**Design Pattern**

**Creational Pattern**

1. Factory Pattern

Là pattern thuộc nhóm creational pattern, chức năng của nó dùng để tạo ra đối tượng mà không cần quan tâm tới logic lúc khởi tạo

Ưu điểm : dễ cài đặt

Nhược điểm :

Gây khó khăn khi có nhiều implentation của class đối tượng.

Khó để thay đổi cài đặt khi khởi tại đối tượng

1. Abstract Factory Pattern

Là Factory Pattern nhưng mở rộng hơn, thay vì khởi tạo đối tượng thì khởi tạo Factory

Ưu điểm : khắc phục nhược điểm trên của Factory Pattern khi có tính phân loại

Nhược điểm : khó để thay đổi cài đặt khi khởi tại đối tượng

1. Singleton Pattern

Tạo ra một đối tượng duy nhất của class trong quá trình chạy

Ưu điểm : đơn giản

Nhược điểm : khó kiểm soát để đảm bảo rằng chỉ có một đối tượng được tạo

1. Builder Pattern

Đưa ra cách tạo ra đối tượng cách chi tiết và dễ tùy chỉnh khi đối tượng có nhiều thuộc tính, khắc phục nhược điểm của Factory Pattern.

Ưu điểm : dễ custom

Nhược điểm : tăng độ phức tạp khi cài đặt

1. Prototype Pattern

Tạo ra các đối tượng mà nguồn để tạo ra đối tượng là giới hạn, VD (lấy object từ database)

Ưu điểm : tối ưu về tốc độ khi không spam vào nguồn dữ liệu khi bị hạn chế về phần cứng

Nhược điểm : khó kiểm soát khi dữ liệu gốc bị thay đổi, dẫn đến không đồng bộ

**Structual Pattern**

1. Adapter Pattern

Cho phép những object không liên quan đến nhau có thể hoạt động và tương thích với nhau

Ưu điểm : các object không cần có mối liên nào giữa chúng

Nhược điểm : cài đặt lằng nhằng nếu logic phức tạp

1. Bridge Pattern

Tạo ra cầu nối giữa các object mà có chung tính trừu tượng

Ưu điểm : object dễ dàng khởi tạo độc lập với nhau

Nhược điểm : tìm ra được tính trừ tượng tổng quát nhất của các object liên quan

1. FilterPattern

Dựa trên ý tưởng loose coupling để tạo ra các tác vụ độc lập nhằm để phân loại dữ liệu

9. Composite Pattern

Tạo ra các cấp độ quản lý của của cùng một loại object

10.Decorator Pattern

Tạo ra đối tượng linh hoạt mà không làm thay đổi đối tượng cha bằng cách tạo ra một wrapper để chứa đối tượng mới

11.Facade Pattern

Che đi logic phức tạp khi mà muốn sử dụng đối tượng

12.FlyWeigth Pattern:

Tạo ra và quản lý đối tượng sao cho giảm thiểu số đối tượng trùng lặp được tạo ra nhằm cả thiện về hiệu năng.

13.Proxy Pattern:

Tạo ra đối tượng mà được ủy quyền thông qua đối tượng gốc, điều này sẽ giúp việc dịnh nghĩa lại một số phương thức dễ dàng hơn và có nhiều tùy biến.

**Behavior Pattern**

14. Chain Responsibility Pattern

Tạo ra đối tượng dựa trên cấp bậc, và hoạt động theo cơ chế thác đổ. Đối tượng cấp cao hơn có thể làm được các task của nó đổ xuống, đối tượng cấp thấp hơn không thể làm được các task dành cho cấp cao.

15. Command Pattern

Các đối tượng Command sẽ được quản lý bởi một đối tượng khác như một tập các chỉ thị để thực hiện một công việc. Tập hợp các command đó sẽ phục vụ cho một công việc nhất định.

16. Interpreter Pattern

Hoạt động như một trình biên dịch để phân tích các cú pháp có thể xuất hiện trong một string, điều này giống như việc parse câu lệnh trong code.

17. Iterator Pattern

Đối tượng Iterator hoạt động như một con trỏ để phục vụ cho việc đọc dữ liệu từ một tập hợp

18. Mediator Pattern

Giúp các class giao tiếp với nhau thông qua một kênh chung, trong vd là system.out

19. Memento Pattern

Dùng để lưu lại các trạng thái cũ khi các object hoạt động.

20. Observer Pattern

Dùng để cập nhật trạng thái của đối tượng trong các mối liên hệ 1-n giữa các object

21. State Pattern:

Dùng để phân chia, tách nhỏ từng state của ứng dụng khi làm việc.