1. What is the **.NET Framework**? What are the different components of it?

* **.Net Framework** provides runtime environment called **Common Language Runtime** (CLR).It provides an environment to run all the .Net Programs. The code which runs under the CLR is called as **Managed Code**. Programmers need not to worry on managing the memory if the programs are running under the CLR as it provides memory management and thread management.
* .NET Class Library
* Common Language runtime
* Dynamic Language runtime
* Application domains
* .Net Framework Security
* Cross Language interoperability
* Side by side execution
* Common Type System

 What is **CLR**in .Net? What is the use of CLR?

CLR provides a managed execution environment for the *.NET* programs by improving the security, including the cross language integration and a rich set of class libraries

CLR provides the services and runtime environment to the MSIL code. Internally CLR includes the JIT(Just-In-Time) compiler which converts the MSIL code to machine code which further executed by CPU. CLR also uses the .NET Framework class libraries. Metadata provides information about the programming language, environment, version, and class libraries to the CLR by which CLR handles the MSIL code. As CLR is common so it allows an instance of a class that written in a different language to call a method of the class which written in another language.

What is the difference between  **CTS**and **CLS**

**Common Language Specification (CLS):**   
It is responsible for converting the different .NET programming language syntactical rules and regulations into CLR understandable format. Basically, it provides **Language Interoperability**. Language Interoperability means providing execution support to other programming languages also in .NET framework.

**Language Interoperability can be achieved in two ways :**

1. **Managed Code:** The MSIL code which is managed by the CLR is known as the Managed Code. For managed code CLR provides **three** .NET facilities:
2. **Unmanaged Code:** Before .NET development, programming languages like.COM Components & Win32 API do not generate the MSIL code. So these are not managed by CLR rather managed by Operating System.

**Common Type System (CTS):**   
Every programming language has its own data type system, so CTS is responsible for understanding all the data type systems of .NET programming languages and converting them into CLR understandable format which will be a common format.

*There are 2 Types of CTS that every .NET programming language have :*

1. **Value Types:** Value Types will store the value directly into the memory location. These types work with stack mechanisms only. CLR allows memory for these at Compile Time.
2. **Reference Types:** Reference Types will contain a memory address of value because the reference types won’t store the variable value directly in memory. These types work with Heap mechanism. CLR allot memory for these at Runtime.

**Garbage Collector:**   
It is used to provide the *Automatic Memory Management* feature. If there was no garbage collector, programmers would have to write the memory management codes which will be a kind of overhead on programmers.

**JIT(Just In Time Compiler):**   
It is responsible for converting the CIL(Common Intermediate Language) into machine code or native code using the Common Language Runtime environment.

**Benefits of CLR:**

* It improves the performance by providing a rich interact between programs at run time.
* Enhance portability by removing the need of recompiling a program on any operating system that supports it.
* Security also increases as it analyzes the MSIL instructions whether they are safe or unsafe. Also, the use of delegates in place of function pointers enhance the type safety and security.
* Support automatic memory management with the help of Garbage Collector.
* Provides cross-language integration because CTS inside CLR provides a common standard that activates the different languages to extend and share each other’s libraries.
* Provides support to use the components that developed in other .NET programming languages.
* Provide language, platform, and architecture independence.
* It allows easy creation of scalable and multithreaded applications, as the developer has no need to think about memory management and security issues.

What is the difference between **value type** and**reference type**?

What is **GAC**in the .Net Framework?

- GAC stands for Global Assembly Cache.  
  
- It is an area of memory reserved to store the assemblies of all .NET applications that are running on a certain machine.  
  
- It shares assemblies among multiple .NET applications.  
  
- The assemblies must have a strong name and must be publicly shared to be installed in the GAC.  
  
**Following are the reasons why it is important to install an assembly into the global assembly cache:**  
  
1. Shared location.  
2. File security.  
3. Side-by-side versioning.  
4. Additional search location.  
  
- It stores assemblies specifically designated to be shared by several applications on the computer.  
  
- An assembly must have a strong name to be installed in the GAC.

What is the **App Domain** in the .Net Framework? What is the use of it

Asp.Net introduces the concept of an Application Domain which is shortly known as AppDomain. It can be considered as a Lightweight process which is both a container and boundary. The .NET runtime uses an AppDomain as a container for code and data, just like the operating system uses a process as a container for code and data. As the operating system uses a process to isolate misbehaving code, the .NET runtime uses an AppDomain to isolate code inside of a secure boundary.  
  
The CLR can allow the multiple .Net applications to be run in a single AppDomain.

AppDomains are created using the CreateDomain method. AppDomain instances are used to load and execute assemblies (Assembly). When an AppDomain is no longer in use, it can be unloaded.  
  
public class MyAppDomain : MarshalByRefObject  
{  
    public string GetInfo()  
    {  
        return AppDomain.CurrentDomain.FriendlyName;  
    }  
}

public class MyApp

{

    public static void Main()

    {

        AppDomain apd = AppDomain.CreateDomain("Rajendrs Domain");

        MyAppDomain apdinfo = (MyAppDomain)apd.CreateInstanceAndUnwrap (Assembly.GetCallingAssembly().GetName().Name, "MyAppDomain");

        Console.WriteLine("Application Name = " + apdinfo.GetInfo());

    }

}

**Advantages**

1. Lower system cost - many application domains can be contained within a single system process.
2. Each application domain can have different security access levels assigned to them, all within a single process.
3. Code in one AppDomain cannot directly access code in another AppDomain.
4. The application in an AppDomain can be stopped without affecting the state of another AppDomain running in the same process.
5. An  Exception in on AppDomain will not affect other AppDomains or crash the entire process that hosts the AppDomains.

What is the difference between **namespace**and **assembly**

|  |  |
| --- | --- |
| **Namespace** | **Assembly** |
| Namespace is the logical naming decided at design time by the developer. | Scope for a particular type is defined at run time using Assembly. |
| Namespace contains set of unique names. | Assembly contains code of the form MSIL ( Microsoft Intermediate Language) |
| Classes available in your program will be logically grouped together under a namespace. | Logical units are physically grouped together as assembly. |
| Namespace can include multiple assemblies. | An assembly can contain types belonging to different namespaces. |
| Namespace doesn't have any classification. | Assembly can be classified as private assembly and public assembly. Private assembly is specific to a single application but shared/public assembly contains libraries which can be used by multiple applications. |
| Namespaces have to be mentioned in Project-Properties. | Assemblies need not be explicitly specified. They are automatically described in metadata and manifest files. |
| Namespaces can be nested. For example: namespace sampleApp1 { namespace SampleApp2 { class sampleClass { … } } } | Such nesting is not permissible in assemblies. |

What is the difference between **EXE**and **DLL**

Difference between exe and dll-1.EXE is an extension used for executable files while DLL is the extension for a dynamic link library. 2.An EXE file can be run independently while a DLL is used by other applications. 3.An EXE file defines an entry point while a DLL does not. 4.A DLL file can be reused by other applications while an EXE cannot. 5.A DLL would share the same process and memory space of the calling application while an EXE creates its separate process and memory space.

What is **Connection Pooling** in .Net

Connection pooling **allows you to reuse connections rather than create a new one every time the ADO.**NET data provider needs to establish a connection to the underlying database. Connection pooling behavior can be controlled by using connection string options

we would like to say that, open the connection object, do all the series of steps (socket, handshake, connection string parsed, authenticate, etc.), do the operations, and close the connection object. But when we say close, don’t make this connection object go for garbage collector rather than cache it in a pool. So that, the next time when someone says open connection, then get the connection object from the pool rather than going and executing the series of expensive steps.

string ConnectionString = "data source=LAPTOP-ICA2LCQL\\SQLEXPRESS; initial catalog=ADODB; integrated security=True; **Pooling=true**;";

What is the difference between “**throw**” and “**throw ex**”

**throw :**If we use "throw" statement, it preserve original error stack information. In exception handling "throw" with empty parameter is also called re-throwing the last exception.  
  
**throw ex :**If we use "throw ex" statement, stack trace of exception will be replaced with a stack trace starting at the re-throw point. It is used to intentionally hide stack trace information.

What is **Managed** and **Unmanaged Code**

Nested Tuples

If you want to include more than eight elements in a tuple, you can do that by nesting another tuple object as the eighth element. The last nested tuple can be accessed using the Rest property. To access the nested tuple's element, use the Rest.Item1.Item<elelementNumber> property.

Example: Nested Tuple

var numbers = Tuple.Create(1, 2, 3, 4, 5, 6, 7, Tuple.Create(8, 9, 10, 11, 12, 13));

numbers.Item1; // returns 1

numbers.Item7; // returns 7

numbers.Rest.Item1; //returns (8, 9, 10, 11, 12, 13)

numbers.Rest.Item1.Item1; //returns 8

numbers.Rest.Item1.Item2; //returns 9

## ValueTuple

ValueTuple<int, string, string> person = (1, "Bill", "Gates");

person.Item1; // returns 1

(int, string, string) person = (1, "James", "Bond");

person.Item1; // returns 1

ValueTuple can include more than eight values.

var person = (Id:1, FirstName:"Bill", LastName: "Gates");

## ValueTuple as Parameter

static void Main(string[] args)

{

DisplayTuple((1, "Bill", "Gates"));

}

static void DisplayTuple((int, string, string) person)

{

Console.WriteLine("{0}, {1}, {2}", person.Item1, person.Item2, person.Item3);

}

--------------

static void Main(string[] args)

{

var person = GetPerson();

Console.WriteLine("{0}, {1}, {2}", person.Item1, person.Item2, person.Item3);

}

static (int, string, string) GetPerson()

{

return (1, "Bill", "Gates");

}

| Generic Collections | Description |
| --- | --- |
| [List<T>](https://www.tutorialsteacher.com/csharp/csharp-list) | Generic List<T> contains elements of specified type. It grows automatically as you add elements in it. |
| [Dictionary<TKey,TValue>](https://www.tutorialsteacher.com/csharp/csharp-dictionary) | Dictionary<TKey,TValue> contains key-value pairs. |
| [SortedList<TKey,TValue>](https://www.tutorialsteacher.com/csharp/csharp-sortedlist) | SortedList stores key and value pairs. It automatically adds the elements in ascending order of key by default. |
| [Queue<T>](https://www.tutorialsteacher.com/csharp/csharp-queue) | Queue<T> stores the values in FIFO style (First In First Out). It keeps the order in which the values were added. It provides an Enqueue() method to add values and a Dequeue() method to retrieve values from the collection. |
| [Stack<T>](https://www.tutorialsteacher.com/csharp/csharp-stack) | Stack<T> stores the values as LIFO (Last In First Out). It provides a Push() method to add a value and Pop() & Peek() methods to retrieve values. |
| Hashset<T> | Hashset<T> contains non-duplicate elements. It eliminates duplicate elements. |

| Non-generic Collections | Usage |
| --- | --- |
| [ArrayList](https://www.tutorialsteacher.com/csharp/csharp-arraylist) | ArrayList stores objects of any type like an array. However, there is no need to specify the size of the ArrayList like with an array as it grows automatically. |
| [SortedList](https://www.tutorialsteacher.com/csharp/csharp-sortedlist) | SortedList stores key and value pairs. It automatically arranges elements in ascending order of key by default. C# includes both, generic and non-generic SortedList collection. |
| [Stack](https://www.tutorialsteacher.com/csharp/csharp-stack) | Stack stores the values in LIFO style (Last In First Out). It provides a Push() method to add a value and Pop() & Peek() methods to retrieve values. C# includes both, generic and non-generic Stack. |
| [Queue](https://www.tutorialsteacher.com/csharp/csharp-queue) | Queue stores the values in FIFO style (First In First Out). It keeps the order in which the values were added. It provides an Enqueue() method to add values and a Dequeue() method to retrieve values from the collection. C# includes generic and non-generic Queue. |
| [Hashtable](https://www.tutorialsteacher.com/csharp/csharp-hashtable) | Hashtable stores key and value pairs. It retrieves the values by comparing the hash value of the keys. |
| BitArray | BitArray manages a compact array of bit values, which are represented as Booleans, where true indicates that the bit is on (1) and false indicates the bit is off (0). |

# **Indexers**

An indexer is a special type of property that allows a class or a structure to be accessed like an array for its internal collection. C# allows us to define custom indexers, generic indexers, and also overload indexers.

An indexer can be defined the same way as property with this keyword and square brackets []

<return type> this[<parameter type> index]

{

get{

// return the value from the specified index of an internal collection

}

set{

// set values at the specified index in an internal collection

}

}

Ex :

class StringDataStore

{

private string[] strArr = new string[10]; // internal data storage

public string this[int index]

{

get => strArr[index];

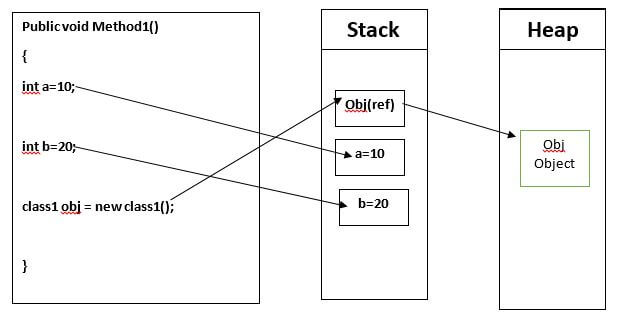
set => strArr[index] = value;

}

}

What is the difference between the **Stack**and the **Heap**

<https://www.c-sharpcorner.com/article/stack-vs-heap-memory-c-sharp/>



What are the **four pillars of the OOPS**

**1. What Is Encapsulation**  
  
Encapsulation is a process of binding data members (variables, properties) and member functions (methods) together. In object oriented programming language we achieve encapsulation through Class.  
  
**Real Life Example Of Encapsulation:**  
  
The real life example of encapsulation will be the Capsule. Capsule binds all chemical contents required for curing specific disease together just like the class which binds data members and member functions.

**2. What Is Abstraction**  
  
Abstraction is the process of showing only essential/necessary features of an entity/object to the outside world and hide the other irrelevant information.  
  
In programming language we achieve the abstraction through public and private access modifiers and a class. So in a class make things (feature) which we want to show as public and thing which are irrelevant make them as private so they won't be available to the outside world.

**3. What Is Inheritance**  
  
The process of creating the new class by extending the the existing class is called inheritance or the process of inheriting the features of base class is called as inheritance.  
  
The existing class is called the base class and new class which is created from it is called the derived class.

**What Is Polymorphism**  
  
Poly means many and Morph means forms. Polymorphism is the process in which an object or function take different forms.  
  
**Real Life Example Of Polymorphism:**  
  
Real life example of Polymorphism is mobile phone. It is a single object but it can be used for making calls, listening music, sending mails, taking pictures, etc (different forms).

##### **What are the advantages of using interfaces?**

1. Interfaces allow us to implement polymorphic behavior. Of course, abstract classes can also be used to implement polymorphic behavior.
2. The Interfaces allow us to develop very loosely coupled systems.
3. Interfaces enable mocking for better unit testing.
4. The Interfaces enable us to implement multiple [**inheritances in C#**](https://dotnettutorials.net/lesson/inheritance-c-sharp/).
5. Interfaces are great for implementing Inversion of Control or Dependency Injection.
6. The Interfaces enable parallel application development.

##### **What do you mean by “Explicitly Implementing an Interface”? Give an example?**

If a class is implementing the inherited interface member by prefixing the name of the interface, then the class is “Explicitly Implementing an Interface member”. The disadvantage of Explicitly Implementing an [**Interface**](https://dotnettutorials.net/lesson/interface-c-sharp/)member is that the class object has to be typecasted to the interface type to invoke the interface member.