Unit -1 Introduction to C# and .NET Framework [7 Hrs]

Object Orientation; Type Safety; Memory Management; Platform Support; C# and CLR; CLR and .NET Framework; Framework Overview; .NET Standard 2.0; Applied Technologies

Introduction to C# Language

C# is a general-purpose, object-oriented programming language. **C# was developed by Anders Hejlsberg and his team during the development of .Net Framework.** It can be used to create desktop, web and mobile applications.

C# is a hybrid of C and C++; it is a Microsoft programming language developed to compete with Sun's Java language. C# is an object-oriented programming language used with XML-based Web services on the .NET platform and designed for improving productivity in the development of Web applications.

C# is an elegant and type-safe object-oriented language that enables developers to build a variety of secure and robust applications that run on the .NET Framework. You can use C# to create Windows client applications, XML Web services, distributed components, client-server applications, database applications, and much, much more. Visual C# provides an advanced code editor, convenient user interface designers, integrated debugger, and many other tools to make it easier to develop applications based on the C# language and the .NET Framework. As an object-oriented language, C# supports the concepts of encapsulation, inheritance, and polymorphism. All variables and methods, including the Main method, the application's entry point, are encapsulated within class definitions.

The following reasons make C# a widely used professional language (Features of C#):

1. It is a modern, general-purpose programming language
2. It is object oriented.
3. It is component oriented.
4. It is easy to learn.
5. It is a structured language.
6. It produces efficient programs.
7. It can be compiled on a variety of computer platforms.
8. It is a part of .Net Framework.

Object Orientation

C# is a rich implementation of the object-orientation paradigm, which includes encapsulation, abstraction, inheritance, and polymorphism. Encapsulation means creating a boundary around an object, to separate its external (public) behaviour from its internal (private) implementation details.

The distinctive features of C# from an object-oriented perspective are:

Unified type system

The fundamental building block in C# is an encapsulated unit of data and functions called a type. C# has a unified type system, where all types ultimately share a common base type. This means that all types, whether they represent business objects or are primitive types such as numbers, share the same basic functionality. For example, an instance of any type can be converted to a string by calling its ToString method.

Classes and interfaces

In a traditional object-oriented paradigm, the only kind of type is a class. In C#, there are several other kinds of types, one of which is an interface. An interface is like a class, except that it only describes members. The implementation for those members comes from types that implement the interface. Interfaces are particularly useful in scenarios where multiple inheritance is required (unlike languages such as C++ and Eiffel, C# does not support multiple inheritance of classes).

Properties, methods, and events

In the pure object-oriented paradigm, all functions are methods. In C#, methods are only one kind of function member, which also includes properties and events. Properties are function members that encapsulate a piece of an object's state, such as a button's colour or a label's text. Events are function members that simplify acting on object state changes.

While C# is primarily an object-oriented language, it also borrows from the functional programming paradigm. Specifically:

Functions can be treated as values

Through the use of delegates, C# allows functions to be passed as values to and from other functions.

C# supports patterns for purity

Core to functional programming is avoiding the use of variables whose values change, in favour of declarative patterns. C# has key features to help with those patterns, including the ability to write unnamed functions on the fly that "capture" variables (lambda expressions), and the ability to perform list or reactive programming via query expressions. C# also makes it easy to define read-only fields and properties for writing immutable (read-only) types.

Type Safety

C# is primarily a type-safe language, meaning that instances of types can interact only through protocols they define, thereby ensuring each type's internal consistency. For instance, C# prevents you from interacting with a string type as though it were an integer type.
More specifically, C# supports static typing, meaning that the language enforces type safety at compile time. This is in addition to type safety being enforced at run- time.

Static typing eliminates a large class of errors before a program is even run. It shifts the burden away from runtime unit tests onto the compiler to verify that all the types in a program fit together correctly. This makes large programs much easier to manage, more predictable, and more robust. Furthermore, static typing allows tools such as IntelliSense in Visual Studio to help you write a program, since it knows for a given variable what type it is, and hence what methods you can call on that variable.

C# is also called a strongly typed language because its type rules (whether enforced statically or at runtime) are very strict. For instance, you cannot call a function that's designed to accept an integer with a floating-point number, unless you first explicitly convert the floating-point number to an integer. This helps prevent mistakes.

Strong typing also plays a role in enabling C# code to run in a sandbox—an environment where every aspect of security is controlled by the host. In a sandbox, it is important that you cannot arbitrarily corrupt the state of an object by bypassing its type rules.

Memory Management

C# relies on the runtime to perform automatic memory management. The Com- mon Language Runtime has a garbage collector that executes as part of your program, reclaiming memory for objects that are no longer referenced. This frees programmers from explicitly deallocating the memory for an object, eliminating the problem of incorrect pointers encountered in languages such as C++.

C# does not eliminate pointers: it merely makes them unnecessary for most programming tasks.

Platform Support

Historically, C# was used almost entirely for writing code to run on Windows platforms. Recently, however, Microsoft and other companies have invested in other platforms, including Linux, macOS, iOS, and Android.

Xamarin™ allows cross platform C# development for mobile applications, and Portable Class Libraries are becoming increasingly widespread.

Microsoft's ASP.NET Core is a cross-platform lightweight web hosting framework that can run either on the .NET Framework or on .NET Core, an open source cross-platform runtime.

C# and CLR

C# depends on a runtime equipped with a host of features such as automatic memory management and exception handling. **At the core of the Microsoft .NET Framework is the Common Language Runtime (CLR), which provides these runtime features**. (The .NET Core and Xamarin frameworks provide similar runtimes.)

The CLR is language-neutral, allowing developers to build applications in multiple languages (e.g., C#, F#, Visual Basic .NET, and Managed C++).

C# is one of several managed languages that get compiled into managed code. Man- aged code is represented in Intermediate Language or IL. The CLR converts the IL into the native code of the machine, such as X86 or X64, usually just prior to execution. This is referred to as Just-In-Time (JIT) compilation. Ahead-of-time compilation is also available to improve start up time with large assemblies or resource constrained devices (and to satisfy iOS app store rules when developing with Xamarin).

The container for managed code is called an assembly or portable executable. An assembly can be an executable file (.exe) or a library (.dll), and contains not only IL, but type information (metadata). The presence of metadata allows assemblies to reference types in other assemblies without needing additional files.

A program can query its own metadata (reflection), and even generate new IL at runtime (reflection.emit)

Introduction and Overview of .Net Framework

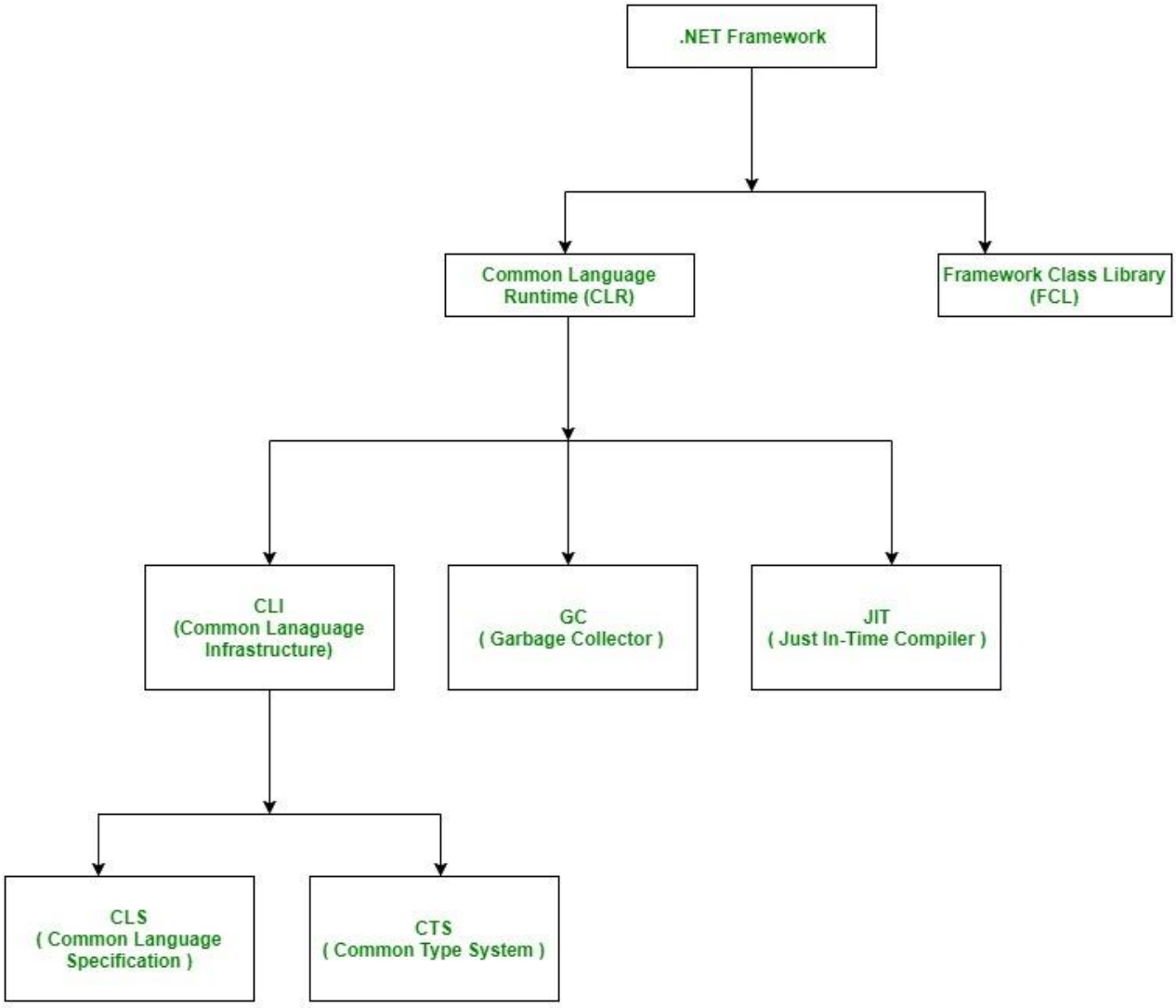
.NET is the framework for which we develop applications. It sits in between our application programs and operating system.

.NET provides an object oriented environment. It ensures safe execution of the code by performing required runtime validations. For example, it is never possible to access an element of an array outside the boundary. Similarly, it is not possible to a program to write into another programs area, etc. The runtime validations performed by .NET makes the entire environment robust.

It is a programming infrastructure created by Microsoft for building, deploying, and running applications and services that use .NET technologies, such as desktop applications and Web services.

The .NET Framework consists of:

- the Common Language Runtime
- the Framework Class Library



Common Language Runtime(CLR):

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

Features of Common Language Runtime:

1. The runtime enforces code access security. For example, users can trust that an executable embedded in a Web page can play an animation on screen or sing a song, but cannot access their personal data, file system, or network.
2. The runtime also enforces code robustness by implementing a strict type-and-code-verification infrastructure called the common type system (CTS). The CTS ensures that all managed code is self-describing.
3. The runtime also accelerates developer productivity. For example, programmers can write applications in their development language of choice, yet take full advantage of the runtime, the class library, and components written in other languages by other developers. Any compiler vendor who chooses to target the runtime can do so. Language compilers that target the .NET Framework make the features of the .NET Framework available to existing code written in that language, greatly easing the migration process for existing applications.
4. While the runtime is designed for the software of the future, it also supports software of today and yesterday.
5. The runtime is designed to enhance performance.
6. The runtime can be hosted by high-performance, server-side applications, such as Microsoft SQL Server and Internet Information Services (IIS).

Framework Class Library (FCL):

The class library is a comprehensive, object-oriented collection of reusable types that you can use to develop applications ranging from traditional command-line or graphical user interface (GUI) applications to applications based on the latest innovations provided by ASP.NET, such as Web Forms and XML Web services.

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

The core libraries are sometimes collectively called the Base Class Library (BCL). The entire framework is called the Framework Class Library (FCL).

Features of Framework Class Library:

An object-oriented class library, the .NET Framework types enable you to accomplish a range of common programming tasks, including tasks such as string management, data collection, database connectivity, and file access. In addition to these common tasks, the class library

includes types that support a variety of specialized development scenarios. For example, you can use the .NET Framework to develop the following types of applications and services:

1. Console applications
2. Windows GUI applications (Windows Forms).
3. Windows Presentation Foundation (WPF) applications.
4. ASP.NET applications.
5. Windows services.
6. Service-oriented applications using Windows Communication Foundation (WCF).
7. Workflow-enabled applications using Windows Workflow Foundation (WF).

What's New in .NET Framework 4.6

- The Garbage Collector (GC) offers more control over when (not) to collect via new methods on the GC class. There are also more fine-tuning options when calling `GC.Collect`.
- There's a brand-new faster 64-bit JIT compiler.
- The `System.Numerics` namespace now includes hardware-accelerated matrix, vector types, `BigInteger` and `Complex`.
- There's a new `System.AppContext` class, designed to give library authors a consistent mechanism for letting consumers switch new API features in or out.
- `Tasks` now pick up the current thread's culture and UI culture when created.
- More collection types now implement `IReadOnlyCollection<T>`.
- WPF has further improvements, including better touch and high-DPI handling.
- ASP.NET now supports HTTP/2 and the Token Binding Protocol in Windows 10.

What's New in .NET Framework 4.7

Framework 4.7 is more of a maintenance release than a new-feature release, with numerous bug fixes and minor improvements. Additionally:

- The `System.ValueTuple` struct is part of Framework 4.7, so you can use tuples in C# 7 without referencing the *System.ValueTuple.dll* assembly.
- WPF has better touch support.
- Windows Forms has better support for high-DPI monitors.

Table 5-1 shows the history of compatibility between each version of C#, the CLR, and the .NET Framework.

Table 5-1. C#, CLR, and .NET Framework versions

C# version	CLR version	.NET Framework versions
1.0	1.0	1.0
1.2	1.1	1.1
2.0	2.0	2.0, 3.0
3.0	2.0 (SP2)	3.5
4.0	4.0	4.0
5.0	4.5 (Patched CLR 4.0)	4.5
6.0	4.6 (Patched CLR 4.0)	4.6
7.0	4.6/4.7 (Patched CLR 4.0)	4.6/4.7

Other Frameworks

The Microsoft .NET Framework is the most expansive and mature framework, but runs only on Microsoft Windows (desktop/server). Over the years, other frame- works have emerged to support other platforms. There are currently three major players besides the .NET Framework, all of which are currently owned by Microsoft:

Universal Windows Platform (UWP)

For writing Windows 10 Store Apps and for targeting Windows 10 devices (mobile, XBox, Surface Hub, Hololens). Your app runs in a sandbox to lessen the threat of malware, prohibiting operations such as reading or writing arbitrary files.

.NET Core with ASP.NET Core

An open source framework (originally based on a cut-down version of the .NET Framework) for writing easily deployable Internet apps and micro services that run on Windows, macOS, and Linux. Unlike the .NET Frame- work, .NET Core can be packaged with the web application and xcopy deployed (self-contained deployment).

Xamarin

For writing mobile apps that target iOS, Android, and Windows Mobile. The Xamarin company was purchased by Microsoft in 2016.

Table 1-1 compares the current platform support for each of the major frameworks

Table 1-1. Platform support for the popular frameworks

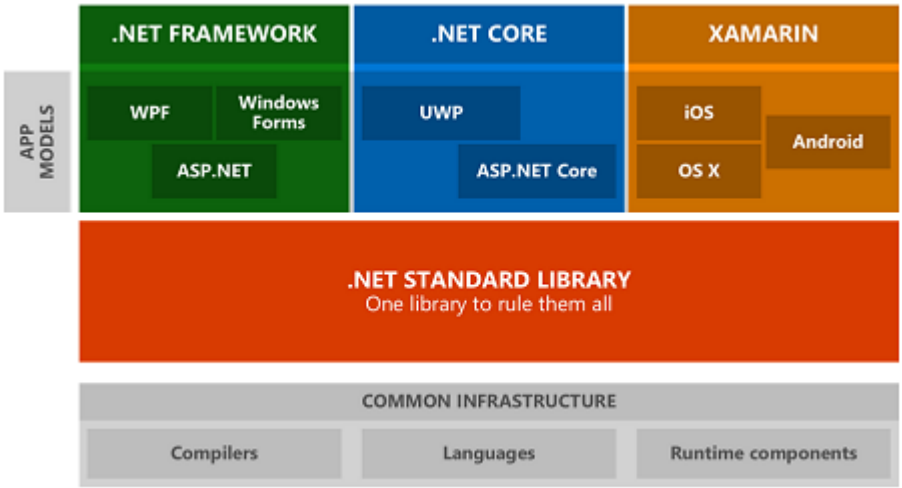
Target operating system	.NET Framework	UWP	.NET Core	Xamarin
Windows 7/8	Yes		Yes	
Windows 10 desktop/server	Yes	Yes	Yes	
Windows 10 devices		Yes		Yes
Linux			Yes	
macOS			Yes	
iOS (iPhone)				Yes
Android				Yes

.Net Standard 2.0

.Net Standard is a specification which dictates what the Base Class Libraries of different .Net platforms should implement to unify the Base Class Libraries of different .Net Platforms. Here, Platform means full .Net Framework, .Net Core, Xamarin, Silverlight, XBox etc. This also enables code sharing between applications that runs on these different platforms. For example, a library or a component that is developed on top of a platform that implements specific .Net Standard version can be shared by all the applications that runs on any of the platforms that implements same .Net Standard version.

.NET Standard is not a Framework; it's merely a specification describing a minimum baseline of functionality (types and members), which guarantees compatibility with a certain set of frameworks. The concept is similar to C# interfaces: .NET Standard is like an interface that concrete types (frameworks) can implement

Net Standard has solved all this in a different way, it provided an API specification which all platforms should implement to remain .Net Standard complaint. This has unified the base class libraries of different .Net platforms and has paved way to share libraries and also brought the BCL evolution centralized as seen below.



.NET Standard 2.0 is a new version that significantly increases the number of APIs compared to the previous version (1.6.1). In fact, the API surface has more than doubled with .NET Standard 2.0.

The .NET Standard 2.0 is supported by the following .NET implementations:

- .NET Core 2.0 or later
- .NET Framework 4.6.1 or later
- Mono 5.4 or later
- Xamarin.iOS 10.14 or later
- Xamarin.Mac 3.8 or later
- Xamarin.Android 8.0 or later
- Universal Windows Platform 10.0.16299 or later

What's new in the .NET Standard 2.0?

The .NET Standard 2.0 includes the following new features:

1) A vastly expanded set of APIs
2) Support for .NET Framework libraries
3) Support for Visual Basic
4) Tooling support for .NET Standard libraries

Older .NET Standards

There are also older .NET Standards in use, most notably 1.1, 1.2, 1.3, and 1.6. A higher-numbered standard is always a strict superset of a lower-numbered standard. For instance, if you write a library that targets .NET Standard 1.6, you will support not only recent versions of the four major frameworks, but also .NET Core 1.0. And if you target .NET Standard 1.3, you support everything we've already mentioned plus .NET Framework 4.6.0 (see Table 5-2).

Table 5-2. Older .NET Standards

If you target...	You also support...
Standard 1.6	.NET Core 1.0
Standard 1.3	Above plus .NET 4.6.0
Standard 1.2	Above plus .NET 4.5.1, Windows Phone 8.1, WinRT for Windows 8.1
Standard 1.1	Above plus .NET 4.5.0, Windows Phone 8.0, WinRT for Windows 8.0

The 1.x standards lack thousands of APIs that are present in 2.0, including much of what we describe in this book. This can make targeting a 1.x standard significantly more challenging, especially if you need to integrate existing code or libraries.

If you need to support older frameworks but don't need cross platform compatibility, a better option is to target an older version of a specific framework. In the case of Windows, a good choice is .NET Framework 4.5 because it's widely deployed (pre-installed on all machines running Windows 8 and later), and it contains most of what's in .NET Frame- work 4.7.

You can also think of .NET Standard as a lowest common denominator. In the case of .NET Standard 2.0, the four frameworks that implement it have a similar Base Class Library, so the lowest common denominator is big and useful. However, if you also want compatibility with .NET Core 1.0 (with its significantly cut-down BCL), the lowest common denominator—.NET Standard 1.x—becomes much smaller and less useful.

The CLR and Core Framework

System Types

The most fundamental types live directly in the System namespace. These include C#'s built-in types, the Exception base class, the Enum, Array, and Delegate base classes, and Nullable, Type, DateTime, TimeSpan, and Guid. The System namespace also includes types for performing mathematical functions (Math), generating random numbers (Random), and converting between various types (Convert and Bit Converter).

Text Processing

The System.Text namespace contains the StringBuilder class (the editable or mutable cousin of string), and the types for working with text encodings, such as UTF-8 (Encoding and its subtypes).

The System.Text.RegularExpressions namespace contains types that perform advanced pattern-based search-and-replace operations.

Collections

The .NET Framework offers a variety of classes for managing collections of items. These include both list- and dictionary-based structures, and work in conjunction with a set of standard interfaces that unify their common characteristics. All collection types are defined in the following namespaces:

```
System.Collections                // Nongeneric collections
System.Collections.Generic        // Generic collections
System.Collections.Specialized    // Strongly typed collections
System.Collections.ObjectModel    // Bases for your own collections
System.Collections.Concurrent     // Thread-safe collection (Chapter 23)
```

Queries

Language Integrated Query (LINQ) was added in Framework 3.5. LINQ allows you to perform type-safe queries over local and remote collections (e.g., SQL Server tables). A big advantage of LINQ is that it presents a consistent querying API across a variety of domains. The essential types reside in the following namespaces, and are part of .NET Standard 2.0:

```
System.Linq                 // LINQ to Objects and PLINQ
System.Linq.Expressions     // For building expressions manually
System.Xml.Linq             // LINQ to XML
```

The full .NET Framework also includes the following,

```
System.Data.Linq            // LINQ to SQL
System.Data.Entity          // LINQ to Entities (Entity Framework)
```

XML

XML is used widely within the .NET Framework, and so is supported extensively. The XML name- spaces are:

```
System.Xml              // XmlReader, XmlWriter + the old W3C DOM
System.Xml.Linq         // The LINQ to XML DOM
System.Xml.Schema       // Support for XSD
```

```
System.Xml.Serialization  // Declarative XML serialization for .NET types
System.Xml.XPath          // XPath query language
System.Xml.Xsl            // Stylesheet support
```

Diagnostics

Diagnostics refers to .NET's logging and assertion facilities and describe how to interact with other processes, write to the Windows event log, and use performance counters for monitoring. The types for this are defined in and under System.Diagnostics. Windows-specific features are not part of .NET Standard, and are avail- able only in the .NET Framework.

Concurrency and Asynchrony

Many modern applications need to deal with more than one thing happening at a time. Since C# 5.0, this has become easier through asynchronous functions and high-level constructs such as tasks and task combinators. Types for working with threads and asynchronous operations are in the System.Threading and System.Threading.Tasks namespaces.

Streams and I/O

The Framework provides a stream-based model for low-level input/output. Streams are typically used to read and write directly to files and network connections, and can be chained or wrapped in decorator streams to add compression or encryption functionality. The .NET Stream and I/O types are defined in and under the System.IO namespace, and the WinRT types for file I/O are in and under Windows.Storage.

Networking

You can directly access standard network protocols such as HTTP, FTP, TCP/IP, and SMTP via the types in System.Net.

```
System.Net
System.Net.Http          // HttpClient
System.Net.Mail          // For sending mail via SMTP
System.Net.Sockets       // TCP, UDP, and IP
```

The latter two namespaces are unavailable to Windows Store applications if you're targeting Windows 8/8.1 (WinRT), but are available to Windows 10 Store apps (UWP) as part of the .NET Standard 2.0 contract. For WinRT apps, use third-party libraries for sending mail, and the WinRT types in Windows.Networking.Sockets for working with sockets.

Serialization

The Framework provides several systems for saving and restoring objects to a binary or text representation. Such systems are required for distributed application technologies, such as WCF, Web Services, and Remoting, and also to save and restore objects to a file. The types for serialization reside in the following namespaces:

```
System.Runtime.Serialization
System.Xml.Serialization
```

Assemblies, Reflection, and Attributes

The assemblies into which C# programs compile comprise executable instructions (stored as intermediate language or IL) and metadata, which describes the pro- gram's types, members, and attributes. Through reflection, you can inspect this metadata at runtime, and do such

things as dynamically invoke methods. With Reflection.Emit, you can construct new code on the fly.

```
System
System.Reflection
System.Reflection.Emit  // .NET Framework only
```

Dynamic Programming

Dynamic Language Runtime, which has been a part of the CLR since Framework 4.0. The types for dynamic programming are in System.Dynamic.

Security

Code access, role, and identity security, and the transparency model introduced in CLR 4.0. Cryptography can be done in the Framework, covering encryption, hashing, and data protection. The types for this are defined in:

System.Security
System.Security.Permissions
System.Security.Policy
System.Security.Cryptography

Advanced Threading

C#'s asynchronous functions make concurrent programming significantly easier because they lessen the need for lower-level techniques. However, there are still times when you need signaling constructs, thread-local storage, reader/writer locks, and so on. Threading types are in the System.Threading namespace.

Applied Technologies

Descriptions of .NET Implementations

	OS	Open Source	Purpose
.NET Framework	Windows	No	Used for building Windows desktop applications and ASP.NET Web apps running on IIS.
.NET Core	Windows, Linux, macOS	Yes	Used for building cross-platform console apps and ASP.NET Core Web apps and cloud services.
Xamarin	iOS, Android, macOS	Yes	Used for building mobile applications for iOS and Android, as well as desktop apps for macOS.
.NET Standard	N/A	Yes	Used for building libraries that can be referenced from all .NET implementations, such as .NET Framework, .NET Core and Xamarin.

Following are the different applied technologies:

User-Interface APIs

User-interface–based applications can be divided into two categories: thin client, which amounts to a website, and rich client, which is a program the end user must download and install on a computer or mobile device.

For thin client applications, .NET provides ASP.NET and ASP.NET Core.For rich-client applications that target Windows 7/8/10 desktop, .NET provides the WPF and Windows Forms APIs. For rich-client apps that target iOS, Android, and Windows Phone, there's Xamarin, and for writing rich-client store apps for Win- dows 10 desktop and devices.

Finally, there's a hybrid technology called Silverlight, which has been largely abandoned since the rise of HTML5.

ASP.NET

Applications written using ASP.NET host under Windows IIS and can be accessed from any web browser. Here are the advantages of ASP.NET over rich-client technologies:

- There is zero deployment at the client end.
- Clients can run a non-Windows platform.
- Updates are easily deployed.

In writing your web pages, you can choose between the traditional Web Forms and the newer MVC (Model-View-Controller) API. Both build on the ASP.NET infra- structure. Web Forms has been part of the Framework since its inception; MVC was written much later in response to the success of Ruby on Rails and MonoRail. It provides, in general, a better programming abstraction than Web Forms; it also allows more control over the generated HTML.

ASP.NET Core

A relatively recent addition, ASP.NET Core is similar to ASP.NET, but runs on both .NET Framework and .NET Core (allowing for cross-platform deployment). ASP.NET Core features a lighter-weight modular architecture, with the ability to self-host in a custom process, and an open source license. Unlike its predecessors, ASP.NET Core is not dependent on System.Web and the historical baggage of Web Forms. It's particularly suitable for micro-services and deployment inside containers.

Windows Presentation Foundation (WPF)

WPF was introduced in Framework 3.0 for writing rich-client applications. The benefits of WPF over its predecessor, Windows Forms, are as follows:

- It supports sophisticated graphics, such as arbitrary transformations, 3D rendering, multimedia, and true transparency. Skinning is supported through styles and templates.
- Its primary measurement unit is not pixel-based, so applications display correctly at any DPI (dots per inch) setting.
- It has extensive and flexible layout support, which means you can localize an application without danger of elements overlapping.
- Rendering uses DirectX and is fast, taking good advantage of graphics hard- ware acceleration.
- It offers reliable data binding.
- User interfaces can be described declaratively in XAML files that can be maintained independently of the "code-behind" files—this helps to separate appearance from functionality.

Windows Forms

Windows Forms is a rich-client API that's as old as the .NET Framework. Com- pared to WPF, Windows Forms is a relatively simple technology that provides most of the features you need in writing a typical Windows application. It also has significant relevancy in maintaining legacy applications. It has a number of drawbacks, though, compared to WPF:

- Controls are positioned and sized in pixels, making it easy to write applications that break on clients whose DPI settings differ from the developer's (although this has improved somewhat in Framework 4.7).
- The API for drawing nonstandard controls is GDI+, which, although reasonably flexible, is slow in rendering large areas (and without double buffering, may flicker).
- Controls lack true transparency.
- Most controls are non-compositional. For instance, you can't put an image control inside a tab control header. Customizing list views and combo boxes is time-consuming and painful.
- Dynamic layout is difficult to get right reliably

Xamarin

Xamarin, now owned by Microsoft, lets you write mobile apps in C# that target iOS and Android, as well as Windows Phone. Being cross-platform, this runs not on the .NET Framework, but its own framework (a derivation of the open source Mono framework).

UWP (Universal Windows Platform)

UWP is for writing apps that target Windows 10 desktop and devices, distributed via the Windows Store. Its rich-client API is designed for writing touch-first user interfaces, and was inspired by WPF and uses XAML for layout. The namespaces are Windows.UI and Windows.UI.Xaml.

Silverlight

Silverlight is also distinct from the .NET Framework, and lets you write a graphical UI that runs in a web browser, much like Macromedia's Flash. With the rise of HTML5, Microsoft has abandoned Silverlight.

Backend Technologies

ADO.NET

ADO.NET is the managed data access API. Although the name is derived from the 1990s-era ADO (ActiveX Data Objects), the technology is completely different. ADO.NET contains two major low-level components:

Provider layer

The provider model defines common classes and interfaces for low-level access to database providers. These interfaces comprise connections, commands, adapters, and readers (forward-only, read-only cursors over a database). The Framework ships with native support for Microsoft SQL Server, and numerous third-party drivers are available for other databases.

DataSet model

A DataSet is a structured cache of data. It resembles a primitive in-memory database, which defines SQL constructs such as tables, rows, columns, relationships, constraints, and views. By programming against a cache of data, you can reduce the number of trips to the server, increasing server scalability and the responsiveness of a rich-client user interface. DataSets are serializable and are designed to be sent across the wire between client and server applications.

Sitting above the provider layer are three APIs that offer the ability to query databases via LINQ:

- Entity Framework (.NET Framework only)
- Entity Framework Core (.NET Framework and .NET Core)
- LINQ to SQL (.NET Framework only)

LINQ to SQL is simpler than Entity Framework, and has historically produced better SQL (although Entity Framework has benefited from numerous updates).
Entity Framework is more flexible in that you can create elaborate mappings between the database and the classes that you query (Entity Data Model), and offers a model that allows third-party support for databases other than SQL Server.
Entity Framework Core (EF Core) is a rewrite of Entity Framework with a simpler design inspired by LINQ to SQL. It abandons the complex Entity Data Model and runs on both .NET Framework and .NET Core.

Windows Workflow (.NET Framework only)

Windows Workflow is a framework for modelling and managing potentially long running business processes. Workflow targets a standard runtime library, providing consistency and interoperability. Workflow also helps reduce coding for dynamically controlled decision-making trees. Windows Workflow is not strictly a backend technology—you can use it anywhere.

Workflow came originally with .NET Framework 3.0, with its types defined in the System.WorkFlow namespace. Workflow was substantially revised in Framework 4.0; the new types live in and under the System.Activities namespace.

COM+ and MSMQ (.NET Framework only)

The Framework allows you to interoperate with COM+ for services such as distributed transactions, via types in the System.EnterpriseServices namespace.

It also supports MSMQ (Microsoft Message Queuing) for asynchronous, one-way messaging through types in System.Messaging.

Distributed System Technologies

Windows Communication Foundation (WCF)

WCF is a sophisticated communications infrastructure introduced in Framework 3.0. WCF is flexible and configurable enough to make both of its predecessors— Remoting and (.ASMX) Web Services—mostly redundant.

WCF, Remoting, and Web Services are all alike in that they implement the following basic model in allowing a client and server application to communicate:

- On the server, you indicate what methods you'd like remote client applications to be able to call.
- On the client, you specify or infer the signatures of the server methods you'd like to call.
- On both the server and the client, you choose a transport and communication protocol (in WCF, this is done through a binding).
- The client establishes a connection to the server.
- The client calls a remote method, which executes transparently on the server.

WCF further decouples the client and server through service contracts and data contracts. Conceptually, the client sends an (XML or binary) message to an end- point on a remote service, rather than directly invoking a remote method. One of the benefits of this decoupling is that clients have no dependency on the .NET plat- form or on any proprietary communication protocols.

For .NET-to-.NET communication, however, WCF offers richer serialization and better tooling than with REST APIs. It's also potentially faster as it's not tied to HTTP and can use binary serialization.

The types for communicating with WCF are in, and below, the System.Service Model namespace.

Web API

Web API runs over ASP.NET/ASP.NET Core and is architecturally similar to Microsoft's MVC API, except that it's designed to expose services and data instead of web pages. Its advantage over WCF is in allowing you to follow popular REST over HTTP conventions, offering easy interoperability with the widest range of platforms.

REST implementations are internally simpler than the SOAP and WS- protocols that WCF relies on for interoperability. REST APIs are also architecturally more elegant for loosely-coupled systems, building on de-facto standards and making excellent use of what HTTP already provides.

Remoting and .ASMX Web Services (.NET Framework only)

Remoting and .ASMX Web Services are WCF's predecessors. Remoting is almost redundant in WCF's wake, and .ASMX Web Services has become entirely redundant.

Remoting is geared toward tightly coupled applications. A typical example is when the client and server are both .NET applications written by the same company (or companies sharing common assemblies). Communication typically involves exchanging potentially complex custom .NET objects that the Remoting infrastructure serializes and deserializes without needing intervention.

The types for Remoting are in or under System.Runtime.Remoting; the types for Web Services are under System.Web.Services.

Scope of .Net Technology:

Over the period of time, many software have evolved along with many new technologies getting introduced. In this race, one which caught the people's interest was .Net. In a very short time this new technology became the boon for the software developer community and now it's been considered as the most growing career option, which clearly indicates that .Net development still rules.

Its growing popularity has made it the first choice of many experienced and fresher and now one can think of having a great career start in this field outside India too. .Net is now part of many international markets like USA, UAE, South Africa, UK and other developing countries and is heading forward with each passing day. With its every new version .Net technologies is evolving at a fast pace and creating amazing job opportunities for the developers.

The availability of RAD in.Net, which means the Rapid Application Development is the reason behind its success. The plus point of learning this technology is that you can develop as many applications as you want for different platforms and environments. You can even use it for building XML web applications and web services that can excellently run on the Internet. .Net is best suited for developing window based applications, web server programs and applications, which are both PC and mobile compatible. It's easy to transfer feature is what makes it the popular choice.

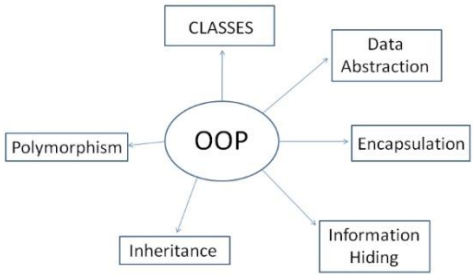
The biggest advantage of learning .net is that one can get a job in various profiles like he/she can be absorbed as a software developer also or a.Net technician too. Today, there are an array of institutes and firms that offer certified and short term course in .Net, which is a great move from career point of view. Whether you are a diploma holder or an Engineer or an MCA, learning .Net will surely set your career and will offer it a right pace and track.

There are ample of career options in this particular field. An interested candidate can go for MCTS(VB.net), MCTS(ASP.net) and MCPD, which are some of the international certifications. You can even choose from Cisco certifications like CCNA, CCNP, CCIE, which will give a new direction to your career.

With so many job prospects in this technology, choosing it will be an ideal choice for your career. This clearly illustrates the future scope of .Net, which is sure to offer you great future ahead in almost all the spheres, ranging from Desktop applications to mobile applications.

Feature of Object Oriented Programming:

Object-oriented programming (OOP) refers to a type of computer programming (software design) in which programmers define not only the data type of a data structure, but also the types of operations (functions) that can be applied to the data structure.
In this way, the data structure becomes an object that includes both data and functions. In addition, programmers can create relationships between one object and another. For example, objects can inherit characteristics from other objects.



Following are the features of OOPS:
Abstraction: The process of picking out (abstracting) common features of objects and procedures.
Class: A category of objects. The class defines all the common properties of the different objects that belong to it.
Encapsulation: The process of combining elements to create a new entity. A procedure is a type of encapsulation because it combines a series of computer instructions.
Information hiding: The process of hiding details of an object or function. Information hiding is a powerful programming technique because it reduces complexity.
Inheritance: a feature that represents the "is a" relationship between different classes.
Interface: the languages and codes that the applications use to communicate with each other and with the hardware.
Object: a self-contained entity that consists of both data and procedures to manipulate the data.
Polymorphism: A programming language's ability to process objects differently depending on their data type or class.
Procedure: a section of a program that performs a specific task.

Procedure-Oriented vs. Object-Oriented Programming

Object-Oriented Programming (OOP) is a high-level programming language where a program is divided into small chunks called objects using the object-oriented model, hence the name. This paradigm is based on objects and classes.

- **Object –** An object is basically a self-contained entity that accumulates both data and procedures to manipulate the data. Objects are merely instances of classes.
- **Class –** A class, in simple terms, is a blueprint of an object which defines all the common properties of one or more objects that are associated with it. A class can be used to define multiple objects within a program.

The OOP paradigm mainly eyes on the data rather than the algorithm to create modules by dividing a program into data and functions that are bundled within the objects. The modules cannot be modified when a new object is added restricting any non-member function access to the data. Methods are the only way to assess the data.

Objects can communicate with each other through same member functions. This process is known as message passing. This anonymity among the objects is what makes the program secure. A programmer can create a new object from the already existing objects by taking most of its features thus making the program easy to implement and modify.

Procedure-Oriented Programming (POP) follows a step-by-step approach to break down a task into a collection of variables and routines (or subroutines) through a sequence of instructions. Each step is carried out in order in a systematic manner so that a computer can understand what to do. The program is divided into small parts called functions and then it follows a series of computational steps to be carried out in order.
It follows a top-down approach to actually solve a problem, hence the name. Procedures correspond to functions and each function has its own purpose. Dividing the program into functions is the key to procedural programming. So a number of different functions are written in order to accomplish the tasks.

Difference between OOP and POP is shown below:

Procedure Oriented Programming	Object Oriented Programming
In POP, program is divided into small parts called **functions**.	In OOP, program is divided into parts called **objects**.
In POP, Importance is not given to **data** but to functions as well as **sequence** of actions to be done.	In OOP, Importance is given to the data rather than procedures or functions because it works as a **real world**.
POP follows **Top Down approach**.	OOP follows **Bottom Up approach**.
POP does not have any access specifier.	OOP has access specifiers named Public, Private, Protected, etc.
In POP, Data can move freely from function to function in the system.	In OOP, objects can move and communicate with each other through member functions.
To add new data and function in POP is not so easy.	OOP provides an easy way to add new data and function.

In POP, Most function uses Global data for sharing that can be accessed freely from function to function in the system.	In OOP, data cannot move easily from function to function, it can be kept public or private so we can control the access of data.
POP does not have any proper way for hiding data so it is **less secure**.	OOP provides Data Hiding so provides **more security**.
In POP, Overloading is not possible.	In OOP, overloading is possible in the form of Function Overloading and Operator Overloading.
Example of POP are : C, VB, FORTRAN, Pascal.	Example of OOP are : C++, JAVA, VB.NET, C#.NET.