Question list:

1) What is C-Sharp (C#)?
2) What is the execution entry point for a C# console application?
3) Explain variable in C# Dot Net.
4) Why to use “using” in C#?
5) Explain namespaces in C#?
6) Explain the types of comments in C#?
7) Explain ‘Params’ Keyword in C#.
8) Difference between if and #if?
9) What is the “this” Pointer?
10) What is a String? What are the properties of a String Class?
11) Why are strings in C# immutable?
12) What is an Escape Sequence? Name some String escape sequences in C#.
13) How do you initiate a string without escaping each backslash?
14) What are Regular expressions? Search a string using regular expressions?
15) What are the basic String Operations? Explain.
16) What is Parsing? How to Parse a Date Time String?
17) What is an Array? Give the syntax for a single and multi-dimensional array?
18) Name some properties of Array.
19) What is an Array Class?
20) List out the differences between Array and Array List in C#?
21) Explain Jagged Arrays in C#?
22) Differentiate between boxing and unboxing.
23) What are cookies?
24) Name some of the disadvantages of cookies.
25) Explain state management in ASP.NET.
26) What are the key differences between function and stored procedure in .Net programming language?
27) How would you retrieve user names for Windows Authentication?
28) Name the advantages of using Session State.
29) What is HTTPhandler?
30) What is Garbage Collector in .NET?
31) Name the methodology used to enforce garbage collection in .NET.
32) What is a Destructor in C#?
33) How many types of indexes are there in .NET?
34) How many types of memories do exist in .Net?
35) Mention the divisions of the Memory Heap?
36) What is LINQ?
37) Name different types of constructors in C#.
38) What are the different events in the Page Life Cycle?
39) Name all the templates of the Repeater control.
40) What code we can use to send e-mail from an ASP.NET application?
41) Mention the design principles used in .NET?

42) What is Cohesion?
43) Explain Coupling in C#?
44) What is IL?
45) What is Managed and Unmanaged code?
46) How do we execute managed code?
47) Explain Code compilation in C#.
48) Name the different types of assemblies?
49) Mention the different parts of an Assembly?
50) What is ADO (ActiveX Data Objects)?
51) List the fundamental objects in ADO.NET?
52) What is a PE file?
53) What is the difference between DLL and EXE?
54) What is Just In Time (JIT)?
55) What is Caching in ASP.NET?
56) What do you mean by OOP concept?
57) What is the difference between the Virtual method and the Abstract method?
58) what do we need to use static in c#?
59) Can a static method access non-static fields?
60) If in a static class, we don't have non static members/functions then does it hold any default constructor?
61) Can static constructor have public/private/internal/protected internal access modifiers?
62) What is the difference between a struct and a class?
63) Are private class members inherited to the derived class?
64) How do you prevent a class from being inherited?
65) What is the difference between Continue and Break Statement?
66) How is Exception Handling implemented in C#?
67) Can we have only "try" block without "catch" block in C#?
68) What is the difference between finally and finalize block?
69) Why to use "finally" block in C#?
70) What you mean by inner exception in C#?
71) What is the difference between "out" and "ref" parameters in C#?
72) What is C# I/O Classes? What are the commonly used I/O Classes?
73) What is StreamReader/StreamWriter class?
74) Explain Delegates in C#?

Answer:

#1 What is C-Sharp (C#)?

C# is a type-safe, managed and object-oriented language, which is compiled by .Net framework for generating intermediate language (IL).

What is the execution entry point for a C# console application?

The Main method.

Explain variable in C# Dot Net.

Variable is a type of container which represent storge locations in the memory. Every variable has a type that determines what values can be storage in the variable.

In C#, a variable is declared like this:

<data type> <name>;

An example could like this:

String name;

That's the most basic version, but the variable doesn't yet have a value. You can assign one at a later point or at the same time as declaring it, like this:

<data type> <name> = <value>;

If this variable is not local to the method you're currently working in (e.g., a class member variable), you might want to assign a visibility to the variable:

<visibility> <data type> <name> = <value>;

And a complete example:

```
private string name = "John Doe";
```

There are different types of variables like that…

- Local Variables
- Instance Variables or Non-Static variables
- Static Variables or Class Variables
- Constant Variables
- Readonly Variables

Local Variables:

A variable defined within a block or method or constructor is called local variable.

- These variables are created when the block is entered or the function is called and destroyed after exiting from the block or when the call returns from the function.
- The scope of these variables exists only within the block in which the variable is declared. i.e., we can access these variables only within that block.

Example 1:

```
// C# program to demonstrate
// the local variables
using System;
class StudentDetails {
        // Method
        public void StudentAge(){
                // local variable age
                int age = 0;
                age = age + 10;
                Console.WriteLine("Student age is : " + age);
        }
        // Main Method
        public static void Main(String[] args){
                // Creating object
                StudentDetails obj = new StudentDetails();
                // calling the function
                obj.StudentAge();
        }
}
```

Output:

Student age is : 10

Explanation :In the above program, the variable “age” is a local variable to the function StudentAge(). If we use the variable age outside StudentAge() function, the compiler will produce an error as shown in below program.

Example 2:

```
// C# program to demonstrate the error
// due to using the local variable
// outside its scope
using System;
class StudentDetails {
        // Method
```

```
        public void StudentAge(){
                // local variable age
                int age = 0;
                age = age + 10;
        }
        // Main Method
        public static void Main(String[] args){
                // using local variable age outside it's scope
                Console.WriteLine("Student age is : " + age);
        }
}
```

Error:

prog.cs(22,43): error CS0103: The name `age' does not exist in the current context

Instance Variables or Non-Static variables:

Instance variables are non-static variables and are declared in a class but outside any method, constructor or block. As instance variables are declared in a class, these variables are created when an object of the class is created and destroyed when the object is destroyed. Unlike local variables, we may use access specifiers for instance variables.

Example:

```
// C# program to illustrate the
// Instance variables
using System;
class Marks {
        // These variables are instance variables.
        // These variables are in a class and
        // are not inside any function
        int engMarks;
        int mathsMarks;
        int phyMarks;
        // Main Method
        public static void Main(String[] args){
                // first object
```

```
            Marks obj1 = new Marks();
            obj1.engMarks = 90;
            obj1.mathsMarks = 80;
            obj1.phyMarks = 93;
            // second object
            Marks obj2 = new Marks();
            obj2.engMarks = 95;
            obj2.mathsMarks = 70;
            obj2.phyMarks = 90;
            // displaying marks for first object
            Console.WriteLine("Marks for first object:");
            Console.WriteLine(obj1.engMarks);
            Console.WriteLine(obj1.mathsMarks);
            Console.WriteLine(obj1.phyMarks);
            // displaying marks for second object
            Console.WriteLine("Marks for second object:");
            Console.WriteLine(obj2.engMarks);
            Console.WriteLine(obj2.mathsMarks);
            Console.WriteLine(obj2.phyMarks);
        }
}
```

Output:

```
Marks for first object:
90
80
93
Marks for second object:
95
70
90
```

Explanation: In the above program the variables, engMarks, mathsMarks, phyMarks are instance variables. If there are multiple objects as in the above program, each object will have its own copies of instance variables. It is clear from the above output that each object will have its own copy of the instance variable.

Static Variables or Class Variables:

Static variables are also known as Class variables. If a variable is explicitly declared with the static modifier or if a variable is declared under any static block then these variables are known as static variables.

- These variables are declared similarly as instance variables, the difference is that static variables are declared using the static keyword within a class outside any method constructor or block.
- Unlike instance variables, we can only have one copy of a static variable per class irrespective of how many objects we create.
- Static variables are created at the start of program execution and destroyed automatically when execution ends.

Note: To access static variables, there is no need to create any object of that class, simply access the variable as:

class_name.variable_name;

Example:

```
// C# program to illustrate

// the static variables

using System;

class Emp {

        // static variable salary

        static double salary;

        static String name = "Aks";

        // Main Method

        public static void Main(String[] args){

                // accessing static variable

                // without object

                Emp.salary = 100000;

                Console.WriteLine(Emp.name + "'s average salary:"+ Emp.salary);

        }

}
```

Output:

Aks's average salary:100000

Note: Initialization of non-static variables is associated with instance creation and constructor calls, so non-static variables can be initialized through the constructor also. We don't initialize a static variable through constructor because every time constructor call, it will override the existing value with a new value.

Difference between Instance variable & Static variable

- Each object will have its own copy of instance variable whereas We can only have one copy of a static variable per class irrespective of how many objects we create.
- Changes made in an instance variable using one object will not be reflected in other objects as each object has its own copy of instance variable. In the case of static, changes will be reflected in other objects as static variables are common to all object of a class.
- We can access instance variables through object references and Static Variables can be accessed directly using class name.
- In the life cycle of a class a static variable i.e. initialized one and only one time, whereas instance variables are initialized for 0 times if no instance is created and n times if n instances are created.

The Syntax for static and instance variables are:

```
class Example
    {
       static int a; // static variable
       int  b;    // instance variable
    }
```

Constants Variables:

If a variable is declared by using the keyword "const" then it as a constant variable and these constant variables can't be modified once after their declaration, so it's must initialize at the time of declaration only.

Example 1:

Below program will show the error because no value is provided at the time of constant variable declaration.

```
// C# program to illustrate the
// constant variables
using System;
class Program {
        // constant variable max
        // but no value is provided
```

```
        const float max;
        // Main Method
        public static void Main(){
                // creating object
                Program obj = new Program();
                // it will give error
                Console.WriteLine("The value of b is = " + Program.b);
        }
}
```

Error

prog.cs(8,17): error CS0145: A const field requires a value to be provided

[check it by the editor]

Example 2:

Program to show the use of constant variables

```
// C# program to illustrate the
// constant variable
using System;
class Program {
        // instance variable
        int a = 10;
        // static variable
        static int b = 20;
        // constant variable
        const float max = 50;
        // Main Method
        public static void Main(){
                // creating object
                Program obj = new Program();
                // displaying result
                Console.WriteLine("The value of a is = " + obj.a);
                Console.WriteLine("The value of b is = " + Program.b);
```

```
            Console.WriteLine("The value of max is = " + Program.max);
        }
}
```

Output:

The value of a is = 10

The value of b is = 20

The value of max is = 50

Important Points about Constant Variables:

- The behavior of constant variables will be similar to the behavior of static variables i.e., initialized one and only one time in the life cycle of a class and doesn't require the instance of the class for accessing or initializing.
- The difference between a static and constant variable is, static variables can be modified whereas constant variables can't be modified once it declared.

Read-Only Variables:

If a variable is declared by using the readonly keyword then it will be read-only variables and these variables can't be modified like constants but after initialization.

- It's not compulsory to initialize a read-only variable at the time of the declaration, they can also be initialized under the constructor.
- The behavior of read-only variables will be similar to the behavior of non-static variables, i.e. initialized only after creating the instance of the class and once for each instance of the class created.

Example 1:

In below program, read-only variables k is not initialized with any value but when we print the value of the variable the default value of int i.e 0 will display as follows:

```
// C# program to show the use
// of readonly variables
// without initializing it
using System;
class Program {
        // instance variable
        int a = 80;
        // static variable
        static int b = 40;
```

```
        // Constant variables
        const float max = 50;
        // readonly variables
        readonly int k;
        // Main Method
        public static void Main(){
                // Creating object
                Program obj = new Program();
                Console.WriteLine("The value of a is = " + obj.a);
                Console.WriteLine("The value of b is = " + Program.b);
                Console.WriteLine("The value of max is = " + Program.max);
                Console.WriteLine("The value of k is = " + obj.k);
        }
}
```

Output:

```
The value of a is = 80
The value of b is = 40
The value of max is = 50
The value of k is = 0
```

Example 2:

To show the initialization of readonly variable in the constructor.

```
// C# program to illustrate the
// initialization of readonly
// variables in the constructor
using System;
class Geeks {
        // instance variable
        int a = 80;
        // static variable
        static int b = 40;
        // Constant variables
        const float max = 50;
```

```
        // readonly variables
        readonly int k;
        // constructor
        public Geeks(){
                // initializing readonly
                // variable k
                this.k = 90;
        }
        // Main Method
        public static void Main(){
                // Creating object
                Geeks obj = new Geeks();
                Console.WriteLine("The value of a is = " + obj.a);
                Console.WriteLine("The value of b is = " + Geeks.b);
                Console.WriteLine("The value of max is = " + Geeks.max);
                Console.WriteLine("The value of k is = " + obj.k);
        }
}
```

Output:

```
The value of a is = 80
The value of b is = 40
The value of max is = 50
The value of k is = 90
```

Example 3:

Program to demonstrate when the readonly variable is initialized after its declaration and outside constructor:

```
// C# program to illustrate the
// initialization of readonly
// variables twice
using System;
class Geeks {
        // instance variable
```

```
        int a = 80;
        // static variable
        static int b = 40;
        // Constant variables
        const float max = 50;
        // readonly variables
        readonly int k;
        // constructor
        public Geeks(){
                // first time initializing
                // readonly variable k
                this.k = 90;
        }
        // Main Method
        public static void Main(){
                // Creating object
                Geeks obj = new Geeks();
                Console.WriteLine("The value of a is = " + obj.a);
                Console.WriteLine("The value of b is = " + Geeks.b);
                Console.WriteLine("The value of max is = " + Geeks.max);
                // initializing readonly variable again
                // will compile time error
                obj.k = 55;
                Console.WriteLine("The value of k is = " + obj.k);
        }
}
```

Error:

prog.cs(41,13): error CS0191: A readonly field `Geeks.k' cannot be assigned to (except in a constructor or a variable initializer)

Important Points about Read-Only Variables:

- The only **difference between read-only and instance variables** is that the instance variables can be modified but read-only variable can't be modified.

- Constant variable is a fixed value for the whole class whereas read-only variables is a fixed value specific to an instance of class.

Why to use “using” in C#?

“Using” statement calls – “dispose” method internally, whenever any exception occurred in any method call and in “Using” statement objects are read only and cannot be reassign-able or modifiable.

Explain namespaces in C#?

Namespaces are containers for the classes. We will use namespaces for grouping the related classes in C#. “Using” keyword can be used for using the namespace in another namespace.

Explain the types of comments in C#?

Below are the types of comments in C#:

Single Line Comment Eg: //

Multiline Comments Eg: /* */

XML Comments Eg: ///

Explain ‘Params’ Keyword in C#.

By using the params keyword, you can specify a method parameter that takes a variable number of arguments. The parameter type must be a single-dimensional array.

No additional parameters are permitted after the params keyword in a method declaration, and only one params keyword is permitted in a method declaration.

If the declared type of the params parameter is not a single-dimensional array, compiler error CS0225 occurs.

When you call a method with a params parameter, you can pass in:

A comma-separated list of arguments of the type of the array elements.

An array of arguments of the specified type.

No arguments. If you send no arguments, the length of the params list is zero.

Example with code:

```
using System;

public class Program {

    public static void Main(string[] args){

        Program p = new Program();
```

```
        p.print(2, 3, 8);
        int[] arr = { 2, 11, 15, 20 };
        p.print(arr);
        Console.ReadLine();
    }
    public void print(params int[] b){
        foreach (int i in b) {
            Console.WriteLine(i);
        }
    }
}
```

Difference between if and #if?

The #if condition is evaluated or decided only at compile time. The "if" is evaluated or decided at run time.

What is the “this” Pointer?

The “this” pointer is silently passed with a call to an instance-level function, which then operates on an object (instance of a class).

Basically, this core mechanism makes it possible to bring operations close to data. It also eliminates the need to have global functions and it gives data structures the intrinsic ability to perform operations on its data.

What is a String? What are the properties of a String Class?

A String is a collection of char objects. We can also declare string variables in c#.

```
string name = “C# Questions”;
```

A string class in C# represents a string.

The properties of String class are **Chars** and **Length**.

Chars get the Char object in the current String.

Length gets the number of objects in the current String.

Why are strings in C# immutable?

Immutable means string values cannot be changed once they have been created. Any modification to a string value results in a completely new string instance, thus an inefficient use

of memory and extraneous garbage collection. The mutable System.Text.StringBuilder class should be used when string values will change.

What is an Escape Sequence? Name some String escape sequences in C#.

An Escape sequence is denoted by a backslash (\). The backslash indicates that the character that follows it should be interpreted literally or it is a special character. An escape sequence is considered as a single character.

String escape sequences are as follows:

- \n – Newline character
- \b – Backspace
- \\ – Backslash
- \‘ – Single quote
- \‘‘ – Double Quote

How do you initiate a string without escaping each backslash?

You put an @ sign in front of the double-quoted string.

String ex = @"This has a carriage return\r\n"

What are Regular expressions? Search a string using regular expressions?

Regular expression is a template to match a set of input. The pattern can consist of operators, constructs or character literals. Regex is used for string parsing and replacing the character string.

For Example:

* matches the preceding character zero or more times. So, a*b regex is equivalent to b, ab, aab, aaab and so on.

Searching a string using Regex

```
static void Main(string[] args)
{
   string[] languages = { "C#", "Python", "Java" };
   foreach(string s in languages)
   {
      if(System.Text.RegularExpressions.Regex.IsMatch(s,"Python"))
      {
         Console.WriteLine("Match found");
```

```
        }
    }
}
```

The above example searches for “Python” against the set of inputs from the languages array. It uses Regex.IsMatch which returns true in case if the pattern is found in the input. The pattern can be any regular expression representing the input that we want to match.

What are the basic String Operations? Explain.

Some of the basic string operations are:

- **Concatenate** – Two strings can be concatenated either by using System.String.Concat or by using + operator.
- **Modify** – Replace(a,b) is used to replace a string with another string. Trim() is used to trim the string at the end or at the beginning.
- **Compare** – System.StringComparison() is used to compare two strings, either case-sensitive comparison or not case sensitive. Mainly takes two parameters, original string, and string to be compared with.
- **Search** – StartWith, EndsWith methods are used to search a particular string.

What is Parsing? How to Parse a Date Time String?

Parsing is converting a string into another data type.

For Example:

```
string text = “500”;

int num = int.Parse(text);
```

500 is an integer. So, Parse method converts the string 500 into its own base type, i.e int.

Follow the same method to convert a DateTime string.

```
string dateTime = “Jan 1, 2018”;

DateTime parsedValue = DateTime.Parse(dateTime);
```

What is an Array? Give the syntax for a single and multi-dimensional array?

An Array is used to store multiple variables of the same type. It is a collection of variables stored in a contiguous memory location.

For Example:

```
double numbers = new double[10];
```

```
int[] score = new int[4] {25,24,23,25};
```

A Single dimensional array is a linear array where the variables are stored in a single row. Above example is a Single dimensional array.

Arrays can have more than one dimension. Multidimensional arrays are also called rectangular arrays.

For Example:

```
int[,] numbers = new int[3,2] { {1,2} ,{2,3},{3,4} };
```

Name some properties of Array.

Properties of an Array include:

- Length – Gets the total number of elements in an array.
- IsFixedSize – Tells whether the array is fixed in size or not.
- IsReadOnly – Tells whether the array is read-only or not.

What is an Array Class?

An Array class is the base class for all arrays. It provides many properties and methods. It is present in the namespace System.

List out the differences between Array and Array List in C#?

Array stores the values or elements of same data type but array list stores values of different data types.

Arrays will use the fixed length but array list does not use fixed length like array.

Explain Jagged Arrays in C#?

If the element of an array is an array, then it's called as jagged array. The elements can be of different sizes and dimensions.

#2 Differentiate between boxing and unboxing.

The following are the difference between boxing and unboxing:

Boxing	Unboxing
Implicit conversion	Explicit conversion
Object type refers to the value type	Retrieve value from the boxed object
int n = 12; object ob = n;	int m = (int) ob;

Example in Code:

```
using System;
public class Program
  {
   public static void Main(string[] args)
     {
     int[] arr = new int[2];
     arr[1] = 10;
     Object obj = arr;                  // here implement boxing
     int[] arr1 = (int[])obj;             // here implement unboxing
     arr1[1] = 100;
     Console.WriteLine(arr [1]);
     ((int []) obj) [1] = 1000;
     Console.WriteLine(arr [1]);
     }
  }
```

What are cookies?

Cookies are small bits of text information. Cookies are created by the server on the client for identifying users. It may contain the username and ID, interests, password remember option, or any other information. Cookies are domain-specific.

Name some of the disadvantages of cookies.

The main disadvantages of cookies include:

- Possible security risk, as they are stored in a clear text
- Not secure, as encryption & decryption is easy
- Cookies can be disabled on any user's computer
- Can be edited or deleted
- Cookies can store limited data.

Explain state management in ASP.NET.

State management is the process of managing the state of one or more user interface controls such as text fields, OK buttons, radio buttons, etc. in a graphical user interface. Two types of state management systems are there in ASP.NET –

1. Client-side state management
2. Server-side state management

What are the key differences between function and stored procedure in .Net programming language?

Function	Store Procedure
A function has a return type and returns value.	A procedure does not have a return type, but it returns values using the out parameters.
A function does not allow output parameters.	A procedure allows both input output parameters.
You can't call stored procedure from a function.	You can call a function from a procedure.
You can call a function using a select statement.	You can't call a procedure using select statements.
Function can't handle exceptions	Procedure can handle exceptions using try-catch block

How would you retrieve user names for Windows Authentication?

User name for Windows Authentication can be retrieved by using:

System.Environment.UserName

Name the advantages of using Session State.

The advantages of Session State include –

- ✓ Easy to implement
- ✓ Stores user states and data across the application
- ✓ Ensures data durability
- ✓ Works in multi-process configuration, thereby ensuring platform scalability
- ✓ Store's session object on the server. Keeping it secure and transparent from the user

What is HTTPhandler?

HttpHandler is a low-level request and response API in ASP.Net. It is used by the ASP.NET web application server to handle specific extension-based requests.

What is Garbage Collector in .NET?

The garbage collector is responsible to free up the unused code objects in the memory. Every time a new object is created, the common language runtime allocates memory for the object.

Name the methodology used to enforce garbage collection in .NET.

The methodology used to enforce garbage collection in .NET is

System.GC.Collect().

What is a Destructor in C#?

A Destructor is used to clean up the memory and free the resources. But in C# this is done by the garbage collector on its own. System. GC.Collect() is called internally for cleaning up. But sometimes it may be necessary to implement destructors manually.

For Example:

```
~Car()
{
   Console.WriteLine("....");
}
```

How many types of indexes are there in .NET?

There are two types of indexes in .Net:

1. Clustered index
2. Non-clustered index

How many types of memories do exist in .Net?

There are two types of memories in .Net

1. Stack memory
2. Heap Memory

Mention the divisions of the Memory Heap?

Following is the division of memory heap into three generations.

- Generation 0 – Used to store short-lived objects. Quick Garbage Collection occurs in this Generation.
- Generation 1 – Used for medium-lived objects.
- Generation 2 – Used for long-lived objects.

What is LINQ?

LINQ notices to Language Integrated Query. It was introduced with visual studio 2008 and is a uniform query syntax in C# and VB.NET for data retrieval from different sources and data manipulation irrespective of the data source.

Name different types of constructors in C#.

Different types of constructors in C# are -

- ✓ Copy Constructor
- ✓ Default Constructor
- ✓ Parameterized constructor
- ✓ Private Constructor
- ✓ Static Constructor

What are the different events in the Page Life Cycle?

Different events in the Page Life Cycle include:

- Page_PreInit
- Page_Init
- Page_InitComplete
- Page_PreLoad
- Page_Load
- Page_LoadComplete
- Page_PreRender
- Render

Name all the templates of the Repeater control.

Templates of the Repeater control are:

- ItemTemplate
- AlternatingItemTemplate
- SeparatorTemplate
- HeaderTemplate
- FooterTemplate

What code we can use to send e-mail from an ASP.NET application?

We can write a given code to send an e-mail:

MailMessage mailMess = new MailMessage ();

mailMess.From = “naukri@gmail.com”;

mailMess.To = “shiksha@gmail.com”;

mailMess.Subject = “Test email”;

mailMess.Body = “Hi check this test mail.”;

SmtpMail.SmtpServer = “localhost”;

SmtpMail.Send (mailMess);

MailMessage and SmtpMail are classes defined System.Web.Mail namespace.

Mention the design principles used in .NET?

In .Net, SOLID design principle is used, take a look at the following design principles:

- Single responsibility principle (SRP)
- Open-Closed Principle (OCP)
- Liskov substitution principle (LSP)
- Interface segregation principle (ISP)
- Dependency inversion principle (DIP)

What is Cohesion?

In OOPS we develop our code in modules. Each module has certain responsibilities. Cohesion shows how much module responsibilities are strongly related.

Higher cohesion is always preferred. Higher cohesion benefits are:

- ✓ Improves maintenance of modules.
- ✓ Increase reusability.

Explain Coupling in C#?

Coupling shows the relationship that how one class is connected or dependent with another class.

There are two types of coupling here...

1. Tight Coupling
2. Loose Coupling.

[N.B. C# preferred loose coupling relationship]

What is IL?

IL refers to Intermediate Language and is an object-oriented programming language to be used by the compilers. It gets changed over to byte code when a virtual machine is being executed. IL is also known as Common Intermediate Language (CIL) or Microsoft Intermediate Language (MSIL).

What is Managed and Unmanaged code?

Managed code is a code that is executed by the CLR (Common Language Runtime) i.e all application code based on .Net Platform. It is considered as managed because of the .Net framework which internally uses the garbage collector to clear up the unused memory.

Unmanaged code is any code that is executed by the application runtime of any other framework apart from .Net. The application runtime will take care of memory, security, and other performance operations.

How do we execute managed code?

To execute managed code, we can follow these steps:

- Select a language compiler depending on the language of the code.
- Convert the code into Intermediate language using its compiler.
- The IL is then targeted to CLR which transforms the code into native code using JIT.
- Execute Native code.

Explain Code compilation in C#.

There are four steps in code compilation, which include:

1. Compiling the source code into Managed code by C# compiler.
2. Combining the newly created code into assemblies.
3. Loading the Common Language Runtime (CLR).
4. Executing the assembly by CLR.

Name the different types of assemblies?

The two different types of assemblies are:

1. Private Assembly: Accessible only to the application.
2. Shared Assembly: Can be shared by multiple applications.

Mention the different parts of an Assembly?

Following are the different parts of an Assembly:

- Manifest – Stores the information about the version of an assembly.
- Type Metadata – Stores the binary information of the program.
- MSIL – Microsoft Intermediate Language code.
- Resources – List of related files.

What is ADO (ActiveX Data Objects)?

ADO is an application program used for writing Windows applications. It is also used to get access to a relational or non-relational database from database providers such as Microsoft and others.

List the fundamental objects in ADO.NET?

Following are the fundamental objects in ADO.NET:

- DataReader- connected architecture
- DataSet- disconnected architecture

What is a PE file?

PE stands for Portable Executable. It is a derivative of the Microsoft Common Object File Format (COFF). Windows executable, .EXE or DLL files follow the PE file format. It consists of four parts:

1. PE/COFF headers- Contains information regarding .EXE or DLL file.
2. CLR header- Contains information about CLR & memory management.
3. CLR data- Contains metadata of DDLs and MSIL code generated by compilers.
4. Native image section- Contains sections like .data, .rdata, .rsrc, .text etc.

What is the difference between DLL and EXE?

.EXE files are single outbound files that cannot be shared with other applications. DLL files are multiple inbound files that are shareable.

What is Just In Time (JIT)?

Just In Time (JIT) is a compiler in the CLR that is responsible for executing .NET programs of different languages. This is done by converting them into machine code. It speeds up the execution of code and supports multiple platforms. There are three types of Just In Time compilers including:

1. Pre-JIT compiler: It compiles all source code into the machine code in one compilation cycle, i.e. at the time of application deployment.
2. Normal JIT compiler: Source code methods needed at the run-time, are compiled into the machine code and is stored in the cache that is to be called later.

3. Econo JIT compiler: Methods required only at the run-time are compiled through this compiler and are not stored for use in the future.

What is Caching in ASP.NET?

Define Caching in ASP.NET.

- Caching technique allows to store/cache page output or application data on the client.

- The cached information is used to serve subsequent requests that avoid the overhead of recreating the same information.

There are several types of caching in ASP.NET:

1. Page Output Caching

2. Page Fragment Caching

3. Data Caching

- This enhances performance when the same information is requested many times by the user.

i. Page Output Caching:

- It is implemented by placing an OutputCache directive at the top of the .aspx page at design time.

Example:

<%@OutputCacheDuration="30" VaryByParam= "empId"%>

- In the above example, 'Duration' parameter specifies for how long the page would be in the cache and the 'VaryByParam' parameter is used to cache different version of the page.

ii. Data Caching:

Data Cache is used to store frequently used data in the Cache memory. It's much efficient to retrieve data from the data cache instead of database or other sources. We need use System.Web.Caching namespace. The scope of the data caching is within the application domain unlike "session". Every user can able to access these objects.

Create: Cache["Employee"] ="DatasetName";

Retrieve: Dataset dsEmployee = (Dataset) Cache ["Employee"]

[It is implemented by using the Cache object to store and quick retrieval of application data.]

iii. Page Fragment Caching:

Sometimes we might want to cache just portions of a page. For example, we might have a header for our page which will have the same content for all users. To specify that a user control should be cached, we use the @OutputCache directive just like we used it for the page.

<%@OutputCache Duration=10 VaryByParam="None" %>

[It is used to store part of a Web form response in memory by caching a user control.]

What do you mean by OOP concept?

OOP stands for Object Oriented Programming. It is a methodology or a kind of design philosophy that helps to write the program where we specify the code in form of classes and objects.

OOP concept involves:

- Class
- Object
- Abstraction
- Polymorphism
- Inheritance
- Encapsulation

Class: A class is a collection of Similar objects and represents description of objects that share same attributes and actions or behaviors.

Example:

(If) Consider Class Name	(Then object is) Here, Objects of this Class
Class-1: Fruit	Object-1: Banana
	Object-2: Apple
	Object-3: Pineapple etc.
Class-2: Flower	Object-1: Rose
	Object-2: Sunflower
	Object-3: Lily
	Object-4: Jasmine etc.
Class-3: Student	Object-1: Mary (001)
	Object-2: Ram (002)
	Object-3: John (003) etc.

Here is the syntax and declaration example of Class:

public class Student {

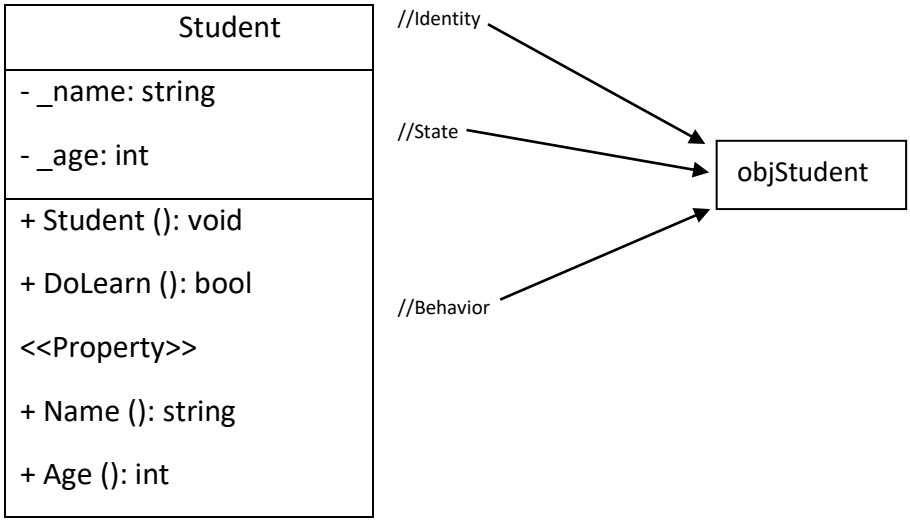
```
    //your code goes here...
}
```

There are 4 types of classes that we can use in C#:

1. **Partial class** – Allows its members to be divided or shared with multiple .cs files. It is denoted by the keyword Partial.
2. **Sealed class** – It is a class that cannot be inherited. To access the members of a sealed class, we need to create the object of the class. It is denoted by the keyword Sealed.
3. **Abstract class** – It is a class whose object cannot be instantiated. The class can only be inherited. It should contain at least one method. It is denoted by the keyword abstract.
4. **Static class** – It is a class that does not allow inheritance. The members of the class are also static. It is denoted by the keyword static. This keyword tells the compiler to check for any accidental instances of the static class.

Object: Object is an instance of a class. It is a logical as well as a physical entity. Object has three features: State, Behavior & Identity.

Consider a Student class here…



Student **objStudent** = new Student ();

According to the above sample we can say that the object called **objStudent** object has been created from the Student class.

In this part, we will be focusing more on other major core OOPs concepts:

Abstraction: Abstraction is a process of hiding the implementation details and showing only functionality to the user.

Abstraction can be achieved by two ways:

1. Abstract class
2. Interface

Abstract class and interface both can have abstract methods which are necessary for abstraction.

The following are some of the key points –

- You cannot create an instance of an abstract class
- You cannot declare an abstract method outside an abstract class
- When a class is declared sealed, it cannot be inherited, abstract classes cannot be declared sealed.

Example:

==========

```
using System;

namespace Demo {
  abstract class Shape {
    public abstract int area();
  }
  class Rectangle: Shape {
    private int length;
    private int width;
    public Rectangle( int a = 0, int b = 0) {
      length = a;
      width = b;
    }
    public override int area () {
      Console.WriteLine("Rectangle class area:");
      return (width * length);
    }
  }
  class RectangleTester {
```

```
    static void Main (string [] args) {
      Rectangle r = new Rectangle (20, 15);
      int a = r.area();
      Console.WriteLine("Area: {0}",a);
      Console.ReadKey();
    }
  }
}
```

Output

Rectangle class area:

Area: 300

Interface:

Interface in C# is **a blueprint of a class**. An interface looks like a class, but has no implementation. It contains are declarations of events, indexers, methods and/or properties.

The reason interfaces only provide declarations is because they are inherited by structs and classes, that must provide an implementation for each interface member declared.

The implementation of the interface's members will be given by class who implements the interface implicitly or explicitly.

- Interfaces specify what a class must do and not how.
- Interfaces can't have private members.
- By default, all the members of Interface are public and abstract.
- The interface will always define with the help of keyword '***interface*** '.
- Interface cannot contain fields because they represent a particular implementation of data.
- *Multiple inheritance* is possible with the help of Interfaces but not with classes.

Syntax for Interface Declaration:

```
interface  <interface_name >
{
   // declare Events
```

```
    // declare indexers
    // declare methods
    // declare properties
}
```

Syntax for Implementing Interface:

```
class class_name : interface_name
```

To declare an interface, use **interface** keyword. It is used to provide total abstraction. That means all the members in the interface are declared with the empty body and are public and abstract by default. A class that implements interface must implement all the methods declared in the interface.

Example 1:

```
// C# program to demonstrate working of
// interface
using System;
// A simple interface
interface inter1
{
        // method having only declaration
        // not definition
        void display ();
}
// A class that implements interface.
class testClass : inter1
{
        // providing the body part of function
        public void display()
        {
                Console.WriteLine("Sudo Placement GeeksforGeeks");
        }
        // Main Method
        public static void Main (String []args)
        {
```

```
                // Creating object
                testClass t = new testClass();
                // calling method
                t.display();
        }
}
```

Output:

Sudo Placement GeeksforGeeks

Example 2:

```
// C# program to illustrate the interface
using System;
// interface declaration
interface Vehicle {
        // all are the abstract methods.
        void changeGear(int a);
        void speedUp(int a);
        void applyBrakes(int a);
}
// class implements interface
class Bicycle : Vehicle{
        int speed;
        int gear;
        // to change gear
        public void changeGear(int newGear)
        {
                gear = newGear;
        }
        // to increase speed
        public void speedUp(int increment)
```

```
        {
                speed = speed + increment;
        }
        // to decrease speed
        public void applyBrakes(int decrement)
        {
                speed = speed - decrement;
        }
        public void printStates()
        {
                Console.WriteLine("speed: " + speed +" gear: " + gear);
        }
}
// class implements interface
class Bike : Vehicle {
        int speed;
        int gear;
        // to change gear
        public void changeGear(int newGear)
        {
                gear = newGear;
        }
        // to increase speed
        public void speedUp(int increment)
        {
                speed = speed + increment;
        }
        // to decrease speed
        public void applyBrakes(int decrement){
                speed = speed - decrement;
        }
```

```
        public void printStates()
        {
                Console.WriteLine("speed: " + speed +" gear: " + gear);
        }
}
class GFG {
        // Main Method
        public static void Main (String [] args)
        {
                // creating an instance of Bicycle
                // doing some operations
                Bicycle bicycle = new Bicycle();
                bicycle.changeGear(2);
                bicycle.speedUp(3);
                bicycle.applyBrakes(1);
                Console.WriteLine("Bicycle present state :");
                bicycle.printStates();
                // creating instance of bike.
                Bike bike = new Bike();
                bike.changeGear(1);
                bike.speedUp(4);
                bike.applyBrakes(3);
                Console.WriteLine("Bike present state :");
                bike.printStates();
        }
}
```

Advantage of Interface:

- ✓ It is used to achieve loose coupling.
- ✓ It is used to achieve total abstraction.
- ✓ To achieve component-based programming

- ✓ To achieve multiple inheritance and abstraction.
- ✓ Interfaces add a plug and play like architecture into applications.

Abstract class vs interface

Abstract Class	Interface
Abstract class contains both declaration and definition part.	Interface contains only a declaration part.
Abstract class doesn't support multiple inheritance.	Interface supports multiple inheritance.
Abstract class contain constructor.	Interface does not contain constructor.
Abstract class can have access specifier for functions.	Interface cannot have access specifier for functions. It is public by default.
Abstract class is fast.	Interface is comparatively slow.
Abstract class can have abstract and non-abstract methods.	Interface can have only abstract methods. Since Java 8, it can have default and static methods also.
Abstract class can have final, non-final, static and non-static variables.	Interface has only static and final variables.
The abstract keyword is used to declare abstract class.	The interface keyword is used to declare interface.
An abstract class cannot be instantiated.	The interface is absolutely abstract and cannot be instantiated.

Simply, abstract class achieves partial abstraction (0 to 100%) whereas interface achieves fully abstraction (100%).

When to use Abstract Interfaces or Classes?

- ➢ According to the mentioned explanations, when we need multiple inheritance, we must use the Interface; Because this is not possible in Abstract classes.
- ➢ When we want to implement all the methods introduced in the base class completely in the derived class, we must use the Interface.
- ➢ When we face many changes in large projects, the use of the Abstract class is recommended; Because by changing it, changes are automatically applied to the derived classes.
- ➢ Due to the fact that it is not possible to define other elements in the Interfaces other than the declaration of methods and properties, if we are required to use these elements, it is necessary to use Abstract classes.
- ➢ If we do not want to implement all the methods in the derived classes and code some of them in the parent class, we must use the Abstract class.

- In general, an interface defines the framework and capabilities of a class and is a contract; But the Abstract class determines the type of class. This difference helps programmers to determine when to use the two.

Polymorphism: Polymorphism is the ability of an object to take on many forms.

There are two types of polymorphism like that.

Static polymorphism (compile time): Compile time polymorphism is method and operators overloading. It is also called early binding.

In method overloading method performs the different task at the different input parameters.

Example:

```
Public class TestData {
        Public int Add (int a, int b, int c) {
                Return a+b+c;
        }
        Public int Add (int a, int b) {
                Return a+b;
        }
}
Class Program {
        Static void main (String [] args) {
                TestData data = new TestData ();
                Int add3 = data.Add(1, 2, 3);
                Int add2 = data.Add(1,2);
        }
}
```

Dynamic polymorphism (Runtime): Runtime polymorphism is done using inheritance and virtual methods. Method overriding is called runtime polymorphism. It is also called late binding.

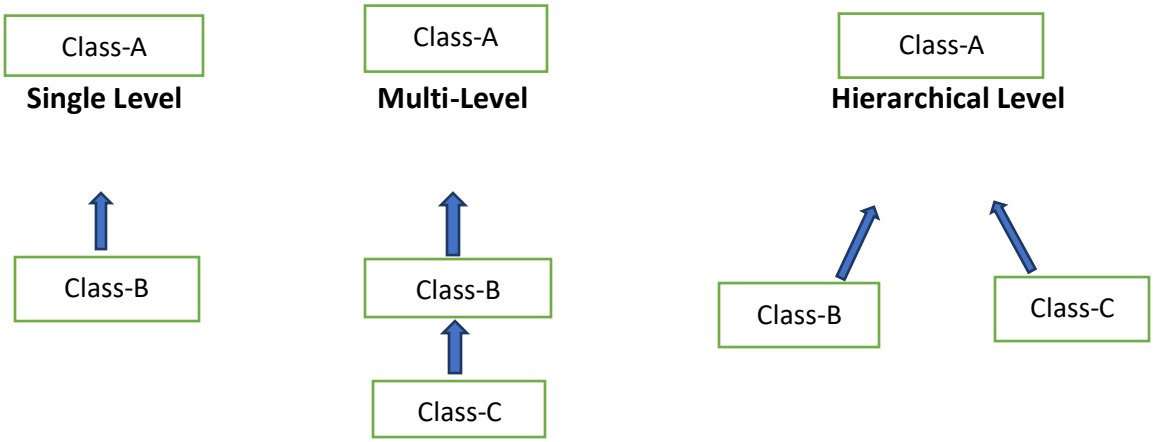
When overriding a method, you change the behavior of the method for the derived class. Overloading a method simply involves having another method with the same prototype.

Example:

```
Class Color {
        Public virtual void Fill () {
                Console.WriteLine("Fill me up with color");
        }
        Public void Fill (string inputColor) {
                Console.WriteLine("Fill me up with {0}", inputColor);
                }
        }
Class Green: Color {
        Public override void Fill () {
                Console.WriteLine("Fill me up with green");
                }
        }
Class Program {
        Public static void Main (string [] args){
                Green C1 = new Green ();
                        C1.Fill();
                C1.Fill("Red");
        Console.ReadKey();
        }
}
```

Inheritance: Inheritance is a mechanism in which one object acquires all the states and behaviors of a parent object. Inheritance uses a parent-child relationship

(Is-A Relationship).



C# multi-level parent inheritance not allow

Example:

```
Using System;
Public class BaseClass {
   Public BaseClass () {
      Console.WriteLine ("Base Class Constructor executed");
 }
   Public void Write () {
      Console.WriteLine ("Write method in Base Class executed");
   }
}
Public class ChildClass: BaseClass {
   Public ChildClass () {
      Console.WriteLine("Child Class Constructor executed");
   }
```

```
    Public static void Main ()  {
        ChildClass CC = new ChildClass ();
        CC.Write ();
    }
}
```

In the Main () method in ChildClass we create an instance of childclass. Then we call the write () method. If you observe the ChildClass does not have a write() method in it. This write () method has been inherited from the parent BaseClass.

The output of the above program is

Output:

Base Class Constructor executed

Child Class Constructor executed

Write method in Base Class executed

this output proves that when we create an instance of a child class, the base class constructor will automatically be called before the child class constructor. So in general Base classes are automatically instantiated before derived classes.

[more details oop link below…]

Encapsulation: Encapsulation is a process of wrapping code and data together into a single unit.

It's implemented by using access modified –

.Net has five access Specifiers

Public -- Object that implement public access modifier are accessible from everywhere in our project.

Private -- Object that implement private access modifier are accessible only inside a class or a structure.

Protected -- The protected keyword implies that the object is accessible inside the class and in all classes that derive from that class.

Internal -- The internal keyword specifics that the object is accessible only inside its own assembly but not in other assemblies.

Protected Internal -- The protected internal access modifier is a combination of protected and internal. As a result, we can access the protected internal member only in the same assembly or in a derived class in other assembly(project).

Abstraction Vs Encapsulation

Abstraction	**Encapsulation**
Abstraction solves the problem in the design level.	Encapsulation solves the problem in the implementation level.
Abstraction is outer layout in terms of design.	Encapsulation is inner layout in terms of implementation.
For Ex: Outer look of a iPhone like it has a display screen.	For Ex: Inner implementation details of a iPhone, how display screen are connect with each other using circuits.

What is the difference between the Virtual method and the Abstract method?

A Virtual method must always have a default implementation. However, it can be overridden in the derived class, though not mandatory. It can be overridden using override keyword.

An Abstract method does not have an implementation. It resides in the abstract class. It is mandatory that the derived class implements the abstract method. An override keyword is not necessary here though it can be used.

What do we need to use static in c#?

In C#, static means something which cannot be instantiated. You cannot create an object of a static class and cannot access static members using an object. C# classes, variables, methods, properties, operators, events, and constructors can be defined as static using the static modifier keyword.

Can a static method access non-static fields?

The Answer is Yes. Static methods can directly access static fields without using class name also. However, to access non-static fields an object needs to be created.

If in a static class, we don't have non static members/functions then does it hold any default constructor?

Yes, it does hold a constructor but it is also declared as static.

Can static constructor have public/private/internal/protected internal access modifiers?

No, static constructor doesn't contain any access modifiers. We have already seen the implementation in previous example.

What is the difference between a struct and a class?

Class	Struct
Supports Inheritance	Does not support Inheritance
Class is Pass by reference (reference type)	Struct is Pass by Copy (Value type)
Members are private by default	Members are public by default
Good for larger complex objects	Good for Small isolated models
Can use waste collector for memory management	Cannot use Garbage collector and hence no Memory management

Are private class members inherited to the derived class?

Yes, the private members are also inherited in the derived class but we will not be able to access them. Trying to access a private base class member in the derived class will report a compile time error.

How do you prevent a class from being inherited?

The sealed keyword prohibits a class from being inherited.

What is the difference between Continue and Break Statement?

Break statement breaks the loop. It makes the control of the program to exit the loop. Continue statement makes the control of the program to exit only the current iteration. It does not break the loop.

How is Exception Handling implemented in C#?

Exception handling is done using four keywords in C#:

try – Contains a block of code for which an exception will be checked.

catch – It is a program that catches an exception with the help of an exception handler.

finally – It is a block of code written to execute regardless of whether an exception is caught or not.

Throw – Throws an exception when a problem occurs.

Can we have only “try” block without “catch” block in C#?

Yes, we can have only try block without catch block.

What is the difference between finally and finalize block?

Finally, block is called after the execution of try and catch block. It is used for exception handling. Regardless of whether an exception is caught or not, this block of code will be executed. Usually, this block will have a clean-up code.

Finalize method is called just before garbage collection. It is used to perform clean up operations of Unmanaged code. It is automatically called when a given instance is not subsequently called.

Why to use “finally” block in C#?

“Finally,” block will be executed irrespective of exception. So, while executing the code in try block when exception is occurred, control is returned to catch block and at last “finally” block will be executed. So, closing connection to database / releasing the file handlers can be kept in “finally” block.

What you mean by inner exception in C#?

Inner exception is a property of exception class which will give you a brief insight of the exception i.e, parent exception and child exception details.

What is the difference between “out” and “ref” parameters in C#?

“out” parameter can be passed to a method and it need not be initialized where as “ref” parameter has to be initialized before it is used.

What is C# I/O Classes? What are the commonly used I/O Classes?

C# has System.IO namespace, consisting of classes that are used to perform various operations on files like creating, deleting, opening, closing, etc.

Some commonly used I/O classes are:

- File – Helps in manipulating a file.
- StreamWriter – Used for writing characters to a stream.
- StreamReader – Used for reading characters to a stream.
- StringWriter – Used for reading a string buffer.
- StringReader – Used for writing a string buffer.
- Path – Used for performing operations related to the path information.

What is StreamReader/StreamWriter class?

StreamReader and StreamWriter are classes of namespace System.IO. They are used when we want to read or write charact90, Reader-based data, respectively.

Some of the members of StreamReader are: Close(), Read(), Readline().

Members of StreamWriter are: Close(), Write(), Writeline().

Syntax example:

```
Class Program1
{
   using (StreamReader sr = new StreamReader("C:\ReadMe.txt")
   {
      //----------------code to read-------------------//
   }
   using (StreamWriter sw = new StreamWriter("C:\ReadMe.txt"))
   {
      //-------------code to write-------------------//
   }
}
```

Explain Delegates in C#?

In simple word, **Delegate is a pointer to a function that points to a function.** It contains the reference to several methods and call them when needed. So, you create numbers of methods as your need and attach to delegates.

There are three steps in defining and using delegates…

1. Declare a delegate
2. Set a target method
3. Invoke a delegate

There are three types of delegates that can be used in C#.

1. Single Delegate
2. Multicast Delegate
3. Generic Delegate

Necessary of delegates

- Programmers often needs to pass a method as a parameter of other methods. For this purpose, we create and use delegates.
- A delegate is a class that encapsulates a method signature. Although it can be used in any context, it often serves as the basis for the event-handling model in C# and Dot Net.

Benefits of delegates

- ✓ Delegates are object-oriented and type-safe and very secure.
- ✓ Delegates make event-handling simple and easy.
- ✓ Delegates allow methods to be passed as parameters.
- ✓ It is used to callback method.
- ✓ Provides a good way to encapsulate the methods.
- ✓ Delegates can also be used in “anonymous methods” invocation.

Reference

1. https://www.naukri.com/learning/articles/top-dot-net-interview-questions-answers/

2. (oop) https://www.aspdotnet-suresh.com/2015/03/oops-object-oriented-programming-concepts-in-csharp-net-with-examples.html

3. (static) https://www.c-sharpcorner.com/UploadFile/36bc15/static-keyword-in-C-Sharp/

4. https://www.mytectra.com/interview-question/c-interview-questions-and-answers-for-5-years-experienced

5. (class type more...) https://www.c-sharpcorner.com/UploadFile/0c1bb2/types-of-classes-in-C-Sharp1/

6. (variable) https://www.geeksforgeeks.org/c-sharp-types-of-variables/

https://csharp.net-tutorials.com/basics/variables/