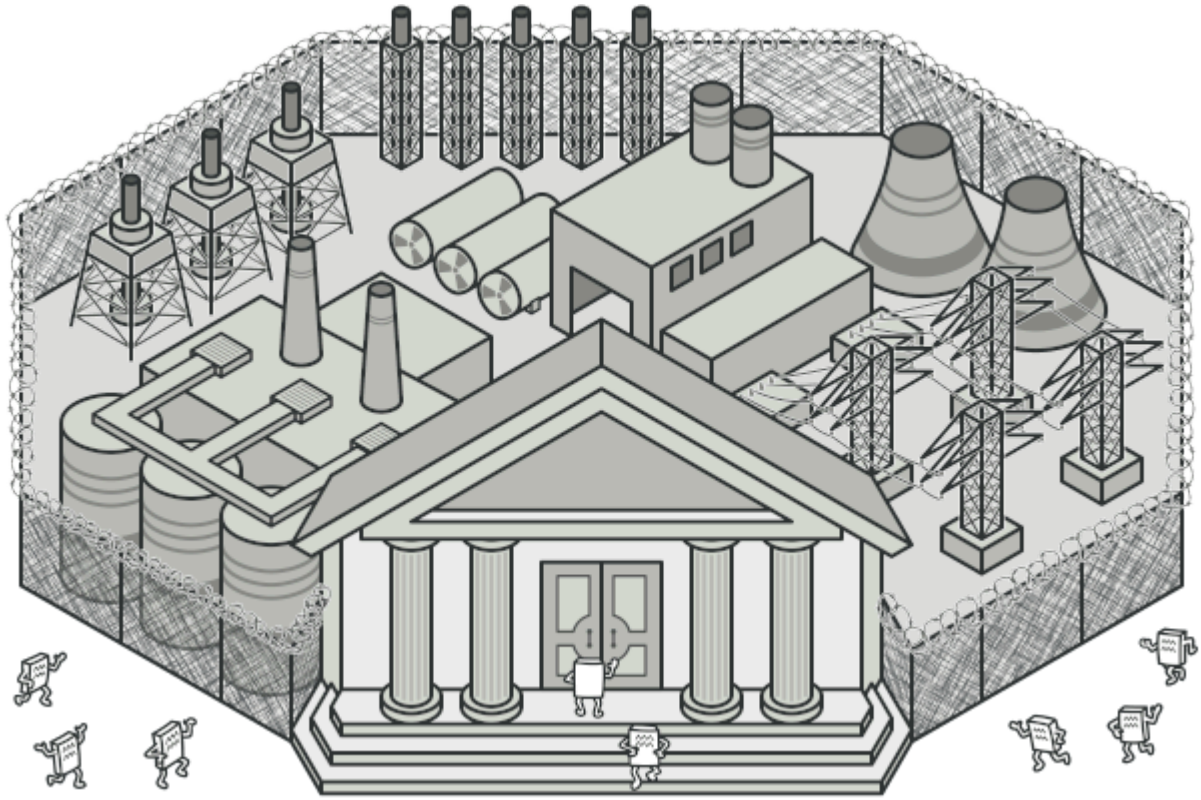
Фасад

Также известен как: Facade

Суть паттерна

Фасад — это структурный паттерн проектирования, который предоставляет простой интерфейс к сложной системе классов, библиотеке или фреймворку.



Проблема

Вашему коду приходится работать с большим количеством объектов некой сложной библиотеки или фреймворка. Вы должны самостоятельно инициализировать эти объекты, следить за правильным порядком зависимостей и так далее.

сторонних классов. Такой код довольно сложно понимать и поддерживать.

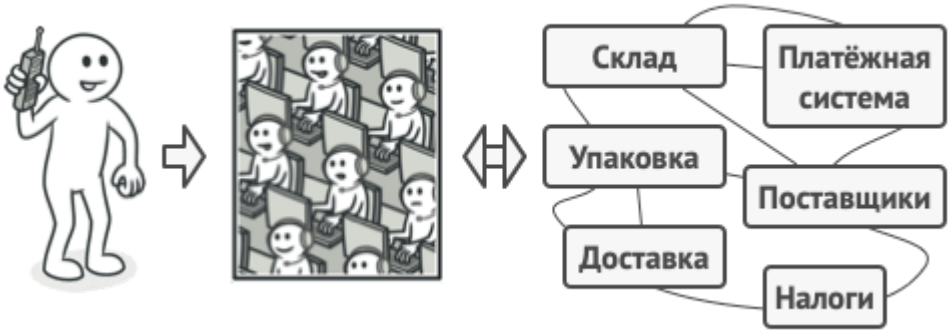
😊 Решение

Фасад — это простой интерфейс для работы со сложной подсистемой, содержащей множество классов. Фасад может иметь урезанный интерфейс, не имеющий 100% функциональности, которой можно достичь, используя сложную подсистему напрямую. Но он предоставляет именно те фичи, которые нужны клиенту, и скрывает все остальные.

Фасад полезен, если вы используете какую-то сложную библиотеку со множеством подвижных частей, но вам нужна только часть её возможностей.

К примеру, программа, заливающая видео котиков в социальные сети, может использовать профессиональную библиотеку сжатия видео. Но все, что нужно клиентскому коду этой программы — простой метод `encode(filename, format)`. Создав класс с таким методом, вы реализуете свой первый фасад.

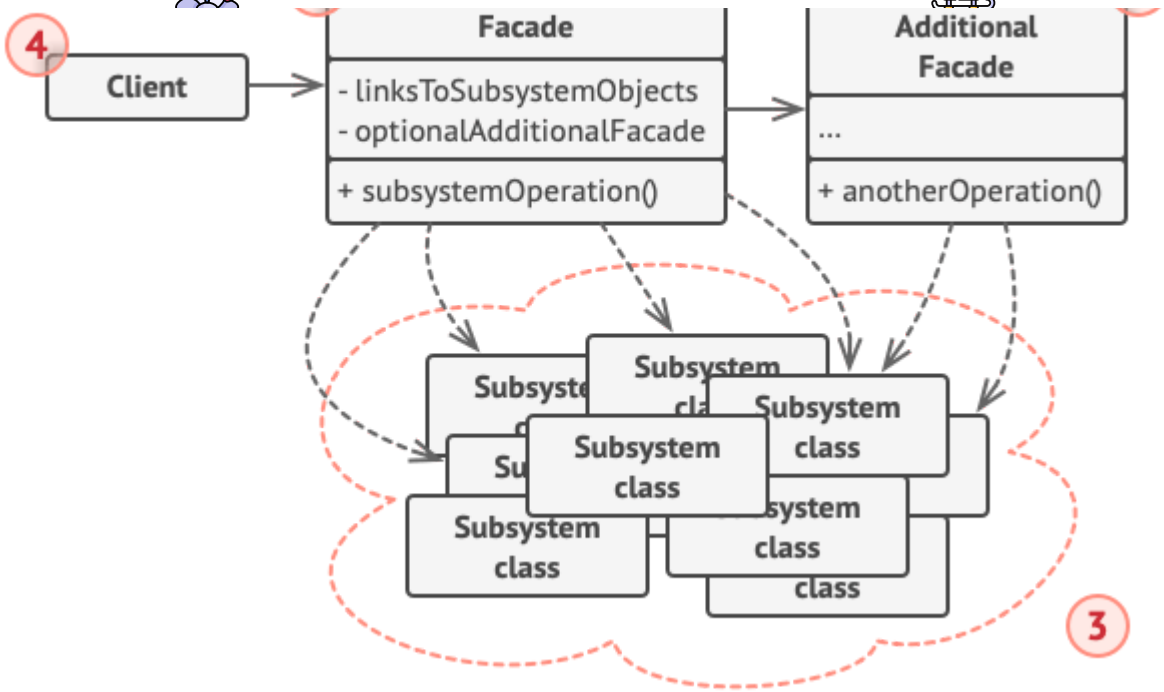
🚗 Аналогия из жизни



Пример телефонного заказа.

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.

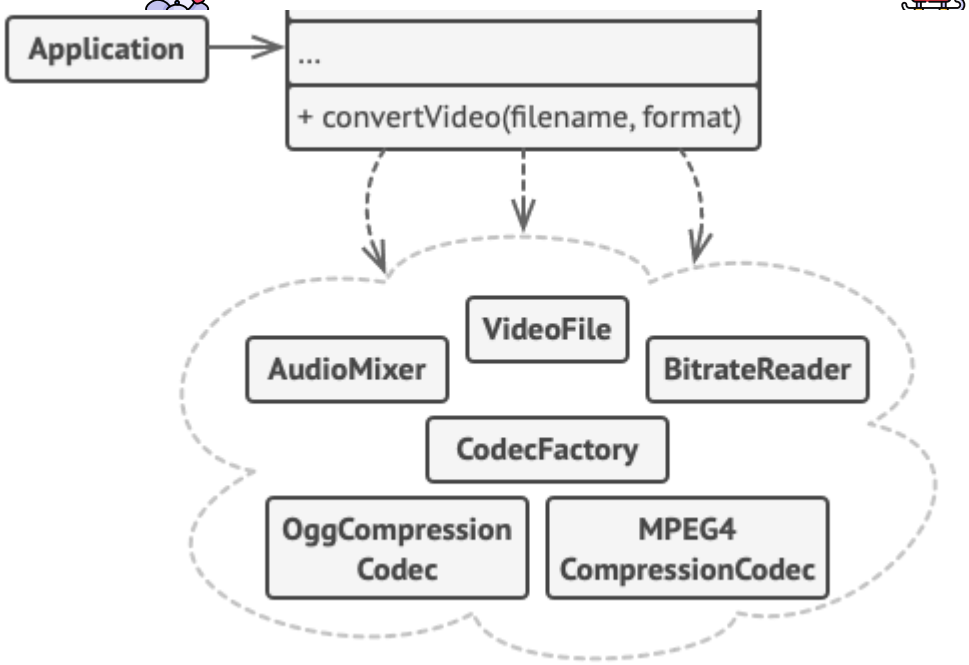
3. **Сложная подсистема** состоит из множества разнообразных классов. Для того, чтобы заставить их что-то делать, нужно знать подробности устройства подсистемы, порядок инициализации объектов и так далее.

   Классы подсистемы не знают о существовании фасада и работают друг с другом напрямую.

4. **Клиент** использует фасад вместо прямой работы с объектами сложной подсистемы.

Псевдокод

В этом примере **Фасад** упрощает работу со сложным фреймворком видеоконвертации.



Пример изоляции множества зависимостей в одном фасаде.

Вместо непосредственной работы с дюжиной классов, фасад предоставляет коду приложения единственный метод для конвертации видео, который сам заботится о том, чтобы правильно сконфигурировать нужные объекты фреймворка и получить требуемый результат.

```
// Классы сложного стороннего фреймворка конвертации видео. Мы
// не контролируем этот код, поэтому не можем его упростить.

class VideoFile
// ...

class OggCompressionCodec
// ...

class MPEG4CompressionCodec
// ...

class CodecFactory
// ...

class BitrateReader
// ...

class AudioMixer
// ...
```

```
// фреймворка, но зато скрывает его сложность от клиентов.
class VideoConverter is
    method convert(filename, format):File is
        file = new VideoFile(filename)
        sourceCodec = (new CodecFactory).extract(file)
        if (format == "mp4")
            destinationCodec = new MPEG4CompressionCodec()
        else
            destinationCodec = new OggCompressionCodec()
        buffer = BitrateReader.read(filename, sourceCodec)
        result = BitrateReader.convert(buffer, destinationCodec)
        result = (new AudioMixer()).fix(result)
        return new File(result)

// Приложение не зависит от сложного фреймворка конвертации
// видео. Кстати, если вы вдруг решите сменить фреймворк, вам
// нужно будет переписать только класс фасада.
class Application is
    method main() is
        convertor = new VideoConverter()
        mp4 = convertor.convert("funny-cats-video.ogg", "mp4")
        mp4.save()
```

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

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**, подумайте о введении дополнительных фасадов.

Преимущества и недостатки

✓ Изолирует клиентов от компонентов сложной подсистемы.

✗ Фасад рискует стать **божественным объектом**, привязанным ко всем классам программы.

Отношения с другими паттернами

- **Фасад** задаёт новый интерфейс, тогда как **Адаптер** повторно использует старый. *Адаптер* оборачивает только один класс, а *Фасад* оборачивает целую подсистему. Кроме того, *Адаптер* позволяет двум существующим интерфейсам работать сообща, вместо того, чтобы задать полностью новый.

- **Абстрактная фабрика** может быть использована вместо **Фасада** для того, чтобы скрыть платформо-зависимые классы.

- **Легковес** показывает, как создавать много мелких объектов, а **Фасад** показывает, как создать один объект, который отображает целую подсистему.

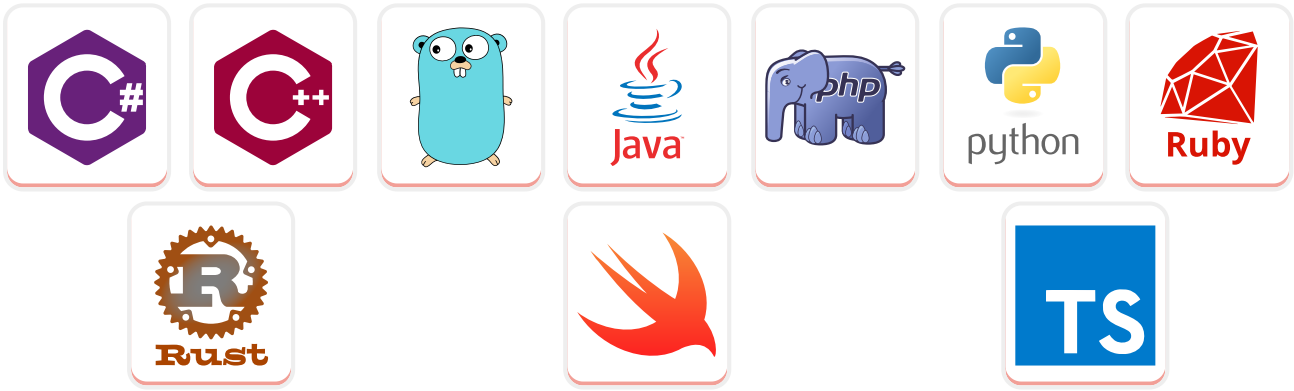
существующих классов.

  - *Фасад* создаёт упрощённый интерфейс к подсистеме, не внося в неё никакой добавочной функциональности. Сама подсистема не знает о существовании *Фасада*. Классы подсистемы общаются друг с другом напрямую.
  - *Посредник* централизует общение между компонентами системы. Компоненты системы знают только о существовании *Посредника*, у них нет прямого доступа к другим компонентам.

- **Фасад** можно сделать **Одиночкой**, так как обычно нужен только один объект-фасад.

- **Фасад** похож на **Заместитель** тем, что замещает сложную подсистему и может сам её инициализировать. Но в отличие от *Фасада*, *Заместитель* имеет тот же интерфейс, что его служебный объект, благодаря чему их можно взаимозаменять.

</> Примеры реализации паттерна



Читай на любом девайсе

Любишь айфоны или андроид?