**Поведенческие паттерны:** Эти паттерны решают задачи эффективного и безопасного взаимодействия между объектами программы.

**Наблюдатель** — это поведенческий паттерн, который позволяет объектам оповещать другие объекты об изменениях своего состояния.

При этом наблюдатели могут свободно подписываться и отписываться от этих оповещений.

Представьте, что вы имеете 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 злиться, без толку тратя своё драгоценное время.

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 покупатель тратит время на периодические проверки, либо магазин тратит ресурсы на бесполезные оповещения.

Давайте называть Издателями те объекты, которые содержат важное или интересное для других состояние. Остальные объекты, которые хотят отслеживать изменения этого состояния, назовём Подписчиками.

Паттерн Наблюдатель предлагает хранить внутри объекта издателя список ссылок на объекты подписчиков, причём издатель не должен вести список подписки самостоятельно. Он предоставит методы, с помощью которых подписчики могли бы добавлять или убирать себя из списка.

Теперь самое интересное. Когда в издателе будет происходить важное событие, он будет проходиться по списку подписчиков и оповещать их об этом, вызывая определённый метод объектов-подписчиков.

Издателю безразлично, какой класс будет иметь тот или иной подписчик, так как все они должны следовать общему интерфейсу и иметь единый метод оповещения.

После того как вы оформили подписку на газету или журнал, вам больше не нужно ездить в супермаркет и проверять, не вышел ли очередной номер. Вместо этого издательство будет присылать новые номера по почте прямо к вам домой сразу после их выхода.

Издательство ведёт список подписчиков и знает, кому какой журнал высылать. Вы можете в любой момент отказаться от подписки, и журнал перестанет вам приходить.

1. **Издатель** владеет внутренним состоянием, изменение которого интересно отслеживать подписчикам. Издатель содержит механизм подписки: список подписчиков и 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 общем интерфейсе подписчиков.
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. Эти классы должны следовать общему интерфейсу подписчиков, чтобы издатель не зависел от конкретных классов подписчиков.
5. По приходу оповещения подписчику нужно получить обновлённое состояние издателя. Издатель может передать это состояние через параметры метода оповещения. Более гибкий вариант — передавать через параметры весь объект издателя, чтобы подписчик мог сам получить требуемые данные. Как вариант, подписчик может постоянно хранить ссылку на объект издателя, переданный ему в конструкторе.
6. **Клиент** создаёт объекты издателей и подписчиков, а затем регистрирует подписчиков на обновления в издателях.

* Описанная проблема может возникнуть при разработке библиотек пользовательского интерфейса, когда вам надо дать возможность сторонним классам реагировать на клики по кнопкам.

Паттерн Наблюдатель позволяет любому объекту с интерфейсом подписчика зарегистрироваться на получение оповещений о событиях, происходящих в объектах-издателях.

* Издатели ведут динамические списки. Все наблюдатели могут подписываться или отписываться от получения оповещений прямо во время выполнения программы.

**Пример**

Этот пример показывает структуру паттерна Наблюдатель, а именно — из каких классов он состоит, какие роли эти классы выполняют и как они взаимодействуют друг с другом.

1. Разбейте вашу функциональность на две части: независимое ядро и опциональные зависимые части. Независимое ядро станет издателем. Зависимые части станут подписчиками.
2. Создайте интерфейс издателей и опишите в нём операции управления подпиской. Помните, что издатель должен работать только с общим интерфейсом подписчиков.
3. Создайте классы конкретных издателей. Реализуйте их так, чтобы после каждого изменения состояния они отправляли оповещения всем своим подписчикам.
4. Создайте интерфейс подписчиков. Обычно в нём достаточно определить единственный метод оповещения.
5. Реализуйте метод оповещения в конкретных подписчиках. Не забудьте предусмотреть параметры, через которые издатель мог бы отправлять какие-то данные, связанные с происшедшим событием.
6. Клиент должен создавать необходимое количество объектов подписчиков и подписывать их у издателей.

Разница между *Посредником и Наблюдателем* не всегда очевидна. Чаще всего они выступают как конкуренты, но иногда могут работать вместе.

Цель Посредника — убрать обоюдные зависимости между компонентами системы. Вместо этого они становятся зависимыми от самого посредника. С другой стороны, цель Наблюдателя — обеспечить динамическую одностороннюю связь, в которой одни объекты косвенно зависят от других.