Observer Practical

Github Link:

console application that simulates a network provider system, with functionality for user subscriptions, color-coding for network identification, and the capability for networks to notify users of faults and downtimes

# Paragraph

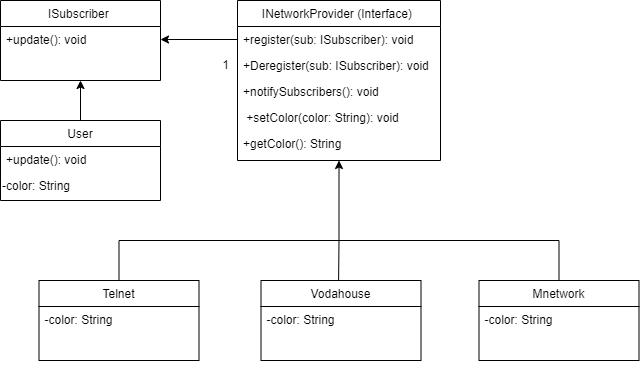
The scenario is suitable for the Observer Pattern because the network providers act as subjects (publishers) while the users act as observers (subscribers). The Observer Pattern allows for a decoupled and scalable design where the network providers don't need to know the details of their subscribers, and subscribers can dynamically subscribe or unsubscribe from multiple providers without affecting the providers themselves.

# Output Screenshots

A screenshot of a computer

Description automatically generated

# Class Diagram



# Code

using System;

using System.Collections.Generic;

namespace ObserverPractical

{

// Interface for subscribers, defining the method to receive updates

public interface ISubscriber

{

// Method to be called when there's an update from the network

void Update(string message);

}

// Interface for network providers, defining methods for subscriber management and notifications

public interface INetworkProvider

{

void RegisterSubscriber(ISubscriber subscriber);

void DeregisterSubscriber(ISubscriber subscriber);

void NotifySubscribers(string message);

void SetColor(string color);

string GetColor();

}

// Represents a user who subscribes to network updates

public class User : ISubscriber

{

public string Name { get; set; } // User's name

public string Color { get; private set; } // The color representing the network provider

public User(string name)

{

Name = name;

}

// Method to handle updates received from subscribed network provider

public void Update(string message)

{

Console.WriteLine($"{Name} received a message: {message}");

}

// Sets the color based on the network provider's identification color

public void SetColor(string color)

{

Color = color;

}

}

// Abstract class representing a generic network provider

public abstract class NetworkProvider : INetworkProvider

{

private readonly List<ISubscriber> subscribers = new List<ISubscriber>(); // List of subscribers

public string Color { get; private set; } // Network identification color

// Registers a subscriber to receive updates

public void RegisterSubscriber(ISubscriber subscriber)

{

subscribers.Add(subscriber);

}

// Deregisters a subscriber from receiving updates

public void DeregisterSubscriber(ISubscriber subscriber)

{

subscribers.Remove(subscriber);

}

// Notifies all registered subscribers of a message/update

public void NotifySubscribers(string message)

{

foreach (var subscriber in subscribers)

{

subscriber.Update($"Notification from {this.GetType().Name} (Color: {Color}): {message}");

}

}

// Sets the network provider's identification color

public void SetColor(string color)

{

Color = color;

}

// Gets the network provider's identification color

public string GetColor()

{

return Color;

}

}

// Specific network provider classes inheriting from NetworkProvider

// Each sets a unique color identifying the network provider

public class Telnet : NetworkProvider

{

public Telnet()

{

SetColor("Blue");

}

}

public class Vodahouse : NetworkProvider

{

public Vodahouse()

{

SetColor("Red");

}

}

public class Mnetwork : NetworkProvider

{

public Mnetwork()

{

SetColor("Yellow");

}

}

public class CMac : NetworkProvider

{

public CMac()

{

SetColor("Black");

}

}

// Main program class to demonstrate usage

class Program

{

static void Main(string[] args)

{

var telnet = new Telnet(); // Create a new network provider

var user1 = new User("Gabriel"); // Create a new user

user1.SetColor(telnet.GetColor()); // Set user's color based on network provider's color

telnet.RegisterSubscriber(user1); // Subscribe the user to the network provider

// network notification

telnet.NotifySubscribers("We are experiencing a temporary outage. We apologize for the inconvenience.");

var Mnetwork = new Mnetwork(); // Create a new network provider

var user2 = new User("Nikita"); // Create a new user

user2.SetColor(Mnetwork.GetColor()); // Set user's color based on network provider's color

Mnetwork.RegisterSubscriber(user2); // Subscribe the user to the network provider

// network notification

Mnetwork.NotifySubscribers("We are experiencing a temporary outage. We apologize for the inconvenience.");

}

}

}