1. Introduction

What is .Net Platform?

The .NET Platform is used to develop enterprise applications based on industry standards. It was introduced to offer a much more powerful, more flexible, and simpler programming model than COM. It is a fully managed, protected, simplified, feature rich application execution environment.

Features of .Net

It is OS independent and hardware independent. Extensively uses Industry standards like HTTP, SOAP, XML and XSD. Easily maintainable due to simplified deployment and version management. A platform to build Web services, Multi-Threaded Applications, Windows Services, Rich Internet Applications as well as Mobile Application. Provides seamless integration to a wide variety of languages.

Main Components of .Net

Common Language Runtime – CLR

Base Class Library (BCL) or Framework Class Library (FCL)

Role of CLR

Running of code, Memory management, Compilation of code (JIT), Provides garbage collection, error handling, Code access security for semi-trusted code and many more.

Role of BCL|FCL

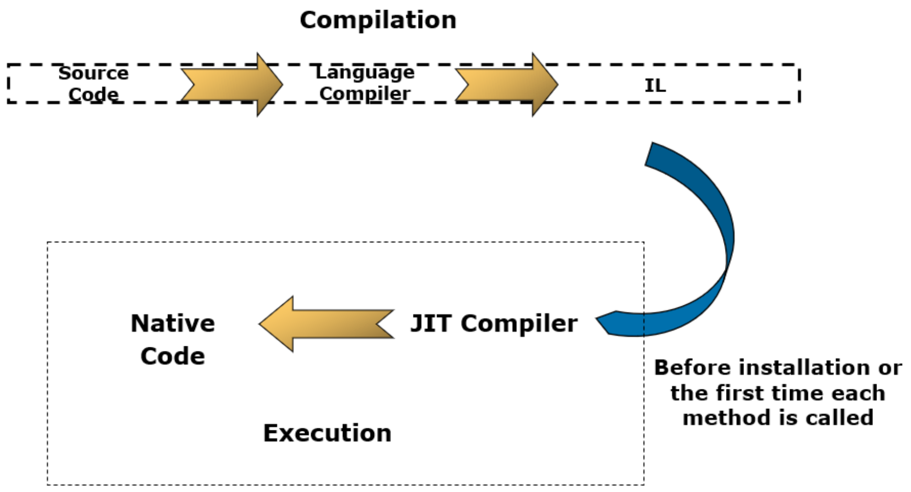
Framework class library is object-oriented collection of reusable classes. You use them to develop applications like:

- System-Side Programing
- Graphical User Interface (GUI) based Desktop applications
- Web Applications or Web Sites
- Web Services and Web API’s
- Distributed Applications

.Net Version

Year	CLR	.NET	Visual Studio	C#	VB
2002	1.0	1.0	2002	1.0	7.0
2003	1.1	1.1	2003	1.0	7.0
2005	2.0	2.0	2005	2.0	8.0
2006	2.0	3.0	.NET 3.0 ext.	2.0	8.0
2007	2.0	3.5	2008	3.0	9.0
2010	4	4	2010	4.0	10.0
2012	4	4.5	2012	5.0	11.0
2015	4	4.5.2	2015	6.0	12.0
2017	4	4.6.2	2017	7.0	12.0

Compilation and Execution in .NET Application



2. Introduction to C#

Why C#?

C# is a strongly typed object-oriented language whose code visually resembles C++ (and Java). This decision by the C# language designers allows C++ developers to easily leverage their knowledge to quickly become productive in C#. C# syntax differs from C++ in some ways, but most of the differences between these languages are semantic and behavioral, stemming from differences in the runtime environments in which they execute.

Features of C#

C# is simple. C# is modern. C# is object oriented. C# is type safe. Provides Interoperability. C# is scalable and updatable.

Command used to compile source code (Application) written in C# in Command Prompt

csc /t:exe FileName.cs   here /t or /target is the compiler switch

Command used to compile source code (Dynamic Link Library) written in C# in Command Prompt

csc /t:library FileName.cs   here /library is the compiler switch

Command used to compile source code (Application) written in C# in Command Prompt, which uses a library

csc /t:exe /r:MathLib.cs Calculator.cs   here /t ot /target and /r or /reference are compiler switches

3. Data Types and Arrays

Types of Data Types in C#

ValueTypes – bool, char, int, float, double, etc.

ReferenceTypes – class, string and arrays

Nullable types

Nullable types represent value-type variables that can be assigned the value of null. You cannot create a nullable type based on a reference type. A nullable type can represent the normal range of values for its underlying value type, plus an additional null value.

What is implicit typed variable?

Type of the local variable being declared is inferred from the expression used to initialize the variable.

What are Arrays?

An array is a data structure that contains several variables called the elements of the array. All the array elements must be of the same type. Arrays are reference type derived from the abstract base type System.Array.

Types of Arrays

Single-dimension array

Multi-dimension array

Jagged array

4. OOPs in C#

Class

A class is a user-defined type (UDT) that is composed of field data (member variables) and methods (member functions) that act on this data.

Constructor

A constructor is a special method with same name as of your class. It is used to initialize values during object creation to your class fields.

Constructor Types

Default Constructor

Parameterized Constructor

Static Constructor

Method

A method is a member that implements a computation or action that can be performed by an object or class. They can take parameters or can be parameter less. May or may not return data.

Types in which a parameter is passed

Pass by value. This is the default.

Pass by reference.

Using out keyword.

Using params keyword.

Method overloading

In C#, two or more methods within the same class can share the same name, if their parameter declarations are different. In such cases, the methods are said to be overloaded, and the process is referred to as method overloading.

Static constructor

A constructor can also be specified as static. A static constructor is typically used to initialize attributes that apply to a class rather than an instance. A static constructor is used to initialize aspects of a class before any objects of the class are created.

Types of Access Modifiers in C#

public

private

protected

internal

protectedinternal

Properties

Properties provide the chance to protect a field in a class by reading and writing to it through the property accessor. Accomplished in programs by implementing the specialized getter and setter methods.

Indexers

Indexers are 'smart arrays'. Indexers permit instances of a class or struct to be indexed in the same way as arrays. Indexers are like properties except that their accessors take parameters. Indexers don't have a name.

Inheritance

Inheritance is a form of software reusability in which classes are created by reusing the data and behaviors of an existing class with new capabilities. A class inheritance hierarchy begins with a base class that defines a set of common attributes and operations that it shares with derived classes. A derived class inherits the resources of the base class and overrides or enhances their functionality with new capabilities.

Method hiding

It is possible for a derived class to define a member that has the same name as a member in its base class. When this happens, the member in the base class is hidden within the derived class. Even though this is not technically an error in C#, the compiler issues a warning message. If you intended to hide a base class member purposely, then to prevent this warning, the derived class member must be preceded by the **new** keyword.

Method overriding

Polymorphism provides a way for a subclass to customize the implementation of a method defined by its base class. If, in a base class, you define a method that may be overridden by a subclass, you should specify the method as virtual.

Interface

An interface defines a contract. Interface is a purely abstract class; it has only signatures, no implementation. May contain methods, properties, indexers and events (no fields, constants, constructors, destructors, operators, nested types).

Object Initializers

An object initializer is used to assign values to an object fields or properties when the object is created. There is no need to explicitly invoke a constructor. It combines object creation and initialization in a single step.

Anonymous Types

Implicit type functionality for objects. Set property values into an object without writing a class definition. The resulting class has no usable name. The class name is generated by the compiler. The created class inherits from Object.

Partial types

Partial types allow classes, structs, and interfaces to be broken into multiple pieces stored in different source files for easier development and maintenance. Additionally, partial types allow separation of machine-generated and user-written parts of types so that it is easier to augment code generated by a tool.

5. Exception Handling

What is an exception?

An exception is an event that occurs during the execution of a program that disrupts the normal flow of instructions during the execution of a program. Exceptions are notifications that some error has occurred in the program. When an exception occurs, you can ignore the exception, or you can write code to deal with the exception, this is known as exception handling.

Try Block

Contains program statements you wish to monitor for exceptions. If an exception occurs within the try block, it is thrown.

Catch Block

Your code catches this exception using catch and handles it in some rational manner. Like writing to the log file. Displaying a user-friendly message regarding what went wrong.

Finally Block

This block is optional. Its where you write your clean up code. Any code that you must execute after you exit a try block irrespective of whether an exception was throw or not is put in a finally block.

Throw keyword

You use the throw keyword to explicitly throw either a built-In or user defined exception based on a condition to halt the flow of execution.

Custom Exceptions

Although C#'s built-in exceptions handle most common errors. You can use custom exceptions to handle errors in your own code. As a rule, exceptions you define should be derived from ApplicationException as this is the hierarchy reserved for application-related exceptions.

6. Collections & Generics

Need for Collections

You are developing an Employee-tracking application. You implement a data structure named Employee to store employee information. However, you do not know the number of records that you need to maintain. You can store the data structure in an array, but then you would need to write code to add each new employee. To add a new item to an array, you first have to create a new array that has room for an additional element. Then, you need to copy the elements from the original array into the new array and add the new element.

To simplify this process, the .NET Framework provides classes that are collectively known as Collections. By using collections, you can store several items within one object. Collections have methods that you can use to add and remove items. These methods automatically resize the corresponding data structures without requiring additional code.

What are collections?

Collections are heterogenous groups of objects dynamic in nature. You can add, insert and remove items. Collections enumerable data structures that can be accessed using indexes or keys.

ArrayList

The ArrayList class is a dynamic array of heterogeneous objects. In an array we can store only objects of the same type. However, in an ArrayList we can have different types of objects. These in turn would be stored as object type only.

Stack

Provides a Last-in-First-out (LIFO) collection of items of the System.Object type. The last added item is always at the top of the Stack and is also the first one to be removed.

Queue

Is a data structure that provides a First-in-First-out collection of items of the System.Object type. Newly added items are stored at the end or the rear of the Queue and items are deleted from the front of the Queue.

HashTable

Creates a collection that uses a hash table for storage. Represents a dictionary of associated keys and values, implemented as a hash table. Provides a faster way of storage and retrieval of items of the object type. Provides support for key based searching.

What is Generics?

Generics permit classes, structs, interfaces, delegates, and methods to be parameterized by the types of data they store and manipulate. The type parameter is specified in < and > delimiters after the class name. The type parameter T acts as a placeholder until an actual type is specified at use.

What are type constraints?

A type parameter constraint specifies a requirement that a type must fulfill in order to be used as an argument for that type parameter. Constraints are declared using the word where, followed by the name of a type parameter, followed by a list of class or interface types and optionally the constructor constraint new().

What are Iterators?

An iterator is a method, get accessor or operator that enables you to support for each iteration in a class or struct without having to implement the entire IEnumerable interface. Instead, you provide just an iterator, which simply traverses the data structures in your class. When the compiler detects your iterator, it will automatically generate the Current, MoveNext and Dispose methods of the IEnumerable or IEnumerable<T> interface.

An iterator is a section of code that returns an ordered sequence of values of the same type. An iterator can be used as the body of a method, an operator, or a get accessor. The iterator code uses the yield return statement to return each element in turn. yield break ends the iteration.

7. Delegates

Delegate

A delegate is a reference type that refers to a Shared method of a type or to an instance method of an object. The closest equivalent of a delegate in other languages is a function pointer. Whereas a function pointer can only reference Shared functions, a delegate can reference both Shared and instance methods.

Types of Delegates

SingleCast Delegate

Multicast Delegate

Event

An event is a way for a class to provide notifications when something of interest happens. Events allow an object to notify other objects that a change has occurred. The other objects can register an interest in an event, and they will be notified when the event occurs. In other words, events allow objects to register that they need to be notified about changes to other objects.

Publisher

A publisher is an object that maintains its internal state. When its state changes, it can raise an event to alert other interested objects about the change.

Subscriber

A subscriber is an object that registers an interest in an event. It is alerted when a publisher raises the event. An event can have zero or more subscribers.

Anonymous Method

Anonymous methods allow you to create an instance of a delegate by specifying a block of inline code to be invoked by the delegate rather than specifying an existing method to be invoked by the delegate. This allows you to conserve code because you do not need to create a new method every time a delegate is passed as a parameter to a method call. Anonymous methods are not supported in Visual Basic.

Lambda Expression

A lambda expression is an anonymous function that can contain expressions and statements and can be used to create delegates or expression tree types. All lambda expressions use the lambda operator =>, which is read as "goes to". The left side of the lambda operator specifies the input parameters (if any) and the right side holds the expression or statement block.

8. Garbage Collection

Garbage Collector

Objects are dynamically allocated from a pool of free memory by using the new operator. Of course, memory is not infinite, and the free memory can be exhausted. Thus, it is possible for new to fail because there is insufficient free memory to create the desired object. For this reason, one of the key components of any dynamic allocation scheme is the recovery of free memory from unused objects, making that memory available for subsequent reallocation. In many programming languages, the release of previously allocated memory is handled manually. C# uses a different, more trouble-free approach garbage collection.

Dispose Method

The .NET Framework garbage collector does not allocate or release unmanaged memory. Unmanaged resources are resources such as file and pipe handles, registry handles, wait handles, or pointers to blocks of unmanaged memory. You implement a Dispose method to release unmanaged resources used by your application. To implement a Dispose method, you need to implement IDisposable interface.

9. IO and Serialization

File

A file is an ordered and named collection of a particular sequence of bytes having persistent storage.

Exploring the System.IO NameSpace

The System.IO namespace is the region of the base class libraries devoted to file-based (and memory-based) input and output services. Following Table shows core types of System.IO namespace.

Directory and File Info Types

System.IO provides four types that allow you to manipulate individual files, as well as interact with a machine’s directory structure. The Directory and File types, expose creation, deletion, and manipulation operations using various static members. The closely related FileInfo and DirectoryInfo types expose similar functionality as instance-level methods.

What is Serialization?

Serialization is the process of writing the state of an object to a byte stream. Object Serialization is the process of reducing the objects instance into a format that can either be stored to disk or transported over a Network. Serialization is useful when you want to save the state of your application to a persistence storage area. Later, you may restore these objects by using the process of deserialization.

What is a Formatter?

A formatter is used to determine the serialization format for objects. All formatters expose an interface called the IFormatter interface. Two formatters inherited from the IFormatter interface and are provided as part of the .NET Framework. These are:

Binary formatter

SOAP formatter

IDeserializationCallback interface

The IDeserializationCallback interface specifies that a class is to be informed when deserialization of the whole object graph has been finished. To enable your class to initialize a nonserialized member automatically, use the IDeserializationCallback interface and then implement IDeserializationCallback.OnDeserialization.

10. Assembly, Reflection and Attribute

Assembly

An assembly can be viewed as a unit of deployment. An assembly is self-describing binary (DLL or EXE) containing collection of types and optional recourses. .NET binaries contain code constructed using Microsoft Intermediate Language (MSIL or simply IL), which is platform and CPU agnostic. It contains "metadata" which completely describes each type.

Types of Assemblies

Private Assembly

Shared Assembly

Private Assemblies

Private assemblies are a collection of types that are only used by the application with which it has been deployed. Private assemblies are required to be located within the main directory of the owing application.

Shared Assemblies

Shared assemblies can be used by several clients on a single machine. Shared assemblies are installed into a machine wide "Global Assembly Cache" (GAC). A shared assembly must be assigned a "shared name" (also known as a "strong name").

Strong Name

For creating a Shared assembly, create a unique "strong name" for the assembly. A strong name contains the following information:

- Friendly string name and optional culture information (just like a private assembly)
- Version identifier
- Public key value
- Embedded digital signature

Reflection

Reflection is ability to find information about types contained in an assembly at run time. .NET provides a whole new set of APIs to introspect assemblies and objects. All the APIs related to reflection are located under System.Reflection namespace. .NET reflection is a powerful mechanism which allows you to inspect type information and invoke methods on those types at runtime.

Attributes

Attributes concept in .NET is a way to mark or store meta data about the code in assembly. Often it is an instruction meant for the runtime. The Runtime can change its behavior or course of action based on the attribute present. In .NET framework there are many built in attributes.

For e.g.: Serializable, NonSerlalized, XmlIgnore, WebMethod to name few

11. Parallel and Async programming with C#

CLR and DLR enhancements

Increased usability with features like Optional parameters & Named arguments. Provides dynamic programming. Introduced Late binding support in C#, while it still remains a statically typed language. Provides support for interaction with dynamic languages like Python & Ruby etc. Improved COM Interop using Optional parameters (Omit ref on COM calls). 'dynamic' keyword allows C# to defer method invocation at runtime i.e Dynamic method Invocation. Co-Variance and Contra-Variance.

Dynamic

C# 4.0 supports late-binding using a new keyword called 'dynamic'. The type 'dynamic' can be thought of like a special version of type 'object', which signals that the object can be used dynamically. C# provides access to new DLR (Dynamic language runtime) through this new dynamic keyword. When dynamic keyword is encountered in the code, compiler will understand this is a dynamic invocation & not the typical static invocation

```
E.g. : dynamic d  = GetDynamicObject();
            d.Add(5);
```

Here d is declared as dynamic, so C# allows you to call method with any name & any parameters on d. Thus in short the compiler defers its job of resolving type/method names to the runtime.

Co-Variance

Covariance means think "out"

E.g. :

```
        public interface IEnumerable<out T> : IEnumerable
    {
        IEnumerator<T> GetEnumerator();
    }
```

Here the 'out' keyword, actually modifies the definition of type parameter, T. Compiler sees this and marks type T as covariant. The out keyword signifies that the generic type parameter, T can appear only in output positions like, as method return value and read-only properties.

Here the interface becomes covariant in 'T'. It means that IEnumerable<A> is implicitly reference convertible to IEnumerable<B>, if A has an implicit reference conversion to B.

Contra-Variance

Contra-Variance means think "in"

In this case, 'in' keyword signifies that the generic type parameters can only be used at input positions like, as method parameters and write-only properties

E.g. :- .NET framework has an interface IComparable<T>, which has a single method called CompareTo :

```
public interface IComparable<in T> {
 bool CompareTo(T other); }
```

Here the 'in' keyword, actually modifies the definition of type parameter, T. Compiler sees this and marks type T as contravariant. Now, the following code :

```
IComparable<Employee> ec = GetEmployeeComparer();
IComparable<Manager> mc = ec;
```

Task Parallel Library

The Task Parallel Library (TPL), as its name implies, is based on the concept of the task. Tasks are a new abstraction in .NET 4 to represent units of asynchronous work. Tasks were not available in earlier versions of .NET, and developers would instead use ThreadPool work items for this purpose. The term task parallelism refers to one or more independent tasks running concurrently. A task represents an asynchronous operation, and in some ways it resembles the creation of a ThreadPool work item, but at a higher level of abstraction.

Tasks provide two primary benefits:

- More efficient and more scalable use of system resources
- More programmatic control than is possible with a thread or work item

Parallel Extensions in .NET

Data Parallelism

This refers to dividing the data across multiple processors for parallel execution. e.g. we are processing an array of 1000 elements we can distribute the data between two processors say 500 each. Using Parallel Foreach, For, Invoke, PLINQ

Task Parallelism

This breaks down the program into multiple tasks which can be parallelized and are executed on different processors. This is supported by Task Parallel Library (TPL) in .NET and the Task class. Implement the "Task Asynchronous Pattern" TAP.

Async

async: This modifier indicates the method is now asynchronous.

It provides a simpler way to perform potentially long-running operations without blocking the callers thread. The caller of this async method can resume its work without waiting for this asynchronous method to finish its job. Reduces developer's efforts for writing an additional code. The method with async modifier has at least one await. This method now runs synchronously until the first await expression written in it.

Await

await: Typically, when a developer writes some code to perform an asynchronous operations using threads, then he/she has to explicitly write necessary code to perform wait operations to complete a task. In asynchronous methods, every operation which is getting performed is called a Task. The 'await' keyword is applied on the task in an asynchronous method and suspends the execution of the method, until the awaited task is not completed. During this time, the control is returned back to the caller of this asynchronous method