Nhóm Hành vi (Behavioral Patterns):

Command Pattern: Quản lý và thực thi các yêu cầu trong một hệ thống theo cách độc lập với người gửi yêu cầu.
State Pattern: Cho phép một đối tượng thay đổi hành vi của mình khi trạng thái nội bộ thay đổi.
Observer Pattern: Xác định một phụ thuộc một-nhiều giữa các đối tượng, khi một đối tượng thay đổi trạng thái, các đối tượng phụ thuộc sẽ được thông báo và cập nhật tự động.
Chain of Responsibility Pattern: Xây dựng một chuỗi các đối tượng xử lý yêu cầu, mỗi đối tượng trong chuỗi có thể xử lý yêu cầu hoặc chuyển tiếp yêu cầu đến đối tượng tiếp theo trong chuỗi.
Mediator Pattern: Định nghĩa một đối tượng trung gian để điều phối giao tiếp giữa các đối tượng khác nhau.
Interpreter Pattern: Định nghĩa cách diễn dịch một ngôn ngữ hoặc biểu thức.
Iterator Pattern: Cung cấp một cách duyệt qua các phần tử của một tập hợp mà không tiết lộ chi tiết triển khai của tập hợp đó.
Memento Pattern: Cho phép lưu trữ và khôi phục trạng thái của một đối tượng mà không tiết lộ chi tiết triển khai.
Strategy Pattern: Định nghĩa một họ các thuật toán, đóng gói chúng và làm cho chúng có thể thay đổi linh hoạt.
Template Method Pattern: Định nghĩa một bản mẫu hoặc cấu trúc cho một thuật toán và cho phép các lớp con triển khai các bước cụ thể của thuật toán đó.

Nhóm Cấu trúc (Structural Patterns):

11. Adapter Pattern: Chuyển đổi giao diện của một lớp thành giao diện khác mà các client mong muốn.
12. Bridge Pattern: Tách biệt một phần của một hệ thống khỏi các thành phần khác và cho phép chúng thay đổi độc lập.
13. Composite Pattern: Tạo ra một cấu trúc phân cấp của các đối tượng và xử lý chúng theo cách đồng nhất.
14. Decorator Pattern: Cho phép thêm chức năng mới cho đối tượng mà không ảnh hưởng đến cấu trúc gốc.
15. Facade Pattern: Cung cấp một giao diện đơn giản cho một hệ thống phức tạp.
16. Flyweight Pattern: Tối ưu hóa việc sử dụng bộ nhớ bằng cách chia sẻ các đối tượng có chung thuộc tính giữa nhiều đối tượng.
17. Proxy Pattern: Định nghĩa một đối tượng trung gian để kiểm soát quyền truy cập vào đối tượng thực.

Nhóm Khởi tạo (Creational Patterns):

18. Prototype Pattern: Tạo ra các đối tượng mới bằng cách sao chép các đối tượng đã tồn tại.
19. Singleton Pattern: Đảm bảo rằng một lớp chỉ có một đối tượng duy nhất và cung cấp điểm truy cập toàn cục đến đối tượng đó.
20. Abstract Factory Pattern: Cung cấp một giao diện để tạo ra các đối tượng liên quan hoặc phụ thuộc mà không cần chỉ định cụ thể các lớp cụ thể.
21. Builder Pattern: Xây dựng một đối tượng phức tạp bằng cách chia quá trình xây dựng thành các bước nhỏ và độc lập.

22. Factory Method Pattern: Định nghĩa một phương thức tạo đối tượng trong một lớp gốc, cho phép các lớp con quyết định loại đối tượng cụ thể sẽ được tạo ra.
23. Subclass Sandbox Pattern: Giới hạn việc mở rộng lớp con bằng cách chỉ cho phép ghi đè các phương thức dự định sẵn.
24. Type Object Pattern: Định nghĩa một lớp đại diện cho một loại đối tượng và cho phép các đối tượng khác tham chiếu đến loại đối tượng đó.

Nhóm Đặc biệt (Special Patterns):

25. Component Pattern: Định nghĩa một cách để tổ chức các đối tượng thành một cấu trúc cây, trong đó mọi đối tượng chung một giao diện chung.
26. Event Queue Pattern: Quản lý và phân phối các sự kiện trong hệ thống theo thứ tự được xếp hàng.
27. Game Loop Pattern: Định nghĩa vòng lặp chính trong một trò chơi để xử lý cập nhật và vẽ các thành phần của trò chơi.
28. Service Locator Pattern: Cung cấp một cách để tìm kiếm và truy cập các dịch vụ trong hệ thống.
29. Data Locality Pattern: Tối ưu hóa hiệu suất bằng cách tập trung dữ liệu liên quan gần nhau trong bộ nhớ.
30. Dirty Flag Pattern: Đánh dấu các đối tượng khi chúng có thay đổi để tối ưu hóa việc lưu trữ hoặc xử lý dữ liệu.
31. Object Pool Pattern: Quản lý một pool các đối tượng sẵn sàng để sử dụng lại, thay vì tạo mới chúng mỗi khi cần.
32. Event Aggregator Pattern: Tập hợp các sự kiện từ nhiều nguồn và phân phối chúng cho các đối tượng quan tâm.
33. Snapshot Pattern: Lưu trữ trạng thái của một đối tượng hoặc toàn bộ hệ thống để có thể khôi phục lại trạng thái đó sau này.
34.Intercepting Filter Pattern: Mẫu này cho phép chúng ta thực hiện các xử lý trung gian trước và sau khi yêu cầu đi qua các thành phần khác nhau.
35.Null Object Pattern: Mẫu này cung cấp một đối tượng "null" thay thế cho các đối tượng thực tế, giúp tránh các kiểm tra null và hành vi không mong muốn.
36.Front Controller Pattern: Mẫu này tập trung vào việc xác định một điểm trung tâm để xử lý các yêu cầu vào của hệ thống.
37.Data Transfer Object (DTO) Pattern: Mẫu này được sử dụng để truyền dữ liệu giữa các thành phần khác nhau của hệ thống một cách hiệu quả.
38.Unit of Work Pattern: Mẫu này quản lý tập trung các thay đổi dữ liệu trong một đơn vị công việc, đồng thời đảm bảo tính nhất quán và ghi lại các thay đổi.
39.Composite Entity Pattern: Mẫu này tạo ra một đối tượng composite để đại diện cho một nhóm các đối tượng liên quan và đồng thời cung cấp cách để thao tác với nhóm đối tượng đó như một đơn vị duy nhất.
40.Double Checked Locking Pattern: Mẫu này được sử dụng để đảm bảo rằng chỉ có một tiến trình được phép truy cập và khởi tạo một đối tượng duy nhất trong môi trường đa luồng.
41.Extension Objects Pattern: Mẫu này cho phép mở rộng các phương thức và thuộc tính của một đối tượng mà không cần sửa đổi mã nguồn gốc của đối tượng đó.
42.Module Pattern: Mẫu này cho phép nhóm các lớp, giao diện và hàm liên quan lại với nhau trong một đơn vị độc lập để tạo thành một module có thể được sử dụng và mở rộng một cách dễ dàng.
43.Remote Proxy Pattern: Mẫu này cho phép truy cập và tương tác với các đối tượng từ xa thông qua việc sử dụng một proxy đại diện.

44.Role Object Pattern: Mẫu này tạo ra các đối tượng đại diện cho các vai trò trong một hệ thống và quản lý cách các đối tượng cộng tác với nhau dựa trên vai trò mà chúng đóng.
45.Value Object Pattern: Mẫu này đại diện cho các đối tượng không thay đổi (immutable) và được sử dụng để biểu diễn các giá trị mà không có tính chất thay đổi.
46.Identity Map Pattern: Mẫu này tạo ra một bản đồ để lưu trữ các đối tượng đã truy cập trong bộ nhớ để tránh việc truy cập nhiều lần và đảm bảo tính nhất quán dữ liệu.
47.Lazy Load Pattern: Mẫu này cho phép tải các đối tượng hoặc dữ liệu chỉ khi chúng được yêu cầu, giúp tối ưu hóa hiệu suất và tài nguyên.
48.Retry Pattern: Mẫu này cho phép tái thử lại các hoạt động thất bại một số lần nhất định để đảm bảo sự ổn định và hoàn chỉnh của hệ thống.