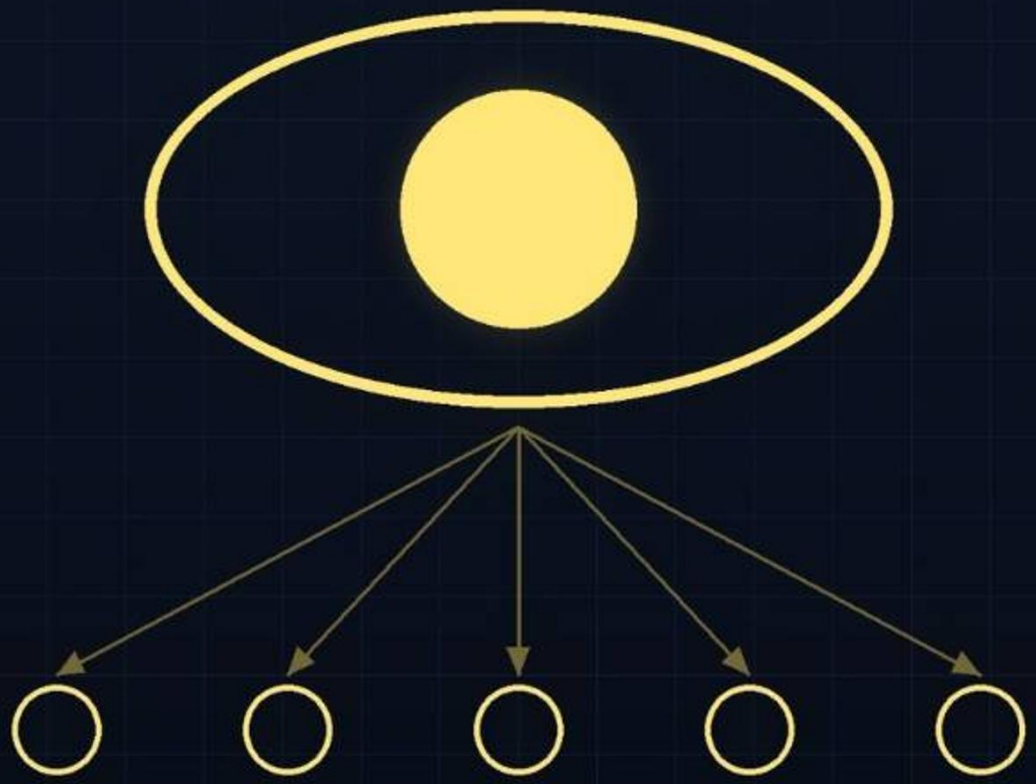
GoF · 19/23 ПОВЕДЕНЧЕСКИЙ

Python C++23 Rust Java



Наблюдатель

Observer

Алексей Примак

Алексей Примак

Наблюдатель. Паттерн GoF на Python, C++23, Rust и Java

«Автор»
2026

Примак А.

Наблюдатель. Паттерн GoF на Python, C++23, Rust и Java /
А. Примак — «Автор», 2026

Как уведомлять множество компонентов об изменении состояния, не заставляя источник событий знать обо всех получателях? «Наблюдатель» показывает решение на примерах обновления графиков, портфеля, интерфейса и уведомлений. Паттерн GoF разобран от общей схемы до реализаций на Python, C++23, Rust и Java. В книге рассматриваются подписка, доставка событий, варианты архитектуры, модификации и самостоятельные задания.

© Примак А., 2026
© Автор, 2026

Содержание

Введение 5
Как читать 6
Глава 19. Реализация на Python 7
19.1. Зачем нужен наблюдатель 7
19.2. Участники и схема 8
19.3. Классический пример: биржевой курс 9
Конец ознакомительного фрагмента. 12

Наблюдатель. Паттерн GoF на Python, C++23, Rust и Java

Введение

Эта книга посвящена одному паттерну проектирования — **Наблюдатель** (Observer). Он относится к каталогу «Банды четырёх» (Gang of Four, GoF). Здесь паттерн разобран сразу в четырёх реализациях: на Python, C++23, Rust и Java. Каждая реализация — отдельная часть книги: сначала задача и общая схема, затем код, модификации, гибридные варианты и самостоятельные работы.

Как читать

Основы каждого языка здесь не повторяются: они собраны в книгах серий по соответствующему языку. Разделы каждой части нумеруются заново, поэтому ссылка «раздел 1.2» указывает на раздел той же части, где она сделана.

Глава 19. Реализация на Python

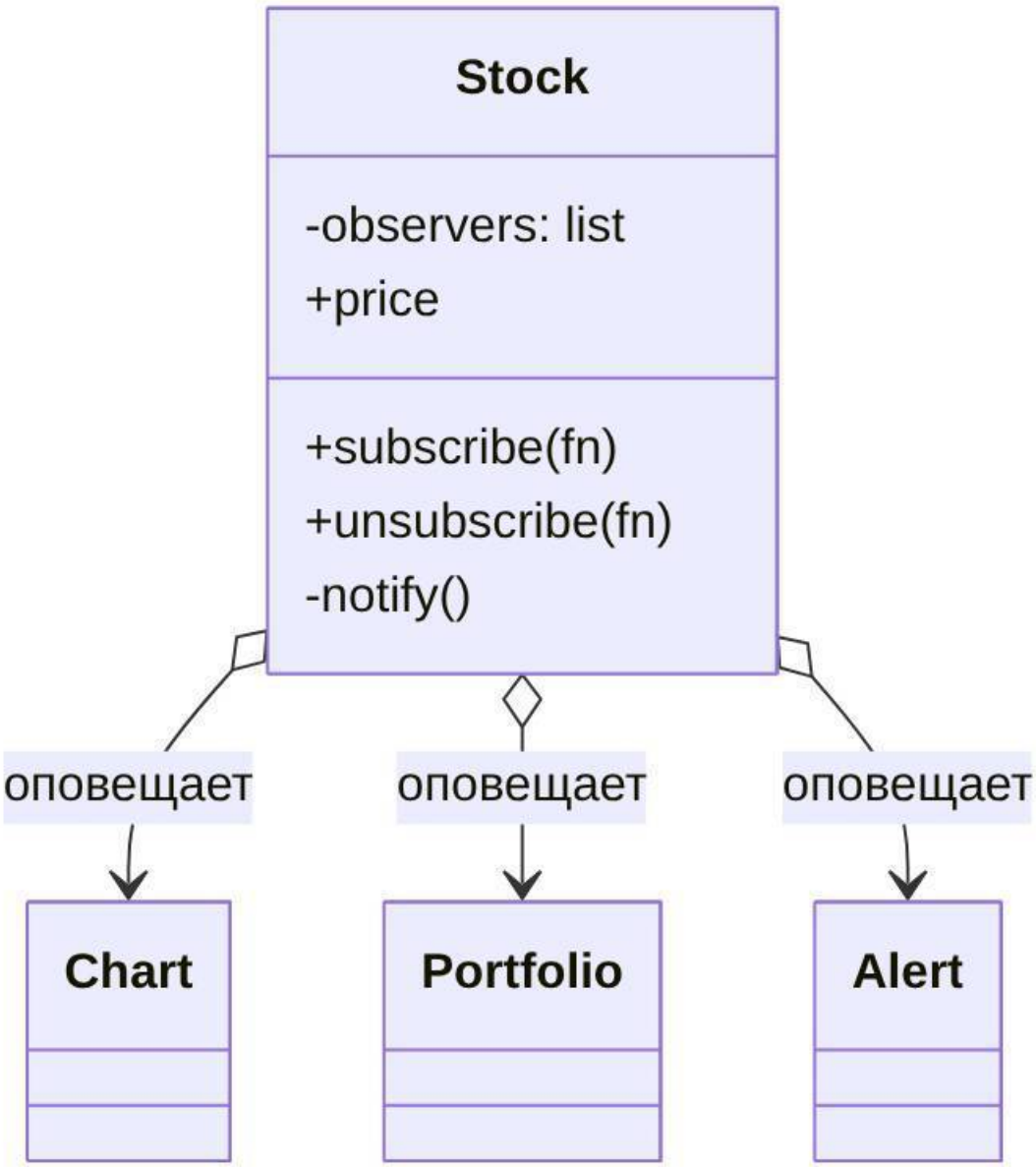
19.1. Зачем нужен наблюдатель

Курс акции изменился — нужно обновить график, пересчитать портфель и, может быть, отправить уведомление. Пользователь сменил язык — нужно перерисовать все окна. Источник событий не должен знать, кто на них реагирует: сегодня это график, завтра добавится журнал, а послезавтра — бот в мессенджере.

Наблюдатель задаёт зависимость «один ко многим»: субъект хранит список подписчиков и оповещает их при изменении своего состояния. Подписчики подключаются и отключаются во время работы, а субъект знает о них только то, что у них есть метод обновления.

Это один из самых распространённых паттернов: обработчики событий в графических интерфейсах, сигналы в Django, реактивные потоки, механизм addEventListener в браузере. В Python наблюдателем часто служит обычная функция обратного вызова.

19.2. Участники и схема



Subject— хранит подписчиков, оповещает их.
Observer— интерфейс обновления.
Concrete Observer— реагирует на изменения.

19.3. Классический пример: биржевой курс

Файл 01_stock.py:

01_stock.py — Python

```python
from typing import Callable

Observer = Callable[[str, float, float], None]


class Stock:
    def __init__(self, ticker: str, price: float) -> None:
        self.ticker = ticker
        self._price = price
        self._observers: list[Observer] = []

    def subscribe(self, fn: Observer) -> Callable[[], None]:
        self._observers.append(fn)
        return lambda: self._observers.remove(fn)
            # функция отписки

    @property
    def price(self) -> float:
        return self._price

    @price.setter
    def price(self, value: float) -> None:
        old, self._price = self._price, value
        if value != old:
            for fn in list(self._observers):
                fn(self.ticker, old, value)


class Portfolio:
    def __init__(self, shares: int) -> None:
        self.shares = shares

    def __call__(self, ticker: str, old: float, new: float) ->
        None:
```

Листинг 19.1 *— Классический пример: биржевой курс*

01_stock.py (продолжение) Python

```python
        print(f"  портфель: {ticker} × {self.shares} = {new *
            self.shares:.0f} ₽")


def chart(ticker: str, old: float, new: float) -> None:
    arrow = "▲" if new > old else "▼"
    print(f"  график: {ticker} {arrow} {new}")


def make_alert(threshold: float) -> Observer:
    def alert(ticker: str, old: float, new: float) -> None:
        if old >= threshold > new:
            print(f"  ВНИМАНИЕ: {ticker} упал ниже {threshold}")
    return alert


if __name__ == "__main__":
    sber = Stock("SBER", 300)
    sber.subscribe(chart)
    stop_portfolio = sber.subscribe(Portfolio(shares=10))
    sber.subscribe(make_alert(290))
    for p in (305, 305, 288):
        print(f"цена → {p}")
        sber.price = p
    stop_portfolio()
    print("цена → 280 (портфель отписан)")
    sber.price = 280
```

```
$ python 01_stock.py
цена → 305
  график: SBER ▲ 305
  портфель: SBER × 10 = 3050 ₽
цена → 305
цена → 288
  график: SBER ▼ 288
  портфель: SBER × 10 = 2880 ₽
  ВНИМАНИЕ: SBER упал ниже 290
цена → 280 (портфель отписан)
  график: SBER ▼ 280
```

Листинг 19.1 (продолжение) *— Классический пример: биржевой курс*

Разбор.Наблюдателем может быть любой вызываемый объект: функция chart, экземпляр класса с __call__ (Portfolio) или замыкание (make_alert). Оповещение происходит в сеттере свойства: клиент просто присваивает sber.price = 288. Повторная та же цена не вызывает уведомлений. subscribe возвращает функцию отписки — удобно, не нужно хранить ссылку

на наблюдателя. Перебор идёт по копии списка (list(…)), чтобы наблюдатель мог отписаться прямо во время оповещения.

Конец ознакомительного фрагмента.

Текст предоставлен ООО «Литрес».

Прочитайте эту книгу целиком, купив полную легальную версию на Литрес.

Безопасно оплатить книгу можно банковской картой Visa, MasterCard, Maestro, со счета мобильного телефона, с платежного терминала, в салоне МТС или Связной, через PayPal, WebMoney, Яндекс.Деньги, QIWI Кошелек, бонусными картами или другим удобным Вам способом.