Creational Patterns

1. **Singleton**: Ensures a class has only one instance and provides a global access point.
2. **Factory Method**: Creates objects without specifying the exact class of object that will be created.
3. **Abstract Factory**: Provides an interface for creating families of related or dependent objects without specifying their concrete classes.
4. **Builder**: Separates the construction of a complex object from its representation.
5. **Prototype**: Creates new objects by copying an existing object, known as the prototype.

Structural Patterns

6. **Adapter (Wrapper)**: Converts the interface of a class into another interface the client expects.
7. **Bridge**: Separates an abstraction from its implementation, allowing them to vary independently.
8. **Composite**: Composes objects into tree structures to represent part-whole hierarchies.
9. **Decorator**: Adds new functionality to an object dynamically.
10. **Facade**: Provides a simplified interface to a more complex system.
11. **Flyweight**: Reduces memory usage by sharing as much data as possible with other similar objects.
12. **Proxy**: Provides a surrogate or placeholder for another object to control access to it.

Behavioral Patterns

13. **Chain of Responsibility**: Passes a request along a chain of handlers.
14. **Command**: Encapsulates a request as an object, allowing parameterization of clients with queues, requests, and operations.
15. **Interpreter**: Implements a specialized language by evaluating sentences in that language.
16. **Iterator**: Provides a way to access elements of a collection sequentially without exposing the underlying structure.
17. **Mediator**: Facilitates communication between different objects without them being aware of each other.
18. **Memento**: Captures and restores an object’s internal state without violating encapsulation.
19. **Observer (Publish-Subscribe)**: Defines a one-to-many dependency between objects where a change in one object triggers updates to all dependents.
20. **State**: Allows an object to change its behavior when its internal state changes.
21. **Strategy**: Encapsulates algorithms inside a class and makes them interchangeable.
22. **Template Method**: Defines the skeleton of an algorithm, deferring some steps to subclasses.
23. **Visitor**: Allows adding new operations to objects without changing their structure.

Concurrency Patterns

24. **Active Object**: Decouples method execution from method invocation to enable asynchronous processing.
25. **Future/Promise**: Represents a result of an operation that may not have been completed yet.
26. **Thread Pool**: Manages a collection of reusable threads to perform tasks.
27. **Producer-Consumer**: Separates the tasks of producing data and consuming data to improve concurrency.
28. **Monitor Object**: Ensures that only one thread can execute a critical section of code at a time.

These patterns help structure and design solutions in object-oriented and functional programming, improving maintainability, scalability, and readability. Let me know if you'd like more details on any specific pattern!