**What is .NET?**

**Ans:**

* It is a product of Microsoft launched in the year 2002,
* .NET is a free, cross-platform, open-source developer platform which can be used for building various kinds of Applications like: Web, Mobile, Desktop, Micro services, Cloud, Machine Learning, Game Development and IoT (Internet of Things).

**How to develop all the above applications by using .NET?**

**Ans:** To develop the above applications, .NET provides with a set of Programming Languages, Technologies & Servers using which we can build any kind of Application.

**What are the Programming Languages, .NET provides to us?**

**Ans:** In .NET there are 30+ programming languages available for a developer to build applications and programmers have a chance of choosing any 1 language from the list.

**Features of .NET:** there are 2 important features in .NET, those are:

1. Language Independent
2. Platform Independent

All Object-Oriented Programming Languages have an important feature that is “Code Re-usability” i.e., the code we write in 1 program can be consumed from another program, for example:

C++ Source Code => Compiled by using C++ Compiler => Generates Object Code => Which can be consumed from another C++ Program.

Java Source Code => Compiled by using Java Compiler => Generates Byte Code => Which can be consumed from another Java Program.

C# Source Code => Compiled by using C# Compiler => Generates CIL Code => Which can be consumed from any .NET Language Program.

F# Source Code => Compiled by using F# Compiler => Generates CIL Code => Which can be consumed from any .NET Language Program.

VB Source Code => Compiled by using VB Compiler => Generates CIL Code => Which can be consumed from any .NET Language Program.

**Note:** Re-usability in CPP and Java Languages is only with-in that language whereas the same re-usability in .NET Languages is across all languages of .NET, and this is what we call as Language Independent.

**Note:** In .NET Languages we will not face compiled code mismatch because all languages are generating CIL or MSIL Code only after the compilation. They don’t face data type mis-match problem also because all languages of .NET adopt a rule known as “Uniform Data Type Structure” i.e., similar types will always be same in size irrespective of their names.

**2. Platform Independent:** it is an approach of executing an application that is developed on 1 platform, in other platforms.

**What is a Platform?**

**Ans:** A platform is an environment under which an application executes, and it is a combination of 2 things, those are Micro-Processor and Operating System.

Text

Description automatically generated

**Note:** up to 1995, application that are developed by using programming languages that are present in the market (E.g., C, CPP, VB, Cobol, Pascal, Fortran, VCPP) are all platform dependent i.e., if we develop any application by using any of these languages on 1 platform, we can't run them on other platforms. For example, we can’t install MS Office on Linux or Mac OS, so it is a platform dependent application.

**What is a Platform?**

**Ans:** It is the environment in which a piece of software is executed. A platform can also be called as the stage on which computer programs can run. Platform can refer to the type of processor (CPU) on which a given operating system runs, the type of operating system on a computer or the combination of the type of hardware and the type of operating system running on it. An example of a common platform is Microsoft Windows running on x86 architecture. Other well-known desktop computer platforms include Linux/Unix and macOS

**What is Platform Independent?**

**Ans:** Applications that are developed by using Java and .NET Languages are Platform Independent i.e.,these applications once developed on a Platform can run on any other Platform (i.e., write once and run anywhere).

**What is Cross-platform?**

**Ans:** In computing, cross-platform software (also multi-platform software or platform-independent software) is computer software that is implemented to run on multiple platforms. For example, a cross-platform application may run on Microsoft Windows, Linux, and macOS. Cross-platform programs may run on as many as all existing platforms, or on few platforms.

**Application developed by using .NET languages on Windows OS:**

Source Code => Compiled by a Language Compiler => CIL Code

**CIL**(Common Intermediate Language)**:**

**Note:** as said earlier, .NET is a collection of programming languages so with whatever .NET Language we develop the application and compile the source code by using an appropriate language compiler, the outcome will be “CIL” (Common Intermediate Language) code only.

We will install CIL Code on Client Machines and to run that code we need to install software known as “.NET Runtime “and inside of this Runtime there will be a component called CLR (Common Language Runtime) which will convert CIL Code into Native Machine Code.

In the year 2002 when Microsoft launched .NET in the market, they provided their first Runtime for Windows O.S. only but not for any other O.S.’s, but they made the specifications to develop the Runtime as open, so 3rd party companies came forward and developed the Runtime’s for other O.S.’s also and the name of that runtime is “.NET Framework”. The first version of .NET Framework is 1.0 and the last version is 4.8.

Diagram, schematic

Description automatically generated

**Note:** with .NET Framework Runtime there is a criticism on .NET that it is not fully Platform Independent because Microsoft has given it only for Windows.

In the year 2016 Microsoft launched a new Runtime into the market with the name “.NET Core” and this runtime is provided for Windows, Linux, and Mac machines also. The first version of .NET Core is 1.0 and the last version is 3.1.

Diagram, schematic

Description automatically generated

On November 10, 2020, Microsoft launched a new Runtime into the market by combining .NET Framework & .NET Core as **1 .NET** which starts from version 5.0 and the latest is 6.0 launched on November 2021.

Diagram, schematic

Description automatically generated

**Note:** the new .NET is nothing but .NET Core only but with-out again calling .NET Core and .NET Framework they made the name simple as just “.NET”.

**What is a .NET Runtime?**

**Ans:** It’s a software which must be installed on Client’s Machine if at all we want to run .NET Application’s on that Machine which sits on top of the O.S. and executes the CIL Code by masking the functionalities of an OS.

In case of platform dependent languages like Cobol, C, CPP, Visual Basic, etc. Compiled Code i.e., Machine Code runs directly under OS., whereas in case of .NET Languages, CIL Code will run under the .NET Runtime.

Text

Description automatically generated with low confidence

**Note:** Application’s that directly run under the O.S. are known as Un-Managed App's whereas App's that run under .NET Runtime are known as Managed App's.

**What is .NET Framework and .NET Core?**

**Ans:** .NET is a developer platform made up of tools, programming languages, and libraries for building many different types of applications. There are various implementations of .NET, and each implementation allows .NET code to execute in different places - Linux, macOS, Windows, iOS, Android, and many more. Various implementations of the .NET include:

1. **.NET Framework:** it is the original implementation of .NET, and it supports running websites, services, desktop apps, and more on Windows.
2. **.NET Core:** it is a cross-platform implementation for running websites, services, and console apps on Windows, Linux, and macOS.
3. **Xamarin/Mono:** it is a .NET implementation for running apps on all the major mobile operating systems, including iOS and Android.

The runtime software internally contains 2 main components in it, those are the “Libraries” and an “Execution Engine” as following:

Diagram, text

Description automatically generated

**Libraries:** A library is a set of re-usable functionalities, and every programming language has built-in libraries to it like Header Files in C & CPP Languages and Packages in Java Language same as that .NET Languages are also provided with built-in libraries and we call them “FCL (Framework Class Libraries)” in .NET Framework and “CORE FX” in .NET Core and .NET.

**Execution Engine:** as discussed earlier, .NET Applications will not run under the OS, but they will be running under the Runtime and in this Runtime, we have an Execution Engine responsible for the execution of Applications and we call this as “CLR (Common Language Runtime)” in .NET Framework and “CORE CLR” in .NET Core and .NET.

**CLR** and **Core CLR** are known as the execution engine of .NET Runtime, where all .NET Application run under the supervision of this CLR and it internally it contains various components in it to manage various actions, like:

Graphical user interface, diagram

Description automatically generated

1. **Security Engine:** this is responsible for the security of our applications, i.e., it will take care that applications don't directly interact with the OS, as well as OS don't directly interact the application.
2. **JIT Compiler:** this is the compiler which is responsible for converting CIL Code into Machine Code based on the platform where we are executing the application adopting a process known as “Conversion gradually during the program’s execution”.

Diagram, schematic

Description automatically generated

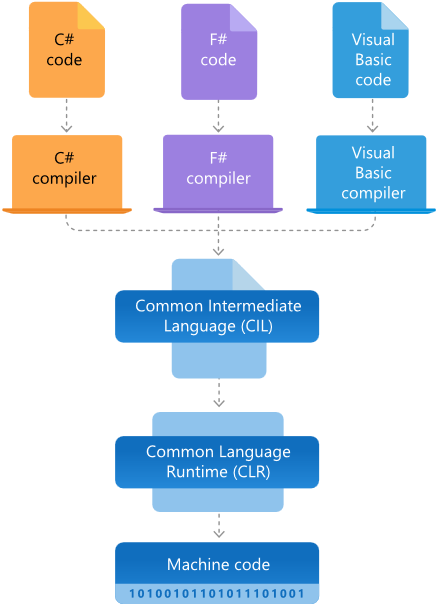
1. **Garbage Collector:** it is responsible for “Automatic Memory Management” where “Memory Management” is a process of allocation and de-allocation of memory that is required for a program to execute, and this is of 2 types:
   * Manual or Explicit
   * Automatic or Implicit

Manual or Explicit means, in this case programmers are responsible for allocation and de-allocation of the memory explicitly. Automatic or Implicit means, here programmers are not at all responsible for allocation and de-allocation of the memory and on behalf of the programmers Garbage Collector will take the responsibility for memory management.

**Architecture of .NET Framework:** The two major components of .NET Framework are the .NET Framework Class Library and the Common Language Runtime.

1. The Class Library provides a set of APIs and types for common functionality. It provides types for strings, dates, numbers, etc. The Class Library includes APIs for reading and writing files, connecting to databases, drawing, and more.
2. The Common Language Runtime (CLR) is the heart of .NET Framework and the execution engine that handles running applications. It provides services like thread management, garbage collection, type-safety, exception handling, and more.

**Architecture of .NET Framework CLR:** .NET applications can be written in any .NET Language like C#, F#, or Visual Basic. Source Code we write by using some .NET Language is compiled into a language-agnostic Common Intermediate Language (CIL) and the compiled code is stored as assemblies (files with a “.dll” or “.exe” extension). When we run the applications, CLR takes the assemblies and uses a just-in-time compiler (JIT) to turn it into machine code that can execute on the specific architecture of the computer it is running on.



**Which version of .NET Framework should I use?**

**Ans:** In most cases, you should use the latest stable release and currently, that's .NET Framework 4.8. Applications that were created with any 4.x version of .NET Framework will run on .NET Framework 4.8. To run an application that was created for an earlier version (for example, .NET Framework 3.5), you should install that version.

**What is .NET Core Runtime?**

**Ans:** This includes everything you need to run a .NET Core Application. The runtime is also included in the SDK. When an app author publishes an app, they can include the runtime with their app. If they don't include the runtime, it's up to the user to install the runtime. There are three different runtimes you can install on Windows:

* ASP.NET Core runtime: Runs ASP.NET Core apps. Includes the .NET Core runtime.
* Desktop runtime: Runs .NET Core WPF and .NET Core Windows Forms desktop apps for Windows. Includes the .NET Core runtime.
* .NET Core runtime: This runtime is the simplest runtime and doesn't include any other runtime. It's highly recommended that you install both ASP.NET Core runtime and Desktop runtime for the best compatibility with .NET Core apps.

**What is the difference between .NET Core and .NET Framework?**

**Ans:** .NET Core and .NET Framework share many of the same components and you can share code across the two. Some key differences include:

* .NET Core is cross-platform and runs on Linux, macOS, and Windows. .NET Framework only runs on Windows.
* .NET Core is open-source and accepts contributions from the community. The .NET Framework source code is available but does not take direct contributions.
* The majority of .NET innovation happens in .NET Core.
* .NET Framework is included in Windows and automatically updated machine-wide by Windows Update. .NET Core is shipped independently.

**C# Namespaces**

Namespaces in C# are used to organize too many classes so that it can be easy to handle the application.

In a simple C# program, we use System.Console where System is the namespace and Console is the class.

To access the class of a namespace, we need to use namespacename.classname(import the namespace).

We can use **using** keyword so that we don't have to use complete name all the time.

In C#, global namespace is the root namespace. The global::System will always refer to the namespace "System" of .Net Framework.

# C# | String vs StringBuilder

* StringBuilder is used to represent a *mutable string* of characters.
* Mutable means the string which can be changed.
* So String objects are immutable but StringBuilder is the mutable string type.
* It will not create a new modified instance of the current string object but do the modifications in the existing string object.
* The complete functionality of StringBuilder is provided by StringBuilder class which is present in System.Text namespace.

**Need of the StringBuilder:**

As stated above that the String class objects are immutable which means that if the user will modify any string object it will result into the creation of a new string object.

It makes the use of string costly.

So when the user needs the repetitive operations on the string then the need of StringBuilder come into existence.

It provides the optimized way to deal with the repetitive and multiple string manipulation operations.

**Example:**

namespace TestApplication

{

internal class StringBuilderTest

{

public static void Method1(string s)

{

string str = "Gubbala";

s=string.Concat(str, s);

}

public static void Method2(StringBuilder sb)

{

sb.Append("Swetha");

}

public static void Main(string[] args)

{

string s = "Sekhar";

Method1(s);

Console.WriteLine("My name is {0}",s);//My name is Sekhar

StringBuilder stringBuilder= new StringBuilder("Kada ");

Method2(stringBuilder);

Console.WriteLine("My wife name is {0}", stringBuilder);

//My wife name is Kada Swetha

Console.ReadLine();

}

}

}

**When to use which one:**

* If a string is going to remain constant throughout the program, then use String class object because a String object is immutable.
* If a string can change (example: lots of logic and operations in the construction of the string) then using a StringBuilder is the best option.

**What is Singleton Design Pattern?**

 Singleton design pattern in C# is one of the most popular design patterns. In this pattern, a class has only one instance in the program that provides a global point of access to it. In other words, a singleton is a class that allows only a single instance of itself to be created and usually gives simple access to that instance.

There are various ways to implement a singleton pattern in C#. The following are the common characteristics of a singleton pattern.

Private and parameterless single constructor

Sealed class.

Static variable to hold a reference to the single created instance

A public and static way of getting the reference to the created instance.

Advantages of Singleton Design Pattern

The advantages of a Singleton Pattern are,

Singleton pattern can implement interfaces.

Can be lazy-loaded and has Static Initialization.

It helps to hide dependencies.

It provides a single point of access to a particular instance, so it is easy to maintain.

Disadvantages of Singleton Design Pattern

The disadvantages of a Singleton Pattern are,

Unit testing is a bit difficult as it introduces a global state into an application

Reduces the potential for parallelism within a program by locking.

**Eager Loading**Eager Loading helps you to load all your needed entities at once; i.e., all your child entities will be loaded at single database call. This can be achieved, using the Include method, which returs the related entities as a part of the query and a large amount of data is loaded at once.

For example, you have a User table and a UserDetails table (related entity to User table), then you will write the code given below. Here, we are loading the user with the Id equal to userId along with the user details.

User usr = dbContext.Users.Include(a => a.UserDetails).FirstOrDefault(a => a.UserId == userId);

If you have multiple level of child entities, then you can load, using the query given below.

User usr = dbContext.Users.Include(a => a.UserDetails.Select(ud => ud.Address)).FirstOrDefault(a => a.UserId == userId);

**Lazy Loading**It is the default behavior of an Entity Framework, where a child entity is loaded only when it is accessed for the first time. It simply delays the loading of the related data, until you ask for it.

For example, when we run the query given below, UserDetails table will not be loaded along with the User table.

User usr = dbContext.Users.FirstOrDefault(a => a.UserId == userId);

It will only be loaded when you explicitly call for it, as shown below.

UserDeatils ud = usr.UserDetails; // UserDetails are loaded here

**Explicit Loading**  
There are options to disable Lazy Loading in an Entity Framework. After turning Lazy Loading off, you can still load the entities by explicitly calling the Load method for the related entities. There are two ways to use Load method Reference (to load single navigation property) and Collection (to load collections), as shown below.

User usr = dbContext.Users.FirstOrDefault(a => a.UserId == userId);

dbContext.Entry(usr).Reference(usr => usr.UserDetails).Load();

When to use what

Use Eager Loading when the relations are not too much. Thus, Eager Loading is a good practice to reduce further queries on the Server.

Use Eager Loading when you are sure that you will be using related entities with the main entity everywhere.

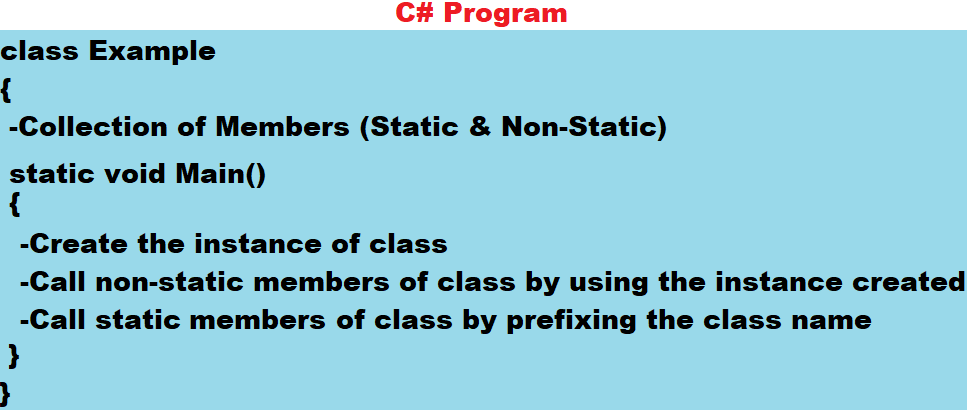
Use Lazy Loading when you are using one-to-many collections.

Use Lazy Loading when you are sure that you are not using related entities instantly.

When you have turned off Lazy Loading, use Explicit loading when you are not sure whether or not you will be using an entity beforehand.

**C# Multithreading**

Multithreading in C# is a process in which multiple threads work simultaneously. It is a process to achieve multitasking. It saves time because multiple tasks are being executed at a time. To create multithreaded application in C#, we need to use **System.Threding** namespace.



**Note:** In Java or C# Languages if at all the class contains only Main method in it, we don't require to create object or instance of that class to run the class.

**Syntax to define a class:**

[<modifiers>] class <Name>

{

-Define Members here [] => Optional

} <> => Any

* modifiers are some special keywords that can be used on a class like public, internal, static, abstract, partial, sealed, etc.
* class is a keyword to tell that we are defining a class just like we used, struct keyword to define a structure in C Language.
* <Name> refers to name of the class for identification.
* Members refer to contents of the class like fields, methods, etc.

**Syntax to define Main Method in the class:**

static void Main( [string[] args] )

{

-Stmt's

}

* static is a keyword we use to declare a member as static member and if a member is declared as static, instance of the class is not required to call or execute it. In C# Main method should be declared static to start the execution from there.
* void is a keyword to specify that the method is non-value returning.
* Main is name of the method, which can't be changed and more over it should be in **Pascal Case** only.
* If required (optional) we can pass parameters to Main method but it should be of type string array only.
* Statements refers to the logic we want to implement.

**Writing the first program in C# using Notepad:**

**Step 1:** Open Notepad and write the following code in it:

class First

{

static void Main()

{

System.Console.Clear();

System.Console.WriteLine("My first C# program using Notepad.");

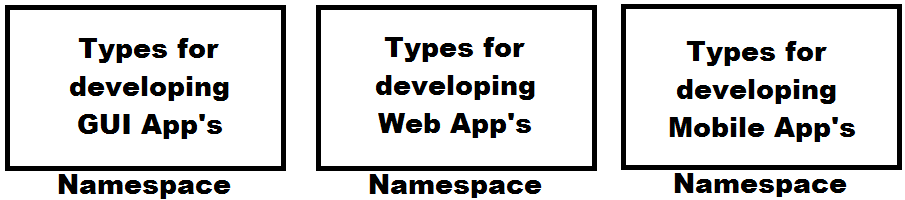
}

}

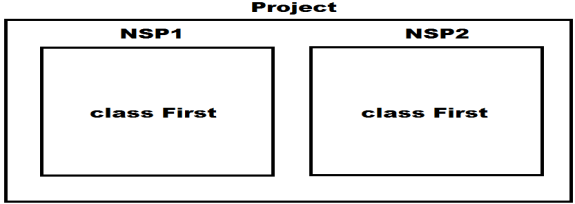
**System.Console.WriteLine & System.Console.Clear:** Console is a pre-defined class under the libraries of our language which provides with a set of static members using which we can perform IO operations on the standard IO devices. WriteLine is a method in the class Console to display output on the monitor, and apart from WriteLine method there are many other methods present in the class Console like: Write, Read, ReadLine, ReadKey, Clear, etc. and all these methods are also static, so we can call them directly by prefixing the class name.

System is a namespace, and a namespace is a logical container for types like: Class, Structure, Interface, Enum and Delegate, and we use these namespaces in a language for 2 reasons:

1. Grouping related types i.e., types that are designed for developing similar kind of App's are grouped together under a namespace for easy access and identification as following:



1. To overcome the naming collision i.e., if a project contains multiple types with the same name, we can overcome conflict between names by putting them under separate namespaces, as following:



**Note:** Every pre-defined type in our Libraries is defined under some namespace and we can also define types under namespaces, and we will learn this process while working with Visual Studio.

If a type is defined under any namespace, then, whenever and wherever we want to consume the type, we need to prefix namespace name to type name, and this is the reason why in our previous program we have referred to “Console” class as “System.Console”. To overcome the problem of prefixing namespace, name every time before the type, we are provided with an option of “importing a namespace” which is done by “**using directive**” as following:

**Syntax: using <namespace>;**

using System;

using Microsoft.VisualBasic;

**Note:** We can import any no. of namespaces as above but each import should be a separate statement.