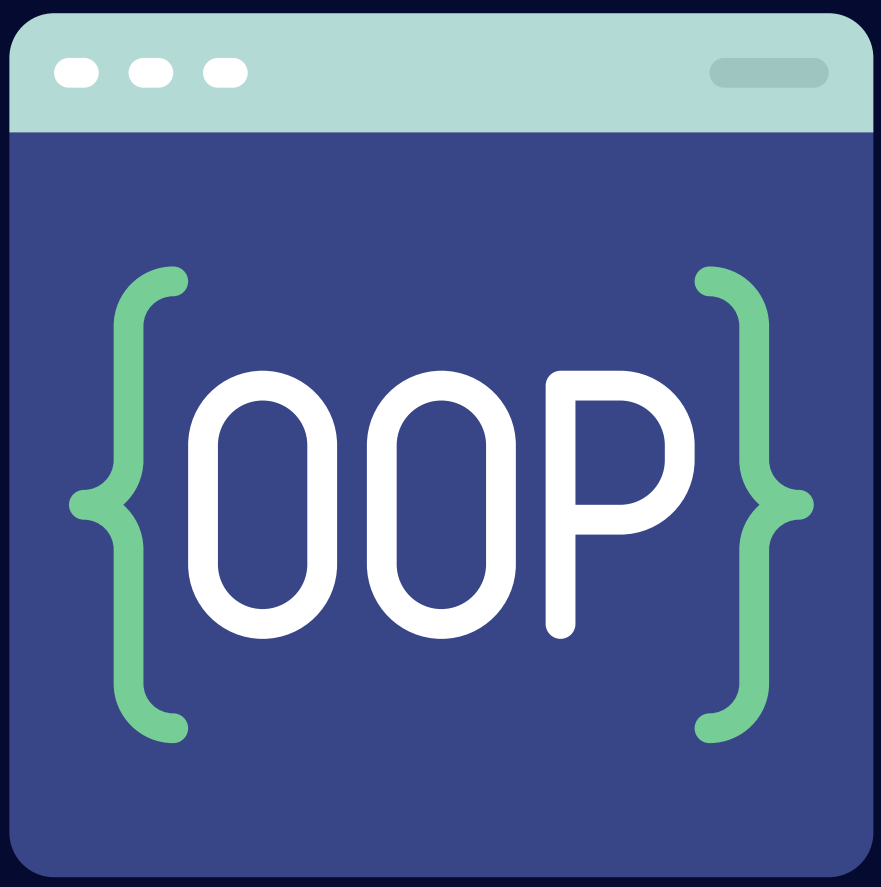
ALL COMMONLY ASKED DIFFRENCES IN OBJECT ORIENTED PROGRAMMING



Class vs Objects

Class	Object
A blueprint or template for creating objects.	An instance of a class. It represents a real entity.
No memory is allocated until objects are created.	Takes up memory when instantiated.
Abstract concept. Does not exist at runtime.	Exists at runtime.
Defines properties (attributes) and behaviors (methods).	Holds specific data and behaves as per the class definition.
Car (a general concept of cars)	myCar = Car() (a specific car object, say a Honda).
Declared once and can be reused multiple times.	Can be created as many times as needed.
Not initialized with values, only defines the structure.	Initialized with specific values for its properties.
Stored as a definition in the code.	Stored in memory with data.
Used to define the type of objects.	Used to interact with real data and invoke methods.

Curious_.Programmer

Encapsulation Vs Abstraction

Encapsulation	Abstraction
Hiding internal data and allowing access through methods.	Hiding complex details and showing only important parts.
To protect data from unauthorized access.	To simplify usage by showing only what's needed.
By using private variables and public methods (getters/setters).	By using abstract classes or interfaces.
Controlling access to data.	Simplifying how you interact with something.
A remote control hiding internal wiring, only showing buttons.	A car's interface (steering, pedals) hides engine details.
Provides a way to safeguard an object's state by restricting direct access.	Provides a way to create a high-level view of an object without showing how it works inside.
Controls data visibility using access modifiers like private, protected, public.	Focuses on showing only what's necessary for the task, leaving out unnecessary details.
Ensures the integrity of the object's state by allowing controlled modification.	Allows working with complex systems without needing to know how they work internally.
A class with private variables and public methods to set/get values.	A Shape interface that only has a draw() method, hiding how different shapes are drawn.

Curious_.Programmer

Overloading Vs Overriding

Method Overloading	Method Overriding
Defining multiple methods with the same name but different parameters in the same class.	Redefining a method in a subclass that already exists in the parent class.
To perform similar tasks with different types or numbers of inputs.	To provide a specific implementation of a method in the subclass.
In the same class.	Between parent and child classes (inheritance).
Must have different parameter lists (number, type, or order).	Must have the same method signature (name and parameters).
Can have different return types.	Must have the same return type as the parent method.
Access modifier can be different.	Access modifier must be the same or more permissive than in the parent class.
Resolved during compile-time.	Resolved during runtime (dynamic binding).
add(int a, int b) and add(double a, double b) in the same class.	A draw() method in Circle class overriding draw() in Shape class.

Curious_.Programmer

Interface Vs Abstract Class

Interface	Abstract Class
A collection of abstract methods that a class must implement.	A class that can have both abstract and concrete (implemented) methods.
To define a contract for what a class must do, without specifying how.	To provide a common base class with some shared functionality and force subclasses to implement certain methods.
Only abstract methods (no implementation allowed).	Can have both abstract methods (no implementation) and concrete methods (with implementation).
A class can implement multiple interfaces.	A class can extend only one abstract class (single inheritance).
Use when you want to specify what methods a class must implement.	Use when you want to provide some shared code along with abstract methods.
Methods are public and cannot have any other access modifier.	Can have any access modifier (public, private, protected).
Cannot have instance variables (only constants).	Can have instance variables and constants.
Cannot have constructors.	Can have constructors.

Curious_.Programmer

Inheritance Vs Composition

Inheritance	Composition
A relationship where a class (subclass) inherits properties and behavior from another class (superclass).	A relationship where a class contains objects of other classes to reuse functionality.
"Is-a" relationship (e.g., Dog is a type of Animal).	"Has-a" relationship (e.g., Car has an Engine).
Involves parent-child (superclass-subclass) hierarchy.	No hierarchy, focuses on including other classes as fields.
Reuses code from the parent class by extending it.	Reuses code by including objects of other classes.
Less flexible, as the subclass is tightly coupled with the superclass.	More flexible, as classes are loosely coupled.
Use when there's a clear hierarchical relationship.	Use when you want to include functionality from multiple classes without inheritance.
Changes in the superclass can affect all subclasses.	Changes in a contained class won't affect the parent class as directly.
class Dog extends Animal	class Car { Engine engine; }