OOPs & Design Patterns

Section A – Core OOPs Concepts

#	Question	Answer
1	What is OOPs and why is it used?	Programming model based on objects; improves modularity, reusability & maintenance.
2	Encapsulation example in Java?	Private fields + public getters/setters to protect data.
3	Abstraction vs Encapsulation	Abstraction = hide details; Encapsulation = bundle data + methods.
4	Polymorphism types	Compile-time (overloading), Runtime (overriding).
5	How can inheritance cause tight coupling?	Child depends on parent implementation → use interfaces / composition.
6	IS-A vs HAS-A	IS-A = Inheritance; HAS-A = Aggregation / Composition.
7	Constructor overloading vs overriding	Overloading ✅ (same class); Overriding ❌ (not allowed).
8	Purpose of super	Access parent members / invoke parent constructor.
9	Rule for method overriding	Same signature, return type covariant, visibility ≥ parent.
10	Abstraction example	Interface PaymentGateway { processPayment(); }

Section B – Implementation & Design Thinking

#	Question	Answer
11	Four pillars of OOPs	Encapsulation, Abstraction, Inheritance, Polymorphism.
12	Access modifiers	public, protected, (default), private – control visibility.
13	Abstract class vs Interface	Abstract class → partial implementation; Interface → contract only.
14	When prefer abstract class	When common base logic shared among subclasses.
15	Multiple inheritance in Java	Classes ❌ (ambiguity), Interfaces ✅ (default methods resolve).
16	Exception handling as OOP concept	Adds abstraction layer for error control.
17	Composition over Inheritance	Use object references instead of deep hierarchy.
18	Loose coupling methods	Interfaces / Dependency Injection / Events.
19	Use of this and super	this() calls own constructor; super() calls parent.
20	OOP → testing ease	Independent modules enable unit testing.

Section C – SOLID Principles

#	Question	Answer
21	Expand SOLID	Single Responsibility, Open/Closed, Liskov Substitution, Interface Segregation, Dependency Inversion.
22	SRP example	Each class has one responsibility (e.g., InvoicePrinter separate from InvoiceCalculator).
23	OCP example	Extend PaymentType via new class without modifying existing factory.
24	LSP violation example	Square extends Rectangle but changes behavior → breaks LSP.
25	Interface Segregation example	Split LargePrinter into BasicPrinter + Scanner interfaces.
26	Dependency Inversion definition	Depend on abstractions, not concrete classes.
27	Patterns supporting SOLID	Factory → OCP; Singleton → SRP; Observer → DIP.

Section D – Design Patterns

#	Question	Answer
28	What is a design pattern?	Reusable template for common design problem.
29	Singleton Pattern	One instance only – e.g., DB Connection.
30	Factory Pattern	Centralized object creation logic; supports OCP.
31	Factory vs Abstract Factory	Factory = one product family; Abstract Factory = multiple families.
32	Observer Pattern	Publisher-Subscriber notification system.
33	MVC Pattern	Model = data, View = UI, Controller = logic – used in Spring / React.
34	DAO Pattern	Encapsulates DB access layer – promotes separation of concerns.
35	Restrict object creation pattern	Singleton.
36	Event notification pattern	Observer.
37	Factory supports OCP how?	Add new subclass without editing client code.

Section E – Code & Output Prediction

#	Question	Answer
38	Method overriding output	Runtime method of child executes (dynamic dispatch).
39	Constructor chaining order	Parent → Child (super() then this()).
40	Missing abstract method	Compiler error → must implement all abstract methods.
41	Factory trace	Returns object matching input type (e.g., CardPayment if "Card").
42	Thread-safe Singleton	Use synchronized method or double-checked locking.

Section F – Project & SME Scenarios

No	Question	Answer
43	Which OOP concept used in your project?	Encapsulation + Abstraction in service layers.
44	Refactor to follow SOLID	Split large class → multiple responsibilities.
45	MVC in multi-module project	Separate Controller per module, shared Model layer.
46	Code reusability for data access	Use DAO + Repository pattern.
47	Show design thinking in interview	Draw class diagram, explain interaction flow.