Java Design Patterns

Observer

Тема

Observer

Суть паттерна

Наблюдатель

Наблюдатель — это поведенческий паттерн проектирования, который создаёт механизм подписки, позволяющий одним объектам следить и реагировать на события, происходящие в других объектах.

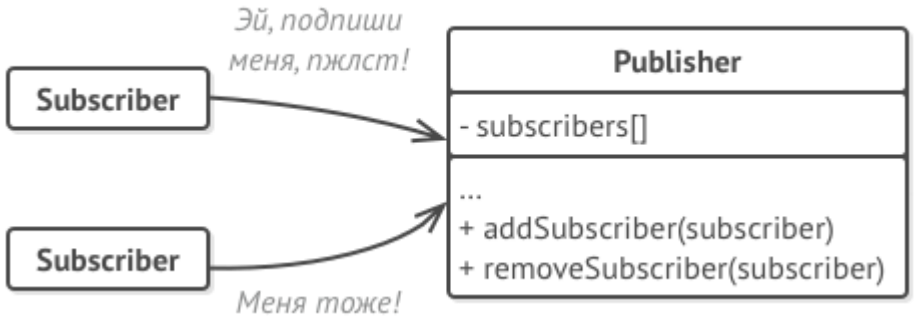
Проблема

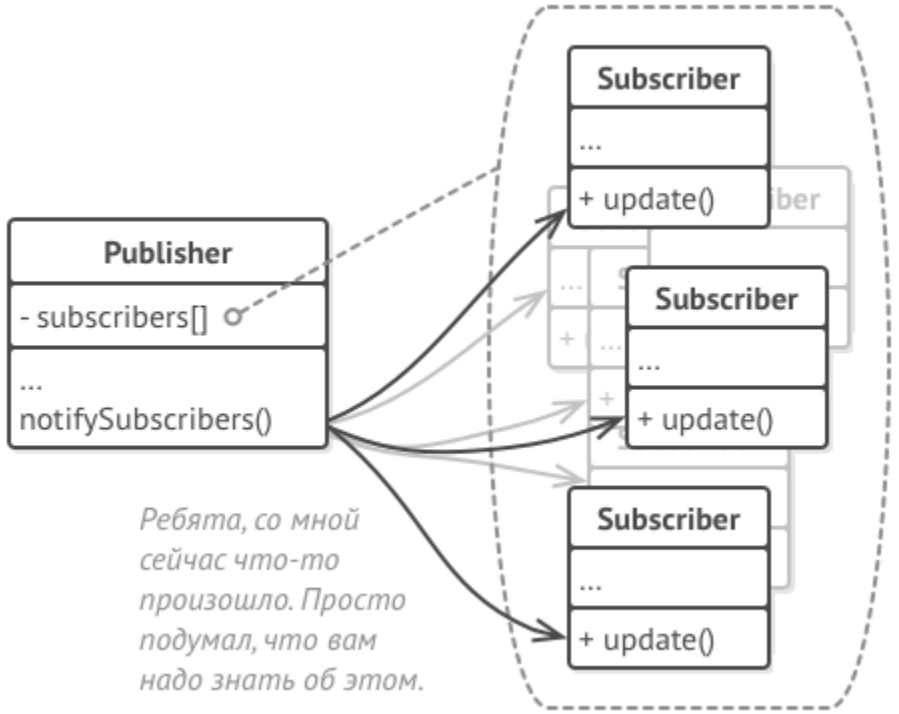
Постановка задачи

Представьте, что у вас есть два объекта: Покупатель и Магазин. В магазин вот-вот должны завезти новый товар, который интересен покупателю.
Покупатель может каждый день ходить в магазин, чтобы проверить наличие товара. Но при этом он будет тратить драгоценное время и злиться.
С другой стороны, магазин может разослать спам каждому своему покупателю. Многих это расстроит, так как товар специфический и не всем он нужен.
Получается конфликт: либо один объект работает неэффективно, тратя ресурсы на периодические проверки, либо второй объект оповещает слишком широкий круг пользователей, тоже тратя ресурсы впустую.

Решение задачи

Давайте будем называть объекты, которые содержат интересное состояние Издателями. А другие объекты, которым интересно это состояние давайте звать Подписчиками.

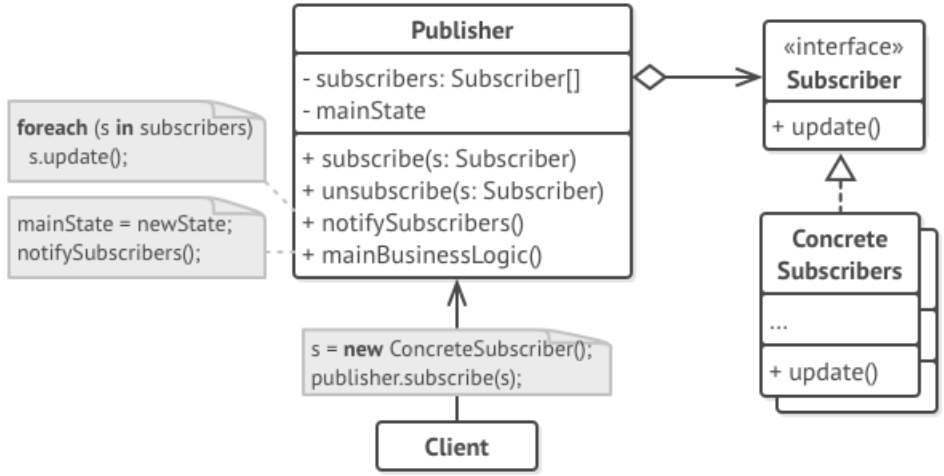




Структура

Структура паттерна

1. Издатель владеет внутренним состоянием, изменение которого интересно для подписчиков. Он содержит механизм подписки — список подписчиков, а также методы подписки/отписки.
2. Когда внутреннее состояние издателя меняется, он оповещает своих подписчиков. Для этого издатель проходит по списку подписчиков и вызывает их метод оповещения, заданный в интерфейсе подписчика.
3. Подписчик определяет интерфейс, которым пользуется издатель для отправки оповещения. В большинстве случаев, для этого достаточно единственного метода.
4. Конкретные подписчики выполняют что-то в ответ на оповещение, пришедшее от издателя.
5. По приходу оповещения, подписчику нужно получить обновлённое состояние издателя
6. Клиент создаёт объекты издателей и подписчиков, а затем регистрирует подписчиков на обновления в издателях.



Применимость

Применение паттерна

1. Когда при изменении состояния одного объекта требуется что-то сделать в других, но вы не знаете наперёд какие именно объекты должны отреагировать.
2. Когда одни объекты должны наблюдать за другими, но только в определённых случаях.

Шаги реализации

Алгоритм реализации паттерна

1. Разбейте вашу функциональность на две части: независимое ядро и опциональные зависимые части. Независимое ядро станет издателем. Зависимые части станут подписчиками.
2. Создайте интерфейс подписчиков. В большинстве случаев, в нём достаточно определить единственный метод оповещения.
3. Создайте интерфейс издателей и опишите в нём операции управления подпиской. Помните, что издатель должен работать только с общим интерфейсом подписчиков.
4. Вам нужно решить, куда поместить код ведения подписки, ведь он обычно бывает одинаков для всех типов издателей. Самый очевидный способ — вынести этот код в промежуточный абстрактный класс, от которого будут наследоваться все издатели.
5. Создайте классы конкретных издателей. Реализуйте их так, чтобы при каждом изменении состояния, они слали оповещения всем своим подписчикам.
6. Реализуйте метод оповещения в конкретных подписчиках. Издатель может отправлять какие-то данные вместе с оповещением (например, в параметрах). Возможен и другой вариант, когда подписчик, получив оповещение, сам берёт из объекта издателя нужные данные. Но при этом подписчик привяжет себя к конкретному классу издателя.
7. Клиент должен создавать необходимое количество объектов подписчиков и подписывать их у издателей.

Преимущества и недостатки

Плюсы и недостатки

Плюсы:

- Издатель не зависит от конкретных классов подписчиков.
- Вы можете подписывать и отписывать получателей на лету.
- Реализует *принцип открытости/закрытости*.

Минусы:

- Наблюдатели оповещаются в случайном порядке.

Отношение с другими паттернами

- Цепочка обязанностей, Команда, Посредник и Наблюдатель показывают различные способы работы отправителей запросов с их получателями:
  - *Цепочка обязанностей* передаёт запрос последовательно через цепочку потенциальных получателей, ожидая, что какой-то из них обработает запрос.
  - *Команда* устанавливает косвенную одностороннюю связь от отправителей к получателям.
  - *Посредник* убирает прямую связь между отправителями и получателями, заставляя их общаться опосредованно, через себя.
  - *Наблюдатель* передаёт запрос одновременно всем заинтересованным получателям, но позволяет им динамически подписывать или отписываться от таких оповещений.
- Разница между Посредником и Наблюдателем не всегда очевидна. Чаще всего они выступают как конкуренты, но иногда могут работать вместе.

Информационный видеосервис для разработчиков программного обеспечения

ITVDN

IT VIDEO DEVELOPERS NETWORK

TestProvider PROMETRIC TEST CENTER CyberBionic systematics Microsoft Partner Silver Learning Windows Azure Cloud Partner EBA European Business Association