OOPS Interview Questions

1. What is OOPs?

Object-Oriented Programming(OOPs) is a type of programming that is based on objects rather than just functions and procedures. Individual objects are grouped into classes. OOPs implements real-world entities like inheritance, polymorphism, hiding, etc into programming. It also allows binding data and code together.

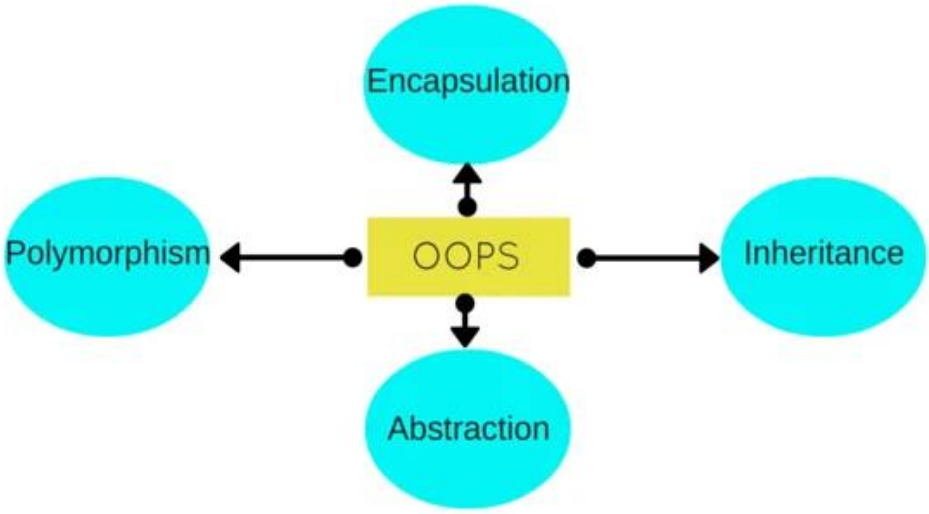
2. What is the difference between OOP and SOP?

Object-Oriented Programming	Structural Programming
Object-Oriented Programming is a type of programming which is based on objects rather than just functions and procedures	Provides logical structure to a program where programs are divided functions
Bottom-up approach	Top-down approach
Provides data hiding	Does not provide data hiding
Can solve problems of any complexity	Can solve moderate problems
Code can be reused thereby reducing redundancy	Does not support code reusability

3. What are the main features of OOPs?

Following are the basic features of OOPs –

- Inheritance
- Encapsulation
- Polymorphism
- Data Abstraction



4. What is an object?

An object is a real-world entity which is the basic unit of OOPs for example chair, cat, dog, etc. Different objects have different states or attributes, and behaviors.

5. What is a class?

A class is a prototype that consists of objects in different states and with different behaviors. It has a number of methods that are common the objects present within that class.

6. What is the difference between a class and a structure?

Class: User-defined blueprint from which objects are created. It consists of methods or set of instructions that are to be performed on the objects.

Structure: A structure is basically a user-defined collection of variables which are of different data types.

7. What is inheritance?

Inheritance is a feature of OOPs which allows classes inherit common properties from other classes. For example, if there is a class such as ‘vehicle’, other classes like ‘car’, ‘bike’, etc can inherit common properties from the vehicle class. This property helps you get rid of redundant code thereby reducing the overall size of the code.

8. What are the different types of inheritance?

- Single inheritance
- Multiple inheritance
- Multilevel inheritance
- Hierarchical inheritance
- Hybrid inheritance

9. What is the difference between multiple and multilevel inheritance?

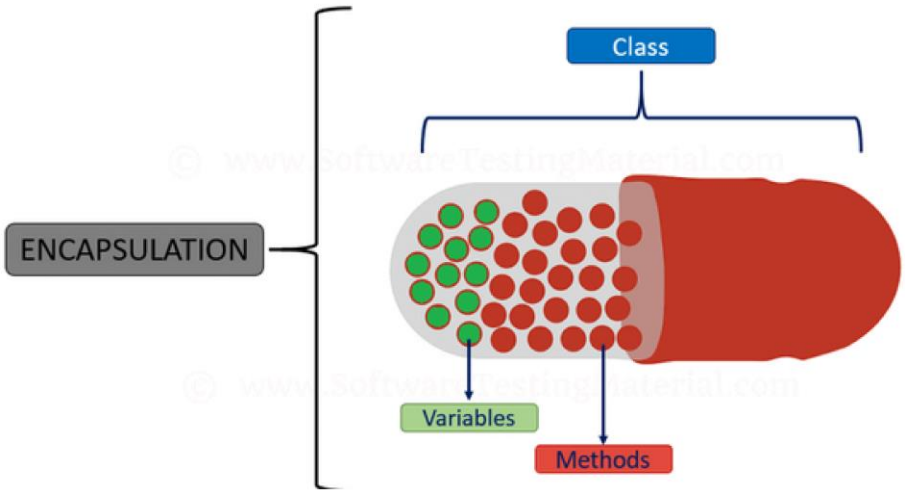
Multiple Inheritance	Multilevel Inheritance
Multiple inheritance comes into picture when a class inherits more than one base class	Multilevel inheritance means a class inherits from another class which itself is a subclass of some other base class
Example: A class defining a child inherits from two base classes Mother and Father	Example: A class describing a sports car will inherit from a base class Car which inturn inherits another class Vehicle

10. Can you call the base class method without creating an instance?

Yes, you can call the base class without instantiating it if:

- It is a static method
- The base class is inherited by some other subclass

11. What is encapsulation?



InterviewBit

Encapsulation is the method of putting everything that is required to do the job, inside a capsule and presenting that capsule to the user. What it means is that by Encapsulation, all the necessary data and methods are bind together and all the unnecessary details are hidden to the normal user. **So Encapsulation is the process of binding data members and methods of a program together to do a specific job, without revealing unnecessary details.**

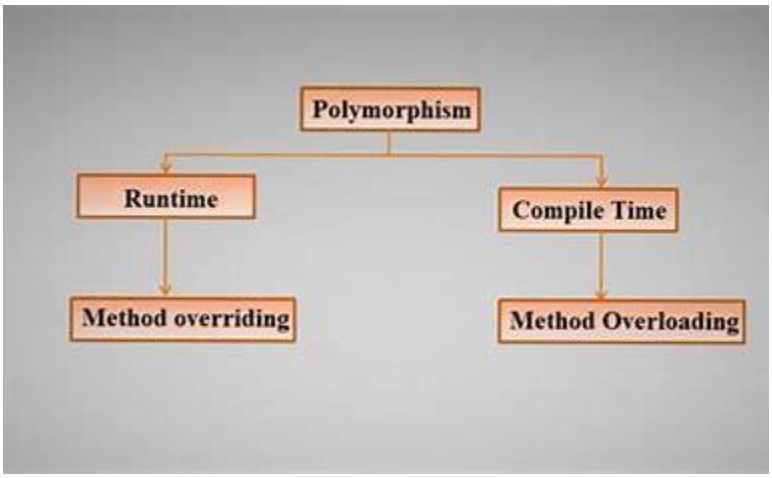
Encapsulation can also be defined in two different ways:

1) **Data hiding:** Encapsulation is the process of hiding unwanted information, such as restricting access to any member of an object.

2) **Data binding:** Encapsulation is the process of binding the data members and the methods together as a whole, as a class.

12. What is Polymorphism?

Polymorphism is composed of two words - “poly” which means “many”, and “morph” which means “shapes”. Therefore Polymorphism refers to something that has many shapes.



In OOPs, Polymorphism refers to the process by which some code, data, method, or object behaves differently under different circumstances or contexts. Compile-time polymorphism and Run time polymorphism are the two types of polymorphisms in OOPs languages.

13. What is Compile time Polymorphism and how is it different from Runtime Polymorphism?

Compile Time Polymorphism: Compile time polymorphism, also known as Static Polymorphism, refers to the type of Polymorphism that happens at compile time. What it means is that the compiler decides what shape or value has to be taken by the entity in the picture.

Example:

```
// In this program, we will see how multiple functions are created with the same name,
// but the compiler decides which function to call easily at the compile time itself.

class CompileTimePolymorphism{
  // 1st method with name add
  public int add(int x, int y){
  return x+y;
  }
  // 2nd method with name add
  public int add(int x, int y, int z){
  return x+y+z;
  }
  // 3rd method with name add
```

```
  public int add(double x, int y){
  return (int)x+y;
  }
  // 4th method with name add
  public int add(int x, double y){
  return x+(int)y;
  }
}
class Test{
  public static void main(String[] args){
  CompileTimePolymorphism demo=new CompileTimePolymorphism();
  // In the below statement, the Compiler looks at the argument types and
decides to call method 1
  System.out.println(demo.add(2,3));
  // Similarly, in the below statement, the compiler calls method 2
  System.out.println(demo.add(2,3,4));
  // Similarly, in the below statement, the compiler calls method 4
  System.out.println(demo.add(2,3.4));
  // Similarly, in the below statement, the compiler calls method 3
  System.out.println(demo.add(2.5,3));
  }
}
```

In the above example, there are four versions of add methods. The first method takes two parameters while the second one takes three. For the third and fourth methods, there is a change of order of parameters. The compiler looks at the method signature and decides which method to invoke for a particular method call at compile time.

Runtime Polymorphism: Runtime polymorphism, also known as Dynamic Polymorphism, refers to the type of Polymorphism that happens at the run time. What it means is it can't be decided by the compiler. Therefore what shape or value has to be taken depends upon the execution. Hence the name Runtime Polymorphism.

Example:

```
class AnyVehicle{
  public void move(){
  System.out.println("Any vehicle should move!!");
  }
}
class Bike extends AnyVehicle{
  public void move(){
  System.out.println("Bike can move too!!");
  }
}
class Test{
```

```
  public static void main(String[] args){
  AnyVehicle vehicle = new Bike();
  // In the above statement, as you can see, the object vehicle is of type
AnyVehicle
  // But the output of the below statement will be “Bike can move too!!”,
  // because the actual implementation of object ‘vehicle’ is decided during
runtime vehicle.move();
  vehicle = new AnyVehicle();
  // Now, the output of the below statement will be “Any vehicle should move!!”,
  vehicle.move();
  }
}
```

As the method to call is determined at runtime, as shown in the above code, this is called runtime polymorphism.

14. What is base keyword?

One uses base keyword to access members of the base class from a derived class.

15. What are Virtual, Override, and New keywords?

- Virtual is used to modify a method, property, indexer, or event declared in the base class and allows it to be overridden in the derived class.
- Override is used to extend or modify a virtual/abstract method, property, indexer, or event of the base class into the derived class.
- New is used to hide a method, property, indexer, or event of the base class into the derived class.
-

16. What is the difference between struct and class?

Category	Struct	Class
Type	It is value type	It is reference type
Inherits from	It inherits from System.Value type	It inherits from System.Object type
Used for	Usually used for smaller amounts of data	Usually used for large amounts of data
Inherited to	It can not be inherited to other type	It can be inherited to other class
Abstract	It can not be abstract	It can be abstract type
New keyword	No need to create object by new keyword	Can not use an object of a class with using new keyword

Default constructor	Do not have permission to create any default constructor	You can create a default constructor

17. What is interface and why to use interface?

- An interface looks like a class, but it has no implementation.
- The only thing it contains is declarations of events, indexers, methods and/or properties.
- The reason interfaces only provide declarations is because they are inherited by structs and classes, which must provide an implementation for each interface member declared.

- **Interfaces are mainly used for,**
    - Extensibility
    - Implementation Hiding
    - Accessing object using interfaces
    - Loose coupling

18. What is abstract class . Define in detail?

- An abstract class is a special kind of class that can not be instantiated.
- So, the question is why do you need a class that can not be instantiated?
- An abstract class is only to be sub-classed (inherited from).
- In other words, it only allows other classes to inherit from it but can not be instantiated.
- The key advantage is that it enforces certain hierarchies for all the sub classes.
- In simple words, it is a kind of contract that forces all the sub classes to carry on the same hierarchies or standards.
  The abstract keyword can be used with class, methods, properties, indexers and events.

19. Can Abstract class be Sealed?

- No, an abstract class cannot be a sealed class.
- Because, the sealed modifier prevents a class from being inherited and the abstract modifier requires a class to be inherited.

20. Can abstract class have Constructors ?

Yes, Abstract class can have constructor

21. Can you declare abstract methods as private ?

No. Abstract methods cannot be private

22. Can abstract class have static methods ?

Yes, Abstract class can have static methods

23. Does Abstract class support multiple Inheritance?

No, Abstract class does not support multiple Inheritance.

24. Abstract class must have only abstract methods. Is it true or false?

False. Abstract class can have both abstract and non abstract methods.

25. When do you use Abstract Class?

When you have a requirement where your base class should provide the default implementation of certain methods whereas other methods should be open to being overridden by child classes that time you have to use abstract classes.

26. Why can Abstract class not be Instantiated?

- Because, it has not fully implemented the class as its abstract methods can not be executed.
- If the compiler allows us to create the object for the abstract class, then you can invoke the abstract method using that object which cannot be executed by CLR at runtime.
- Hence, to restrict the calling of abstract methods, the compiler does not allow you to instantiate an abstract class.

27. Which type of members can you define in an Abstract class?

You can define all static and non-static members including properties, fields, indexers and also abstract methods.

28. What is Operator Overloading?

- Overloaded operators are functions with special names the keyword operator followed by the symbol for the operator being defined.
- Similar to any other function, an overloaded operator has a return type and a parameter list.\

29. Is it possible to restrict object creation?

Yes, it is possible to restrict object creation by using following,

- Abstract Class
- Static Class
- Private or Protected Constructor

30. Can you inherit Enum ?

- No, you cannot inherit Enum .
- Because, Enums are by default sealed. So, you can not inherit them.

31. Is it possible to achieve Method extension using Interface?

- Yes, it is possible to achieve Method extension using Interface.
- Most of the LINQ is built around interface extension methods.
- Interfaces were actually one of the driving forces for the development of extension methods.
- Since they can not implement any of their own functionality and extension methods are the easiest way of associating actual code with interface definitions.

32. Is it possible that a Method can return multiple values at a time?

- Yes, it is it possible that a Method can return multiple values at a time by using the following:

- KeyValue pair
- Ref or Out parameters
- Struct or Class
- Tuple

33. What is Constant?

- Constant is known as “const” keyword .
- It is also known as immutable values.
- Which are known at compile time and do not change their values at run time like in any function or constructor for the life of application till the application is running.

34. What is Readonly?

- Readonly is known as “readonly” keyword .
- It is also known immutable values.
- They are known at compile and run time and do not change their values at run time like in any function for the life of application till the application is running.
- You can assay their value by constructor when we call constructor with “new” keyword.

35. What is Static keyword?

- The static keyword is used to specify a static member.
- It means static members are common to all the objects and they do not get tied to a specific object.
- Static keyword can be used with classes, fields, methods, properties, operators, events, and constructors
- But, static can not be used with indexers, destructors, or types other than classes.
- Key points about Static keyword,
    - If the static keyword is applied to a class, all the members of the class must be static.
    - Static methods can only access static members of same class.
    - Static properties are used to get or set the value of static fields of a class.
    - Static constructor cannot be parameterized.
    - Access modifiers cannot be applied on static constructor. Because, it is always a public default constructor which is used to initialize static fields of the class.

36. What is Static ReadOnly?

- Static Readonly type variable value can be assigned at runtime or at compile time and can be changed at runtime.
- Such variable's value can only be changed in the static constructor and can not be changed further.
- It can change only once at runtime.

37. Can “this” be used within a Static Method?

- No, “this” cannot be used within a static method.
- Because, keyword 'this' returns a reference to the current instance of the class containing it.
- Static methods (or any static member) do not belong to a particular instance.
- They exist without creating an instance of the class and call with the name of a class not by instance so you can not use this keyword in the body of static methods.

38. What is the Difference between this and base ?

- **THIS**
    - THIS refers to the current instance (not the “current class”).
    - It can only be used in non-static methods/members. Because, in a static method there is no current instance.
    - Calling a method on this will call the method in the same way as it would if you called it on a variable containing the same instance.
- **BASE**
    - BASE is a keyword that allows inherited method call, i.e. it calls the specified method from the base type.
    - It can only be used in a non-static method.
    - It is usually used in a virtual method override, but actually can be used to call any method in the base type.
    - It is very different from normal method invocation because it circumvents the normal virtual-method dispatch.
    - It calls the base method directly even if it is virtual.

39.What is Partial Class?

- A partial class is only used to split the definition of a class in two or more classes in a same source code file or more than one source files.

- You can create a class definition in multiple files but it will be compiled as one class at run time and also when you'll create an instance of this class. So, you can access all the methods from all source file with a same object.
- Partial Class can be created in the same namespace and it does not allowed to create a partial class in different namespace.
- You can use "partial" keyword with all the class name which you want to bind together with the same name of class in same namespace.

40. What is Sealed Class?

- Sealed class is used to restrict the inheritance feature of object oriented programming.
- Once a class is defined as a sealed class, the class can not be inherited.
- A class, which restricts inheritance for security reason is declared as a sealed class.
- Sealed class is the last class in the hierarchy.
- Sealed class can be a derived class but cannot be a base class.
- A sealed class can not also be an abstract class. Because abstract class has to provide functionality and here you are restricting it to inherit.
- If you have ever noticed, structs and enum are sealed.