1) Introduce yourself.
2) Tell me about your last project.
3) Which technology is used.
4) Explain Oops concepts.

----------C#----------

5) What is inheritance?

Acquiring (taking) the properties of one class into another class is called inheritance. Inheritance provides reusability by allowing us to extend an existing class. The reason behind OOP programming is to promote the reusability of code and to reduce complexity in code and it is possible by using inheritance. The inheritance concept is based on a base class and derived class. Let us see the definition of a base and derived class.

Base class - is the class from which features are to be inherited into another class. Derived class - it is the class in which the base class features are inherited.

Advantages

Reduce code redundancy.
Provides code reusability.
Reduces source code size and improves code readability.
Code is easy to manage and divided into parent and child classes.
Supports code extensibility by overriding the base class functionality within child classes.

Disadvantages

In Inheritance base class and child classes are tightly coupled. Hence If you change the code of parent class, it will get affects to the all the child classes.

In class hierarchy many data members remain unused and the memory allocated to them is not utilized. Hence affect performance of your program if you have not implemented inheritance correctly.

-Types of inheritance?.

1. Single Inheritance

when a single derived class is created from a single base class then the inheritance is called as single inheritance.

2. Hierarchical Inheritance

when more than one derived class are created from a single base class, then that inheritance is called as hierarchical inheritance.

3. Multi Level Inheritance
when a derived class is created from another derived class, then that inheritance is called as multi level inheritance.

4. Hybrid Inheritance
Any combination of single, hierarchical and multi level inheritances is called as hybrid inheritance.

5. Multiple Inheritance
when a derived class is created from more than one base class then that inheritance is called as multiple inheritance. But multiple inheritance is not supported by .net using classes and can be done using interfaces.

Handling the complexity that causes due to multiple inheritance is very complex. Hence it was not supported in dotnet with class and it can be done with interfaces.

6) What is constructor?

Constructor is a special method that get invoked/called automatically, whenever an object of a given class gets instantiated.

Constructor(s) in a class is/are special methods which get called automatically when an object of a class is created. Constructors are specially used to initialize data members. There are different types of constructors you can write in a class -

1. Default Constructor
2. Parameterized Constructor
3. Copy Constructor
4. Static Constructor

1. Default Constructor

When you do not declare any constructor, the class will call its default constructor which has a default public access modifier. The default constructor is a parameter less constructor which will be called by a class object.

Let's see an example of a Default Constructor -

```
public class Person
{
   private int m_PersonID;
   private string m_FirstName, m_LastName, m_City;
   public Person()
   {
      m_PersonID = 19929;
      m_FirstName = "No First Name";
      m_LastName = "No Last Name";
      m_City = "No City";
   }
}
```

2.Parameterized constructors

Now let’s see parameterized constructors. You can also call it as constructor overloading. Default constructors always initialize the objects with the same values. In case you want to initialize the class with different values, you can use Parameterized constructors.

```
public class Person
{
   private int m_PersonID;
   private string m_FirstName, m_LastName, m_City;
   public Person()
   {
      m_PersonID = 19929;
      m_FirstName = "No First Name";
      m_LastName = "No Last Name";
      m_City = "No City";
   }
   public Person(string firstName,string lastName)
   {
      m_FirstName = firstName;
      m_LastName = lastName;   }}
```

3.Copy Constructor

Now let's see an example of Copy Constructor. Copy constructor is the parameterized constructor which takes a parameter of the same type. It allows you to initialize a new object with the existing object values.

```
public class Person
{
   private int m_PersonID;
   private string m_FirstName, m_LastName, m_City;
   public Person()
   {
      m_PersonID = 19929;
      m_FirstName = "No First Name";
      m_LastName = "No Last Name";
      m_City = "No City";
   }
   public Person(string firstName,string lastName)
   {
      m_FirstName = firstName;
      m_LastName = lastName;
   }
   //copy constructor
   public Person(Person person)
   {
      m_PersonID = person.m_PersonID;
      m_FirstName = person.m_FirstName;
      m_LastName = person.m_LastName;
      m_City = person.m_City;
   }
}
```

Here’s an example:

```
// Instance constructor.
Person p1 = new Person(1, "DotNet", "Curry", "Pune");
// Copy Constructor
Person p2 = new Person(p1);
```

To invoke the parameterized constructor, use this Person p1 = new Person(“DotNet”, “Curry”);

4. Static Constructors

Static constructor is used to initialize the static data members of the class. Static constructor is only called once while creation of the first instance of the class. After that, no instance of a class will call the static constructor. You can also use static constructor to execute some code of the class which must be executed only once.

```
public class Person
{
   static Person()
   {
      //Static Members
   }}
```

In inheritance, the calling of the constructor starts from the parent class.
Let's see how to use these constructors -

```
static void Main(string[] args)
{
   Person p1 = new Person();//This will call Default Constructor
   Person p2 = new Person("Pravin", "D");//This will call two parameterized Constructor
   Person p3 = new Person(p2);//This will call Copy Constructor
}
```

It is worth mentioning that you can also create a private constructor, which is generally used in classes that contain static members only. If you create a private constructor, you cannot create an instance of a class.

```
class Person
{
   // Private Constructor:
   private Person() { }   ... }
```

What is a Destructor in C# ?
Destructor is a special method that get invoked/called automatically whenever an object of a given class gets destroyed. Main idea behind using destructor is to free the memory used by the object.

This default constructor will be executed whenever the class is initialized – Person p = new Person();

Note: If the class is abstract, then the accessibility of the default constructor is protected. Otherwise, the accessibility for the default constructor is public

-Explain the use of Virtual Keyword in C# ?
When we want to give permission to a derived class to override a method in base class, Virtual keyword is used.

7) What is Polymorphism?
   -Types of Polymorphism.
The ability of a programming language to process objects in different ways depending on their data type or class is known as Polymorphism. There are two types of polymorphism
   Compile time polymorphism. Best example is Overloading
   Runtime polymorphism. Best example is Overriding.

8) What is the difference between method overriding and method overloading?
In method overriding, we change the method definition in the derived class that changes the method behavior. Method overloading is creating a method with the same name within the same class having different signatures.

- Overloading (Called as Early Binding or Compile Time Polymorphism or static binding)
Explain Overloading in C# ?
When methods are created with the same name, but with different signature its called overloading. For example, WriteLine method in console class is an example of overloading. In the first instance, it takes one variable. In the second instance, “WriteLine” method takes two variable.

```
Console.WriteLine(x);
Console.WriteLine("The message is {0}", Message);
```

Different types of overloading in C# are

- Constructor overloading
- Function overloading
- Operator overloading

What is Constructor Overloading in C# .net ?

In Constructor overloading, n number of constructors can be created for the same class. But the signatures of each constructor should vary. For example

```
public class Employee
{
 public Employee()
 { }
 public Employee(String Name)
 { }
}
```

What is Function Overloading in C# .net ?

In Function overloading, n number of functions can be created for the same class. But the signatures of each function should vary. For example

```
public class Employee
{
 public void Employee()
 { }
 public void Employee(String Name)
 { }
}
```

What is Operator Overloading in C# .net ?

We had seen function overloading in the previous example. For operator Overloading, we will have a look at the example given below. We had defined a class rectangle with two operator overloading methods.

We had seen function overloading in the previous example. For operator Overloading, we will have a look at the example given below. We had defined a class rectangle with two operator overloading methods.

```
class Rectangle
{
 private int Height;
 private int Width;
 public Rectangle(int w,int h)
 {
   Width=w;
   Height=h;
 }
```

```
 public static bool operator >(Rectangle a,Rectangle b)
 {
   return a.Height > b.Height ;
 }
 public static bool operator <(Rectangle a,Rectangle b)
 {
   return a.Height < b.Height ;
 }
}
```

Let us call the operator overloaded functions from the method given below. When first if condition is triggered, the first overloaded function in the rectangle class will be triggered. When second if condition is triggered, the second overloaded function in the rectangle class will be triggered.

```
public static void Main()
{
Rectangle obj1 =new Rectangle();
Rectangle obj2 =new Rectangle();
 if(obj1 > obj2)
 {
  Console.WriteLine("Rectangle1 is greater than Rectangle2");
 }
 if(obj1 < obj2)
 {
  Console.WriteLine("Rectangle1 is less than Rectangle2");
 }
}
```

- Overriding (Called as Late Binding or Run Time Polymorphism or dynamic binding)

What is method Overriding?

1) When two or more methods (functions) have the exact same method name, return type, number of parameters, and types of parameters as the method in the parent class is called method Overriding.

```
class parentClass {
public virtual void Disp(int i)
{
System.Console.WriteLine("parentClass Display" + i);
} }
class childClass : parentClass {
 public override void Disp(int i)
{
System.Console.WriteLine("childClass Display" + i);
} }
```

2) If derived class defines same method as defined in its base class, it is known as method overriding in C#. It is used to achieve runtime polymorphism. It enables you to provide specific implementation of the method which is already provided by its base class.

To perform method overriding in C#, you need to use virtual keyword with base class method and override keyword with derived class method.

What is Data Encapsulation ?
Data Encapsulation is defined as the process of hiding the important fields from the end user. In the above example, we had used getters and setters to set value for MinSalary. The idea behind this is that, private field “minimumSalary” is an important part of our classes. So if we give a third party code to have complete control over the field without any validation, it can adversely affect the functionality. This is inline with the OOPS Concept that an external user should know about the what an object does. How it does it, should be decided by the program. So if a user set a negative value for MinSalary, we can put a validation in the set method to avoid negative values as shown below

```
set
{
 if(value > 0)
 {
  minSalary = value;
 }
}
```

--What is Method Hiding in C# ?
If the derived class doesn't want to use methods in the base class, derived class can implement it's own version of the same method with same signature. For example, in the classes given below, DriveType() is implemented in the derived class with same signature. This is called Method Hiding.

9) Explain interface.

Can Multiple Inheritance implemented in C# ?
In C#, derived classes can inherit from one base class only. If you want to inherit from multiple base classes, use interface.

10) What’s the difference between an interface and abstract class?
Interfaces have all the methods having only declaration but no definition. In an abstract class, we can have some concrete methods. In an interface class, all the methods are public. An abstract class may have private methods.

A class can implement any number of interfaces but a subclass can at most use only one abstract class.
An abstract class can have non-abstract methods (concrete methods) while in case of interface all the methods has to be abstract.
An abstract class can declare or use any variables while an interface is not allowed to do so.
In an abstract class all data member or functions are private by default while in interface all are public, we can’t change them manually.
In an abstract class we need to use abstract keyword to declare abstract methods while in an interface we don’t need to use that.
An abstract class can’t be used for multiple inheritance while interface can be used as multiple inheritance.
An abstract class use constructor while in an interface we don’t have any type of constructor.

What is an Interface in C# ?
An interface is similar to a class with method signatures. There wont be any implementation of the methods in an Interface. Classes which implement interface should have an implementation of methods defined in the abstract class.

What is Abstract Class in C#?
If we don't want a class to be instantiated, define the class as abstract. An abstract class can have abstract and non abstract classes. If a method is defined as abstract, it must be implemented in derived class. For example, in the classes given below, method DriveType is defined as abstract.

```
abstract class Car
{ public Car() {  Console.WriteLine("Base Class Car"); }
 public abstract void DriveType();}
class Ford : Car{ public void DriveType()
 {  Console.WriteLine("Right Hand "); }}
```

11) What you mean by delegate in C#?
Delegates are type safe pointers unlike function pointers as in C++. Delegate is used to represent the reference of the methods of some return type and parameters.
-What are the types of delegates in C#?
Below are the uses of delegates in C# -

- Single Delegate
- Multicast Delegate
- Generic Delegate

- What are the three types of Generic delegates in C#?
Below are the three types of generic delegates in C# -

- Func
- Action
- Predicate

- What are the differences between events and delegates in C#?
Main difference between event and delegate is event will provide one more of encapsulation over delegates. So when you are using events destination will listen to it but delegates are naked, which works in subscriber/destination model.
-Can we use delegates for asynchronous method calls in C#?
Yes. We can use delegates for asynchronous method calls.
-What are the uses of delegates in C#?
Below are the list of uses of delegates in C# -

- Callback Mechanism
- Asynchronous Processing
- Abstract and Encapsulate method
- Multicasting

12) Explain Action.
This is a function object. Action objects return no values. The Action type is similar to a void method. This generic type is found in the System namespace. ex-void
To specify an Action, we must have no return value. The Action must never return a value onto the evaluation stack. Often we use lambdas to specify Actions. ex-Lambdas
Start. The Actions point to anonymous functions. These functions cannot return values onto the evaluation stack. An Action instance can receive parameters, but cannot return values. ex-Return

An Action is a type of delegate:

- It returns no value.
- It may take 0 parameter to 16 parameters.

13)What is LINQ in C#?
LINQ stands for Language Integrated Query. LINQ is a data querying methodology which provides querying capabilities to .NET languages with a syntax similar to a SQL query
LINQ has a great power of querying on any source of data. The data source could be collections of objects, database or XML files. We can easily retrieve data from any object that implements the IEnumerable<T> interface.
Advantages of LINQ
LINQ offers an object-based, language-integrated way to query over data no matter where that data came from. So through LINQ we can query database, XML as well as collections.
Compile time syntax checking.
It allows you to query collections like arrays, enumerable classes etc in the native language of your application, like VB or C# in much the same way as you would query a database using SQL.

14) Explain sealed class in C#?
Sealed class is used to prevent the class from being inherited from other classes. So "sealed" modifier also can be used with methods to avoid the methods to override in the child classes.

15) List out the differences between Array and ArrayList in C#?
Array stores the values or elements of same data type but arraylist stores values of different datatypes.
Arrays will use the fixed length but arraylist does not uses fixed length like array.

16) 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 other namespace.

17) What is the difference between "constant" and "readonly" variables in C#?
"Const" keyword is used for making an entity constant. We cannot modify the value later in the code. Value assigning is mandatory to constant variables.
"readonly" variable value can be changed during runtime and value to readonly variables can be assigned in the constructor or at the time of declaration.

18) Explain "static" keyword in C#?
"Static" keyword can be used for declaring a static member. If the class is made static then all the members of the class are also made static. If the variable is made static then it will have a single instance and the value change is updated in this instance.

19) What is the difference between "dispose" and "finalize" variables in C#?
Dispose - This method uses interface – "IDisposable" interface and it will free up both managed and unmanaged codes like – database connection, files etc.
Finalize - This method is called internally unlike Dispose method which is called explicitly. It is called by garbage collector and can't be called from the code.

20) How the exception handling is done in C#?
In C# there is a "try… catch" block to handle the error.

21) 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.

22) What is the difference between “finalize” and “finally” methods in C#?
Finalize – This method is used for garbage collection. So before destroying an object this method is called as part of clean up activity.
Finally – This method is used for executing the code irrespective of exception occurred or not.

23) 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.

24) Explain the features of C#?
Below are some of the features supported in C# -
Constructors and Destructors
Properties, Passing Parameters, Arrays, Main, XML Documentation and, Indexers

25) List some of the advantages of C#?
Below are the advantages of C# -
Easy to learn, Object oriented, Component oriented, Part of .NET framework

26) What are the differences between static, public and void in C#?
Static classes/methods/variables are accessible throughout the application without creating instance. Compiler will store the method address as an entry point.
Public methods or variables are accessible throughout the application.
Void is used for the methods to indicate it will not return any value.

27) 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.

28) Explain Jagged Arrays in C#?
If the elements of an array is an array then it’s called as jagged array. The elements can be of different sizes and dimensions.

29) Can we override private virtual method in C#?
No. We can’t override private virtual methods as it is not accessible outside the class.

30) What are reference types in C#?
Below are the list of reference types in C# -
class, string, interface, object

31) What are value types in C#?
Below are the list of value types in C# -
decimal, int, byte, enum, double, long, float

32) Explain access modifier – “protected internal” in C#?
“protected internal” can be accessed in the same assembly and the child classes can also access these methods.

33) What is the difference between “StringBuilder” and “String” in C#?
StringBuilder is mutable, which means once object for stringbuilder is created, it later be modified either using Append, Remove or Replace.
String is immutable and it means we cannot modify the string object and will always create new object in memory of string type.

34) How we can sort the array elements in descending order in C#?
“Sort()” method is used with “Reverse()” to sort the array in descending order.

35) List out some of the exceptions in C#?
Below are some of the exceptions in C# -
NullReferenceException
ArgumentNullException
DivideByZeroException
IndexOutOfRangeException
InvalidOperationException
StackOverflowException etc.

36) What is Nullable Types in C#?
Variable types does not hold null values so to hold the null values we have to use nullable types. So nullable types can have values either null or other values as well.
Eg: Int? mynullablevar = null;

37) Is C# code is unmanaged or managed code?
C# code is managed code because the compiler – CLR will compile the code to Intermediate Language.

38) Explain Hashtable in C#?
It is used to store the key/value pairs based on hash code of the key. Key will be used to access the element in the collection. For example,
Hashtable myHashtbl = new Hashtable();
myHashtbl.Add("1", "TestValue1");
myHashtbl.Add("2", "TestValue2");

39) What is enum in C#?
enum keyword is used for declaring an enumeration, which consists of named constants and it is called as enumerator lists. Enums are value types in C# and these can’t be inherited. Below is the sample code of using Enums
Eg: enum Fruits { Apple, Orange, Banana, WaterMelon};

40) What is the difference between “continue” and “break” statements in C#?
“continue” statement is used to pass the control to next iteration. This statement can be used with – “while”, “for”, “foreach” loops.
“break” statement is used to exit the loop.

41)What you mean by boxing and unboxing in C#?
Boxing – This is the process of converting from value type to reference type. For example,
int myvar = 10;
object myObj = myvar;

UnBoxing – It's completely opposite to boxing. It's the process of converting reference type to value type. For example,
int myvar2 = (int)myObj;

42)Explain Partial Class in C#?
Partial classes concept added in .Net Framework 2.0 and it allows us to split the business logic in multiple files with the same class name along with "partial" keyword.

43) Explain Anonymous type in C#?
This is being added in C# 3.0 version. This feature enables us to create an object at compile time. Below is the sample code for the same –
Var myTestCategory = new { CategoryId = 1, CategoryName = "Category1"};

44)Explain Copy constructor in C#?
If the constructor contains the same class in the constructor parameter then it is called as copy constructor.

```
class MyClass
{
 public string prop1, prop2;
 public MyClass(string a, string b)
 {
 prop1 = a;
 prop2 = b;
 }

 public MyClass(MyClass myobj) // Copy Constructor
 {
 prop1 = myobj.prop1;
 prop2 = myobj.prop2;
 }
}
```

45) Explain Static constructor in C#?
If the constructor is declared as static then it will be invoked only once for all number of instances of a class. Static constructor will initialize the static fields of a class.

```
class MyClass
{
 public string prop1, prop2;
 public MyClass(string a, string b)
 {
 prop1 = a;
 prop2 = b;
 }
```

```
Static MyClass()
 {
 Console.WriteLine("Static Constr Test");
 }
 public MyClass(MyClass myobj) // Copy Constructor
 {
 prop1 = myobj.prop1;
 prop2 = myobj.prop2;
 }
}
```

46) Explain Indexers in C#?
Indexers are used for allowing the classes to be indexed like arrays. Indexers will resemble the property structure but only difference is indexer's accessors will take parameters. For example,

```
class MyCollection<T>
{
 private T[] myArr = new T[100];
 public T this[int t]
 {
 get
 {
 return myArr[t];
 }
 set
 {
 myArr[t] = value;
 }
 }
}
```

47) What are the collection types can be used in C#?
Below are the collection types in C# -

- ArrayList
- Stack
- Queue
- SortedList
- HashTable
- Bit Array

48) Explain Attributes in C#?
Attributes are used to convey the info for runtime about the behavior of elements like – "methods", "classes", "enums" etc.
Attributes can be used to add metadata like – comments, classes, compiler instruction etc.
Below are the predefined attributes in C# -

- Conditional
- Obsolete
- Attribute Usage

49) What is Thread in C#?

Thread is an execution path of a program. Thread is used to define the different or unique flow of control. If our application involves some time consuming processes then it's better to use Multithreading., which involves multiple threads.

Below are the states of thread –

- Unstarted State
- Ready State
- Not Runnable State
- Dead State

Below are the methods and properties of thread class –

- CurrentCulture
- CurrentThread
- CurrentContext
- IsAlive
- IsThreadPoolThread
- IsBackground
- Priority

50) What is a class ?

A class is the generic definition of what an object is. A Class describes all the attributes of the object, as well as the methods that implement the behavior of the member object. In other words, class is a template of an object. For ease of understanding a class, we will look at an example. In the class Employee given below, Name and Salary are the attributes of the class Person. The Setter and Getter methods are used to store and fetch data from the variable.

51) What is an Object?

An object is an instance of a class. It contains real values instead of variables. For example, let us create an instance of the class Employee called "John".

52) What are the Access Modifiers in C# ?

Different Access Modifier are - Public, Private, Protected, Internal, Protected Internal

Public – When a method or attribute is defined as Public, it can be accessed from any code in the project. For example, in the above Class "Employee" getName() and setName() are public.

Private - When a method or attribute is defined as Private, It can be accessed by any code within the containing class only. For example, in the above Class "Employee" attributes name and salary can be accessed within the Class Employee Only. If an attribute or class is defined without access modifiers, it's default access modifier will be private.

Protected - When attribute and methods are defined as protected, it can be accessed by any method in the inherited classes and any method within the same class. The protected access modifier cannot be applied to classes and interfaces. Methods and fields in a interface can't be declared protected.

Internal – If an attribute or method is defined as Internal, access is restricted to classes within the current project assembly.

Protected Internal – If an attribute or method is defined as Protected Internal, access is restricted to classes within the current project assembly and types derived from the containing class.

53) Explain Static Members in C# ?
If an attribute's value had to be same across all the instances of the same class, the static keyword is used. For example, if the Minimum salary should be set for all employees in the employee class, use the following code.
private static double MinSalary = 30000;
To access a private or public attribute or method in a class, at first an object of the class should be created. Then by using the object instance of that class, attributes or methods can be accessed. To access a static variable, we don't want to create an instance of the class containing the static variable. We can directly refer that static variable as shown below.
double var = Employee.MinSalary ;

54) What is serialization?
When we want to transport an object through network then we have to convert the object into a stream of bytes. The process of converting an object into a stream of bytes is called Serialization. For an object to be serializable, it should implement ISerialize Interface. De-serialization is the reverse process of creating an object from a stream of bytes.

55) What is a garbage collector?
Garbage collector is a background process which checks for unused objects in the application and cleans them up.
What are generations in garbage collector?
--> Generation define the age of the object.

How many types of Generations are there in GC garbage collector?
--> There are 3 generations gen 0 , gen 1 and gen 2.

Does garbage collector clean unmanaged objects ?
--> No.

When we define clean up destructor , how does it affect garbage collector?
--> If you define clean up in destructor garbage collector will take more time to clean up the objects and more and more objects are created in Gen 2..

How can we overcome the destructor problem ?
--> By implementing "Idisposable" interface.

Where do we normally put unmanaged code clean up?.
--> In finalize a.k.a destructor.

When you create any object in C#, CLR (common language runtime) allocates memory for the object from heap. This process is repeated for each newly created object, but there is a limitation to everything, Memory is not un-limited and we need to clean some used space in order to make room for new objects, Here, the concept of garbage collection is introduced, Garbage collector manages allocation and reclaiming of memory. GC (Garbage collector) makes a trip to the heap and collects all objects that are no longer used by the application and then makes them free from memory.