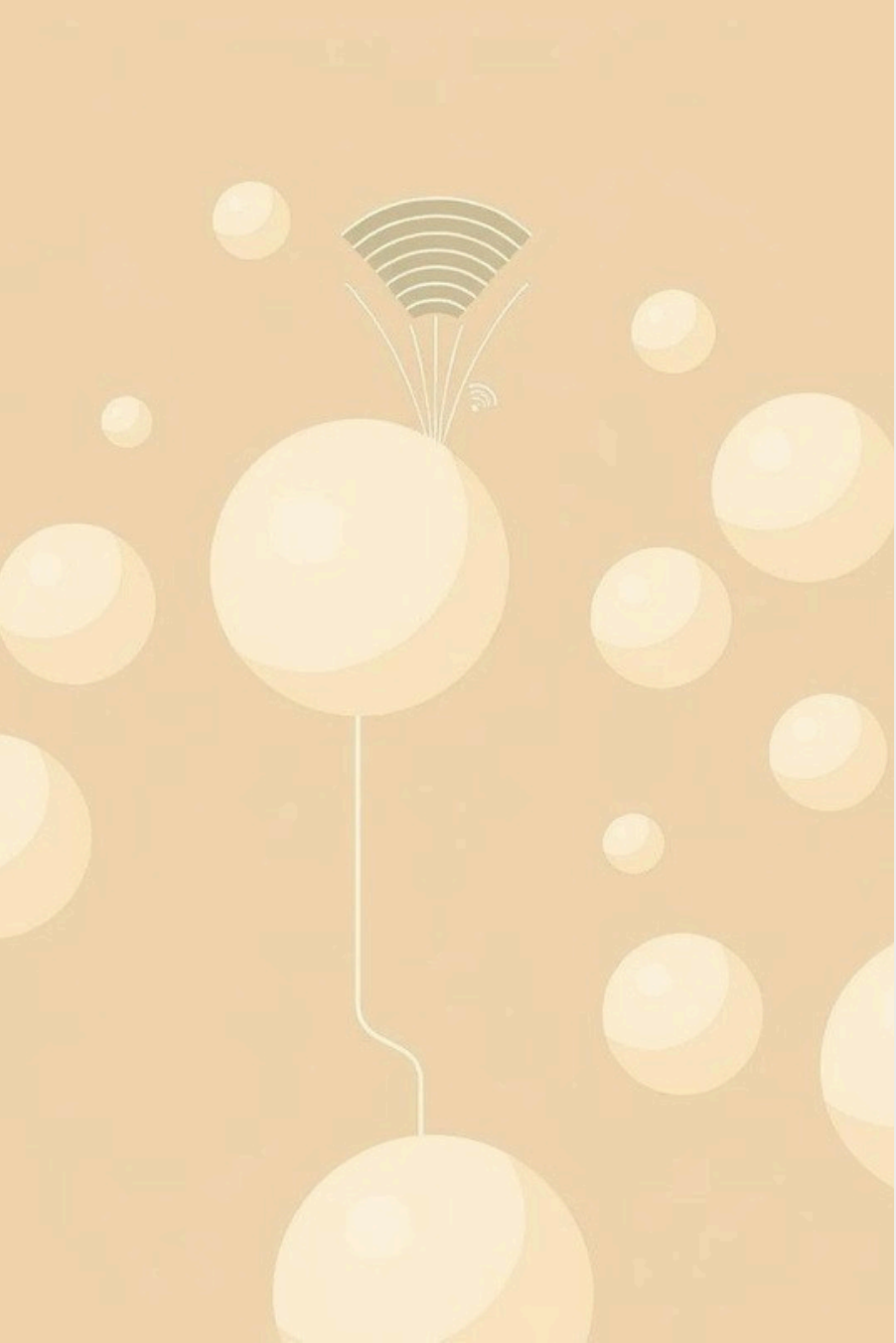
Dominando el Patrón Observer

Angel E. Concepción Capellán ID 1124530

ING-IDS309-01 ARQUITECTURA DE SOFTWARE

Docente: GABRIEL ELIVAN VILLALONA EUSEBIO

Tabla de Contenidos: Su Viaje a Través del Patrón Observer



¿Qué es el Patrón Observer?

El Patrón Observer es un patrón de diseño de comportamiento que define una dependencia uno-a-muchosentre objetos. Cuando el estado de un objeto,conocidocomoel **Sujeto (Subject)**, cambia, todos sus dependientes, llamados **Observadores (Observers)**, son notificados automáticamente y actualizados.

Este patrón es ideal para escenarios donde se requiere una actualización en tiempo real o una propagación de cambios de manera eficiente y desacoplada, sin que el sujeto necesite conocer los detalles de sus observadores.

El Problema: Desacoplamiento y Actualizaciones en Tiempo Real

El reto:

Imagina sistema donde múltiples componentes necesitan reaccionar a cambios en undatocentral .Una implementación directa podría llevara un acoplamiento fuerte, donde el objeto central sabe demasiado sobre sus dependientes, haciendo el sistema rígido y difícil de mantener.

Acoplamiento Fuerte

El Sujetotendría que conocer y llamar directamente a todos los Observadores, lo que lo haría inflexible ante la adición o eliminación de observadores.

Mantenimiento Complejo

Cambios en un Observador podrían requerir modificaciones en el Sujeto, aumentando la complejidad del mantenimiento.

Escalabilidad Limitada

Dificultad para ñadirnuevos observadores o para que el Sujeto maneje un número creciente de dependencias.

El patrón Observer resuelve este problema al establecer una comunicación indirecta y flexible.

Caso Práctico: Sistema de Alertas en la Bolsa de Valores

Consideremos un sistema demonitoreo de labolsa de valores.Tenemos:

- **Acciones:** El precio de una acción es un **Sujeto** que puede cambiar constantemente.
- **Inversores:** Cada inversor es un **Observador** interesado en una acción específica. Quiere ser notificado inmediatamente si el precio de esa acción cambia.

Sin el patrón Observer, la acción necesitaría una lista de todos los inversores y llamar a cada uno para notificarles un cambio. Esto es ineficiente y acopla la acción a la lógica de notificación de cada inversor.

Con Observer, la acción solo necesita mantener una lista genérica de observadores y notificarlos a través de una interfaz común, sin saber quiénes son o cómo procesan la notificación.

Componentes Principales del Patrón Observer

Subject (Sujeto)

Es el objeto quemantiene el estado y al que los observadores se suscriben. Proporciona métodos para adjuntar (attach), desvincular (detach) y notificar (notify) a los observadores.

Observer (Observador)

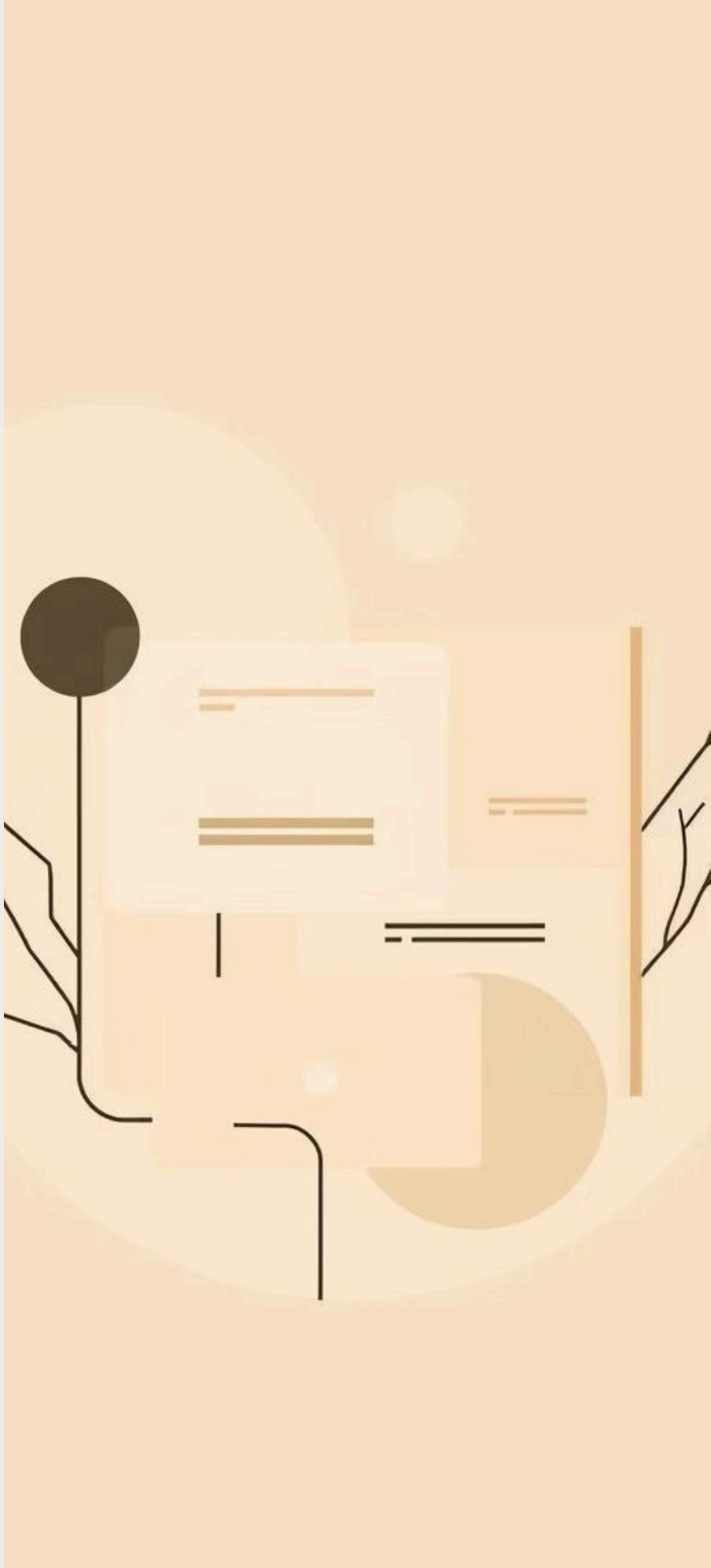
Define una interfazdeactualización para los objetos que deben ser notificados de los cambios en el sujeto. Normalmente, tiene un método update().

Concrete Subject (Sujeto Concreto)

Es laimplementaciónconcretadel Sujeto.Almacena el estado que interesa a los observadores y los notifica cuando este estado cambia.

Concrete Observer (Observador Concreto)

Esl aimplementación concreta delObservador.Almacena la referencia al Sujeto Concreto y actualiza su estado cuando recibe una notificación.



Ventajas y Desventajas

Ventajas

Acoplamiento Débil
ElSujetoy losObservadores son independientes. El Sujeto no conoce las implementaciones específicas de los Observadores.

Reusabilidad
LosObservadores pueden ser reutilizados con diferentes Sujetos sin modificaciones.

Extensibilidad
Esfácilañadirnuevos Observadores sin modificar el Sujeto existente.

Comunicación en Tiempo Real
Idealparasistemasdondeloscambiosdeben ser propagados instantáneamente.

Desventajas

Orden de Notificación
El orden en que los Observadores son notificados puede ser impredecible y causar problemas si el orden es importante.

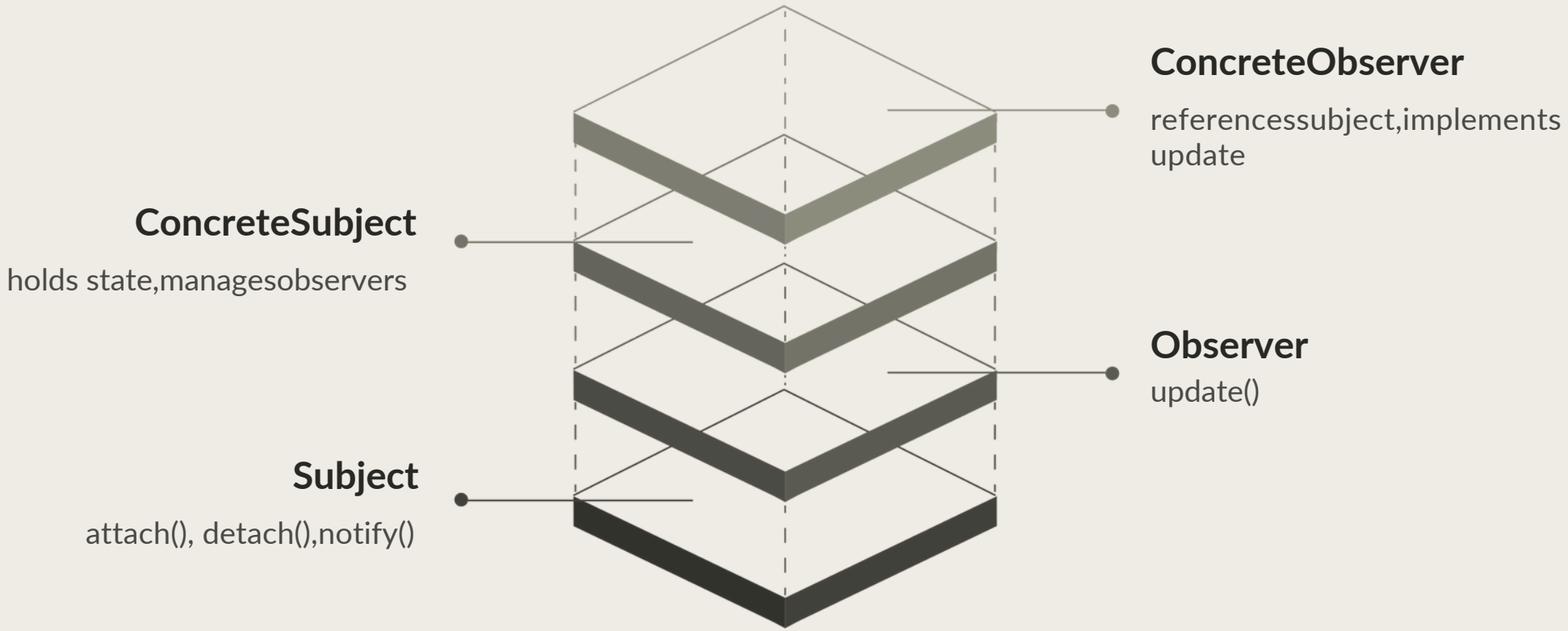
Sobrecarga de Observadores
SihaymuchosObservadoresylasactualizaciones son frecuentes, el rendimiento puede verse afectado.

Fugas de Memoria
SilosObservadoresnose desvinculan correctamente del Sujeto, pueden generarse referencias circulares y fugas de memoria.

Complejidad Adicional
Puedeañadir unacapadeabstracción que no siempre es necesaria para sistemas simples.

Diagrama UML del Patrón Observer

Estediagramailustralaestructurafundamentaldelpatrón Observer, mostrandolasrelacionesentreelSujeto, el Observador y sus implementaciones concretas.



El Sujeto mantiene una lista de Observadores. Cuando su estado cambia, invoca el método `notify()`, que a su vez llama al método `update()` en cada Observador registrado.

Implementación en Código: Sistema de Alertas Bursátiles

Acontinuación, semuestra unaimplementaciónsimplificadadelpatrónObserverutilizandoelcasoprácticodelsistemadealertasenlabolsadevalores.Este ejemplo utiliza Java, pero el concepto es aplicable a cualquier lenguaje orientado a objetos.

Interfaz Observer

```
public interface Observer {
void update(String stockSymbol, double price);
}
```

Interfaz Subject

```
public interface Subject {
    void attach(Observer observer);
    void detach(Observer observer);
    void notifyObservers();
}
```

Stock (Sujeto Concreto)

```
public class Stock implements Subject {
    privateList<Observer> observers = new ArrayList<>();
    privateString symbol;
    privatedouble price;
    publicStock(String symbol, double price) {

        this.symbol = symbol;
        this.price = price;
    }
    publicvoid setPrice(double newPrice) {

        System.out.println("Price of " + symbol + " changed from " +
price+"to " + newPrice);
        this.price = newPrice;
        notifyObservers();
    }
    @Override
    publicvoid attach(Observer observer) {

        observers.add(observer);
    }
    @Override
    publicvoid detach(Observer observer) {

        observers.remove(observer);
    }
    @Override
    publicvoid notifyObservers() {

         for(Observer observer : observers) {
            observer.update(symbol, price);
        }
    }
}
```

Investor (Observador Concreto)

```
public class Investor implements Observer {
private String name;
 public Investor(String name) {

        this.name = name;
    }

 @Override
public void update(String stockSymbol, double price) {
        System.out.println(name + " received alert: " + stockSymbol
+ " is now $" + price);
    }
}
```

Este código demuestra cómo la clase`Stock` (Sujeto Concreto) maneja el registro y la notificación de los`Investor` (Observadores Concretos) cada vez que el precio de la acción cambia.

Conclusiones y Siguientes Pasos

Recapitulación del Patrón Observer

Claro

Hemos explorado el Patrón Observer,un mecanismo robustoparala notificacióndecambios de estado, promoviendo el desacoplamiento y la flexibilidad en el diseño de software. Es una herramienta esencial para la creación de sistemas dinámicos y reactivos.

El patrón facilita una comunicación limpia y estructurada entre objetos.

Didáctico

Su lógica es intuitiva, lo que lo hace fácil de enseñar y de aprender.

Profesional

Adoptar este patrón eleva la calidad y mantenibilidad del código.