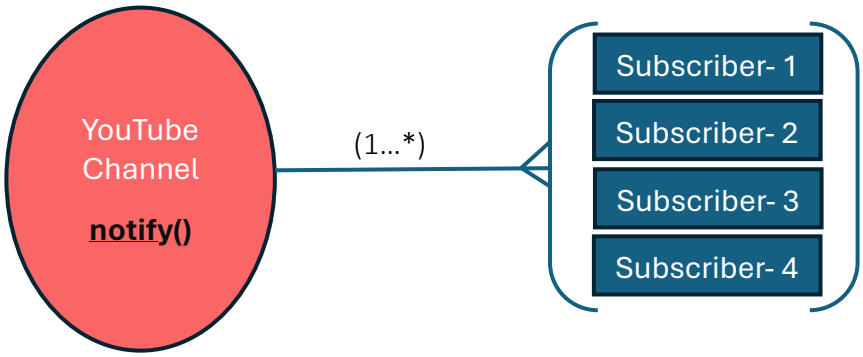
Observer Design Pattern

Think of a YouTube Channel (*Subject*). When a user subscribes to the channel (becomes an *Observer*), they get notified whenever a new video (state change) is uploaded. It is a real-world example of observer design pattern.

Defⁿ: The *Observer Design Pattern* is a ***behavioural design pattern*** that defines a ***one-to-many*** dependency between ***objects*** so that when ***one object*** (called the Subject) ***changes its state***, all its ***dependents*** (called Observers) are ***notified*** and ***updated automatically***.



1. Polling Technique:

Polling is when the **observer** (client) **repeatedly asks** (or "polls") the subject (server) at **regular intervals** whether there is any new data or change.

Example:

A weather subscriber object checks the youtube channel object every 10 minutes to see if the weather has uploaded any new video.

Advantages:

- ✓ Simple to implement.
- ✓ Observer has the control when to check.

Disadvantages:

- × Wastes resources if nothing has changed.
- × May miss real-time updates (delayed response).
- × Increases server load with frequent checks.

2. Pushing Technique:

Pushing is when the **subject** (server) **actively sends updates** to the **observer** (client) **as soon as** something **changes**.

Example:

You get notified instantly when youtuber uploads a video — the youtube channel object "pushes" the notification to your app.

Advantages:

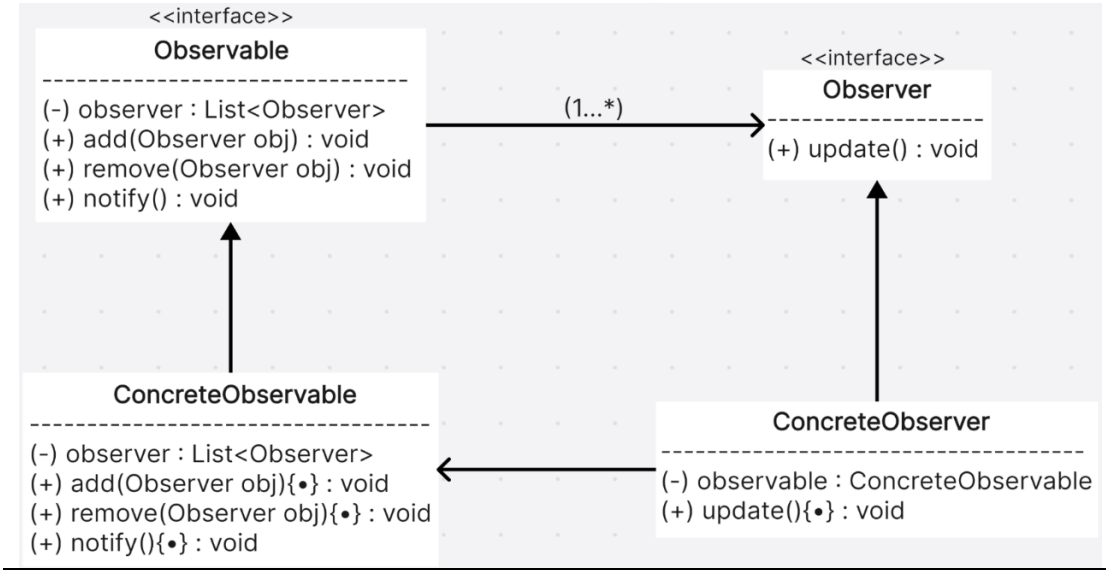
- ✓ Real-time notifications.
- ✓ More efficient — no need for repeated checking.

Disadvantages:

- × More complex to implement (e.g., needs WebSockets or similar).
- × Requires persistent connection in some cases.

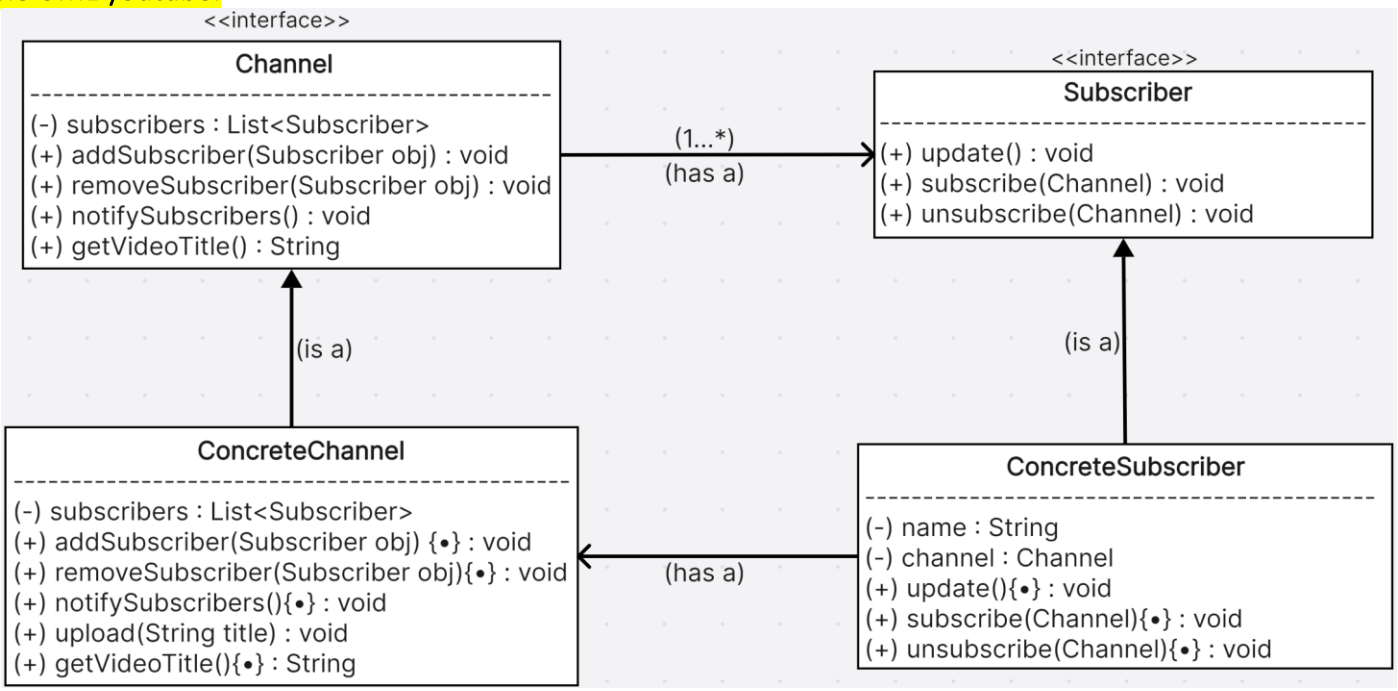
@Sibasundaj8

Standard UML of Observer Design Pattern:



Let's implement the notification system of youtube using observer design pattern.

Sample UML youtube:



🎯 This is a clean, correct **UML** for implementing a **YouTube**-style **notification system** using the **Observer Design Pattern**, with:

- Proper interface usage
- Clean architecture
- Good naming
- Solid relationships

Practical use cases of Observer Design Pattern
🖥️ **GUI Frameworks (e.g., Java Swing, .NET).** Use case: Event handling for buttons, sliders, windows, etc.
🔔 **Notification Systems.** Use case: Email, SMS, push notifications, etc.
📈 **Model-View-Controller (MVC) Architecture.** Use case: Keeping UI (View) updated with the data (Model).
⏰ **Real-Time Monitoring Systems.** Use case: Monitoring servers, sensors, or system metrics.
📡 **Social Media and News Feeds.** Use case: Following topics or users to receive updates.
🤖 **Game Development.** Use case: Notifying multiple systems when an event occurs.

Code Link: https://github.com/sibasundarj8/-System-Design-/tree/main/Codes/12___Observer%20Design%20Pattern%20code

@Sibasundaj8