



ШАБЛОН (ПАТЕРН) ПРОЄКТУВАННЯ ПЗ **OBSERVER**

Верясов Владислав ПЗПІ-22-2

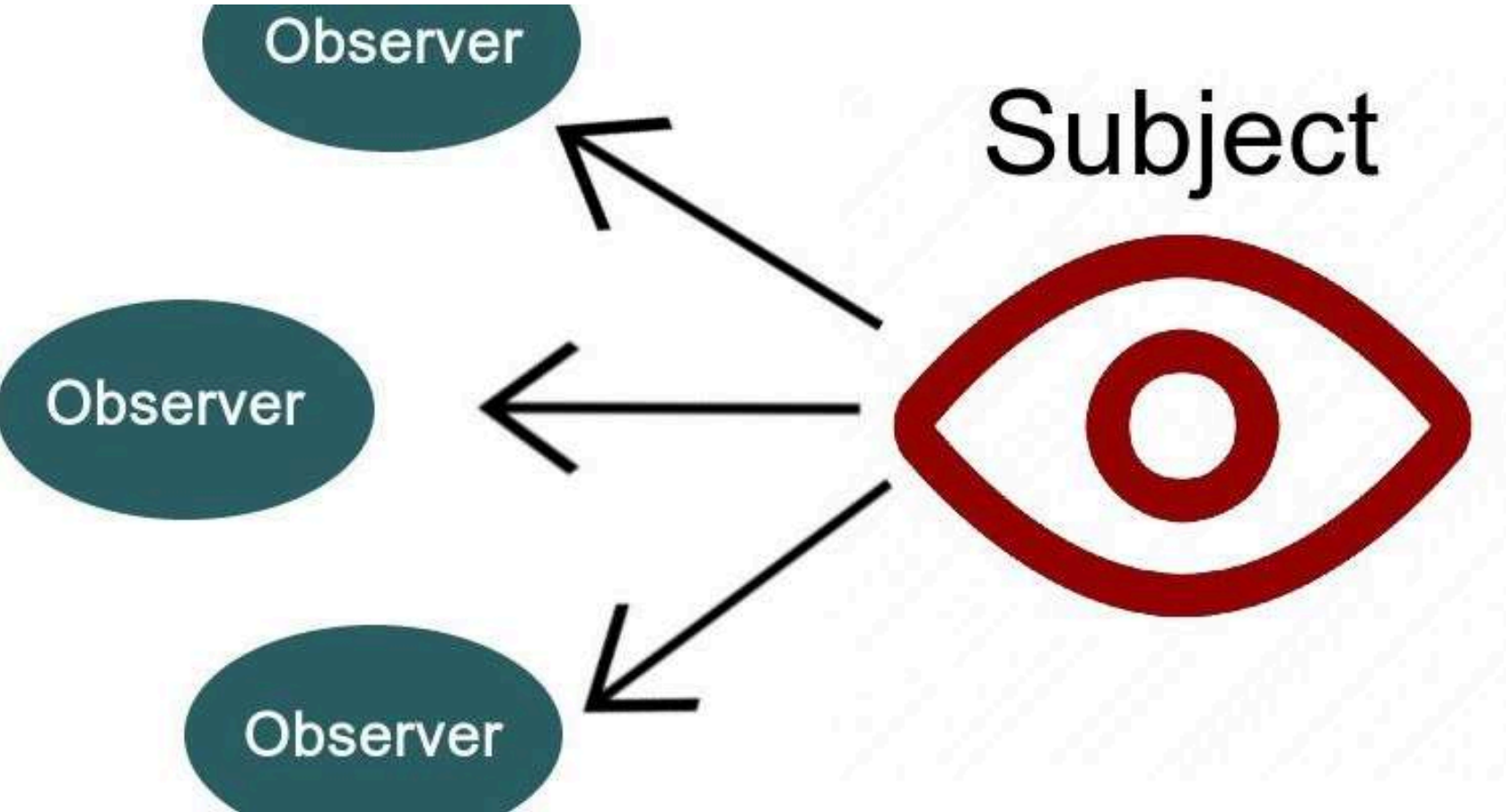
Що таке патерн проектування?

Патерн проектування (Design Pattern) — це узагальнене рішення для часто виникаючої проблеми у розробці програмного забезпечення. Це не конкретний код, який можна просто скопіювати, а радше концепція або підхід, що допомагає будувати архітектурно правильні та гнучкі рішення. Патерни дозволяють зробити код більш зрозумілим, масштабованим і підтримуваним.

Категорії

Патерни поділяються на три основні категорії:

1. Породжувальні (створення об'єктів, наприклад, Singleton, Factory Method).
2. Структурні (організація об'єктів, наприклад, Adapter, Decorator).
3. Поведінкові (взаємодія між об'єктами, наприклад, Observer, Strategy, Command).



Патерн Observer

Observer (Спостерігач) — це поведінковий патерн, який визначає залежність «один-до-багатьох» між об'єктами. Якщо один об'єкт змінює свій стан, всі залежні об'єкти автоматично отримують сповіщення та можуть відповідно відреагувати.

Цей патерн корисний, коли необхідно забезпечити автоматичну синхронізацію між частинами системи. Він широко використовується в реалізації подієвої моделі, наприклад, у графічних інтерфейсах, підписках на оновлення даних, системах сповіщень тощо.

Структура патерна

Observer складається з таких основних компонентів:

- Subject (Суб'єкт/Об'єкт спостереження) — зберігає список підписників (спостерігачів) і забезпечує їх сповіщення про зміни.
- Observer (Спостерігач) — інтерфейс або абстрактний клас, що містить метод оновлення, який викликається при зміні стану суб'єкта.
- ConcreteObserver (Конкретний спостерігач) — конкретний клас, що реалізує інтерфейс Observer і виконує певну дію при отриманні сповіщення.

Приклад реалізації на Python

```
from abc import ABC, abstractmethod
```

```
# Інтерфейс спостерігача
```

```
class Observer(ABC):
```

```
    @abstractmethod
```

```
    def update(self, message: str):
```

```
        pass
```

```
# Конкретний спостерігач
```

```
class ConcreteObserver(Observer):
```

```
    def __init__(self, name):
```

```
        self.name = name
```

```
    def update(self, message: str):
```

```
        print(f"{self.name} отримав повідомлення:  
{message}")
```

```
# Суб'єкт (Об'єкт спостереження)
```

```
class Subject:
```

```
    def __init__(self):
```

```
        self._observers = []
```

```
    def add_observer(self, observer: Observer):
```

```
        self._observers.append(observer)
```

```
    def remove_observer(self, observer: Observer):
```

```
        self._observers.remove(observer)
```

```
    def notify_observers(self, message: str):
```

```
        for observer in self._observers:
```

```
            observer.update(message)
```

```
# Приклад використання
```

```
subject = Subject()
```

```
observer1 = ConcreteObserver("Спостерігач 1")
```

```
observer2 = ConcreteObserver("Спостерігач 2")
```

```
subject.add_observer(observer1)
```

```
subject.add_observer(observer2)
```

```
subject.notify_observers("Дані оновлені!")
```


Event listeners

Розглянемо патерн Observer детальніше на прикладі JavaScript та його системи обробки подій — event listeners, які чудово ілюструють цей патерн у дії. GUI-фреймворки (Qt, React, JavaFX) — підписка компонентів на зміни стану.

Є форма з кнопкою "Зберегти". Коли користувач натискає кнопку:

1. Дані форми зберігаються.
2. Форма закривається.
3. Наприклад, з'являється повідомлення "Успішно збережено".

Усі ці дії — спостерігачі, які реагують на одну подію — клік по кнопці. Це і є патерн Observer у дії.

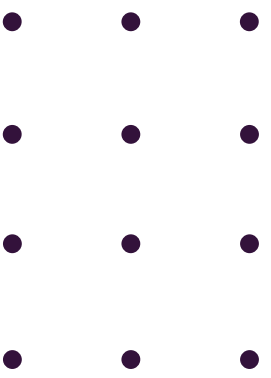


Як це працює в термінах патерну Observer

JS Event Listeners — це типовий приклад патерну Observer:

- Елемент (кнопка) є джерелом подій (Subject).
- Підписники на події (addEventListener) — це спостерігачі (Observers).
- Вони автоматично реагують на зміну стану (натискання кнопки), не знаючи один про одного — що і забезпечує гнучкість і незалежність компонентів у системі.

Реалізація власного Observer-патерна

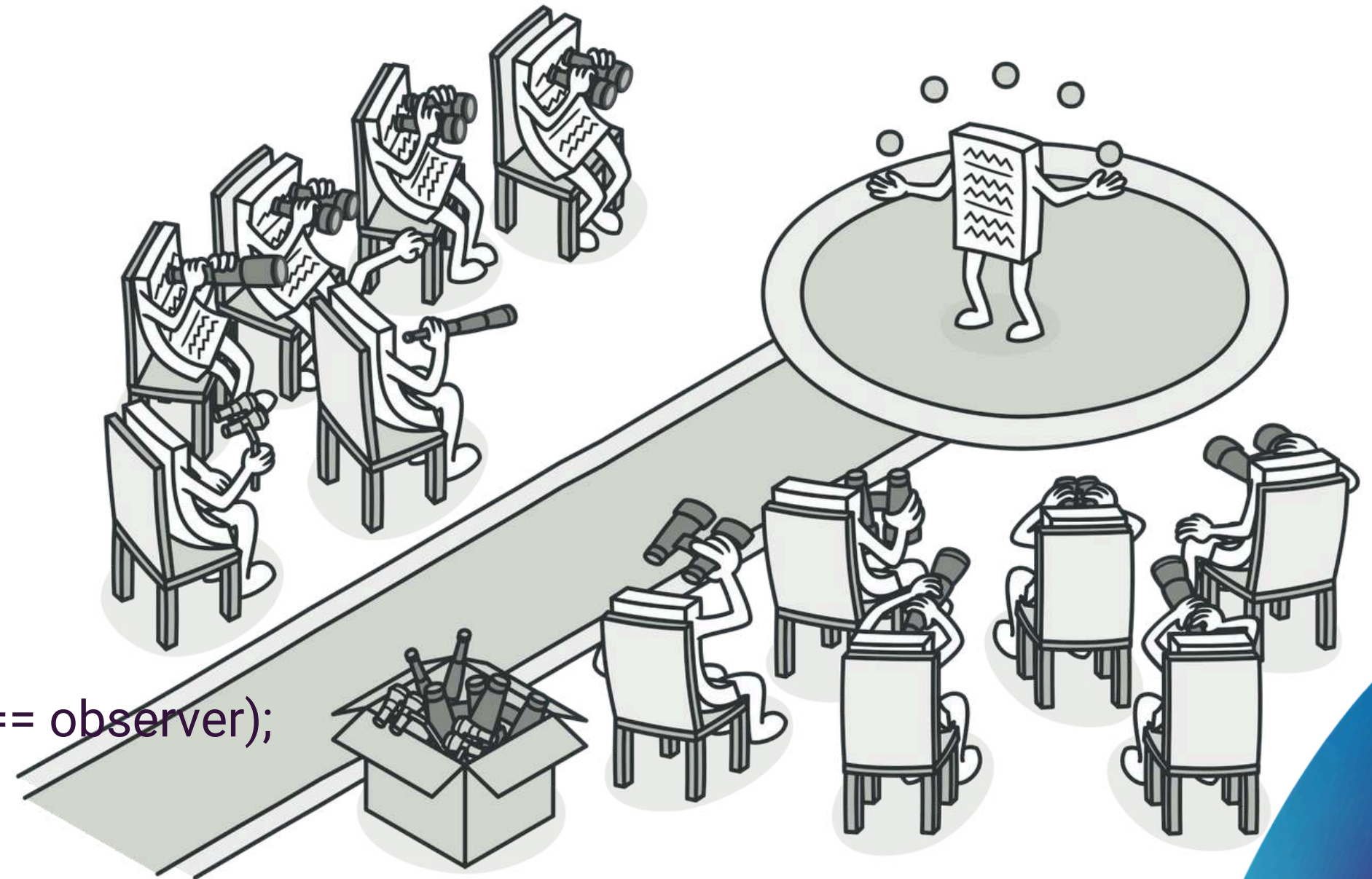


```
// === Subject (Об'єкт спостереження) ===  
class EventManager {  
  constructor() {  
    this.observers = [];  
  }
```

```
// Додати спостерігача  
subscribe(observer) {  
  this.observers.push(observer);  
}
```

```
// Відписатися  
unsubscribe(observer) {  
  this.observers = this.observers.filter(sub => sub !== observer);  
}
```

```
// Сповістити всіх спостерігачів  
notify(data) {  
  this.observers.forEach(observer => observer(data));  
}
```



Де використовується Observer?

Де використовується Observer?

- GUI-фреймворки (Qt, React, JavaFX) — підписка компонентів на зміни стану.
- Системи публікації-підписки (Pub-Sub), наприклад, у мікросервісних архітектурах.
- Реактивне програмування (RxJS, Reactor, Kotlin Flow).
- Подієві системи (Event Listeners у JavaScript, C# Events, Java Listeners).



Переваги та недоліки

✓ Плюси

- Слабке зчеплення між об'єктами.
- Полегшує масштабованість та розширюваність системи.
- Дозволяє легко додавати нових спостерігачів без змін у логіці суб'єкта.

✗ Мінуси

- Може ускладнювати відстеження залежностей у великих системах.
- Ризик витоку пам'яті, якщо спостерігачі не видаляються після використання.

Висновок

Патерн Observer є потужним інструментом для побудови гнучких і підтримуваних програмних систем. Його застосування особливо корисне там, де потрібно підтримувати залежності між об'єктами без жорсткого зв'язування.

