| Pattern | Motivation | Components | Problem | Solution | Benefits | Applications | Example |
|---|---|---|---|---|---|---|---|
| **Factory Method** | Let subclasses decide which concrete class to instantiate. | Creator (factory method), ConcreteCreators, Products, ConcreteProducts | Client is tightly coupled to concrete classes. | Define a virtual create() in base creator; override in subclasses. | Decouples client from implementations; open for extension. | GUI toolkits, loggers, parsers | VehicleFactory → CarFactory calling create() to get a Car instance. |
| **Abstract Factory** | Create families of related objects without specifying classes. | AbstractFactory, ConcreteFactories, AbstractProducts, ConcreteProducts | Creating related objects often leads to incompatible combinations. | Expose an interface for creating related objects; implement per family. | Ensures compatibility; isolates concrete factories. | UI themes (Windows vs. Mac), cross-DB drivers | UIFactory → WinFactory yields WinButton + WinCheckbox. |
| **Singleton** | Ensure a class has only one instance, provide global access. | Singleton class with private constructor + static instance access | Global shared object must be unique; race-conditions in lazy creation. | Hide constructor; expose a static getInstance() (thread-safe). | Controlled access, reduced namespace pollution. | Configuration managers, logging, thread pools | Config.getInstance().load("app.cfg") |
| **Builder** | Construct complex objects step by step. | Builder interface, ConcreteBuilders, Director, Product | Telescoping constructors or complex assembly logic scattered in client. | Separate construction and representation via builder methods. | More readable construction; supports variations and immutability. | HTML/XML builders, object serializers | new CarBuilder().withWheels(4).withColor("red").build() |
| **Prototype** | Clone objects without coupling to their concrete classes. | Prototype (cloneable interface), ConcretePrototypes, Client | Creating from scratch or via constructors is costly or complex. | Copy an existing prototypical instance via clone(). | Faster object creation; runtime configuration. | Graphic editors (copy/paste), object pools | Shape prototype = registry.get("circle"); Shape clone = prototype.clone(); |
| **Adapter** | Let incompatible interfaces work together. | Target interface, Adapter, Adaptee | You have an existing class whose interface doesn't match what you need. | Wrap the adaptee with an adapter implementing the target interface. | Reuses legacy code; decouples clients from adaptee. | Third-party libraries, legacy APIs | class SocketAdapter implements USB { Socket s; … } |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **Bridge** | Decouple abstraction from implementation so they vary independently. | Abstraction, RefinedAbstraction, Implementor, ConcreteImplementors | Changes in abstraction or implementation ripple through the hierarchy. | Put implementation behind an interface and hold it in the abstraction. | Independent extensibility of both sides. | Device drivers, remote controls, GUI renderers | RemoteControl → TVImpl vs. RemoteControl → RadioImpl |
| **Composite** | Treat individual objects and compositions uniformly. | Component, Leaf, Composite | Need to treat single objects and groups of objects the same way. | Define a tree structure where both leaf and composite implement the same interface. | Simplifies client code; easy to add new components. | File systems, UI hierarchies, graphic scenes | Folder contains Files and Folders; client calls print()` on each. |
| **Decorator** | Add responsibilities to objects dynamically. | Component, ConcreteComponent, Decorator, ConcreteDecorators | Subclass explosion when combining features; need runtime extension. | Wrap objects with decorator objects that implement the same interface. | More flexible than static inheritance; can combine features. | I/O streams (Java), GUI widgets | new BorderDecorator(new ScrollDecorator(textView)).draw() |
| **Facade** | Provide a unified interface to a set of interfaces. | Facade, Subsystem classes | Clients must interact with a complex subsystem API. | Define a high-level façade class that delegates to subsystem classes. | Simplifies use; decouples client from subsystem. | Complicated libraries, middleware | VideoConversionFacade.convert("video.avi", "mp4") |
| **Flyweight** | Use sharing to support many fine-grained objects efficiently. | Flyweight, FlyweightFactory, UnsharedFlyweight | Huge numbers of similar objects consume too much memory. | Extract intrinsic state, share it via a factory; pass extrinsic state externally. | Significant memory savings; centralized state. | Text editors (character formatting), particle systems | char c = flyweightFactory.get('a'); c.draw(x, y, fontInfo); |
| **Proxy** | Provide a placeholder to control access to another object. | Subject, RealSubject, Proxy | Need to add behavior (lazy load, logging, access control) before/after real object. | Proxy implements subject interface and holds a reference to real subject. | Controls access, adds layers without changing the real subject. | Virtual proxies, access control, caching | Image img = new ImageProxy("large.jpg"); img.display(); |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **Chain of Responsibility** | Decouple sender and receiver by giving multiple objects a chance. | Handler, ConcreteHandlers | Sender doesn't know which receiver will handle the request. | Link handlers in a chain; pass the request along until one handles it. | Dynamic handler assignment; reduced coupling. | Event handling, logging frameworks | spamFilter → virusFilter → inboxHandler |
| **Command** | Encapsulate a request as an object. | Command, ConcreteCommands, Invoker, Receiver | Need to parameterize objects with operations, support undo/redo, or queue requests. | Create command objects with execute() and optional undo(). | Queuing, logging, undo/redo, macros. | GUI buttons, task schedulers | Command cmd = new PrintCommand(doc); button.setCommand (cmd); |
| **Interpreter** | Evaluate sentences in a language. | AbstractExpression, TerminalExpression, NonTerminalExpression, Context | Building a parser or evaluator for a simple language. | Define a class for each grammar rule; interpret by walking the parse tree. | Easy to extend grammar; clear separation of grammar. | SQL parsing, mathematical expression evaluators | Expr parse = new Add(new Number(1), new Number(2)); parse.interpret(ctx); |
| **Iterator** | Provide a way to access elements of an aggregate sequentially without exposing its representation. | Iterator, ConcreteIterator, Aggregate, ConcreteAggregate | Clients need to traverse different collections in a uniform way. | Give each collection an iterator; clients use hasNext()/next(). | Simplifies traversal; supports multiple simultaneous traversals. | Java/C# collections, XML DOM traversal | for (Iterator it = list.iterator(); it.hasNext(); ) { … } |
| **Mediator** | Define an object that encapsulates how a set of objects interact. | Mediator, ConcreteMediator, Colleagues | Tight coupling and complex communication between multiple classes. | Centralize communication logic in a mediator; colleagues talk only to mediator. | Simplifies object protocols; reduces coupling. | Dialog boxes, air traffic control | ChatRoomMediator relays messages between User objects. |
| **Memento** | Capture and externalize an object's internal state so it can be restored later. | Memento, Originator, Caretaker | Need to rollback an object to a previous state without exposing its internals. | Originator creates mementos; caretaker stores them; originator restores. | Preserves encapsulation; easy undo/rollback. | Text editors (undo), transaction snapshots | caretaker.addMemento(originator.saveState()); originator.restore(caretaker.getMemento()); |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **Observer** | Define a one-to-many dependency so that when one object changes state, all dependents are notified. | Subject, Observer, ConcreteSubject, ConcreteObservers | Many objects need to stay in sync with another's state change. | Subject holds a list of observers and notifies them on state change. | Loose coupling; dynamic subscription. | MVC frameworks, event listeners | button.addListener(clickListener); button.click(); |
| **State** | Allow an object to alter its behavior when its internal state changes. | Context, State interface, ConcreteStates | Large conditional trees based on state; behavior scattered in context. | Extract state-specific behavior into separate state classes. | Simplifies state transitions; isolates behavior. | Protocol parsers, UI workflows | context.setState(new LockedState()); context.handleEvent(); |
| **Strategy** | Define a family of algorithms, encapsulate each one, and make them interchangeable. | Strategy interface, ConcreteStrategies, Context | Conditional logic to switch between algorithms at runtime. | Encapsulate each algorithm in its own class; context holds a reference. | Simplifies swapping algorithms; open for extension. | Sorting, compression, payment methods | context.setStrategy(new QuickSort()); context.sort(data); |
| **Template Method** | Define the skeleton of an algorithm, deferring steps to subclasses. | AbstractClass (with template method), ConcreteClasses | Code duplication among similar algorithms with slight variations. | Put invariant steps in a base class; override hook methods in subclasses. | Promotes reuse; enforces algorithm structure. | Frameworks (setup-execute-teardown), data processing pipelines | abstract class Game { play() { init(); start(); end(); } } |
| **Visitor** | Represent an operation to be performed on elements of an object structure. | Visitor, ConcreteVisitors, Element, ConcreteElements, ObjectStructure | Adding operations to object structures without modifying their classes. | Visitor interface with visit methods; elements accept visitors. | Easy to add new operations; separates algorithm from object structure. | Compiler AST traversals, serialization | element.accept(new PrintVisitor()); |