Abstract class VS Interface

An abstract class is a base class that can contain both implemented and abstract methods, fields, and even constructors. It’s used when you want to share common logic among subclasses.  
An interface, on the other hand, only defines a contract (method and property signatures) with no implementation. It’s used to enforce a structure across classes, and a class can implement multiple interfaces.  
In short: use an abstract class for shared behavior, and an interface for defining a structure.

1. What are the 4 pillars of OOP?

**Encapsulation** → Wrapping data & methods in a class, hiding internal details.

 **Abstraction** → Hiding implementation details, showing only essential info.  
(Abstract classes, Interfaces)

 **Inheritance** → A class can inherit properties/methods from another.

 **Polymorphism** → Same method name behaves differently (overloading, overriding).

**2. Class vs Object**

* **Class** → Blueprint/template.
* **Object** → Instance of a class.

**3. Difference between Class and Abstract Class**

* **Class** → Can be instantiated, fully implemented methods.
* **Abstract Class** → Cannot be instantiated, may contain abstract + concrete methods.

**4. Difference between Abstract Class and Interface**

* **Abstract Class** → Can have state (fields), abstract & concrete methods, single inheritance.
* **Interface** → Only contract (methods/properties), no state, multiple inheritance possible.

**5. What is Encapsulation?**

* Binding data + methods inside a class and restricting direct access.
* Done with **access modifiers** (public, private, protected).

**6. What is Abstraction?**

* Showing only essential features, hiding internal implementation.
* Achieved via **abstract classes** or **interfaces**.

**7. What is Polymorphism?**

* **Compile-time (Overloading):** Same method name, different params.
* **Runtime (Overriding):** Subclass provides different implementation of a method.

**8. Difference between Composition and Inheritance**

* **Inheritance** → "is-a" relationship. (Dog is an Animal)
* **Composition** → "has-a" relationship. (Car has an Engine)

**9. Static vs Instance Members**

* **Static** → Belong to class, shared across all objects.
* **Instance** → Belong to object, unique per instance.

**Design Pattern**

A design pattern is a proven, reusable solution to a common software design problem. It helps developers write maintainable, scalable, and readable code. Examples include: Singleton for single instances, Factory for dynamic object creation, Decorator for adding behavior dynamically, and Observer for event-driven communication.

**Pro tip for Angular interviews:**

* **Singleton** → Angular services
* **Observer** → RxJS Observables
* **Decorator** → Angular decorators (@Component, @Directive)