Course Number	Course Name	L-T-P- Credits	Year of Introduction
503	C# Programming	3L-1T-0P-=4C	2018

Syllabus

UNIT	Contents
1	**The .net Framework:** Introduction, common language runtime, common type system, common language specification, the base class library, the .net class library, Intermediate language, Just in time compilation, garbage collection, assemblies, web services, COM, localization
2	**Introduction to C#:** Evaluation of C#, characteristics of C#, application of C#, difference between C++ and C#, difference between Java and C#. Introduction to C# environment: The .NET strategy, the origins of the .NET technology, the .NET framework, the common language runtime, framework base classes, user and programs interface, visual studio .NET, .NET languages, benefits of the .NET approach, C# and .NET. Data types, identifiers, variables, constants, C# statements, OOPs concept, array and strings, operators, control statements, type conversions, Mathematical functions.
3	**Classes and Objects:** Basic principles of OOP's, class, objects, constructors, static members, static constructors, private constructors, copy constructors, destructors, member initialization, this reference, nesting of classes, constant members, read only members, properties, indexers. Inheritance and polymorphism: overloading, inheritance, overriding, interfaces
4	**Visual studio IDE features, introduction to Window forms, components, control:** textbox, label, link label, status bar, checked list box, combo box, list box, list view, radio button, button, panel, group box, dialog box, menu control, properties, methods, events of controls.
5	**ADO.net:** the component model, creating database connection, database command, data repeater, connecting to data sources, choosing a .net data provider, manage a connection, building command objects, executing commands, building datasets and data tables, data adapter
6	**Managing Console, I/O operations:** Console class, console input, console output, formatted output, numeric formatting, standard numeric format, custom numeric format. Managing Errors and Exceptions: Types of errors, exceptions, syntax of exception handling code, multiple catch statement, the exception hierarchy, general catch handler, using final statement, nested try blocks, throwing our own exceptions, checked and unchecked operators, using exceptions for debugging.

Course Objectives

- Learn the fundamentals of C# programming in Visual Studio.
- To Use .Net Framework
- To Handle Exceptions in C#
- To implement Object oriented technology in C#
- To operate with Arrays
- To use Class Designer and Object Test Bench tools.

Expected Outcome:

This COURSE focuses on building applications with a graphical user interface (GUI) for the Microsoft Windows operating system although GUI interfaces on other operating systems, and on the Web, Topics include: event-driven programming, Win32 API, dialog boxes and standard GUI controls, dynamic link libraries, .NET Framework. The C# programming languages will be used to build applications.

Reference Books:

- The Complete Visual C# Programmer's Guide
- A Programmer's Introduction to C# 2.0, Third Edition
- 3. C# and the .NET Platform, Second Edition

Course Overview:

The demand for IT skills has never been greater as the world continues down the path of revolution. In fact, the US Bureau of Labor Statistics projects the **IT industry to grow 22% by 2029.** Driving this demand is the growing need for new applications and software across all kinds of industry.

While there is definite growth forecasted, the past year has not been without its challenges due to the **COVID-19 outbreak.** The pandemic has rocked the global economy to its core and has significantly impacted the IT skills workforce.

Despite all of this, the principle of providing valuable information to students. This purpose is what motivated us.

C# is a modern, general-purpose, object-oriented programming language developed by Microsoft and approved by European Computer Manufacturers Association (ECMA) and International Standards Organization (ISO). C# was developed by Anders Hejlsberg and his team during the development of .Net Framework.

C# is designed for Common Language Infrastructure (CLI), which consists of the executable code and runtime environment that allows use of various high-level languages on different computer platforms and architectures. The Integrated Development Environment (IDE) available for C# provides immediate error correction, strong typing, refactoring and reformatting. Students shall be exposed to such tools for the development of code.

The course will address the basics of the programming language, the need for its study, the features of the language and subsequently developing program code. The following questions

Will be answered during the course:

a) Why do we need to study C#?
b) How C# is different from other programming languages?
c) How it implements object-oriented concepts of programming?
d) What are the basic building blocks of C#?
e) How to develop and execute programs in C# using control statements and loops?
f) How namespaces are developed and used in C#?
g) How to handle Errors and Exceptions?
h) What are console I/O operations in c#?
i) What is ADO.NET?
j) How to connect with Different database?

This course will equip the students with an understanding of the fundamental principles of the language. **The students are expected to review the course readings and the indicated portion of the prescribed text for class discussions prior to attending each session.**

2. Evaluation Criteria:

Component	Description	Weight age
First Internal Examination	First internal question paper will be based on first 3 unit of syllabus.	10 marks
Second Internal Examination	Second internal question paper will be based on last 3 unit of syllabus.	10 marks
CES Quiz	Class test will be conducted based on the different aspects of C# Language Programming Constructs	5 marks
CES Quiz	Class test will be conducted based on the different aspects of C# Language Programming Constructs	5 MARKS
CES practical	Will have multiple choice, true false and fill in the blanks type question based on fundamental concepts of C# programming	5 marks
Attendance	Above 75% - 10 marks Below 75% - 0 mark	10 marks

Note:

All three CES will be mandatory. If any student misses anyone CES in that case the weightage of each CES would be 3.33 marks and if a student attempts all three CES then his/her best two CES will be considered, in that case the weightage would be 5 marks each.

Recommended/ Reference Text Books and Resources:

Text Book	**1.** The Complete Reference: C#; Herbert Schildt. edition Tata McGraw-Hill
Course Reading	**1.** C# Programming Wrox publications. **2.** E. Balgurusamy Programming with C#
References	1. Programming C# 5.0: Building Windows 8, Web, and Desktop Applications for the .NET 4.5 Framework by Ian Grifiths. 2. 2. C# 5.0 Unleashed by by Bart De Smet.
Internet Resource:	1. www.oracle.com 2. www.tutorialspoint.com/Charp 3. www.Csharpbeginnerstutorial.com 4. https://eclipse.org 5. https://visualstudio.org

Session Plan:

Module I: Introduction to C#

Session	Topics	Learning Outcomes
1	Dot Net Framework	LO1.To know the Dot Net framework and able to list out the features of thelanguage.CO1
2	Compilation and execution of C# program- CLR, CTS, CTS, Namespaces Declaration, Keywords in C#	LO1. Be able to understand how the compilation and execution is done and namespaces and keywords in C#. CO1
3	Writing and executing a simple C# Program Data Types Value type and reference type	LO1. Understand and execute first C# program and understand the value and reference type. CO1
4	Type conversion –boxing and unboxing	LO1. To understand type conversion and var variable. CO1
5	Operators in C# Control Statements and looping	LO1. Understand Operators in C# Statements. CO1
6	The base class library, the .net class library, garbage collection, assemblies	LO1. Understand the concept of libraries and Garbage collection. CO1
7	web services, COM, localization	LO1. Understand the concept of web services, COM, localization. CO1

Module II: Methods and Arrays

Session	Topics	Learning Outcomes
7	Evaluation of C# characteristics of C#, Application of C#	LO2. Understand characteristics of C#. CO2
8	Difference between C++ And C#, Java and C#	LO2. Understand the difference between C#, java and C++. CO2
9	Data types, variables, constants, C# statements	LO2. Understand the variables, constants, C# statements. CO2
10	OOPs concept and strings	LO2. Understand the OOPs concept in c#. CO2
11	Arrays and Multidimensional arrays	LO2. To understand multidimensional arrays. CO2
12	Jagged arrays	LO2.Understand jagged arrays. CO2
13	for each loop	LO2. Understand foreach loop. CO1, CO2
14	Mathematical in C#	LO2. Understand mathematical methods. CO2

Module III: Classes and objects.

Session	Topics	Learning Outcomes
15	static members	LO3.To Understand the static members. CO3
16	member initialization	LO3.Understand how to define classes in C#. CO3
17	Default constructor Parameterized constructor	LO3.Know default constructor Parameterized constructor.CO3
18	Overloaded Constructor	LO3. How overloading of constructor is done in C#. CO3
19	Static Constructor Copy Constructor	LO3. Understand the Static and copy Constructor. CO3
20	Destructors	LO3.Understand the concept of Destructors.CO3
21	Constant Members	LO3. Understand the constant members.CO3

22	"this" keyword	LO3. understand "this" keyword CO3
23	Properties	LO3. Understand the concept of Properties. CO3
24	Auto Implemented Properties	LO3. Understand Auto Implemented Proper
25	Member Initializer	LO3. Understand initializer. CO3 member
26	Nested classes	LO3. Understand Partial methods and partial classes. CO3
27	Indexers Program using Indexers	LO3. Understand and implement Indexers. CO3
28	Introduction to Inheritance and its types	LO3. Know inheritance and its types. CO3

Module IV: **Visual studio IDE features**

29	Introduction to Inheritance and its Types	LO4. Know inheritance and its types. CO4
30	Visual studio IDE features	LO4. Understand textbox, label, link label, status bar, checked list box, combo box, list box, list view, radio button etc. CO4
31	Introduction to Window forms, components, control	LO4. Knows the menu control, properties, methods, events of controls. CO4

Module V: **ADO.net**

32	ADO.net	LO5. Know the concept of ADO.NET . CO5
33	Building databases	LO5.Know how to build database. CO5
34	Database connection	LO5. Know how to make database connection. CO5

Module VI: Managing Console I/O operations

35	Console class	LO6. Can implement the console input, console output, formatted output, CO6
36	numeric formatting	LO6. Know numeric formattingCO6
37	Standard numeric format, Custom numeric format.	LO6. Understand standard numeric format, custom numeric format in C#. CO6
38	Managing Errors	LO6 Understand how to Manage Errors. CO6
39	Exceptions	LO6 Understand exceptions. CO6
40	Checked and unchecked exceptions	LO6 Understand Checked and unchecked exceptions. CO6

SHEKHAR KUNDRA

Mobile: 9811945176
E-Mail: shekhar.kundra@gmail.com

Experience and Accomplishments

In my career of 18+ years, I have 17+ Years of experience as Analysis, Designing, Coding as a freelancer, as well as 18+ Years of experience in Teaching.

Skill Sets

As Teacher

- ✓ **Operating Systems:** Windows, Linux, Unix.
- ✓ **Databases:** Oracle8i & 9i, MS Access, SQL Server 2008 to 2014, MongoDB
- ✓ **Programming Languages:** C, C with Data structure, C++, C++ with Data structure, Java, Shell Programming in Unix and Linux, VB 6.0, VB .Net, C#, ADO .Net, MVC Concept, ASP .Net Core and currently working on Node JS as MEAN full-stack Web developer.
- ✓ **System Concepts:** Management information system, Software Engineering, OOAD, etc,
- ✓ **Knowledge of:** Pfsens as Router, AWS Working, API for different servers, HTML, CSS, jQuery, different tools for SEO and to make the website light and responsive, etc,

As a Software Engineer

Software engineers, I develop systems and software for businesses. A software engineers worked with users to determine their software needs. Designing, developing and testing a system or application according to the users' specifications.

NISHA MALHOTRA

Mobile: 9899995540
E-Mail: nisha.jan89@gmail.com

Academic Qualifications

- I have done M. Tech. from Netaji Subhas Institute of Technology (NSIT) , Delhi University and B. Tech (Computer Science Engineering) from N.C college of Engineering, Kurukshetra University.
- Also done Certifications of CISCO:CISCO Certified Network Associate (CCNA).

Teaching Subjects

- Java Programming
- C# programming
- Linux operating system
- Data Structures
- Database Management System
- Operating System
- Software Engineering
- C, C++
- Object oriented programming and analysis
- Compiler design

Work Experience

- Done training and developed Joining Report Module in Integrated Management Information Dissemination System (IMIS) from Defence Research and Development Organization (DRDO), Ministry of Defence, Delhi.
- Done training in Virtual Network Computing from Information Technology Department, AAI (Airports Authority of India), Safdurjung Airport, New Delhi
- Worked as a Lecturer in Computer Science Department at The Gate Academy Institute, Pitampura New Delhi.
- Worked at IITM, affiliated with G.G.S Indraprastha University, Delhi as an Assistant Professor
- Works at Bharati Vidyapeeth Institute of Management and Research, Paschim Vihar, Delhi, as a visiting faculty
- Also works at Bharati Vidyapeeth College of Engineering Paschim Vihar, Delhi, as a visiting faculty

STUDY NOTES

UNIT 1

The .net Framework:

- **Introduction**
- **common language runtime**
- **common type system**
- **common language specification**
- **the base class library, the .net class library**
- **Intermediate language, just in time compilation**
- **garbage collection, assemblies**
- **web services, COM, localization**

.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.

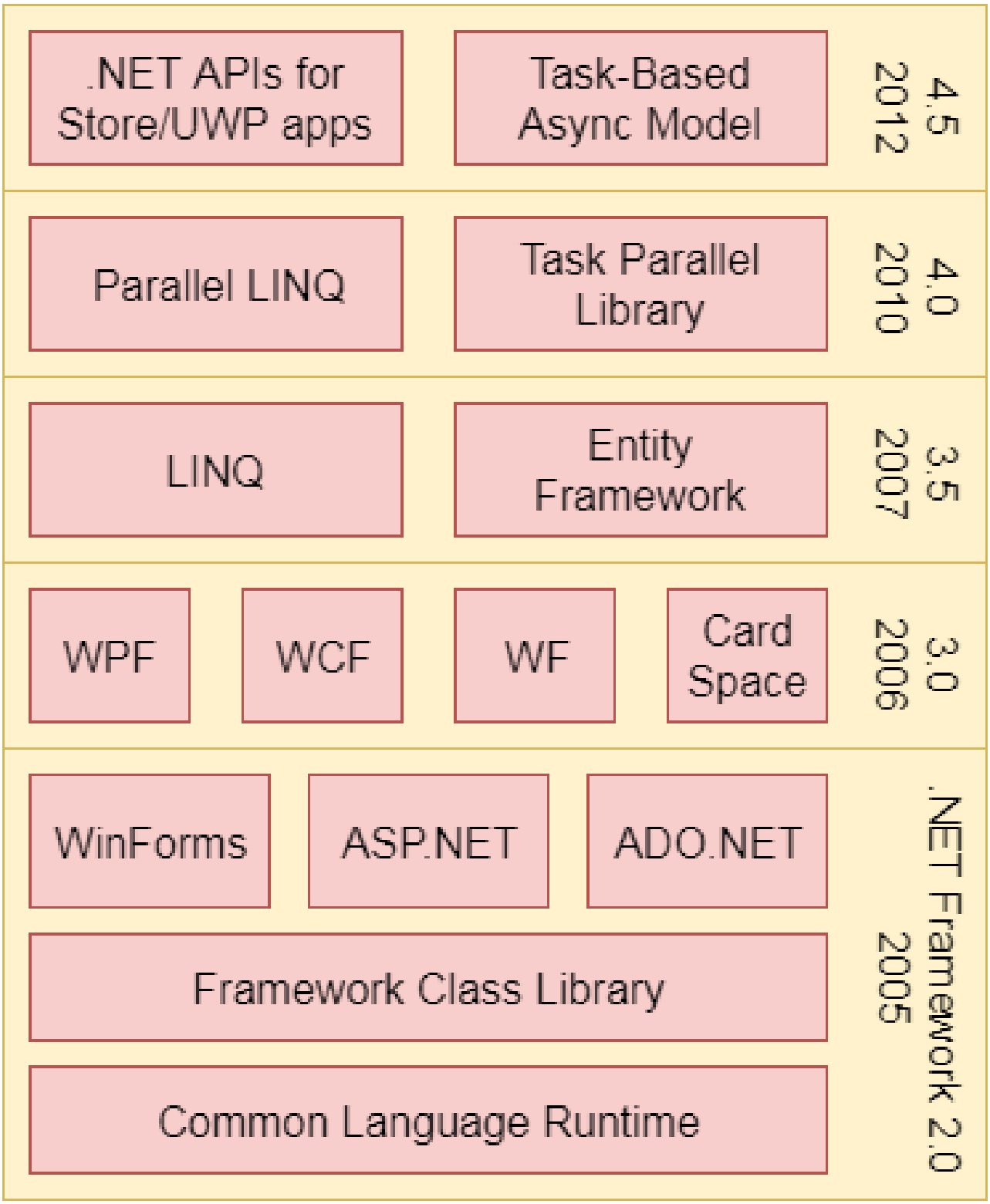
This framework provides various services like memory management, networking, security, memory management, and type-safety.

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.

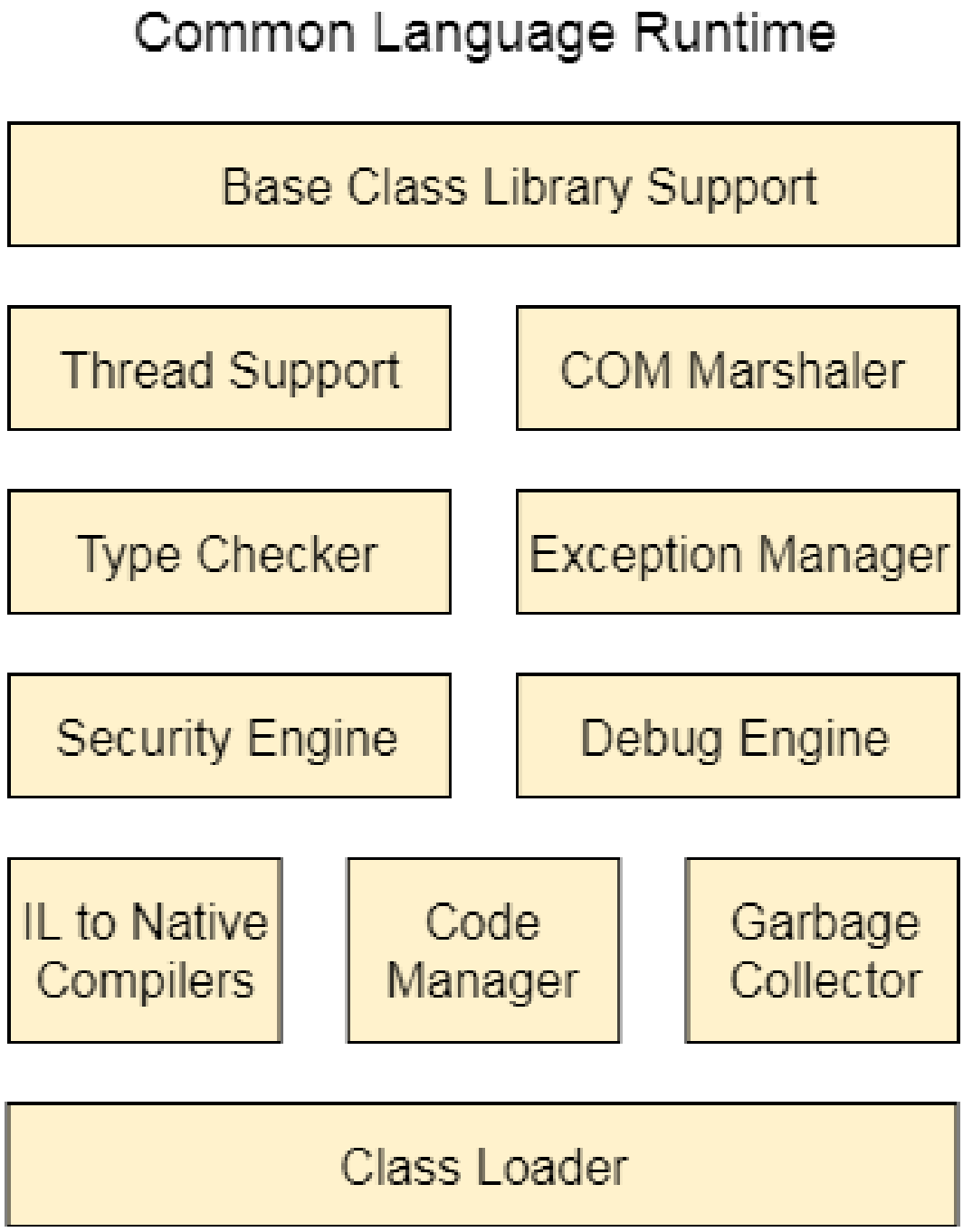
The .NET Framework is composed of four main components:

1. Common Language Runtime (CLR)
2. Framework Class Library (FCL),
3. Core Languages (WinForms, ASP.NET, and ADO.NET), and
4. 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
Framework Class Library
Common Language Runtime
.NET Framework 2.0 2005

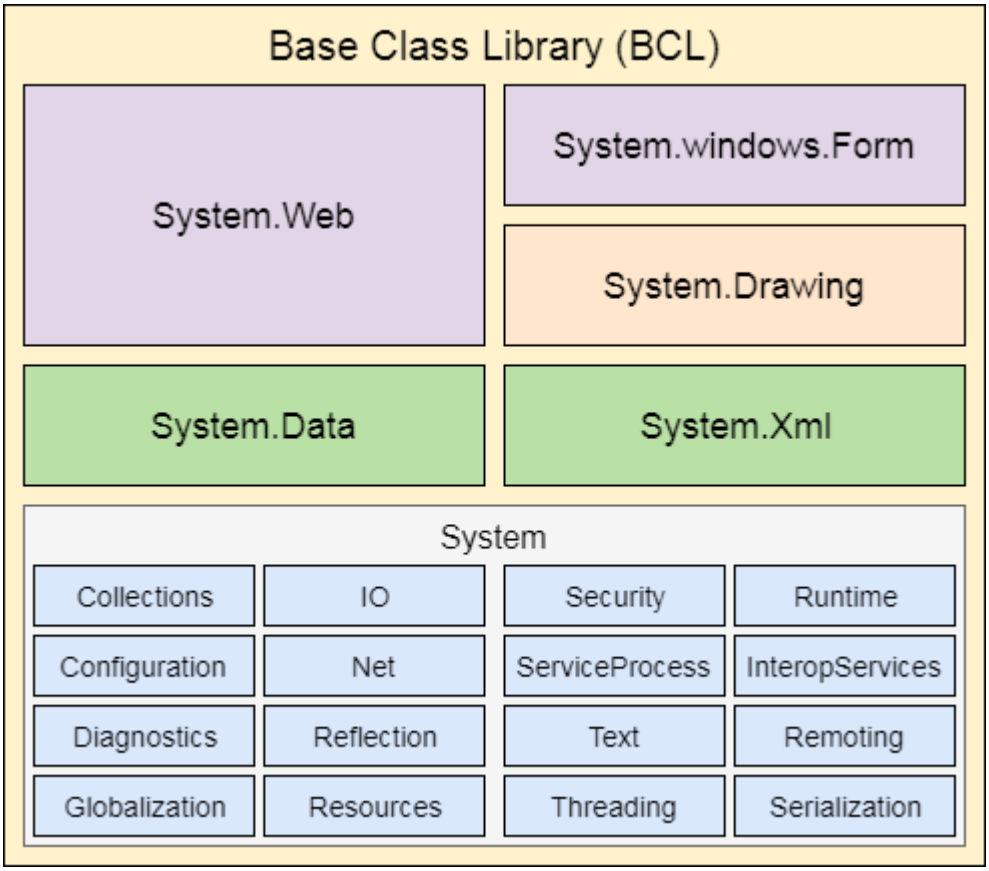
CLR (Common Language Runtime)

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 is given below:



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 table 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 developer’s efforts 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.

.NET Common Language Runtime (CLR)

.NET CLR is a run-time environment that manages and executes the code written in any .NET programming language.

It converts code into native code which further can be executed by the CPU.

.NET CLR Functions

Following are the functions of the CLR.

- It converts the program into native code.
- Handles Exceptions
- Provides type-safety
- Memory management
- Provides security
- Improved performance
- Language independent
- Platform independent
- Garbage collection
- Provides language features such as inheritance, interfaces, and overloading for object-oriented programming's.

.NET CLR Versions

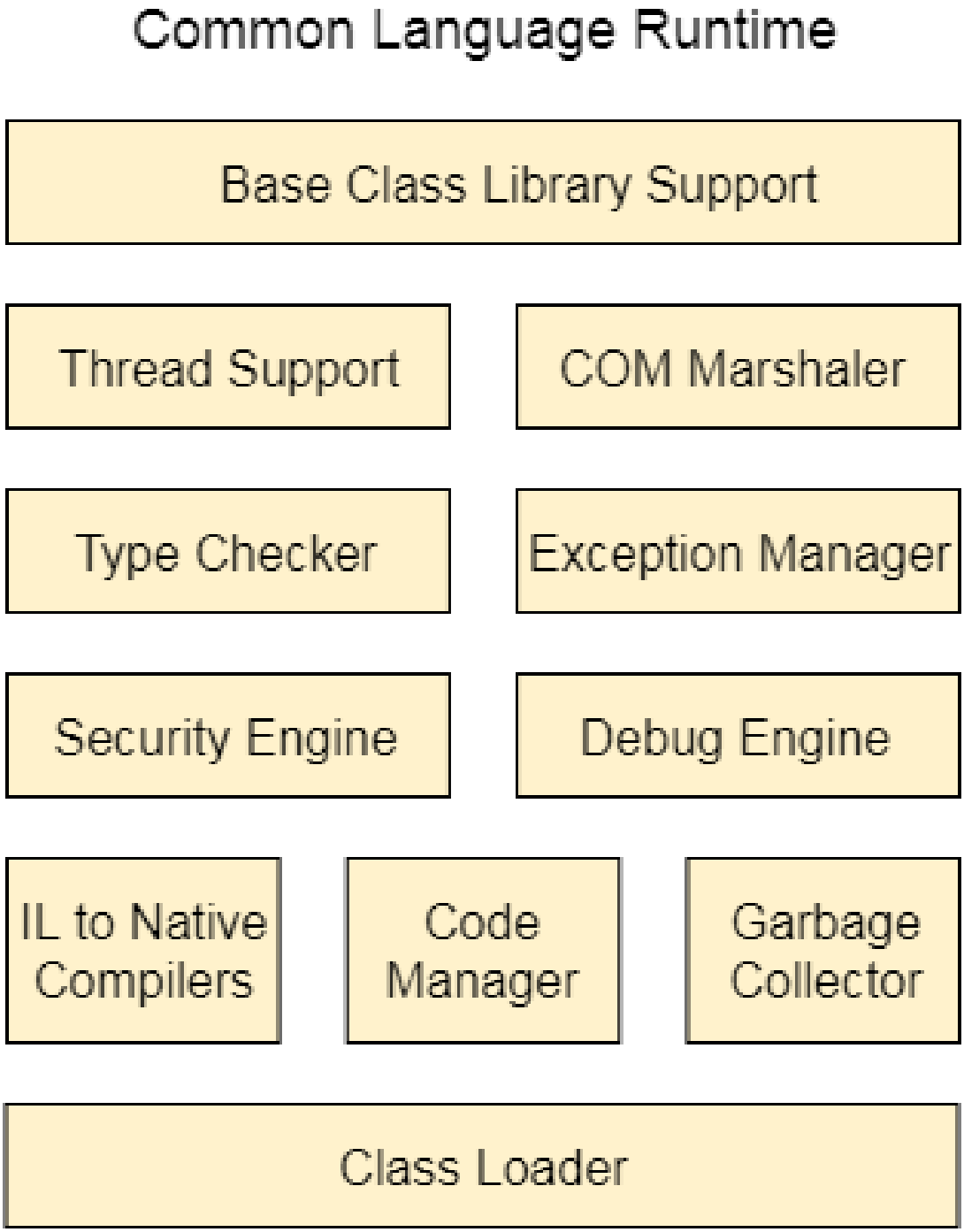
The CLR updates itself time to time to provide better performance.

.NET version	CLR version
1.0	1.0
1.1	1.1
2.0	2.0
3.0	2.0

3.5	2.0
4	4
4.5	4
4.6	4
4.6	4

.NET CLR Structure

Following is the component structure of Common Language Runtime.


Common Language Runtime
Base Class Library Support
Thread Support
COM Marshaler
Type Checker
Exception Manager
Security Engine
Debug Engine
IL to Native Compilers
Code Manager
Garbage Collector
Class Loader

Base Class Library Support

It is a class library that provides support of classes to the .NET application.

Thread Support

It manages the parallel execution of the multi-threaded application.

COM

It provides communication between the COM objects and the application.

Type Checker

It checks types used in the application and verifies that they match to the standards provided by the CLR.

Code Manager

It manages code at execution run-time.

Garbage Collector

It releases the unused memory and allocates it to a new application.

Exception Handler

It handles the exception at runtime to avoid application failure.

Class Loader

It is used to load all classes at run time.

.NET Framework Class Library

.NET Framework Class Library is the collection of classes, namespaces, interfaces and value types that are used for .NET applications.

It contains thousands of classes that supports the following functions.

- Base and user-defined data types
- Support for exceptions handling
- input/output and stream operations
- Communications with the underlying system
- Access to data

- Ability to create Windows-based GUI applications
- Ability to create web-client and server applications
- Support for creating web services

.NET Framework Class Library Namespaces

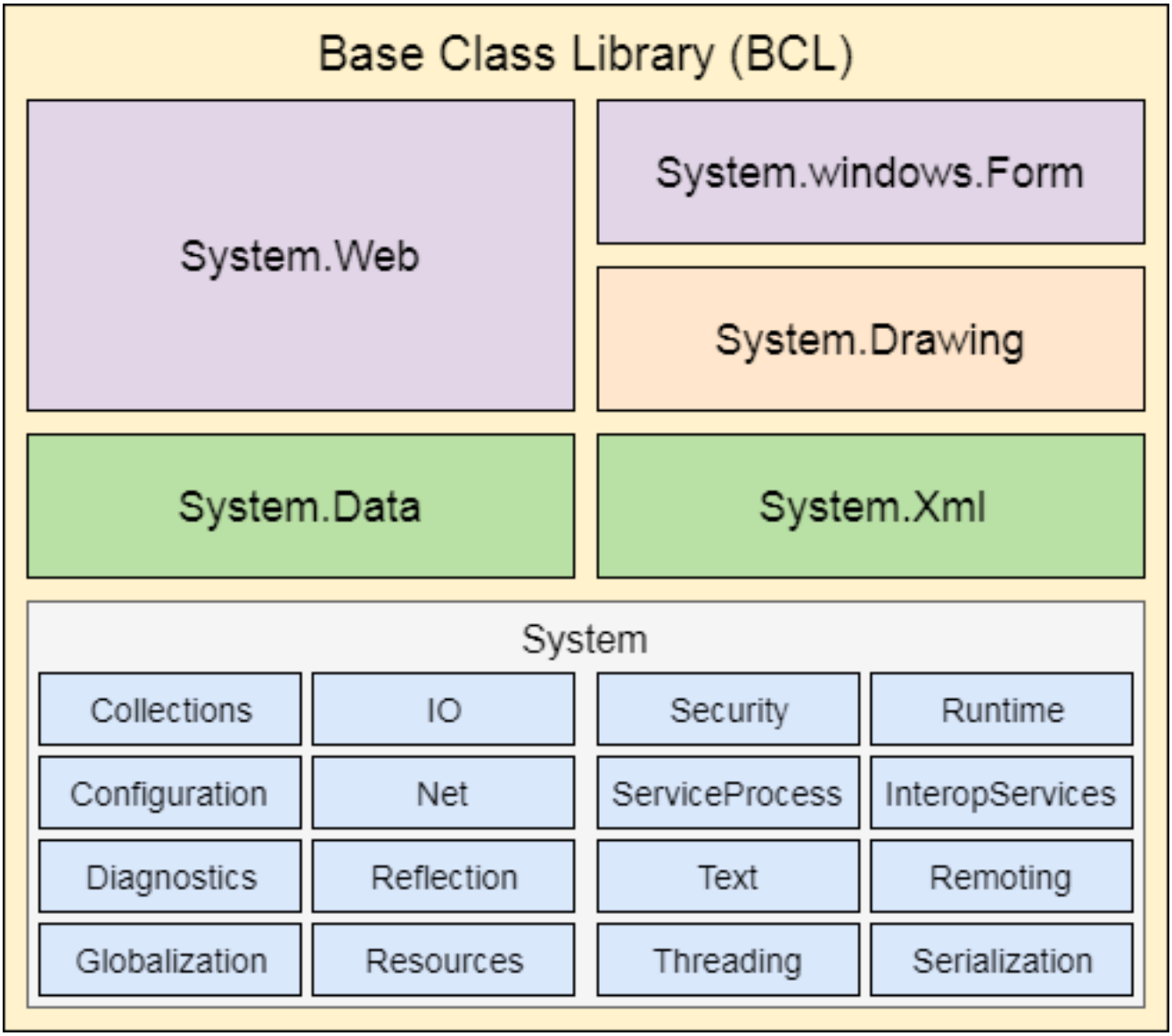
Following are the commonly used namespaces that contains useful classes and interfaces and defined in Framework Class Library.

Namespaces	Description
System	It includes all common datatypes, string values, arrays and methods for data conversion.
System.Data, System.Data.Common, System.Data.OleDb, System.Data.SqlClient, System.Data.SqlTypes	These are used to access a database, perform commands on a database and retrieve database.
System.IO, System.DirectoryServices, System.IO.IsolatedStorage	These are used to access, read and write files.
System.Diagnostics	It is used to debug and trace the execution of an application.
System.Net, System.Net.Sockets	These are used to communicate over the Internet when creating peer-to-peer applications.
System.Windows.Forms, System.Windows.Forms.Design	These namespaces are used to create Windows-based applications using Windows user interface components.

System.Web, System.WebCaching, System.Web.UI, System.Web.UI.Design, System.Web.UI.WebControls, System.Web.UI.HtmlControls, System.Web.Configuration, System.Web.Hosting, System.Web.Mail, System.Web.SessionState	These are used to create ASP. NET Web applications that run over the web.
System.Web.Services, System.Web.Services.Description, System.Web.Services.Configuration, System.Web.Services.Discovery, System.Web.Services.Protocols	These are used to create XML Web services and components that can be published over the web.
System.Security, System.Security.Permissions, System.Security.Policy, System.WebSecurity, System.Security.Cryptography	These are used for authentication, authorization, and encryption purpose.
System.Xml, System.Xml.Schema, System.Xml.Serialization, System.Xml.XPath, System.Xml.Xsl	These namespaces are used to create and access XML files.

.NET Framework Base Class Library

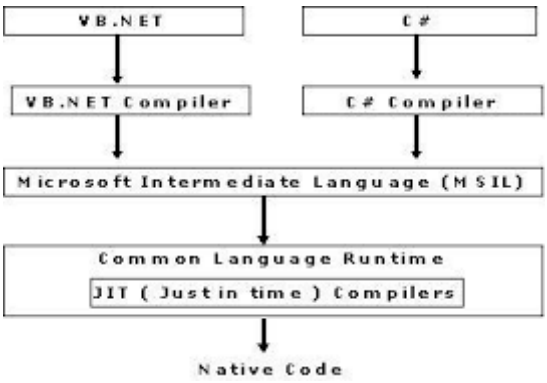
.NET Base Class Library is the sub part of the Framework that provides library support to Common Language Runtime to work properly. It includes the System namespace and core types of the .NET framework.



MSIL

Microsoft Intermediate Language (MSIL) is a language used as the output of a number of compilers (C#, VB, .NET, and so forth). **The IL Dasm** (Intermediate Language Disassembler) program that ships with the .NET Framework SDK (Framework SDK\Bin\ildasm.exe) allows the user to see MSIL code in human-readable format. By using this utility, we can open any .NET executable file (EXE or DLL) and see MSIL code.

The **IL Asm** program (Intermediate Language Assembler) generates an executable file from the MSIL language. We can find this program in the WINNT\Microsoft.NET\Framework\vn.nn.nn directory.
Any Visual C++ programmer starting with .NET development is interested in what happens in the low level of the .NET Framework. Learning MSIL gives a user the chance to understand some things that are hidden from a programmer working with C# or VB.NET. Knowing MSIL gives more power to a .NET programmer. We never need to write programs in MSIL directly, but in some difficult cases it is very useful to open the MSIL code in ILDasm and see how things are done.



A MSIL reference in DOC format is available to a .NET developer and may be found in the Framework SDK directory:

- Framework SDK\Tool Developers Guide\docs\Partition II Metadata.doc **(Metadata Definition and Semantics)**. In this file, I found a description of all MSIL directives such as **.entrypoint**, **.locals**, and so on.
- Framework SDK\Tool Developers Guide\docs\Partition III CIL.doc **(CIL Instruction Set)** contains a full list of the MSIL commands.

I also used in my work in an ILDAsm tutorial from MSDN and an excellent article in the May 2001 issue of MSDN Magazine: "ILDASM is Your New Best Friend" by **John Robbins**.
I think the best way to learn the language is to write some programs in it. This is a reason I decided to make several small MSIL programs. Actually, I didn't write this code—the C# compiler generated it. I made some minor changes and added a lot of notes describing how MSIL is working.

Reading the sample projects attached to this article may help a .NET programmer understand Intermediate Language and easily read MSIL code when this is necessary.

General Information

All operations in MSIL are executed on the stack. When a function is called, its parameters and local variables are allocated on the stack. Function code starting from this stack state may push some values onto the stack, make operations with these values, and pop values from the stack.

Execution of both MSIL commands and functions is done in three steps:

1. Push command operands or function parameters onto the stack.
2. Execute the MSIL command or call function. The command or function pops their operands (parameters) from the stack and pushes onto the stack result (return value).
3. Read result from the stack.

Steps 1 and 3 are optional. For example, the **void** function doesn't push a return value to the stack.
The stack contains objects of value types and references to objects of reference type. Reference type objects are kept in the heap.

MSIL commands used to push values onto the stack are called **ld...** (load). Commands used to pop values from the stack are called **St...** (store), because values are stored in variables. Therefore, we will call the **push** operation **loading** and the **pop** operation **storing**.

JIT

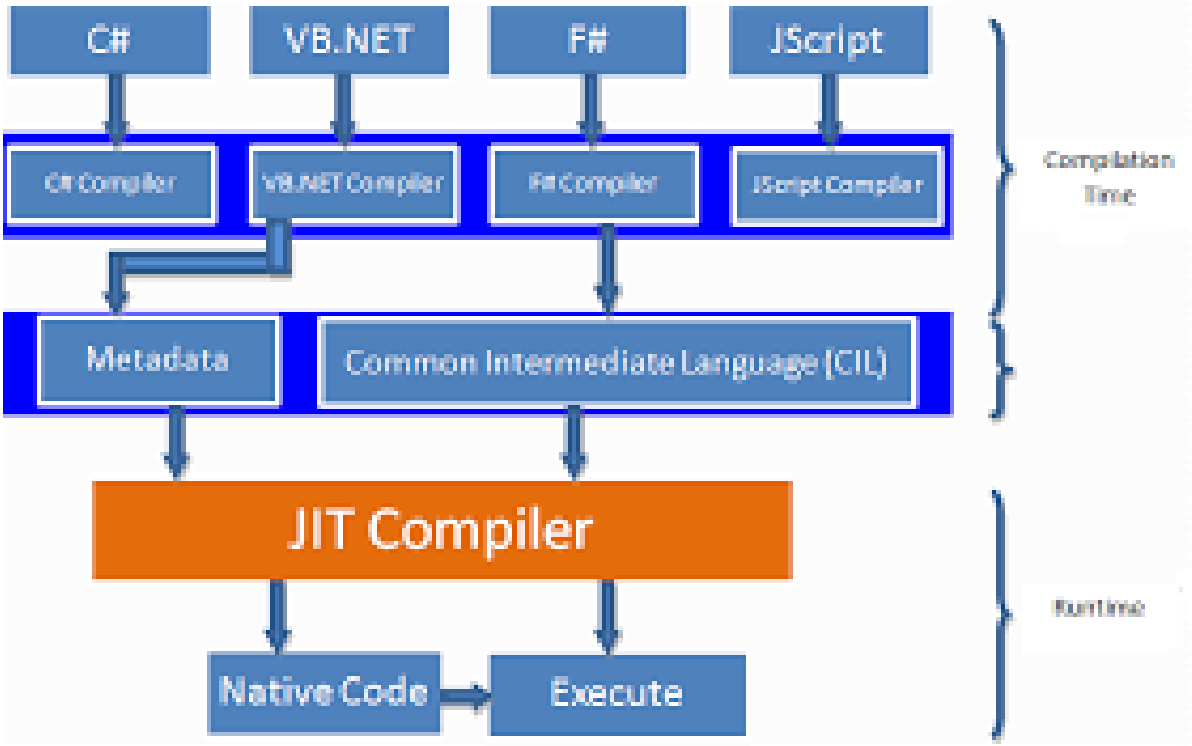
In the .NET Framework, all the Microsoft .NET languages use a Common Language Runtime, which solves the problem of installing separate runtimes for each of the programming languages. When the Microsoft .NET Common Language Runtime is installed on a computer then it can run any language that is Microsoft .NET compatible. Before the Microsoft Intermediate Language (MSIL) can be executed, it must be converted by a .NET Framework Just-In-Time (JIT) compiler to native code, which is CPU-specific code that runs on the same computer architecture as the JIT compiler.

JIT (JUST-IN-TIME) COMPILER

A Web Service or Web Forms file must be compiled to run within the CLR. Compilation can be implicit or explicit. Although you could explicitly call the appropriate compiler to compile your Web Service or Web Forms files, it is easier to allow the file to be complied implicitly. Implicit compilation occurs when you request the .asmx via HTTP-SOAP, HTTP-GET, or HTTP-POST. The parser (xsp.exe) determines whether a current version of the assembly resides in memory or in the disk. If it cannot use an existing version, the parser makes the appropriate call to the respective compiler (as you designated in the Class property of the .asmx page).

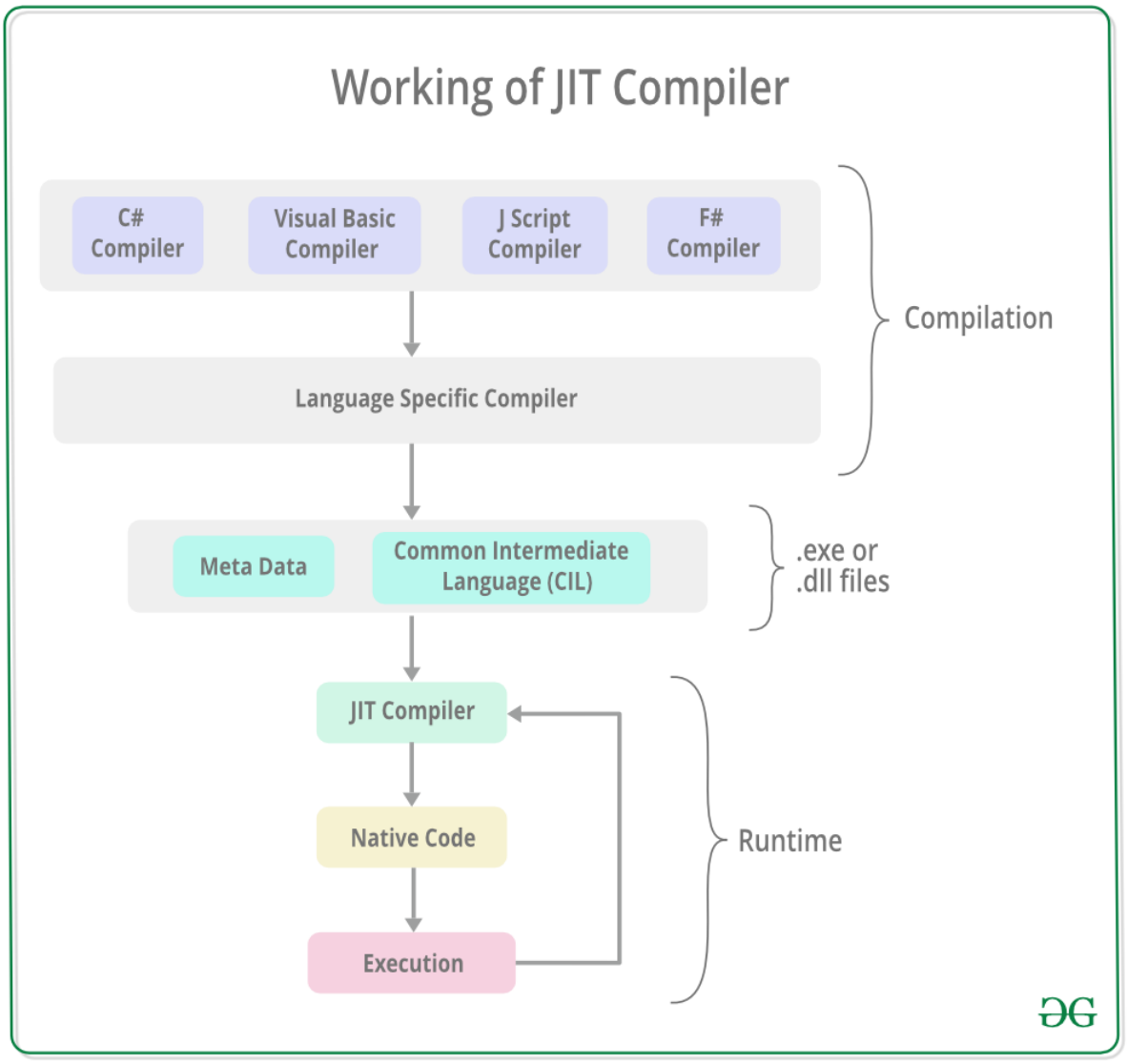
When the Web Service (or Web Forms page) is implicitly compiled, it is actually compiled twice. On the first pass, it is compiled into IL. On the second pass, the Web Service (now an assembly in IL) is compiled into machine language. This process is called Just-In-Time JIT compilation because it does not occur until the assembly is on the target machine. The reason you do not compile it ahead of time is so that the specific JITter for your OS and processor type can be used. As a result, the assembly is compiled into the fastest possible machine language code, optimized and enhanced for your specific configuration. It also enables you to compile once and then run on any number of operating systems.

How JIT Works?



Before MSIL(MS Intermediate Language) can be executed, it must converted by .net Framework Just in time (JIT) compiler to native code, which is CPU specific code that run on some computer architecture as the JIT compiler. Rather than using time and memory to convert all the MSIL in portable executable (PE) file to native code, it converts the MSIL as it is needed during execution and stored in resulting native code so it is accessible for subsequent calls.

The runtime supplies another mode of compilation called install-time code generation. The install-time code generation mode converts MSIL to native code just as the regular JIT compiler does, but it converts larger units of code at a time, storing the resulting native code for use when the assembly is subsequently loaded and executed. As part of compiling MSIL to native code, code must pass a verification process unless an administrator has established a security policy that allows code to bypass verification. Verification examines MSIL and metadata to find out whether the code can be determined to be type safe, which means that it is known to access only the memory locations it is authorized to access.



JIT Types

In Microsoft .NET there are three types of JIT (Just-In-Time) compilers which are Explained as Under,

- Pre-JIT Compiler (Compiles entire code into native code completely)
- Econo JIT Compiler (Compiles code part by part freeing when required)
- Normal JIT Compiler (Compiles only that part of code when called and places in cache

Description

- *Pre-JIT COMPILER*

  Pre-JIT compiles complete source code into native code in a single compilation cycle. This is done at the time of deployment of the application.

- *Econo-JIT COMPILER*

  Econo-JIT compiles only those methods that are called at runtime. However, these compiled methods are removed when they are not required.

- *Normal-JIT COMPILER*

  Normal-JIT compiles only those methods that are called at runtime. These methods are compiled the first time they are called, and then they are stored in cache. When the same methods are called again, the compiled code from cache is used for execution.

These methods are compiled the first time they are called, and then they are stored in cache. When the same methods are called again, the compiled code from cache is used for execution.
Brought to you by:

Garbage Collection

Automatic memory management is made possible by **Garbage Collection in .NET Framework**. When a class object is created at runtime, certain memory space is allocated to it in the heap memory. However, after all the actions related to the object are completed in the program, the memory space allocated to it is a waste as it cannot be used. In this case, garbage collection is very useful as it automatically releases the memory space after it is no longer required.

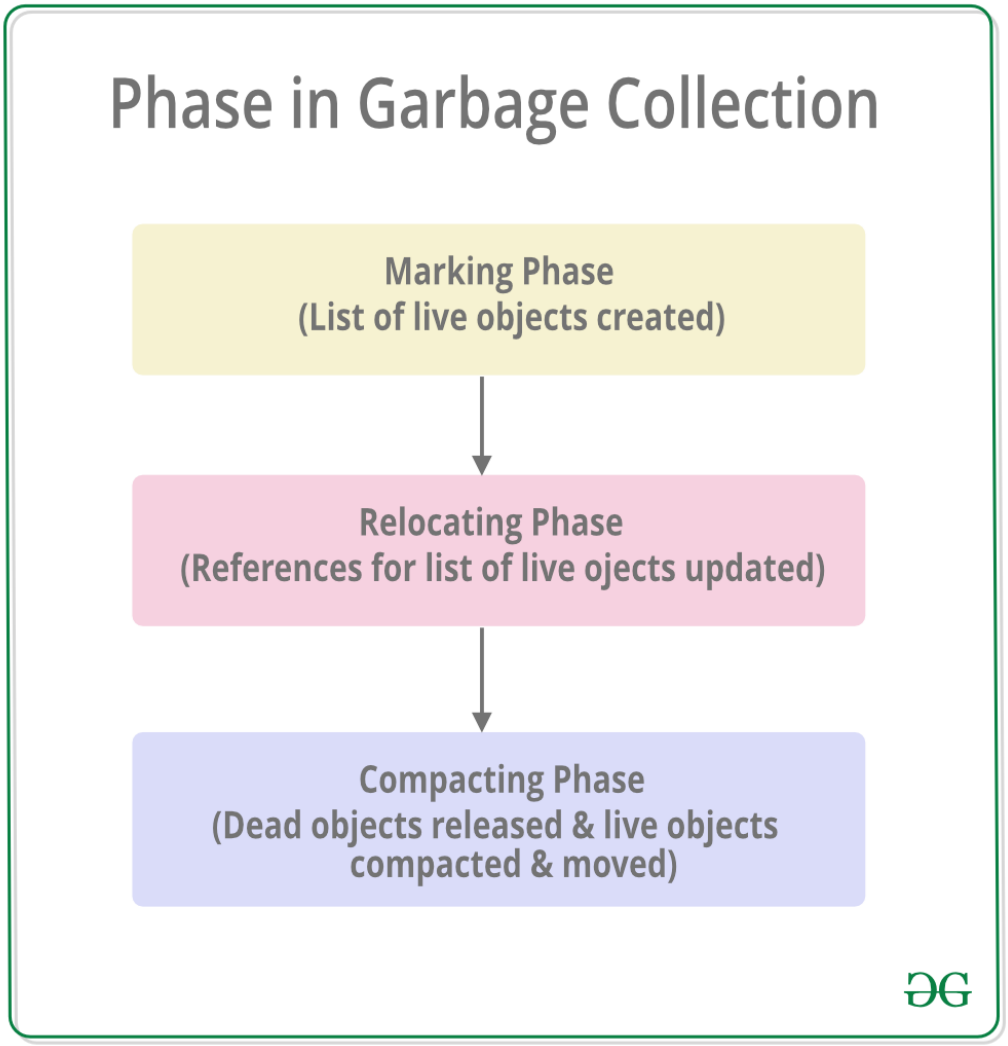
Garbage collection will always work on **Managed Heap** and internally it has an Engine which is known as the **Optimization Engine**.

Garbage Collection occurs if at least one of multiple conditions is satisfied. These conditions are given as follows:

- If the system has low physical memory, then garbage collection is necessary.
- If the memory allocated to various objects in the heap memory exceeds a pre-set threshold, then garbage collection occurs.
- If the *GC.Collect* method is called, then garbage collection occurs. However, this method is only called under unusual situations as normally garbage collector runs automatically.

Phases in Garbage Collection

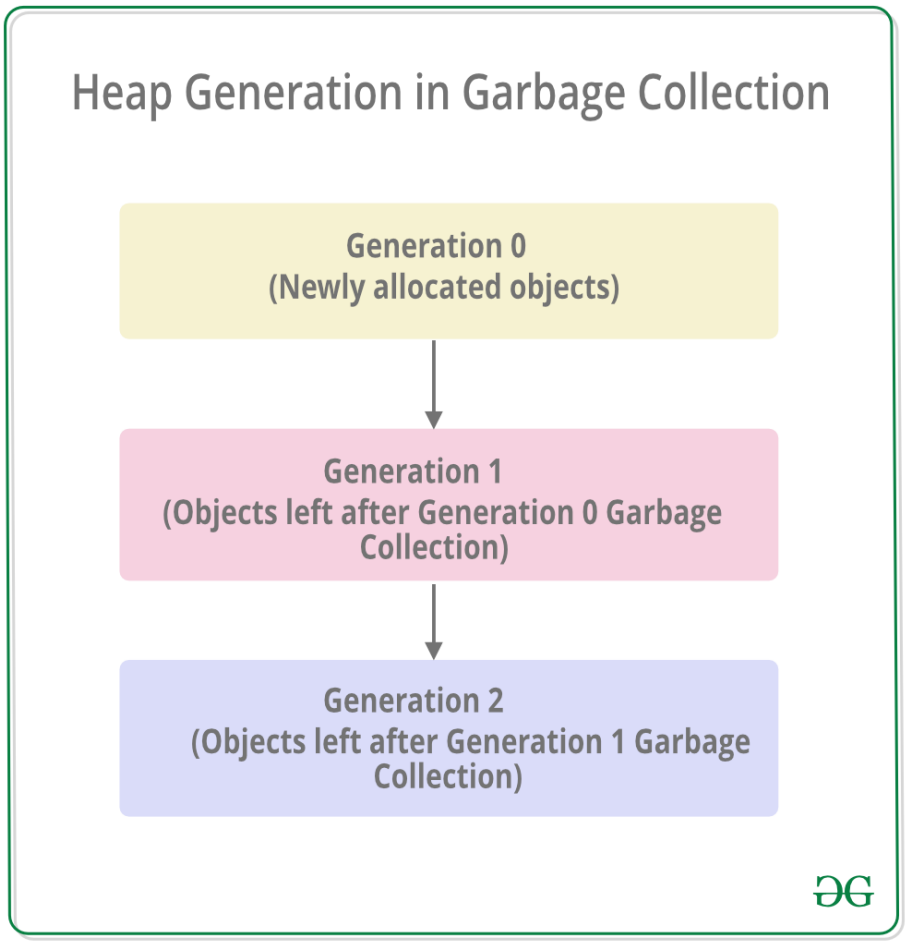
There are mainly **3** phases in garbage collection. Details about these are given as follows:



1. **Marking Phase:** A list of all the live objects is created during the marking phase. This is done by following the references from all the root objects. All of the objects that are not on the list of live objects are potentially deleted from the heap memory.
2. **Relocating Phase:** The references of all the objects that were on the list of all the live objects are updated in the relocating phase so that they point to the new location where the objects will be relocated to in the compacting phase.
3. **Compacting Phase:** The heap gets compacted in the compacting phase as the space occupied by the dead objects is released and the live objects remaining are moved. All the live objects that remain after the garbage collection are moved towards the older end of the heap memory in their original order.

Heap Generations in Garbage Collection

The heap memory is organized into 3 generations so that various objects with different lifetimes can be handled appropriately during garbage collection. The memory to each Generation will be given by the **Common Language Runtime (CLR)** depending on the project size. Internally, Optimization Engine will call the *Collection Means Method* to select which objects will go into Generation 1 or Generation 2.



- **Generation 0:** All the short-lived objects such as temporary variables are contained in the generation 0 of the heap memory. All the newly allocated objects are also generation 0 objects implicitly unless they are large objects. In general, the frequency of garbage collection is the highest in generation 0.
- **Generation 1:** If space occupied by some generation 0 objects that are not released in a garbage collection run, then these objects get moved to generation 1. The objects in this generation are a sort of buffer between the short-lived objects in generation 0 and the long-lived objects in generation 2.
- **Generation 2:** If space occupied by some generation 1 objects that are not released in the next garbage collection run, then these objects get moved to generation 2. The objects in generation 2 are long lived such as static objects as they remain in the heap memory for the whole process duration.

Assembly

The .NET assembly is the standard for components developed with the Microsoft.NET. Dot NET assemblies may or may not be executable, i.e., they might exist as the executable (.exe) file or dynamic link library (DLL) file. All the .NET assemblies contain the definition of types, versioning information for the type, meta-data, and manifest. The designers of .NET have worked a lot on the component (assembly) resolution.

An assembly can be a single file or it may consist of the multiple files. In the case of multi-file, there is one master module containing the manifest while other assemblies exist as non-manifest modules. A module in .NET is a subpart of a multi-file .NET assembly. Assembly is one of the most interesting and extremely useful areas of .NET architecture along with reflections and attributes.

.NET supports three kinds of assemblies:

1. private
2. shared
3. satellite

Private Assembly

Private assembly requires us to copy separately in all application folders where we want to use that assembly's functionalities; without copying, we cannot access the private assembly features and power. Private assembly means every time we have one, we exclusively copy into the BIN folder of each application folder.

Public Assembly

Public assembly is not required to copy separately into all application folders. Public assembly is also called Shared Assembly. Only one copy is required in system level, there is no need to copy the assembly into the application folder.

Public assembly should install in GAC.

Shared assemblies (also called strong named assemblies) are copied to a single location (usually the Global assembly cache). For all calling assemblies within the same application, the same copy of the shared assembly is used from its original location. Hence, shared assemblies are not copied in the private folders of

each calling assembly. Each shared assembly has a four-part name including its face name, version, public key token, and culture information. The public key token and version information makes it almost impossible for two different assemblies with the same name or for two similar assemblies with a different version to mix with each other.

W**ebServices**

Web service is simply an application that exposes a Web-accessible API. That means you can invoke this application programmatically over the Web. Using SOAP Protocol .

XML+HTTP =SOAP (Simple Object Access protocol)

Web services allow applications to share data.

Web services can be called across platforms and operating systems regardless of programming language.

.NET is Microsoft's platform for XML Web services.

Using SOAP you can invoke the Webservices. (HTTP-GET, HTTP-POST, SOAP)

SOAP -Simple Object Access protocol is message-based protocol. Based on Request and Response. Here I am not going to explain more details about SOAP.

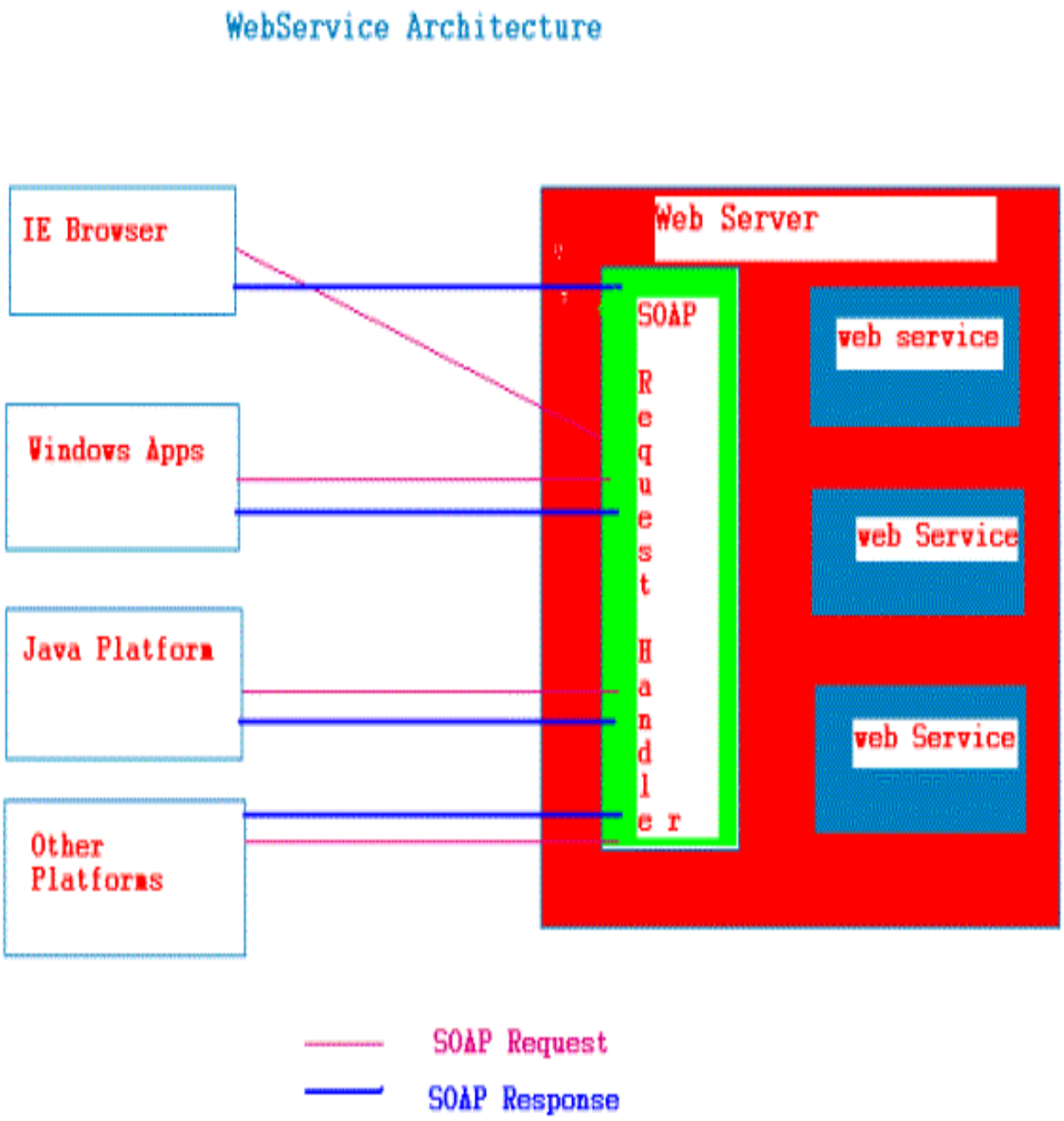
Web Service Application

Like Pay per view Channel we can make Software as pay per use, using Web service. For example, let say Info vision Inc developed Expensive Software for 3D Virtual Modeling. Lot of company does not want acquire the licensee because it is expensive and they need to pay for Support etc... instead of this if Info vision makes this software as a Webservice, most of the company will use as pay per use.

Another example Credit card validation we can expose as a webservice. (Good example for webservice).

Using VisualStudio.NET we can easily create Webservices. ASP.NET Webservice this project type will be used to create Webservice.

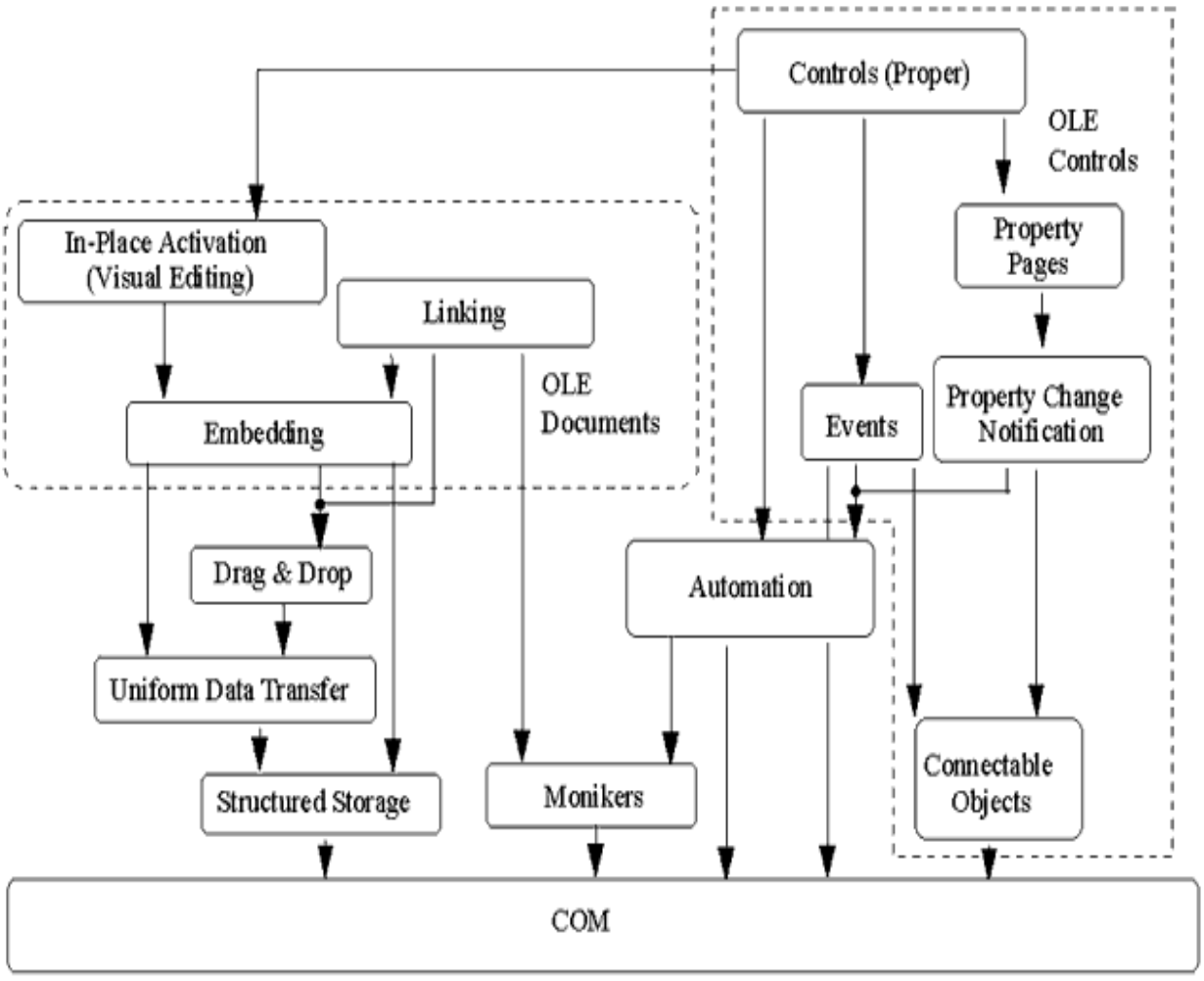
Web Services Architecture



Simple Object Access Protocol

COM

The Component Object Model (COM) is a software architecture that allows applications to be built from binary software components. COM is the underlying architecture that forms the foundation for higher-level software services, like those provided by OLE. OLE services span various aspects of commonly needed system functionality, including compound documents, custom controls, interapplication scripting, data transfer, and other software interactions.



OLE technologies build on one another, with COM as the foundation.

These services provide distinctly different functionality to the user. However they share a fundamental requirement for a mechanism that allows binary software components, derived from any combination of pre-existing customers' components and components from different software vendors, to connect to and communicate with each other in a well-defined manner. This mechanism is supplied by COM, a software architecture that does the following:

- Defines a binary standard for component interoperability
- Is programming-language-independent
- Is provided on multiple platforms (Microsoft® Windows®, Windows 95, Windows NT™, Apple® Macintosh®, and many varieties of UNIX®)
- Provides for robust evolution of component-based applications and systems
- Is extensible by developers in a consistent manner
- Uses a single programming model for components to communicate within the same process, and also across process and network boundaries
- Allows for shared memory management between components
- Provides rich error and status reporting
- Allows dynamic loading and unloading of components

It is important to note that COM is a general architecture for component software. Although Microsoft is applying COM to address specific areas such as controls, compound documents, automation, data transfer, storage and naming, and others, any developer can take advantage of the structure and foundation that COM provides.

How does COM enable interoperability? What makes it such a useful and unifying model? To address these questions, it will be helpful to first define the basic COM design principles and architectural concepts. In doing so, we will examine the specific problems that COM is meant to solve, and how COM provides solutions for these problems.

The Component Software Problem

The most fundamental problem that COM solves is: How can a system be designed so that binary executables from different vendors, written in different parts of the world and at different times, are able to interoperate? To solve this problem, we must first find answers to these four questions:

- **Basic interoperability.** How can developers create their own unique binary components, yet be assured that these binary components will interoperate with other binary components built by different developers?
- **Versioning.** How can one system component be upgraded without requiring all the system components to be upgraded?
- **Language independence.** How can components written in different languages communicate?
- **Transparent cross-process interoperability.** How can we give developers the flexibility to write components to run in-process or cross-process, and even cross-network, using one simple programming model?

Additionally, high performance is a requirement for a component software architecture. Although cross-process and cross-network transparency is a laudable goal, it is critical for the commercial success of a binary component marketplace that components interacting within the same address space be able to use each other's services without any undue "system" overhead. Otherwise, the components will not realistically be scalable down to very small, lightweight pieces of software equivalent to C++ classes or graphical user interface (GUI) controls.

COM Fundamentals

The Component Object Model defines several fundamental concepts that provide the model's structural underpinnings. These include:

- A binary standard for function calling between components.
- A provision for strongly-typed groupings of functions into interfaces.
- A base interface providing:

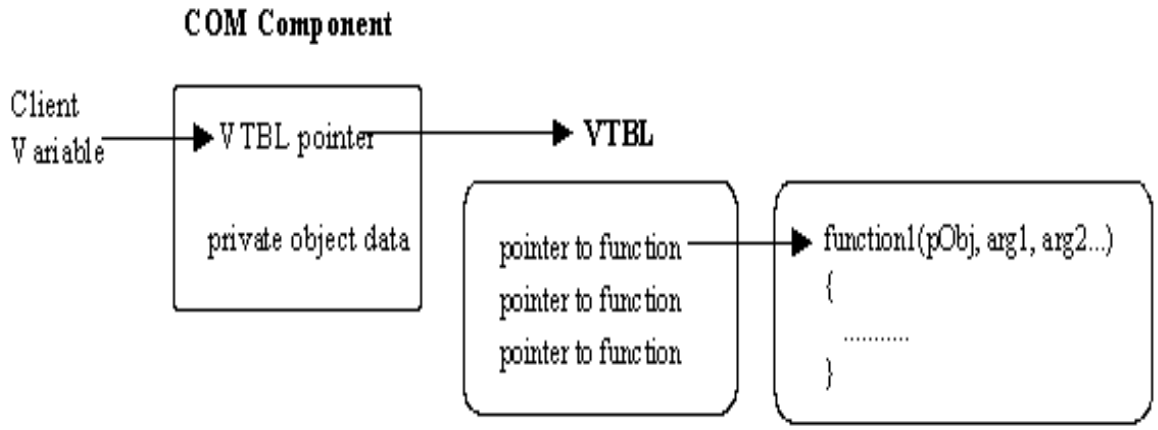
- A way for components to dynamically discover the interfaces implemented by other components.
- Reference counting to allow components to track their own lifetime and delete themselves when appropriate.

- A mechanism to identify components and their interfaces uniquely, worldwide.
- A "component loader" to set up component interactions and, additionally (in the cross-process and cross-network cases), to help manage component interactions.

Binary Standard

For any given platform (hardware and operating system combination), COM defines a standard way to lay out virtual function tables (vtables) in memory, and a standard way to call functions through the vtables. Thus, any language that can call functions via pointers (C, C++, Smalltalk, Ada, and even BASIC) all can be used to write components that can interoperate with other components written to the same binary standard. Indirection (the client holds a pointer to a vtable) allows for vtable sharing among multiple instances of the same object class. On a system with hundreds of object instances, vtable sharing can reduce memory requirements considerably, because additional vtables pointing to the same component instance consume much less memory than multiple instances of the same component.

Localization



Developing a world-ready application, including an application that can be localized into one or more languages, involves three steps: globalization, localizability review, and localization.

Globalization

This step involves designing and coding an application that is culture-neutral and language-neutral, and that supports localized user interfaces and regional data for all users. It involves making design and programming decisions that are not based on culture-specific assumptions. While a globalized application is not localized, it nevertheless is designed and written so that it can be subsequently localized into one or more languages with relative ease.

Localizability review

This step involves reviewing an application's code and design to ensure that it can be localized easily and to identify potential roadblocks for localization, and verifying that the application's executable code is separated from its resources. If the globalization stage was effective, the localizability review will confirm the design and coding choices made during globalization. The localizability stage may also identify any remaining issues so that an application's source code doesn't have to be modified during the localization stage.

Localization

This step involves customizing an application for specific cultures or regions. If the globalization and localizability steps have been performed correctly, localization consists primarily of translating the user interface.

Following these three steps provides two advantages:

- It frees you from having to retrofit an application that is designed to support a single culture, such as U.S. English, to support additional cultures.
- It results in localized applications that are more stable and have fewer bugs.

.NET provides extensive support for the development of world-ready and localized applications. In particular, many type members in the .NET class library aid globalization by returning values that reflect the conventions of either the current user's culture or a specified culture. Also, .NET supports satellite assemblies, which facilitate the process of localizing an application.

MCQ

Which of the following is not a .NET compatible language?

A. VB

B. C#

C. Java

D. F#

ANSWER: C

Features of automatic memory management in .Net

A. Allocating memory

B. Releasing memory

C. Implementing finalizers

D. All of the above

ANSWER: D

Which of the following manages the execution of .NET programs?

A. CLS

B. CLR

C. CTS

D. All of the above

ANSWER: B

Which of the following allows cross-language communication and type safety?

A. CLS

B. CTS

C. Both of the above

D. None of the above

ANSWER: C

_____defines a set of rules and restrictions that every language must follow which runs under the .NET framework.

A. CLS

B. CLR

C. CTS

D. All of the above

ANSWER: A

_____describes the data types that can be used by managed code.

A. CLS

B. CLR

C. CTS

D. All of the above

ANSWER: C

_____is a file that is automatically generated by the compiler upon successful compilation of every .NET application.

A. Assembly

B. text file

C. JIT

D. none of the above

ANSWER: A

_____is any piece of software that makes itself available over the internet.

A. Base class library

B. Assembly

C. COM

D. Web services

ANSWER: C

In which of the following generation of garbage collection all the short-lived objects such as temporary variables

A. Generation 0

B. Generation 1

C. Generation 2

D. All of the above

ANSWER: A

_____ is a method to facilitate communication between different applications and languages.

A. web services

B. class library

C. CTS

D. COM

ANSWER: D

_____is the process of designing the application in such a way that it can be used by users from across the globe (multiple cultures).

A. Localization

B. web services

C. Globalization

D. COM

ANSWER: C

UNIT II

Introduction to C#:

- **Evaluation of C#**
- **Characteristics of C#, application of C#**
- **Difference between C++ and C#, Java and C#**
- **Introduction to C# environment: The .NET strategy, origins of the .NET**
- **Data types, identifiers, variables, constants, C# statements**
- **OOPs concept, array and strings**
- **operators, control statements, type conversions**
- **Mathematical functions.**

C# Tutorial

C# tutorial provides basic and advanced concepts of C#. Our C# tutorial is designed for beginners and professionals.

C# is a programming language of .Net Framework.

Our C# tutorial includes all topics of C# such as first example, control statements, objects and classes, inheritance, constructor, destructor, this, static, sealed, polymorphism, abstraction, abstract class, interface, namespace, encapsulation, properties, indexer, arrays, strings, regex, exception handling, multithreading, File IO, Collections etc.

What is C#

C# is pronounced as "C-Sharp". It is an object-oriented programming language provided by Microsoft that runs on .Net Framework.

By the help of C# programming language, we can develop different types of secured and robust applications:

- Window applications
- Web applications
- Distributed applications
- Web service applications
- Database applications etc.

C# is approved as a standard by ECMA and ISO. C# is designed for CLI (Common Language Infrastructure). CLI is a specification that describes executable code and runtime environment.

C# programming language is influenced by C++, Java, Eiffel, Modula-3, Pascal etc. languages.

C# is a simple, modern, general-purpose, object-oriented programming language developed by

Microsoft within its .NET initiative led by Anders Hejlsberg.

Features of C#

1. SIMPLE

1. Pointers are missing in C#.
2. Unsafe operations such as direct memory manipulation are not allowed.
3. In C# there is no usage of "::" or "->"operators.
4. Since it's on .NET, it inherits the features of automatic memory management and garbage collection.
5. Varying ranges of the primitive types like Integer, Floats etc.
6. Integer values of 0 and 1 are no longer accepted as Boolean values. Boolean values are pure true or false values in C# so no more errors of "="operator and"=="operator.
7. "==" is used for comparison operation and "=" is used for assignment operation.

2. MODERN

1. C# has been based according to the current trend and is very powerful and simple for building interoperable, scalable, robust applications.
2. C# includes built in support to turn any component into a web service that can be invoked over the internet from any application running on any platform.

3. OBJECTORIENTED

1. C# supports Data Encapsulation, inheritance, polymorphism, interfaces.
2. (int, float, double) are not objects in java but C# has introduces structures(structs) which enable the primitive types to become objects.

```
int i=1;
string a=i .Tostring(); //conversion (or) Boxing
```

4. TYPESAFE

1. In C# we cannot perform unsafe casts like convert double to a boolean.
2. Value types (primitive types) are initialized to zeros and reference types (objects and classes) are initialized to null by the compiler automatically.
3. Arrays are zero base indexed and are bound checked.
4. Overflow of types can be checked.

5. INTEROPERABILITY

1. C# includes native support for the COM and windows-based applications.
2. Allowing restricted use of native pointers.
3. Users no longer have to explicitly implement the unknown and other COM interfaces, those features are built-in.
4. C# allows the users to use pointers as unsafe code blocks to manipulate your old code.
5. Components from VB NET and other managed code languages and directly be used in C#.

6. SCALABLE ANDUPDATEABLE

1. .NET has introduced assemblies which are self-describing by means of their manifest. manifest establishes the assembly identity, version, culture and digital signature etc. Assemblies need not to be register anywhere.
2. To scale our application, we delete the old files and updating them with new ones. No registering of dynamic linking library.
3. Updating software components is an error prone task. Revisions made to the code can affect the existing program C# support versioning in the language. Native support for interfaces and method overriding enable complex frame works to be developed and evolved overtime.

Dot Net Framework

.NET is a software development platform developed by Microsoft. It runs on Microsoft Windows OS. .NET provides tools and libraries that allow developers to develop applications and services much easily, faster and secure by using a convenient way. The.Net Platform

provides a new environment for creating and running robust, scalable and distributed applications over the web. The .Net Framework provides an environment for building, deploying and running web services and other applications. It consists of three distinct technologies:

- Common Language Runtime (CLR)
- Framework Base Classes
- User and program interfaces (ASP.NET and WinForms)

The Common Language Runtime

CLR is the heart and soul of the .Net Framework. CLR is the runtime environment in which programs written in C# and other .Net languages are executed.

CLR provides various services:

- Loading and execution of programs
- Memory isolation for applications
- Verification of type safety
- Compilation of IL into native executable code
- Providing metadata
- Memory management (automatic garbage collection)
- Enforcement of security
- Interoperability with other systems
- Managing exceptions and errors
- Support for tasks such as debugging and profiling

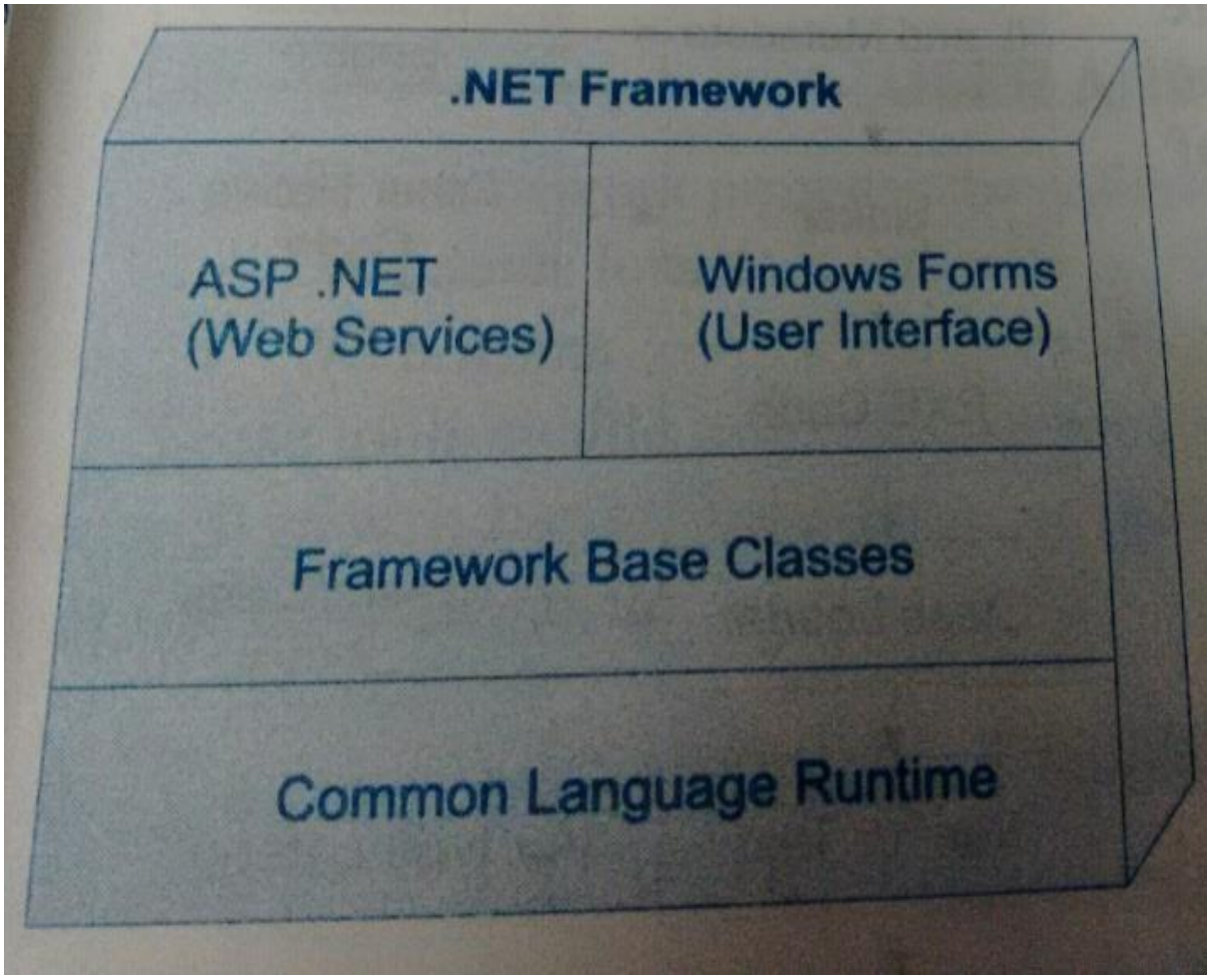
Framework Base classes

.Net supplies a library of classes that we can use to implement applications quickly. We can use the base classes in the system namespaces for different tasks:

- I/O operations
- String Handling
- Windows messages
- Database management
- Security etc.

User and Program Interfaces

The .Net Framework provides the following tools for managing user and application interfaces: Windows forms, console applications web forms web services etc. These tools enable users to develop user friendly desktop based as well as web-based applications using a wide variety of languages on the .Net platform.

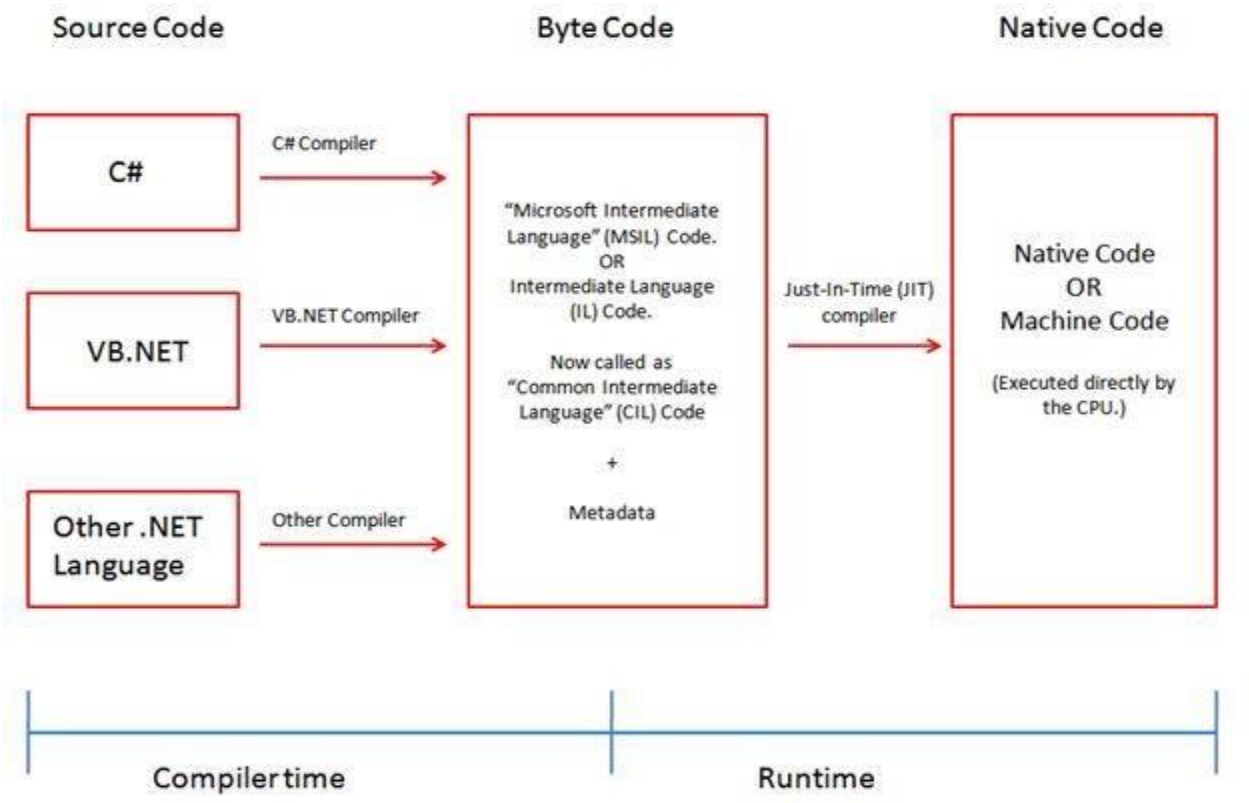


Compilation and Execution of C# program

Code Execution Process

The Code Execution Process involves the following two stages:

1. Compiler time process.
2. Run time process.



1. Compiler time process

1. The .Net framework has one or more language compliers, such as Visual Basic, C#, Visual C++, JScript, or one of many third-party compilers such as an Eiffel, Perl, or COBOL compiler.
2. Any one of the compilers translate your source code into Microsoft Intermediate Language (MSIL) code.
3. For example, if you are using the C# programming language to develop an application, when you compile the application, the C# language compiler will convert your source code into Microsoft Intermediate Language (MSIL) code.
4. In short, VB.NET, C# and other language compilers generate MSIL code. (In other words, compiling translates your source code into MSIL and generates the required metadata.)
5. Currently "Microsoft Intermediate Language" (MSIL) code is also known as "Intermediate Language" (IL) Code **or** "Common Intermediate Language" (CIL)Code.

   SOURCE CODE-----.NETCOMLIPER ------ > BYTE CODE (MSIL + METADATA)

2. Run time process.

1. The Common Language Runtime (CLR) includes a JIT compiler for converting MSIL to native code.
2. The JIT Compiler in CLR converts the MSIL code into native machine code that is then executed by the OS.
3. During the runtime of a program the "Just in Time" (JIT) compiler of the Common Language Runtime (CLR) uses the Metadata and converts Microsoft Intermediate Language (MSIL) into native code.

   BYTE CODE (MSIL + META DATA) ----- Just-In-Time(JIT)compiler ------ >NATIVE CODE

Namespaces

A **namespace** is designed for providing a way to keep one set of names separate from another. The class names declared in one namespace does not conflict with the same class names declared in another.

Defining a Namespace

A namespace definition begins with the keyword **namespace** followed by the namespace name as follows:

```
namespace namespace_name
{
// code declarations
}
```

Example

```
using System;
namespace first_space
{
  Class namespace_cl
  {
    public void func()
    {
```

```
        Console.WriteLine("Inside first_space");
      }
    }
}

namespace second_space
{
   class namespace_cl
   {
      public void func()
      {
        Console.WriteLine("Inside second_space");
      }
   }
}

class TestClass
{
   static void Main(string[] args)
   {
      first_space.namespace_cl fc = new first_space.namespace_cl();
      second_space.namespace_clsc = new second_space.namespace_cl();
      fc.func();
      sc.func();
      Console.ReadKey();
   }
}
```

Output Inside first_space
Inside second_space

The *using* Keyword

The **using** keyword states that the program is using the names in the given namespace.

Data types in C#

The variables in C#, are categorized into the following types:

- Value types
- Reference types
- Pointer types

Value Type

Value type variables can be assigned a value directly. They are derived from the class**System.ValueType**.

The value types directly contain data. Some examples are **int, char, and float**, which stores numbers, alphabets, and floating-point numbers, respectively. When you declare an **int** type, the system allocates memory to store the value.

Reference Type

The reference types do not contain the actual data stored in a variable, but they contain a reference to the variables.

In other words, they refer to a memory location. Using multiple variables, the reference types can refer to a memory location. If the data in the memory location is changed by one of the variables, the other variable automatically reflects this change in value. Example of **built- in** reference types are: **object**,**dynamic,** and **string**.

Pointer Type

Pointer type variables store the memory address of another type. Pointers in C# have the same capabilities as the pointers in C or C++.

Syntax for declaring a pointer type is:

```
type* identifier;
```

Type conversion in C#

Type conversion is converting one type of data to another type. It is also known as Type Casting. In C#, type casting has two forms:

- **Implicit type conversion** - These conversions are performed by C# in a type-safe manner. For example, are conversions from smaller to larger integral types and conversions from derived classes to base classes.

- **Explicit type conversion** - These conversions are done explicitly by users using the pre-defined functions. Explicit conversions require a cast operator.

The following example shows an explicit type conversion:

```
Using System;
namespace Type Conversion
Application
{
Class ExplicitConversion
{
staticvoidMain(string[] args)
{
double d =5673.74;
int i;

// cast double to int.
        i =(int)d;
Console.WriteLine(i);
Console.ReadKey();
}
}
```

Output5673

C# Type Conversion Methods

C# provides the following built-in type conversion methods,some of them are:

Sr.No	Methods & Description
1	**ToBoolean** Converts a type to a Boolean value, where possible.
2	**ToByte** Converts a type to a byte.
3	**ToChar** Converts a type to a single Unicode character, where possible.
4	**ToDateTime** Converts a type (integer or string type) to date-time structures.
5	**ToDecimal** Converts a floating point or integer type to a decimal type.
6	**ToDouble** Converts a type to a double type.

Boxing and Unboxing

Boxing is the process of converting a value type to the type **object** or to any interface type implemented by this value type. When the CLR boxes a value type, it wraps the value inside a System. Object and stores it on the managed heap. Unboxing extracts the value type from the object. Boxing is implicit; unboxing is explicit. The concept of boxing and unboxing underlies the C# unified view of the type system in which a value of any type can be treated as an object.
In the following example, the integer variable i is *boxed* and assigned to object o.
C#

```
int i = 123;
// The following line boxes i.
object o =i;
```

The object o can then be unboxed and assigned to integer variable i:
C#

```
o = 123;
i = (int)o; // unboxing
```

Var Variable

Var is an implicit type. It aliases any type. The aliased type is determined by the C# compiler. It makes programs shorter and easier to read. It was introduced in C# 3.0. var is used to declare implicitly typed local variable means it tells the compiler to figure out the type of the variable at compilation time. A var variable must be initialized at the time of declaration. Var variable can take any type of data type

Valid varstatements

1. **var str="1";**
2. **var num=0;**
3. **var b =3.4**

example

```
using system;

Class Progrm {

public static void Main()

    { var a = 3;

 Console.writeLine(a);

} }
```

Output 3

Arrays in C#

An array stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type stored at contiguous memory locations.

Declaring Arrays

To declare an array in C#, you can use the following syntax:

```
datatype[] arrayName;
```

example inaa[];

Initializing an Array

When the array variable is initialized, you can assign values to the array.

```
Example int[] a = new a[10];
```

Assigning Values to an Array

You can assign values to individual array elements, you can assign values to the array at the time of declaration, as shown:

```
double[] balance = { 2340.0, 4523.69, 3421.0};
```

You can also create and initialize an array, as shown:

```
int [] marks = new int[5] { 99, 98, 92, 97, 95};
```

You may also omit the size of the array, as shown:

```
int [] marks = new int[] { 99, 98, 92, 97, 95};
```

Accessing Array Elements

Example

```
using System;
namespace ArrayApplication
{
  class MyArray
  {
    static void Main(string[] args)
    {
      int [] n = new int[10]; /* n is an array of 10 integers */
      int i,j;

      /* initialize elements of array n */
      for ( i = 0; i< 10; i++ )
      {
        n[i ] = i + 100;
      }

      /* output each array element's value */
      for (j = 0; j < 10; j++ )
      {
        Console.WriteLine("Element[{0}] = {1}", j, n[j]);
      }
      Console.ReadKey();
    }
  }
}
```

Multidimensional Arrays

Multi-dimensional arrays are also called rectangular array. You can declare a 2-dimensional array of strings as:

```
string [,] names;
```

or, a 3-dimensional array of int variables as:

```
int [ , , ] m;
```

Two-Dimensional Arrays

The simplest form of the multidimensional array is the 2-dimensional array. A 2-dimensional array is a list of one-dimensional arrays.

A 2-dimensional array can be thought of as a table, which has x number of rows and y number of columns. Following is a 2-dimensional array, which contains 3 rows and 4 columns:

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Thus, every element in the array a is identified by an element name of the form a[i , j], where a is the name of the array, and i and j are the subscripts that uniquely identify each element in array a.

Initializing Two-Dimensional Arrays

Multidimensional arrays may be initialized by specifying bracketed values for each row. The Following array is with 3 rows and each row has 4 columns.

```
int[,] a =newint[3,4]{
{0,1,2,3},/*  initializers for row indexed by 0*/
{4,5,6,7},/*  initializers for row indexed by 1*/
{8,9,10,11}/* initializers for row indexed by 2 */
};
```

Accessing Two-Dimensional Array Elements

An element in 2-dimensional array is accessed by using the subscripts. That is, row index and column index of the array. For example,

```
int val = a[2,3];
```

example

```
using System;
namespace ArrayApplication
{
  class MyArray
  {
    static void Main(string[] args)
    {
```

```
        /* an array with 5 rows and 2 columns*/
        int[,] a = new int[5, 2] {{0,0}, {1,2}, {2,4}, {3,6}, {4,8} };
        int i, j;

        /* output each array element's value */
        for (i = 0; i< 5; i++)
        {
          for (j = 0; j < 2; j++)
          {
            Console.WriteLine("a[{0},{1}] = {2}", i, j, a[i,j]);
          }
        }
        Console.ReadKey();
      }
   }
}
```

Jagged Arrays

A Jagged array is an array of arrays. You can declare a jagged array named*scores*of type **int** as:

```
int [][] scores;
```

Declaring an array, does not create the array in memory. To create the above array:

```
int[][] scores = new int[5][];
for (int i = 0; i<scores.Length; i++)
{
  scores[i] = new int[4];
}
```

You can initialize a jagged array as:

```
int[][] scores = new int[2][]{new int[]{92,93,94},new int[]{85,66,87,88}};
```

Where, scores is an array of two arrays of integers - scores[0] is an array of 3 integers and scores[1] is an array of 4 integers.

```
using System;
namespace ArrayApplication
{
  class MyArray
  {
    static void Main(string[] args)
    {
      /* a jagged array of 5 array of integers*/
      int[][] a = new int[][]{new int[]{0,0},new int[]{1,2},new int[]{2,4},new int[]{ 3, 6 }, new int[]{ 4, 8 } };
      int i, j;

      /* output each array element's value */
      for (i = 0; i< 5; i++)
      {
        for (j = 0; j < 2; j++)
        {
          Console.WriteLine("a[{0}][{1}] = {2}", i, j, a[i][j]);
        }
      }
      Console.ReadKey();
    }
  }
}
```

Example

foreach loop

foreach loop is similar to for loop. It is used to iterate through the items in a collection.eg: foreach loop can be used with arrays or collections such as array list, hash tables etc.

syntax foreach(type variable inexpression)

```
{   body of theloop
}
```

The type and variable declare the iteration variable. During execution, the iteration variable represents the array element or collection for which an iteration is currently being performed. 'in' is a keyword. The expression must be an array or collection type.

Example

```
Using System;
Class Program
{ public static void Main()
    {   int[] numbers =new int[3];
        numbers[0]=10;
        numbers[1]=20;
        numbers [2]=30;
foreach( int k innumbers)
   {console.writeLine(k);
    } } }
```

Strings in C#

C# supports a predefined reference data type known as string. We can use string to declare string type object. The System.String class also defines the properties and methods to work with string datatypes.

Creating a String Object

You can create string object using one of the following methods:

- By assigning a string literal to a String variable
- By using a String class constructor
- By using the string concatenation operator(+)
- By retrieving a property or calling a method that returns a string
- By calling a formatting method to convert a value or an object to its string

representation Example

```
using System;
namespace CSharpStrings
{
   class Program
   {
      static void Main(string[] args)
      {
         string firstName = "ram";
         string lastName = "sham";
         string age = "33";
         string numberString = "33.23";
         Console.WriteLine("First Name: {0}", firstName);
         Console.WriteLine("Last Name: {0}", lastName);
         Console.WriteLine("Age: {0}", age);
         Console.WriteLine("Number: {0}", numberString);
```

```
            Console.ReadKey();
        }
    }
```

Methods

Method	Description
Clone()	Returns a reference to this instance of String.
Compare()	Used to compare the two string objects.
CompareOrdinal(String, Int32, String, Int32, Int32)	Compares substrings of two specified String objects by evaluating the numeric values of the corresponding Char objects in each substring.
CompareOrdinal(String, String)	Compares two specified String objects by evaluating the numeric values of the corresponding Char objects in each string.
CompareTo()	Compare the current instance with a specified Object or String object.
Concat()	Concatenates one or more instances of String, or the String representations of the values of one or more instances of Object.
Contains(String)	Returns a value indicating whether a specified substring occurs within this string.
Copy(String)	Creates a new instance of String with the same value as a specified String.
CopyTo(Int32, Char[], Int32, Int32)	Copies a specified number of characters from a specified position in this instance to a specified position in an array of Unicode characters.
EndsWith()	Determines whether the end of this string instance matches a specified string.
Equals()	Determines whether two String objects have the

	same value.
Format()	Converts the value of objects to strings based on the formats specified and inserts them into another string.
GetEnumerator()	Retrieves an object that can iterate through the individual characters in this string.
GetHashCode()	Returns the hash code for this string.
GetType()	Gets the Type of the current instance. (Inherited from Object)
GetTypeCode()	Returns the TypeCode for class String.
IndexOf()	Reports the zero-based index of the first occurrence of a specified Unicode character or string within this instance. The method returns -1 if the character or string is not found in this instance.
IndexOfAny()	Reports the index of the first occurrence in this instance of any character in a specified array of Unicode characters. The method returns -1 if the characters in the array are not found in this instance.
Insert(Int32, String)	Returns a new string in which a specified string is inserted at a specified index position in this instance.
Intern(String)	Retrieves the system's reference to the specified String.
IsInterned(String)	Retrieves a reference to a specified String.
IsNormalized()	Indicates whether this string is in a particular Unicode normalization form.
IsNullOrEmpty(String)	Indicates whether the specified string is null or an Empty string.
IsNullOrWhiteSpace(String)	Indicates whether a specified string is null, empty, or consists only of white-space characters.
Join()	Concatenates the elements of a specified array or the members of a collection, using the specified separator between each element or member.

LastIndexOf()	Reports the zero-based index position of the last occurrence of a specified Unicode character or string within this instance. The method returns -1 if the character or string is not found in this instance.
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
Normalize()	Returns a new string whose binary representation is in a particular Unicode normalization form.
PadLeft()	Returns a new string of a specified length in which the beginning of the current string is padded with spaces or with a specified Unicode character.
PadRight()	Returns a new string of a specified length in which the end of the current string is padded with spaces or with a specified Unicode character.
Remove()	Returns a new string in which a specified number of characters from the current string are deleted.
Replace()	Returns a new string in which all occurrences of a specified Unicode character or String in the current string are replaced with another specified Unicode character or String.
Split()	Returns a string array that contains the substrings in this instance that are delimited by elements of a specified string or Unicode character array.
StartsWith(String)	Determines whether the beginning of this string instance matches a specified string.
Substring(Int32)	Retrieves a substring from this instance.
ToCharArray()	Copies the characters in this instance to a Unicode character array.
ToLower()	Returns a copy of this string converted to lowercase.
ToLowerInvariant()	Returns a copy of this String object converted to lowercase using the casing rules of the invariant culture.
ToString()	Converts the value of this instance to a String.

ToUpper()	Returns a copy of this string converted to uppercase.
ToUpperInvariant()	Returns a copy of this String object converted to uppercase using the casing rules of the invariant culture.
Trim()	Returns a new string in which all leading and trailing occurrences of a set of specified characters from the current String object are removed.
TrimEnd(Char[])	Removes all trailing occurrences of a set of characters specified in an array from the current String object.
TrimStart(Char[])	Removes all leading occurrences of a set of characters specified in an array from the current String object.

MCQ

_____ is the process of customization to make our application behave as per the current culture and locale. These two things go together.

A. Localization

B. web services

C. Globalization

D. COM

ANSWER: A

which of the following is true about assemblies in C#

A. It can be either a Dynamic Link Library or an executable file.

B. It is generated only once for an application and upon each subsequent compilation the assembly gets updated.

C. Dot nET supports three kinds of assemblies

D. All of the above

ANSWER: D

CLS stands for

A. Common language Subsystem

B. Common language Specification

C. Component language subsytem

D. None of the above

ANSWER: B

Which of the following statements are TRUE about the .NET CLR?

A. It provides a language-neutral development & execution environment.

B. It ensures that an application would not be able to access memory that it is not authorized to access.

C. It provides services to run "managed" applications.

D. All of the above

ANSWER: D

Which of the following statements is correct about Managed Code?

A. Managed code is the code that is compiled by the JIT compilers.

B. Managed code is the code where resources are Garbage Collected.

C. Managed code is the code that runs on top of Windows.

D. Managed code is the code that is written to target the services of the CLR.

ANSWER: D

Which of the following components of the .NET framework provide an extensible set of classes that can be used by any .NET compliant programming language?

A. Dot net class libraries

B. Common Language Runtime

C. Common Language Infrastructure

D. Component Object Model

ANSWER: A

Which of the following jobs are NOT performed by Garbage Collector?

A. Freeing memory on the stack.

B. Closing unclosed database collections.

C. Closing unclosed files.

D. All of the above

ANSWER: D

Which of the following .NET components can be used to remove unused references from the managed heap?

A. Common Language Infrastructure

B. CLR

C. Garbage Collector

D. Class Loader

ANSWER: C

Which of the following constitutes the .NET Framework?

A. CLR

B. Framework Class Library

C. Windows Services

D. Both A and B

ANSWER: D

Which of the following benefits do we get on running managed code under CLR?

A. Type safety of the code running under CLR is assured.

B. It is ensured that an application would not access the memory that it is not authorized to access.

C. It launches separate process for every application running under it.

D. All of the above

UNIT III

- **Constructors, Overloaded Constructors, static constructors, Copy constructors, Destructors**
- **'this' reference, Constant Members**
- **Properties, Auto Implemented Properties**
- **Object Initializer, Collection Initializer, Anonymous Types, Extension Methods**
- **Indexers**
- **Inheritance, interface and polymorphism**

Member access modifiers

All types and type members have an accessibility level, which controls whether they can be used from other code in your assembly or other assemblies. You can use the following access modifiers to specify the accessibility of a type or member when you declare it:

public

The type or member can be accessed by any other code in the same assembly or another assembly that references it.

private

The type or member can be accessed only by code in the same class or struct.

protected

The type or member can be accessed only by code in the same class or struct, or in a class that is derived from that class.

internal

The type or member can be accessed by any code in the same assembly, but not from another assembly.

protected internal

The type or member can be accessed by any code in the assembly in which it is declared, or from within a derived class in another assembly. Access from another assembly must take place within a class declaration that derives from the class in which the protected internal element is declared, and it must take place through an instance of the derived class type.

Constructors

A class **constructor** is a special member function of a class that is executed whenever we create new objects of that class.

A constructor has exactly the same name as that of class and it does not have any return type.

Example

```
using System;
namespace LineApplication
{
  class Line
  {
    private double length; //
    Length of a line public
    Line()
    {
      Console.WriteLine("Object is being created");
    }

    public void setLength( doublelen )
    {
      length = len;
    }

    public double getLength()
    {
      return length;
```

```
        // set line length
        line.setLength(6.0);
        Console.WriteLine("Length of line : {0}", line.getLength());
        Console.ReadKey();
     }
  }
}
```

Output Object is being created
Length of line : 6

A **default constructor** does not have any parameter but if you need, a constructor can have parameters. Such constructors are called **parameterized constructors**.

Overloaded Constructors in C#

Constructor Overloading is a technique to create multiple constructors with different set of parameters and different number of parameters. It allows us touse aclass in a different manner. A same class may behave different type based on constructors overloading. With one object initialization it may show simple string message whereas in the second initialization of object with different parameter it may done arithmetic calculation.

Example

```
using System;
using System.Collections.Generic;
usingSystem.Linq;
usingSystem.Text;

namespace Constructor_Overloading
{
    class GameScore
    {
        string user;
        int age;
        //Default Constructor
        public GameScore()
        {
```

```
            user = "Steven";
            age = 28;
            Console.WriteLine("Previous User {0} and he was {1} year old", user,
age);
        }

        //Parameterized Constructor
        public GameScore(string name, int age1)
        {
            user = name;
            age = age1;
            Console.WriteLine("Current User {0} and he is {1} year old", user, age);
        }
    }

    class Program
    {
        static void Main(string[]args)
        {
            GameScoregs = new GameScore(); //Default Constructor Called
            GameScore gs1 = new GameScore("Clark", 35);//Overloaded Constructor.
            Console.ReadLine();
        }
    }
}
```

Static Constructors in C#

A static constructor is used to initialize any static data, or to perform a particular action that needs to be performed once only. It is called automatically before the first instance is created or any static members are referenced.
C#

```
class SimpleClass
{
   // Static variable that must be initialized at run time.
   static readonly long baseline;

   // Static constructor is called at most one time, before any
   // instance constructor is invoked or member is accessed.
   static SimpleClass()
   {
      baseline = DateTime.Now.Ticks;
   }
```

}
Static constructors have the following properties:

- A static constructor does not take access modifiers or have parameters.
- A static constructor is called automatically to initialize the class before the first instance is created or any static members are referenced.
- A static constructor cannot be called directly.
- The user has no control on when the static constructor is executed in the program.
- A typical use of static constructors is when the class is using a log file and the constructor is used to write entries to this file.
- Static constructors are also useful when creating wrapper classes for unmanaged code, when the constructor can call the **Load Library** method.
- If a static constructor throws an exception, the runtime will not invoke it a second time, and the type will remain uninitialized for the lifetime of the application domain in which your program is running.

Copy Constructor

A parameterized constructor that contains a parameter of same class type is called as copy constructor. Main purpose of copy constructor is to initialize new instance to the values of an existing instance.

Example

```
using System;
namespace ConsoleApplication3
{
class Sample
{
public string param1, param2;
public Sample(string x, string y)
{
param1 = x;
param2 = y;
}
publicSample(Sampleobj)       // CopyConstructor
{
param1 =
obj.param1; param2
= obj.param2;
}
}
class Program
{
```

```
{
Sample obj = new Sample("Welcome", "Aspdotnet-Suresh"); // Create instance to class Sample
Sample obj1=new Sample(obj); // Here obj details will copied to obj1
Console.WriteLine(obj1.param1 +" to " + obj1.param2);
Console.ReadLine();
}
}
}
```

Destructors

'Destructors' are used to destruct instances of classes. When we are using destructors in C#, we have to keep in mind the following things:

- A class can only have one destructor.
- Destructors cannot be inherited or over loaded.
- Destructors cannot be called. They are invoked automatically.
- A destructor does not take modifiers or have parameters.

Example

```
using System;
class A
{
public A()
 {
  Console.WriteLine("Creating A");
 }
 ~A()
 {
  Console.WriteLine("Destroying A");
 }
}

class B:A
{
public B()
 {
  Console.WriteLine("Creating B");
 }
 ~B()
 {
  Console.WriteLine("Destroying B");
 }
```

```
}
class C:B
{
public C()
 {
  Console.WriteLine("Creating C");
 }

 ~C()
 {
  Console.WriteLine("Destroying C");
 }
}
class App
{
publicstaticvoidMain()
 {
  C c=new C();
  Console.WriteLine("Object Created ");
  Console.WriteLine("Press enter to Destroy it");
  Console.ReadLine();
  c=null;
//GC.Collect();
  Console.Read();
 }

}
```

Properties in C#

Properties are named members of classes, structures, and interfaces. Member variables or methods in a class or structures are called **Fields**. Properties are an extension of fields and are accessed using the same syntax. They use**accessors**through which the values of the private fields can be read, written or manipulated.

Accessors

The **accessor** of a property contains the executable statements that helps in getting (reading or computing) or setting (writing) the property. The accessor declarations can contain a get accessor, a set accessor, or both. For example:

```
// Declare a Code property of type string:
publicstringCode
```

```
{
get
{
return code;
}
set
{
    code = value;
}
}

// Declare a Name property of type string:
publicstringName
{
get
{
return name;
}
set
{
    name = value;
}
}

// Declare a Age property of type int:
publicintAge
{
get
{
return age;
```

```
}
set
{
    age = value;
}
}
```

Auto implemented properties in C#

Auto-implemented properties make property-declaration more concise when no additional logic is required in the property accessors. They also enable client code to create objects. When you declare a property as shown in the following example, the compiler creates a private, anonymous backing field that can only be accessed through the property's get and set accessors.

Example

```
class Customer
{
   // Auto-Impl Properties for trivial get and set
   public double TotalPurchases{ get; set; }
   public string Name { get; set; }
   public int CustomerID{ get; set; }

   // Constructor
   public Customer(double purchases, string name, int ID)
   {
      TotalPurchases = purchases;
      Name = name;
      CustomerID = ID;
   }
   // Methods
   public string GetContactInfo() {return "ContactInfo";}
   public string GetTransactionHistory() {return "History";}

   // .. Additional methods, events, etc.
}

class Program
{
   static void Main()
   {
      // Intialize a new object.
      Customer cust1 = new Customer ( 4987.63, "Northwind",90108 );
```

```
        //Modify a property
        cust1.TotalPurchases += 499.99;
    }
}
```

Object Initializer

Object initializers let you assign values to any accessible fields or properties of an object at creation time without having to invoke a constructor followed by lines of assignment statements. The object initializer syntax enables you to specify arguments for a constructor or omit the arguments (and parentheses syntax). Eg: Cat cat = new Cat { Age = 10, Name = "Fluffy" };

Collection Initializer

Collection initializers let you specify one or more element initializers when you initialize a collection class that implements IEnumerableor a class with an **Add** extension method. The element initializers can be a simple value, an expression or an object initializer. By using a collection initializer you do not have to specify multiple calls to the **Add** method of the class in your source code; the compiler adds the calls.

The following examples shows two simple collection initializers:

List<int> digits = new List<int>{ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };

List<int> digits2 = new List<int> { 0 + 1, 12 % 3, MakeInt() };

Inheritance

Base and Derived Classes

A class can be derived from more than one class or interface, which means that it can inherit data and functions from multiple base classes or interfaces.

The syntax used in C# for creating derived classes is as follows:

```
<acess-specifier> class <base_class>
{
  ...
}
class <derived_class> :<base_class>
{
  ...
}
```

```
using System;
namespace InheritanceApplication
{
  class Shape
  {
    public void setWidth(int w)
    {
      width = w;
    }
```

```
   public void setHeight(int h)
   {
      height = h;
   }
   protected int width;
   protected int height;
}

// Derived class
class Rectangle: Shape
{
   public int getArea()
   {
      return (width * height);
   }
}

class RectangleTester
{
   static void Main(string[] args)
   {
      Rectangle Rect = new Rectangle();

      Rect.setWidth(5);
      Rect.setHeight(7);

      // Print the area of the object.
      Console.WriteLine("Total area: {0}", Rect.getArea());
      Console.ReadKey();
   }
```

```
  }
}
```

Subclass Constructor

This constructor is used to construct the instance variables of both the subclass and the super class. The subclass constructor uses the keyword ‘base’ to invoke the constructor method of the super class.

Example

Hiding methods

if a method is not overriding the derived method, it is hiding it. A hiding method has to be declared using the **new** keyword. The correct class definition in the second listing is thus:

```
using System;
namespace Polymorphism
{
   class A
   {
      public void Foo() { Console.WriteLine("A::Foo()"); }
   }

   class B : A
   {
      public new void Foo() { Console.WriteLine("B::Foo()"); }
   }

   class Test
   {
      static void Main(string[] args)
      {
         Aa;
         Bb;

         a = newA();
         b = newB();
         a.Foo();  // output -->"A::Foo()"
         b.Foo();  // output -->"B::Foo()"
```

```
            a = new B();
            a.Foo(); // output --> "A::Foo()"
        }
    }
}
```

Abstract class If a class is defined as abstract then we can't create an instance of that class. By the creation of the derived class object where an abstract class is inherit from, we can call the method of the abstract class.

Abstract method

An Abstract method is a method without a body. The implementation of an abstract method is done by a derived class. When the derived class inherits the abstract method from the abstract class, it must override the abstract method. This requirement is enforced at compile time and is also called dynamic polymorphism.

The syntax of using the abstract method is as follows:

<access-modifier>abstract<return-type>method name (parameter)

The abstract method is declared by adding the abstract modifier the method.

Sealed classes

Sealed class is used to define the inheritance level of a class.

The sealed modifier is used to prevent derivation from a class. An error occurs if a sealed class is specified as the base class of another class.

Some points to remember:

1. A class, which restricts inheritance for security reason is declared, sealed class.
2. Sealed class is the last class in the hierarchy.
3. Sealed class can be a derived class but can't be a base class.
4. A sealed class cannot also be an abstract class. Because abstract class has to provide functionality and here, we are restricting it to inherit.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace sealed_class
{
   class Program
   {
      public sealed class BaseClass
      {
         public void Display()
      {
         Console.WriteLine("This is a sealed class which can;t be further inherited");
      }
   }

      public class Derived :BaseClass
      {
         // this Derived class can;t inherit BaseClass because it is sealed
      }

      static void Main(string[] args)
      {
         BaseClass obj = new BaseClass();

         obj.Display();

         Console.ReadLine();
      }
   }
}
```

Sealed methods -Sealed method is used to define the overriding level of a virtual method.Sealed keyword is always used with override keyword.

Operator Overloading

Overloaded operators are functions with special names the keyword **operator** followed by the symbol for the operator being defined. similar to any other function, an overloaded operator has a return type and a parameter list.

Example of unary operator overloading

```
01 using System;
02 class bank
03 {
04     int x;
05     int y;
06    public bank(int a, int b)
07    {
08        x = a;
09        y = b;
10    }
11    public bank()
12    {
13    }
14    public void display()
15    {
16        Console.Write(" " + x);
17        Console.Write(" " + y);
18        Console.WriteLine();
19    }
20    public static bank operator -(bank b)
21    {
22        b.x = -b.x;
23        b.y = -b.y;
24        return b;
25    }
26 }
27 class program
28 {
29    public static void Main()
30    {
```

```
31      bank ba1 = new bank(10,-20);
32      ba1.display();
33      bank ba2 = new bank();
34      ba2.display();
35      ba2.display();
        Console.ReadLine();
        }
        }
```

Example of binary operator overloading

```
01 using System;
02 namespace binary_overload
03 {
04    class complexNumber
05    {
06        int x;
07        double y;
08        public complexNumber(int real, double imagnary)
09        {
10            x = real;
11            y = imagnary;
12        }
13        public complexNumber()
14        {
15        }
16  public static complexNumber operator + (complexNumber c1, complexNumber c2)
17        {
18            complexNumber c = new complexNumber();
19            c.x=c1.x+c2.x;
20            c.y=c1.x-c2.y;
21            return c;
22        }
23        public void show()
24        {
25            Console.Write(x);
26            Console.Write("+j"+y);
27            Console.WriteLine();
28        }
29    }
```

```
30  class Program
31  {
32      static void Main(string[] args)
33      {
34          complexNumber p, q, r;
35          p = new complexNumber(10, 2.0);
36          q = new complexNumber(20, 15.5);
37          r = p + q;
38          Console.Write("p=");
39          p.show();
40          Console.Write("q=");
41          q.show();
42          Console.Write("r=");
43          r.show();
44          Console.ReadLine();
45      }
46  }
47 }
```

Operators that can be overloaded in c# as shown below:

Category	Operators
Binary arithmetic	+,-,*,/'%
Logical operator	==,!=,>=,<=,>,<
Unary arithmetic	+,-++,--
Binary Bitwise	|,&,^,>>,<<
Unary Bitwise	~,!,true,false

Operators that can be overloaded in C# as shown below:

Category	Operators
Conditional operators	||,&&
Compound Assignment	-=,+=,*=,%=
Other	(),[],=,?:,->,new,typeof,sizeof,as,is

Indexers

An **indexer** allows an object to be indexed such as an array. When you define an indexer for a class, this class behaves similar to a **virtual array**. You can then access the instance of this class using the array access operator ([]).

Syntax

A one dimensional indexer has the following syntax:

```
element-type this[int index]
{
// The get accessor.
get
{
// return the value specified by index
}

// The set accessor.
set
{
// set the value specified by index
}
}
```

Use of Indexers

Declaration of behavior of an indexer is to some extent similar to a property. similar to the properties, you use **get** and **set** accessors for defining an indexer. However, properties return or set a specific data member, whereas indexers return or sets a particular value from the object instance. In other words, it breaks the instance data into smaller parts and indexes each part, gets or sets each part. Defining a property involves providing a property name. Indexers are not defined with names, but with this keyword, which refers to the object instance.

MCQ

1. What will be the output of the following C# code?

```
 class sample
 {
     public  int i;
     void display()
     {
         Console.WriteLine(i);
     }
}
class sample1 : sample
{
    public  int j;
    public void display()
    {
        Console.WriteLine(j);
    }
}
class Program
{
    static void Main(string[] args)
    {
        sample1 obj = new sample1();
        obj.i = 1;
        obj.j = 2;
        obj.display();
        Console.ReadLine();
    }
}
```

a) 1
b) 3
c) 2
d) Compile Time error

Answer: c
Explanation: class sample & class sample1 both contain display() method, class sample1 inherits class sample, when display() method is called by object of class sample 1, display() method of class sample 1 is executed rather than that of Class sample.
advertisement

2. What will be the Correct statement in the following C# code?

```
class sample
{
   protected int index;
   public sample()
   {
      index = 0;
   }
}
class sample 1: sample
{
   public void add()
   {
      index += 1;
   }
}
class Program
{
   static void Main(string[] args)
   {
      sample 1 z = new sample 1();
      z . add();
   }
}
```

a) Index should be declared as protected if it is to become available in inheritance chain
b) Constructor of sample class does not get inherited in sample 1 class
c) During constructing an object referred to by z, Firstly constructor of sample class will be called followed by constructor of sample 1 class
d) All of the mentioned

Answer: d

Explanation: None.

3. The following C# code is run on single level of inheritance. What will be the Correct statement in the following C# code?

```
class sample
{
    int i = 10;
    int j = 20;
    public void display()
    {
        Console.WriteLine("base method ");
    }
}
class sample1 : sample
{
    public  int s = 30;
}
class Program
{
    static void Main(string[] args)
    {
        sample1 obj = new sample1();
        Console.WriteLine("{0}, {1}, {2}", obj.i,  obj.j,  obj.s);
        obj.display();
        Console.ReadLine();
    }
}
```

a)

10, 20, 30
base method

b) 10, 20, 0

c) compile time error

d) base method

Answer: c

Explanation: ‘i’ and ‘ j’ are inaccessible due to protection level. Declare them as public variable and hence will be accessed in code.

4. What will be size of the object created depicted by C# code snippet?

```
class baseclass
{
    private int a;
    protected int b;
    public int c;
}
class derived : baseclass
{
    private int x;
    protected int y;
    public int z;
}
class Program
{
    static Void Main(string[] args)
    {
        derived a = new derived();
    }
}
```

a) 20 bytes
b) 12 bytes
c) 16 bytes
d) 24 bytes

Answer: d
Explanation: Explained in fundamentals of inheritance.
5. What will be the output of the following C# code?

```
class sample
{
   public sample()
   {
      Console.WriteLine("THIS IS BASE CLASS constructor");
   }
}
public class sample1 : sample
{
```

```
}
class Program
{
   static void Main(string[] args)
   {
      sample1 obj = new sample1();
      Console.ReadLine();
   }
}
```

a) Code executes successfully prints nothing
b) This is base class constructor
c) Compile time error
d) None of the mentioned

Answer: c
Explanation: Base class accessibility level is much less compared to derived class. Declare it public to get desired output.

6. Select the statement which should be added to the current C# code to get the output as 10 20?

```
class baseclass
{
   protected int a = 20;
}
class derived : baseclass
{
   int a = 10;
   public void math()
   {
      /* add code here */
   }
}
```

a) Console.writeline(a + ” ” + this.a);
b) Console.Writeline(mybase.a + ” ” + a);
c) console.writeline(a + ” ” + base.a);
d) console.writeline(base.a + ” ” + a);

Answer: c
Explanation: None.

7. What will be the Correct statement in the following C# code?

```
class baseclass
{
    int a;
    public baseclass(int a1)
    {
        a = a1;
        console.writeline(" a ");
    }
    class derivedclass : baseclass
    {
        public derivedclass (int a1) : base(a1)
        {
            console.writeline(" b ");
        }
    }
    class program
    {
        static void main(string[] args)
        {
            derivedclass d =  new derivedclass(20);
        }
    }
}
```

a) Compile time error

b)

```
b
a
```

c)

```
a
b
```

d) The program will work correctly if we replace base(a1) with base.baseclass(a1)

Answer: c

Explanation: None.

Output :

a
b

8. Which C# statement should be added in function a() of class y to get output “i love csharp”?

```
class x
{
   public void a()
   {
      console.write("bye");
   }
}
class y : x
{
   public void a()
   {
   /* add statement here */
      console.writeline("  i love csharp ");
   }
}
class program
{
   static void main(string[] args)
   {
      y obj =  new obj();
      obj.a();
   }
}
```

a) x.a();
b) a();
c) base.a();
d) x::a();

Answer: c
Explanation: None.

9. Which statements are correct?

a) If a base class consists of a member function fun() and a derived class do not have any function with this name. An object of derived class can access fun()
b) A class D can be derived from class C, which is derived from class B which in turn is derived from class A
c) If a base class and a derived class each include a member function with same name, the member function of the

derived class will be called by object of derived class
d) All of the mentioned

Answer: d
Explanation: None.
10. What will be the output of the following C# code?

```
class A
{
   public int i;
   protected int j;
}
class B : A
{
   public int j;
   public void display()
   {
      base.j = 3;
      Console.WriteLine(i + " " + j);
   }
}
class Program
{
   static void Main(string[] args)
   {
      B obj = new B();
      obj.i = 1;
      obj.j = 2;
      obj.display();
      Console.ReadLine();
   }
}
```

a) 2 1
b) 1 0
c) 0 2
d) 1 2

Answer: d

Explanation: Both class A & B have members with same name that is j, member of class B will be called by default if no specifier is used. i contains 1 & j contains 2, printing 1 2.

Output:

1, 2

11. What will be the output of the following C# code?

```
class A
{
    public int i;
    private int j;
}
class B :A
{
    void display()
    {
        base.j = base.i + 1;
        Console.WriteLine(base.i + "  " + base.j);
    }
}
class Program
{
   static void Main(string[] args)
   {
       B obj = new B();
       obj.i = 1;
       obj.j = 2;
       obj.display();
       Console.ReadLine();
   }
}
```

a) 1, 3

b) 2, 3

c) 1, 2

d) compile time error

Answer: d

Explanation: Class contains a private member variable j, this cannot be inherited by subclass B and does not have access to it.

12. Which of these keywords is used to refer to member of base class from a sub class?

a) upper
b) base
c) this
d) none of the mentioned

Answer: b
Explanation: Whenever a subclass needs to refer to its immediate super class, it can do so by use of the keyword base.
13. Which of these operators must be used to inherit a class?
a) :
b) &
c) ::
d) extends

Answer: a
Explanation:

```
class a
{

}
class b : a
{

}
```

14. What will be the output of the following C# code?

```
using System;
public class BaseClass
{
   public BaseClass()
   {
      Console.WriteLine("I am a base class");
   }
}
public class ChildClass : BaseClass
{
   public ChildClass()
   {
      Console.WriteLine ("I am a child class");
```

```
    }
 static void Main()
 {
    ChildClass CC = new ChildClass();
 }
 }
```

a)

I am a base class
I am a child class

b)

I am a child class
I am a base class

c) Compile time error

d) None of the mentioned

Answer: a

Explanation: This is because base classes are automatically instantiated before derived classes. Notice the output, The BaseClass constructor is executed before the ChildClass constructor.

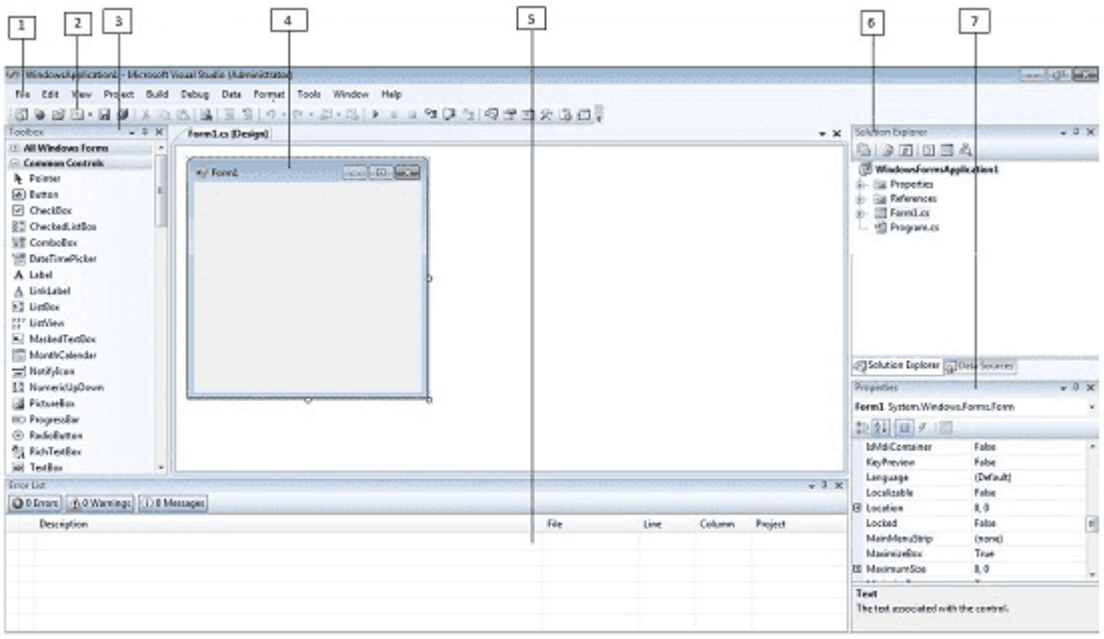
Output: I am a base class
I am a child class

UNIT IV

Visual studio IDE features, introduction to Window forms, components, control:

- **Textbox, label, link label, status bar**
- **Checked list box, combo box, list box, list view**
- **Radio button, button, panel, group box, dialog box**
- **menu control, properties, methods, events of control**
-

C# is designed for building a variety of applications that run on the .NET Framework. Before you start learning more about C# programming, it is important to understand the development environment and identify some of the frequently using programming tools in the Visual Studio IDE.



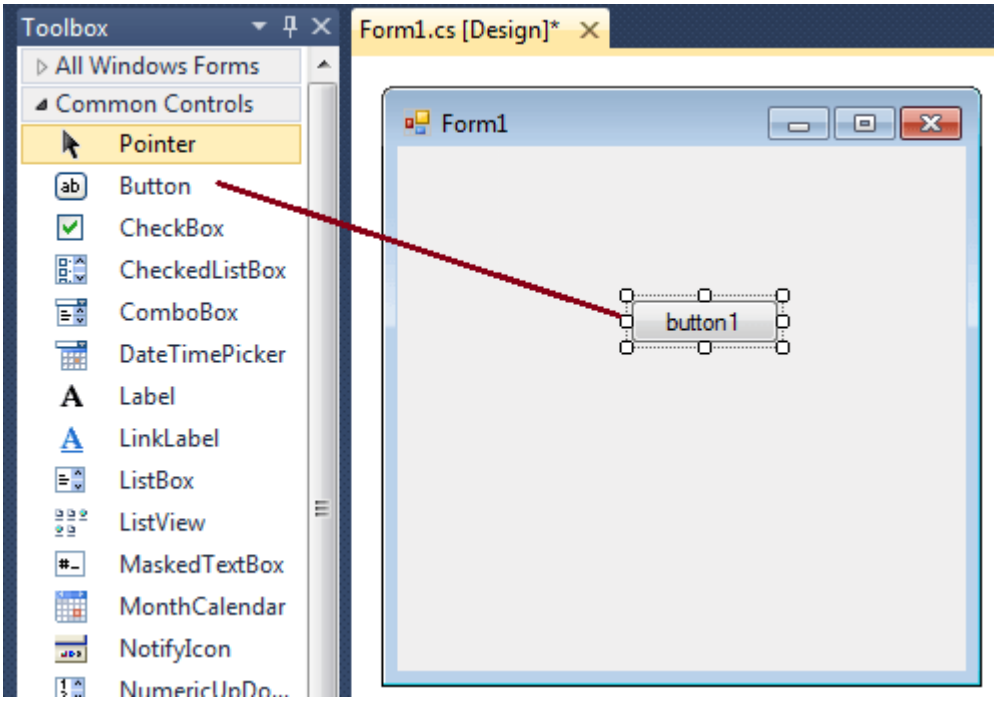
- 1. Menu Bar
- 2. Standard Toolbar
- 3. ToolBox
- 4. Forms Designer
- 5. Output Window
- 6. Solution Explorer
- 7. Properties Window

Combined with the .NET Framework, C# enables the creation of Windows applications, Web services, database tools, components, controls, and more. Visual Studio organizes your work in projects and

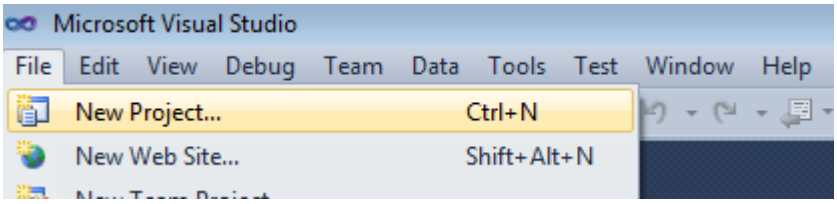
solutions. A solution can contain more than one project, such as a DLL and an executable that references that DLL. From the following C# chapters, you will learn how to use these Visual Studio features for your C# programming needs.

C# Windows Forms

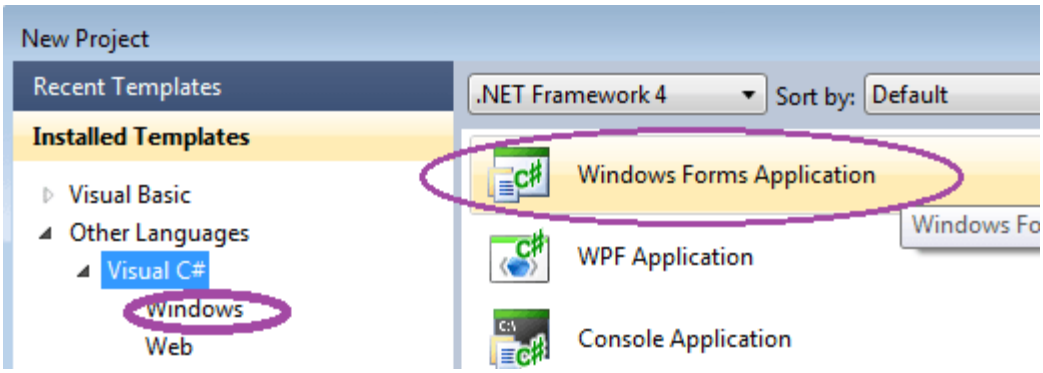
C# programmers have made extensive use of forms to build user interfaces. Each time you create a Windows application, Visual Studio will display a default blank form, onto which you can drag the controls onto your applications main form and adjust their size and position.



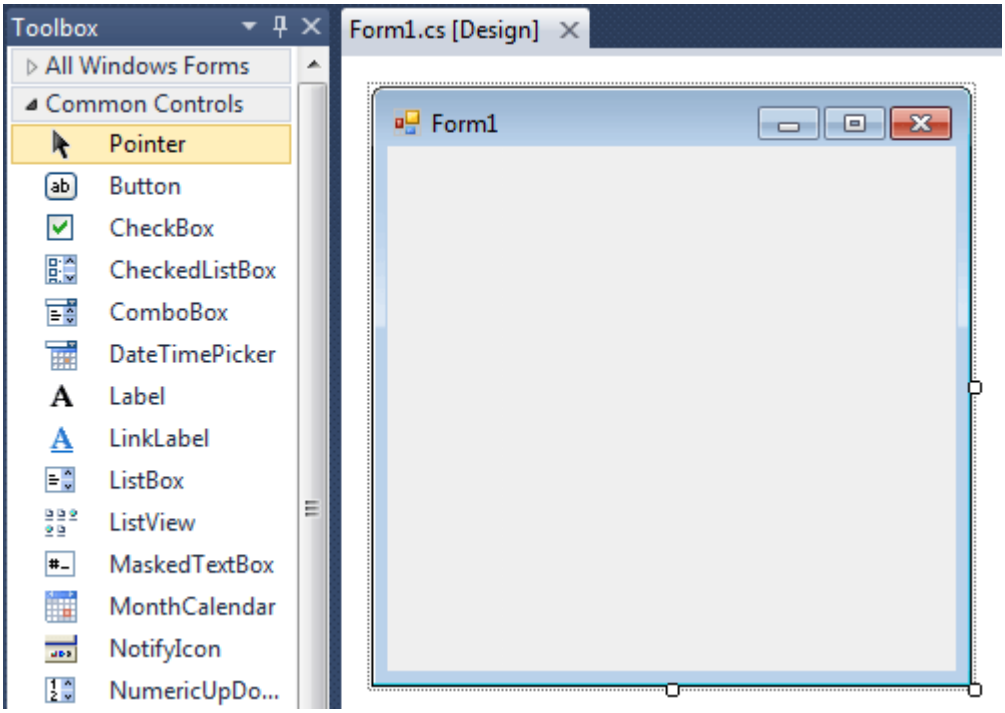
The first step is to start a new project and build a form. Open your Visual Studio and select File->New Project and from the new project dialog box select Other Languages->Visual C# and select Windows Forms Application. Enter a project name at the bottom of the dialouge box and click OK button. The following picture shows how to create a new Form in Visual Studio.



Select Windows Forms Application from New Project dialog box.

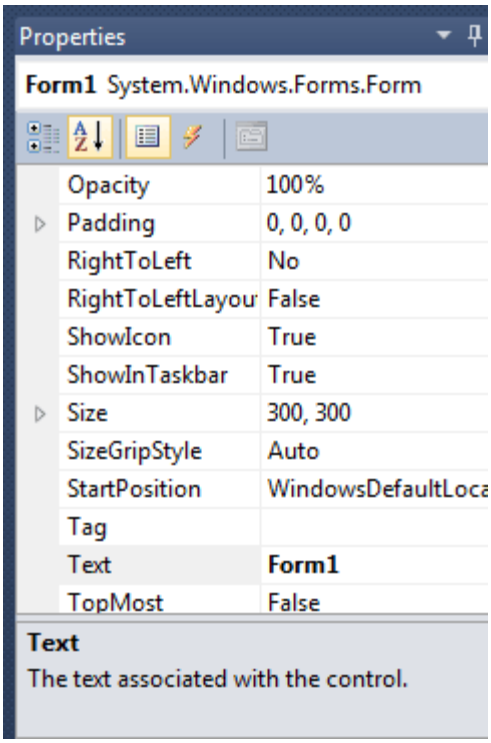


After selecting Windows Forms Application , you can see a default Form (Form1) in your new C# project. The Windows Form you see in Designer view is a visual representation of the window that will open when your application is opened. You can switch between this view and Code view at any time by right-clicking the design surface or code window and then clicking View Code or View Designer. The following picture shows how is the default Form (Form1) looks like.



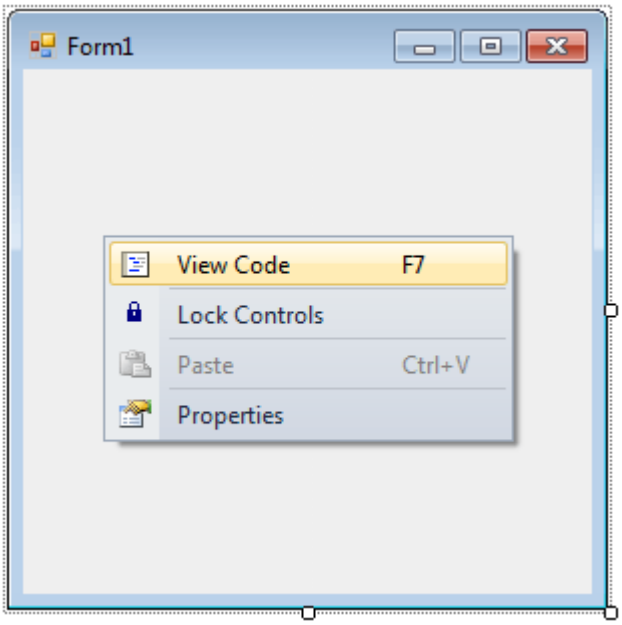
At the top of the form there is a title bar which displays the forms title. Form1 is the default name, and you can change the name to your convenience . The title bar also includes the control box, which holds the minimize, maximize, and close buttons.

If you want to set any properties of the Form, you can use Visual Studio Property window to change it. If you do not see the Properties window, on the View menu, click Properties window. This window lists the properties of the currently selected Windows Form or control, and its here that you can change the existing values.



For example , to change the forms title from Form1 to MyForm, click on Form1 and move to the right side down Properties window, set Text property to MyForm. Then you can see the Title of the form is changed. Likewise you can set any properties of Form through Properties window.

You can also set the properties of the Form1 through coding. For coding, you should right-click the design surface or code window and then clicking View Code.



When you right click on Form then you will get code behind window, there you can write your code

For example, if you want to change the back color of the form to Brown, you can code in the Form1_Load event like the following.

Likewise, you can change other properties of Form1 through coding.

How to Pass Data Between Forms

In C#, there are many situations the new programmers face the same problem about how to pass data and values from one form to another. The following link will guide you .... Pass Data Between Forms

Form on Top of All Other Windows

You can bring a Form on top of C# application by simply setting the Form.topmost form property to true will force the form to the top layer of the screen. More about.... How to keep Form on Top of All Other Windows

MDI Form

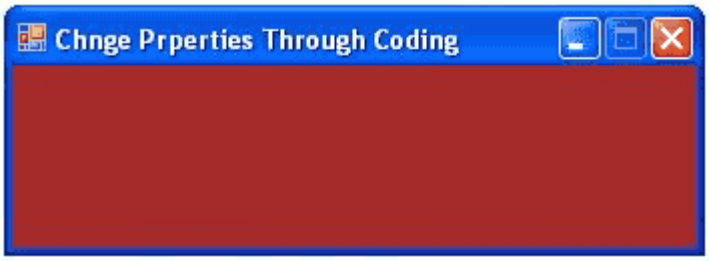
A C# Multiple Document Interface (MDI) programs can display multiple child windows inside them. This is in contrast to single document interface (SDI) applications, which can manipulate only one document at a time. More about.... C# MDI Form

The following C# source code shows how to change the Title, BackColor, Size, Location and MaximizeBox properties of Form1. Copy and paste the following C# source code to source code editor of your Visual Studio.

```
using System;
using System.Drawing;
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
   public partial class Form1 : Form
   {
      public Form1()
      {
InitializeComponent();
      }

      private void Form1_Load(object sender, EventArgs e)
```

```
        {
this.Text = "Change Prperties Through Coding";
this.BackColor = Color.Brown;
this.Size = new Size(350, 125);
this.Location = new Point(300, 300);
this.MaximizeBox = false;
        }
    }
}
```

When you execute (press F5 key) the program the form is look like the following image.



The Windows based programs you create using C# run in the context of a form. When you close the form, the application also ends.

C# Label Control

Labels are one of the most frequently used C# control. We can use the Label control to display text in a set location on the page. Label controls can also be used to add descriptive text to a Form to provide the user with helpful information. The Label class is defined in the System.Windows.Forms namespace.

Add a Label control to the form - Click Label in the Toolbox and drag it over the forms Designer and drop it in the desired location.

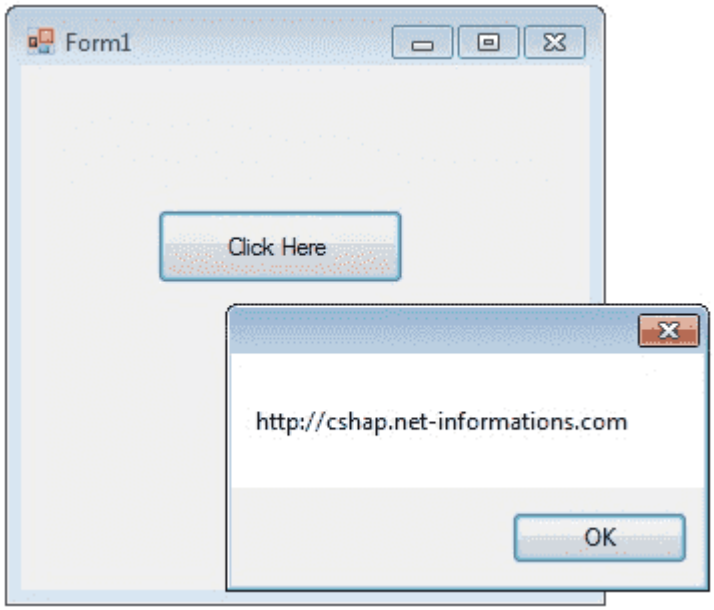
If you want to change the display text of the Label, you have to set a new text to the Text property of Label.

In addition to displaying text, the Label control can also display an image using the Image property, or a combination of the ImageIndex and ImageList properties.

The following C# source code shows how to set some properties of the Label through coding.

C# Button Control

Windows Forms controls are reusable components that encapsulate user interface functionality and are used in client side Windows applications. A button is a control, which is an interactive component that enables users to communicate with an application. The Button class inherits directly from the ButtonBase class. A Button can be clicked by using the mouse, ENTER key, or SPACEBAR if the button has focus.



When you want to change display text of the Button , you can change the Text property of the button.

Similarly if you want to load an Image to a Button control , you can code like this.

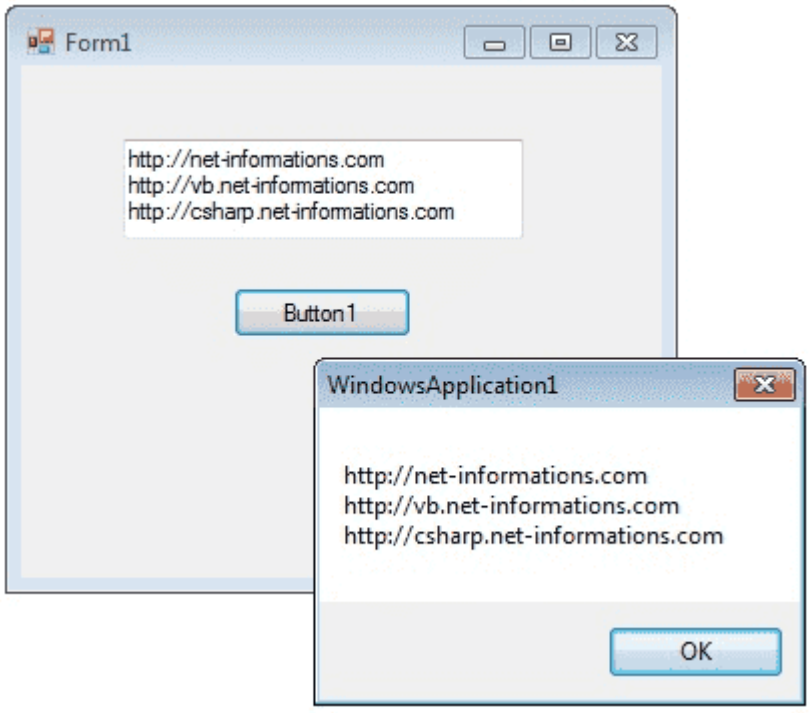
How to Call a Button's Click Event Programmatically

The Click event is raised when the Button control is clicked. This event is commonly used when no command name is associated with the Button control. Raising an event invokes the event handler through a delegate.

The following C# source code shows how to change the button Text property while Form loading event and to display a message box when pressing a Button Control.

C# TextBox Control

A TextBox control is used to display, or accept as input, a single line of text. This control has additional functionality that is not found in the standard Windows text box control, including multiline editing and password character masking.



A text box object is used to display text on a form or to get user input while a C# program is running. In a

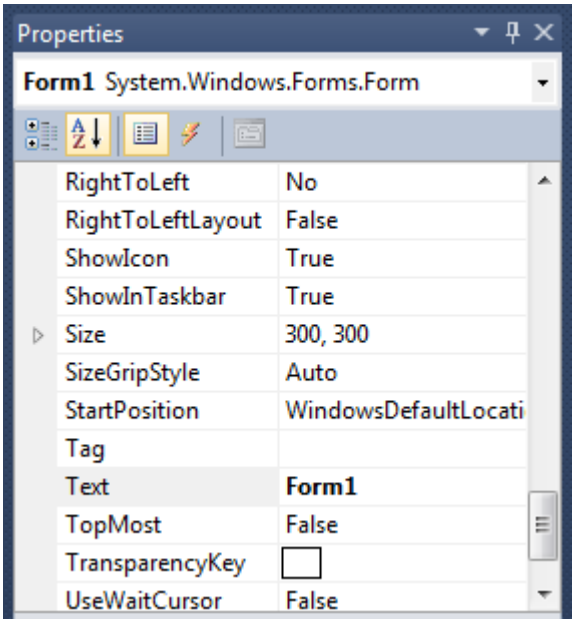
text box, a user can type data or paste it into the control from the clipboard.

For displaying a text in a TextBoxcontrol , you can code like this.

You can also collect the input value from a TextBox control to a variable like this way.

C# TextBox Properties

You can set TextBox properties through Property window or through program. You can open Properties window by pressing F4 or right click on a control and select Properties menu item.



The below code set a textbox width as 250 and height as 50 through source code.

Background Color and Foreground Color

You can set background color and foreground color through property window and programmatically.

Textbox BorderStyle

You can set 3 different types of border style for textbox, they are None, Fixed Single and fixed3d.

TextBox Events

Keydown event

You can capture which key is pressed by the user using KeyDown event

e.g.

```
private void textBox1_KeyDown(object sender, KeyEventArgs e)
{
    if (e.KeyCode == Keys.Enter)
    {
        MessageBox.Show("You press Enter Key");

    }
    if (e.KeyCode == Keys.CapsLock)
    {
        MessageBox.Show("You press Caps Lock Key");
    }
}
```

TextChanged Event

When user input or setting the Text property to a new value raises the TextChanged event

e.g.

```
private void textBox1_TextChanged(object sender, EventArgs e)
{
    label1.Text = textBox1.Text;
}
```

Textbox Maximum Length

Sets the maximum number of characters or words the user can input into the text box control.

Textbox ReadOnly

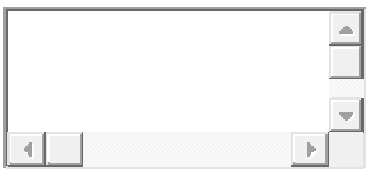
When a program wants to prevent a user from changing the text that appears in a text box, the program can set the controls Read-only property is to True.

Multiline TextBox

You can use the Multiline and ScrollBars properties to enable multiple lines of text to be displayed or entered.

Textbox password character

TextBox controls can also be used to accept passwords and other sensitive information. You can use the PasswordChar property to mask characters entered in a single line version of the control



The above code set the PasswordChar to * , so when the user enter password then it display only * instead

of typed characters.

How to Newline in a TextBox

You can add new line in a textbox using many ways.

or

How to retrieve integer values from textbox ?

Parse method Converts the string representation of a number to its integer equivalent.

String to Float conversion

String to Double conversion

Looking for a C# job ?

There are lot of opportunities from many reputed companies in the world. Chances are you will need to prove that you know how to work with .Net Programming Language. These C# Interview Questions have

been designed especially to get you acquainted with the nature of questions you may encounter during your interview for the subject of .Net Programming. Here's a comprehensive list of .Net Interview Questions, along with some of the best answers. These sample questions are framed by our experts team who trains for .Net training to give you an idea of type of questions which may be asked in interview.

Go to... C# Interview Questions

How to allow only numbers in a textbox

Many of us have faced a situation where we want the user to enter a number in a TextBox. Click the following link that are going to make a Numeric Textbox which will accept only numeric values; if there are any values except numeric. More about.... How do I make a textbox that only accepts numbers

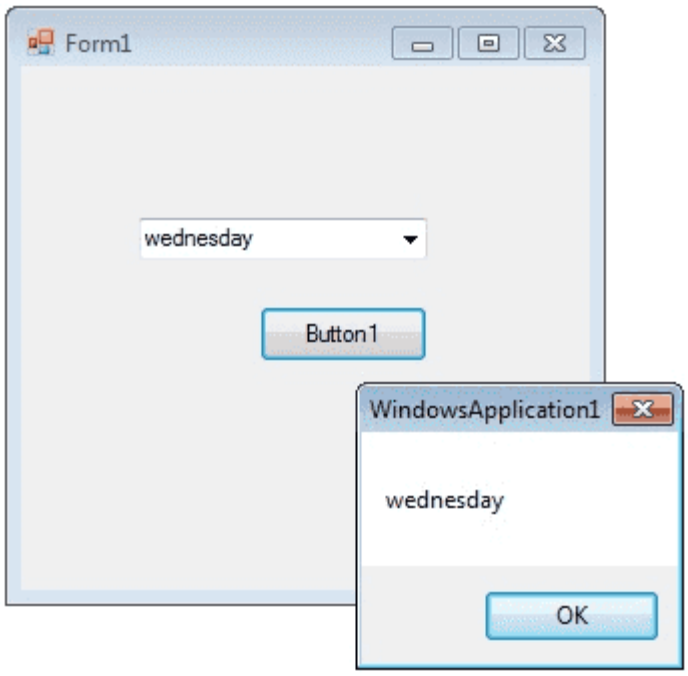
Autocomplete TextBox

From the latest version of Visual Studio, some of the controls support Autocomplete feature including the TextBox controls. The properties like Auto Complete Custom Source, Auto Complete Mode and Auto Complete Source to perform a TextBox that automatically completes user input strings by comparing the prefix letters being entered to the prefixes of all strings in a data source. More about.... C# Autocomplete TextBox

From the following C# source code you can see some important property settings to a TextBox control.

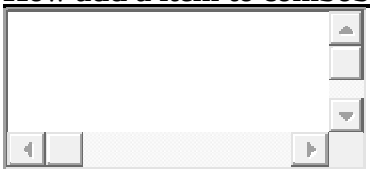
C# ComboBox Control

C# controls are located in the Toolbox of the development environment, and you use them to create objects on a form with a simple series of mouse clicks and dragging motions. A ComboBox displays a text box combined with a ListBox, which enables the user to select items from the list or enter a new value.



The user can type a value in the text field or click the button to display a drop-down list. You can add individual objects with the Add method. You can delete items with the Remove method or clear the entire list with the Clear method.

How add a item to combobox



Combo Box Selected Item

How to retrieve value from ComboBox

If you want to retrieve the displayed item to a string variable, you can code like this

How to remove an item from ComboBox

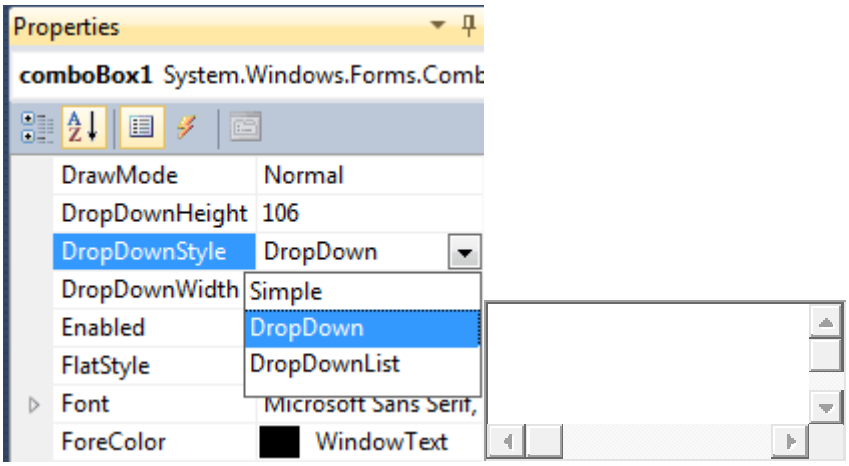
You can remove items from a combobox in two ways. You can remove item at the specified index or giving a specified item by name.

The above code will remove the second item from the combobox.

The above code will remove the item "Friday" from the combobox.

DropDownStyle

The DropDownStyle property specifies whether the list is always displayed or whether the list is displayed in a drop-down. The DropDownStyle property also specifies whether the text portion can be edited.



ComboBox Selected Value

How to set the selected item in a comboBox

You can display selected item in a combobox in two ways.

ComboBoxDataSource Property

How to populate a combo box with a DataSet ?

You can Programmatically Binding DataSource to ComboBox in a simple way..

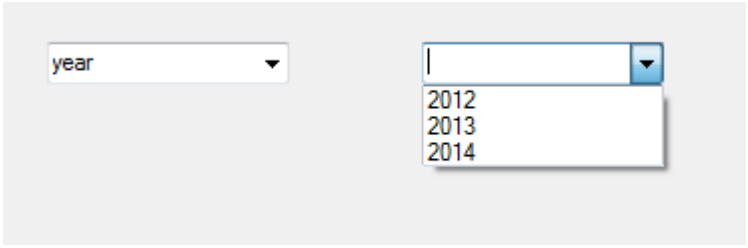
Consider an sql string like...."select au_id,au_lname from authors";

Make a datasource and bind it like the following...

ComboboxSelectedIndexChanged event

The SelectedIndexChanged event of a combobox fire when you change the slected item in a combobox. If you want to do something when you change the selection, you can write the program on SelectedIndexChanged event. From the following code you can understand how to set values in the SelectedIndexChanged event of a combobox. Drag and drop two combobox on the Form and copy and paste the following source code.

Output



ComboBox Databinding

You can bind data to a Combobox from various resources like Dataset, List, Enum, Dictionary etc. From the following link you can study more about ... ComboBox Databinding

ComboBox Default Value

How to set a default value for a Combo Box

You can set combobox default value by using SelectedIndex property

Above code set 6th item as combobox default value

ComboBoxreadonly

How to make a combobox read only

You can make a ComboBoxreadonly, that means a user cannot write in a combo box but he can select the given items, in two ways. By default, DropDownStyle property of a Combobox is DropDown. In this case user can enter values to combobox. When you change the DropDownStyle property to DropDownList, the Combobox will become read only and user can not enter values to combobox. Second method, if you want the combobox completely read only, you can set comboBox1.Enabled = false.

Autocomplete ComboBox

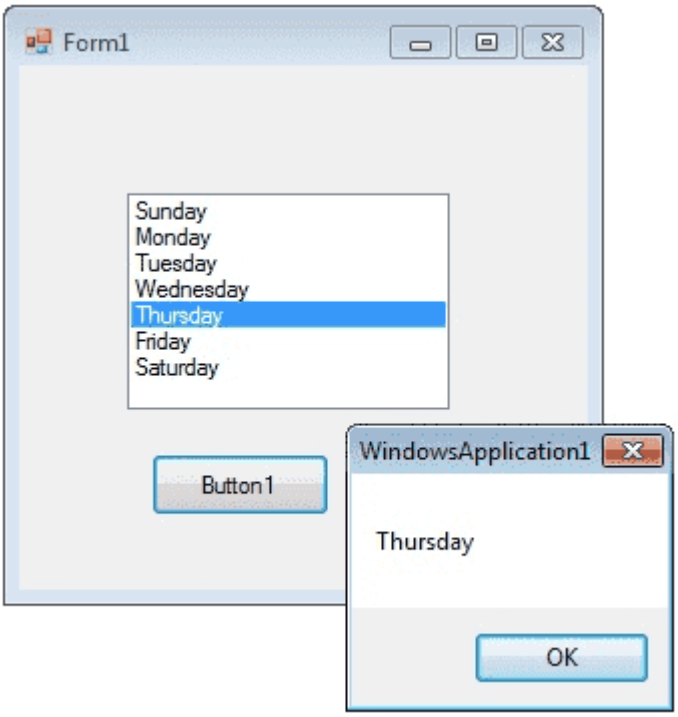
From the latest version of Visual Studio, some of the controls support Autocomplete feature including the ComboBox controls. From the following link you can see how to make .... Autocomplete ComboBox

ComboBox Example

The following C# source code add seven days in a week to a combo box while load event of a Windows Form and int Button click event it displays the selected text in the Combo Box.

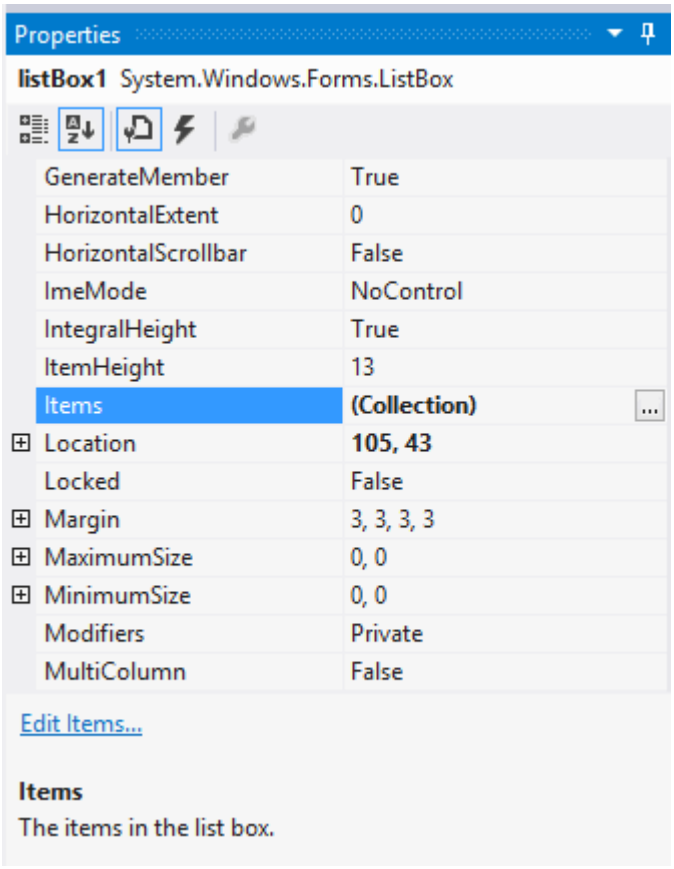
C# ListBox Control

The **ListBox control** enables you to display a list of items to the user that the user can select by clicking.



Setting ListBox Properties

You can set **ListBox properties** by using Properties Window. In order to get Properties window you can Press F4 or by right-clicking on a control to get the "Properties" menu item.

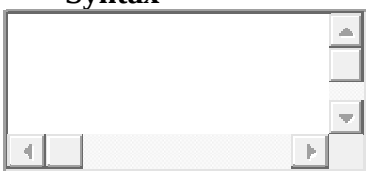


You can set property values in the right side column of the property window.

C# Listbox example

Add Items in a Listbox

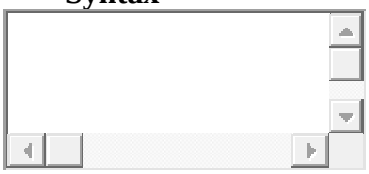
Syntax



In addition to display and selection functionality, the **ListBox** also provides features that enable you to efficiently add items to the ListBox and to find text within the items of the list. You can use the Add or Insert method to add items to a list box. The **Add method** adds new items at the end of an unsorted list box.

If the **Sorted property** of the C# ListBox is set to true, the item is inserted into the list alphabetically. Otherwise, the item is inserted at the end of the ListBox.

Insert Items in a Listbox
Syntax



You can **inserts** an item into the list box at the specified index.

Listbox Selected Item

If you want to retrieve a **single selected** item to a variable , use the following code.

Or you can use

Or

Selecting Multiple Items from Listbox

The **SelectionMode property** determines how many items in the list can be selected at a time. A ListBox control can provide single or **multiple selections** using the SelectionModeproperty . If you change the selection mode property to multiple select , then you will retrieve a collection of items from ListBox1.SelectedItems property.

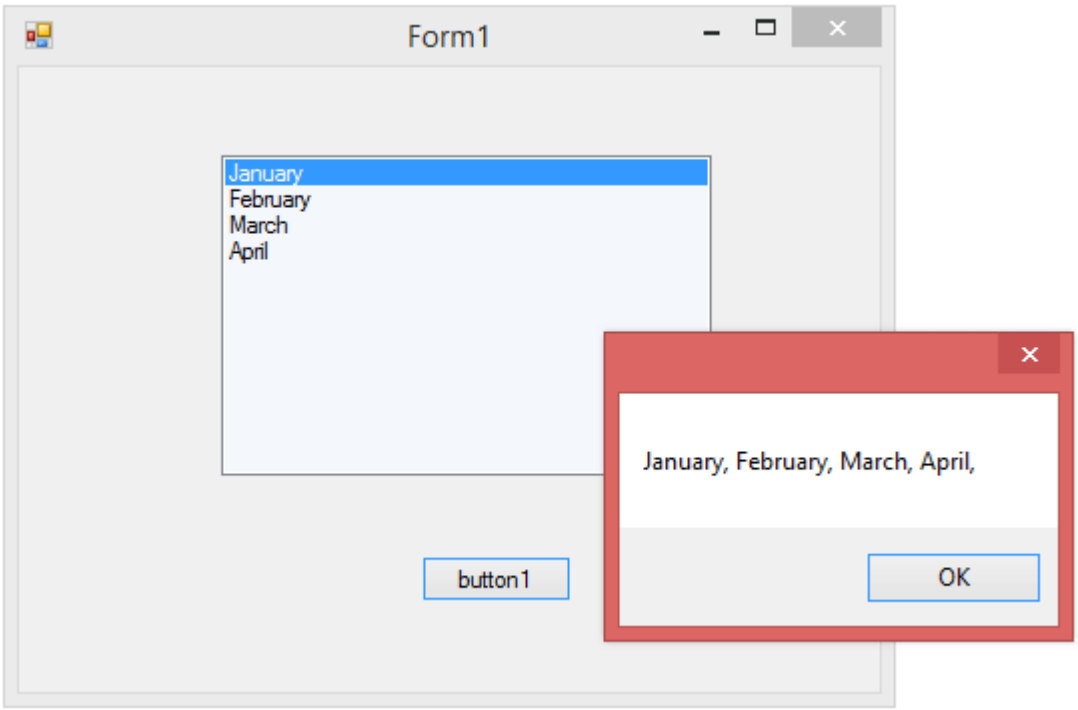
The ListBox class has two SelectionMode. **Multiple** or **Extended** .

In **Multiple mode** , you can select or deselect any item in a ListBox by clicking it. In Extended mode, you need to hold down the Ctrl key to select additional items or the Shift key to select a range of items.

The following C# program initially fill seven days in a week while in the form load event and set the selection mode property to **MultiSimple** . At the Button click event it will display the selected items.

Getting All Items from List box

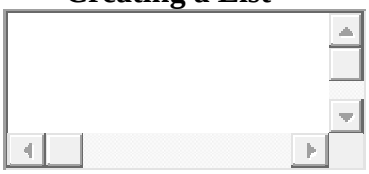
The Items collection of C# WinformsListbox returns a **Collection type** of Object so you can use ToString() on each item to display its text value as below:



How to bind a ListBox to a List ?

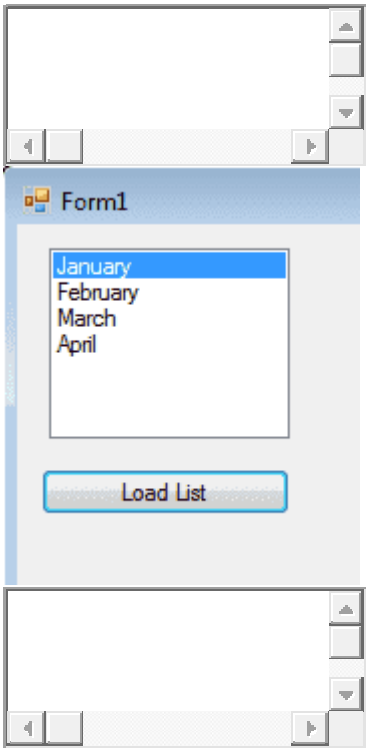
You can bind a List to a ListBox control by create a **fresh List Object** and add items to the List.

Creating a List



Binding to List

The next step is to bind this List to the Listbox. In order to do that you should **set datasource** of the Listbox.

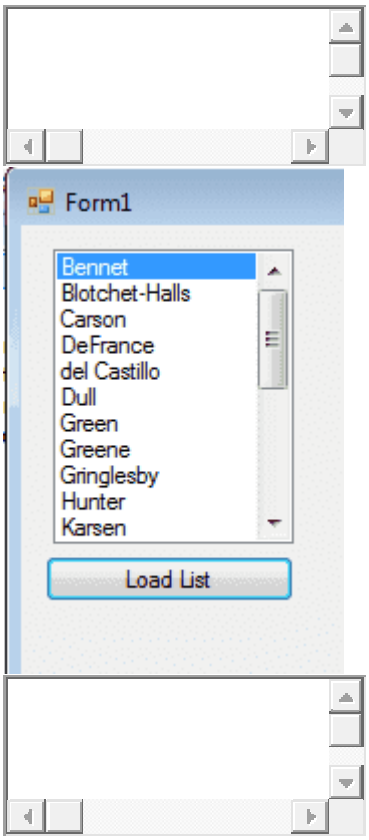


How to bind a listbox to database values ?

First of all, you could create a **connection string** and fetch data from database to a Dataset.

Set Datasource for ListBox

Next step is that you have set Listbox **datasoure** as Dataset.



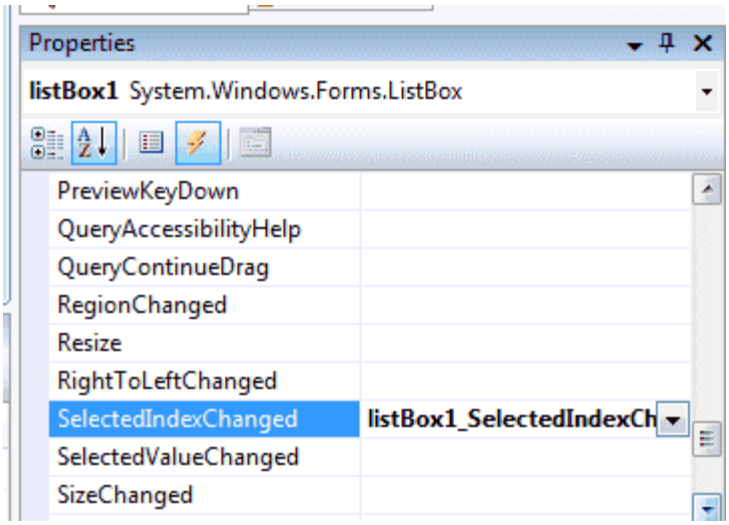
How to refresh DataSource of a ListBox ?
How to clear the Listbox if its already binded with datasource ?

When you want to clear the Listbox, if the ListBox already **binded with Datasource** , you have to set the Datasource of Listbox as null.

How to SelectedIndexChanged event in ListBox ?

This event is fired when the item selection is changed in a **ListBox** . You can use this event in a situation that you want select an item from your listbox and accodring to this selection you can perform other programming needs.

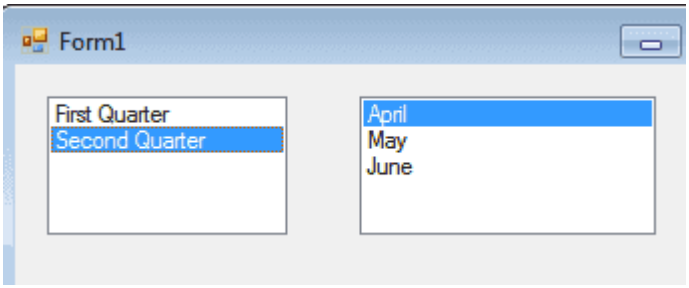
You can add the event handler using the **Properties Window** and selecting the Event icon and double-clicking on SelectedIndexChanged as you can see in following image.



The event will fire again when you select a new item. You can write your code within **SelectedIndexChanged** event . When you double click on ListBox the code will automatically come in you code editor like the following image.

```
private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    //your code here
}
```

From the following example you can understand how to fire the SelectedIndexChanged event.

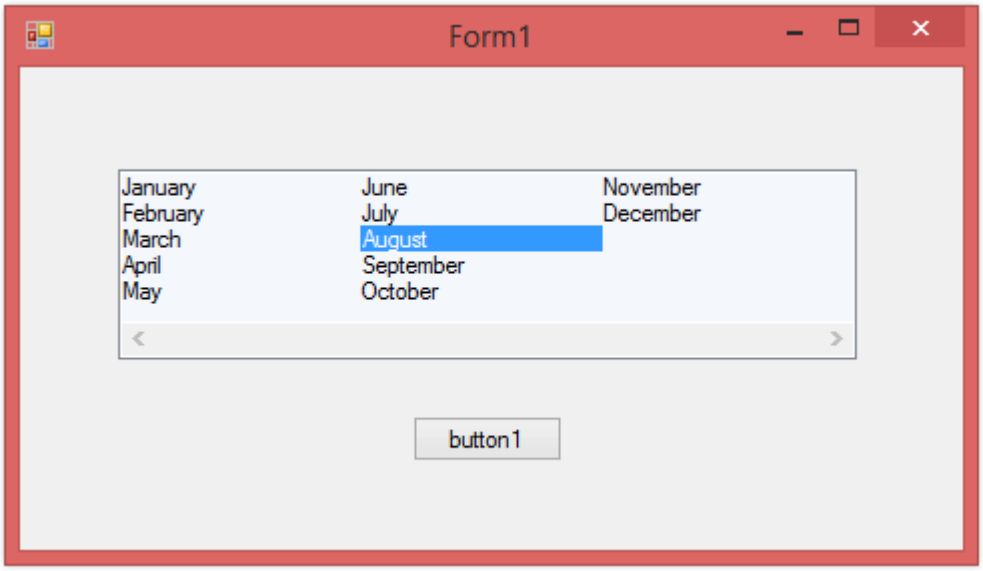


First you should drag two listboxes on your Form. First listbox you should set the List as **Datasource** , the List contents follows:

When you load this form you can see the listbox is **populated** with List and displayed first quarter and second quarter. When you click the "Fist Quarter" the next listbox is populated with first quarter months and when you click "Second Quarter" you can see the second listbox is changed to second quarter months. From the following program you can understand how this happened.

C# Listbox Column

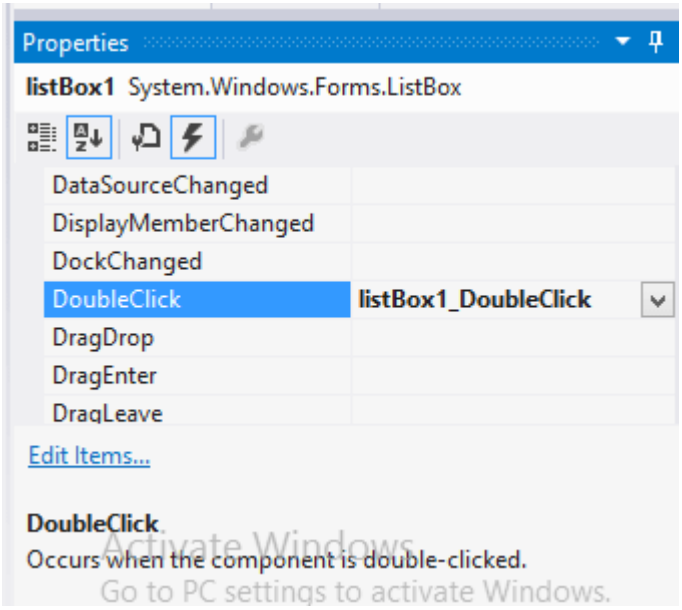
A **multicolumn ListBox** places items into as many columns as are needed to make vertical scrolling unnecessary. The user can use the keyboard to navigate to columns that are not currently visible. First of all, you have Gets or sets a value indicating whether the ListBox supports **multiple columns** .



The following C# code example demonstrates a simple muliple column ListBox.

Listbox Item Double Click Event

Add an event handler for the **Control.DoubleClick** event for your ListBox.



The following program shows when you double click the Listbox Item that event handler open up a MessageBox displaying the selected item.

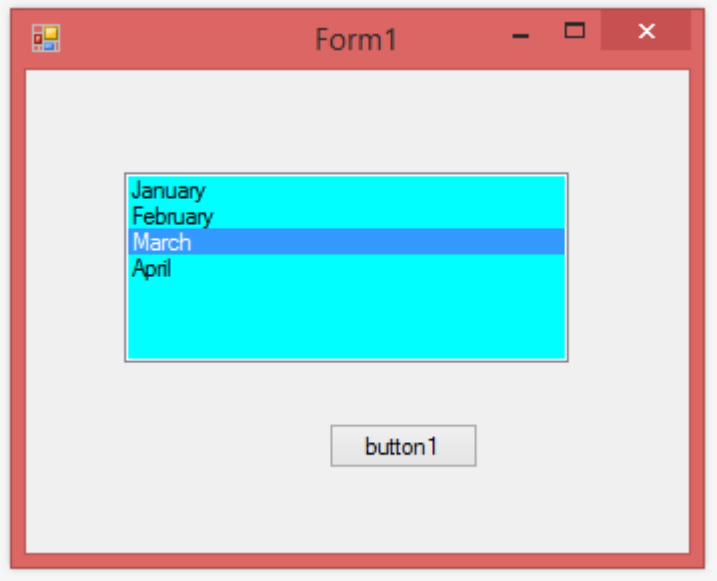
Listbox vertical scrollbar

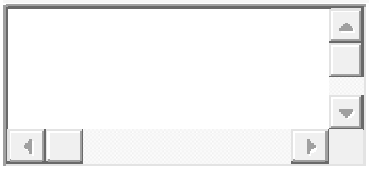
ListBox only shows a **vertical scroll bar** when there are more items then the displaying area. You can fix with ListBox.ScrollAlwaysVisible Property if you want the bar to be visible all the time. If **ListBox.ScrollAlwaysVisible** Property is true if the vertical scroll bar should always be displayed; otherwise, false. The default is false.

Background and Foreground

The Listbox **BackColor** and **ForeColor properties** are used to set the background and foreground colors respectively. If you click on these properties in the Properties window, then the Color Dialog pops up and you can select the color you want.

Alternatively, you can set **background** and foreground colors from source code. The following code sets the BackColor and ForeColor properties:



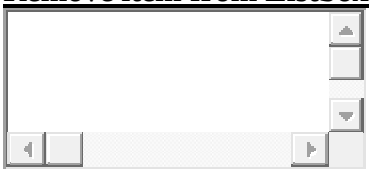


How to clear all data in a listBox?

Listbox **Items.Clear()** method clear all the items from ListBox.

ListBox.ClearSelected Method Unselects all items in the ListBox.

Remove item from Listbox



RemoveAt method removes the item at the **specified index** within the collection. When you remove an item from the list, the indexes change for subsequent items in the list. All information about the **removed item** is deleted.

Listbox Vs ListView Vs GridView

C# ListBox has many similarities with **ListView** or **GridView** (they share the parent class ItemsControl), but each control is oriented towards different situations. ListBox is best for **general UI** composition, particularly when the elements are always intended to be selectable, whereas ListView or GridView are best for **data binding** scenarios, particularly if virtualization or large data sets are involved. One most important difference is listview uses the extended selection mode by default .

keyPress event in C#

Handle Keyboard Input at the Form Level in C#

Windows Forms processes keyboard input by raising keyboard events in response to Windows messages. Most Windows Forms applications process keyboard input exclusively by handling the keyboard events.

How do I detect keys pressed in C#

You can detect most physical key presses by handling the KeyDown or KeyUp events. Key events occur in the following order:

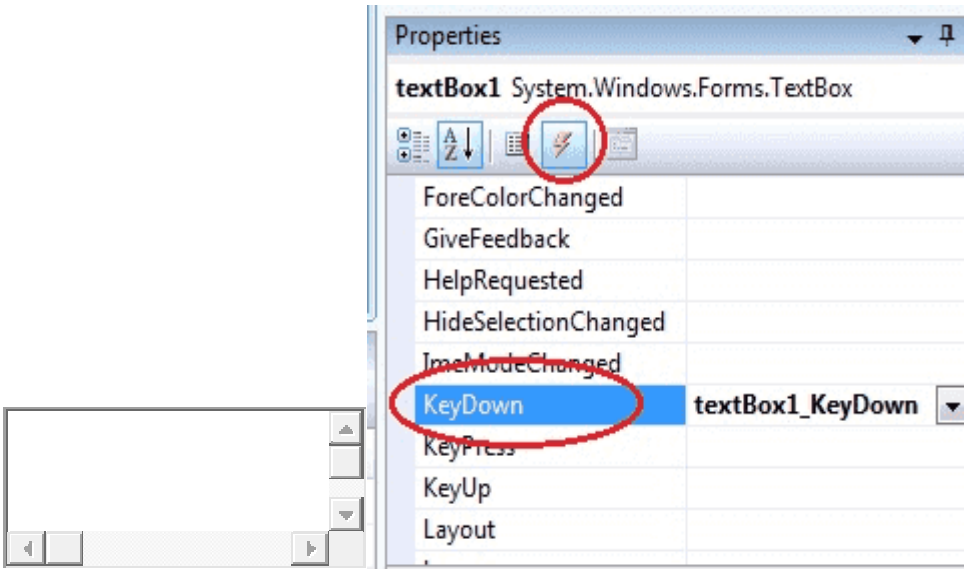
- KeyDown
- KeyPress
- KeyUp

How to detect when the Enter Key Pressed in C#

The following C# code behind creates the KeyDown event handler. If the key that is pressed is the Enter key, a MessegeBox will displayed .

How to get TextBox1_KeyDown event in your C# source file ?

Select your TextBox control on your Form and go to Properties window. Select Event icon on the properties window and scroll down and find the KeyDown event from the list and double click the Keydown Event. The you will get the KeyDown event in your source code editor.



Difference between the KeyDown Event, KeyPress Event and KeyUp Event

KeyDownEvent : This event raised as soon as the user presses a key on the keyboard, it repeats while the user keeps the key depressed.

KeyPressEvent : This event is raised for character keys while the key is pressed and then released. This event is not raised by noncharacter keys, unlike KeyDown and KeyUp, which are also raised for noncharacter keys

KeyUpEvent : This event is raised after the user releases a key on the keyboard.

KeyPressEvent :

KeyDownEvent :

How to Detecting arrow keys in C#

In order to capture keystrokes in a Forms control, you must derive a new class that is based on the class of the control that you want, and you override the ProcessCmdKey().

More about.... Detecting arrow keys from C#

KeyUpEvent :

The following C# source code shows how to capture Enter KeyDown event from a TextBox Control.

MCQ

_____control is used to display text on the form and it does not take part in user input or in mouse or keyboard events
A. Textbox
B. Checkbox
C. Button
D. Label

ANSWER: D

In which control the user can enter data in the application.
A. Textbox
B. Checkbox
C. Button
D. Label

ANSWER: A

Which property is used to set the character used to mask characters of a password in a single-line TextBox control.
A. PasswordChar
B. PassChrac

ANSWER: A

Which control allows the user to interact with the application or software.
A. Textbox
B. Checkbox
C. Button
D. Label

ANSWER: C

which event occur when the button is clicked.
A. click
B. clickEvent

ANSWER: A

Which event occur when the mouse pointer placed on the button.
A. click
B. clickEvent
C. MouseHover
D. MouseLeave

ANSWER: C

Which control can display a hyperlink
A. Textbox
B. Checkbox

C. LinkLabel
D. Label

ANSWER: C

Which control allows the user to select multiple items from a list of items.

A. Textbox
B. Checkbox
C. LinkLabel
D. CheckedListBox

ANSWER: D

Which property property is used to set the width of the of the drop-down portion of a ComboBox control.
A. DropDownWidth
B. Width

ANSWER: A

Which property is used to set the maximum number of items to be shown in the drop-down portion of the ComboBox control.
A. MaxDropDownItems
B. DoropDown

ANSWER: A

_____control is used to show multiple elements in a list
A. Textbox
B. Checkbox
C. LinkLabel
D. ListBox

ANSWER: D

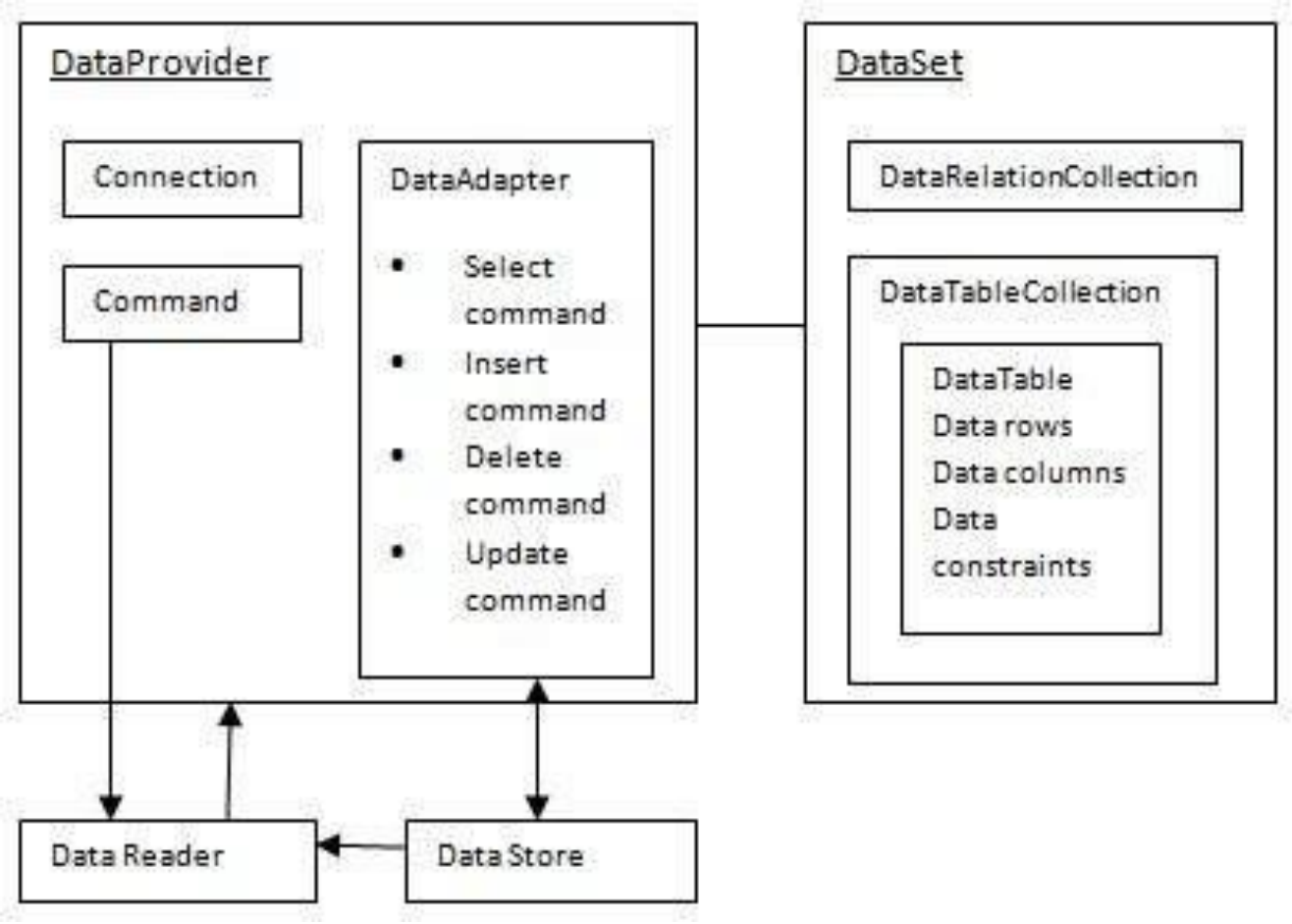
UNIT V

ADO.net:

- **The component model, creating database connection, database command, data repeater,**
- **connecting to data sources, choosing a .net data provider, manage a connection, building command objects, executing commands**
- **building datasets**
- **data tables and data adapter**

ADO.NET provides a bridge between the front end controls and the back end database. The ADO.NET objects encapsulate all the data access operations and the controls interact with these objects to display data, thus hiding the details of movement of data.

The following figure shows the ADO.NET objects at a glance:



The DataSet Class

The dataset represents a subset of the database. It does not have a continuous connection to the database. To update the database a reconnection is required. The DataSet contains DataTable objects and DataRelation objects. The DataRelation objects represent the relationship between two tables.

Following table shows some important properties of the DataSet class:

Properties	Description
CaseSensitive	Indicates whether string comparisons within the data tables are case-sensitive.
Container	Gets the container for the component.
DataSetName	Gets or sets the name of the current data set.
DefaultViewManager	Returns a view of data in the data set.
DesignMode	Indicates whether the component is currently in design mode.
EnforceConstraints	Indicates whether constraint rules are followed when attempting any update operation.
Events	Gets the list of event handlers that are attached to this component.
ExtendedProperties	Gets the collection of customized user information associated with the DataSet.
HasErrors	Indicates if there are any errors.
IsInitialized	Indicates whether the DataSet is initialized.
Locale	Gets or sets the locale information used to compare strings within the table.

Namespace	Gets or sets the namespace of the DataSet.
Prefix	Gets or sets an XML prefix that aliases the namespace of the DataSet.
Relations	Returns the collection of DataRelation objects.
Tables	Returns the collection of DataTable objects.

The following table shows some important methods of the DataSet class:

Methods	Description
AcceptChanges	Accepts all changes made since the DataSet was loaded or this method was called.
BeginInit	Begins the initialization of the DataSet. The initialization occurs at run time.
Clear	Clears data.
Clone	Copies the structure of the DataSet, including all DataTable schemas, relations, and constraints. Does not copy any data.
Copy	Copies both structure and data.
CreateDataReader()	Returns a DataTableReader with one result set per DataTable, in the same sequence as the tables appear in the Tables collection.
CreateDataReader(DataTable[])	Returns a DataTableReader with one result set per DataTable.

EndInit	Ends the initialization of the data set.
Equals(Object)	Determines whether the specified Object is equal to the current Object.
Finalize	Free resources and perform other cleanups.
GetChanges	Returns a copy of the DataSet with all changes made since it was loaded or the AcceptChanges method was called.
GetChanges(DataRowState)	Gets a copy of DataSet with all changes made since it was loaded or the AcceptChanges method was called, filtered by DataRowState.
GetDataSetSchema	Gets a copy of XmlSchemaSet for the DataSet.
GetObjectData	Populates a serialization information object with the data needed to serialize the DataSet.
GetType	Gets the type of the current instance.
GetXML	Returns the XML representation of the data.
GetXMLSchema	Returns the XSD schema for the XML representation of the data.
HasChanges()	Gets a value indicating whether the DataSet has changes, including new, deleted, or modified rows.
HasChanges(DataRowState)	Gets a value indicating whether the DataSet has changes, including new, deleted, or modified rows, filtered by DataRowState.

IsBinarySerialized	Inspects the format of the serialized representation of the DataSet.
Load(IDataReader, LoadOption, DataTable[])	Fills a DataSet with values from a data source using the supplied IDataReader, using an array of DataTable instances to supply the schema and namespace information.
Load(IDataReader, LoadOption, String[])	Fills a DataSet with values from a data source using the supplied IDataReader, using an array of strings to supply the names for the tables within the DataSet.
Merge()	Merges the data with data from another DataSet. This method has different overloaded forms.
ReadXML()	Reads an XML schema and data into the DataSet. This method has different overloaded forms.
ReadXMLSchema(0)	Reads an XML schema into the DataSet. This method has different overloaded forms.
RejectChanges	Rolls back all changes made since the last call to AcceptChanges.
WriteXML()	Writes an XML schema and data from the DataSet. This method has different overloaded forms.
WriteXMLSchema()	Writes the structure of the DataSet as an XML schema. This method has different overloaded forms.

The DataTable Class

The DataTable class represents the tables in the database. It has the following important properties; most of these properties are read only properties except the PrimaryKey property:

Properties	Description
ChildRelations	Returns the collection of child relationship.
Columns	Returns the Columns collection.
Constraints	Returns the Constraints collection.
DataSet	Returns the parent DataSet.
DefaultView	Returns a view of the table.
ParentRelations	Returns the ParentRelations collection.
PrimaryKey	Gets or sets an array of columns as the primary key for the table.
Rows	Returns the Rows collection.

The following table shows some important methods of the DataTable class:

Methods	Description
AcceptChanges	Commits all changes since the last AcceptChanges.
Clear	Clears all data from the table.
GetChanges	Returns a copy of the DataTable with all changes made since the AcceptChanges method was called.
GetErrors	Returns an array of rows with errors.
ImportRows	Copies a new row into the table.

LoadDataRow	Finds and updates a specific row, or creates a new one, if not found any.
Merge	Merges the table with another DataTable.
NewRow	Creates a new DataRow.
RejectChanges	Rolls back all changes made since the last call to AcceptChanges.
Reset	Resets the table to its original state.
Select	Returns an array of DataRow objects.

The DataRow Class

The DataRow object represents a row in a table. It has the following important properties:

Properties	Description
HasErrors	Indicates if there are any errors.
Items	Gets or sets the data stored in a specific column.
ItemArrays	Gets or sets all the values for the row.
Table	Returns the parent table.

The following table shows some important methods of the DataRow class:

Methods	Description

AcceptChanges	Accepts all changes made since this method was called.
BeginEdit	Begins edit operation.
CancelEdit	Cancels edit operation.
Delete	Deletes the DataRow.
EndEdit	Ends the edit operation.
GetChildRows	Gets the child rows of this row.
GetParentRow	Gets the parent row.
GetParentRows	Gets parent rows of DataRow object.
RejectChanges	Rolls back all changes made since the last call to AcceptChanges.

The DataAdapter Object

The DataAdapter object acts as a mediator between the DataSet object and the database. This helps the Dataset to contain data from multiple databases or other data source.

The DataReader Object

The DataReader object is an alternative to the DataSet and DataAdapter combination. This object provides a connection oriented access to the data records in the database. These objects are suitable for read-only access, such as populating a list and then breaking the connection.

DbCommand and DbConnection Objects

The DbConnection object represents a connection to the data source. The connection could be shared among different command objects.

The DbCommand object represents the command or a stored procedure sent to the database from retrieving or manipulating data.

Example

So far, we have used tables and databases already existing in our computer. In this example, we will create a table, add column, rows and data into it and display the table using a GridView object.

The source file code is as given:

```
<%@PageLanguage="C#"AutoEventWireup="true"CodeBehind="Default.aspx.cs"Inherits="createdatabas
e._Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<htmlxmlns="http://www.w3.org/1999/xhtml">

<headrunat="server">
<title>
      Untitled Page
</title>
</head>

<body>
<formid="form1"runat="server">

<div>
<asp:GridViewID="GridView1"runat="server">
</asp:GridView>
</div>

</form>
</body>

</html>
```

The code behind file is as given:

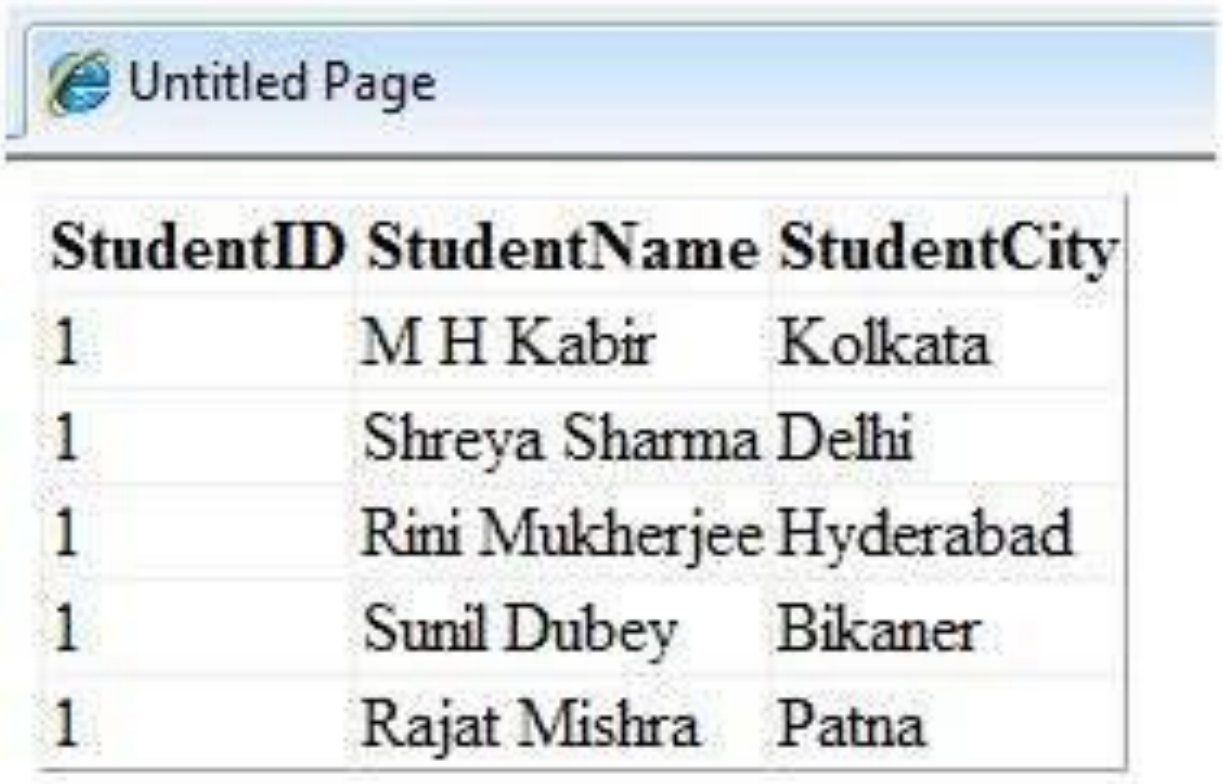
```
namespacecreatedatabase
{
publicpartialclass_Default:System.Web.UI.Page
{
protectedvoidPage_Load(object sender,EventArgs e)
{
if(!IsPostBack)
{
DataSet ds =CreateDataSet();
GridView1.DataSource=ds.Tables["Student"];
GridView1.DataBind();
}
}

privateDataSetCreateDataSet()
```

```
{
//creating a DataSet object for tables
DataSet dataset =newDataSet();

// creating the student table
DataTableStudents=CreateStudentTable();
dataset.Tables.Add(Students);
return dataset;
}

privateDataTableCreateStudentTable()
{
DataTableStudents=newDataTable("Student");

// adding columns
AddNewColumn(Students,"System.Int32","StudentID");
AddNewColumn(Students,"System.String","StudentName");
AddNewColumn(Students,"System.String","StudentCity");

// adding rows
AddNewRow(Students,1,"M H Kabir","Kolkata");
AddNewRow(Students,1,"Shreya Sharma","Delhi");
AddNewRow(Students,1,"Rini Mukherjee","Hyderabad");
AddNewRow(Students,1,"Sunil Dubey","Bikaner");
AddNewRow(Students,1,"Rajat Mishra","Patna");

returnStudents;
}

privatevoidAddNewColumn(DataTabletable,stringcolumnType,stringcolumnName)
{
DataColumn column =table.Columns.Add(columnName,Type.GetType(columnType));
}

//adding data into the table
privatevoidAddNewRow(DataTabletable,intid,stringname,string city)
{
DataRownewrow=table.NewRow();
newrow["StudentID"]= id;
newrow["StudentName"]= name;
newrow["StudentCity"]= city;
table.Rows.Add(newrow);
}
}
}
```

When you execute the program, observe the following:

- The application first creates a data set and binds it with the grid view control using the DataBind() method of the GridView control.
- The Createdataset() method is a user defined function, which creates a new DataSet object and then calls another user defined method CreateStudentTable() to create the table and add it to the Tables collection of the data set.
- The CreateStudentTable() method calls the user defined methods AddNewColumn() and AddNewRow() to create the columns and rows of the table as well as to add data to the rows.

When the page is executed, it returns the rows of the table as shown:



MCQ

1. To use the .NET Framework Data Provider for SQL Server, an application must reference the _____ namespace.

a) System.Data.Client

b) System.Data.SqlClient

c) System.Data.Sql

d) None of the mentioned

Answer: b

Explanation: System.Data.SqlClient provides access to versions of SQL Server, which encapsulates database-specific protocols.

2. Point out the correct statement.

a) Using the System.Data.SqlClient, you can fill a memory-resident DataSet that you can use to query and update the database

b) System.Data.SqlClient includes a tabular data stream (TDS) parser to communicate directly with SQL Server

c) SqlBulkCopyColumnMapping lets you efficiently bulk load a SQL Server table with data from another source

d) None of the mentioned

Answer: b

Explanation: The.NET Framework Data Provider for SQL Server describes a collection of classes used to access a SQL Server database in the managed space.

3. Valid Code for Creating a SqlConnection Object would be _____

a)


```
SqlConnection conn = NEW SqlConnection(
  "Data Source=(local);Initial Catalog=Northwind;Integrated Security=SSPI");
```

b)

```
SqlConnect conn = NEW SqlConnection(
  "Data Source=(local);Initial Catalog=Northwind;Integrated Security=SSPI");
```

c)

```
SqlConnection conn = NEW SqlConnect(
  "Data Source=(local);Initial Catalog=Northwind;Integrated Security=SSPI");
```

d) All of the mentioned

Answer: a

Explanation: A SqlConnection is an object, just like any other C# object. The SqlConnection object instantiated above uses a constructor with a single argument of type string.

4. Code snippet for having a named instance of SQL Server would be _____

a) “Server=localhost\sqlexpress”

b) “Server=local\sqlexpress”

c) “Server=host\sqlexpress”

d) “Ser=localhost\sqlexpress”

Answer: a

Explanation: An instance of the SqlConnection class in .NET Framework is supported by the Data Provider for SQL Server Database.

5. Point out the wrong statement.

a) The goal of dotConnect for SQL Server is to enable developers to maintain database applications

b) dotConnect for SQL Server combines connected and disconnected data access models in single SqlDataTable component

c) dotConnect for SQL Server supports new ADO.NET features and technologies as soon as they are released

d) None of the mentioned

Answer: a

Explanation: The goal of dotConnect for SQL Server is to enable developers to write efficient and flexible database applications. The dotConnect for SQL Server assemblies are implemented using optimized code and advanced data access algorithms.

6. Which of the following is enumeration for ADO.net with SQL Server?

a) SqlInfo

b) SqlBulkCopyOptions

c) SqlNotification

d) All of the mentioned

7. Syntax for closing and opening the connection in ADO.net is _____

a) sqlConn.Open() and sqlConn.close()

b) sqlConn.open() and sqlConn.Close()

c) sqlConn.Open() and sqlConn.Close()

d) none of the mentioned

Answer: c

Explanation: The Close method rolls back any pending transactions and releases the Connection from the

SQL Server Database.

8. Which of the following gives trusted Connection from a CE device?

a)

```
connetionString="Data Source=ServerName;
Initial Catalog=DatabaseName;Integrated Security=SSPI;
User ID=myDomain\UserName;Password=Password;
```

b)

```
connetionString="Data Source=ServerName;
Integrated Security=SSPI;User ID=myDomain\UserName;P
assword=Password;
```

c)

```
connetionString="Data Source=ServerName;
Initial Catalog=DatabaseName;User ID=myDomain\UserName;
Password=Password;
```

d) All of the mentioned

Answer: a

Explanation: Code snippet will work only on CE device.

9. The main features of dotConnect for SQL Server includes _____

a) Extra data binding capabilities

b) Ability of monitoring query execution

c) Supports the latest versions of SQL Server

d) All of the mentioned

Answer: d

Explanation: dotConnect for SQL Server includes base-class-based provider model, provider factories, connection string builder, metadata schemas, asynchronous commands, pooling enhancements, batch update support, provider-specific types, server enumeration, database change notification support and so on.

10. _____object is used to fill a DataSet/DataTable with query results in ADO.net.

a) DataReader

b) Dataset

c) DataAdapter

d) DataTables

Answer: c

Explanation: A DataAdapter object can be thought of as the adapter between the connected and disconnected data models.

UNIT VI

Managing Console I/O operations:

- **Console class, console input, console output, formatted output,**
- **numeric formatting, standardnumericformat,customnumericformat.**
- **ManagingErrors**

A **file** is a collection of data stored in a disk with a specific name and a directory path. When a file is opened for reading or writing, it becomes a **stream**.

The stream is basically the sequence of bytes passing through the communication path. There are two main streams: the **input stream** and the **output stream**. The **input stream** is used for reading data from file (read operation) and the **output stream** is used for writing into the file (write operation).

C# I/O Classes

The System.IO namespace has various classes that are used for performing numerous operations with files, such as creating and deleting files, reading from or writing to a file, closing a file etc.

The following table shows some commonly used non-abstract classes in the System.IO namespace –

Sr.No.	I/O Class & Description
1	**BinaryReader** Reads primitive data from a binary stream.
2	**BinaryWriter** Writes primitive data in binary format.
3	**BufferedStream** A temporary storage for a stream of bytes.
4	**Directory** Helps in manipulating a directory structure.

5	**DirectoryInfo** Used for performing operations on directories.
6	**DriveInfo** Provides information for the drives.
7	**File** Helps in manipulating files.
8	**FileInfo** Used for performing operations on files.
9	**FileStream** Used to read from and write to any location in a file.
10	**MemoryStream** Used for random access to streamed data stored in memory.
11	**Path** Performs operations on path information.
12	**StreamReader** Used for reading characters from a byte stream.
13	**StreamWriter** Is used for writing characters to a stream.
14	**StringReader** Is used for reading from a string buffer.
15	**StringWriter** Is used for writing into a string buffer.

The FileStream Class

The **FileStream** class in the System.IO namespace helps in reading from, writing to and closing files. This class derives from the abstract class Stream.

You need to create a **FileStream** object to create a new file or open an existing file. The syntax for creating a **FileStream** object is as follows −

```
FileStream<object_name> = new FileStream( <file_name>, <FileMode Enumerator>,
<FileAccess Enumerator>, <FileShare Enumerator>);
```

For example, we create a FileStream object **F** for reading a file named **sample.txt as shown** −

```
FileStream F = new FileStream("sample.txt", FileMode.Open, FileAccess.Read,
FileShare.Read);
```

Sr.No.	Parameter & Description
1	**FileMode** The **FileMode** enumerator defines various methods for opening files. The members of the FileMode enumerator are − • **Append** − It opens an existing file and puts cursor at the end of file, or creates the file, if the file does not exist. • **Create** − It creates a new file. • **CreateNew** − It specifies to the operating system, that it should create a new file. • **Open** − It opens an existing file. • **OpenOrCreate** − It specifies to the operating system that it should open a file if it exists, otherwise it should create a new file. • **Truncate** − It opens an existing file and truncates its size to zero bytes.
2	**FileAccess** **FileAccess** enumerators have members: **Read**, **ReadWrite** and **Write**.
3	**FileShare** **FileShare** enumerators have the following members −

	• **Inheritable** − It allows a file handle to pass inheritance to the child processes • **None** − It declines sharing of the current file • **Read** − It allows opening the file for readin. • **ReadWrite** − It allows opening the file for reading and writing • **Write** − It allows opening the file for writing

Advanced File Operations in C#

The preceding example provides simple file operations in C#. However, to utilize the immense powers of C# System.IO classes, you need to know the commonly used properties and methods of these classes.

Sr.No.	Topic & Description
1	Reading from and Writing into Text files It involves reading from and writing into text files. The **StreamReader** and **StreamWriter** class helps to accomplish it.
2	Reading from and Writing into Binary files It involves reading from and writing into binary files. The **BinaryReader** and **BinaryWriter** class helps to accomplish this.
3	Manipulating the Windows file system It gives a C# programamer the ability to browse and locate Windows files and directories.

In C#, everything the user types is a string and the compiler would hardly analyze it without your explicit asking it to do so. Therefore, if you want to get a number from the user, first request a string. after getting the string, you must convert it to a number. To perform this conversion, each data type of the .NET Framework provides a mechanism called Parse. To use Parse(), type the data type, followed by a period, followed by Parse, and followed by parentheses. In the parentheses of Parse, type the string that you requested from the user.

- **Exception Handling inC#**

Types of errors are : Compile time error and run time error.

Exception - An exception is a problem that arises during the execution of a program. A C#

exception is a response to an exceptional circumstance that arises while a program is running, such as an attempt to divide by zero.

C# exception handling is built upon four keywords: **try**, **catch**, **finally**, and **throw**.

- **try**: A try block identifies a block of code for which particular exceptions is activated. It is followed by one or more catchblocks.

- **catch**: A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The catch keyword indicates the catching of anexception.

- **finally**: The finally block is used to execute a given set of statements, whether an exception is thrown or not thrown. For example, if you open a file, it must be closed whether an exception is raised ornot.

- **throw**: A program throws an exception when a problem shows up. This is done using a throwkeyword.

MCQ

1. Which of the classes provide the operation of reading from and writing to the console in C#.NET?
a) System.Array
b) System.Output
c) System.ReadLine
d) System.Console

Answer: d
Explanation: The method for reading and writing to the console in C#.NET is provided by System.Console class. This class gives us access to the standard input, output and standard error streams.
2. Which of the given stream methods provide access to the output console by default in C#.NET?
a) Console.In
b) Console.Out
c) Console.Error
d) All of the mentioned

Answer: b
Explanation: The standard output stream Console.Out sends output to the screen by default.
3. Which of the given stream methods provide access to the input console in C#.NET?
a) Console.Out
b) Console.Error
c) Console.In
d) All of the mentioned

Answer: c
Explanation: Console.In is an instance of TextReader, and we can use the methods and properties defined by TextReader to access it to read the input from the keyboard.

4. The number of input methods defined by the stream method Console.In in C#.NET is?
a) 4

b) 3
c) 2
d) 1

Answer: b
Explanation: Two basic methods : read() and readline() and third method readkey() introduced in .NET FrameWork 2.0.
5. Select the correct methodS provided by Console.In?
a) Read(), ReadLine()
b) ReadKey(), ReadLine()
c) Read(), ReadLine(), ReadKey()
d) ReadKey(), ReadLine()

Answer: c
Explanation: The two method Read() and ReadLine() available in .NET Framework 1.0 and Third method ReadKey() was added by .NET Framework 2.0.
6. Choose the output returned when read() reads the character from the console?
a) String
b) Char
c) Integer
d) Boolean

Answer: c
Explanation: Read() returns the character read from the console. It returns the result. The character is returned as an int, which should be cast to char.
7. Choose the output returned when an error condition is generated while read() reads from the console.
a) False
b) 0
c) -1
d) All of the mentioned

Answer: c
Explanation: Read() returns –1 on error. This method also throws an IOException on failure.

8. Choose the object of TextReader class.

a) Console.In

b) Console.Out

c) Console.Error

d) None of the mentioned

Answer: a

Explanation: Console.In is an instance(object) of TextReader class and we can use the methods and properties defined by TextReader to invoke the object console.in.

9. Choose the object/objects defined by the Textwriter class.

a) Console.In

b) Console

c) Console.Error

d) None of the mentioned

Answer: c

Explanation: Console.Out and Console.Error are objects of type TextWriter class.

10. What will be the output of the following C# code?

```
 static void Main(string[] args)
 {
    int a = 10, b = 0;
    int result;
    Console.Out.WriteLine("This will generate an exception.");
    try
    {
       result = a / b; // generate an exception
    }
    catch (DivideByZeroException exc)
    {
       Console.Error.WriteLine(exc.Message);
    }
    Console.ReadLine();
 }
```

a) This will generate an exception

b) 0
c) Compile time error
d)

This will generate an exception
Attempted to Divide by Zero

Answer: d
Explanation: None.
11. Choose the methods provided by Console.Out and Console.Error?
a) Write
b) WriteLine
c) WriteKey
d) Write & WriteLine

Answer: d
Explanation: None.
12. What will be the output of the following C# code?

```
 static void Main(string[] args)
 {
    Console.WriteLine("This is a Console Application:");
    Console.Write("Please enter your lucky number:");
    string val1 = Console.ReadLine();
    int val2 = System.Convert.ToInt32(val1, 10);
    val2 = val2 * val2;
    Console.WriteLine("square of number is:" +val2);
    Console.Read();
 }
```

a) Compile time error
b) Runs successfully does not print anything
c) Runs successfully, ask for input and hence displays the result
d) Syntax Error

Answer: c

Explanation: None.

Output : This is a Console Application:

Please enter your lucky number: 3

Square of number is : 9

PRACTICE QUESTIONS

- Write a program in C# Sharp to find the sum of all elements of thearray.
- Explain the compilation and execution process of C#program.
- How do the value types differ from referencetype?
- Explain the operators inC#.
- What is “fallthrough” in switch statement. How is it achieved inC#.
- Give an example where foreach loop isused.
- What are the different types of arrays. Explain jagged array with aneg.
- What is a copy constructor? Why do we need such aconstructor?
- Define inheritance. What are the different types ofinheritance?
- What is a property? Explain with a help of anexample.
- What is an indexer? What is it used for?
- What is hiding a method? How is it different fromoverriding?
- What are sealed classes and sealed methods. Give examples.
- What is explicit interface implementation? When is it used?
- Explain exception handling inC#.
- Write a C# Sharp program to print Hello and your name in a separateline
- Write a C# Sharp program to print the sum of twonumbers
- Write a C# Sharp program to print the result of dividing twonumbers.
- Write a C# Sharp program that takes a number as input and print itsmultiplication table.
- Write a C# Sharp program to swap twonumbers.
- Write a C# Sharp program to print on screen the output of adding, subtracting, multiplying and dividing of two numbers which will be entered by theuser
- Write a C# Sharp program that takes four numbers as input to calculate and printthe average.
- Write a C# Sharp program that takes a number and a width also a number, as input and then displays a triangle of that width, using thatnumber.
- Write a C# Sharp program to check whether a given number is positive ornegative.
- Write a C# Sharp program to find whether a given year is a leap year ornot.
- Write a Program in C# to Check whether a number is Palindrome ornot.
- Write a Program in C# to demonstrate Command line argumentsProcessing.

- Write a Program in C# to find the roots of QuadraticEquation.
- Write a Program in C# to demonstrate boxing andUnBoxing.
- Write a Program in C# to implement Stackoperations.
- Write a program to demonstrate Operatoroverloading.
- Write a Program in C# to find the second largest element in asingle
- dimensionalarray.
- Write a Program in C# to multiply to matrices using Rectangulararrays.
- Find the sum of all the elements present in a jagged array of 3 innerarrays.
- Write a program to reverse a given string usingC#.
- Using Try, Catch and Finally blocks write a program in C# todemonstrate
- errorhandling.
- Design a simple calculator using Switch Statement inC#.
- Demonstrate Use of Virtual and override key words in C# with asimple
- program
- Implement linked lists in C# using the existing collections namespace.
- Write a program to demonstrate abstract class and abstract methods inC#.
- Write a program in C# to build a class which implements an interface which alreadyexists.
- Write a program to illustrate the use of different properties inC#.
- Demonstrate arrays of interface types with a C#program.

FORMAT OFINTERNAL QUESTION PAPER

1st Internal Examination (2020)

Course: Semester:
Subject: Course Code:
Max. Marks: 40 Max. Time: 2 Hours

Instructions (if any):- Use of calculator for subjects like Financial Mgt. Operation etc. allowed if required. (Scientific calculator is not allowed).
Use of unfair means will lead to cancellation of paper followed by disciplinary action.
Question No. 1 is compulsory. Attempt any two questions from Q2 to Q5.
Attempt any two question from section 2.

Section 1

(Theoretical Concept and Practical/Application oriented)

Answer in 400 words. Each question carry 06 marks.

Q. 1
Q. 2
Q.3
Q. 4
Q.5 Write Short Note on any two. Answer in 300 words. Each carry 03 marks.
a)
b)
c)

Section 2

(Analytical Question / Case Study / Essay Type Question to test analytical and Comprehensive Skills)

Answer in 800 words. Attempt any 2 questions.Each question carry 11 marks

Q6.
Q7.
Q8.

PREVIOUS YEAR INTERNAL QUESTION PAPERS

- **Old syllabus Internal Question papers (for reference)**

Bharati Vidyapeeth Deemed University,
Institute of Management and Research (BVIMR),
New Delhi 1st Internal Examination
(August, 2016)

Course– BCA Semester –V

Subject –C#Programming Course Code – 503

Max.Marks: 40 Max. Time: 2 Hours

Instruction: 1. Read all the questionscarefully

Q.1 Attempt any five questions. Answer in50 words [5 x2]

a) What are regular expressions? When do we usethem?
b) What is a verbatim string? Where do we useit?
c) What is Methodhiding?
d) What is the difference between value type and referencetype?
e) Write an example where foreach statement isused.
f) What are sealedclasses?
g) Briefly explain any two stringmethods.
h) What is the difference between 'protected' and 'protectedinternal'?

Q.2 Attempt any two questions. Answer in200 words [2 x 5]

a Explain the features of C#programming.
b Discuss the implementation of OOPS concepts in C# programming.
c Explain partial methods in C# with the help of an example.

Attempt any two questions. Answer in200 words [2 x5]

a) Explain jagged arrays with the help of anexample.
b) Write a program in C# to show the effective use ofIndexers.
c) Why we use properties in C#. Explain with anexample.

Attempt any one. Answer in600 words **[**10 x1]

a) Define the term "Constructor". Explain different types of constructors inC#.
b) Define inheritance. Explain its types along with suitableexamples.

Bharati Vidyapeeth Deemed University
Institute of Management and Research (BVIMR)
New Delhi 2nd Internal Examination
(October, 2016)

Course– BCA Semester –V
Subject –C#Programming Course Code –503
Max.Marks: 40 Max. Time: 2 Hours

Q.1 Attempt any five questions. Answer in50 words [5 x2]

a) What is explicit interface implementation? When is it used?
b) What are object initializer and collectioninitializer?
c) Why is it necessary to overload anoperator?
d) What is an operator method? Describe its syntax.
e) What are the characteristics of a multicastdelegate?
f) When do you consider it necessary to define your own exceptionclasses?
g) Define general catch handler? Write itssyntax.
h) What is the importance of delegates and events in designinglarge applications?

Q.2 Attempt any two questions. Answer in200 words [2 x5]

a) What is a delegate? Give an example to show the use of adelegate.
b) What are extension methods? Give an example where extension methods areused.
c) Define an event. Give an example to show the implementation of anevent handler.

Attempt any two questions. Answer in200words [2 x5]

a) Write a C# program to display the Student Details using Select ClauseLINQ.
b) Write a program in C# to show an anonymoustype.
c) Define an Exception called "NoMatchException" that is thrown when a string is not equal to "XYZ". Write a C# Program that uses thisexception.

Attempt any one. Answer in600 words [10 x1]

a)Explain LINQ indetail.
b)Describe various forms of implementing interfaces. Give examples of C# code for each use.

Bharati Vidyapeeth Deemed University,
Institute of Management and Research (BVIMR), New Delhi
First Internal Examination

Subject: C# Programming Course: BCA
Subject Code :503
Max. Marks: 40 Max. Time: 2 Hours

Question No.1 is Compulsory. Attempt any two questions from Q2 to Q5.
Attempt any two question from section 2.

Section 1

Answer in 400 words. Each Questions Carry 06 Marks

Q. 1 Discuss data types in C# with suitable example.
Q. 2 What is Boxing and unboxing in C# explain with example.
Q. 3 Why C# is termed as pure object-oriented language.
Q. 4 What is the use of "base" keyword in C#? Explain with suitable example.
Q. 5 Write short notes on any two of the following
a)Relational operators
b) "var" Variable
c) CLR

Section 2

Answer in 800 words. Attempt any 2. Each Questions Carry 11 Marks

Q. 6Explain Inheritance with the help of example.
Q. 7Explain in detail the features of C# programming.
Q. 8write short notes on any two of the following
a) Method hiding
b) Jagged Array
c) "static" keyword

First Internal Examination (2019)

Subject: C# Programming — Course: BCA
Subject Code :503 — Sem : Vth
Max. Marks: 40 — Max. Time: 2 Hours

Instructions (if any): - Use of calculator for subjects like Financial Mgt. Operation etc. allowed if required. (Scientific calculator is not allowed).
Use of unfair means will lead to cancellation of paper followed by disciplinary action.

Question No.1 is Compulsory. Attempt any two questions from Q2 to Q5. Attempt any two question from section 2.

Section 1

Answer in 400 words. Each Questions Carry 06 Marks

Q. 1 Explain CLR.
Q. 2 Which keyword is used for calling supper class constructor, explain with example.
Q. 3 Why C# is termed as pure object-oriented language.
Q. 4 Explain conditional operator, with example.
Q. 5 write short notes on any two of the following
a) Boxing
b) Unboxing
c) CLI

Section 2

Answer in 800 words. Attempt any 2. Each Questions Carry 11 Marks

Q. 6 What is Inheritance, Does C# support multipath inheritance, support you answer with suitable example.
Q. 7 Discuss different data type in C# with suitable example.
Q. 8 write short notes on any two of the following
a) this
b) overriding
c) var

****

Second Internal Examination (2019)

Subject: C# Programming — Course: BCA
Subject Code :503 — Sem : Vth
Max. Marks: 40 — Max. Time: 2 Hours

Instructions (if any): - Use of calculator for subjects like Financial Mgt. Operation etc. allowed if required. (Scientific calculator is not allowed).
Use of unfair means will lead to cancellation of paper followed by disciplinary action.

Question No.1 is Compulsory. Attempt any two questions from Q2 to Q5. Attempt any two question from section 2.

Section 1

Answer in 400 words. Each Questions Carry 06 Marks

Q. 1 Explain function Overloading with example.
Q. 2 Write the difference between classes and Interface with example.
Q. 3 What is operator overloading explain with example, also write the list of no-overloadable operator.
Q. 4 Explain object and collection initializer.
Q. 5 write short notes on any two of the following with example
a) sealed classes
b) Virtual Function
c) Auto Implement property

Section 2

Answer in 800 words. Attempt any 2. Each Questions Carry 11 Marks

Q. 6 What Exception handling, support you answer with suitable example.
Q. 7 Explain Indexer with suitable example.
Q. 8 write short notes on any two of the following with example
a) partial classes and methods
b) Linq
c) Delegate

****

Previous Year University Question

- **Old syllabus University Question papers (for reference)**

B.C.A. SEM–V (2014 COURSE) CBCS : SUMMER - 2018
SUBJECT: C# PROGRAMMING

Day : **Wednesday**
Date : **02/05/2018**

Time : **02.00 PM TO 05.00 PM**
Max. Marks : 100.

S-2018-1712

N.B.:

1) Attempt any **FOUR** questions from Section-I and any **TWO** questions from Section-II.
2) Answers to both the sections should be written in **SEPARATE** answer books.
3) Figures to the **RIGHT** indicate full marks.

SECTION-I

Q.1 Explain in detail the features of C# programming. **(15)**

Q.2 Explain in detail the following C# methods : **(15)**
(a) Read () (b) ReadLine () (c) WriteLine ()

Q.3 How to declare interfaces? Explain in detail the implementation of interfaces with suitable example. **(15)**

Q.4 Explain in detail the various C# mechanisms to handle programming errors. **(15)**

Q.5 Describe 'Constructor' and their types in brief. **(15)**

Q.6 Explain in detail the following access modifiers with example: **(15)**
a) Public
b) Private
c) Protected

Q.7 Write short notes on any **THREE** of the following: **(15)**
a) LINQ
b) Multicast delegates
c) 'Var' variable
d) 'static' Keyword.

SECTION-II

Q.8 Write a C# program to demonstrate difference between sealed class and abstract class. **(20)**

Q.9 Write C# program to transpose given matrjx elements. Make necessary assumption for order of matrix. **(20)**
(Transpose operation: interchanging row values and column values.)

Q.10 Write a C# program with use of methods to do the following: **(20)**
a) Accept a number from the user.
b) Count number of 1's in the input.
c) Find out the sum of odd position digits and even position digits separately.
d) Reverse of the number.
Hint: (8140216 ∴ No. of 1's is 2; 8 + 4 + 2 + 6 = 20 and 1 + 0 + 1 = 2: Reverse of input 8140216 is 6120418)

* * *

B.C.A. SEM–V (2014 Course) CBCS : SUMMER - 2019
SUBJECT : C# PROGRAMMING

Day : Monday
Date : 22/04/2019
Time : 02.00 PM TO 05.00 PM
Max. Marks : 100

S-2019-2077

N.B.

1) Attempt **ANY FOUR** questions from Section – I and **ANY TWO** questions from Section – II.
2) Answers to both the sections should be written in SAME answer book.
3) Figures to the right indicate **FULL** marks.

SECTION – I

Q.1 "C # is very powerful language in terms of security and other features". Justify. **[15]**

Q.2 Discuss the importance of following keywords: **[15]**
a) 'var' **b)** 'this' **c)** 'ref'

Q.3 Explain in detail looping structures mentioned below with suitable example: **[15]**
a) do.....while ( )
b) while ( )
c) for ( )

Q.4 Illustrate concept of interface with suitable example. **[15]**

Q.5 Differentiate the following: **[15]**
a) Value Type and Reference Type
b) Abstract Class and Sealed Class
c) Method Overloading and Method Overriding

Q.6 How generic classes do helps in coding? Illustrate with suitable example. **[15]**

Q.7 Write short notes on **ANY THREE** of the following: **[15]**
a) Delegates
b) Abstract class
c) Constructor and their types
d) Indexers

SECTION – II

Q.8 **a)** Write C# program to generate divide by zero exception and handle with try and catch block. **[10]**

b) Write C# program to demonstrate jagged array (make necessary assumption). **[10]**

Q.9 Write a C# program for 'BCABank' wherein 'BCABankTran' is a class, contains methods withdrawal(), deposit(), balance_enquiry() to operate account types such as saving and current with the help of method overloading. **[20]**

Q.10 **a)** Write C# program to demonstrate the implementation of Inheritance using virtual function. **[10]**

b) Write C# program to find prime numbers between 10.....100. **[10]**

* * * *

AMOOR- V (2014 Course) CBCS : SUMMER - 2017

Subject : C# Programming

Day : Saturday
Date : 15/04/2017

35641

Time : 02.00 PM TO 05.00 PM
Max Marks : 100 Total Pages : 1

N.B.:

1) Attempt **ANY FOUR** questions from Section – I and attempt **ANY TWO** questions from Section – II.
2) Answers to both the sections should be written in the **SEPARATE** answer books.
3) Figures to the right indicate **FULL** marks

SECTION – I

Q.1 Write note on C # Programming with the help of features, compilation and execution. **[15]**

Q.2 Discuss concept of 'Operator Overloading' with suitable example. **[15]**

Q.3 Discuss implementation of OOPS concepts in C # Programming. **[15]**

Q.4 Define term 'Constructor' and discuss their types with applicability. **[15]**

Q.5 Discuss: 'Inheritance' along with its type and suitable example. **[15]**

Q.6 Write comparative note on: **[15]**
a) Abstract class and Sealed class
b) Value type and Reference type

Q.7 Write note on **ANY THREE** of the following: **[15]**
a) C # Operators
b) Jagged Array
c) 'this' reference
d) Loops in C #

SECTION – II

Q.8 Using Try, Catch and Finally block. Write a C# program to demonstrate various types of error. **[20]**

Q.9 a) Write a C # program to sort elements stored in an array. **[10]**

b) Write a C # Program to find roots of Quadratic equation. **[10]**

Q.10 Write a C # Program to demonstrate implementation of 'Interface' *(Make necessary assumption).* **[20]**

* * * *

1

AMOOR- V (2014 Course) CBCS : SUMMER - 2017

Subject : C# Programming

Day : Saturday
Date : 15/04/2017

35641

Time : 02.00 PM TO 05.00 PM
Max Marks : 100 Total Pages : 1

N.B.:

1) Attempt **ANY FOUR** questions from Section – I and attempt **ANY TWO** questions from Section – II.
2) Answers to both the sections should be written in the **SEPARATE** answer books.
3) Figures to the right indicate **FULL** marks .

SECTION – I

Q.1 Write note on C # Programming with the help of features, compilation and execution. **[15]**

Q.2 Discuss concept of 'Operator Overloading' with suitable example. **[15]**

Q.3 Discuss implementation of OOPS concepts in C # Programming. **[15]**

Q.4 Define term 'Constructor' and discuss their types with applicability. **[15]**

Q.5 Discuss: 'Inheritance' along with its type and suitable example. **[15]**

Q.6 Write comparative note on: **[15]**
a) Abstract class and Sealed class
b) Value type and Reference type

Q.7 Write note on **ANY THREE** of the following: **[15]**
a) C # Operators
b) Jagged Array
c) 'this' reference
d) Loops in C #

SECTION – II

Q.8 Using Try, Catch and Finally block. Write a C# program to demonstrate various types of error. **[20]**

Q.9 **a)** Write a C # program to sort elements stored in an array. **[10]**

b) Write a C # Program to find roots of Quadratic equation. **[10]**

Q.10 Write a C # Program to demonstrate implementation of 'Interface' *(Make necessary assumption).* **[20]**

* * * *

CBCS : WINTER - 2016

Subject : C# Programming

Day : Friday
Date : 11/11/2016

32286

Time : 02.00 PM TO 05.00 PM
Max Marks : 100 Total Pages : 1

N.B.:
1) Attempt **ANY FOUR** questions from Section – I and attempt **ANY TWO** questions from Section – II.
2) Answers to both the sections should be written in the **SEPARATE** answer books.
3) Figures to the right indicate **FULL** marks.

SECTION – I

Q.1 Discuss features of C # programming under Dot Net Framework. **[15]**

Q.2 Define term 'Constructor' and discuss their types with applicability. **[15]**

Q.3 Write a comparative note on: **[15]**
a) Partial class and Sealed class
b) Value type and Reference type

Q.4 Discuss term 'Inheritance' along with its type and suitable example. **[15]**

Q.5 Discuss implementation of OOPS concepts in C # programming. **[15]**

Q.6 Write a note on 'Operator Overloading' with suitable example. **[15]**

Q.7 Write note on **ANY THREE** of the following: **[15]**
a) Loops in C #
b) Jagged Array
c) Indexers
d) Abstract class

SECTION – II

Q.8 Write a C # Program to demonstrate implementation of "Interface" *(Make necessary assumption).* **[20]**

Q.9 Using Try, Catch and Finally block. Write a program in C # to demonstrate various types of error. **[20]**

Q.10 a) Write a C # Program to sort 10 Integers/elements with the help of a method *Sort ()*. **[10]**

b) Write C # Program to check whether a number is Palindrome or not. **[10]**

* * * *

1

Solution of this paper

Subject – C# Programming

Date – 11-11-16

Q1. Discuss features of C# programming under dotnet framework. 15

A.1 C# is object oriented programming language. It provides a lot of **features** that are

given below :

1) Simple

C# is a simple language in the sense that it provides structured approach (to break the problem into parts), rich set of library functions, data types etc.

2) Modern ProgrammingLanguage

C# programming is based upon the current trend and it is very powerful and simple for building scalable, interoperable and robust applications.

3) ObjectOriented

C# is object oriented programming language. OOPs makes development and maintenance easier where as in Procedure-oriented programming language it is not easy to manage if code grows as project size grow.

4) TypeSafe

C# type safe code can only access the memory location that it has permission to execute. Therefore it improves a security of the program.

5) Interoperability

Interoperability process enables the C# programs to do almost anything that a native C++ application can do.

6) Scalable and Updateable

C# is automatic scalable and updateable programming language. For updating our application we delete the old files and update them with new ones.

7) ComponentOriented

C# is component oriented programming language. It is the predominant software development methodology used to develop more robust and highly scalable applications.

8) Structured ProgrammingLanguage

C# is a structured programming language in the sense that we can break the program into parts using functions. So, it is easy to understand and modify.

9) Rich Library

C# provides a lot of inbuilt functions that makes the development fast.

10) FastSpeed

The compilation and execution time of C# language is fast.

11) Automatic GarbageCollection

Garbage collector manages allocation and reclaiming of memory. GC (Garbage collector) makes a trip to the heap and collects all objects that are no longer used by the application and then makes them free from memory.

12) Properties

Properties are named members of classes, structures, and interfaces. Member variables or methods in a class or structures are called **Fields**. Properties are an extension of fields and are accessed using the same syntax. They use **accessors** through which the values of the private fields can be read, written or manipulated.

13) Events

Events are user actions such as key press, clicks, mouse movements, etc., or some occurrence such as system generated notifications. Applications need to respond to events when they occur. For example, interrupts. Events are used for inter-process communication.

14) Delegates

A **delegate** is a reference type variable that holds the reference to a method. The reference can be changed at runtime.

15) Indexers

An **indexer** allows an object to be indexed such as an array. When you define an indexer for a class, this class behaves similar to a **virtual array**. You can then access the instance of this class using the array access operator ([ ]).

16) Easy to useGenerics

Generics allow you to delay the specification of the data type of programming elements in a class or a method, until it is actually used in theprogram.

17) LINQ

LINQ (Language Integrated Query) allows writing queries even without the knowledge of query languages like SQL, XML etc.

18) RegularExpressions

A **regular expression** is a pattern that could be matched against an input text. The .Net framework provides a regular expression engine that allows such matching.

19) Namespaces

A **namespace** is designed for providing a way to keep one set of names separate from another. The class names declared in one namespace does not conflict with the same class names declared in another.

Q.2 Define term "Constructor" and discuss their types with applicability. 15

In C#, constructor is a special method which is invoked automatically at the time of object creation. It is used to initialize the data members of new object generally. The constructor in C# has the same name as class orstruct.

Types of constructors:

a) Default Constructor - A constructor which has no argument is known as default constructor. It is invoked at the time of creatingobject.

Example of default constructor

```
using System;
  public class Employee
   {
     public Employee()
     {
```

```
        Console.WriteLine("Default Constructor Invoked");
    }
}
class Test{
    public static void Main(string[] args)
    {
        Employee e1 = new Employee();
        Employee e2 = new Employee();
    }
}
```

Output:

```
Default Constructor Invoked
Default Constructor Invoked
```

b) Parameterized constructor - A constructor which has parameters is called parameterized constructor. It is used to provide different values to distinctobjects.

Example of Parameterized constructor

```
using System;
public class Employee
{
    public int id;
    public String name;
    public floatsalary;
    public Employee(int i, String n,floats)
    {
        id = i;
        name = n;
        salary = s;
    }
    public void display()
    {
        Console.WriteLine(id + " " + name+" "+salary);
    }
}
class TestEmployee{
```

```
 public static void Main(string[] args)
  {
     Employee e1 = new Employee(101, "Ram", 890000f);
     Employee e2 = new Employee(102, "Sham", 490000f);
     e1.display();
     e2.display();

  }
}
```

Output:

```
101 Ram 890000
102 Sham 490000
```

c) Copy Constructor - The **constructor** which creates an object by copying variables from another object is called a **copy constructor**. The purpose of a **copy constructor** is to initialize a new instance to the values of an existinginstance.

Example of Copy Constructor

```
using System;
namespace ConsoleApplication3
{
class Sample
{
public string param1, param2;
public Sample(string x, string y)
{
param1 = x;
param2 = y;
}
publicSample(Sampleobj)       // CopyConstructor
{
param1 =
obj.param1; param2
= obj.param2;
}
}
class Program
{
static void Main(string[] args)
{
```

```
Sample obj1=new Sample(obj); // Here obj details will copied to obj1
Console.WriteLine(obj1.param1 +" to " + obj1.param2);
Console.ReadLine();
}
}
}
```

Output

```
Welcome to Aspdotnet-Suresh
```

d) Static Constructor - C# static constructor is used to initialize static fields. It can also be used to perform any action that is to be performed only once. It is invoked automatically before first instance is created or any static member isreferenced.

Example of Static Constructor

```
using System;
  public class Account
   {
      public int id;
      public String name;
      public static float rateOfInterest;
      public Account(int id, String name)
      {
         this.id = id;
         this.name = name;
      }
      static Account()
      {
         rateOfInterest = 9.5f;
      }
      public void display()
      {
         Console.WriteLine(id + " " + name+" "+rateOfInterest);
      }
  }
  class TestEmployee{
     public static void Main(string[] args)
      {
         Account a1 = new Account(101, "Ram");
```

```
        Account a2 = new Account(102, "Sham");
        a1.display();
        a2.display();

    }
}
```

Output:

```
101 Ram 9.5
102 Sham 9.5
```

Write a Comparitive noteon : ***15***

a) Partial class and Sealedclass

b) Value type and referencetype

A.3

a) PartialClass

Each class in C# resides in a separate physical file with a .cs extension. C# provides the ability to have a single class implementation in multiple .cs files using the ***partial*** modifier keyword. The *partial* modifier can be applied to a class, method, interface orstructure.

Partial Class Requirements:

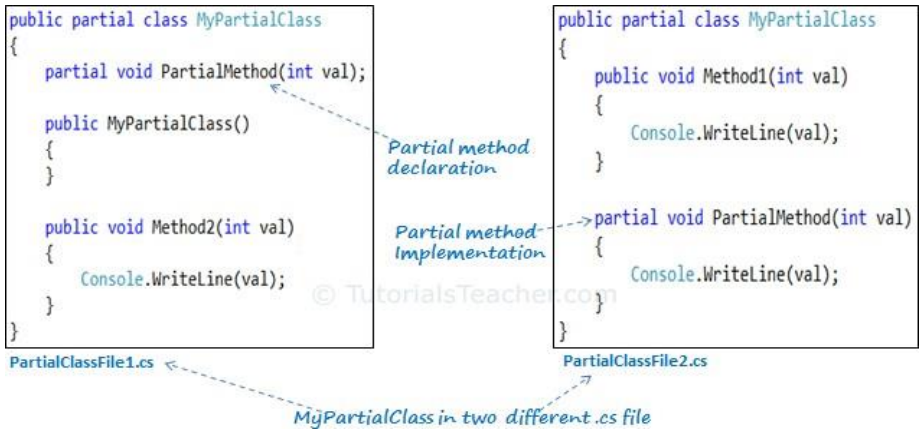
- All the partial class definitions must be in the same assembly andnamespace.
- All the parts must have the same accessibility like public or private,etc.
- If any part is declared abstract, sealed or base type then the whole class is declared of the sametype.
- Different parts can have different base types and so the final class will inherit all the base types.
- The Partial modifier can only appear immediately before the keywords class, struct, orinterface.
- Nested partial types areallowed.

Advantages of Partial Class

- Multiple developers can work simultaneously with a single class in separatefiles.
- When working with automatically generated source, code can be added to the class without having to recreate the source file. For example, Visual Studio separates HTML code for the UI and server side code into two separate files: .aspx and .csfiles.

Partial Methods

A partial class or struct may contain partial methods. A partial method must be declared in one of the partial classes. A partial method may or may not have an implementation. If the partial method doesn't have an implementation in any part then the compiler will not generate that method in the final class.



Sealed class

C# sealed keyword applies restrictions on the class and method. If you create a sealed class, it cannot be derived. If you create a sealed method, it cannot be overridden.

C# sealed class cannot be derived by any class.

```
using System;
sealed public class Animal{
  public void eat() { Console.WriteLine("eating..."); }
}
public class Dog: Animal
{
  public void bark() { Console.WriteLine("barking..."); }
}
public class TestSealed
{
  public static void Main()
  {
    Dog d = new Dog();
    d.eat();
    d.bark();
```

```
    }
}
```

C# Sealed method

The sealed method in C# cannot be overridden further. It must be used with override keyword in method.

```
using System;
public class Animal{
   public virtual void eat() { Console.WriteLine("eating..."); }
   public virtual void run() { Console.WriteLine("running..."); }

}
public class Dog: Animal
{
   public override void eat() { Console.WriteLine("eating bread..."); }
   public sealed override void run() {
   Console.WriteLine("running very fast...");
   }
}
public class BabyDog : Dog
{
   public override void eat() { Console.WriteLine("eating biscuits..."); }
   public override void run() { Console.WriteLine("running slowly..."); }
}
public class TestSealed
{
   public static void Main()
   {
      BabyDog d = new BabyDog();
      d.eat();
      d.run();
   }
}
```

b) Value type and Referencetype

- A data type is a value type if it holds a data value within its own memory space. It means variables of these data types directly contain their values. For example, consider integer variable int i = 100;Thesystem stores 100 in the memory space allocated for the variable 'i'. The following data types are all of value type: bool, byte, char, decimal, double, enum, float, intetc.

Example: Value Type

```
static void ChangeValue(int x)
{
   x = 200;

   Console.WriteLine(x);
}

static void Main(string[] args)
{
   int i = 100;

   Console.WriteLine(i);

   ChangeValue(i);

   Console.WriteLine(i);
}
```

Reference Type

Unlike value types, a reference type doesn't store its value directly. Instead, it stores the address where the value is being stored. In other words, a reference type contains a pointer to another memory location that holds the data.

For example, consider following string variable:

```
string s = "Hello World!!";
```

The following data types are of reference type:

- String
- All arrays, even if their elements are value types
- Class
- Delegates

Example: Reference Type Variable

```
static void ChangeReferenceType(Student std2)
{
   std2.StudentName = "Steve";
}
```

```
static void Main(string[]args)
{
   Student std1 = new Student();
   std1.StudentName ="Bill";

   ChangeReferenceType(std1);

   Console.WriteLine(std1.StudentName);
}
```

Q4. Discuss term 'Inheritance'along with its types and examples.

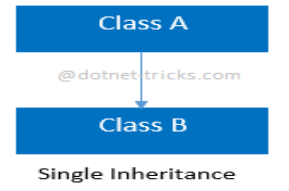
A4. Inheritance, in C#, is the ability to create a class that inherits attributes and behaviors from an existing class. The newly created class is the derived (or child) class and the existing class is the base (or parent)class.

Different Types of Inheritance

OOPs supports the six types of inheritance as given below-

1. Single inheritance

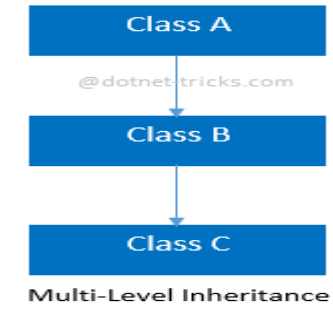
In this inheritance, a derived class is created from a single base class.




```
//Base Class
class A
{
publicvoidfooA()
{
//TO DO:
}
}

//Derived Class
class B : A
{
publicvoidfooB()
{
//TO DO:
}
}
```

2. Multi-level inheritance

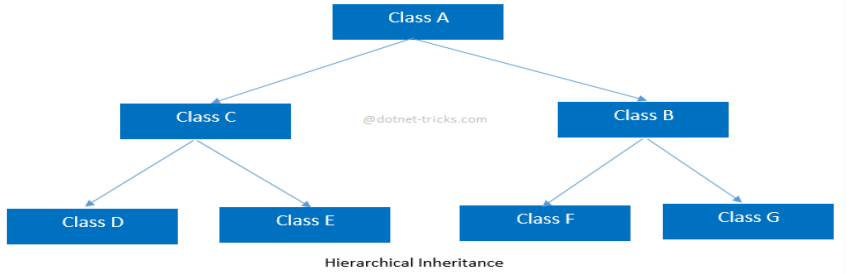
In this inheritance, a derived class is created from another derived class.




```
//Base Class
class A
{
publicvoidfooA()
{
//TO DO:
}
}

//Derived Class
class B : A
{
publicvoidfooB()
{
//TO DO:
}
}

//Derived Class
class C : B
{
publicvoidfooC()
{
//TO DO:
}}
```

Hierarchical inheritance

In this inheritance, more than one derived classes are created from a singlebase.



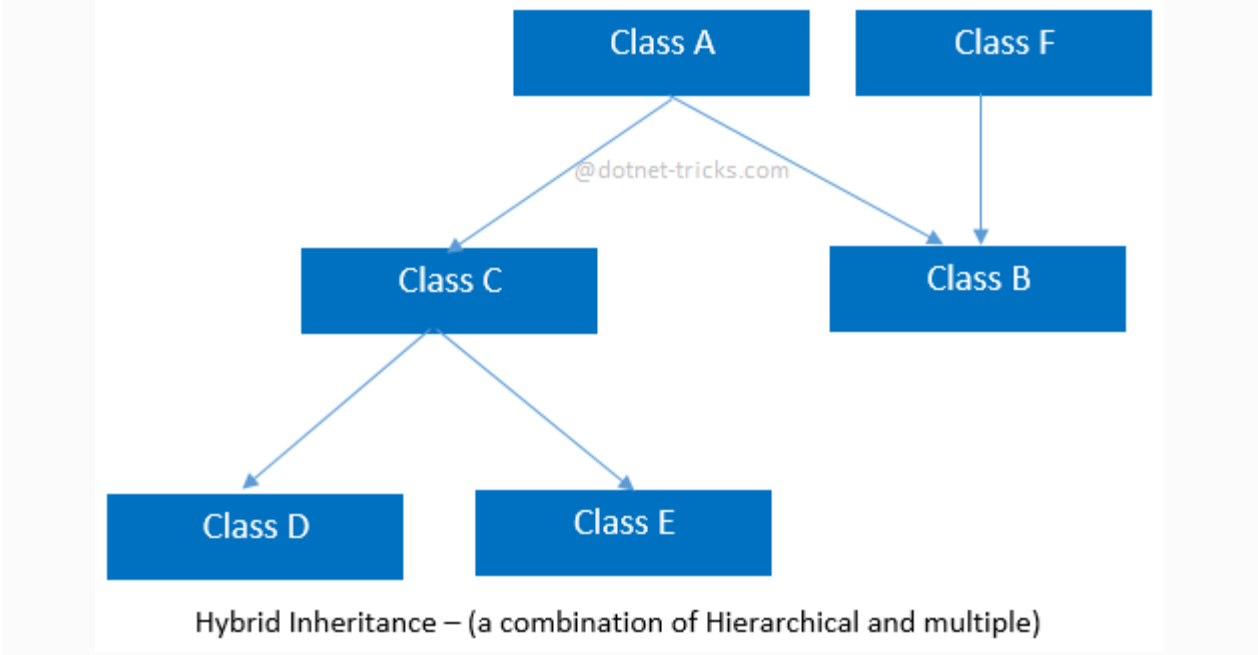
Hierarchical Inheritance

```
//Base Class
class A
{
publicvoidfooA()

{
//TO DO:
}
}

//Derived Class
class B : A
{
publicvoidfooB()
{
//TO DO:
}
}

//Derived Class
class C : A
{
publicvoidfooC()
{
//TO DO:
}
}

//Derived Class
class D : C
{
publicvoidfooD()
{
//TO DO:
}
}

//Derived Class
```

```
class E : C
{
publicvoidfooE()
{
//TO DO:
}
}

//Derived Class
class F : B
{
publicvoidfooF()
{
//TO DO:
}
}

//Derived Class
class G :B
{
publicvoidfooG()
{
//TO DO:
}
}
```

Hybrid inheritance

This is combination of more than one inheritance. Hence, it may be a combination of Multilevel and Multiple inheritance or Hierarchical and Multilevel inheritance or Hierarchical and Multipath inheritance or Hierarchical, Multilevel and Multiple inheritance.
Since .NET Languages like C#, F# etc. does not support multiple and multipath inheritance. Hence hybrid inheritance with a combination of multiple or multipath inheritance is not supported by .NET Languages.



Hybrid Inheritance – (a combination of Hierarchical and multiple)

```
//Base Class
class A
{
publicvoidfooA()
{
//TO DO:
}
}

//Base Class
class F
{
publicvoidfooF()
{
//TO DO:
}
}

//Derived Class
class B : A, F
{
publicvoidfooB()
{
//TO DO:
}
}

//Derived Class
```

```
class C : A
{
publicvoidfooC()
{
//TO DO:
}
}

//Derived Class
class D : C
{
publicvoidfooD()
{
//TO DO:
}
}

//Derived Class
class E : C
{
publicvoidfooE()
{
//TO DO:
}
}
```

Q5. Discuss implementation of OOPS concepts in C#.

A5. Object oriented programming (OOP) is a programming structure where programs are organized around objects as opposed to action and logic.

OOP has the following important features.

Class

A class is the core of any modern Object Oriented Programming language such as C#.

In OOP languages it is mandatory to create a class for representing data.

A class is a blueprint of an object that contains variables for storing data and functions to perform operations on the data.

A class will not occupy any memory space and hence it is only a logical representation of data.

To create a class, you simply use the keyword "class" followed by the class name:

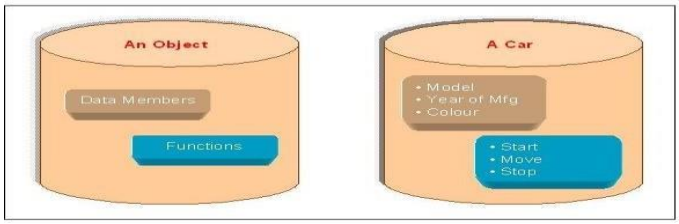
```
class Employee
{


}
```

Object

Objects are the basic run-time entities of an object oriented system. They may represent a person, a place or any item that the program must handle.

"An object is a software bundle of related variable and methods."

"An object is an instance of a class"



A class will not occupy any memory space. Hence to work with the data represented by the class you must create a variable for the class, that is called an object.

When an object is created using the new operator, memory is allocated for the class in the heap, the object is called an instance and its starting address will be stored in the object in stack memory.

When an object is created without the new operator, memory will not be allocated in the heap, in other words an instance will not be created and the object in the stack contains the value **null**.

When an object contains null, then it is not possible to access the members of the class using that object.

```
class Employee
{


}
```

Syntax to create an object of class Employee:

```
Employee objEmp = new Employee();
```

All the programming languages supporting Object Oriented Programming will be supporting these three main concepts,

1. Encapsulation
2. Inheritance
3. Polymorphism

Abstraction

Abstraction is "To represent the essential feature without representing the background details."

Abstraction lets you focus on what the object does instead of how it does it.

Abstraction provides you a generalized view of your classes or objects by providing relevant information.

Abstraction is the process of hiding the working style of an object, and showing the information of an object in an understandable manner.

Real-world Example of Abstraction

Suppose you have an object Mobile Phone.

Suppose you have 3 mobile phones as in the following:

Nokia 1400 (Features: Calling, SMS)
Nokia 2700 (Features: Calling, SMS, FM Radio, MP3, Camera)
Black Berry (Features:Calling, SMS, FM Radio, MP3, Camera, Video Recording, Reading E-mails)

Abstract information (necessary and common information) for the object "Mobile Phone" is that it makes a call to any number and can send SMS.

So that, for a mobile phone object you will have the abstract class as in the following,

```
abstract class MobilePhone {

    public void Calling();

    public void SendSMS();

}

public class Nokia1400: MobilePhone {}
```

```
public class Nokia2700: MobilePhone {
    public void FMRadio();
    public void MP3();
    public void Camera();
}
public class BlackBerry: MobilePhone {
    public void FMRadio();
    public void MP3();
    public void Camera();
    public void Recording();
    public void ReadAndSendEmails();
}
```

Abstraction means putting all the variables and methods in a class that are necessary.

For example: Abstract class and abstract method.

Abstraction is a common thing.

Example

If somebody in your collage tells you to fill in an application form, you will provide your details, like name, address, date of birth, which semester, percentage you have etcetera.

If some doctor gives you an application to fill in the details, you will provide the details, like name, address, date of birth, blood group, height and weight.

See in the preceding example what is in common?

Age, name and address, so you can create a class that consists of the common data. That is called an abstract class.

That class is not complete and it can be inherited by other classes.

Encapsulation

Wrapping up a data member and a method together into a single unit (in other words class) is called Encapsulation.

Encapsulation is like enclosing in a capsule. That is enclosing the related operations and data related to an object into that object.

Encapsulation is like your bag in which you can keep your pen, book etcetera. It means this is the property of encapsulating members and functions.

Encapsulation means hiding the internal details of an object, in other words how an object does something.

Encapsulation prevents clients from seeing its inside view, where the behaviour of the abstraction is implemented.

Encapsulation is a technique used to protect the information in an object from another object.

Hide the data for security such as making the variables private, and expose the property to access the private data that will be public.

So, when you access the property you can validate the data and set it.

Example

```
class Demo {
   private int _mark;
   public int Mark {
      get {
         return _mark;
      }
```

```
        set {
            if (_mark > 0) _mark = value;
            else _mark = 0;
        }
    }
}
```

Real-world Example of Encapsulation

Let's use as an example Mobile Phones and Mobile Phone Manufacturers.
Suppose you are a Mobile Phone Manufacturer and you have designed and developed a Mobile Phone design (a class). Now by using machinery you are manufacturing Mobile Phones (objects) for selling, when you sell your Mobile Phone the user only learns how to use the Mobile Phone but not how the Mobile Phone works.

This means that you are creating the class with functions and by with objects (capsules) of which you are making available the functionality of your class by that object and without the interference in the original class.

Example

TV operation

It is encapsulated with a cover and we can operate it with a remote and there is no need to open the TV to change the channel.
Here everything is private except the remote, so that anyone can access the remote to operate and change the things in the TV.

Inheritance

When a class includes a property of another class it is known as inheritance.

Inheritance is a process of object reusability.

For example, a child includes the properties of its parents.

```

```

```
    }
    public void print() {
        Console.WriteLine("I'm a Parent Class.");
    }
}
public class ChildClass: ParentClass {
    public ChildClass() {
        Console.WriteLine("Child Constructor.");
    }
    public static void Main() {
        ChildClass child = new ChildClass();
        child.print();
    }
}
```

Output

ParentConstructor.
ChildConstructor.
I'm a ParentClass.

Polymorphism

Polymorphism means one name, many forms.

One function behaves in different forms.

In other words, "Many forms of a single object is called Polymorphism."

Real-world Example of Polymorphism

Example

A teacher behaves students.

A teacher behaves his/her seniors.

Here teacher is an object but the attitude is different in different situations.

Example

A person behaves the son in a house at the same time that the person behaves an employee in an office.

Example

Your mobile phone, one name but many forms:

- As phone
- Ascamera
- As mp3player
- Asradio

Q6. Write a note on Operator overloading with example.

A6. Overloaded operators are functions with special names the keyword **operator**followed by the symbol for the operator being defined. similar to any other function, an overloaded operator has a return type and a parameter list.'

There are two types:

1. Unaryoperator
2. Binaryoperator

Overloading Unary Operators

The return type can be of any type except void for unary operators like !, ~, + and dot (.) but the return type must be the type of 'Type' for – and ++ operators and must be a bool type for true as well as false operators. But do remember that the true and false operators can be overloaded as pairs only. The compilation error arises if a class declares one of these operators without declaring the other.

The following syntax shows the use of Unary operator –

```
operator (object);

here, operator is a symbol that denotes a unary operator.

operator a;
```

```
using System;
namespace Calculator {
```

```
class Calculator {

   public int number1 , number2;
   public Calculator(int num1 , int num2)
   {
      number1 = num1;
      number2 = num2;
   }

// Function to perform operation
// By changing sign of integers
public static Calculator operator -(Calculator c1)
{
   c1.number1 = -c1.number1;
   c1.number2 = -c1.number2;
   return c1;
}

// Function to print the numbers
public void Print()
{
   Console.WriteLine ("Number1 = " + number1);
   Console.WriteLine ("Number2 = " + number2);
}
}

class EntryPoint
{

   // Driver Code
   static void Main(String []args)
   {

      // using overloaded - operator
      // with the class object
      Calculator calc = new Calculator(15, -25);

      calc =-calc;

      // To display the result
      calc.Print();
   }
}
}
```

Overloading Binary Operators

Binary Operators will work with two Operands. Examples of binary operators include the **Arithmetic Operators**(+, -, *, /, %), Arithmetic Assignment operators (+=, -+, *=, /+, %=) and **Relational Operators**etc. Overloading a binary operator is similar to overloading a unary operator, except that a binary operator requires an additional parameter. **Syntax:**

```
operator operator (object1, object2);

Here, second "operator" is a symbol that

denotes a binary operator.

operator + (a, b);
```

```
using System;
namespace BinaryOverload{

class Calculator {

   public int number =0;

   // no-argument constructor
   public Calculator(){}


   // parameterized constructor
   public Calculator(int n)
   {
      number = n;
   }

   // Overloading of Binary "+" operator
   public static Calculator operator + (Calculator Calc1,
                                 Calculator Calc2)
   {
      Calculator Calc3 = new Calculator(0);
      Calc3.number = Calc2.number + Calc1.number;
      return Calc3;
   }

   // function to display result
   public void display()
   {
      Console.WriteLine("{0}", number);
   }
}
```

```
class CalNum {

   // Driver Code
   static void Main(string[] args)
   {

      Calculator num1 = new Calculator(200);
      Calculator num2 = new Calculator(40);
      Calculator num3 = new Calculator();


      num3 = num1 + num2;

      num1.display(); // Displays 200

      num2.display(); // Displays 40

      num3.display(); // Displays 240

   }
}
}
```

Q7. Write note on the following

a)Loops inC#

looping - the ability to repeat a block of code Xtimes.

Types ofloops

While Loop

The while loop is probably the most simple one, so we will start with that. The while loop simply executes a block of code as long as the condition you give it is true.

Example

```
using System;

namespace ConsoleApplication1
{
   class Program
   {
      static void Main(string[] args)
      {
```

```
        int number = 0;

        while(number < 5)
        {
           Console.WriteLine(number);
           number = number + 1;
        }

        Console.ReadLine();
     }
  }
}
```

The do loop

The opposite is true for the do loop, which works like the while loop in other aspects through. The do loop evaluates the condition after the loop has executed, which makes sure that the code block is always executed at least once.

```
do
{
Console.WriteLine(number);
   number = number +1;
}while(number <5);
```

The output is the same though - once the number is more than 5, the loop is exited.

The for loop

The for loop is a bit different. It's preferred when you know how many iterations you want, either because you know the exact amount of iterations, or because you have a variable containing the amount. Here is an example on the for loop.

```
usingSystem;

namespaceConsoleApplication1
{
classProgram
{
staticvoidMain(string[] args)
{
int number =5;

for(int i =0; i< number; i++)
Console.WriteLine(i);

Console.ReadLine();
```

```
}
}
}
```

The foreach loop

The last loop we will look at, is the foreach loop. It operates on collections of items, for instance arrays or other built-in list types.

```
using System;
using System.Collections;

namespace ConsoleApplication1
{
   class Program
   {
      static void Main(string[] args)
      {
         ArrayList list = new ArrayList();
         list.Add("John Doe");
         list.Add("Jane Doe");
         list.Add("Someone Else");

         foreach(string name in list)
            Console.WriteLine(name);

         Console.ReadLine();
      }
   }
}
```

b) JaggedArrays

A Jagged array is an array of arrays. You can declare a jagged array named *scores* of type **int** as −

```
int [][] scores;
```

Declaring an array, does not create the array in memory. To create the above array −

```
int[][] scores =newint[5][];
for(int i =0; i<scores.Length; i++){
  scores[i]=newint[4];
}
```

You can initialize a jagged array as −

```
int[][] scores =newint[2][]{newint[]{92,93,94},newint[]{85,66,87,88}};
```

Where, scores is an array of two arrays of integers - scores[0] is an array of 3 integers and scores[1] is an array of 4 integers.

Example

The following example illustrates using a jagged array −

```
usingSystem;

namespaceArrayApplication{
classMyArray{
staticvoidMain(string[] args){

/* a jagged array of 5 array of integers*/
int[][] a =newint[][]{newint[]{0,0},newint[]{1,2},
newint[]{2,4},newint[]{3,6},newint[]{4,8}};
int i, j;

/* output each array element's value */
for(i =0; i<5; i++){
for(j =0; j <2; j++){
Console.WriteLine("a[{0}][{1}] = {2}", i, j,a[i][j]);
}
}
Console.ReadKey();
}
}
}
```

c) Indexers

An **indexer** allows an object to be indexed such as an array. When you define an indexer for a class, this class behaves similar to a **virtual array**. You can then access the instance of this class using the array access operator ([ ]).

Syntax

A one dimensional indexer has the following syntax −

```
element-type this[int index] {

  // The get accessor.
  get {
```

```
    // return the value specified by index
  }

  // The set accessor.
  set {
    // set the value specified by index
  }
}
```

Use of Indexers

Declaration of behavior of an indexer is to some extent similar to a property. similar to the properties, you use **get** and **set** accessors for defining an indexer. However, properties return or set a specific data member, whereas indexers returns or sets a particular value from the object instance. In other words, it breaks the instance data into smaller parts and indexes each part, gets or sets each part.

Defining a property involves providing a property name. Indexers are not defined with names, but with the **this**keyword, which refers to the object instance. The following example demonstrates the concept −

```
usingSystem;

namespaceIndexerApplication{

classIndexedNames{
privatestring[] namelist =newstring[size];
staticpublicint size =10;

publicIndexedNames(){
for(int i =0; i< size;i++)
        namelist[i]="N.A.";
}
publicstringthis[int index]{
get{
string tmp;

if( index>=0&& index <= size-1){
            tmp = namelist[index];
}else{
            tmp ="";
}

return( tmp );
}
set{
if( index>=0&& index <= size-1){
```

```
            namelist[index]= value;
}
}
}
staticvoidMain(string[] args){
IndexedNames names=newIndexedNames();
        names[0]="Zara";
        names[1]="Riz";
        names[2]="Nuha";
        names[3]="Asif";
        names[4]="Davinder";
        names[5]="Sunil";
        names[6]="Rubic";

for(int i =0; i<IndexedNames.size; i++){
Console.WriteLine(names[i]);
}
Console.ReadKey();
}
}
}
```

e) Abstract class

An abstract class is used to define what is known as a base class. A base class is a class which has the most basic definition of a particular requirement.

Features:

1. An abstract calss can inherit from a class and one or more interfaces.
2. An abstract class can implement code with non-Abstractmethods.
3. An Abstract class can have modifiers for methods, propertiesetc.
4. An Abstract class can have constants andfields.
5. An abstract class can implement aproperty.
6. An abstract class can have constructors or destructors.
7. An abstract class cannot be inherited from bystructures.
8. An abstract class cannot support multipleinheritance.

Section II

Q8. WAP in C# to demonstrate implementations of interfaces.

A8. Like a class, an interface can have methods and variables, but the methods declared in interface are by default abstract (only method signature, no body).

Syntax :

```
interface <interface_name> {

   // declare constant fields
   // declare methods that abstract
   // by default.
}
```

```
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System;

namespace InterfaceApplication{

  public interface ITransactions {
    // interface members
    void showTransaction();
    double getAmount();
  }
  public class Transaction :ITransactions {
    private string tCode;
    private string date;
    private double amount;

    public Transaction() {
      tCode = " ";
      date = " ";
      amount = 0.0;
    }
    public Transaction(string c, string d, double a) {
      tCode = c;
      date = d;
      amount = a;
    }
    public double getAmount() {
      return amount;
    }
```

```
    public void showTransaction() {
      Console.WriteLine("Transaction: {0}", tCode);
      Console.WriteLine("Date: {0}", date);
      Console.WriteLine("Amount: {0}", getAmount());
    }
  }
  class Tester {

    static void Main(string[] args) {
      Transaction t1 = new Transaction("001", "8/10/2012", 78900.00);
      Transaction t2 = new Transaction("002", "9/10/2012", 451900.00);

      t1.showTransaction();
      t2.showTransaction();
      Console.ReadKey();
    }
  }
}
```

Q9 WAP in C# to demonstrate try, catch and finally block.

A9. An exception is a problem that arises during the execution of a program. A C# exception is a response to an exceptional circumstance that arises while a program is running, such as an attempt to divide by zero.

Exceptions provide a way to transfer control from one part of a program to another. C# exception handling is built upon four keywords: **try**, **catch**, **finally**, and **throw**.

- **try** − A try block identifies a block of code for which particular exceptions is activated.It is followed by one or more catchblocks.
- **catch** − A program catches an exception with an exception handler at the place ina program where you want to handle the problem. The catch keyword indicates the catching of anexception.
- **finally** − The finally block is used to execute a given set of statements, whether an exception is thrown or not thrown. For example, if you open a file, it must beclosed whether an exception is raised ornot.
- **throw** − A program throws an exception when a problem shows up. This is done usinga throwkeyword.

Syntax

Assuming a block raises an exception, a method catches an exception using a combination of the try and catch keywords. A try/catch block is placed around the code that might generate an exception. Code within a try/catch block is referred to as protected code, and the syntax for using try/catch looks like the following −

```
try {
   // statements causing exception
} catch( ExceptionName e1 ) {
   // error handling code
} catch( ExceptionName e2 ) {
   // error handling code
} catch( ExceptionNameeN ) {
   // error handling code
} finally {
   // statements to be executed
}
using System;

namespace ErrorHandlingApplication{
   classDivNumbers {
      int result;

      DivNumbers() {
         result = 0;
      }
      public void division(int num1, int num2) {
         try {
            result = num1 / num2;
         } catch (DivideByZeroException e) {
            Console.WriteLine("Exception caught: {0}", e);
         } finally {
            Console.WriteLine("Result: {0}", result);
         }
      }
      static void Main(string[] args) {
         DivNumbers d = new DivNumbers();
         d.division(25, 0);
         Console.ReadKey();
      }
   }
}


using System;

namespace UserDefinedException{
   classTestTemperature {
      static void Main(string[] args) {
         Temperature temp = new Temperature();
         try {
            temp.showTemp();
```

```
      } catch(TempIsZeroException e) {
        Console.WriteLine("TempIsZeroException: {0}", e.Message);
      }
      Console.ReadKey();
    }
  }
}
public class TempIsZeroException: Exception {
  public TempIsZeroException(string message): base(message) {
  }
}
public class Temperature {
  int temperature = 0;

  public void showTemp() {

    if(temperature == 0) {
      throw (new TempIsZeroException("Zero Temperature found"));
    } else {
      Console.WriteLine("Temperature: {0}", temperature);
    }
  }
}
```

Q10. A) WAP in C# to sort 10 integer elements with help of a function.

```
using System;
public class Exercise11
{
   public static void Main()
{
            int[] arr1 = new int[10];
   int n, i, j, tmp;


     Console.Write("\n\nSort elements of array in ascending order :\n");
     Console.Write(" ---------------------------------------- \n");

   Console.Write("Input the size of array : ");
   n = Convert.ToInt32(Console.ReadLine());

     Console.Write("Input {0} elements in the array :\n",n);
     for(i=0;i<n;i++)
        {
            Console.Write("element - {0} : ",i);
```

```
                    arr1[i] = Convert.ToInt32(Console.ReadLine());
                }

    for(i=0; i<n; i++)
    {
        for(j=i+1; j<n; j++)
        {
            if(arr1[j] < arr1[i])
            {
                tmp = arr1[i];
                arr1[i] = arr1[j];
                arr1[j] = tmp;
            }
        }
    }
    Console.Write("\nElements of array in sorted ascending order:\n");
    for(i=0; i<n; i++)
    {
        Console.Write("{0} ", arr1[i]);
    }
                            Console.Write("\n\n");
  }

}
```

b) WAP in C# to check whether a no.is palindrome or not.

```
class Program {

    static void Main(string[] args) {

        int num, rem, sum = 0, temp;

        //clrscr();

        Console.WriteLine("\n >>>> To Find a Number is Palindrome or not <<<< ");

        Console.Write("\n Enter a number: ");

        num = Convert.ToInt32(Console.ReadLine());

        temp = num;

        while (num > 0) {
```

```
            rem = num % 10; //for getting remainder by dividing with 10
            num = num / 10; //for getting quotient by dividing with 10
            sum = sum * 10 + rem;
            /*multiplying the sum with 10 and adding
            remainder*/
        }
        Console.WriteLine("\n The Reversed Number is: {0} \n", sum);
        if (temp == sum) //checking whether the reversed number is equal to entered number
        {
            Console.WriteLine("\n Number is Palindrome \n\n");
        } else {
            Console.WriteLine("\n Number is not a palindrome \n\n");
        }
        Console.ReadLine();
    }
}
```