1. What is Object-Oriented Programming (OOP)?

Object-Oriented Programming (OOP) is a programming paradigm that organizes code around objects, which are instances of classes. The four fundamental principles of OOP are Encapsulation,Inheritance,Polymorphism, and Abstraction.

2. What is a class and what is an object?

A class serves as a blueprint or template for creating objects in object-oriented programming (OOP). It encapsulates data (fields or properties) and behavior (methods or functions) that define the characteristics and actions of the objects instantiated from it.

An object is a instance of a class. It represents a specific entity that has the properties and behaviors defined by its class.

3. What is the difference between a class and an object?

Class

A blueprint/template for creating objects.

Defines properties (fields) and behaviors (methods).

Doesn’t occupy memory until an object is created.

Object

A real instance of a class.

Created using the new keyword.

Occupies memory and can be used to access class members.

4. What is encapsulation and why is it important?

Encapsulation is the process of hiding the internal state of an object and only exposing necessary functionality through public methods or properties.

-> Protects data from unauthorized access or modification.

5. What is abstraction and how is it implemented in C#?

Abstraction involves defining a blueprint for classes by specifying what methods must be implemented, without providing their implementation. In C#, abstraction is achieved using abstract classes or interfaces.

Abstract Class : An abstract class is a class that cannot be instantiated on its own. It may contain abstract methods (without implementation) and non-abstract methods (with implementation). Use

the abstract keyword. Child class uses override to provide implementation.

interfaces : An interface is a contract that contains only the signatures of methods, properties, events, or indexers. A class or struct that implements the interface must provide implementation for all its members.

6. What is inheritance and what are its types in C#?

Inheritance allows a derived class (child) to inherit fields, properties, and methods from a base class (parent). This promotes code reuse and establishes a hierarchical relationship between classes.

7. What is polymorphism and what are its types?

Polymorphism allows objects of different classes to be treated as instances of a common base class, with methods behaving differently based on the actual object type. In C#, polymorphism is achieved through method overriding (runtime polymorphism) or method overloading (compile-time polymorphism).

Method Overloading : Creating multiple methods with the same name but different parameters within the same class.

Method Overriding : Redefining a method in a derived (child) class that already exists in the base (parent) class with the same name, same parameters, and same return type. Requires virtual (in base class) and override (in child class) keywords.

8. What is the difference between compile-time and run-time polymorphism?

Compile-time polymorphism, also called static polymorphism, is resolved during compilation and is achieved through method overloading.

Run-time polymorphism, also called dynamic polymorphism, is resolved at execution time and is achieved using method overriding with virtual and override keywords.

9. What is an abstract class and when should it be used?

An abstract class is a class that cannot be instantiated on its own. It may contain abstract methods (without implementation) and non-abstract methods (with implementation). Use the abstract keyword. Child class uses override to provide implementation.

We use it when we want to provide some default behavior to child classes while also forcing them to implement certain methods.

10. What is an interface and when should it be used?

An interface is a contract that contains only the signatures of methods, properties, events, or indexers. A class or struct that implements the interface must provide implementation for all its members.

we use interfaces to keep controllers independent from business logic. For example, I create an IProductService interface that defines operations like GetAllProducts(). The ProductService implements it with actual logic, and the controller only depends on the interface. This makes the project loosely coupled.

11. What is a sealed class and when should it be used?

A sealed class is a class that cannot be inherited. A sealed method cannot be overridden further.

I would use it when I want to prevent changes to the implementation. especially for sensitive classes like logging.

12. What is a static class and static member?

A static class is a class that cannot be instantiated and contains only static members. We use it for utility or helper functions, like Math in .NET.

A static member belongs to the class itself, not an instance, and is shared across all objects.

13. What is the difference between const and readonly?

const is a compile-time constant, its value is fixed when you compile the code, and it must be assigned at declaration.

readonly is a runtime constant — it can be assigned either at declaration or inside the constructor, and once set, it cannot change.

14. What is a constructor and a destructor?

A constructor is a special method used to initialize an object when it is created. It has the same name as the class and no return type. For example, in Web API, I might use a constructor to inject services into a controller.

A destructor is the opposite — it is automatically called when the object is destroyed by the garbage collector. It’s used to clean up unmanaged resources.

15. What is the difference between a default constructor and a parameterized constructor?

A default constructor is a parameterless constructor that initializes an object with default values. If we don’t define any constructor, the compiler itself provides one.

A parameterized constructor accepts arguments and allows us to create an object EX : in a Web API project, I might use a parameterized constructor to inject dependencies.

16. What is the difference between this and base keywords?

The this Keyword Refers to the current instance of the class. it is mainly used to access current members or resolve naming conflicts.

The base keyword refers to the parent class and is used to call base class constructors or access base class methods and properties.

17. What is a nested class in C#?

A nested class is a class defined inside another class. It is useful when the inner class is strongly related to the outer class and is not meant to be used independently.

EX: in a User class, I could create a nested Address class if the address is only meaningful within the user context.

18. Difference between value types and reference types?

Value types (like int, bool, struct) store data directly in memory and are allocated on the stack.

Reference types (like class, object, string) store a reference to the memory location, and the actual object is on the heap.

Value types are copied by value, while reference types are copied by reference.

19. What are properties in C# and how do they relate to encapsulation?

Properties are members that provide controlled access to private fields using get and set accessors.

They support encapsulation by hiding internal implementation but exposing safe access points.

20. What is an indexer and how is it used?

An indexer allows objects to be indexed like arrays using square brackets. For example, in a custom collection class, I can define an indexer so that I can access items using myCollection[0].

21. Difference between shallow copy and deep copy?

A shallow copy copies only the top-level object, so references inside still point to the same memory. A deep copy creates a completely new object along with copies of all nested objects.

22. Difference between virtual, override, and new keywords?

virtual marks a method in the base class as overridable. override changes the implementation of that method in the derived class. new hides the base class method without overriding it, creating a new method in the derived class.

23. What is a delegate and how is it used?

A delegate is a type-safe function pointer that can hold references to methods with a specific signature. It is used for callbacks, passing methods as parameters, and implementing events.

24. What are lambda expressions and how are they related to delegates?

A lambda expression is a short way of writing anonymous methods using => syntax. They are often used to implement delegates and expression trees in a more concise form.

25. What is dependency injection and how does it relate to OOP?

Dependency Injection is a design pattern where dependencies are provided to a class instead of creating them inside. It supports the OOP principle of loose coupling by separating object creation from usage. In ASP.NET Core Web API, services are injected into controllers using constructors.

26. What is Delegates in C#?

Delegates in C# are type-safe references to methods. They let us pass methods as parameters, implement callbacks, and build flexible and decoupled code. Without delegates, methods would be tightly bound to specific implementations.

27. What is IEnumerable?

IEnumerable is an interface in C# that represents a collection we can loop through using foreach. It is mainly used for read-only access to data. while working with LINQ Queries.

28. What is IQueryable?

IQueryable is an interface used to build queries for remote data sources like databases. EF Core translates it into SQL so filtering happens at the database level, not in memory.

29. What is the difference between LINQ First(), FirstOrDefault(), Single(), and SingleOrDefault() in Entity Framework?

First() - Returns the first element that matches the condition.

FirstOrDefault() - Returns the first element if found.

Single() - Expects exactly one element in the result.

SingleOrDefault() - Expects zero or one element in the result.

FindAsync() (Entity Framework only) - Used to find an entity by primary key.

30. What is the difference between AddTransient, AddScoped, and AddSingleton lifetimes in ASP.NET Core Dependency Injection?

AddTransient - Lifetime: A new instance is created every time it’s requested.

AddScoped - Lifetime: One instance is created per HTTP request.

AddSingleton - Lifetime: A single instance is created once for the entire application lifetime.

31. What are ACID properties in SQL databases, and why are they important?

ACID stands for:

Atomicity

All or nothing rule.

A transaction must execute completely or not at all.

Example: In money transfer, if ■500 is deducted from Account A, it must also be credited to Account B. If one fails, the whole transaction rolls back.

Consistency

A transaction should always bring the database from one valid state to another.

Constraints, rules, and triggers must be preserved.

Example: If Account A has ■1000, you cannot withdraw ■2000 because it violates consistency rules.

Isolation

Multiple transactions running at the same time should not interfere with each other.

Each transaction should behave as if it’s the only one executing.

Example: Two people buying the last seat in a train — isolation ensures only one booking succeeds.

Durability

Once a transaction is committed, the changes are permanent, even if the system crashes.

Example: If money is transferred and the system crashes, the committed transfer still remains in the database after restart.

32. Dependency Injection ?

Dependency Injection means instead of creating dependencies inside a class, we inject them from outside through constructor.

This reduces tight coupling, improves testability, and makes code easier to maintain.

33. Async/Await vs. Multithreading

Async/await is about asynchronous programming (freeing a thread during I/O wait), while multithreading is about parallel execution of multiple tasks on different CPU cores.

Async is for I/O-bound work, multithreading is for CPU-bound work.

Async/Await :

async/await in C# is used to write asynchronous code that doesn’t block the thread.

It improves responsiveness and scalability, especially in I/O-bound operations like DB queries or API calls.”