Designmönster: Observer pattern

Varför valde jag Observer Pattern? Observer Pattern är perfekt för realtidskommunikation, där flera användare ska få uppdateringar automatiskt utan att behöva ladda om sidan. I Tech Talk Forum (TTF) behöver användare få nya meddelanden direkt när de skickas i en kanal. Observer Pattern gör att varje användare ("Observer") får en notifiering när något händer i chattkanaler ("Subject").

Förbättringar:

- Användarna får omedelbara uppdateringar, vilket förbättrar användarupplevelsen.
- Det är effektivt, eftersom servern inte behöver skicka uppdateringar till alla användare hela tiden – bara när något händer.
- Strukturen gör det enkelt att lägga till fler funktioner (t.ex. pushnotiser).

Use case: notifiering av nya meddelanden

- **Pre-villkor:** Användaren är medlem i en chattkanal.
- **Post-villkor:** Alla medlemmar i kanalen får det nya meddelandet i realtid.

Huvudflöde:

1. En användare skickar ett meddelande i en kanal.
2. Meddelandet skickas till servern.
3. Servern notifierar alla anslutna användare i kanalen.
4. Meddelandet visas i realtid på deras skärmar.

User Story: *"Som användare vill jag automatiskt få nya meddelanden i mina kanaler i realtid, så att jag snabbt kan delta i diskussioner utan att behöva uppdatera sidan."*

Designmönster: Singleton Pattern

Varför valde jag Singleton Pattern? Singleton Pattern hjälper oss att säkerställa att det endast finns en instans av de centrala komponenterna i vår chat applikation, till exempel servern och databasanslutningarna. Detta mönster passar bra för en chat applikation där vi vill ha en global kontrollpunkt för att hantera alla användares meddelanden, anslutningar och eventuell databasinteraktion.

Förbättringar:

- Enkel hantering av resurser: Vi slipper skapa flera instanser av samma resurs, vilket sparar minne och processorkraft.
- Centraliserad hantering: Alla meddelanden, användaranslutningar och serverfunktioner hanteras genom en enda instans, vilket förenklar kodbasen och gör den mer överskådlig.
- Prestanda Fördelar: Eftersom vi använder en och samma instans för servern, hanterar vi alla anslutningar och meddelanden på ett effektivt sätt, utan att skapa onödiga anslutningar eller instanser.

Use case: Skicka meddelande i en gruppchatt

- **Pre-villkor:** Server Instansen (som är en Singleton) är redan igång och hanterar serverkommunikation.
- **Post-villkor:** Användaren skickar ett meddelande till en grupp som ska tas emot av alla medlemmar i gruppen.

Huvudflöde:

1. Användaren öppnar chattapplikationen och ansluter till en grupp.
2. När användaren skriver ett meddelande, skickar applikationen det till singleton-server instansen.

3. Server Instansen hanterar meddelandet och skickar det vidare till alla medlemmar i gruppen.
4. Alla användare i gruppen tar emot meddelandet i realtid.

User Story: *"Som användare vill jag kunna skicka meddelanden i en gruppchatt så att alla medlemmar i gruppen får meddelandet i realtid."*

Kodexempel:

```
class User : IObserver
{
    2 references
    public string Name { get; }

    2 references
    public User(string name)
    {
        Name = name;
    }

    2 references
    public void Update(string message)
    {
        Console.WriteLine($"{Name} fick ett nytt meddelande: {message}");
    }
}
2 references
class ChatChannel
{
    private List<IObserver> users = new List<IObserver>();

    2 references
    public void AddUser(IObserver user)
    {
        users.Add(user);
    }

    1 reference
    public void SendMessage(string message)
    {
        Console.WriteLine($"Nytt meddelande i kanalen: {message}");
        foreach (var user in users)
        {
            user.Update(message);
        }
    }
```

No issues found

```
public class Chatserver
{
    private static Chatserver _instance;

    private List<string> _connections;

    1 reference
    private Chatserver()
    {
        _connections = new List<string>();
    }
    1 reference
    public static Chatserver GetInstance()
    {
        if (_instance == null)
        {
            _instance = new Chatserver();
        }
        return _instance;
    }
    2 references
    public void AddUser(string user)
    {
        _connections.Add(user);
        Console.WriteLine($"{user} har anslutit.");
    }
    1 reference
    public void SendMessage(string message)
    {
        foreach (var user in _connections)
        {
            Console.WriteLine($"Skickar till {user}: {message}");
        }
    }
}
}
```

```
0 references
class Program
{
    0 references
    static void Main()
    {
        var server = Chatserver.GetInstance();
        ChatChannel channel = new ChatChannel();

        server.AddUser("User1");
        server.AddUser("User2");

        User Maria = new User("Maria");
        User Frank = new User("Frank");

        channel.AddUser(Maria);
        channel.AddUser(Frank);

        server.SendMessage("Hej alla!");
        channel.SendMessage("Välkommen till Tech Talk Forum");

    }
}
```