1. **OOPs and its principles.**

The four principles of object-oriented programming are encapsulation, abstraction, inheritance, and polymorphism.

* 1. **Abstraction**: *Abstractions is the design time process in which we hide the implementation and show only necessary things. For example, in C# we have a method called as Console.WriteLine which is use to display text on the console. We use this method everywhere but don’t know the internal implementation detail of this method. The part we can see is the method definition, the input parameters as well as return type of the method. This is what is called as abstraction. In this Console.Writeline method Microsoft is exposing only the required details (that is method name, input types and return types) but they are not exposing the implementation details of the method this is what is called as abstraction.*
  2. **Encapsulation:** *Encapsulation is implementation time process. It is a process of binding the data members and member functions into a single unit.*

*Let's assume we have to create a method to insert user’s data and pass it to other developers to use. So first, I need to create a class with a “AddUser” method to insert the data into database with validation.*

*Let there will be three fields: Name, Email and Phone number. So these inputs have to validate first and then insert into db.**For this we have to create two private methods one for validation and another for db insertion. So these methods cannot be accessed from outside of class and whole add user functionality is encapsulated inside a class.*

*Here, we are hiding the procedure of adding data into database from other users, this is Abstraction. And putting all the three methods into one User class and providing other users to use it, that is called Encapsulation.*

*So procedure hiding is Abstraction and putting every necessary thing into one is Encapsulation.*

* 1. **Polymorphism:** *In object-oriented programming world, polymorphism is expressed as 'one interface, multiple functions'. Polymorphism can be static or dynamic. In static polymorphism, the response to a function is determined at the compile time. In dynamic polymorphism, it is decided at run-time.*
* ***Static or Compile Time Polymorphism***

*It is also known as Early Binding. Method overloading is an example of Static Polymorphism. In Overloading, the method / function has the same name but different signatures. It is also known as Compile Time Polymorphism because the decision of which method is to be called is made at compile time. Overloading is the concept in which method names are the same with a different set of parameters.*

*Here the compiler checks the number of parameters passed and the type of parameter and make the decision of which method to call and it throw an error if no matching method is found.*

* ***Dynamic / Runtime Polymorphism***

*Dynamic / runtime polymorphism is also known as late binding. Here, the method name and the method signature (number of parameters and parameter type must be the same and may have a different implementation). Method overriding is an example of dynamic polymorphism.*

*Method overriding can be done using inheritance. With method overriding it is possible for the base class and derived class to have the same method name and same something. The compiler would not be aware of the method available for overriding the functionality, so the compiler does not throw an error at compile time. The compiler will decide which method to call at runtime and if no method is found then it throws an error.*

* 1. **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.*

*Types:*

*Single, Multilevel, Hierarchical (This is the type of inheritance in which there are multiple classes derived from one base class. This type of inheritance is used when there is a requirement of one class feature that is needed in multiple classes).*

1. **OOPs- Association, aggregations and composition.**
   1. **Association:** *Association is a relationship among the objects. Association is "\*a\*" relationship among objects. In Association, the relationship among the objects determine what an object instance can cause another to perform an action on its behalf. We can also say that an association defines the multiplicity among the objects. We can define a one-to-one, one-to-many, many-to-one and many-to-many relationship among objects. Association is a more general term to define a relationship among objects. Association means that an object "uses" another object.*

*For example, Managers and Employees, multiple employees may be associated with a single manager and a single employee may be associated with multiple managers.*

* 1. **Aggregation:** *Aggregation is a special type of Association. Aggregation is "\*the\*" relationship among objects. We can say it is a direct association among the objects. In Aggregation, the direction specifies which object contains the other object. There are mutual dependencies among objects.*

*For example, departments and employees, a department has many employees, but a single employee is not associated with multiple departments.*

*UML Representation of Aggregation (white diamond).*

*Here, the lives of both objects are independent of each other. That means that in this Association (Aggregation) the object has their own life cycle. Employees may exist without a department. Here, department can be called an owner object and the employee can be called a child object. The owner and child objects cannot belong to a different parent object.*

* 1. **Composition:** *Composition is special type of Aggregation. It is a strong type of Aggregation. In this type of Aggregation, the child object does not have their own life cycle. The child object's life depends on the parent's life cycle. Only the parent object has an independent life cycle. If we delete the parent object, then the child object(s) will also be deleted. We can define the Composition as a "Part of" relationship.*

*For example, the company and company location, a single company has multiple locations. If we delete the company then all the company locations are automatically deleted. The company location does not have their independent life cycle, it depends on the company object's life (parent object).*

*UML Representation of Composition (black diamond).*

1. **OOPS- SOLID principles and implementation.**
   1. *S stands for SRP (Single responsibility principle): - A class should take care of only one responsibility.*
   2. *O stands for OCP (Open closed principle): - Extension should be preferred over modification.*
   3. *L stands for LSP (Liskov substitution principle): - A parent class object should be able to refer child objects seamlessly during runtime polymorphism.*
   4. *I stand for ISP (Interface segregation principle): - Client should not be forced to use a interface if it does not need it.*
   5. *D stands for DIP (Dependency inversion principle): - High level modules should not depend on low level modules but should depend on abstraction.*
2. **C#- IOC containers.**

*IOC means Inversion of control. IOC is a concept where the flow of application is inverted. So for example rather than the caller calling the method.*

1. **OOPs- Static class vs Singleton class. Why should we use singleton over static classes?**
   1. **Singleton Class:** *Ensures a class has only one instance and provides a global point of access to it.*

*A singleton is a class that only allows a single instance of itself to be created, and usually gives simple access to that instance.*

*Most commonly, singletons don't allow any parameters to be specified when creating the instance, since a second request of an instance with a different parameter could be problematic*

***Advantages:***

*- One of the key advantages of singleton over static class is that it can implement interfaces and extend classes while the static class cannot (it can extend classes, but it does not inherit their instance members). If we consider a static class, it can only be a nested static class as top-level class cannot be a static class. Static means that it belongs to a class it is in and not to any instance. So, it cannot be a top-level class.*

*- Static class will have all its member as static only unlike Singleton.*

*- It can be lazily loaded whereas static will be initialized whenever it is first loaded.*

*- Singleton object stores in Heap but, static object stores in stack.*

*- We can clone the object of Singleton but, we cannot clone the static class object.*

*- Singleton can use the Object-Oriented feature of polymorphism but static class cannot.*

***Differences with Static class*:**

1. *A Static Class cannot be extended whereas a singleton class can be extended.*
2. *A Static Class can still have instances (unwanted instances) whereas a singleton class prevents it.*
3. *A Static Class cannot be initialized with a STATE (parameter), whereas a singleton class can be.*
4. *A Static class is loaded automatically by the CLR when the program or namespace containing the class is loaded.*
5. **C#- Singleton class with thread safe implementation.**

public sealed class Singleton

{

Singleton()

{

}

private static readonly object padlock = new object();

private static Singleton instance = null;

public static Singleton Instance

{

get

{

lock (padlock)

{

if (instance == null)

{

instance = new Singleton();

}

return instance;

}

}

}

}

1. **C#- Generics, delegates, generic delegates and collections.**

**Generics:** *Generic is the type which is specified later or at the time of object creation. When they are instantiated, the needed type is provided as a parameter. In short, a Generic allows us to write a class or method that can work with any data type.*

***Features of generics***

1. *It helps you to maximize code reuse type safety and performance.*
2. *The System.Collection.Generic namespace contains several new generic collection classes. You need to use the generic collection classes instead of the collection classes in the System.Collections namespace.*
3. *You can create user-defined generic interfaces, classes, methods, events and delegates.*
4. *A common use of Generics is to create collection classes.*

*Example:*

static void swap<t>(ref t p, ref t q) // generic method

{

t temp;

temp = p;

p = q;

q = temp;

}

**Delegates***: A delegate is a reference type variable that holds the reference to a method. The reference can be changed at runtime. Delegate are especially used for implementing events and call-back methods. All delegates are implicitly derived from System. Delegates class System.*

*We can declare a delegate type inside or outside the class because a delegate is a class type.*

1. **C#- Concepts of virtual override and new keyword in inheritance.**
2. **C#- Async and await.**
3. **Design level difference between interface and abstract classes.**
4. **Web API vs WCF.**
5. **SQL- normalization and their types.**
6. **SQL- de-normalization and its benefits.**
7. **SQL- practical implementations of joins, LEFT, INNER, RIGHT, SELF, CROSS and FULL.**
8. **SQL- temp table vs table variable vs common table expressions and their uses.**
9. **SQL-Views and Updatable views.**
10. **SQL- Triggers with types and uses.**
11. **Stored procedures and its practical uses and best practices.**
12. **SQL- Aggregate functions, rank, dense rank and row numbers.**
13. **SQL- Aggregation queries with group by.**
14. **SQL- Get product count using group by.**
15. **SQL- Union vs Union All.**
16. **SQL- Union vs Join.**
17. **SQL- Find intersection of two tables using full join.**
18. **ASPNET- Session vs View state.**
19. **DATAWAREHOUSE- OLTP vs OLAP Servers.**