Object Oriented programming (OOPS) – Interview Questions

1) What are benefits of Object Oriented Programming Language (OOPS) ?

- **Software Complexity** can be easily managed.
- If we considered the **Classes** and related **objects** separately the Designing will be easy.
- Software depends on **Object-Oriented Programming** is very much near to real life environment. So that, understanding of program is easy.
- Through **Inheritance**, we can eliminate unnecessary Code.
- In OOPS, **Data Hiding** provides more security. So that, unauthorized users can't destroy the Programs.
- **Object-Orientation** contributes to the solution of many problems associated with **development** and **quality** of Software Products.

2) What is a Class?

A class is a Data Structure that contains,

Data members -> Constants and fields.

Function members -> methods, properties, events, indexers, constructors, destructors and operators.

1) What is an Object?

Object is an instance of a class.

2) What is Main ()?

It is the starting point for the execution of a program. It returns a value.

3) What is Void ()?

Void () specifies that the function does not return a value.

4) What is Constant?

Constants are used by **const** keyword. Constants are referring to fixed values that do not change during the execution of a program.

5) What is NEW operator?

NEW operator is used to create object of any type and initialize the memory.

6) What is Data type?

Every variable in c# is associated with a Data type. Data types specify the size and type of values that can be stored.

7) What is Member function?

A function contained within a class is called as a **Member function**.

8) What is Identifier?

Identifiers refer to the name of variables, functions, arrays, classes, etc. created by the programmer.

9) What is type casting?

Type casting is converting one type of variable to another type.

10) What is the purpose of Virtual?

Virtual is a keyword used in C#. If we declare a method in Base class as **Virtual**, then Derived class can override Base class method.

11) What is a Constructor?

Constructor is a function with the same name as that of the class. Constructors are normally used to initialize variables and to allocate memory.

- Constructor should be declared in the **Public** section.
- Constructor has the same name as that of **class name**.
- Constructors are invoked automatically when the **objects** are created.
- Constructors can't be **Virtual**.

12) What is destructor? Can destructors have access modifiers?

It is used to destroy the objects.

No, destructors cannot have access modifiers.

13) What are Indexers?

Indexers are location indicators and are used to access class objects just like accessing elements in an array.

14) In C#, can we create an object of reference type using const keyword?

No.

A constant member may not be created of an object that is of reference type. Because, its value is decided dynamically at runtime.

15) What is Operator Overloading?

The concept of using one operator for different purposes is known as operator overloading.

16) What is Function Overloading?

Function Overloading means we can have more than one function with the same name but different number of arguments in our program.

17) What is Method Overloading?

Method overloading means, we can have more than one method with same name but different number of arguments.

18) What is Method Overriding?

Method overriding means, we can have more than one method with same name , same number of arguments and same type of arguments.

19) What is Encapsulation?

Encapsulation means, wrapping up (or) Grouping of data and functions into a single unit.

(or)

Encapsulation means, hiding the implementation of the class and exposing the functionality.

22) What is Polymorphism?

Polymorphism means one name having multiple forms.

Polymorphism allows us to have more than one function with the same name in program.

- Compile time polymorphism (Method Overloading) -> Executes during compilation time.
- Runtime Polymorphism (Method Overriding) -> Executes during runtime.

23) What is Data Abstraction?

Abstraction means representing essential features without including the background details.

(or)

Abstraction is a process of hiding the implementation details and displaying the essential features.

24) What is the difference between a Struct and a Class?

Struct is a value-type.

Class is a reference type.

Another difference is that structs cannot inherit.

25) What is Abstract Class?

Abstract class is a class that cannot be instantiated, but it must be inherited.

26) What is a Static Class?

If we declare a class as **Static**, then we can access that class methods directly using class name without creating an object.

27) What is Sealed Class?

Sealed classes are used to restrict the inheritance feature of object oriented programming.

Once a class is defined as **sealed** class, this class cannot be inherited.

28) What is Delegate Class?

Delegate is an object that holds the reference to a method.

Delegates are reference types that derive from **System.Delegate** class.

29) What is Inheritance?

A class inherit properties of another class is known as Inheritance.

Inheritance provides Reusability. This means that we can add additional features to an existing class without modifying it. This is possible by deriving a new class from the existing one.

(or)

The Derived Class inherits some (or) all of the properties of Bass Class.

- **Single Inheritance ->** A Derived class can inherit properties from one Bass class.
- **Multiple Inheritance ->** A Derived class can inherit properties from more than one class.
- **Multi-Level Inheritance ->** A class can derived from another derived class.
- **Hierarchical Inheritance ->** Properties of one class is inherited by more than one class.

30) While using inheritance, derived class constructor will call base class constructor?

Yes, Base class constructor will be called before child class constructor.

31) How does u create multiple inheritances in c# and .net?

Multiple inheritances are created by using **interfaces**.

32) What is the syntax to inherit from a class in C#?

Place a colon and then the name of the base class.

Syntax: **class MyNewClass: MyBaseClass**

33) Does C# support multiple-inheritance?

No.

34) Why C# doesn't support multiple inheritance?

C# does not support multiple inheritance because of **name collision** (or) **ambiguity**.

Example:

```
public class Parent1
  {
     public string PrintData()
     {
        return "This is parent1";
     }
  }

  public class Parent2
  {
     public string PrintData()
     {
        return "This is parent2";
     }
  }

public class Child : Parent1, Parent2
  {
     public string GetData()
       {
          return this.PrintData();
       }
  }
```

In the above example, there are two Parent class **Paretn1** and **Parent2**. This is inherited by **Child** class, In **GetData()** method, **child** calls the parent class method **PrintData().**

In this case which method will be executed?

It is very difficult for **CLR** to identify which method to call. It shows that multiple inheritance create **ambiguity** to oops concept. In order to avoid this ambiguity we are going for **multiple interface** implementations.

35) Why we create static class in c#?

We create **static** class, because we can call its method directly without creating an object.

It means, we can access static class methods directly using **class name** without creating an object.

36) What is Interface?

Interface is used to define a set of properties, methods, and events without any implementation.

Interface contains methods without implementation. They are implemented by classes.

37) Difference between Abstract and Interface?

Abstract Class	Interface
Accessibility modifier (Public/Private/internal) is allowed for abstract class.	Interface doesn't allow accessibility modifier. Because, Interfaces always Public.
Class can inherit only one abstract class.	Class can implement more than one interface.
An abstract class cannot support multiple inheritance	interface can support multiple inheritance.
Abstract class can implement methods	Interface cannot implement methods.
Abstract classes can have implementations for some of its methods	Interface class can't have implementation for any of its methods.

38) Can we set the specifier of the main() method in C# as private?

Yes.

Example:

```
private static void Main()
 {
   //This code isn't invoked automatically by other assemblies
 }
```

39) What is If Statement?

If condition executes the statement once if the condition is **True**.

If condition is **False** it executes **ELSE** part.

40) What is While Statement?

While condition executes the statement repeatedly as long as the expression is **True**.

41) What is For Statement?

For loop executes a statement or a block of statements repeatedly until a specified expression evaluates to false. For loop is used to run statements a fixed number of times.

42) What is Foreach Statement?

Foreach loop executes a block of code on each element in an array or a collection of items.

43) What is an array?

Array is a data structure that stores a collection of individual values that are of the same data type.

44) What is an ArrayList ()?

ArrayList object is used to store a collection of values in the form of a List.

45) Difference between Array and ArrayList()?

Major difference in between **Array** and **ArrayList** is,

Elements can be added & removed from an **ArrayList** at the runtime. In **Array**, number of dimensions and size of the array are fixed at instantiation.

46) Can you store multiple data types in System.Array?

No.

47) What is the use of Sealed Class in C# ?

Using **Sealed** Class, we can prevent **inheritance** and **overriding** of a class.

48) What is Generics in C#?

- Generics are most commonly used to create a collection.
- It is derived from **System.collection.Generics** namespace.
- Advantages of Generics are Performance, Code Reuse and Type Safety.
- Using generics we can we can work with any type (like int, string, etc.).

49) Will the finally block get executed if an exception has not occurred?

Yes.

50) Can multiple catch blocks be executed for a single try statement?

No.

Once the proper catch block processed, control is transferred to the finally block.

51) A Try block having 4 catch blocks. Will fire all catch block or not?

No.

If **Try** block having more than one **catch** block, will fire the first relevant catch block after that cursor will be moved to the finally block (if exists) leaving all remaining catch blocks.

So in all cases only one catch block will fire.

52) Can you prevent your class from being inherited by another class?

Yes. The keyword **sealed** will prevent the class from being inherited.

53) Can you allow a class to be inherited, but prevent the method from being over-ridden?

Yes. Just leave the class **public** and make the method **sealed**.

54) Why can't you specify the accessibility modifier for methods inside the interface?

Because, Interfaces always **Public**.

55) Can you inherit multiple interfaces?

Yes.

56) What is OUT keyword?

Out keyword is used for passing a variable for output purpose.

- It has same concept as **ref** keyword, but passing a **ref** parameter needs variable to be initialized while **out** parameter is passed without initialized.
- It is useful when we want to return more than one value from the method.

57) Can you declare an override method to be static, if the original method is not static?

No. The signature of the **virtual** method must remain the same.

58) What are the different ways a method can be overloaded?

- Different number of parameters
- Different order of parameters
- Different parameter data types

59) What is the use of "throw" keyword in C#?

The **throw** keyword is used to throw an exception programmatically in C#.

60) What is a Property in C#? Can you have different access modifiers on the get/set methods of a property in C#?

- Properties encapsulate data members of class.
- Properties provide the facility to protect a field in a class by reading and writing.
- In which, **GET** returns the property value and **SET** sets some private variable.

No. it's not possible. The access modifiers have to be same for **GET** and **SET**.

61) What is Reflection?

- Reflection is the ability to get (or) find the information about object in an assembly at runtime.
- It is derived from namespace **System.Reflection** .
- Using reflection can dynamically load assembly.

62) Difference b/w Delegate and Multicast Delegate?

Delegate: It holds the reference of a method. A Delegate may be used to call a method asynchronously.

Multicast delegate: It contains one (or) more methods references. Its return type is always **void**.

63) What is enum?

Enum is the keyword used to define an enumeration. It is used determine the unique values.

64) Can we specify the access modifiers for the get and set in a property?

Yes

65) What is the difference between GetType() and typeof?

GetType() Operates on object.

Typeof () Operates on type.

66) What is the difference between "Convert" class and "Parse()" method?

The **Convert** class is used to convert any value from any data type to another data type.

Example:

1) char ch = Convert.ToChar("x") -> string to char

2) float f = Convert.ToSingle(45) -> int to float

3) int x = Convert.ToInt(4.5f) -> float to int

The **Parse()** method is used to convert only one **string** value to any other data type.

Example:

1) int x = int.Parse("600"); -> string to int

2) char ch = char.Parse("xdnf"); -> string to char

67) An Interface is a Value type or reference type?

Both

68) Difference between Stack and Heap?

Stack:

- Memory will be allocated at the compile time.
- Here the memory is allocated by the compiler.
- Memory will be allocated only in sequential locations.
- The memory will also be deleted by the compiler.
- There is lot of chance of memory wastage.

Heap:

- Memory will be allocated at the run time.
- Here the memory is allocated by the user.
- Memory will be allocated in sequential locations and non- sequential locations.
- The memory must be deleted explicitly by the user.
- There is no chance of memory wastage if the memory is handled perfectly.

69) Can we assign values to read only variables? If yes then how?

Yes, we can assign values to read only variables either at a time of declaration or in constructors.

70) Can we declare event and delegates in an interface?

No

We cannot declare **delegates** in **interface** . But, we can declare **events** in **interface**.

So **interface** can only contains the signature of the following members,

- Methods
- Properties
- Indexers
- Events

71) What is the use of Delegate?

When a class contains 10 methods and need to access those methods then we normally access these objects through **obj.methodname()** one by one.

But with the help of the **delegate** we can access all 10 methods a single time with the help of the delegate.

Delegates are created at run time.

72. What is a Partial Class? Give me an example?

Partial class means that class definition can be split into multiple physical files.

The compiler intelligently combines the definitions together into a single class at compile-time.

Example:

```
partial class Employee
  {
      string m_Name;
      public string Name
       {
          get { return m_Name; }
          set { m_Name = value; }
       }
  }

partial class Employee
  {
      int m_Age;
      public int Age
       {
         get { return m_Age; }
         set { m_Age = value; }
       }
   }

public class ExampleofPartical
  {
     public void Method1()
      {
         Employee objClass1 = new Employee();
         objClass1.Name="vinay";
         objClass1.Age = 12;
      }
  }
```

72) What is mean by Static method?

Static method can be accessed without creating the instance of the class.

- Static key word is used to mention the static method.
- Static methods can be created inside the normal class or static class.
- If we create the static method inside the normal class, then that static method will not be able to access by creating instance of the class. example

Example:

```
Math.Add();
```

73) Can we override static method?

No, compiler will not allow overriding the **static** method.

74) What are uses of static class and method?

- Compiler will not allow creating the instance of the class.
- Static class also makes the implementation simpler and faster.
- Cannot inherit a **static** class since it is **sealed**.

75.What is static constructor?

Static constructor is used to initialize any static data (or) to perform a particular action that needs performed once only.

It is called automatically before the first instance is created or any static members are referenced.

Example:

```
namespace StaticConstructor
{
   public class MyStaticClass
   {
      static int count;
      static MyStaticClass()
      {
         count = 0;
         Console.WriteLine("Static class is initialized");
      }

      public static void MyMethod(string name)
      {
         Console.WriteLine("Hello " + name);
      }
   }

   class Program
   {
      static void Main(string[] args)
      {
         MyStaticClass.MyMethod("Static Constructor");
         Console.ReadLine();
      }
   }
}
```

Output:

Static class is initialized

Hello Static Constructor

75.What is Nested Classes?

Classes with in classes are called as Nested class.

Example:

```
namespace NestedClasses
{
    public class FirstClass
    {
        public void Display()
        {
            Console.WriteLine("Class1");
        }
        public class SecondClass
        {
            public void Display()
            {
                Console.WriteLine("Class2");
            }

            public class ThirdClass
            {
                public void Display()
                {
                    Console.WriteLine("Class3");
                }
            }
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            //Creating instance of the nested class
            FirstClass L1 = new FirstClass();
            FirstClass.SecondClass L2 = new FirstClass.SecondClass();
            FirstClass.SecondClass.ThirdClass L3 = new FirstClass.SecondClass.ThirdClass();
            L1.Display();
            L2.Display();
            L3.Display();
            Console.ReadLine();
        }
    }
}
```

Output:

Class1

Class2

Class3

75.What is mean by Shadowing (VB.NET) / new (C#)?

We can implement Shadowing in C# using **new** keyword.

Using **new** keyword in **derived class**, we can hide implementation of **Base Class** in program.

(or)

When the method is defined in **base class** is not overridable and we need to provide different implementation for the same in **derived class**. In this kind of scenario we can use hide the **base class** implementation and provide new implementation using **Shadows** (VB.NET)/**new**(C#) keyword.

Example:

```
namespace Shadow_or_New_Keyword
{
  public class ParentClass
  {
    public void Display()
    {
      Console.WriteLine("Parent class");
    }

  }
  public class ChildClass : ParentClass
  {

    public new void Display()
    {
      Console.WriteLine("Child class");
    }
  }
  class Program
  {
    static void Main(string[] args)
    {
      ParentClass p = new ParentClass();
      ChildClass c = new ChildClass();
      ParentClass pc = new ChildClass();
      p.Display();
      c.Display();
      pc.Display();
      Console.ReadLine();
    }
  }
}
```

Output:

```
Parent class
Child class
Parent class
```

75) Why Main () method is static?

To ensure there is only one entry point to the application.

76) How a base class method is hidden?

Using **new** keyword in the derived class, we can hide base class method (or) suppressed.

New implementation can be added to the derived class.

77) What does the keyword virtual mean in the method definition?

The method can be over-ridden.

78) What is Method overloading? Give me an example?

Method overloading means, we can have more than one method with same name but different number of arguments in program.

Example:

```
namespace MethodOverloaing
{
  class AddingNumbers
  {
    public int Add(int a, int b)
    {
      return a + b;
    }

    public int Add(int a, int b, int c)
    {
      return a + b + c;
    }
  }

  class MainClass
  {
    static void Main(string[] args)
    {
      AddingNumbers obj = new AddingNumbers();

      Console.WriteLine("First Method Result is: " + obj.Add(2, 3));
      Console.WriteLine("Second Method Result is: " + obj.Add(2, 3, 4));
      Console.ReadLine();
    }
  }
}
```

Output:

First Method Result is: 5

Second Method Result is: 9

79) What is Method overriding? Give me an example?

- Using **Method overriding** , we can override **Base class** method.
- Method Overriding means we can have more than one method with same name, same number of arguments and same type of arguments.

Example:

```
namespace MethodOverriding
{
   class parent
   {
      public virtual void method1()
      {
         Console.WriteLine("Base class method");
      }
   }

   class child : parent
   {
      public override void method1()
      {
         Console.WriteLine("Derived class method");
      }
   }
   class Program
   {
      static void Main(string[] args)
      {
         parent p = new parent();
         child c = new child();
         parent c1 = new child();

         p.method1();
         c.method1();
         c1.method1();

         Console.ReadLine();
      }
   }
}
```

Output:

Base class method

Derived class method

Derived class method

80) Will sealed class allows inheritance, if not why?

No. Because, **sealed** means it is not inheritable.

81) What are the advantages of Private constructor? from here I have to study

- **Private constructor** will prevent the user from creating the instance of the class which contains only static members.
- **Private constructor** is used for implementing the singleton pattern.

82) Overloaded constructor will call default constructor internally?

No, overload constructor will not call default constructor.

83) What is difference between Overridable and Virtual?

- **Overridable** and **Virtual** are used in parent class to indicate that a method can be overridden.
- **Overridable** keyword is used in **VB.NET** and **Virtual** keyword is used in **C#**.

84) What is difference between Overrides and Override?

- **Overrides** and **Override** are used in the child class to indicate that you are overriding a method.
- **Overrides** keyword is used in **VB.NET** and **Override** keyword is used in **C#**.

85) What is mean by Abstraction?

Abstraction is the process of showing necessary information and hiding unwanted information.

Example:

Let us consider the **Calculate Salary** in your **Employee** class, which takes **Employee ID** as parameter and returns the **Salary** of the employee for the current month as an integer value. Now if someone wants to use that method. He does not need to care about how **Employee** object calculates the salary? An only thing he needs to be concern is name of the method, its input parameters and format of resulting member.

86) What is mean by abstraction class?

- **Abstract** classes contain one or more abstract methods that do not have implementation.
- An **abstract** class is a parent class that allows inheritance but can never be instantiated.
- Abstract classes allow specialization of inherited classes.

87) Can we have different access modifier for Get/Set of the properties?

Yes, in C# 3.0 and above, we can use different access modifier for Get/Set of the properties.

But this is not possible in C#2.0 and lower versions.

88) What is ENUM?

ENUM means Enumeration; it is used to group related sets of constants.

Example:

```
namespace Enum
{
  class Program
  {
    enum E
    {
      None,
      BoldTag,
      ItalicsTag,
      HyperlinkTag,
    };

    static void Main()
    {
      // A.
      // These values are enum E types.
      E en1 = E.BoldTag;
      E en2 = E.ItalicsTag;

      if (en1 == E.BoldTag)
      {
        // Will be printed.
        Console.WriteLine("Bold");
      }
      if (en1 == E.HyperlinkTag)
      {
        // Won't be printed.
        Console.WriteLine("Not true");
      }
    }
  }
}
```

Output:

```
Bold
```

89) What id mean by Interface?

Interface defines the set of properties, signature of the methods and events. It does not include any implementation.

Class which implements the interface can provide the behavior to the implemented method.

Example:

```
namespace InterfaceExample
{
  interface A
  {
    void Read();
  }

  class B : A
  {
    public void Read()
    {
      Console.WriteLine("Read");
    }
  }

  class Program
  {
    static void Main()
    {
      A obj = new B(); // Create instance.
      obj.Read(); // Call method on interface.
      Console.Read();
    }
  }
}
```

Output:

Read

90) What is a multicast delegate?

Multicast delegate stores the reference of multiple methods and eventually fires off several methods.

Multicast delegate must have a return type of **void**.

Example:

If a class contains 10 methods and need to access those methods then we normally access these methods using class object like **obj.methodname()** one by one. But with the help of the **multicast delegate** we can access all 10 methods a single time with the help of the delegate.

91) Can you override private virtual methods?

No

92) What is Event?

Event is an action. Events are nothing but notification.

It provides a notification when event has occurred.

93) Can events have access modifiers?

Yes

94) Can we have static/shared events?

Yes, we can have static(C#)/shared (VB.Net) event.

But only shared method can raise shared events.

95) Can we have goto (or) return statement in finally block?

No.

100) Do events have a return type?

Yes, events return a delegate.

Example:

```
delegate void dd();
event dd myevent;
```

In the above example, **myevent** is the event name and returns the delegate **dd**;

101) What is the Difference between Convert.ToString() and .ToString()?

Convert.ToString() can handle NULLs.

.ToString can't handle Nulls. It will return a Null Reference Exception.

So it would be good if we use **Convert.ToString()** all the time for casting purposes.

102) What is a Static in C#?

If we declare class as **Static** then, we can access that class methods without creating an object.

103) What difference between overloading and overriding?

Overriding : Derived classes follow the same base class method signatures.

Overloading : Derived classes have different method signature with different parameters.

104) What does a method's signature consist of?

Name of the method and parameters.

105) What are access specifiers and method access specifiers?

Public -> Available throughout application.

Private -> Available for class and its inherited class.

Protected -> Restricted to that class only.

ProtectedInternal -> Available throughout that assembly.

106) Where is a protected class-level variable available?

It is available to any sub-class derived from base class.

107) Are private class-level variables inherited?

Yes, but they are not accessible.

108) what is operator overloading?

Operator overloading is used to provide a custom functionality to existing operators.
For Example +,-,* and / operators are used for mathematical functionality. But we can overload these operators to perform custom operation on classes or structures.

Example:

To concatenate the two strings we have to use "**Concat**" method like below,

```
string str1 = null;
string str2 = null;
string str3 = null;

str1 = "Hello";
str2 = "world";
str3 = string.Concat(str1, str2);
```

But, **.Net** provides in build operator overloading for **string** we can use **"+ "**operator for concatenating the string value like below,

str3=str1+str2

Similarly we can also implement operator overloading for classes or structure like below,

Employee3= Employee1 + Employee2

109) What is Sealed Class? How to do it?

Selead Class is used to restrict inheritance and overriding feature of class.

(Or)

If we need to prevent a class from being inherited, then we have to use **sealed** keyword for that class.

Example:

```
namespace Sealed_Clas
{
   public class MyBaseClass
   {
      public void Display()
      {
         Console.WriteLine("Base class");
      }
   }

   //Compile Success: This class cannot be inherited
   public sealed class MySealedClass : MyBaseClass
   {
      public void Dis()
      {
         Console.WriteLine("Sealed class");
      }
   }

   //Compilation Error: cannot derive from sealed type MySealedClass
   public class MyChildClass : MySealedClass
   {

   }

   class Program
   {
      static void Main(string[] args)
      {
         Console.ReadLine();
      }
   }
}
```

Output:

Error: **Sealed_Clas.MyChildClass** cannot derive from sealed type **Sealed_Clas.MySealedClass**

110) what is an indexer?

Indexer is a location indicator used to access class objects just like accessing elements in an array.

Example:

In the below example we have created new index for class of type **string**. During "**get"** and "**set"** operation string manipulations are done.

```
namespace Indexer
{
  public class MyClassForIndexer
  {
    private string m_Name = "This is example for indexer";
    public string this[int index]
    {
      get
      {
        return m_Name.Substring(index);
      }
      set
      {
        m_Name = m_Name.Insert(index, value);
      }
    }
}

  class Program
  {
    static void Main(string[] args)
    {
      MyClassForIndexer ind = new MyClassForIndexer();
      Console.WriteLine(ind[0]);
      ind[7] = "Appended String";
      Console.WriteLine(ind[0]);
      Console.ReadLine();
    }
  }
}
```

Output:

This is example for indexer

This is Appended String example for indexer

111) what is mean by Delegate?

Delegate is a type that holds a reference to a method or a function.

- Once a delegate is assigned a method, it behaves exactly like that method.
- We can call the method using delegate instead of directly calling the method.
- Using delegate, we can also pass the parameter and get return value. Any method with matched the signature of the delegate can be assigned.

Simply we can say .NET implements the concept of function pointers using delegate.

There are three steps to following for using Delegate

1. **Declaration**
2. **Instantiation**
3. **Invocation**

Example:

In the below example we have declared the new delegate with name **MyDelegate**, which accept **string** as parameter and return value as **string**. Two methods **SayHello** and **SayBye** function will be called using delegate.

```
//Declaring the delegate
    delegate string MyDelegate(string name);
    class delegateExample
    {
      //function called by delegate dynamically
      public static string SayHello(string name)
      {
        return "Hello " + name;
      }

      public static string SayBye(string name)
      {
        return "Bye " + name;
      }
    }
```

After declaration of delegate, we have initialized with **SayHello** function. Now this delegate will hold reference to specified function. Function will be called using **Invoke ()** method of delegate. In this example we have called two methods (**SayHello** and **SayBye**) with same signature (parameter type and return type).

```
static void Main(string[] args)
    {
      //Initialllizing delegate with function name
      MyDelegate delg = new MyDelegate(delegateExample.SayHello);
      //Invoking function using delegate
      Console.WriteLine(delg.Invoke("Sam"));

      delg = new MyDelegate(delegateExample.SayBye);
      //Invoking diffent function using same delegate
      Console.WriteLine(delg.Invoke("Sam"));

      Console.ReadLine();
    }
```

Output:

Hello sam
Bye sam

112) Difference between Value types and Reference types?

Value Types:

- Value Types cannot inherit from another class or **struct**.
- Value types can only inherit from interfaces.
- Value types are stored on the stack.

Reference Types:

- Reference types can inherit from another class or interface.
- Reference types are stored on the managed heap.

113) what is Thread?

Threads are the basic unit to which an operating system allocates processor time, and more than one thread can be executing code inside that process.

114) what is mean by process?

Process is a running thread of an application.

115) what is Multi-threading?

Multi-threading is the collection of thread executed with in the same program to generate the output.

116) what is Multi-tasking?

Multi-tasking means we can run multiple programs at same time example **Word, Excel,** etc.

117) what is the namespace used for threading?

System. Threading

.NET program always has at least two threads running one is the **main program** and second is the **Garbage collector.**

118) Can you explain in brief how can we implement threading?

This sample explains about the implementation of the threading.

Let start this example by creating a class with two methods called **"Thread1()", "Thread2()".**

```
using System.Threading;

namespace Threading
{
  class TestClass
  {
    public void Thread1()
    {
      int index = 0;
      for (index = 0; index < 100; index++)
      {
        Console.WriteLine("This is from first thread: {0}", index.ToString());
      }
    }

    public void Thread2()
    {
      int index = 0;
      for (index = 0; index < 100; index++)
      {
        Console.WriteLine("This is from second thread: {0}", index.ToString());
      }
    }
  }

  class Program
  {
    static void Main(string[] args)
    {
      TestClass objTestClass = new TestClass();
      Thread th1 = new Thread(new ThreadStart(objTestClass.Thread1));
      Thread th2 = new Thread(new ThreadStart(objTestClass.Thread2));
      th1.Start();
      th2.Start();
      Console.ReadLine();
    }
  }
}
```

119) what are different levels of threading priorities are available?

Priority of the thread execution can be changed by using the **"Priority"** property of the thread instance.

Example:

ThreadName.Priority = ThreadPriority.BelowNormal;

Following are the different level of the Thread priority available

1. **ThreadPriority.Highest**
2. **ThreadPriority.AboveNormal**
3. **ThreadPriority.Normal**
4. **ThreadPriority.BelowNormal**
5. **ThreadPriority.Lowest**

120) What is mean by Theard.Sleep()?

The **Thread.sleep()** method effectively **"pauses"** the current thread execution for a given period of time. This method takes an integer value as parameter that determines how long the thread needs to be paused.

Example: **System.Threading.Thread.Sleep(4000);**

121) How can we make a thread sleep for infinite period?

using *System.Threading.*Timeout*.Infinite*

122) What is mean by Thread.Suspend and Resume?

Thread*.Suspend() -* *This method is used to suspend the thread execution. If the method is already suspended, it does not have any effect.*

Thread*.Resume() -* *Suspended thread can be resumed using this method call.*

123) What the way to stop a long running thread?

Thread.Abort() stops the thread execution at that moment itself.

124) How will we get current thread instance?

using System.Threading.Thread.CurrentThread we will be able to get the current thread instance.

125) How we can make the thread run in background?

By setting **ThreadName.IsBackground = true** will run the Thread in background process.

Example:

```
TestClass _objTestClass = new TestClass();
Thread th1 = new Thread (new ThreadStart(_objTestClass.Thread1 ));
th1.IsBackground = true;
th1.Start();
```

126) How to debug the thread?

Threading application can be debugged using **Debug->Windows->Threads or "Ctrl+Alt+H"** in menu.

127) what are different states of a thread?

Thread status can be known by using **ThreadName.ThreadState** property.

Followings are the list of state of a thread,

- **ThreadState.Aborted**
- **ThreadState.AbortRequested**
- **ThreadState.Background**
- **ThreadState.Running**
- **ThreadState.Stopped**
- **ThreadState.StopRequested**
- **ThreadState.Suspended**
- **ThreadState.SuspendRequested**
- **ThreadState.Unstarted**
- **ThreadState.WaitSleepJoin**

128) Can we use events with threading?

Yes, we can use events with thread.

This is one of the techniques to synchronize one thread with other.

129) Can we have multiple constructors in class?

Yes we can have multiple constructors in class with different parameters.

Example:

```
class A
  {
    public A()
    {
      Console.WriteLine("Constructor A");
    }

    public A(string s)
    {
      Console.WriteLine("Constructor A with single parameter");
    }

    public A(string s, string t)
    {
      Console.WriteLine("Constructor A with two parameters");
    }
  }

  class Program
  {
    static void Main(string[] args)
    {
      A b1 = new A();
      A b2 = new A("vinay");
      A b3 = new A("vinay","kumar");
      Console.Read();
    }
  }
```

Output:

Constructor A

Constructor A with single parameter

Constructor A with two parameters

130) which Constructor gets fired first "Static" (or) "Non-Static" while creating an object of a class?

Static constructor will be fired first.

Example:

```
class Test
  {
    public Test()
    {
      Console.WriteLine(" public Test()");
    }

    static Test()
    {
      Console.WriteLine(" static Test()");
    }
  }

  class Program
  {
    static void Main(string[] args)
    {
      Test test = new Test();
      Console.Read();
    }
  }
```

Output:

Static Test()

Public Test()

131) Can we use static constructors to initialize non static members?

Yes, it is possible. But we have to create an object of the class inside the **static constructor** and then initialize the **non-static** member through the object reference.

Example:

```
class A
  {
    int a;
    static A()
    {
      A p = new A();
      p.a = 45;
      System.Console.WriteLine(p.a);
    }

    static void Main(string[] args)
    {
      Console.ReadLine();
    }
  }
```

Output: 45

132) Can abstract class have constructors?

Yes, an abstract class does have a constructor. It will be called when it subclass is instantiated.

Example:

```
abstract class bank
    {
        public bank()
        {
            Console.WriteLine("Bank");
        }
    }

   class icici : bank
    {
       icici()
       {
           Console.WriteLine("Icici");
       }
        static void Main(string[] args)
        {
            bank b = new icici();
            Console.ReadLine();
        }
    }
```

In the above example, when the object of "**icici**" class is created, they will be displayed in the output.

Output:

Bank

Icici

133) What will be the output of below code?

```
class A
  {
    class B
    {
      public string str = "welcome";
    }
  }

  class Program
  {
    static void Main(string[] args)
    {
      A.B Obj = new A.B();
      Console.WriteLine(Obj.str);
      Console.ReadLine();
    }
  }
```

Ans: The compiler will generate an error ->**The Type or Namespace 'B' could not be found.**

It is because since **B** is the inner class, it is now **private**.

Solution:

1) We have to change the access modifier of **B** to **internal/public/protected internal**

2) We have to give the full path of class **B** when creating the object.

i.e. A.B obj=new A.B();

Then we can see the value of the field **str** in the output.

Modified Example:

```
class A
  {
    internal class B (Or) public class B (Or) protected internal class B
    {
      public string str = "Welcome";
    }
  }

  class Program
  {
    static void Main(string[] args)
    {
      A.B Obj = new A.B();
      Console.WriteLine(Obj.str);
      Console.ReadLine();
    }
  }
```

Output:

Welcome

134) Difference between Default Constructor, Private Constructor and Static Constructor?

Default Constructor:

- ✓ A constructor that takes no parameters is called a default constructor.
- ✓ Default constructors are invoked whenever an object is instantiated by using the **NEW** operator.
- ✓ If a class does not have a constructor, a default constructor is automatically generated.

Private Constructor:

- ✓ **Private constructor** is called after the first instance of class is created. The private constructor will be executed each time it is called.
- ✓ A class with **Private constructor** cannot be inherited.
- ✓ **Private constructor** are used to prevent creating instances of a class.
- ✓ The private Constructor may have parameters.
- ✓ To access the methods or properties in a class with **private constructor** we need to assign methods or properties with "static" keyword.

If you try to instantiate a class, when you build your application you've got the below error,

Error: "... is inaccessible due to its protection level"

Example code :

```
public class Person
{
    private Person()
    {
    }
    public static void Speak()
    {
        Console.WriteLine("I spoke");
    }
}

class Program
{
    static void Main(string[] args)
    {
        Person person = new Person();
        // Test.StaticConstructor.Person.Person()' is inaccessible
due to its protection level
        Console.Read();
    }
}
```

Static Constructor:

- ✓ **Static constructor** is called before the first instance of class is created.
- ✓ **Static constructors** are used to initialize any static data.
- ✓ **Static constructor** called only once.
- ✓ In **Static constructor** you can't access any methods which are not static.
- ✓ The static constructor cannot have parameters.

Example code :

```
public class Car
{
    static Car()
    {
        Console.WriteLine("Static constructor is called");
    }

    public static string Color;
    public static void Drive()
    {
        Console.WriteLine("Car is driven");
    }
}
class Program
{
    static void Main(string[] args)
    {
        Car.Drive(); // static constructor is invoked
        Car car = new Car(); // static constructor is not invoked
        Car.Color = "Green"; // static constructor is not invoked

        Console.Read();
    }
}
```

Output :

```
Static constructor is called
Car is driven
```

135) why we can't create object for Static Class?

Generally Object can be created while executing the constructor only. But, Static blocks and Static Classes are executed before executing constructor. So, we can't create object for Static Class.

136) what is the difference between constant and Read-only fields?

Constant

- ✓ We can assign a value to **Constant** fields at the time of declaration and after that they can't be modified.
- ✓ Constant values will evaluate at compile time only.
- ✓ By default **constant** are **static**, hence you cannot define a **constant** type as **static**.

Note: Constant = Value assigned at Compile time and unchangeable once established.

```
public const int X = 10;
```

```
void Calculate(int Z)
{
 const int X = 10, X1 = 50;
 const int Y = X + X1; //no error, since its evaluated a compile time
 const int Y1 = X + Z; //gives error, since its evaluated at run time
}
```

```
const MyClass obj1 = null;//no error, since its evaluated a compile time
const MyClass obj2 = new MyClass();//gives error, since its evaluated at run time
```

Read-only

- ✓ We can assign a value to **Read-only** fields at the time of declaration (or) with in the Constructor of same class and after that they can't be modified.
- ✓ Read-only values will evaluate at runtime only.

Note: Read-only = value assigned at run time and unchangeable once established.

```
class MyClass
{
 readonly int X = 10; // initialized at the time of declaration
 readonly int X1;

 public MyClass(int x1)
 {
 X1 = x1; // initialized at run time
 }
}
```