1. Шаблон фасад (англ. *Facade*)

Пожалуй, самый известный шаблон проектирования в Python - Фасад.

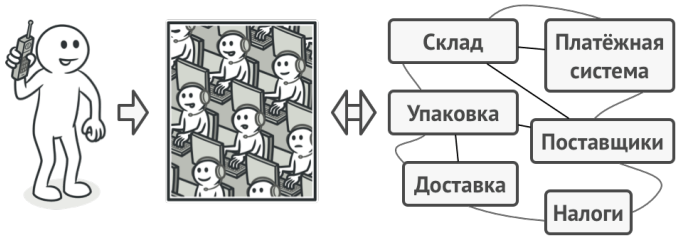
2. Шаблоны проектирования — это руководства по решению повторяющихся проблем.

3. Шаблон фасад (англ. *Facade*) — структурный шаблон проектирования, позволяющий скрыть сложность системы путём сведения всех возможных внешних вызовов к одному объекту, делегирующему их соответствующим объектам системы.

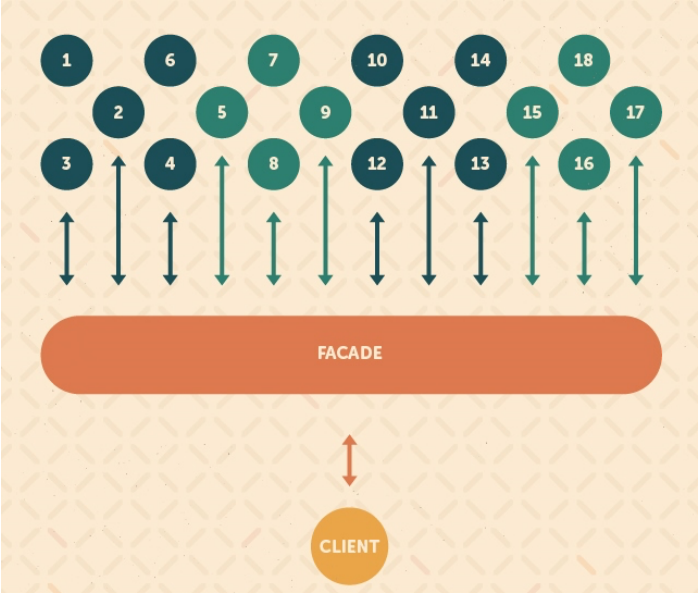
Что значит структурный?
*__Структурные шаблоны__ — шаблоны проектирования, в которых рассматривается вопрос о том, как из классов и объектов образуются более крупные структуры.
Т.е. занимаются объединением отдельных классов и объектов в сложные группы.

4. Пример из жизни: Как вы включаете компьютер? Нажимаем на кнопку включения, скажете вы. Это то, во что вы верите, потому что вы используете простой интерфейс, который компьютер предоставляет для доступа снаружи. Внутри же должно произойти гораздо больше вещей. Вы сами все знаете. Этот простой интерфейс для сложной подсистемы называется фасадом.

5. 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.



Простыми словами: Предоставляет упрощенный интерфейс для сложной системы.

6. Применимость: Паттерн часто встречается в клиентских приложениях, написанных на Python, которые используют классы-фасады для упрощения работы со сложными библиотеки или API.

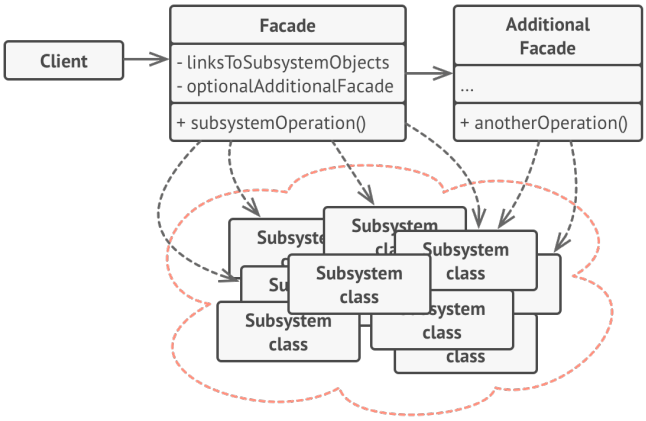
Шаблон применяется для установки некоторого рода политики по отношению к другой группе объектов. Если политика должна быть яркой и заметной, следует воспользоваться услугами шаблона Фасад. Если же необходимо обеспечить скрытность и аккуратность (прозрачность), более подходящим выбором является шаблон Заместитель (Proxy).

7. Признаки применения паттерна: Фасад угадывается в классе, который имеет простой интерфейс, но делегирует основную часть работы другим классам. Чаще всего, фасады сами следят за жизненным циклом объектов сложной системы.

Применяют тогда, когда есть проблема - проблема такая…

8. 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.
В результате бизнес-логика ваших классов тесно переплетается с деталями реализации сторонних классов. Такой код довольно сложно понимать и поддерживать.

Но всегда есть решение
9. 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. 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.

10. Структура


Фасад предоставляет быстрый доступ к определённой функциональности подсистемы. Он «знает», каким классам нужно переадресовать запрос, и какие данные для этого нужны.
Дополнительный фасад можно ввести, чтобы не «захламлять» единственный фасад разнородной функциональностью. Он может использоваться как клиентом, так и другими фасадами.
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.
Клиент использует фасад вместо прямой работы с объектами сложной подсистемы.

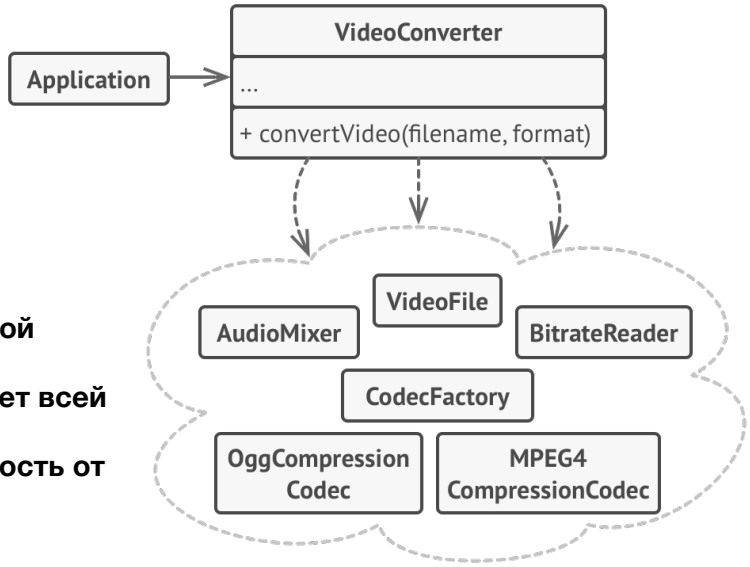
11. Пример
В этом примере **Фасад** упрощает работу со сложным фреймворком видеоконвертации.

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.

12. // Классы сложного стороннего фреймворка конвертации видео.
// Мы не можем его упростить.

```
class VideoFile
// ...

class OggCompressionCodec
// ...

class MPEG4CompressionCodec
// ...
```

```
class CodecFactory
// ...

class BitrateReader
// ...

class AudioMixer
// ...
```




```
// Вместо этого мы создаём Фасад — простой интерфейс для работы
// со сложным фреймворком. Фасад не имеет всей функциональности
// 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("youtubevideo.ogg", "mp4")
        mp4.save()
```

13. 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.
- **Посредник** и **Фасад** похожи тем, что пытаются организовать работу множества существующих классов.
- **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.

14.