the values assigned to these instance variables.

A Rectangle Class in VB.Net

For example, let us consider a Rectangle object. It has attributes like length and width. Depending upon the design, it may need ways for accepting the values of these attributes, calculating area and displaying details.

Let us look at an implementation of a Rectangle class and discuss VB.Net basic syntax on the basis of our observations in it –

[Live Demo](#)

```
Imports System
Public Class Rectangle
    Private length As Double
    Private width As Double

    'Public methods
    Public Sub AcceptDetails()
        length = 4.5
```



```
width = 3.5
End Sub

Public Function GetArea() As Double
    GetArea = length * width
End Function
Public Sub Display()
    Console.WriteLine("Length: {0}", length)
    Console.WriteLine("Width: {0}", width)
    Console.WriteLine("Area: {0}", GetArea())

End Sub

Shared Sub Main()
    Dim r As New Rectangle()
    r.Acceptdetails()
    r.Display()
    Console.ReadLine()
End Sub
End Class
```

When the above code is compiled and executed, it produces the following result –

```
Length: 4.5
Width: 3.5
Area: 15.75
```

In previous chapter, we created a Visual Basic module that held the code. Sub Main indicates the entry point of VB.Net program. Here, we are using Class that contains both code and data. You use classes to create objects. For example, in the code, r is a Rectangle object.

An object is an instance of a class –

```
Dim r As New Rectangle()
```

A class may have members that can be accessible from outside class, if so specified. Data members are called fields and procedure members are called methods.

Shared methods or **static** methods can be invoked without creating an object of the class. Instance methods are invoked through an object of the class –

```
Shared Sub Main()
    Dim r As New Rectangle()
    r.Acceptdetails()
    r.Display()
    Console.ReadLine()
End Sub
```



Identifiers

An identifier is a name used to identify a class, variable, function, or any other user-defined item. The basic rules for naming classes in VB.Net are as follows –

- A name must begin with a letter that could be followed by a sequence of letters, digits (0 - 9) or underscore. The first character in an identifier cannot be a digit.
- It must not contain any embedded space or symbol like ? - +! @ # % ^ & * () [] { } . ; : " ' / and \. However, an underscore (_) can be used.
- It should not be a reserved keyword.

VB.Net Keywords

The following table lists the VB.Net reserved keywords –

AddHandle r	AddressOf	Alias	And	AndAlso	As	Boolean
ByRef	Byte	ByVal	Call	Case	Catch	CBool
CByte	CChar	CDate	CDec	CDbl	Char	CInt
Class	CLng	CObj	Const	Continue	CSByte	CShort
CSng	CStr	CType	CUInt	CULng	CUShort	Date
Decimal	Declare	Default	Delegate	Dim	DirectCast	Do
Double	Each	Else	ElseIf	End	End If	Enum
Erase	Error	Event	Exit	False	Finally	For
Friend	Function	Get	GetType	GetXML	Global	GoTo



				Namespace		
Handles	If	Implements	Imports	In	Inherits	Integer
Interface	Is	IsNot	Let	Lib	Like	Long
Loop	Me	Mod	Module	MustInherit	MustOverride	MyBase
MyClass	Namespace	Narrowing	New	Next	Not	Nothing
Not Inheritable	Not Overridable	Object	Of	On	Operator	Option
Optional	Or	OrElse	Overloads	Overridable	Overrides	ParamArray
Partial	Private	Property	Protected	Public	RaiseEvent	ReadOnly
ReDim	REM	Remove Handler	Resume	Return	SByte	Select
Set	Shadows	Shared	Short	Single	Static	Step
Stop	String	Structure	Sub	SyncLock	Then	Throw
To	True	Try	TryCast	TypeOf	UInteger	While



Widening	With	WithEvents	WriteOnly	Xor	
----------	------	------------	-----------	-----	--

Operator	Description	Example
^	Raises one operand to the power of another	B^A will give 49
+	Adds two operands	A + B will give 9
-	Subtracts second operand from the first	A - B will give -5
*	Multiplies both operands	A * B will give 14
/	Divides one operand by another and returns a floating point result	B / A will give 3.5
\	Divides one operand by another and returns an integer result	B \ A will give 3
MOD	Modulus Operator and remainder of after an integer division	B MOD A will give 1

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. VB.Net is rich in built-in operators and provides following types of commonly used operators –

- Arithmetic Operators
- Comparison Operators
- Logical/Bitwise Operators
- Bit Shift Operators
- Assignment Operators
- Miscellaneous Operators

This tutorial will explain the most commonly used operators.

Arithmetic Operators

Following table shows all the arithmetic operators supported by VB.Net. Assume variable **A** holds 2 and variable **B** holds 7, then –

Show Examples

Comparison Operators

Following table shows all the comparison operators supported by VB.Net. Assume variable **A** holds 10 and variable **B** holds 20, then –

Show Examples

Operator	Description	Example
=	Checks if the values of two operands are equal or not; if yes, then condition becomes true.	(A = B) is not true.
<>	Checks if the values of two operands are equal or not; if values are not equal, then condition becomes true.	(A <> B) is true.

>	Checks if the value of left operand is greater than the value of right operand; if yes, then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand; if yes, then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand; if yes, then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand; if yes, then condition becomes true.	(A <= B) is true.

Apart from the above, VB.Net provides three more comparison operators, which we will be using in forthcoming chapters; however, we give a brief description here.

- **Is Operator** – It compares two object reference variables and determines if two object references refer to the same object without performing value comparisons. If object1 and object2 both refer to the exact same object instance, result is **True**; otherwise, result is False.
- **IsNot Operator** – It also compares two object reference variables and determines if two object references refer to different objects. If object1 and object2 both refer to the exact same object instance, result is **False**; otherwise, result is True.
- **Like Operator** – It compares a string against a pattern.

Logical/Bitwise Operators

Following table shows all the logical operators supported by VB.Net. Assume variable A holds Boolean value True and variable B holds Boolean value False, then –

Show Examples

Operator	Description	Example
----------	-------------	---------



And	It is the logical as well as bitwise AND operator. If both the operands are true, then condition becomes true. This operator does not perform short-circuiting, i.e., it evaluates both the expressions.	(A And B) is False.
Or	It is the logical as well as bitwise OR operator. If any of the two operands is true, then condition becomes true. This operator does not perform short-circuiting, i.e., it evaluates both the expressions.	(A Or B) is True.
Not	It is the logical as well as bitwise NOT operator. Use to reverses the logical state of its operand. If a condition is true, then Logical NOT operator will make false.	Not(A And B) is True.
Xor	It is the logical as well as bitwise Logical Exclusive OR operator. It returns True if both expressions are True or both expressions are False; otherwise it returns False. This operator does not perform short-circuiting, it always evaluates both expressions and there is no short-circuiting counterpart of this operator.	A Xor B is True.
AndAlso	It is the logical AND operator. It works only on Boolean data. It performs short-circuiting.	(A AndAlso B) is False.
OrElse	It is the logical OR operator. It works only on Boolean data. It performs short-circuiting.	(A OrElse B) is True.
IsFalse	It determines whether an expression is False.	
IsTrue	It determines whether an expression is True.	

Bit Shift Operators

We have already discussed the bitwise operators. The bit shift operators perform the shift operations on binary values. Before coming into the bit shift operators, let us understand the bit operations.

Bitwise operators work on bits and perform bit-by-bit operations. The truth tables for $\&$, $|$, and \wedge are as follows –

p	q	p & q	p q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

Assume if A = 60; and B = 13; now in binary format they will be as follows –

A = 0011 1100

B = 0000 1101

A&B = 0000 1100

A|B = 0011 1101

A^B = 0011 0001

~A = 1100 0011

We have seen that the Bitwise operators supported by VB.Net are And, Or, Xor and Not. The Bit shift operators are \gg and \ll for left shift and right shift, respectively.

Assume that the variable A holds 60 and variable B holds 13, then –

Show Examples

Operator	Description	Example
----------	-------------	---------



And	Bitwise AND Operator copies a bit to the result if it exists in both operands.	(A AND B) will give 12, which is 0000 1100
Or	Binary OR Operator copies a bit if it exists in either operand.	(A Or B) will give 61, which is 0011 1101
Xor	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A Xor B) will give 49, which is 0011 0001
Not	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(Not A) will give - 61, which is 1100 0011 in 2's complement form due to a signed binary number.
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240, which is 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15, which is 0000 1111



Assignment Operators

There are following assignment operators supported by VB.Net –

Show Examples

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	$C = A + B$ will assign value of $A + B$ into C
+=	Add AND assignment operator, It adds right operand to the left operand and assigns the result to left operand	$C += A$ is equivalent to $C = C + A$
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assigns the result to left operand	$C -= A$ is equivalent to $C = C - A$
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assigns the result to left operand	$C *= A$ is equivalent to $C = C * A$
/=	Divide AND assignment operator, It divides left operand with the right operand and assigns the result to left operand (floating point division)	$C /= A$ is equivalent to $C = C / A$
\=	Divide AND assignment operator, It divides left operand with the right operand	$C \setminus = A$ is equivalent



	and assigns the result to left operand (Integer division)	to $C = C \backslash A$
$\wedge =$	Exponentiation and assignment operator. It raises the left operand to the power of the right operand and assigns the result to left operand.	$C \wedge A$ is equivalent to $C = C \wedge A$
$\ll =$	Left shift AND assignment operator	$C \ll = 2$ is same as $C = C \ll 2$
$\gg =$	Right shift AND assignment operator	$C \gg = 2$ is same as $C = C \gg 2$
$\& =$	Concatenates a String expression to a String variable or property and assigns the result to the variable or property.	$\text{Str1} \& = \text{Str2}$ is same as $\text{Str1} = \text{Str1} \& \text{Str2}$

Miscellaneous Operators

There are few other important operators supported by VB.Net.

Show Examples

Operator	Description	Example
AddressOf	Returns the address of a procedure.	<code>AddHandler Button1.Click, AddressOf Button1_Click</code>



Await	It is applied to an operand in an asynchronous method or lambda expression to suspend execution of the method until the awaited task completes.	<pre>Dim result As res = Await AsyncMethodThatReturnsResult() Await AsyncMethod()</pre>
GetType	It returns a Type object for the specified type. The Type object provides information about the type such as its properties, methods, and events.	<pre>MsgBox(GetType(Integer).ToString())</pre>
Function Expression	It declares the parameters and code that define a function lambda expression.	<pre>Dim add5 = Function(num As Integer) num + 5 'prints 10 Console.WriteLine(add5(5))</pre>
If	It uses short-circuit evaluation to conditionally return one of two values. The If operator can be called with three arguments or with two arguments.	<pre>Dim num = 5 Console.WriteLine(If(num >= 0, "Positive", "Negative"))</pre>

Operators Precedence in VB.Net

Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has higher precedence than the addition operator –

For example, $x = 7 + 3 * 2$; here, x is assigned 13, not 20 because operator * has higher precedence than +, so it first gets multiplied with $3*2$ and then adds into 7.

Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

Show Examples

Operator	Precedence
----------	------------

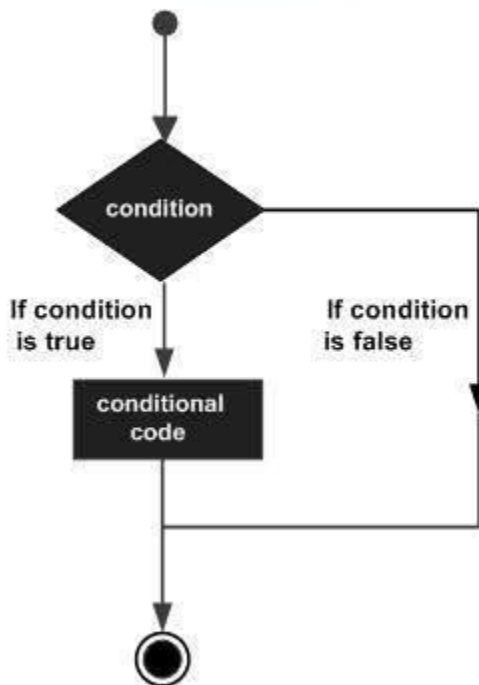


Await	Highest
Exponentiation (^)	
Unary identity and negation (+, -)	
Multiplication and floating-point division (*, /)	
Integer division (\)	
Modulus arithmetic (Mod)	
Addition and subtraction (+, -)	
Arithmetic bit shift (<<, >>)	
All comparison operators (=, <>, <, <=, >, >=, Is, IsNot, Like, TypeOf...Is)	
Negation (Not)	
Conjunction (And, AndAlso)	
Inclusive disjunction (Or, OrElse)	
Exclusive disjunction (Xor)	Lowest

Decision making structures require that the programmer specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the

condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.

Following is the general form of a typical decision making structure found in most of the programming languages –



VB.Net provides the following types of decision making statements. Click the following links to check their details.

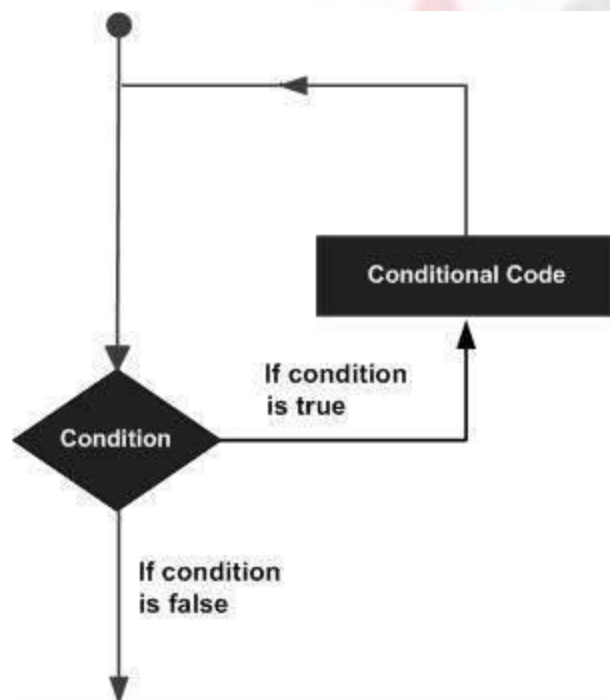
Statement	Description
If ... Then statement	An If...Then statement consists of a boolean expression followed by one or more statements.
If...Then...Else statement	An If...Then statement can be followed by an optional Else statement , which executes when the boolean expression is false.

<u>nested If statements</u>	You can use one If or Else if statement inside another If or Else if statement(s).
<u>Select Case statement</u>	A Select Case statement allows a variable to be tested for equality against a list of values.
<u>nested Select Case statements</u>	You can use one select case statement inside another select case statement(s).

There may be a situation when you need to execute a block of code several number of times. In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on.

Programming languages provide various control structures that allow for more complicated execution paths.

A loop statement allows us to execute a statement or group of statements multiple times and following is the general form of a loop statement in most of the programming languages –





VB.Net provides following types of loops to handle looping requirements. Click the following links to check their details.

Loop Type	Description
Do Loop	It repeats the enclosed block of statements while a Boolean condition is True or until the condition becomes True. It could be terminated at any time with the Exit Do statement.
For...Next	It repeats a group of statements a specified number of times and a loop index counts the number of loop iterations as the loop executes.
For Each...Next	It repeats a group of statements for each element in a collection. This loop is used for accessing and manipulating all elements in an array or a VB.Net collection.
While... End While	It executes a series of statements as long as a given condition is True.
With... End With	It is not exactly a looping construct. It executes a series of statements that repeatedly refer to a single object or structure.
Nested loops	You can use one or more loops inside any another While, For or Do loop.

Loop Control Statements

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

Control Statement	Description
Exit statement	Terminates the loop or select case statement and transfers execution to the statement immediately following the loop or select case.



Continue statement	Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.
GoTo statement	Transfers control to the labeled statement. Though it is not advised to use GoTo statement in your program.

VB.Net provides the following control statements. Click the following links to check their details.



Unit 3 Object Oriented Programming in vb.net

Introduction

Vb.net is an object oriented programming (OOP) language which supports various OOP concepts such as class, objects, methods, Procedure, functions, inheritance constructors and so on.

Visual basic comes with thousands of built in classes and we know that a form which we design is also an object, windows forms are objects

Which derived from the system.Windows.Form.Formclass and that our code is part of that class.

For example

```
Public class form1
```

```
Inherits system.Windows.Forms.Form
```

```
Private sub form1_Load(by val sender as system.object,by val e as system.Eventargs)Handles  
mybase.load
```

```
-----  
-----
```

```
End sub
```

```
End class.
```

For example

As already we created a textbox at runtime.where we have created an objects textbox1 of class textbox and various members of it are configured as shown below.

```
Private sub button1_click(by val sender as system.object,by val e as system.Eventargs) Handles  
button1.click
```

```
Dim textbox1 as new textbox()
```

```
Textbox1.size= new size(170,30)
```

```
Textbox1.location= new point(100,40)
```



```
Textbox1.text="Welcome to visual basic"
```

```
Me.controls.add(Textbox)
```

```
End sub
```

Class and object

A class is a user defined data type used to define new data types.

A Class may contain any combination of encapsulated data

VB.NET Classes and Object

A class is a group of different data members or objects with the same properties, processes, events of an object, and general relationships to other member functions. Furthermore, we can say that it is like a template or architect that tells what data and function will appear when it is included in a class object. For example, it represents the method and variable that will work on the object of the class.

Objects are the basic run-time units of a class. Once a class is defined, we can create any number of objects related to the class to access the defined properties and methods. For example, the Car is the Class name, and the speed, mileage, and wheels are attributes of the Class that can be accessed by the Object.

Creating the Class

We can create a class using the Class keyword, followed by the class name. And the body of the class ended with the statement End Class. Following is the general syntax for creating classes and objects in the [VB.NET programming language](http://www.dacc.edu.in).

1. [Access_Specifier] [Shadows] [MustInherit | NotInheritable] [Partial] Class ClassName
2. ' Data Members or Variable Declaration
3. ' Methods Name
4. ' Statement to be executed
5. End Class

Where,

- **Access_Specifier:** It defines the access levels of the class, such as Public, Private or Friend, Protected, Protected Friend, etc. to use the method. (It is an optional parameter).
- **Shadows:** It is an optional parameter. It represents the re-declaration of variables and hides an identical element name or set of overloaded elements in a base class.
- **MustInherit:** It is an optional parameter that specifies that the class can only be used as a base class, and the object will not directly access the base class or the abstract class.
- **NotInheritable:** It is also an optional parameter that representing the class not being used as a base class.
- **Partial:** As the name defines, a Partial represents the partial definition of the class (optional).
- **Implements:** It is used to specify interfaces from which the class inherits (optional).

My_program.vb

1. Public Class My_program
2. ' properties, method name, etc
3. ' Statement to be executed
4. End Class

In the above syntax, we have created a class with the name 'My_program' using the Class keyword.

The Syntax for creating an object

1. Dim Obj_Name As Class_Name = New Class_Name() ' Declaration of object
2. Obj_Name.Method_Name() ' Access a method using the object

In the above syntax, we have created an instance (**Obj_Name**) for the class **Class_Name**. By using the object name '**Obj_Name**' to access all the data members and the method name of **Class_Name**.

Let's create a program to find the Area and Parameter of a rectangle using the class and object in VB.NET.

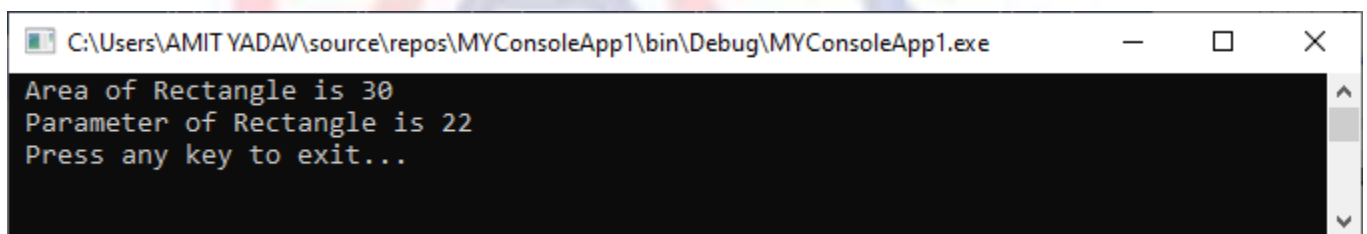
My_program.vb

1. Imports System
2. Module My_program



```
3. Sub Main()  
4.     Dim rect As Rectangle = New Rectangle() 'create an object  
5.     Dim rect2 As Rectangle = New Rectangle() 'create an object  
6.     Dim area, para As Integer  
7.  
8.     'rect specification  
9.     rect.setLength = (5)  
10.    rect.setBreadth= (6)  
11.  
12.    'rect2 specification  
13.    rect2.setLength = (5)  
14.    rect2.setBreadth = (10)  
15.  
16.    'Area of rectangle  
17.    area = rect.length * rect.Breadth  
18.    'area = rect.GetArea()  
19.    Console.WriteLine(" Area of Rectangle is {0}", area)  
20.  
21.    'Parameter of rectangle  
22.    'para = rect.GetParameter()  
23.    para = 2 (rect2.length + rect.Breadth)  
24.    Console.WriteLine(" Parameter of Rectangle is {0}", para)  
25.    Console.WriteLine(" Press any key to exit...")  
26.    Console.ReadKey()  
27. End Sub  
28. Public Class Rectangle  
29.     Public length As Integer  
30.     Public Breadth As Integer  
31.  
32.     Public Sub setLength(ByVal len As Integer)  
33.         length = len  
34.     End Sub  
35.  
36.     Public Sub setBreadth(ByVal bre As Integer)  
37.         Breadth = bre  
38.     End Sub  
39.     Public Function GetArea() As Integer  
40.         Return length * Breadth
```

```
41. End Function
42.
43. Public Function GetParameter() As Integer
44.     Return 2 * (length + Breadth)
45. End Function
46. End Class
47. End Module
```

Output:

```
C:\Users\AMIT YADAV\source\repos\MYConsoleApp1\bin\Debug\MYConsoleApp1.exe
Area of Rectangle is 30
Parameter of Rectangle is 22
Press any key to exit...
```

Member Function

The member function of a class is used to define the structure of member inside the definition of the class. It can be accessed by all defined objects of the class and operated on the data member. Furthermore, member variables are the attributes of an object to be implemented to a member function. And we can access member variables using the public member function.

Constructor and Destructor

In VB.NET, the constructor is a special method that is implemented when an object of a particular class is created. Constructor is also useful for creating and setting default values for a new object of a data member.

When we create a class without defining a constructor, the compiler automatically creates a default constructor. In this way, there is a constructor that is always available in every class. Furthermore, we can create more than one constructor in a class with the use of **New** keyword but with the different parameters, and it does not return anything.

Default Constructor: In VB.NET, when we create a constructor without defining an argument, it is called a default constructor.

VB.NET Default Constructor Syntax

The following is the syntax for creating a constructor using the New keyword in VB.NET.

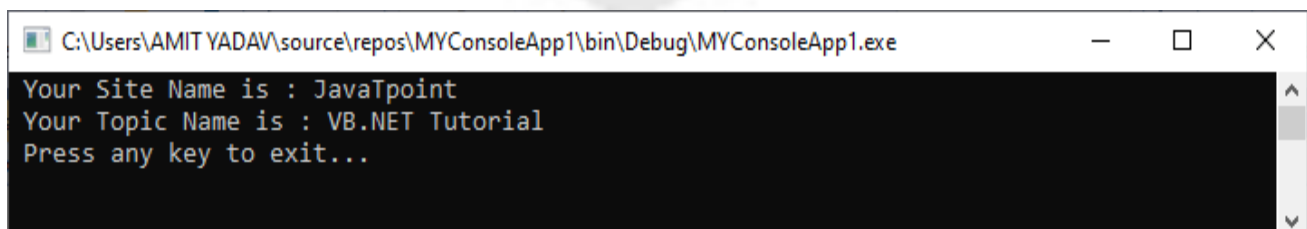
1. Public Class MyClass
2. ' Creates a Constructor using the New
3. Public Sub New()
4. 'Statement to be executed
5. End Sub
6. End Class

Let's create a program to define the default constructor in a VB.NET programming language.

Default_Const.vb

1. Imports System
2. Module Default_Const
3. Class Tutor
4. Public name As String
5. Public topic As String
6. ' Create a **default** constructor
7. Public Sub New()
8. name = "JavaTpoint"
9. topic = "VB.NET Tutorial"
10. End Sub
11. End Class
12. Sub Main()
13. ' The constructor will automatically call when the instance of the **class** is created
14. Dim tutor As Tutor = New Tutor() ' Create an object as a tutor
15. Console.WriteLine(" Your Site Name is : {0}", tutor.name)
16. Console.WriteLine(" Your Topic Name is : {0}", tutor.topic)
17. Console.WriteLine(" Press any key to exit...")
18. Console.ReadKey()
19. End Sub
20. End Module

Output:



```
C:\Users\AMIT YADAV\source\repos\MYConsoleApp1\bin\Debug\MYConsoleApp1.exe
Your Site Name is : JavaTpoint
Your Topic Name is : VB.NET Tutorial
Press any key to exit...
```


In the above example, we created a class '**Tutor**' and defined a default constructor method with **New()** keyword without passing any arguments. When the object (tutor) is created, the default constructor is called into the class.

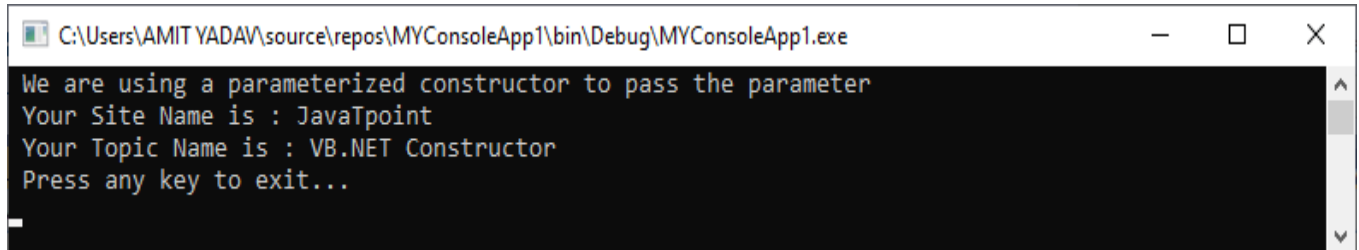
Parameterized Constructor

In VB.NET, when we pass one or more arguments to a constructor, the constructor is known as a parameterized constructor. And the object of the class should be initialized with arguments when it is created.

Let's create a program to use the parameterized constructor to pass the argument in a class.

Para_Const.vb

```
1. Imports System
2. Module Para_Const
3.     Class Tutor
4.         Public name As String
5.         Public topic As String
6.         ' Create a parameterized constructor to pass parameter
7.         Public Sub New(ByVal a As String, ByVal b As String)
8.             name = a
9.             topic = b
10.            Console.WriteLine(" We are using a parameterized constructor to pass the parameter")
11.        End Sub
12.    End Class
13.    Sub Main()
14.        ' The constructor will automatically call when the instance of the class is created
15.        Dim tutor As Tutor = New Tutor("JavaTpoint", "VB.NET Constructor")
16.        Console.WriteLine(" Your Site Name is : {0}", tutor.name)
17.        Console.WriteLine(" Your Topic Name is : {0}", tutor.topic)
18.        Console.WriteLine(" Press any key to exit...")
19.        Console.ReadKey()
20.    End Sub
21. End Module
```

Output:

```
C:\Users\AMIT YADAV\source\repos\MYConsoleApp1\bin\Debug\MYConsoleApp1.exe
We are using a parameterized constructor to pass the parameter
Your Site Name is : JavaTpoint
Your Topic Name is : VB.NET Constructor
Press any key to exit...
```

VB.NET Destructor

In VB.NET, Destructor is a special function that is used to destroy a class object when the object of the class goes out of scope. It can be represented as the **Finalize()** method and does not accept any parameter nor return any value. Furthermore, it can be called automatically when a class object is not needed.

VB.NET Destructor Syntax

Destructor using the Finalize() method in VB.NET.

1. Class My_Destructor
2. ' define the destructor
3. Protected Overrides Sub Finalize()
4. ' Statement or code to be executed
5. End Sub
6. End Class

Let's create a program to use the Finalize() method in VB.NET Destructor.

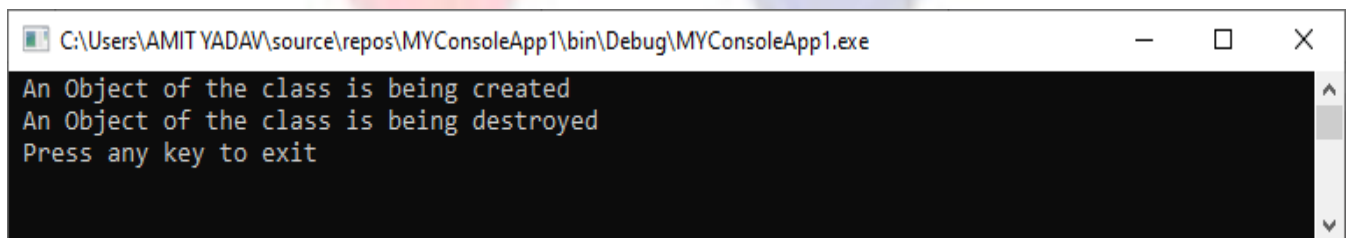
Destruct.vb

1. Imports System
2. Module Destruct
3. Class Get_Destroy
4. Public Sub New()
5. Console.WriteLine(" An Object of the class is being created")
6. End Sub
7. ' Use Finalize() method of Destructor
8. Protected Overrides Sub Finalize()
9. Console.WriteLine(" An Object of the class is being destroyed")



```
10. Console.WriteLine(" Press any key to exit")
11. End Sub
12. End Class
13.
14. Sub Main()
15. Get_Details() ' After invoking the Get_Details() method, garbage collector is called automat
    ically
16. GC.Collect()
17. Console.ReadKey()
18. End Sub
19. Public Sub Get_Details()
20. Dim dest As Get_Destroy = New Get_Destroy()
21. End Sub
22. End Module
```

Output:



Constructor Overloading

In VB.NET, the overloading of constructors is a concept in which we can overload constructors by defining more than one constructor with the same name but with different parameter lists to perform different tasks.

Let's create a program to use the Constructor Overloading in a class.

Const_Over.vb

```
1. Imports System
2. Module Const_Over
3. Class Details
4. Public name As String
5. Public course As String
6. ' Define Default Constructor
```



```
7. Public Sub New()  
8.     name = "Prince"  
9.     course = "Coputer Science"  
10.    Console.WriteLine(" Uses of Overloading Constructor")  
11. End Sub  
12. ' Create a parametrized constructor  
13. Public Sub New(ByVal a As String, ByVal b As String)  
14.     name = a  
15.     course = b  
16. End Sub  
17. End Class  
18. Sub Main()  
19.     ' Called default constructor  
20.     Dim det As Details = New Details()  
21.     ' Called the parametrized constructor  
22.     Dim det1 As Details = New Details("Peter", "Knowledge of Data Mining")  
23.     Console.WriteLine(" Your Details are: Name : {0} and Course : {1}", det.name, det.course)  
  
24.     Console.WriteLine(" Your Overloaded Details are: Name : {0} and Course :{1}", det1.name, det1.course)  
25.     Console.WriteLine(" Press any key to exit...")  
26.     Console.ReadKey()  
27. End Sub  
28. End Module
```

Output:

```
C:\Users\AMIT YADAV\source\repos\MYConsoleApp1\bin\Debug\MYConsoleApp1.exe  
Uses of Overloading Constructor  
Your Details are: Name : Prince and Course : Coputer Science  
Your Overloaded Details are: Name : Peter and Course :Knowledge of Data Mining  
Press any key to exit...
```

Inheritance

In VB.NET, **Inheritance** is the process of creating a new class by inheriting properties and functions from the old class and extending the functionality of the existing class. It also provides a reusable and faster implementation time of the code. When we create a derived or inherited



class, inheritance allows us to inherit all the properties of the existing class. The old class is known as the **base class**, and the inherited class is known as the **derived class**.

Inheritance Syntax

The following is the syntax of Inheritance in VB.NET.

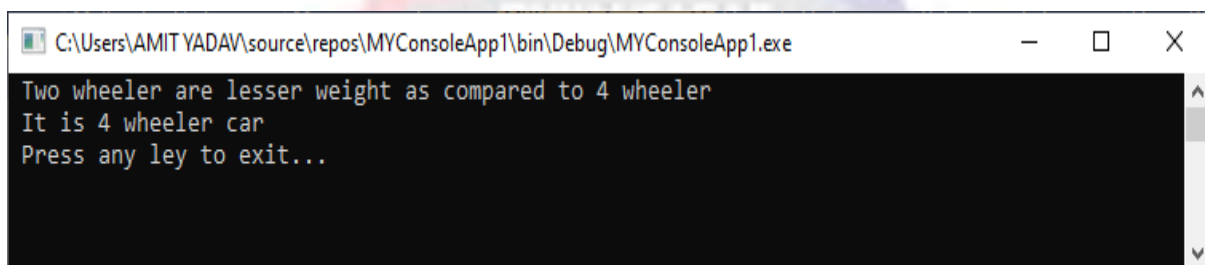
1. <access_modifier> Class <Name_of_baseClass>
2. ' Implementation of base **class**
3. End Class
4. <access_modifier> Class <Name_of_derivedClass>
5. Inherits Name_of_baseClass
6. ' Implementation of derived **class**
7. End Class

Let's create a program to understand the concept of Inheritance in VB.NET

Simple_Inherit.vb

1. Imports System
2. Module Simple_Inherit
3. Public Class Vehicle
4. Public Sub speed()
5. Console.WriteLine(" 4 Wheeler cars are more luxurious than 2 Wheeler")
6. End Sub
- 7.
8. Public Overridable Sub perform()
9. Console.WriteLine(" Both Vehicles are good, but in a narrow lane, two-wheeler are more comfortable than 4-Wheeler")
10. End Sub
11. End Class
- 12.
13. Class Bike
14. Inherits Vehicle
15. Public Overrides Sub perform()
16. Console.WriteLine(" Two-wheeler are lesser weight as compared to 4 wheeler")

```
17. End Sub
18. End Class
19. Class Car
20. Inherits Vehicle
21. Public Overrides Sub perform()
22. Console.WriteLine(" It is 4 wheelar car")
23. End Sub
24. End Class
25.
26. Sub Main()
27. Dim vehicle As Vehicle = New Vehicle()
28. Dim bike As Bike = New Bike()
29. Dim car As Car = New Car()
30. vehicle = bike
31. vehicle.perform()
32. vehicle = car
33. vehicle.perform()
34. Console.WriteLine(" Press any ley to exit...")
35. Console.ReadKey()
36. End Sub
37. End Module
```

Output:

```
C:\Users\AMIT YADAV\source\repos\MYConsoleApp1\bin\Debug\MYConsoleApp1.exe
Two wheeler are lesser weight as compared to 4 wheeler
It is 4 wheeler car
Press any ley to exit...
```

Let's create a program of inheritance using the MyBase in VB.NET.

Inherit_class.vb

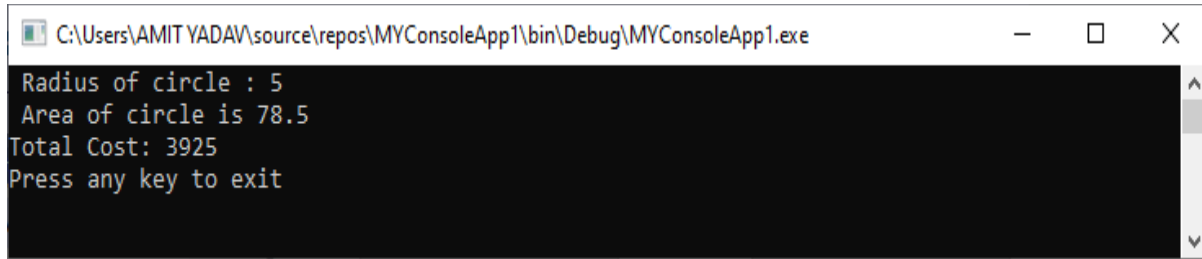
```
1. Module Inherit_class
2. Class Circle 'base class
3. Protected radius As Integer
4. Public Const PI As Double = 3.14
```



```
5.      Public Sub New(ByVal r As Integer)
6.          radius = r
7.      End Sub
8.      Public Function GetAreaCircle() As Double
9.          Return (PI * radius * radius)
10.     End Function
11.     Public Overridable Sub Show()
12.         Console.WriteLine(" Radius of circle : {0}", radius)
13.         Console.WriteLine(" Area of circle is {0}", GetAreaCircle())
14.     End Sub
15. End Class

16.
17. 'Derived Class
18. Class MyDeriveClass : Inherits Circle
19.     Private dimen As Double
20.     Public Sub New(ByVal r As Integer)
21.         MyBase.New(r)
22.     End Sub
23.     Public Function Getdimen() As Double
24.         Dim dimen As Double
25.         dimen = GetArea() * 50
26.         Return dimen
27.     End Function
28.     Public Overrides Sub Show()
29.         MyBase.Show()
30.         Console.WriteLine("Total Cost: {0}", Getdimen())
31.     End Sub
32. End Class

33.
34. Class Circle_Class
35.     Shared Sub Main()
36.         Dim c As MyDeriveClass = New MyDeriveClass(5)
37.         c.Show()
38.         Console.WriteLine("Press any key to exit")
39.         Console.ReadKey()
40.     End Sub
41. End Class
42. End Module
```

Multi-Level Inheritance

VB.NET only supports single inheritance that means a class can be inherited from the base class. However, VB.NET uses hierarchical inheritance to extend one class to another class, which is called **Multi-Level Inheritance**. For example, Class **C** extends Class **B**, and Class **B** extends Class **A**, Class **C** will inherit the member of both class **B** and Class **A**. The process of extending one class to another is known as multilevel inheritance.

1. Public Class A
2. ' implementation of code
3. End Class
- 4.
5. Public Class B
6. Inherits A
7. ' Implementation of Code
8. End Class
- 9.
10. Public Class C
11. Inherits A
12. ' Implementation of code
13. End Class

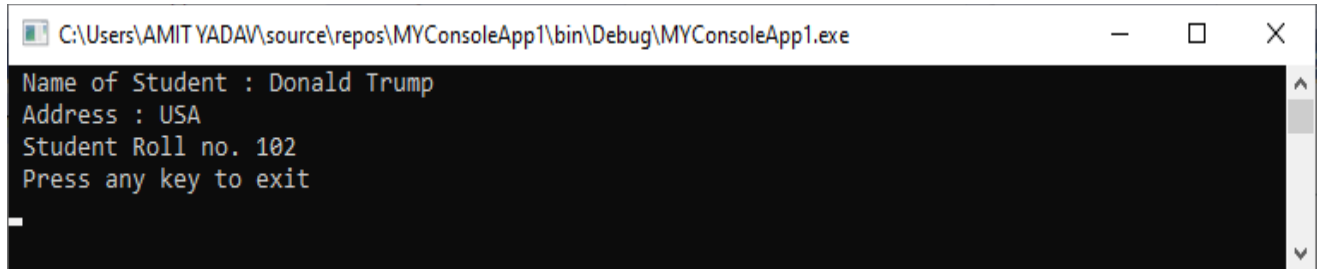
Let's create a program to understand the concept of Multi-Level Inheritance in VB.NET

Multi_inherit1.vb

1. Module Multi_inherit1
2. Public Class A
3. Public SName As String
4. Public Sub Display()



```
5.      Console.WriteLine(" Name of Student : {0}", SName)
6.      End Sub
7.      End Class
8.      Public Class B
9.          Inherits A
10.         Public place As String
11.         Public Sub GetPlace()
12.             Console.WriteLine(" Address : {0}", place)
13.         End Sub
14.     End Class
15.
16.     Public Class C
17.         Inherits B
18.         Public rollno As Integer
19.         Public Sub GetRollno()
20.             Console.WriteLine(" Student Roll no. {0}", rollno)
21.         End Sub
22.     End Class
23.
24.     Class Profile
25.         Public Sub Main(ByVal args As String())
26.             Dim c As C = New C()
27.             c.SName = "Donald Trump"
28.             c.place = "USA"
29.             c.rollno = 102
30.             c.Display()
31.             c.GetPlace()
32.             c.GetRollno()
33.             Console.WriteLine(" Press any key to exit")
34.             Console.ReadKey()
35.         End Sub
36.     End Class
37. End Module
```



```
C:\Users\AMIT YADAV\source\repos\MYConsoleApp1\bin\Debug\MYConsoleApp1.exe
Name of Student : Donald Trump
Address : USA
Student Roll no. 102
Press any key to exit
```

Interface

In VB.NET, the interface is similar to a class for inheriting all properties, methods, and events that a class or structure can implement. Using the interface in VB.NET, we can use multiple inheritances of classes. It uses the **Implements** keyword to implement the interfaces, and the **Interface** keyword is used to create the interface. All interfaces in VB.NET starts with **i**.

The Syntax for implementing an interface in a class.

1. Class MyClass
2. Inherits IClass
3. Private Sub FetchDetails() Implements IClass.FetchDetails
4. ' Method Implementation
5. End Sub
6. End Class

In the above snippets, we inherited an interface (**IClass**) in the Class **MyClass** that implemented to the interface method defined in a class.

Let's create and implement an instance using a class in VB.NET.

Get_Interface.vb

1. Module Get_Interface
2. Interface IStudent
3. Sub Details(ByVal y As String)
4. End Interface
5. Class Student



```
6. Implements IStudent
7. Public Sub Details(ByVal a As String) Implements IStudent.Details
8.     ' Throw New NotImplementedException()
9.     Console.WriteLine(" Name of Student: {0}", a)
10. End Sub
11. End Class
12.
13. Class Student1
14.     Implements IStudent
15.     Public Sub Details(ByVal a As String) Implements IStudent.Details
16.         'Throw New NotImplementedException()
17.         Console.WriteLine(" Course: {0}", a)
18.     End Sub
19. End Class
20. Sub Main(ByVal args As String())
21.     Dim stud As IStudent = New Student()
22.     stud.Details(" James Watt")
23.     Dim stud1 As IStudent = New Student1()
24.     stud1.Details("Computer Science")
25.     Console.WriteLine(" Press any key to exit...")
26.     Console.ReadKey()
27. End Sub
28. End Module
```

Output:

The screenshot shows a Windows console window titled "C:\Users\AMIT YADAV\source\repos\MYConsoleApp1\bin\Debug\MYConsoleApp1.exe". The output text is as follows:

```
Name of Student: James Watt
Course: Computer Science
Press any key to exit...
```



VB.NET Multithreading

What is VB.NET Thread?

When two or more processes execute simultaneously in a program, the process is known as multithreading. And the execution of each process is known as the **thread**. A single thread is used to execute a single logic or task in an application. By default, each application has one or more threads to execute each process, and that thread is known as the **main thread**.

To create and access a new thread in the Thread **class**, we need to import the **System.Threading** namespace. When the execution of a program begins in VB.NET, the Main thread is automatically called to handle the program logic. And if we create another thread to execute the process in Thread class, the new thread will become the **child** thread for the **main** thread.

Create a new Thread

In **VB.NET**, we can create a thread by extending the Thread class and pass the ThreadStart delegate as an argument to the Thread constructor. A **ThreadStart()** is a method executed by the new thread. We need to call the **Start()** method to start the execution of the new thread because it is initially in the **unstart** state. And the PrintInfo parameter contains **an executable statement** **that** is executed when creating a new thread.

1. ' Create a **new** thread
2. Dim th As Thread = New Thread(New ThreadStart(PrintInfo))
3. ' Start the execution of newly thread
4. th.Start()

Let's write a program to create and access the thread in Thread class.

create_Thread.vb

1. Imports System.Threading 'Imports the System.Threading namespace.
2. Module create_Thread
3. Sub Main(ByVal args As String())
4. ' create a **new** thread
5. Dim th As Thread = New Thread(New ThreadStart(AddressOf PrintInfo))
6. ' start the newly created thread
7. th.Start()



```
8. Console.WriteLine(" It is a Main Thread")
9. End Sub
10. Private Sub PrintInfo()
11. For j As Integer = 1 To 5
12. Console.WriteLine(" value of j is {0}", j)
13. Next j
```

Method	Description
Abort()	As the name defines, it is used to terminate the execution of a thread.
AllocateDataSlot()	It is used to create a slot for unnamed data on all threads.
AllocateNamedDatsSlot()	It is used to create a slot for defined data on all threads.
Equals()	It is used to check whether the current and defined thread object are equal.
Interrupt()	It is used to interrupt a thread from the Wait, sleep, and join thread state.
Join()	It is a synchronization method that stops the calling thread until the execution thread completes.
Resume()	As the name suggests, a Resume() method is used to resume a thread that has been suspended.
Sleep()	It is used to suspend the currently executing thread for a specified time.
Start()	It is used to start the execution of thread or change the state of an

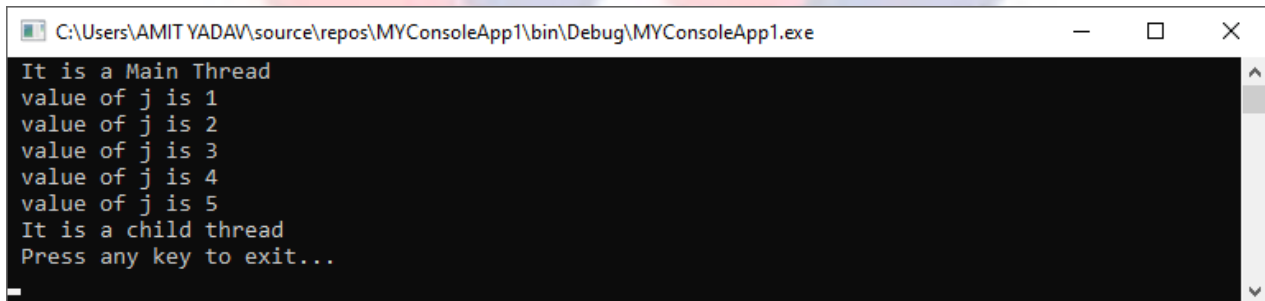
	ongoing instance.
Suspend()	It is used to stop the currently executing thread.

```

14. Console.WriteLine(" It is a child thread")
15. Console.WriteLine(" Press any key to exit...")
16. Console.ReadKey()
17. End Sub
18. End Module

```

Output:



```

C:\Users\AMIT YADAV\source\repos\MYConsoleApp1\bin\Debug\MYConsoleApp1.exe
It is a Main Thread
value of j is 1
value of j is 2
value of j is 3
value of j is 4
value of j is 5
It is a child thread
Press any key to exit...

```

In the above program, the main and child threads begin their execution simultaneously. The execution of the main thread is stopped after completing its function, but the child thread will continue to execute until its task is finished.

VB.NET Thread Methods

The following are the most commonly used methods of Thread class.

VB.NET Thread Life Cycle

In VB.NET Multithreading, each thread has a life cycle that starts when a new object is created using the Thread Class. Once the task defined by the thread class is complete, the life cycle of a thread will get the end.

There are some states of thread cycle in VB.NET programming.

State	Description
-------	-------------

Unstarted	When we create a new thread, it is initially in an unstarted state.
Runnable	When we call a Start() method to prepare a thread for running, the runnable situation occurs.
Running	A Running state represents that the current thread is running.
Not Runnable	It indicates that the thread is not in a runnable state, which means that the thread in sleep() or wait() or suspend() or is blocked by the I/O operation.
Dead	If the thread is in a dead state, either the thread has been completed its work or aborted.

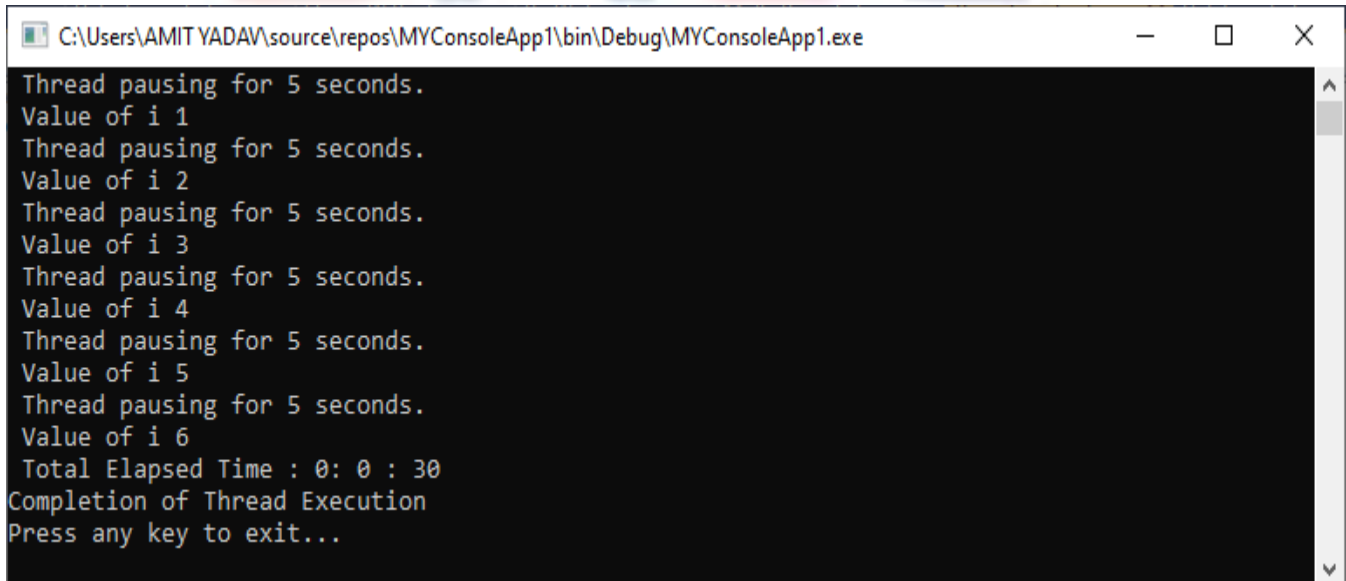
Let's create a program to manage a thread by using various methods of Thread Class.

Thread_cycle.vb

```
1. Imports System.Threading
2. Module Thread_cycle
3.     Sub Main(ByVal args As String())
4.         Dim s As Stopwatch = New Stopwatch()
5.         s.Start()
6.         Dim t As Thread = New Thread(New ThreadStart(AddressOf PrintInfo))
7.         t.Start()
8.         ' Halt another thread execution until the thread execution completed
9.         t.Join()
10.        s.[Stop]()
11.        Dim t1 As TimeSpan = s.Elapsed
12.        Dim ft As String = String.Format("{0}: {1} : {2}", t1.Hours, t1.Minutes, t1.Seconds)
13.        Console.WriteLine(" Total Elapsed Time : {0}", ft)
14.        Console.WriteLine("Completion of Thread Execution ")
15.        Console.WriteLine("Press any key to exit...")
16.        Console.ReadKey()
17.    End Sub
18.    Private Sub PrintInfo()
```

```
19. For j As Integer = 1 To 6
20.     Console.WriteLine(" Halt Thread for {0} Second", 5)
21.     ' It pause thread for 5 Seconds
22.     Thread.Sleep(5000)
23.     Console.WriteLine(" Value of i {0}", j)
24. Next
25. End Sub
26. End Module
```

Output:



In the above example, we used a different method of the **Thread** class such as the **Start()** method to start execution of the thread, the **Join()** method is used to stop the execution of the thread until the execution of the thread was completed. The **Sleep()** method is used to pause the execution of threads for 5 seconds.

Multithreading

When two or more processes are executed in a program to simultaneously perform multiple tasks, the process is known as **multithreading**.

When we execute an application, the Main thread will automatically be called to execute the programming logic synchronously, which means it executes one process after another. In this



way, the second process has to wait until the first process is completed, and it takes time. To overcome that situation, VB.NET introduces a new concept Multithreading to execute multiple tasks at the same time by creating multiple threads in a program.

Let's write a program of multiple threads to execute the multiple tasks at the same time in the VB.NET application.

Multi_thread.vb

```
1. Imports System.Threading
2. Module Multi_thread
3.     Sub Main(ByVal arg As String())
4.         Dim th As Thread = New Thread(New ThreadStart(AddressOf PrintInfo))
5.         Dim th2 As Thread = New Thread(New ThreadStart(AddressOf PrintInfo2))
6.         th.Start()
7.         th2.Start()
8.         Console.ReadKey()
9.     End Sub
10.    Private Sub PrintInfo()
11.        For j As Integer = 1 To 5
12.            Console.WriteLine(" value of j is {0}", j)
13.            Thread.Sleep(1000)
14.        Next
15.        Console.WriteLine(" Completion of First Thread")
16.    End Sub
17.    Private Sub PrintInfo2()
18.        For k As Integer = 1 To 5
19.            Console.WriteLine(" value of k is {0}", k)
20.        Next
21.        Console.WriteLine(" Completion of First Thread")
22.    End Sub
23. End Module
```

Output:



Constructor

- Constructor is a special kind of sub procedure
- A constructor has the following properties
- It always has the name 'New'
- Being a sub procedure it does not return any value
- The job of constructor is to create the object specified by a class and put it into valid state.
- It is automatically called when a new instance or object of a class is created, hence called a constructor.
- The constructor contains initialization code for each object like assigning default values to fields.
- Let us create a console application having a class "Student" with constructor

```
Imports System
Class student
Dim name as string
Public sub New()
Name= "unknown"
```

```
Console.WriteLine("Constructor called.....")
```

```
End sub
```

```
Public sub display()
```

```
Console.WriteLine("The name of student is" & name)
```

```
End sub
```

```
End class
```

VB.NET Exception Handling

What is an Exception?

An exception is an unwanted error that occurs during the execution of a program and can be a system exception or application exception. Exceptions are nothing but some abnormal and typically an event or condition that arises during the execution, which may interrupt the normal flow of the program.

An exception can occur due to different reasons, including the following:

A user has entered incorrect data or performs a division operator, such as an attempt to divide by zero.

A connection has been lost in the middle of communication, or system memory has run out.

Exception Handling

When an error occurred during the execution of a program, the exception provides a way to transfer control from one part of the program to another using exception handling to handle the error. VB.NET exception has four built-in keywords such as Try, Catch, Finally, and Throw to handle and move controls from one part of the program to another.

Keyword	Description
Try	A try block is used to monitor a particular exception that may throw an exception within the application. And to handle these exceptions, it always follows one or more catch blocks.
Catch	It is a block of code that catches an exception with an exception handler at the place in a program where the problem arises.
Finally	It is used to execute a set of statements in a program, whether an exception has occurred.
Throw	As the name suggests, a throw handler is used to throw an exception after the occurrence of a problem.



Exception Classes in VB.NET

In VB.net, there are various types of exceptions represented by classes. And these exception classes originate from their parent's class 'System.Exception'.

The following are the two exception classes used primarily in VB.NET.

1. **System.SystemException**
2. **System.ApplicationException**

System.SystemException: It is a base class that includes all predefined exception classes, and some system-generated exception classes that have been generated during a run time such as **DivideByZeroException**, **IndexOutOfRangeException**, **StackOverflowExpression**, and so on.

System.ApplicationException: It is an exception class that throws an exception defined within the application by the programmer or developer. Furthermore, we can say that it is a user-defined exception that inherits from **System.ApplicationException** class.

Syntax of exception handler block

1. Try
2. ' code or statement to be executed
3. [Exit Try block]
4. ' **catch** statement followed by Try block
5. Catch [Exception name] As [Exception Type]
6. [Catch1 Statements] Catch [Exception name] As [Exception Type]
7. [Exit Try]
8. [Finally
9. [Finally Statements]]
10. End Try

In the above syntax, the Try/Catch block is always surrounded by a code that can throw an exception. And the code is known as a protected code. Furthermore, we can also use multiple catch statements to catch various types of exceptions in a program, as shown in the syntax.

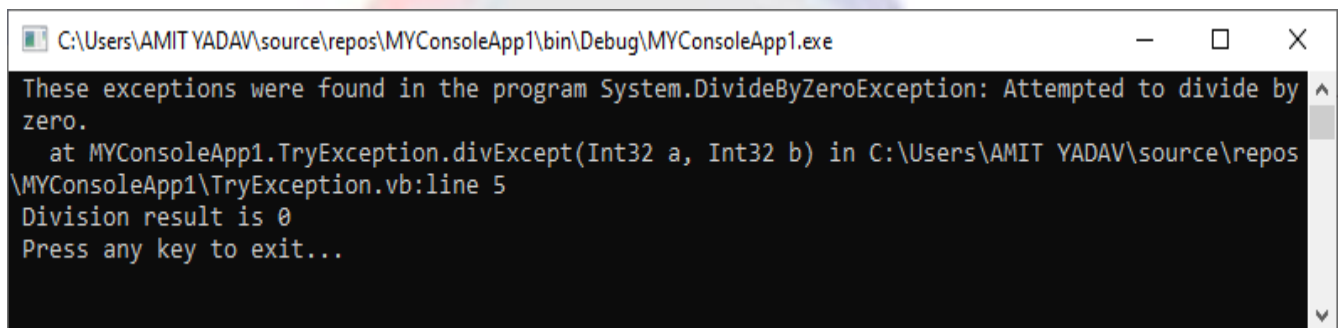
Example to Exception Handle

Let's create a program to handle an exception using the Try, Catch, and Finally keywords for Dividing a number by zero in VB.NET programming.

TryException.vb

```
1. Module TryException
2.     Sub divExcept(ByVal a As Integer, ByVal b As Integer)
3.         Dim res As Integer
4.         Try
5.             res = a \ b
6.             ' Catch block followed by Try block
7.             Catch ex As DivideByZeroException
8.                 Console.WriteLine(" These exceptions were found in the program {0}", ex)
9.                 ' Finally block will be executed whether there is an exception or not.
10.            Finally
11.                Console.WriteLine(" Division result is {0}", res)
12.            End Try
13.        End Sub
14.    Sub Main()
15.        divExcept(5, 0) ' pass the parameters value
16.        Console.WriteLine(" Press any key to exit...")
17.        Console.ReadKey()
18.    End Sub
19. End Module
```

Output:



Creating User-Defined Exceptions

It allows us to create custom exceptions derived from the **ApplicationException** class.

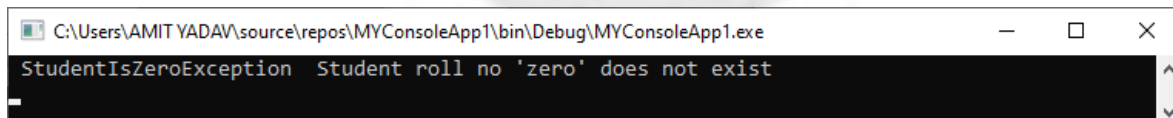
Let's create a program to understand the uses of User-Defined Exceptions in VB.NET Exception Handling.



User_Exception.vb

```
1. Module User_Exception
2.     Public Class StudentIsZeroException : Inherits Exception
3.
4.         Public Sub New(ByVal stdetails As String)
5.             MyBase.New(stdetails)
6.         End Sub
7.     End Class
8.     Public Class StudentManagement
9.         Dim stud As Integer = 0
10.        Sub ShowDetail()
11.            If (stud = 0) Then
12.                Throw (New StudentIsZeroException(" Student roll no 'zero' does not exist"))
13.            Else
14.                Console.WriteLine(" Student is {0}", stud)
15.            End If
16.        End Sub
17.    End Class
18.    Sub Main()
19.        Dim stdmg As StudentManagement = New StudentManagement()
20.        Try
21.            stdmg.ShowDetail()
22.        Catch ex As StudentIsZeroException
23.            Console.WriteLine(" StudentIsZeroException {0}", ex.message)
24.        End Try
25.        Console.ReadKey()
26.    End Sub
27. End Module
```

Output:

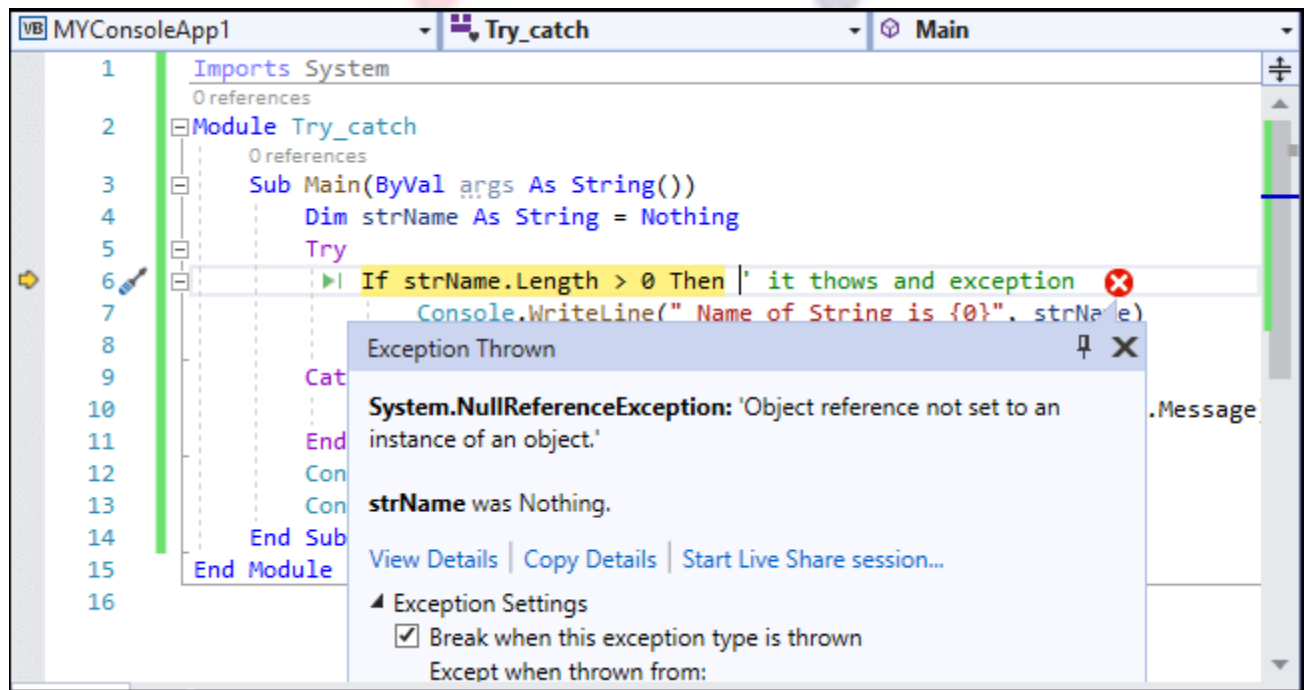


Using Try-Catch Statement

Lets' create a program using the Try-Catch statement in VB.NET to handle the exceptions.

Try_catch.vb

```
1. Imports System
2. Module Try_catch
3.     Sub Main(ByVal args As String())
4.         Dim strName As String = Nothing
5.         Try
6.             If strName.Length > 0 Then ' it thows and exception
7.                 Console.WriteLine(" Name of String is {0}", strName)
8.             End If
9.         Catch ex As Exception ' it catches an exception
10.            Console.WriteLine(" Catch exception in a proram {0}", ex.Message)
11.        End Try
12.        Console.WriteLine(" Press any key to exit...")
13.        Console.ReadKey()
14.    End Sub
15. End Module
```

Output:**Throwing Objects**

In VB.NET exception handling, we can throw an object exception directly or indirectly derived from the **System.Exception** class. To throw an object using the throw statement in a catch block, such as:

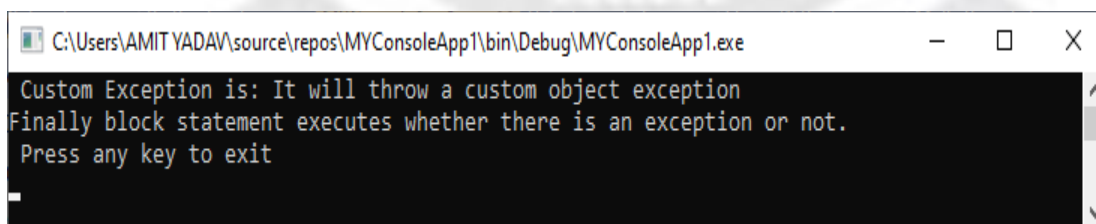
1. Throw [expression]

Let's create a program to throw an object in VB.NET exception.

throwexcept.vb

1. Imports System
2. Module throwexcept
3. Sub Main()
4. Try
5. Throw New ApplicationException("It will throw a custom object exception")
6. Catch ex As Exception
7. Console.WriteLine(" Custom Exception is: {0}", ex.Message)
8. Finally
9. Console.WriteLine("Finally block statement executes whether there is an exception or not.")
10. End Try
11. Console.WriteLine(" Press any key to exit")
12. Console.ReadKey()
13. End Sub
14. End Module

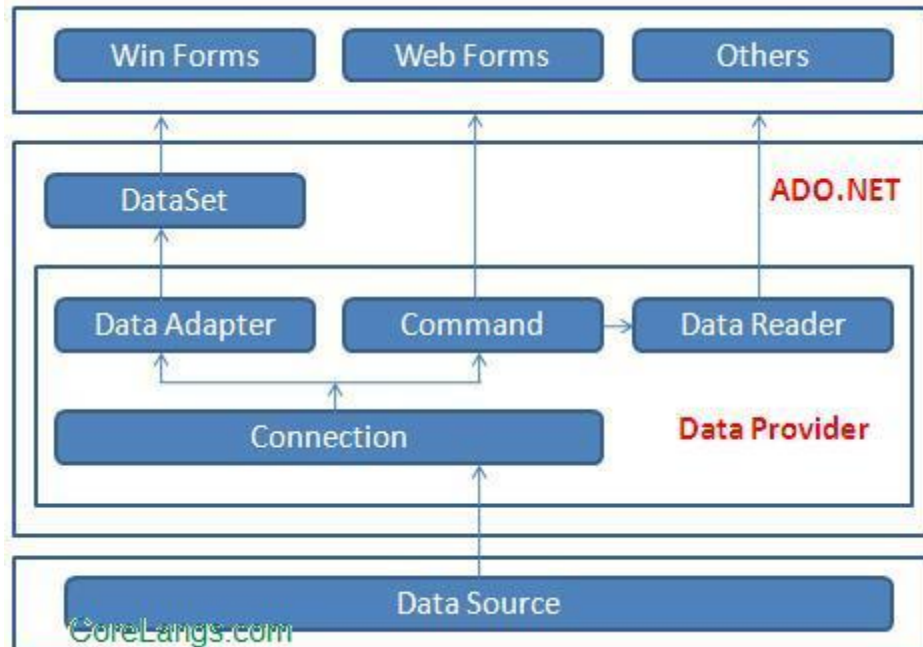
Output:



```
C:\Users\AMITYADAV\source\repos\MYConsoleApp1\bin\Debug\MYConsoleApp1.exe
Custom Exception is: It will throw a custom object exception
Finally block statement executes whether there is an exception or not.
Press any key to exit
```

Chapter 4. Architecture of ADO.NET

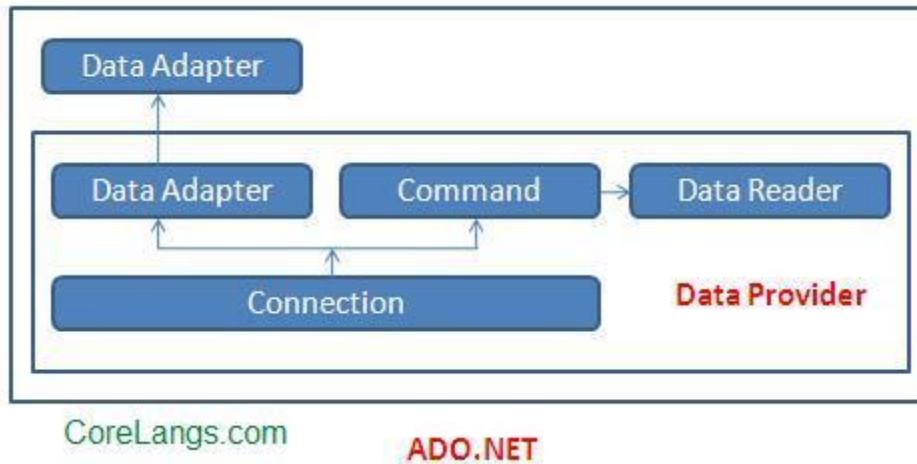
ADO.NET Architecture



ADO.NET

ADO.NET consist of a set of Objects that expose data access services to the .NET environment. It is a data access technology from Microsoft .Net Framework , which provides communication between relational and non relational systems through a common set of components .

System.Data namespace is the core of ADO.NET and it contains classes used by all data providers. ADO.NET is designed to be easy to use, and Visual Studio provides several wizards and other features that you can use to generate ADO.NET data access code.

Data Providers and DataSet

The two key components of ADO.NET are Data Providers and DataSet . The Data Provider classes are meant to work with different kinds of data sources. They are used to perform all data-management operations on specific databases. DataSet class provides mechanisms for managing data when it is disconnected from the data source.

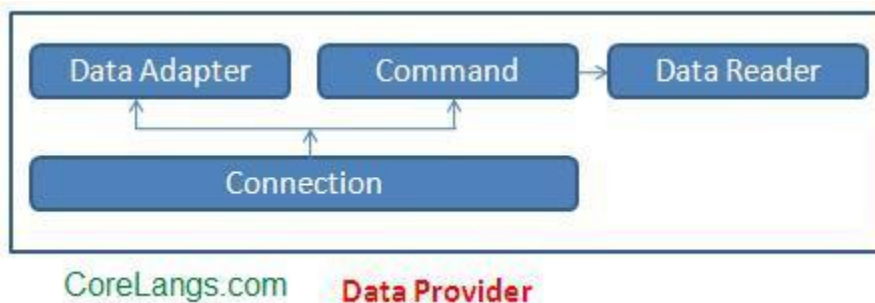
Data Providers

The .Net Framework includes mainly three Data Providers for ADO.NET. They are the Microsoft SQL Server Data Provider , OLEDB Data Provider and ODBC Data Provider . SQL Server uses the SqlConnection object , OLEDB uses the OleDbConnection Object and ODBC uses OdbcConnection Object respectively.

[ASP.NET SQL Server Connection](#)

[ASP.NET OLEDB Connection](#)

[ASP.NET ODBC Connection](#)





A data provider contains Connection, Command, DataAdapter, and DataReader objects. These four objects provides the functionality of Data Providers in the ADO.NET.

Connection

The Connection Object provides physical connection to the Data Source. Connection object needs the necessary information to recognize the data source and to log on to it properly, this information is provided through a connection string.

[ASP.NET Connection](#)

Command

The Command Object uses to perform SQL statement or stored procedure to be executed at the Data Source. The command object provides a number of Execute methods that can be used to perform the SQL queries in a variety of fashions.

[ASP.NET Command](#)

DataReader

The DataReader Object is a stream-based , forward-only, read-only retrieval of query results from the Data Source, which do not update the data. DataReader requires a live connection with the database and provides a very intelligent way of consuming all or part of the result set.

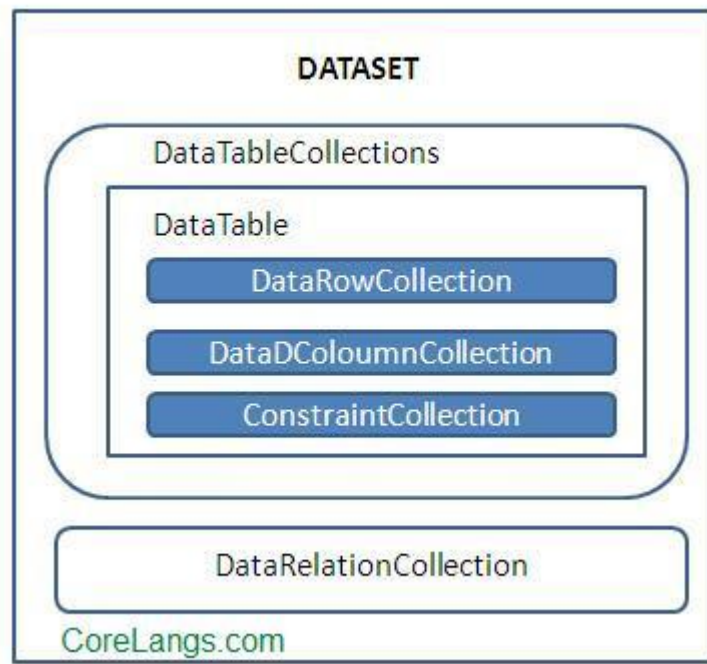
[ASP.NET DataReader](#)

DataAdapter

DataAdapter Object populate a Dataset Object with results from a Data Source . It is a special class whose purpose is to bridge the gap between the disconnected Dataset objects and the physical data source.

[ASP.NET DataAdapter](#)





DataSet

DataSet provides a disconnected representation of result sets from the Data Source, and it is completely independent from the Data Source. DataSet provides much greater flexibility when dealing with related Result Sets.

DataSet contains rows, columns, primary keys, constraints, and relations with other DataTable objects. It consists of a collection of DataTable objects that you can relate to each other with DataRelation objects. The DataAdapter Object provides a bridge between the DataSet and the Data Source.

ADO.NET Introduction

It is a module of .Net Framework which is used to establish connection between application and data sources. Data sources can be such as SQL Server and XML. ADO.NET consists of classes that can be used to connect, retrieve, insert and delete data.

All the ADO.NET classes are located into **System.Data.dll** and integrated with XML classes located into **System.Xml.dll**.

ADO.NET has two main components that are used for accessing and manipulating data are the .NET Framework data provider and the DataSet.

NET Framework Data Providers



These are the components that are designed for data manipulation and fast access to data. It provides various objects such as **Connection, Command, DataReader and DataAdapter** that are used to perform database operations. We will have a detailed discussion about **Data Providers** in new topic.

The DataSet

It is used to access data independently from any data resource. DataSet contains a collection of one or more DataTable objects of data. The following diagram shows the relationship between .NET Framework data provider and DataSet.

ADO.NET DataTable

DataTable represents relational data into tabular form. ADO.NET provides a DataTable class to create and use data table independently. It can also be used with DataSet also. Initially, when we create DataTable, it does not have table schema. We can create table schema by adding columns and constraints to the table. After defining table schema, we can add rows to the table.

We must include **System.Data** namespace before creating DataTable.

DataTable Class Signature

1. public class DataTable : System.ComponentModel.MarshalByValueComponent, System.ComponentModel.IListSource,
2. System.ComponentModel.ISupportInitializeNotification, System.Runtime.Serialization.IEnumerable, Serializable,
3. System.Xml.Serialization.IXmlSerializable

DataTable Properties

The following table contains the DataTable class properties.

DataTable Methods

The following table contains the DataTable class methods.

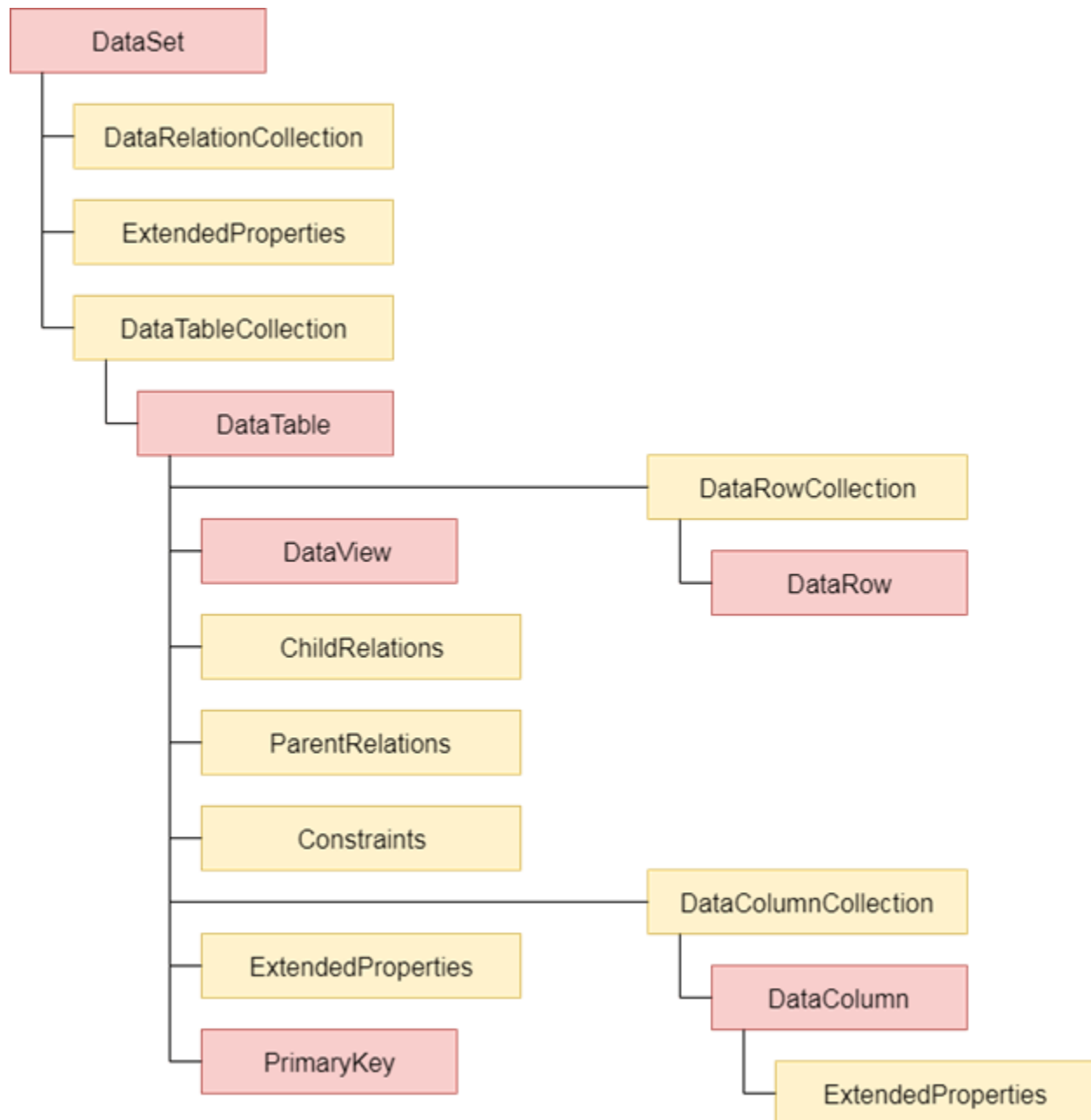


Fig: ADO.NET Architecture

Which one should we use **DataReader** or **DataSet**?

We should consider the following points to use **DataSet**.



- It caches data locally at our application, so we can manipulate it.
- It interacts with data dynamically such as binding to windows forms control.
- It allows performing processing on data without an open connection. It means it can work while connection is **disconnected**.

If we required some other functionality mentioned above, we can use **DataReader** to improve performance of our application.

DataReader does not perform in disconnected mode. It requires DataReader object to be **connected**.

.NET Framework Data Providers Objects

Following are the core object of Data Providers.

.NET Framework Data Provider for SQL Server

Data provider for SQL Server is a lightweight component. It provides better performance because it directly access SQL Server without any middle connectivity layer. In early versions, it interacts with ODBC layer before connecting to the SQL Server that created performance issues.

The .NET Framework Data Provider for SQL Server classes is located in the **System.Data.SqlClient** namespace. We can include this namespace in our C# application by using the following syntax.

1. using System.Data.SqlClient;

This namespace contains the following important classes.

Which .NET Framework Data Provider is better

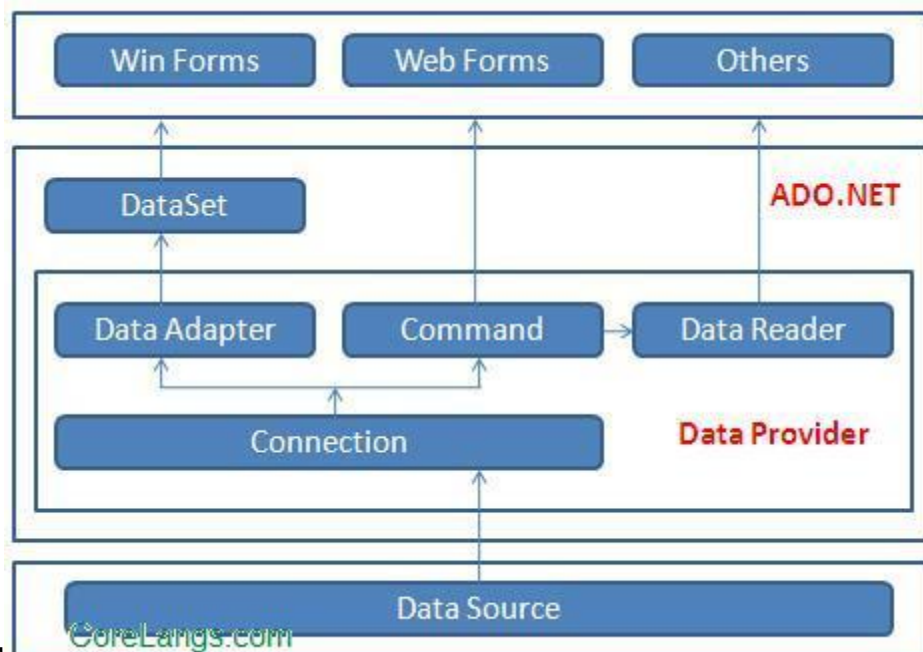
Selection of data provider is depends on the design and data source of our application. Choice of optimum .NET Framework data provider can improve the performance, capability and integrity of our application. The following table demonstrates advantages and disadvantages of data provider.

ADO.NET Tutorial provides basic and advanced concepts of ADO.NET. Our ADO.NET Tutorial is designed for beginners and professionals both.

ADO.NET is a module of .Net Framework which is used to establish connection between application and data sources. Data sources can be such as SQL Server and XML. ADO.NET consists of classes that can be used to connect, retrieve, insert and delete data.

Our ADO.NET Tutorial includes all topics of ADO.NET Tutorial such as ADO Net Tutorial with introduction, data providers, sql server connectivity, connection, command, datareader, dataset, data adapter, data tables, web form examples, mvc examples etc.

ADO.NET Architecture

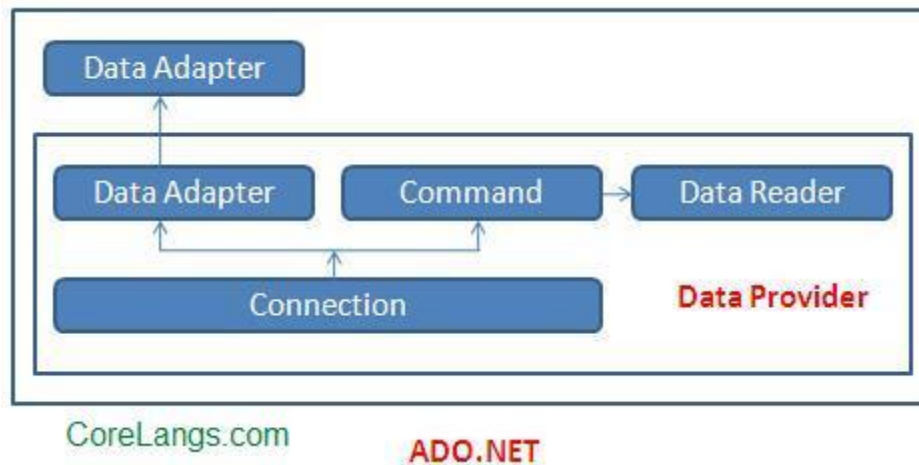


ADO.NET

ADO.NET consist of a set of Objects that expose data access services to the .NET environment. It is a data access technology from Microsoft .Net Framework , which provides communication between relational and non relational systems through a common set of components .

System.Data namespace is the core of ADO.NET and it contains classes used by all data providers. ADO.NET is designed to be easy to use, and Visual Studio provides several wizards and other features that you can use to generate ADO.NET data access code.

Data Providers and DataSet



The two key components of ADO.NET are Data Providers and DataSet . The Data Provider classes are meant to work with different kinds of data sources. They are used to perform all data-management operations on specific databases. DataSet class provides mechanisms for managing data when it is disconnected from the data source.

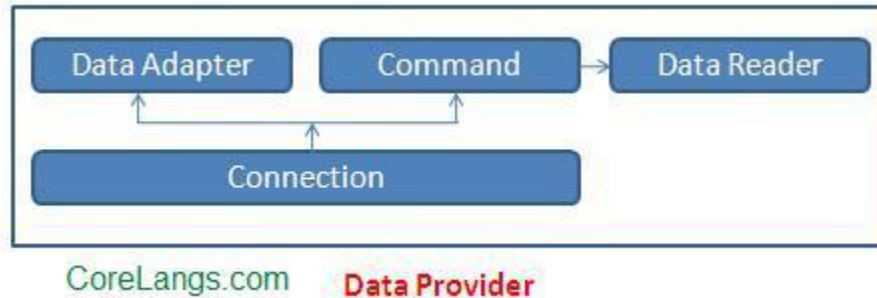
Data Providers

The .Net Framework includes mainly three Data Providers for ADO.NET. They are the Microsoft SQL Server Data Provider , OLEDB Data Provider and ODBC Data Provider . SQL Server uses the SqlConnection object , OLEDB uses the OleDbConnection Object and ODBC uses OdbcConnection Object respectively.

[ASP.NET SQL Server Connection](#)

[ASP.NET OLEDB Connection](#)

ASP.NET ODBC Connection



A data provider contains Connection, Command, DataAdapter, and DataReader objects. These four objects provide the functionality of Data Providers in the ADO.NET.

Connection

The Connection Object provides physical connection to the Data Source. Connection object needs the necessary information to recognize the data source and to log on to it properly, this information is provided through a connection string.

ASP.NET Connection

Command

The Command Object is used to perform SQL statement or stored procedure to be executed at the Data Source. The command object provides a number of Execute methods that can be used to perform the SQL queries in a variety of fashions.

ASP.NET Command

DataReader

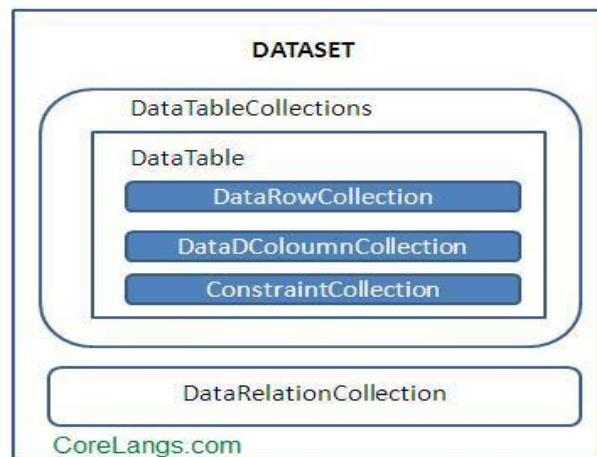
The DataReader Object is a stream-based , forward-only, read-only retrieval of query results from the Data Source, which do not update the data. DataReader requires a live connection with the database and provides a very intelligent way of consuming all or part of the result set.

ASP.NET DataReader

DataAdapter

DataAdapter Object populate a Dataset Object with results from a Data Source . It is a special class whose purpose is to bridge the gap between the disconnected Dataset objects and the physical data source.

ASP.NET DataAdapter



DataSet

DataSet provides a disconnected representation of result sets from the Data Source, and it is completely independent from the Data Source. DataSet provides much greater flexibility when dealing with related Result Sets.

DataSet contains rows, columns, primary keys, constraints, and relations with other DataTable objects. It consists of a collection of DataTable objects that you can relate to each other with DataRelation objects. The



DataAdapter Object provides a bridge between the DataSet and the Data Source.

Advantages of ADO.Net

There are some similarities and differences between ADO and ADO.NET, but the way they operate and their foundations are quite different.

ADO stands for ActiveX Data Objects and it relies on COM whereas ADO.NET relies on managed providers defined by the .NET Common Language Runtime. Both models require a connection to a data store to fetch data, and the code for getting a connection is similar for both.

A major difference in creating connections with ADO and ADO.NET is that ADO fits all connections to all types of data sources into a single Connection object. ADO.NET can have separate Objects that represent connections to different data sources.

In ADO.NET you can create multiple data provider namespaces to connect specifically with a particular data source, making access faster and more efficient and allowing each namespace to exploit the features of its targeted data provider.

SQL Server Connection

VB.Net

```
Dim connectionString As String
```



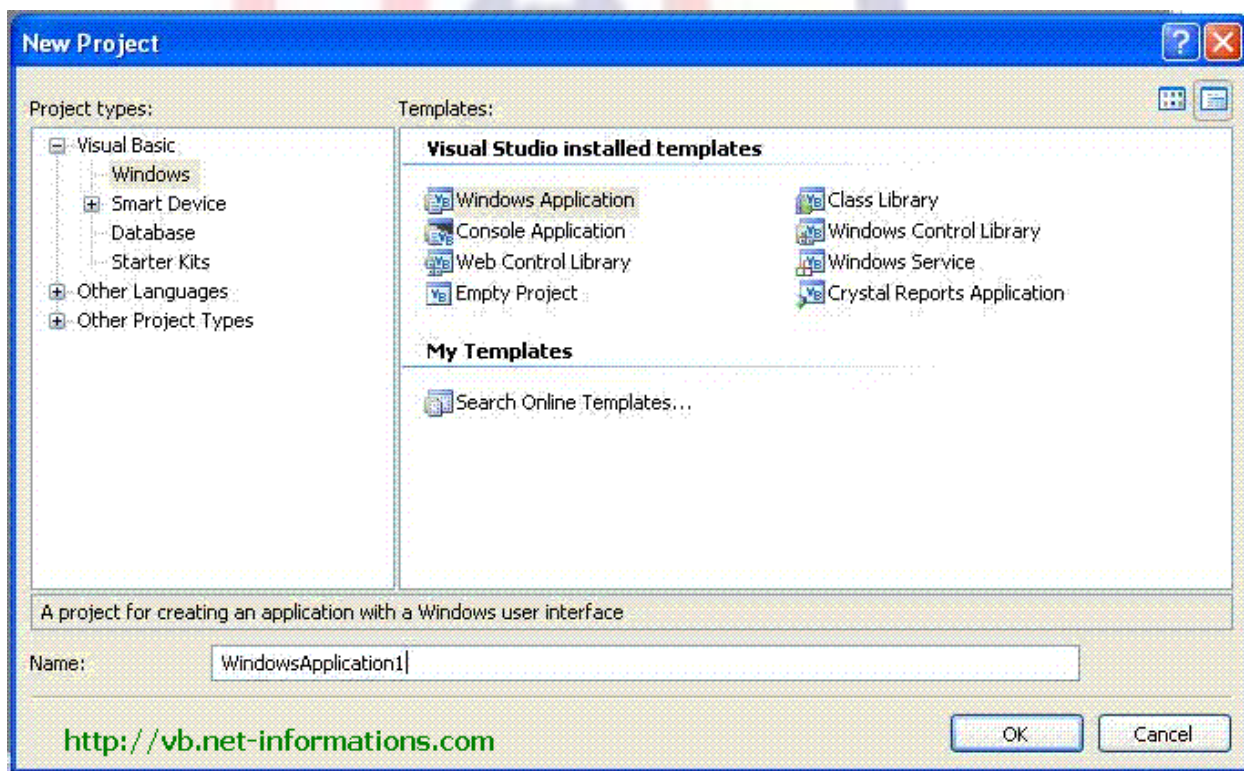
```
connectionString = ConfigurationManager.ConnectionStrings("SQLDbConnection").ToString
```



Unit 5. Crystal Report

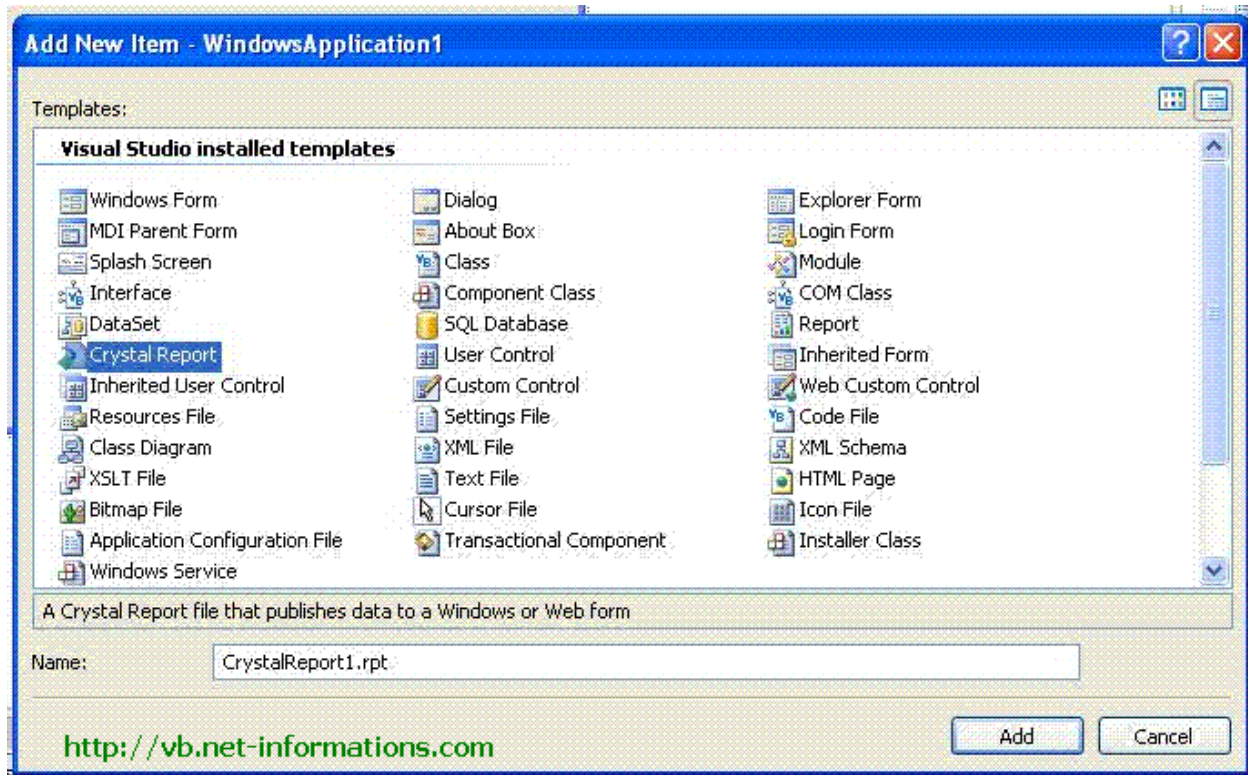
All Crystal Reports programming samples in this tutorials is based on the following database (crystaldb) . Please take a look at the database structure before you start this tutorial - Click here to see Database Structure .

Open Visual Studio .NET and select a new Visual Basic .NET Project.



Create a new Crystal Reports for Product table from the above database crystalDB. The Product Table has three fields (Product_id,Product_name,Product_price) and we are showing the whole table data in the Crystal Reports.

From main menu in Visual Studio select PROJECT-->Add New Item . Then Add New Item dialogue will appear and select Crystal Reports from the dialogue box.

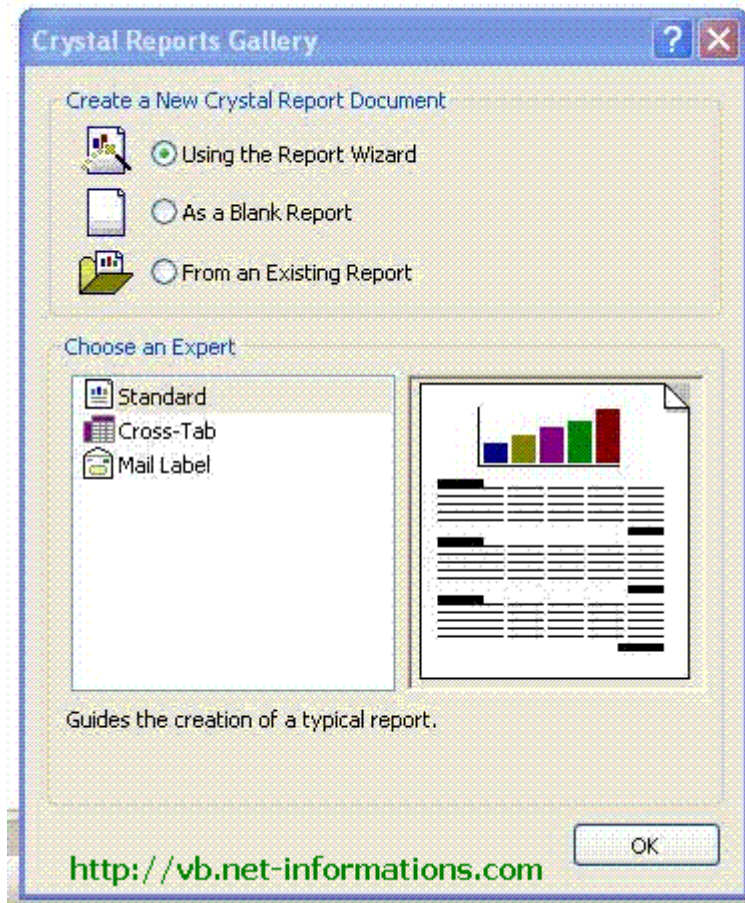


Select Report type from Crystal Reports gallery.

Accept the default settings and click OK.

Next step is to select the appropriate connection to your database. Here we are going to select OLEDB connection for SQL Server

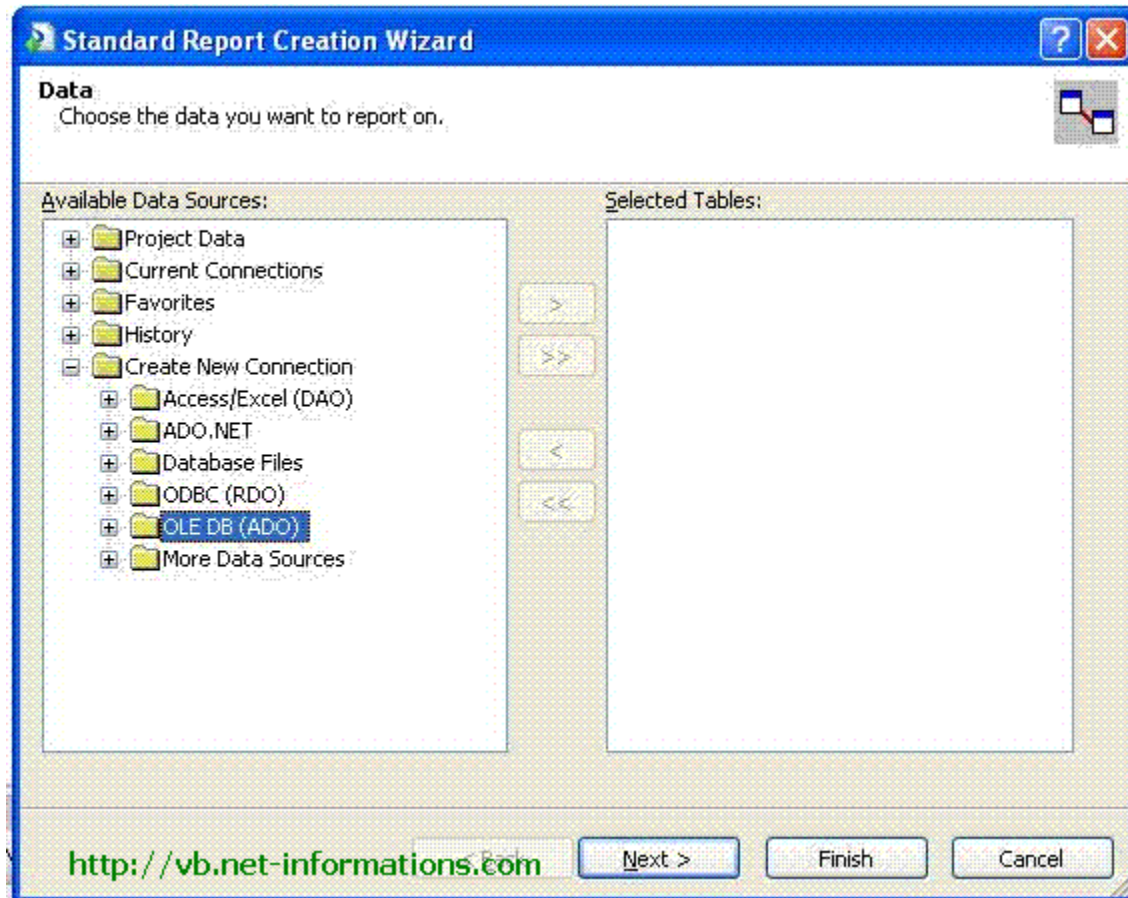
Select OLE DB (ADO) from Create New Connection.



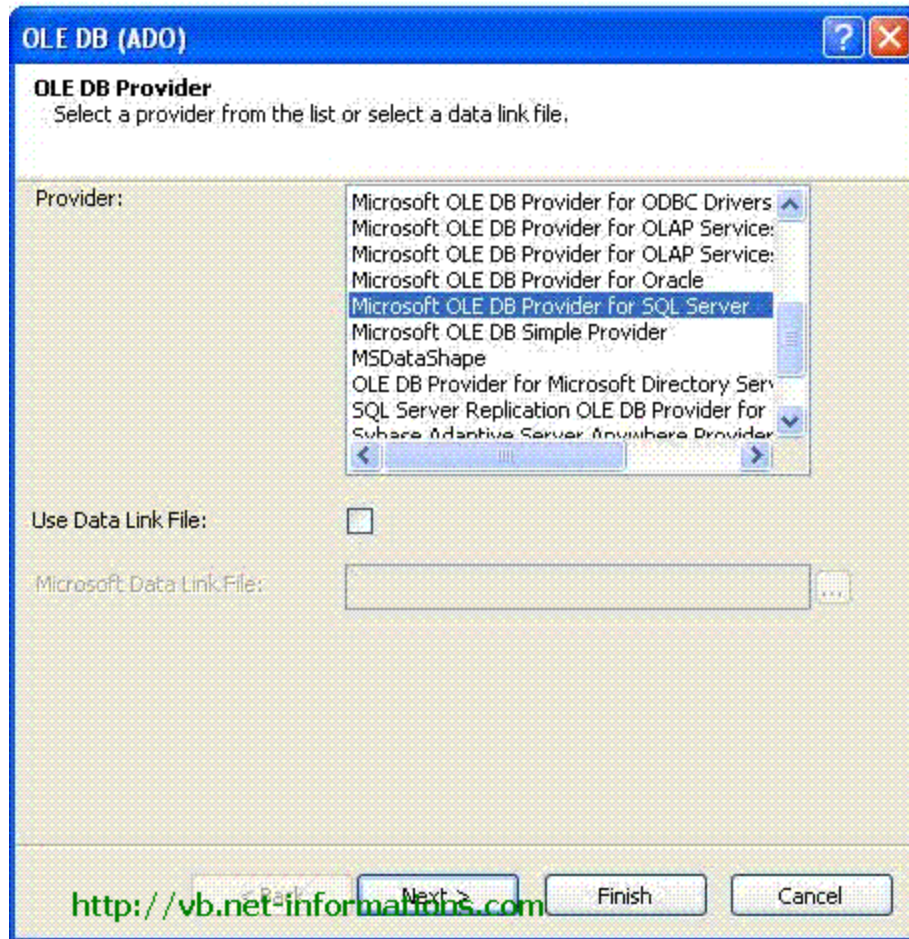
Accept the default settings and click OK.

Next step is to select the appropriate connection to your database. Here we are going to select **OLEDB connection for SQL Server**

Select OLE DB (ADO) from Create New Connection.



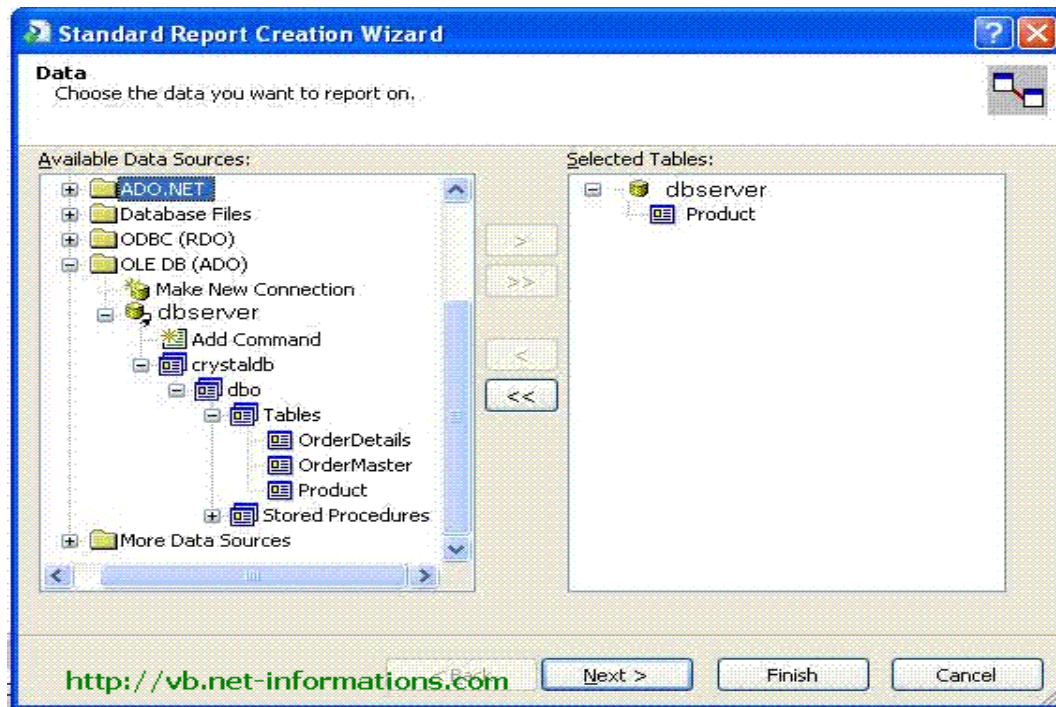
Select Microsoft OLE DB Provider for SQL Server .



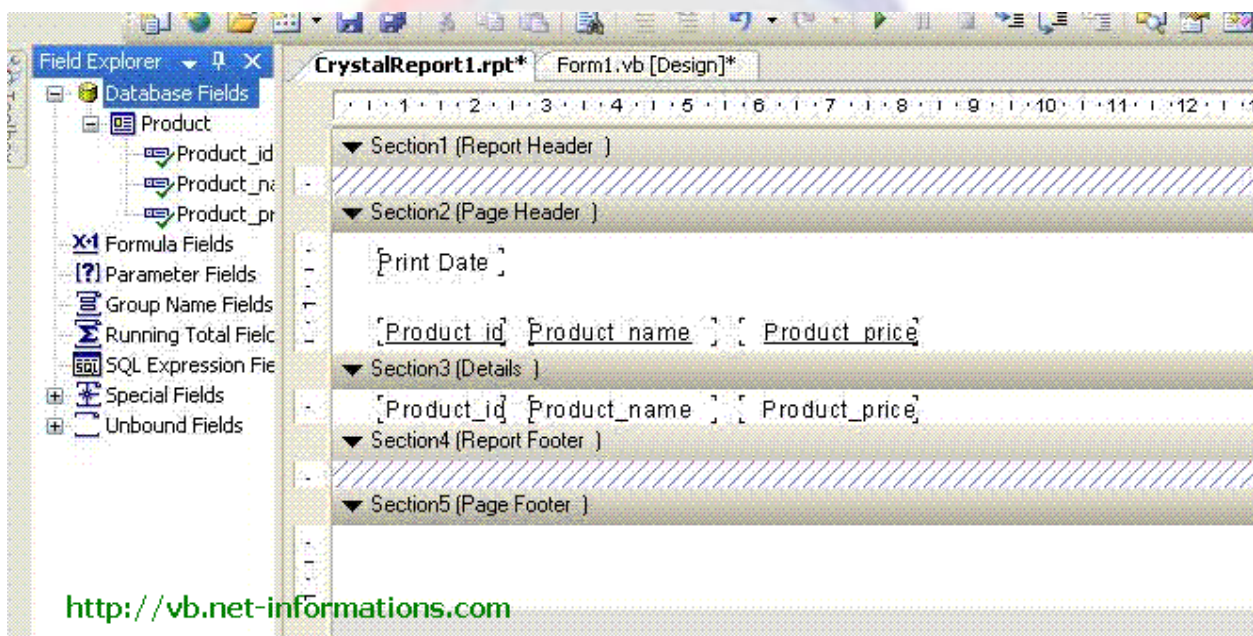
Next screen is the SQL Server authentication screen . Select your Sql Server name , enter userid , password and select your Database Name . Click next , Then the screen shows OLE DB Property values , leave it as it is , and click finish.

Then you will get your Server name under OLEDB Connection from there select database name (Crystalldb) and click the tables , then you can see all your tables from your database.

From the tables list select Product table to the right side list .

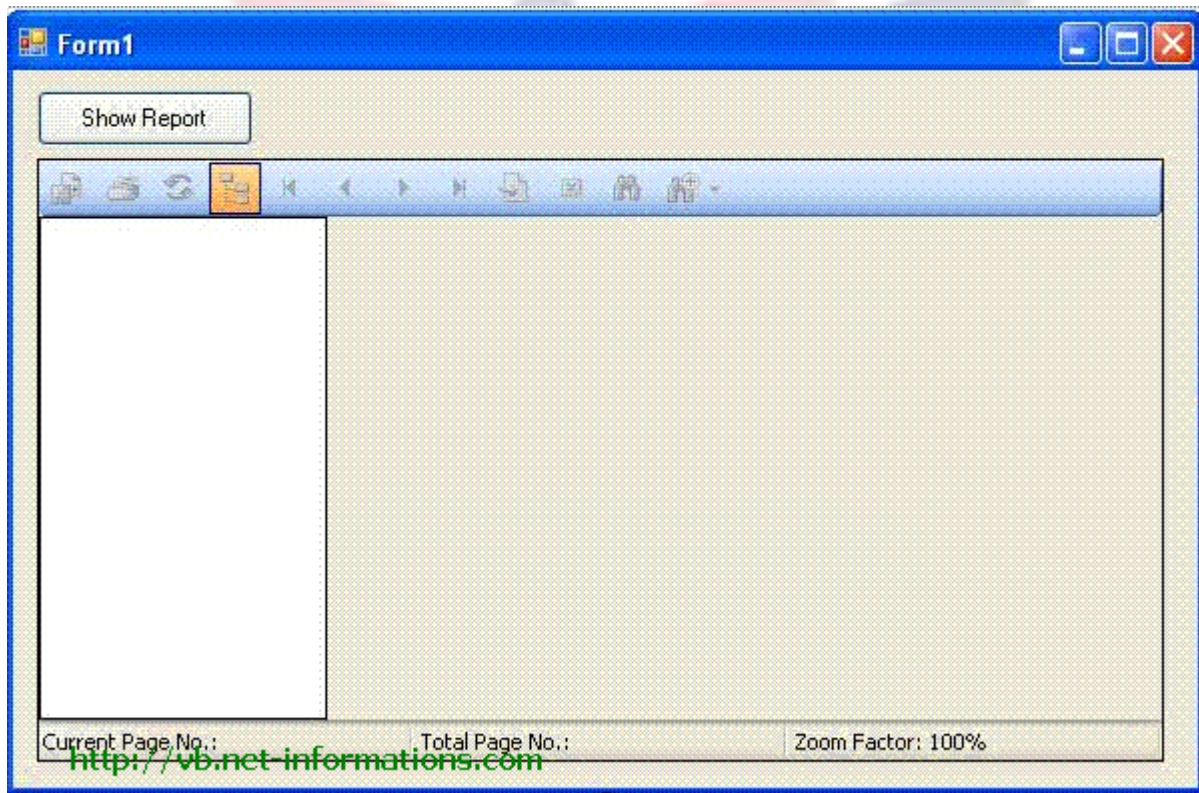


Click Finish Button. Then you can see the Crystal Reports designer window . You can arrange the design according your requirements. Your screen look like the following picture.



Now the designing part is over and the next step is to call the created Crystal Reports in VB.NET through Crystal Reports Viewer control .

Select the default form (Form1.vb) you created in VB.NET and drag a button and CrystalReportViewer control to your form.



VB.NET Crystal Reports from multiple tables

Imports CrystalDecisions.CrystalReports.Engine

Public Class Form1

Private Sub Button1_Click(ByVal sender As System.Object,

ByVal e As System.EventArgs) Handles Button1.Click

Dim cryRpt As New ReportDocument

```
cryRpt.Load("PUT CRYSTAL REPORT PATH HERE\CrystalReport1.rpt")
```

```
CrystalReportViewer1.ReportSource = cryRpt
```

```
CrystalReportViewer1.Refresh()
```

```
End Sub
```

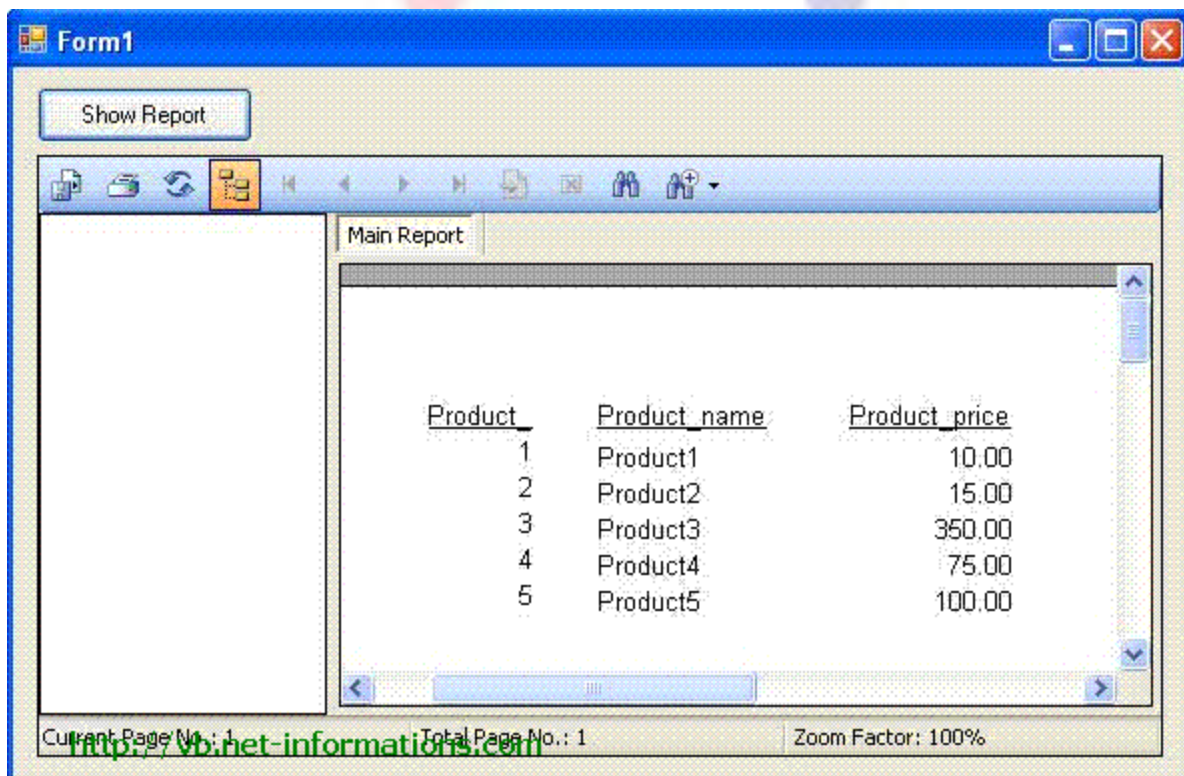
```
End Class
```

NOTES:

```
cryRpt.Load("PUT CRYSTAL REPORT PATH HERE\CrystalReport1.rpt")
```

The Crystal Reports is in your project location, there you can see CrystalReport1.rpt . So give the full path name of report here.

After you run the source code you will get the report like this.





Reference Books:

1. Programming Microsoft Visual Basic.NET – Francesco Balena
2. The Complete Reference -Visual Basic .NET – Jeffrey R. Shapiro
3. Murach's VB.NET database programming with ADO.NET -Anne Prince and Doug Lowe
4. The Visual Basic.NET COACH
5. Visual Basic .NET 2003 in 21 Days. – Steven Holzner, SAMS Publications.
6. Mastering Crystal Report - BPB Publication
7. Crystal Report – The Complete Reference :- Tata McGraw Hill

Link

- 1.http://vb.net-informations.com/crystal-report/vb.net_crystal_report_step_by_step.html
- 2.<http://vb.net-informations.com/>