**Суть паттерна**

**Фасад** — это структурный паттерн проектирования, который предоставляет простой интерфейс к сложной системе классов, т.е. он скрывает сложность системы, сводя все возможные выводы к 1 объекту.

## Задача

Упростить понимание кода, содержащего множество классов.   
Минимизировать зависимость подсистемы некоторой сложной системы и обмен информации между ними.  
Использовать только часть возможностей какой-либо сложной библиотеки.

**Аналогия из жизни**

Когда вы звоните в магазин и делаете заказ по телефону, сотрудник службы поддержки является вашим фасадом ко всем службам и отделам магазина. Он предоставляет вам упрощённый интерфейс к системе создания заказа, платёжной системе и отделу доставки.

## Структура

1. **Фасад** предоставляет быстрый доступ к определённой функциональности подсистемы. Он «знает», каким классам нужно переадресовать запрос, и какие данные для этого нужны.
2. **Дополнительный фасад** можно ввести, чтобы не «захламлять» единственный фасад разнородной функциональностью. Он может использоваться как клиентом, так и другими фасадами.
3. **Сложная подсистема** состоит из множества разнообразных классов. Для того, чтобы заставить их что-то делать, нужно знать подробности устройства подсистемы, порядок инициализации объектов и так далее. Классы подсистемы не знают о существовании фасада и работают друг с другом напрямую.
4. **Клиент** использует фасад вместо прямой работы с объектами сложной подсистемы.

## Применимость

**Когда вам нужно представить простой или урезанный интерфейс к сложной подсистеме.**

 Часто подсистемы усложняются по мере развития программы. Применение большинства паттернов приводит к появлению меньших классов, но в бóльшем количестве. Такую подсистему проще повторно использовать, настраивая её каждый раз под конкретные нужды, но вместе с тем, применять подсистему без настройки становится труднее. Фасад предлагает определённый вид системы по умолчанию, устраивающий большинство клиентов.

**Когда вы хотите разложить подсистему на отдельные слои.**

 Используйте фасады для определения точек входа на каждый уровень подсистемы. Если подсистемы зависят друг от друга, то зависимость можно упростить, разрешив подсистемам обмениваться информацией только через фасады.

Например, возьмём ту же сложную систему видеоконвертации. Вы хотите разбить её на слои работы с аудио и видео. Для каждой из этих частей можно попытаться создать фасад и заставить классы аудио и видео обработки общаться друг с другом через эти фасады, а не напрямую.

**Шаги реализации**

1. Определите, можно ли создать более простой интерфейс, чем тот, который предоставляет сложная подсистема. Вы на правильном пути, если этот интерфейс избавит клиента от необходимости знать о подробностях подсистемы.
2. Создайте класс фасада, реализующий этот интерфейс. Он должен переадресовывать вызовы клиента нужным объектам подсистемы. Фасад должен будет позаботиться о том, чтобы правильно инициализировать объекты подсистемы.
3. Вы получите максимум пользы, если клиент будет работать только с фасадом. В этом случае изменения в подсистеме будут затрагивать только код фасада, а клиентский код останется рабочим.
4. Если ответственность фасада [**начинает размываться**](https://refactoring.guru/ru/smells/large-class), подумайте о введении дополнительных фасадов.

**Преимущества и недостатки**

* Изолирует клиентов от компонентов сложной подсистемы.
* Фасад рискует стать [**божественным объектом**](https://refactoring.guru/ru/antipatterns/god-object), привязанным ко всем классам программы.