#	Concept	Description	Analogy / Purpose
1	Encapsulation	Bundling data and methods within a class.	Like a pill capsule that hides the ingredients inside.
2	Inheritance	A class inherits properties/methods from a parent class.	Like a child inheriting traits from its parents.
3	Polymorphism	Same interface can represent different underlying forms.	Like the word “run” having different meanings.
4	Abstraction	Hiding complex details while exposing essential features.	Driving a car without knowing its internal mechanics.
5	Class	Blueprint for creating objects.	Architectural plans used to build houses.
6	Object	An instance of a class.	A specific house built from an architectural plan.
7	Constructor	Special method to initialize objects.	Laying the foundation when constructing a building.
8	Destructor	Method called when an object is destroyed to free resources.	Demolishing a building when it’s no longer needed.
9	Method Overloading	Same method name with different parameters.	Using the same tool in different ways depending on the task.
10	Method Overriding	Redefining a parent class’s method in a child class.	A tailor modifying a standard suit for a specific client.
11	Static Members	Belong to the class rather than any instance.	A company logo that represents the entire organization.
12	Instance Members	Belong specifically to an individual object/instance.	Personal belongings unique to each house.
13	Access Modifiers	Controls the visibility of class members (public, private, protected).	Using locks, keys, and passcodes to secure different rooms.
14	Public Access	Members accessible from any part of the program.	An open park accessible to everyone.
15	Private Access	Members accessible only within the class itself.	A personal diary kept under lock and key.
16	Protected Access	Members accessible within the class and its subclasses.	A family recipe shared only with relatives.
17	Data Members	Variables defined within a class to hold object state.	The properties of a house such as size or color.
18	Member Functions	Methods defined within a class that operate on data members.	Tools in a toolbox that perform specific tasks.
19	Composition	Building complex types by combining objects.	A car built from different parts (engine, wheels, etc.).
20	Aggregation	A form of association representing a whole-part relationship.	A team made up of individual players.
21	Association	A broad term for relationships between classes.	Friends who regularly interact.
22	Interface	Specifies a set of methods a class must implement without code.	A contract outlining responsibilities without dictating how to complete them.

#	Concept	Description	Analogy / Purpose
23	Abstract Class	A class that cannot be instantiated and is designed to be subclassed.	A template for blueprints that must be customized.
24	Virtual Functions	Functions meant to be overridden in derived classes.	A flexible policy that adapts based on who implements it.
25	Operator Overloading	Allowing built-in operators to work with user-defined types.	Defining how “+” works when adding complex numbers.
26	Constructor Overloading	Multiple constructors in a class with different parameters.	Different entry doors to the same building for various purposes.
27	Copy Constructor	Creates a new object as a copy of an existing object.	Duplicating a key to open the same door.
28	This Pointer	Refers to the current object instance inside class methods.	A mirror reflecting the current self.
29	Singleton Pattern	Ensures a class has only one instance across the program.	A nation’s president with a single office.
30	Factory Pattern	A method for creating objects without specifying the exact class.	A car dealership that offers various models without customers knowing the inner workings.
31	Adapter Pattern	Allows incompatible interfaces to work together.	A travel adapter converting plug types between countries.
32	Bridge Pattern	Separates an abstraction from its implementation, allowing them to vary independently.	A bridge connecting two separate islands, each with its own culture.
33	Composite Pattern	Composes objects into tree structures to represent whole-part hierarchies.	An organization chart showing a company’s structure.
34	Decorator Pattern	Dynamically adds behavior to an object without changing its interface.	Adding layers of paint or accessories to a plain car.
35	Facade Pattern	Provides a simplified interface to a complex system.	A universal remote that controls several devices.
36	Flyweight Pattern	Reduces memory usage by sharing common parts of objects.	Reusing the same architectural blueprint for multiple identical houses.
37	Proxy Pattern	Provides a surrogate or placeholder to control access to an object.	A security guard managing access to a restricted area.
38	Command Pattern	Encapsulates a request as an object, allowing parameterization of clients.	A remote control that holds commands like ‘play’ or ‘pause’.
39	Interpreter Pattern	Defines a representation for a grammar and an interpreter to process sentences in that grammar.	Translating a language using a dictionary.
40	Iterator Pattern	Provides a way to access elements of a collection sequentially.	A bookmark that helps you navigate through a book.
41	Mediator Pattern	Encapsulates how a set of objects interact, promoting loose coupling.	A traffic controller coordinating aircraft movements.
42	Memento Pattern	Captures and externalizes an	A save state in a video game.

#	Concept	Description	Analogy / Purpose
		object’s internal state without violating encapsulation.	
43	Observer Pattern	Allows objects to subscribe and receive updates when changes occur.	A newsletter subscription where readers get updates.
44	State Pattern	Allows an object to alter its behavior when its internal state changes.	A traffic light that changes its signals based on time of day.
45	Strategy Pattern	Encapsulates different algorithms and makes them interchangeable.	Choosing different routes to a destination based on traffic.
46	Template Pattern	Defines the skeleton of an algorithm, deferring steps to subclasses.	A baking recipe that allows for ingredient variations.
47	Visitor Pattern	Separates an algorithm from an object structure by moving operational logic into a separate class.	A museum guide who interprets different exhibits.
48	Dependency Injection	Supplying a class with its dependencies rather than having it instantiate them.	Installing batteries into a remote instead of building them inside.
49	Loose Coupling	Minimizes dependencies between components to allow changes independently.	Modular furniture that can be rearranged with ease.
50	High Cohesion	Keeping related functions and data together in a module/class.	A well-organized toolbox where similar tools are grouped.
51	DRY Principle	“Don't Repeat Yourself” – avoid duplication of code.	Using one stamp for multiple envelopes instead of multiple stamps.
52	KISS Principle	“Keep It Simple, Stupid” – simplicity improves readability and maintainability.	A straightforward recipe versus a convoluted one.
53	YAGNI Principle	“You Aren't Gonna Need It” – build only what is necessary.	Packing only essentials for a trip rather than excessive gear.
54	SOLID Principles	Set of five design principles for better software development.	Building a house with a solid foundation and structure.
55	Single Responsibility Principle	A class should have only one reason to change.	A dedicated chef specializing in one cuisine.
56	Open/Closed Principle	Classes should be open for extension but closed for modification.	Upgrading a car’s software without changing its hardware.
57	Liskov Substitution Principle	Objects of a superclass should be replaceable with objects of a subclass without affecting correctness.	Replacing a standard battery with a compatible rechargeable one.
58	Interface Segregation Principle	Many client-specific interfaces are better than one general-purpose interface.	Offering different connectors for different devices.

#	Concept	Description	Analogy / Purpose
59	Dependency Inversion Principle	Depend on abstractions rather than concrete implementations.	Plugging appliances into standard outlets rather than custom ones.
60	Reusability	Writing code that can be reused in other parts of the application.	Using a universal remote that works with various devices.
61	Maintainability	Code should be easy to update and modify over time.	A well-labeled filing system that is easy to navigate.
62	Extensibility	The design allows for future growth without major rework.	A modular home that can be expanded room by room.
63	Robustness	Creating systems that handle errors and unexpected situations gracefully.	A shock-absorbing car suspension system that handles bumps smoothly.
64	Error Handling	Managing runtime errors with try-catch blocks or similar constructs.	Using airbags in a car to protect in case of an accident.
65	Exception Classes	Specialized classes to represent various error conditions.	Different colored warning lights that signal specific issues.
66	Custom Exceptions	Allowing developers to create their own exception types.	Customizing alerts to handle unique failure modes.
67	Garbage Collection	Automatic memory management to free unused objects.	A cleaning service that takes away discarded items.
68	Dynamic Binding	Method calls are resolved at runtime rather than compile time.	Switching radio stations on a car’s infotainment system as needed.
69	Late Binding	Synonymous with dynamic binding—deciding the method implementation at runtime.	Adjusting a GPS route on the fly during a journey.
70	Early Binding	Method calls bound at compile time for efficiency.	Pre-booking a fixed dinner menu rather than ordering on the spot.
71	Message Passing	Objects communicate by sending messages to invoke behavior.	Exchanging text messages to coordinate plans.
72	Event Handling	Mechanism to respond to user actions or system triggers.	An alarm that sounds when triggered by motion.
73	Observer Mechanism	Objects receive notifications about state changes in other objects.	A weather app pushing alerts when conditions change.
74	Immutable Objects	Objects whose state cannot be changed once created.	A sealed letter that cannot be altered once mailed.
75	Mutable Objects	Objects that allow their state to change over time.	A whiteboard that you can rewrite and erase information on.
76	Method Chaining	Allows multiple methods to be called in a single statement.	Building a stack of Lego blocks one on top of another in sequence.
77	Fluent Interface	Designing method calls to be chained in a readable manner.	Conversational language used to instruct a virtual assistant.
78	Encapsulation Boundary	The demarcation where an object’s internals are hidden.	A bank vault that limits access to its contents.
79	Modularization	Dividing software into separate, interchangeable modules.	Using separate building blocks to construct a model.
80	Separation of	Dividing a program into distinct	A restaurant where chefs, waiters,

#	Concept	Description	Analogy / Purpose
	Concerns	features that overlap as little as possible.	and managers each focus on their roles.
81	Component-Based Architecture	Organizing code into reusable, self-contained components.	Electronic circuits built from standard, interchangeable parts.
82	Code Refactoring	Restructuring existing code without changing its behavior to improve readability.	Reorganizing a messy room without buying new furniture.
83	Testability	Designing software so that it can be easily tested.	Assembling a toy with easy-to-check components.
84	Mock Objects	Simulated objects that mimic the behavior of real objects for testing purposes.	Using a stunt dummy for practice before a real performance.
85	Unit Testing	Testing small parts (units) of code independently.	Checking each ingredient separately when cooking a complex recipe.
86	Integration Testing	Testing the interaction between integrated modules.	Ensuring that individual train cars connect properly to form a complete train.
87	Code Comments	Inline explanations within code to clarify intent.	Margin notes in a textbook explaining key ideas.
88	Documentation	Detailed explanations of software design and usage.	A user manual guiding you through assembling furniture.
89	Design Patterns	Reusable solutions to common design problems in software.	Standardized blueprints for constructing different building types.
90	Code Smells	Symptoms in the codebase that may indicate deeper problems.	Odd sounds in machinery signaling that maintenance is needed.
91	Refactoring Patterns	Standardized approaches to restructuring code.	Renovation strategies for updating a classic building.
92	Immutable Pattern	Practices to enforce immutability in objects.	Setting concrete in a mold so its shape cannot change later.
93	Wrapper Class	A class that encapsulates primitive data types to provide object features.	A gift box that wraps a simple item to add presentation and protection.
94	Extension Methods	Methods that add functionality to an existing type without modifying its definition.	Adding accessories to a standard smartphone to enhance its functions.
95	Partial Classes	Splitting a class definition across multiple files.	Collaborating on chapters of the same book that are later compiled.
96	Multi-threading in OOP	Designing objects to run concurrently in multiple threads.	Multiple cooks preparing different dishes simultaneously in a busy kitchen.
97	Covariance and Contravariance	Type system rules that govern type substitutability in method return values and parameters.	Adapting interchangeable parts in mechanical systems.

#	Concept	Description	Analogy / Purpose
98	Reflection	The ability of a program to inspect and modify its structure at runtime.	A mirror that not only reflects an image but can also annotate it.
99	Self-Documentation	Designing code to be clear and readable so that it explains itself.	Signs on a road that clearly explain directions without needing extra instructions.
100	Design by Contract	Defining formal, precise, and verifiable interface specifications for software components.	Establishing a rental agreement outlining the responsibilities of both the landlord and tenant.

Each row represents a concept or related design pattern often encountered in object-oriented programming. These concepts together help build robust, scalable, and maintainable systems. Feel free to ask if you'd like further details or code examples for any of these items!