Tìm hiểu lại về event và delegate và action

Tìm hiểu xem thread là gì

Tìm hiểu singleton .. là gì

Core CLRS

Cross-platform support

Just-in-time (JIT) compilation

Garbage collection

WPF

Deserialization

Action

DirectoryInfo

Tim hiểu về các layout panel trong wpf

Bridge

Decorator

Strategy

**Serialization**

**Tuần tự hóa nhị phân:**

Binary Serialization là quá trình chuyển đổi các đối tượng thành định dạng nhị phân có thể được lưu trữ hoặc truyền tải hiệu quả. Nó bảo toàn toàn bộ trạng thái của đối tượng, bao gồm thông tin loại và các thành viên riêng tư. Binary Serialization phù hợp với các tình huống mà hiệu suất và hiệu quả là rất quan trọng.

**Tuần tự hóa XML:**

XML Serialization chuyển đổi các đối tượng thành định dạng XML (Ngôn ngữ đánh dấu mở rộng), giúp chúng có thể đọc được và tương thích với nhiều nền tảng và ngôn ngữ lập trình khác nhau. XML Serialization thường được sử dụng để trao đổi dữ liệu giữa các hệ thống không đồng nhất và để lưu trữ các thiết lập cấu hình trong các tệp XML.

**Tuần tự hóa JSON:**

JSON (JavaScript Object Notation) Serialization tương tự như XML Serialization nhưng sử dụng định dạng JSON để biểu diễn dữ liệu đối tượng. JSON Serialization nhẹ, dễ đọc và được sử dụng rộng rãi cho các ứng dụng và dịch vụ web dựa trên web do khả năng tương thích với JavaScript và các công nghệ web hiện đại.

**Chuỗi hóa hợp đồng dữ liệu:**

Data Contract Serialization là một tính năng của nền tảng Windows Communication Foundation (WCF) cho phép các đối tượng được tuần tự hóa và hủy tuần tự hóa bằng cách sử dụng các thuộc tính DataContract và DataMember. Data Contract Serialization cung cấp khả năng kiểm soát chi tiết đối với quy trình tuần tự hóa và thường được sử dụng trong các hệ thống phân tán và dịch vụ web.

### ****Creational Patterns (Mẫu Tạo Ra)****

Nhóm mẫu này tập trung vào cách tạo đối tượng, làm cho quá trình tạo ra đối tượng trở nên linh hoạt và hiệu quả hơn.

* **Factory Method:** Cung cấp một cách tạo đối tượng mà không cần chỉ định lớp cụ thể.
* **Abstract Factory:** Tạo ra các họ đối tượng liên quan hoặc phụ thuộc mà không cần chỉ định lớp cụ thể.
* **Builder:** Tạo đối tượng phức tạp từng bước.
* **Prototype:** Tạo đối tượng mới bằng cách sao chép đối tượng mẫu.
* **Singleton:** Đảm bảo chỉ có một thể hiện duy nhất của một lớp và cung cấp một điểm truy cập toàn cục.

### 2. ****Structural Patterns (Mẫu Cấu Trúc)****

Nhóm mẫu này tập trung vào cách tổ chức các lớp và đối tượng để tạo thành cấu trúc lớn hơn và dễ quản lý hơn.

* **Adapter (Wrapper):** Cho phép các lớp có giao diện không tương thích làm việc cùng nhau.
* **Bridge:** Tách biệt một lớp trừu tượng khỏi lớp cụ thể của nó để chúng có thể thay đổi độc lập.
* **Composite:** Cho phép bạn cấu trúc các đối tượng thành các cấu trúc cây, để khách hàng có thể làm việc với đối tượng đơn lẻ hoặc tập hợp các đối tượng một cách đồng nhất.
* **Decorator:** Thêm chức năng cho đối tượng mà không thay đổi mã nguồn của đối tượng đó.
* **Facade:** Cung cấp một giao diện đơn giản cho một hệ thống phức tạp.
* **Flyweight:** Giảm thiểu số lượng đối tượng tạo ra bằng cách chia sẻ các đối tượng giống nhau.
* **Proxy:** Cung cấp một đại diện hoặc thay thế cho một đối tượng để kiểm soát quyền truy cập đến đối tượng đó.

### 3. ****Behavioral Patterns (Mẫu Hành Vi)****

Nhóm mẫu này tập trung vào cách các đối tượng và lớp tương tác với nhau và phân phối trách nhiệm giữa chúng.

* **Chain of Responsibility:** Cho phép một đối tượng truyền yêu cầu qua một chuỗi các đối tượng xử lý cho đến khi yêu cầu được xử lý.
* **Command:** Biến một yêu cầu thành đối tượng để bạn có thể truyền, lưu trữ và thực thi yêu cầu sau.
* **Interpreter:** Cung cấp cách để xử lý ngôn ngữ hoặc cú pháp bằng cách định nghĩa các quy tắc ngữ nghĩa.
* **Iterator:** Cung cấp một cách để truy cập các phần tử của một tập hợp mà không cần phải lộ cấu trúc bên trong của tập hợp.
* **Mediator:** Cung cấp một đối tượng để trung gian giao tiếp giữa các đối tượng mà không yêu cầu các đối tượng biết đến nhau.
* **Memento:** Cho phép bạn lưu trữ và khôi phục trạng thái của một đối tượng mà không cần lộ cấu trúc bên trong của nó.
* **Observer:** Cho phép một đối tượng thông báo và cập nhật các đối tượng khác khi nó thay đổi trạng thái.
* **State:** Cho phép một đối tượng thay đổi hành vi của nó khi trạng thái nội bộ của nó thay đổi.
* **Strategy:** Cho phép định nghĩa một họ các thuật toán, đóng gói từng thuật toán và làm cho chúng có thể thay đổi một cách linh hoạt.
* **Template Method:** Định nghĩa một thuật toán trong lớp cơ sở, để các lớp con có thể thay đổi một số bước của thuật toán mà không thay đổi cấu trúc của nó.
* **Visitor:** Cho phép thêm hành vi mới vào các lớp hiện có mà không thay đổi các lớp đó.