**Observer pattern**

## Lý thuyết Observer Pattern:

Observer Pattern (mẫu thiết kế quan sát viên) là một mẫu thiết kế hành vi (behavioral design pattern) phổ biến trong lập trình hướng đối tượng. Mẫu này giúp bạn thiết lập một mối quan hệ một-nhiều giữa các đối tượng, trong đó một đối tượng (Subject) sẽ phát đi thông báo khi trạng thái của nó thay đổi, và các đối tượng khác (Observers) sẽ lắng nghe và phản hồi những thông báo này.

Cấu trúc:

* Subject: Lớp đại diện cho đối tượng có thể thay đổi trạng thái và phát đi thông báo.
* Observer: Lớp đại diện cho các đối tượng muốn nhận thông báo từ Subject.
* Attach() & Detach() Methods: Các phương thức để Observer đăng ký hoặc hủy đăng ký nhận thông báo từ Subject.
* Notify() Method: Phương thức Subject sử dụng để gửi thông báo cho các Observer đã đăng ký.

Lợi ích:

* Giảm thiểu sự phụ thuộc: Giúp các Observer không phụ thuộc trực tiếp vào Subject, giúp dễ dàng thay đổi hoặc thêm/bỏ Observer.
* Tăng tính linh hoạt: Giúp bạn dễ dàng thêm/bỏ Observer mới mà không ảnh hưởng đến Subject và các Observer khác.
* Tái sử dụng: Subject và Observer có thể được tái sử dụng trong nhiều trường hợp khác nhau.

## Ví dụ

namespace Observer

{

    public class Observer : IObserver

    {

        private Dictionary<string, Delegate> observers = new Dictionary<string, Delegate>();

        public void Attach(string obsName, Delegate method)

        {

            if (!observers.ContainsKey(obsName))

            {

                observers.Add(obsName, method);

            }

            else

            {

                observers[obsName] = Delegate.Combine(observers[obsName], method);

            }

        }

        public void Detach(string obsName, Delegate method)

        {

            if (observers.ContainsKey(obsName))

            {

                observers[obsName] = Delegate.Remove(observers[obsName], method);

            }

        }

        public void Notify(string obsName, params object[] args)

        {

            if (observers.ContainsKey(obsName))

            {

                Delegate del = observers[obsName];

                if (del != null)

                {

                    if (del.GetInvocationList().Length > 0)

                    {

                        foreach (Delegate handler in del.GetInvocationList())

                        {

                            if (handler.Method.GetParameters().Length == 0)

                            {

                                handler.DynamicInvoke();

                            }

                            else

                            {

                                ParameterInfo[] parametersInfo = handler.Method.GetParameters();

                                object[] parameters = new object[parametersInfo.Length];

                                for (int i = 0; i < parametersInfo.Length; i++)

                                {

                                    if (i < args.Length && args[i] != null && args[i].GetType() == parametersInfo[i].ParameterType)

                                    {

                                        parameters[i] = args[i];

                                    }

                                    else

                                    {

                                        parameters[i] = GetDefaultValue(parametersInfo[i].ParameterType);

                                    }

                                }

                                handler.DynamicInvoke(parameters);

                            }

                        }

                    }

                }

            }

        }

        private object GetDefaultValue(Type type)

        {

            if (type.IsValueType)

            {

                return Activator.CreateInstance(type);

            }

            return null;

        }

    }

}

Trong ví dụ trên tôi tạo các class Observer và khai báo các hàm sau

* Attach(string obsName, Delegate method):
* Phương thức này cho phép đính kèm (attach) một phương thức (delegate) vào một key cụ thể trong dictionary của observers.
* Nếu key chưa tồn tại, phương thức sẽ được thêm vào dictionary.
* Nếu key đã tồn tại, phương thức sẽ được kết hợp với các phương thức khác đã được đính kèm vào key đó.
* Detach(string obsName, Delegate method):
* Phương thức này cho phép tách rời (detach) một phương thức khỏi key cụ thể trong dictionary của observers.
* Nếu key tồn tại và chứa phương thức cần tách ra, phương thức sẽ được loại bỏ khỏi danh sách phương thức của key đó.
* Notify(string obsName, params object[] args):
* Phương thức này thông báo (notify) tất cả các phương thức đã được đính kèm vào một key cụ thể.
* Nó kiểm tra xem key có tồn tại trong dictionary không. Nếu có, nó lấy danh sách các phương thức đã đính kèm vào key đó.
* Với mỗi phương thức, nó kiểm tra số lượng và loại tham số của phương thức, sau đó gọi phương thức đó với đối số được truyền vào nếu có.
* private object GetDefaultValue(Type type):
* Phương thức này trả về giá trị mặc định cho một kiểu dữ liệu nếu nó là kiểu giá trị (value type).
* Nếu kiểu dữ liệu là value type, nó sẽ tạo một đối tượng mới của kiểu đó bằng cách sử dụng Activator.CreateInstance.
* Nếu kiểu dữ liệu không phải là value type, nó trả về null.