Rachit Manglik • 2nd
Software engineer 2 | C# | .NET 8 | MVC | MySQL | REST API | jQuery | Apache Kafka | RabbitM...
1d • Edited •
Follow

🎯 **Observer Pattern**

Imagine **one main object** (**Subject**) is being "watched" by others (Observers).

Whenever the main object **changes**, it **automatically tells everyone watching**. No need to ask again and again — observers will get notified when subject changes it's state

🚀 **Observer pattern in layman terms**

🔷You're **waiting for Mangoes** to come back on Blinkit.

🔷You click "**Notify Me**" — now **you're an *Observer***.

🔷When Mangoes are restocked, Blinkit (the *Subject*) will alert you. **You didn't have** to **keep checking manually**.

🚀 **Why do we need observer pattern** 🤔

🔷**Every dependent object keeps asking about the state of the subject** (**in this case it is mangoes**).

"Has the value changed yet?" That's like refreshing Blinkit every minute to see if mangoes are back.

🔷**Problem It Solves**- You dont have to **manually check** if something has changed. You will be notified when the state of the subject changes.

🚀 **Understanding terms related to this pattern** 👀 (**Refer** to the **code** attached)😎

🔷**Subject** - The **Out-of-Stock Item** (e.g., Mangoes).

🔷**Observers** - All users who pressed "**Notify Me**".

🔷**Attach**() - When you click "Notify Me", you get **added to** the **observer list**.

🔷**Notify**()- When the item is back in stock, Blinkit **notifies** all **observers**.

🚀 **Pros**:

🔷**Open/Closed Principle** You can make new subscriber (observer) classes **without modifying** the publisher (subject).

🔷**Dynamic Subscription** Observers can **subscribe/unsubscribe at runtime**, allowing flexible relationships between objects.

🔷**Automatic Notification** When the subject's state changes, all observers get updated automatically—no

manual polling needed.

🚀 **Cons**:

🔷**Notification Order Isn't Guaranteed**- Observers are notified in no specific order, which might cause issues if sequence matters.

🔷**Tight Coupling via Interfaces** Even though loosely coupled in behavior, all observers must implement a common interface, which might increase complexity.

🔷**Potential Memory Leaks** If observers aren't properly unsubscribed, they may remain in memory for longer time period.

🚀🚀 #ObserverPattern #DesignPatterns #BehavioralPattern #SoftwareDesign #SOLIDPrinciples #CleanCode #SoftwareEngineering #EventDriven #CSharp #DotNet #OOP #Developers #Coding #SoftwareDevelopment

```
// Observer interface
public interface IObserver
{
    void Update(string productName);
}

// Subject interface
public interface IProductNotifier
{
    void AddObserver(IObserver observer);
    void RemoveObserver(IObserver observer);
    void NotifyObservers();
}

// Concrete Subject
public class Product : IProductNotifier
{
    private List<IObserver> _observers = new();
    private bool _isAvailable;
    private string _productName;

    public Product(string name)
    {
        _productName = name;
    }

    public void AddObserver(IObserver observer) => _observers.Add(observer);

    public void RemoveObserver(IObserver observer) => _observers.Remove(observer);

    public void NotifyObservers()
    {
        foreach (var observer in _observers)
        {
            observer.Update(_productName);
        }
    }

    public void SetAvailability(bool isAvailable)
    {
        _isAvailable = isAvailable;
        if (_isAvailable)
        {
            NotifyObservers(); // Notify only when back in stock
        }
    }
}

// Concrete Observer
public class Customer : IObserver
{
    private string _customerName;

    public Customer(string name)
    {
        _customerName = name;
    }

    public void Update(string productName)
    {
        Console.WriteLine($"Hey {_customerName}, {productName} is now back in stock!");
    }
}


var mangoes = new Product("Mangoes");

var customer1 = new Customer("Amit");
var customer2 = new Customer("Sneha");

mangoes.AddObserver(customer1);
mangoes.AddObserver(customer2);

// Later when mangoes are restocked
mangoes.SetAvailability(true);
```