**Bài 1 tổng hợp:**

**Factory pattern**

Mục tiêu: giải quyết vấn đề khởi tạo một loại đối tượng, mà không cần phải chỉ định rõ chúng thuộc những lớp cụ thể nào. Factory method giải quyết vấn đề này bằng cách định nghĩa một phương thức cho việc tạo đối tượng, và việc quyết định kiểu đối tượng nào được tạo ra thì phụ thuộc vào các lớp con.

Factory Pattern được sử dụng trong trường hợp:

 Có một super class với nhiều class con và dựa trên yêu cầu input, cần trả về một sub class. Việc khởi tạo một lớp sẽ phụ thuộc vào lớp Factory thay vì do Client khởi tạo.

 Dùng Factory Pattern trong trường hợp cần làm giảm sự phụ thuộc giữa các module (loose coupling). Sử dụng hướng tiếp cận với Interface.

 Dùng Factory Pattern trong trường hợp cần tạo chuơng trình độc lập với những lớp cụ thể cần tạo 1 đối tượng, code ở phía client không thay đổi khi thay đổi logic ở factory hay sub class.

 Trường hợp cần mở rộng code sau khi thiết kế ứng dụng. Khi cần mở rộng, tạo ra sub class và implement thêm vào factory method.

 Trường hợp cần quản lý được life cycle của các object được tạo bởi Factory Pattern.

**Strategy Pattern**

Mục tiêu:

✓ Cung cấp một họ giải thuật khác nhau để giải quyết cùng 1 vấn đề nào đó và cho phép client chọn lựa linh động dễ dàng một giải thuật cụ thể theo từng tình huống sử dụng.

✓ Strategy pattern rất hữu ích khi chúng ta có nhiều thuật toán cho tác vụ cụ thể và chúng ta muốn ứng dụng linh hoạt lựa chọn thuật toán trong thời gian chạy cho tác vụ cụ thể.



**Bài 2 tổng hợp**

**State**

**Mục tiêu:**

State Pattern : Allow an object to alter its behavior when its internal state changes.The object will appear to change its class.

✓ Cho phép 1 đối tượng thay đổi hành vi khi trạng thái bên trong của nó thay đổi.

✓ Về nguyên lý chung, hành vi của đối tượng có thể phụ thuộc vào trạng thái hiện hành của đối tượng đó. Cách tốt nhất để giúp đối tượng thay đổi linh động và dễ dàng một hành vi phù hợp theo từng trạng thái là dùng mẫu State

