Sample Question Net Centric with answers from Chapter 1.

Chapter 1.

1. **What is .net framework? List any five advantages of .net framework.**

   NET is a software framework which is designed and developed by Microsoft. The first version of the .Net framework was 1.0 which came in the year 2002. In easy words, it is a virtual machine for compiling and executing programs written in different languages like C#,VB. Net etc. It is used to develop Form-based applications, Web-based applications, and Web services. There is a variety of programming languages available on the .Net platform, VB.Net and C# being the most common ones. It is used to build applications for Windows, phone, web, etc. It provides a lot of functionalities and also supports industry standards.

   Advantages:

   Memory Management:

   In many programming languages, programmers are responsible for allocating and releasing memory and for handling object lifetimes. In .NET Framework applications, the CLR provides these services on behalf of the application.

   Common Type System:

   In traditional programming languages, basic types are defined by the compiler, which complicates cross-language interoperability. In the .NET Framework, basic types are defined by the .NET Framework type system and are common to all languages that target the .NET Framework.

   Extensive Class Library:

   Instead of having to write vast amounts of code to handle common low-level programming operations, programmers can use a readily accessible library of types and their members from the .NET Framework Class Library.

   Version compatibility

   Microsoft ensures that older versions of the .NET Framework can work seamlessly with later versions without modifications.

   Reliability

   .NET has been used to develop and run thousands of applications since its release in 2002. Despite the creation of new versions, the earlier renditions still deliver a reliable performance.

   Portability

   Applications developed on the .NET Framework can work on any Windows platform. It also has cross-platform capabilities, allowing developers to run applications on other operating systems. Third parties can create compatible implementations of the framework on other platforms using conforming languages.

Security

The .NET Framework provides a robust security mechanism that validates and verifies applications before granting the user access to the program or its source code.

Open source

Another great advantage of the .NET Framework is its open-source structure. A community of more than 60,000 programmers from thousands of companies like Google, Samsung, Red Hat and the Technical Steering Group contribute to the .NET Framework through the .NET Foundation. This supportive community improves the framework and provides support for users who may encounter technical challenges while interacting with the platform.

2. **What is .net framework? What are the main component of .net framework explain?**

.net framework explain above in question 1.

1.1 Main Components of .NET Framework

- **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.
- **Framework Class Library(FCL):** 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. Below is the overview of .NET Framework

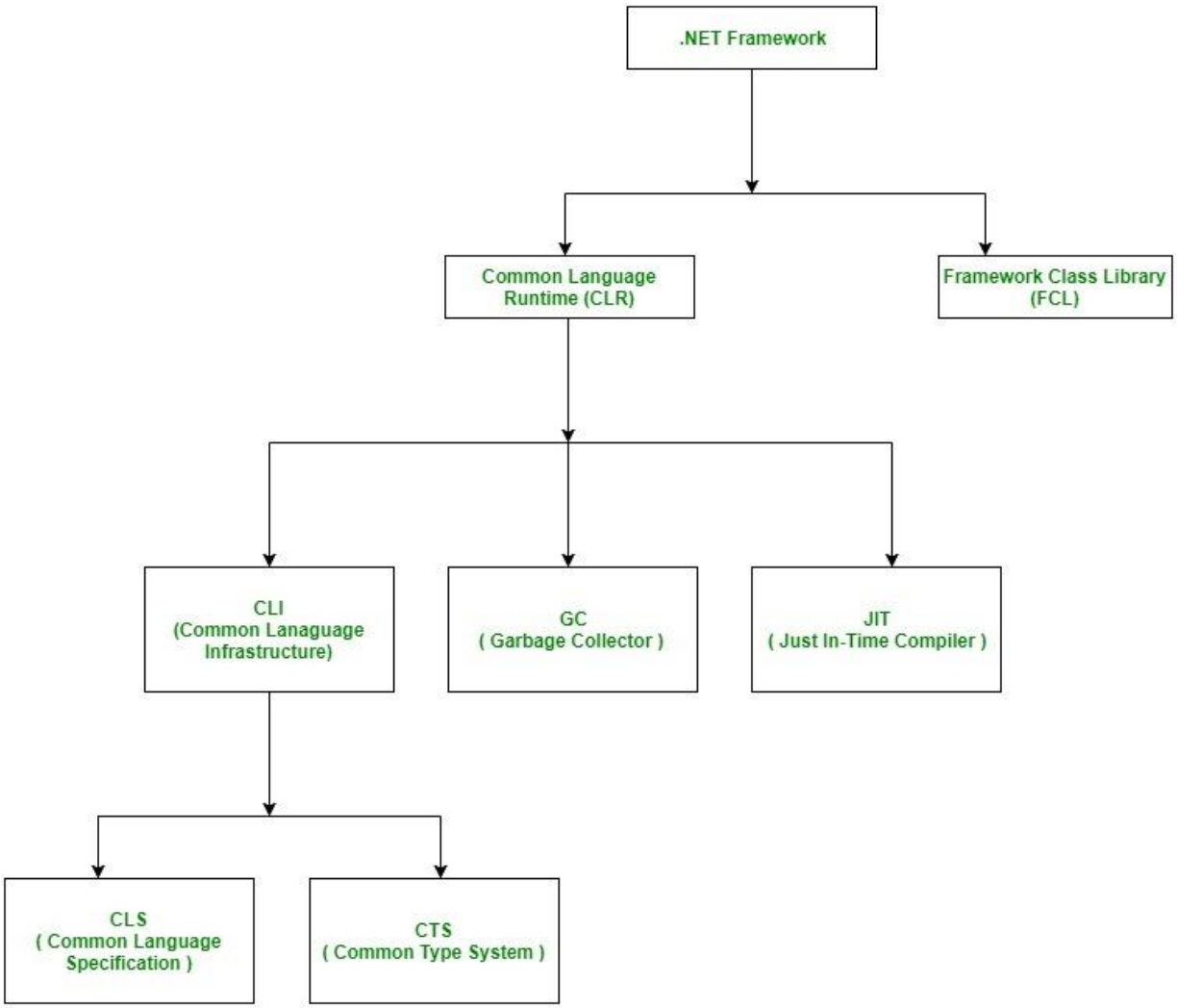


Figure 1: .Net Framework

CLR: CLR provides an environment to execute .NET applications on target machines. CLR is also a common runtime environment for all .NET code irrespective of their programming language, as the compilers of respective language in .NET Framework convert every source code into a common language known as MSIL or IL (Intermediate Language).CLR also provides various services to execute processes, such as memory management service and security services. CLR performs various tasks to manage the execution proc ess of .NET applications

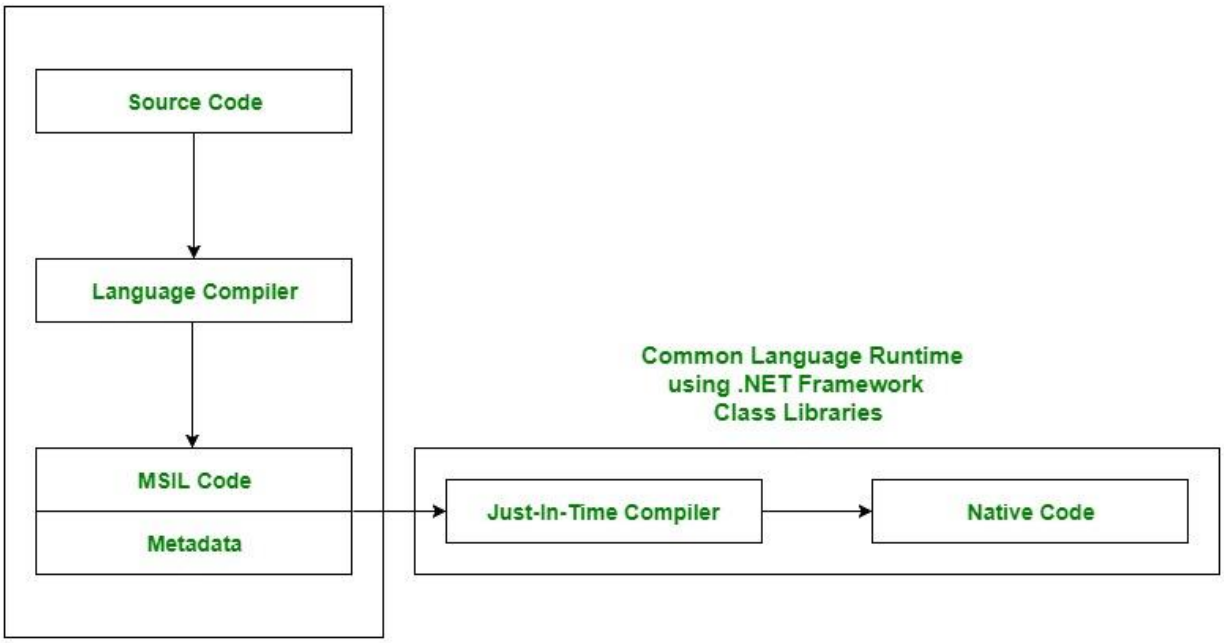


Figure 2: Common Language Runtime

Main components of CLR:

- Common Language Specification (CLS)
- Common Type System (CTS)
- Garbage Collection (GC)
- Just In – Time Compiler (JIT)

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 the Language Interoperability. Language Interoperability means to provide the 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. Managed code CLR provides **three** .NET facilities:

- CAS(Code Access Security)
- Exception Handling
- Automatic Memory Management.

2. **Unmanaged Code:** Before .NET development the programming language 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** mechanism 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 allots 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.

3.What is CLR(Common Language Runtime)? Explain main component of CLR and advantages.

CLR: CLR provides an environment to execute .NET applications on target machines. CLR is also a common runtime environment for all .NET code irrespective of their programming language, as the compilers of respective language in .NET Framework convert every source code into a common language known as MSIL or IL (Intermediate Language).CLR also provides various services to execute processes, such as memory management service and security services. CLR performs various tasks to manage the execution proc ess of .NET applications

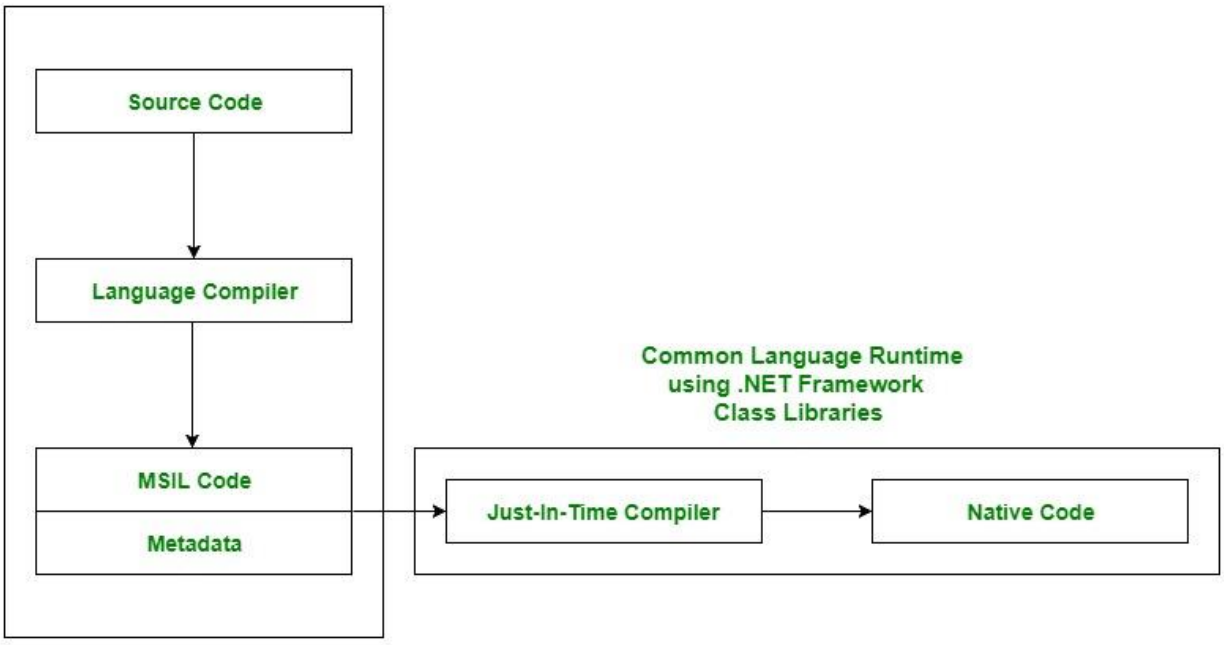


Figure 3: Common Language Runtime

Main components of CLR:

- Common Language Specification (CLS)
- Common Type System (CTS)
- Garbage Collection (GC)
- Just In – Time Compiler (JIT)

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 the Language Interoperability. Language Interoperability means to provide the execution support to other programming languages also in .NET framework.

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** mechanism 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 allots 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.

3. Explain Compilation and execution of .Net applications?

The Code Execution Process involves the following two stages:

1. Compiler time process.
2. Runtime process.

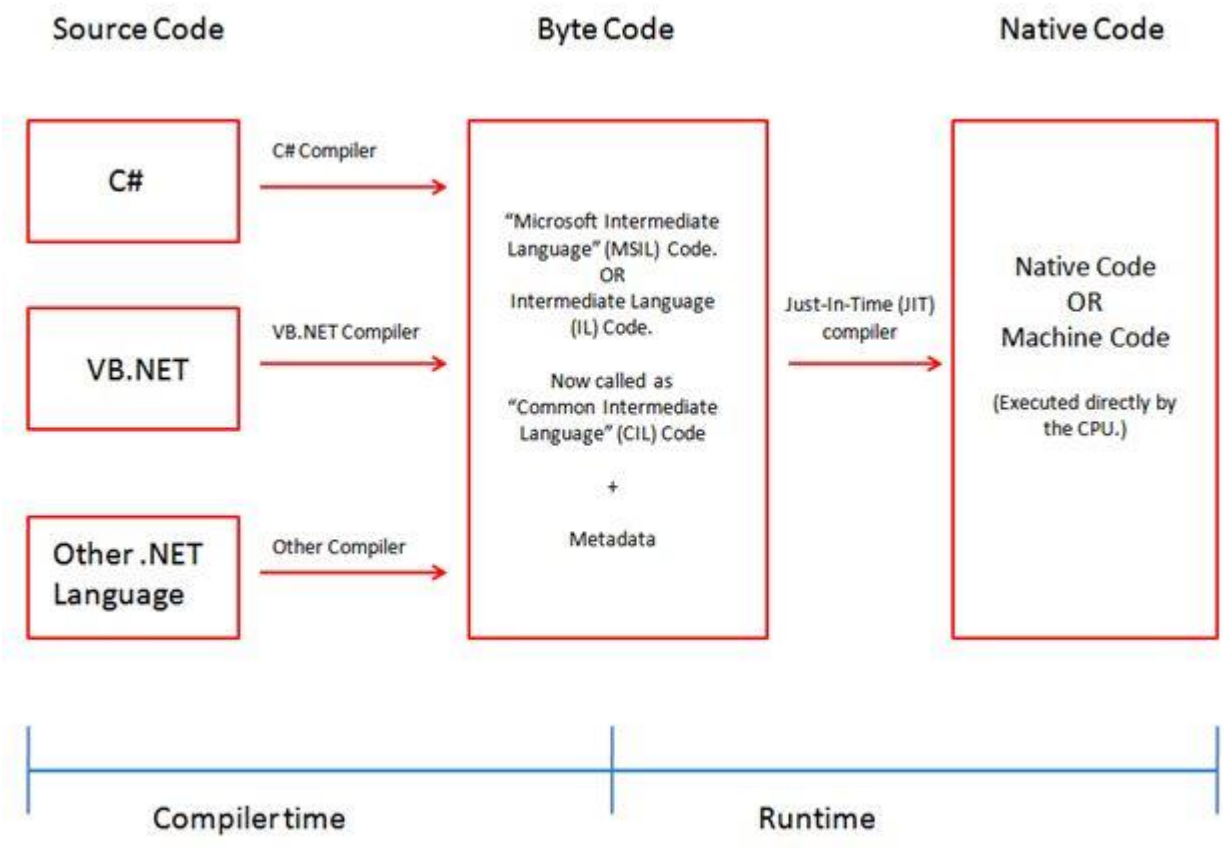


Figure 4: Code Execution Process in .net Framework

1. Compiler time process

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

SOURCE CODE -----.NET COMLIPER------> BYTE CODE (MSIL + META DATA)

2. Runtime process.

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

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

4. What is Constructor and Destructor? List advantages of using constructor in C#?/ Why

Why we use Constructor in C# and advantage of Constructor?

A constructor is a specialized function that is used to initialize fields. A constructor has the same name as the class. Instance constructors are invoked with the new operator and can't be called in the same manner as other member functions. There are some important rules pertaining to constructors as in the following;

Classes with no constructor have an implicit constructor called the default constructor, that is parameter less. The default constructor assigns default values to fields.

- A public constructor allows an object to be created in the current assembly or referencing assembly.
- Only the extern modifier is permitted on the constructor.
- A constructor returns void but does not have an explicitly declared return type.
- A constructor can have zero or more parameters.
- Classes can have multiple constructors in the form of default, parameter or both.

Syntax of a Constructor

```
class clsAddition{

          public clsAddition(){
          //Code goes here
          }
  }
```

Destructors:

The purpose of the destructor method is to remove unused objects and resources. Destructors are not called directly in the source code but during garbage collection. Garbage collection is nondeterministic. A destructor is invoked at an undetermined moment. More precisely a programmer can't control its execution; rather it is called by the Finalize () method. Like a constructor, the destructor has the same name as the class except a destructor is prefixed with a tilde (~). There are some limitations of destructors as in the following;

- Destructors are parameterless.
- A Destructor can't be overloaded.

- Destructors are not inherited.
- Destructors can cause performance and efficiency implications.

Syntax:

Public class customer

```
{
        ~customer()
        {
            Dispose();
        }
}
```

Advantages of Constructor

- Automatic initialization of objects at the time of their declaration.
- Multiple ways to initialize objects according to the number of arguments passes while declaration.
- The objects of child class can be initialized by the constructors of base class.

5. What are the different types of properties supported by C#?

Properties are the special type of class members that provides a flexible mechanism to read, write, or compute the value of a private field. Properties can be used as if they are public data members, but they are actually special methods called ***accessors***. This enables data to be accessed easily and helps to promote the flexibility and safety of methods. Encapsulation and hiding of information can also be achieved using properties. It uses pre-defined methods which are "get" and "set" methods which help to access and modify the properties.

Accessors: The block of "set" and "get" is known as "Accessors". It is very essential to restrict the accessibility of property. There are two type of accessors i.e. **get accessors** and **set accessors**. There are different types of properties based on the "get" and set accessors:

- **Read and Write Properties:** When property contains both get and set methods.
- **Read-Only Properties:** When property contains only get method.
- **Write Only Properties:** When property contains only set method.
- **Auto Implemented Properties:** When there is no additional logic in the property accessors and it introduce in C# 3.0.

The syntax for Defining Properties:

```
<access_modifier> <return_type> <property_name>
{
        get { // body }
        set { // body }
}
```

Where, <access_modifier> can be public, private, protected or internal. <return_type> can be any valid C# type. <property_name> can be user-defined.Properties can be different access modifiers like public, private, protected, internal. Access modifiers define how users of the class can access the property. The get and set accessors for the same property may have different access modifiers. A property may be declared as a **static property** by using the static keyword or may be marked as a **virtual property** by using the virtual keyword.

Read only Properties:

```
    public class Student
    {
        public string Name
        {
            get { return "Sunil Chaudhary"; }
        }
    }
```

Read Write only Properties:

```
public class Student
    {
        private string name = "";
        public string Name
        {
            get { return name; }
            set { name = value; }
        }
    }
```

Write only Properties:

```
public class Student
    {
        private string name = "";
        public string Name
        {

            set { name = value; }
        }
    }
```

Automatic Properties:

```
public class Student
    {
        public int Id { get; set; }
        public string Name { get; set; }
    }
```

6. What is array? explain any five properties and methods of array.

An array is a collection of the same type variable. Whereas a string is a sequence of Unicode characters or array of characters.

In C#, an array index starts at zero. That means the first item of an array starts at the 0th position. The position of the last item on an array will the total number of items - 1. So if an array has 10 items, the last 10th item is in 9th position.

C#, arrays can be declared as fixed-length or dynamic. A *fixed-length* array can store a predefined number of items. A *dynamic array* does not have a predefined size. The size of a *dynamic array* increases as you add new items to the array. You can declare an array of fixed length or dynamic.

Initializing Array :

```
// Initialize a fixed array
        int[] staticIntArray = new int[3] { 1, 3, 5 };
```

C# Array Properties

Property	Description
IsFixedSize	It is used to get a value indicating whether the Array has a fixed size or not.
IsReadOnly	It is used to check that the Array is read-only or not.
Length	It is used to get the total number of elements in all the dimensions of the Array.
Rank	It is used to get the rank (number of dimensions) of the Array.
SyncRoot	It is used to get an object that can be used to synchronize access to the Array.

C# Array Methods

Method	Description
Clone()	It is used to create a shallow copy of the Array.
Copy(Array,Array,Int32)	It is used to copy elements of an array into another array by specifying starting index.
CopyTo(Array,Int32)	It copies all the elements of the current one-dimensional array to the specified one-

	dimensional array starting at the specified destination array index
Empty<T>()	It is used to return an empty array.
IndexOf(Array,Object)	It is used to search for the specified object and returns the index of its first occurrence in a one-dimensional array.
Initialize()	It is used to initialize every element of the value-type Array by calling the default constructor of the value type.
Reverse(Array)	It is used to reverse the sequence of the elements in the entire one-dimensional Array.
Sort(Array)	It is used to sort the elements in an entire one-dimensional Array.
ToString()	It is used to return a string that represents the current object.

6.What is Indexer in C#? Difference between Indexers and Properties

C# indexers are usually known as smart arrays. A C# indexer is a class property that allows you to access a member variable of a class or struct using the features of an array. In C#, indexers are created using this keyword. Indexers in C# are applicable on both classes and structs.

Array access operator i.e (**[ ]**) is used to access the instance of the class which uses an indexer. A user can retrieve or set the indexed value without pointing an instance or a type member. **Indexers** are almost similar to the **Properties**.

Syntax:

```
[access_modifier] [return_type] this [argument_list]
{
  get
  {
     // get block code
  }
  set
  {
    // set block code
  }
}
```

In the above syntax:

- **access_modifier:** It can be public, private, protected or internal.
- **return_type:** It can be any valid C# type.
- **this:** It is the keyword which points to the object of the current class.
- **argument_list:** This specifies the parameter list of the indexer.

- **get{ } and set { }:** These are the **accessors**.

Indexers	Properties
Indexers are created with this keyword.	Properties don't require this keyword.
Indexers are identified by signature.	Properties are identified by their names.
Indexers are accessed using indexes.	Properties are accessed by their names.
Indexer are instance member, so can't be static.	Properties can be static as well as instance members.
A get accessor of an indexer has the same formal parameter list as the indexer.	A get accessor of a property has no parameters.
A set accessor of an indexer has the same formal parameter list as the indexer, in addition to the value parameter.	A set accessor of a property contains the implicit value parameter.

7.Does C# supports multiple inheritance? How can you implement multiple inheritance in C#?

C# does not support multiple inheritance , because **they reasoned that adding multiple inheritance added too much complexity to C# while providing too little benefit**. In C#, the classes are only allowed to inherit from a single parent class, which is called single inheritance .

Multiple Inheritance can be achieved in **C#** using Interfaces. This is the simple mathematical operation program demonstrating how **multiple inheritance** can be achieved in **C#** using Interface Concept. Here are more articles on **inheritance** and object oriented programming in **C#**

Below Example shows how to implement multiple inheritance

```
class Program
    {
        static void Main(string[] args)
        {
            C cee = new C();
            Console.WriteLine(cee.MethodA());
            Console.WriteLine(cee.MethodB());
            Console.ReadLine();
        }
    }
    interface IA
    {
        string MethodA();

    }
    interface IB
    {
        string MethodB();

    }
```

```
class C : IA, IB
{
    public string MethodA()
    {
        return "MethodA";
    }

    public string MethodB()
    {
        return "MethodB";
    }
}
```

8.what is polymorphism? Explain overriding in C#

Polymorphism is the ability to treat the various objects in the same manner. It is one of the significant benefits of inheritance. We can decide the correct call at runtime based on the derived type of the base reference. This is called late binding.

In the following example, instead of having a separate routine for the hrDepart, itDepart and financeDepart classes, we can write a generic algorithm that uses the base type functions. The method LeaderName() declared in the base abstract class is redefined as per our needs in 2 different classes.

Example:

```
public abstract class Employee
 {
     public virtual void LeaderName()
     {
     }
 }

 public class hrDepart : Employee
 {
     public override void LeaderName()
     {
         Console.WriteLine("Mr. jone");
     }
 }
 public class itDepart : Employee
 {
     public override void LeaderName()
     {
         Console.WriteLine("Mr. Tom");
     }
 }

 public class financeDepart : Employee
 {
     public override void LeaderName()
     {
         Console.WriteLine("Mr. Linus");
     }
 }

 class Program
 {
     static void Main(string[] args)
     {
         hrDepart obj1 = new hrDepart();
         itDepart obj2 = new itDepart();
         financeDepart obj3 = new financeDepart();

         obj1.LeaderName();
         obj2.LeaderName();
         obj3.LeaderName();

         Console.ReadLine();
     }
 }
```

Virtual Methods: By declaring a base class function as virtual, you allow the function to be overridden in any derived class. The idea behind a virtual function is to redefine the implementation of the base class method in the derived class as required. If a method is virtual in the base class then we have to provide the override keyword in the derived class.

9.What is delegates and event? Different between delegates and events.

Delegates and Events

Delegates:

In C# delegates are used as a function pointer to refer a method. It is specifically an object that refers to a method that is assigned to it. The same delegate can be used to refer different methods, as it is capable of holding the reference of different methods but, one at a time. Which method will be invoked by the delegate is determined at the runtime. The syntax of declaring a delegate is as follow:

```
delegate return_type delegate_name(parameter_list);
```

Events:

Events are the action performed which changes the state of an object. Events are declared using delegates, without the presence of delegates you can not declare events. You can say that an event provides encapsulation to the delegates.

```
event event_delegate event_name;
```

Below are some differences between the Delegates and Interfaces in C#:

DELEGATE	INTERFACE
It could be a method only.	It contains both methods and properties.
It can be applied to one method at a time.	If a class implements an interface, then it will implement all the methods related to that interface.
Delegates can me implemented any number of times.	Interface can be implemented only one time.
It is used to handling events.	It is not used for handling events.
It can access anonymous methods.	It can not access anonymous methods.
It does not support inheritance.	It supports inheritance.
It created at run time.	It created at compile time.

10.What is Abstract class? How it is different from interface.

An abstract class is an incomplete class or special class we can't be instantiated. The purpose of an abstract class is to provide a blueprint for derived classes and set some rules what the derived classes must implement when they inherit an abstract class.

We can use an abstract class as a base class and all derived classes must implement abstract definitions. An abstract method must be implemented in all non-abstract classes using the override keyword. After overriding the abstract method is in the non-Abstract class. We can derive this class in another class and again we can override the same abstract method with it.

Syntax:

```
public abstract class Shape
    {
        public abstract void draw();
    }
```

Difference between Abstract Class and Interface

ABSTRACT CLASS	INTERFACE
It contains both declaration and definition part.	It contains only a declaration part.
Multiple inheritance is not achieved by abstract class.	Multiple inheritance is achieved by interface.
It contain constructor.	It does not contain constructor.
It can contain static members.	It does not contain static members.
It can contain different types of access modifiers like public, private, protected etc.	It only contains public access modifier because everything in the interface is public.
The performance of an abstract class is fast.	The performance of interface is slow because it requires time to search actual method in the corresponding class.
It is used to implement the core identity of class.	It is used to implement peripheral abilities of class.
A class can only use one abstract class.	A class can use multiple interface.

If many implementations are of the same kind and use common behavior, then it is superior to use abstract class.	If many implementations only share methods, then it is superior to use Interface.
Abstract class can contain methods, fields, constants, etc.	Interface can only contain methods .
It can be fully, partially or not implemented.	It should be fully implemented.

11. why use partial class in c#?

Typically, a class will reside entirely in a single file. However, in situations where multiple developers need access to the same class, then having the class in multiple files can be beneficial. The partial keywords allow a class to span multiple source files. When compiled, the elements of the partial types are combined into a single assembly.

There are some rules for defining a partial class as in the following;

- A partial type must have the same accessibility.
- Each partial type is preceded with the "partial" keyword.
- If the partial type is sealed or abstract then the entire class will be sealed and abstract.

Example:

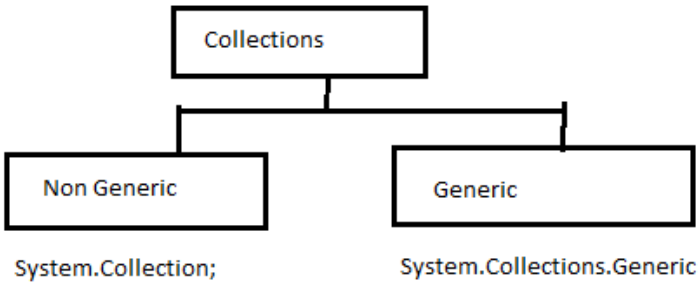
```
public partial class partialclassDemo
    {
        public void method1()
        {
            Console.WriteLine("method from part1 class");
        }
    }
    public partial class partialclassDemo

    {
        public void method2()
        {
            Console.WriteLine("method from part2 class");
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            //partial class instance
            partialclassDemo obj = new partialclassDemo();
            obj.method1();
            obj.method2();
            Console.ReadLine();
        }
    }
```

12.what is collection? Explain types of collection available in C#?

Collection

Collections standardize the way of which the objects are handled by your program. In other words, it contains a set of classes to contain elements in a generalized manner. With the help of collections, the user can perform several operations on objects like the store, update, delete, retrieve, search, sort etc.



Generic collection in C# is defined in `System.Collection.Generic` namespace. It provides a generic implementation of standard data structure like linked lists, stacks, queues, and dictionaries. These collections are type-safe because they are generic means only those items that are type-compatible with the type of the collection can be stored in a generic collection, it eliminates accidental type mismatches. Generic collections are defined by the set of interfaces and classes. Below table contains the frequently used classes of the `System.Collections.Generic` namespace:

CLASS NAME	DESCRIPTION
Dictionary<TKey,TValue>	It stores key/value pairs and provides functionality similar to that found in the non-generic Hashtable class.
List<T>	It is a dynamic array that provides functionality similar to that found in the non-generic ArrayList class.
Queue<T>	A first-in, first-out list and provides functionality similar to that found in the non-generic Queue class.
SortedList<TKey,TValue>	It is a sorted list of key/value pairs and provides functionality similar to that found in the non-generic SortedList class.
Stack<T>	It is a first-in, last-out list and provides functionality similar to that found in the non-generic Stack class.
HashSet<T>	It is an unordered collection of the unique elements. It prevent duplicates from being inserted in the collection.
LinkedList<T>	It allows fast inserting and removing of elements. It implements a classic linked list.

Non-Generic collection in C# is defined in `System.Collections` namespace. It is a general-purpose data structure that works on object references, so it can handle any type of object, but not in a safe-type manner. Non-generic collections are defined by the set of interfaces and classes. Below table contains the frequently used classes of the `System.Collections` namespace:

CLASS NAME	DESCRIPTION
ArrayList	It is a dynamic array means the size of the array is not fixed, it can increase and decrease at runtime.
Hashtable	It represents a collection of key-and-value pairs that are organized based on the hash code of the key.
Queue	It represents a first-in, first out collection of objects. It is used when you need a first-in, first-out access of items.
Stack	It is a linear data structure. It follows LIFO(Last In, First Out) pattern for Input/output.

Generic Collections:
Generic Collections work on the specific type that is specified in the program whereas non-generic collections work on the object type.

- Specific type
- Array Size is not fixed
- Elements can be added / removed at runtime.

13. what is file IO? Perform basic create,delete,reading,writing,copy operation in textfile with example
Generally, the file is used to store the data. The term File Handling refers to the various operations like creating the file, reading from the file, writing to the file, appending the file, etc. There are two basic operation which is mostly used in file handling is reading and writing of the file. The file becomes stream when we open the file for writing and reading. A stream is a sequence of bytes which is used for communication. Two stream can be formed from file one is input stream which is used to read the file and another is output stream is used to write in the file. In C#, *System.IO* namespace contains classes which handle input and output streams and provide information about file and directory structure.

Example Create,Copy,Delete,Read,Write File using System.IO
Creating File:

```
class Program
    {
        static void Main(string[] args)
        {
            string fileLoc = @"E:\sample1.txt";
            FileStream fs = null;
            if (!File.Exists(fileLoc))
            {
                using (fs = File.Create(fileLoc))
                {

                }
            }
            Console.WriteLine("File Created");
            Console.ReadLine();
        }
    }
```

Writing in File:

```
class Program
    {
        static void Main(string[] args)
        {
            string fileLoc = @"E:\sample1.txt";
            if (File.Exists(fileLoc))
            {
                using (StreamWriter sw = new StreamWriter(fileLoc))
                {
                    sw.Write("Some sample text for the file");
                }
            }
            Console.WriteLine("Write is Success");
            Console.ReadLine();
        }
    }
```

Reading From File:

```
class Program
    {
        static void Main(string[] args)
        {
            string fileLoc = @"E:\sample1.txt";
            if (File.Exists(fileLoc))
            {
                using (TextReader tr = new StreamReader(fileLoc))
                {
                    Console.WriteLine(tr.ReadLine());
                }
            }
            Console.ReadLine();
        }
    }
```

Copy a Text File

```
 class Program
    {
        static void Main(string[] args)
        {
            string fileLoc = @"E:\sample1.txt";
            string fileLocCopy = @"F:\sample1.txt";
            if (File.Exists(fileLoc))
            {
                // If file already exists in destination, delete it.
                if (File.Exists(fileLocCopy))
                    File.Delete(fileLocCopy);
                File.Copy(fileLoc, fileLocCopy);
            }
            Console.WriteLine("File Copied");
            Console.ReadLine();
        }
    }
```

Delete a Text File

```
class Program
    {
        static void Main(string[] args)
        {
            string fileLoc = @"E:\sample1.txt";

            if (File.Exists(fileLoc))
            {
                File.Delete(fileLoc);
            }
            Console.WriteLine("File Deleted");
            Console.ReadLine();
        }
    }
```

14.What is LINQ? Draw architecture of LINQ and explain advantages of LINQ

LINQ in C# is used to work with data access from sources such as objects, data sets, SQL Server, and XML. LINQ stands for Language Integrated Query. LINQ is a data querying API with SQL like query syntaxes. LINQ provides functions to query cached data from all kinds of data sources. The data source could be a collection of objects, database or XML files. We can easily retrieve data from any object that implements the IEnumerable<T> interface.

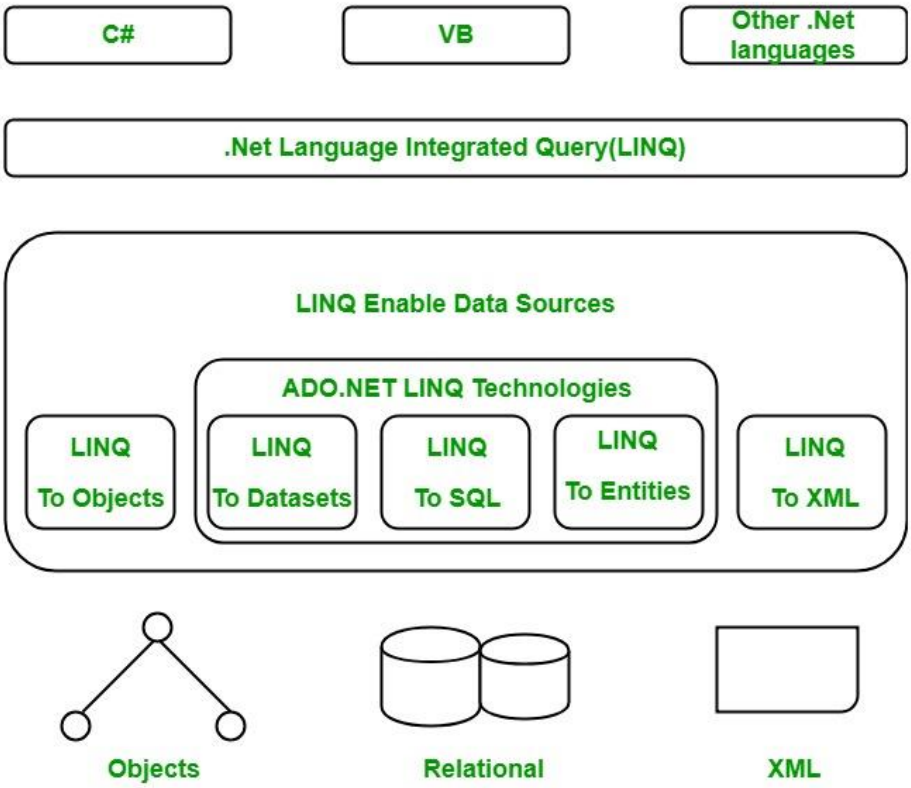


Figure 5: Architecture of LINQ

Advantages of LINQ

User does not need to learn new query languages for a different type of data source or data format.

- It increase the readability of the code.
- Query can be reused.
- It gives type checking of the object at compile time.
- It provides IntelliSense for generic collections.
- It can be used with array or collections.
- LINQ supports filtering, sorting, ordering, grouping.
- It makes easy debugging because it is integrated with C# language.
- It provides easy transformation means you can easily convert one data type into another data type like transforming SQL data into XML data.

15. Why Lambda expression is important in C#?

Lambda expressions are anonymous functions that contain expressions or sequence of operators. All lambda expressions use the lambda operator =>, that can be read as “goes to” or “becomes”. The left side of the lambda operator specifies the input parameters and the right side holds an expression or a code block that works with the entry parameters. Usually lambda expressions are used as predicates or instead of delegates (a type that references a method).

Advantages:

1. Lambda Expression is a shorter way of representing anonymous method.
2. Lambda Expression can have zero or more parametes.
3. Lambda expressions themselves do not have type, so CLR doesn’t support the concept of Lambda expression.
4. Variables defined within a lambda expression are accessible only within the body of lambda expression.
5. Lambda expression can have multiple statements in body expression in curly brackets {}.
6. With in lambda expressions, it is possible to access variables present outside of the lambda expression block by a feature known as closure but vice versa is not possible.
7. Unsafe code inside any lambda expression can’t be used.
8. The local variable in lambda expressions is not to be garbage collected unless the delegate referencing the same becomes eligible for the act of garbage collection.
9. Lambda expressions can also use with Func and Action delegates.

16.why exceptional handling is done in C#? how runtime error is handled using multiple catch block?

Try statements and Exceptions

try-catch statement is useful to handle unexpected or runtime exceptions which will occur during the execution of the program. The **try-catch** statement will contain a **try** block followed by one or more **catch** blocks to handle different exceptions.

In c#, whenever an exception occurred in the **try** block, then the CLR (common language runtime) will look for the **catch** block that handles an exception. In case, if the currently executing method does not contain such a **catch** block, then the CLR will display an unhandled exception message to the user and stops the execution of the program.

Syntax:

```
 try
{
    // put the code here that may raise exceptions
}
Catch(Excetion ex)
{
    // handle exception here
  Throw ex;
}
finally
{
    // final cleanup code
}
```

Keyword	Definition
try	Used to define a try block. This block holds the code that may throw an exception.
catch	Used to define a catch block. This block catches the exception thrown by the try block.
finally	Used to define the finally block. This block holds the default code.
throw	Used to throw an exception manually.

Example: (DivideByZeroException)

```
class Program
    {
        static void Main(string[] args)
        {
            try
            {
                int i = 20;
                // Suspect code
                int result = i / 0;
            }
            catch (DivideByZeroException ex)
            {
                Console.WriteLine("Attempted divide by zero. {0}", ex.Message);
            }

        }

    }
```

Example: (Multiple Catch Block)

```
class Program
    {

        static void Main(string[] args)
        {
            try
            {

                int i = 20; int j = 0;
                double result = i/ j;

                object obj = default;
                int i2 = (int)obj; // Suspect of casting error

            }
            catch (StackOverflowException ex)
            {
                Console.WriteLine("Overflow. {0}", ex.Message);
            }
            catch (DivideByZeroException ex)
            {
                Console.WriteLine("Attempted divide by zero. {0}", ex.Message);
            }
            catch (InvalidCastException ex)
            {
                Console.WriteLine("Invalid casting. {0}", ex.Message);
            }
Catch(Exception ex)
{
}
Finally
{
}

            Console.ReadLine();

        }

    }
```

17. What is role of attribute in C#?explain different types of Attributes implementations provided by the *.NET Framework*?

Attributes are used in C# to convey declarative information or metadata about various code elements such as methods, assemblies, properties, types, etc. Attributes are added to the code by using a declarative tag that is placed using square brackets ([ ]) on top of the required code element.

[[*target*:]? *attribute-name* (

positional-param+ |

[*named-param* = *expr*]+ |

positional-param+, [*named-param* = *expr*]+)?]

There are two types of Attributes implementations provided by the *.NET Framework* are:

1. Predefined Attributes
2 Custom Attributes

Properties of Attributes:

- Attributes can have arguments just like methods, properties, etc. can have arguments.
- Attributes can have zero or more parameters.
- Different code elements such as methods, assemblies, properties, types, etc. can have one or multiple attributes.
- Reflection can be used to obtain the metadata of the program by accessing the attributes at run-time.
- Attributes are generally derived from the **System.Attribute Class**.

Predefined attributes are those attributes that are a part of the .NET Framework Class Library and are supported by the C# compiler for a specific purpose. Some of the predefined attributes that are derived from the *System.Attribute* base class are given as follows:

Attributes	Description
[Serialization]	By marking this attributes, a class is able to persist its current state into stream.
[NonSerialization]	It specify that a given class or filed should not persisted during the serialization process.
[Obsolete]	It is used to mark a member or type as deprecated. If they are attempted to be used somewhere else then compiler issues a warning message.
[DllImport]	This allows .NET code to make call an unmanaged C or C++ library.
[WebMethod]	This is used to build XML web services and the marked method is being invoked by HTTP request.
[CLSCompliant]	Enforce the annotated items to conform to the semantics of CLS.

18. explain uses of Asynchronous Programming?

With visual studio 2012, now developers can use 'await' and 'async' keywords. Async/await methods are sequential in nature, but asynchronous when compiled and executed. The added benefit of using the 'async' keyword is that it provides a simpler way to perform potentially long-running operations without blocking the callers thread. The caller thread/method can continue its work without waiting for this asynchronous method to complete its job. This approach reduces additional code development effort. Each 'async' modifier/method should have at least one 'await'.

The asynchronous model works best when:

- There is a large number of tasks, so there is probably always at least one task that can make progress;
- Tasks perform many I/O operations, which causes the synchronous program to spend a lot of time in blocking mode when other tasks can be performed;
- Tasks are largely independent of each other, so there is no need for intertask interaction (and therefore, no need to wait for one task from another).

Async

This keyword is used to qualify a function as an asynchronous function. In other words, if we specify the async keyword in front of a function then we can call this function asynchronously. Have a look at the syntax of the asynchronous method.

```
public static async void CallProcess()
 {
     await LongProcess();
    Console.WriteLine("Long Process finish");
 }
```

Await

This keyword is used when we want to call any function asynchronously. Have a look at the following example to understand how to use the await keyword.

```
public static Task LongProcess()
{
    return Task.Run(() =>
    {
        System.Threading.Thread.Sleep(5000);
    });
}
```