212 Назаров Захар

ЭССЕ

GOF pattern

Для начала разберёмся что это такое и какие виды существуют.

Что такое паттерны проектирования?

Паттернами проектирования (Design Patterns) называют решения часто встречающихся проблем в области разработки программного обеспечения. В данном случае предполагается, что есть некоторый набор общих формализованных проблем, которые довольно часто встречаются, и паттерны предоставляют ряд принципов для решения этих проблем.

Концепцию паттернов впервые описал Кристофер Александер в книге «Язык шаблонов. Города. Здания. Строительство».

Идея показалась привлекательной авторам Эриху Гамму, Ричарду Хелму, Ральфу Джонсону и Джону Влиссидесу, их принято называть «бандой четырёх» (Gang of Four). В 1995 году они написали книгу «Design Patterns: Elements of Reusable Object-Oriented Software», в которой применили концепцию типовых паттернов в программировании. В книгу вошли 23 паттерна, решающие различные проблемы объектно-ориентированного дизайна.

Зачем знать паттерны?

Самое главная причина — паттерны упрощают проектирование и поддержку программ.

Проверенные решения.

Ваш код более предсказуем когда вы используете готовые решения, вместо повторного изобретения велосипеда.

Стандартизация кода.

Вы делаете меньше ошибок, так как используете типовые унифицированные решения, в которых давно найдены все скрытые проблемы.

Общий язык.

Вы произносите название паттерна, вместо того, чтобы час объяснять другим членам команды какой подход вы придумали и какие классы для этого нужны.

Каталог шаблонов проектирования

Порождающие паттерны:

Порождающие паттерны — это паттерны, которые абстрагируют процесс инстанцирования или, иными словами, процесс порождения классов и объектов. Среди них выделяются следующие:

Абстрактная фабрика (Abstract Factory)

Строитель (Builder)

Фабричный метод (Factory Method)

Прототип (Prototype)

Одиночка (Singleton)

Структурные паттерны:

Структурные паттерны - рассматривает, как классы и объекты образуют более крупные структуры - более

сложные по характеру классы и объекты. К таким шаблонам относятся:

Адаптер (Adapter)

Мост (Bridge)

Компоновщик (Composite)

Декоратор (Decorator)

Фасад (Facade)

Приспособленец (Flyweight)

Заместитель (Proxy)

Поведенческие паттерны:

Поведенческие паттерны - они определяют алгоритмы и взаимодействие между классами и объектами, то есть их поведение. Среди подобных шаблонов можно выделить следующие:

Цепочка обязанностей (Chain of responsibility)

Команда (Command)

Интерпретатор (Interpreter)

Итератор (Iterator)

Посредник (Mediator)

Хранитель (Memento)

Наблюдатель (Observer)

Состояние (State)

Стратегия (Strategy)

Шаблонный метод (Template method)

Но нам из этого списка интересен только:

Фасад (Facade)

Назначение:

предоставляет унифицированный интерфейс вместо набора интерфейсов некоторой подсистемы. Фасад определяет интерфейс более высокого уровня, который упрощает использование подсистемы.

Шаблон Фасад объединяет группу объектов в рамках одного специализированного интерфейса и переадресует вызовы его методов к этим объектам.

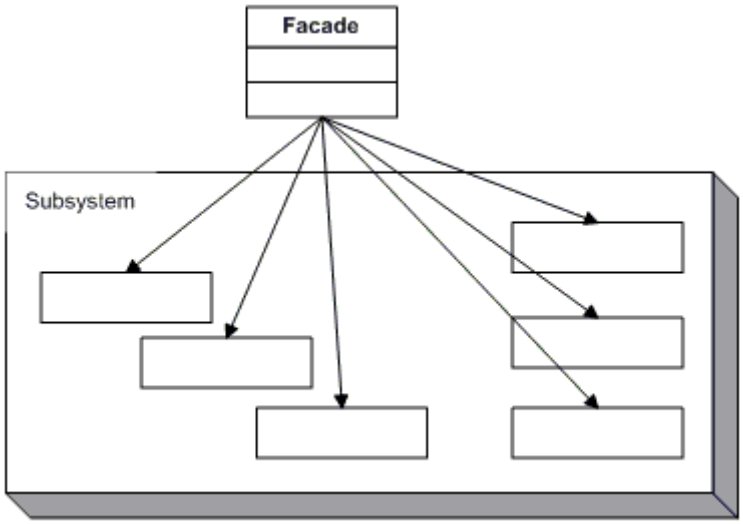
Когда использовать фасад?

- Когда имеется сложная система, и необходимо упростить с ней работу. Фасад позволит определить одну точку взаимодействия между клиентом и системой.
- Когда надо уменьшить количество зависимостей между клиентом и сложной системой. Фасадные объекты позволяют отделить, изолировать компоненты системы от клиента и развивать и работать с ними независимо.
- Когда нужно определить подсистемы компонентов в сложной системе. Создание фасадов для компонентов каждой отдельной подсистемы позволит упростить взаимодействие между ними и повысить их независимость друг от друга.

Схема примера использования паттерна "Facade" из реальной жизни:

Фасад определяет унифицированный интерфейс более высокого уровня для подсистемы, которая упрощает ее использование. Потребители сталкиваются с фасадом при заказе из каталога. Потребитель звонит на один номер и разговаривает с представителем службы поддержки клиентов. Представитель службы поддержки клиентов выступает в качестве фасада, предоставляя интерфейс отделу исполнения заказов, отделу биллинга и отделу доставки.

UML схема паттерна facade:



Участники: 1) Facade

- Знает, какие классы подсистемы отвечают за запрос.
- Делегирует запросы клиентов соответствующим объектам подсистем.

2) Классы подсистемы

- Реализовывают функционал подсистемы
- Обрабатывают методы, назначенные объектом Facade.
- Не знают о фасаде и не ссылаются на него.

Заключение:

Очень многие библиотеки или подсистемы приложений содержат встроенные фасады, которые являются высокоуровневыми классами, предназначенными для решения типовых операций. Фасады делают базовые сценарии простыми, а сложные сценарии — возможными. Если клиенту нужна лишь базовая функциональность, достаточно воспользоваться фасадом; если же его функциональности недостаточно, можно использовать более низкоуровневые классы модуля или библиотеки напрямую.

Инкапсуляция стороннего кода

Использование фасадов не только упрощает использование библиотек или сторонних компонентов, но и решает ряд насущных проблем.

Повторное использование кода и лучших практик. Многие библиотеки довольно сложные, поэтому их корректное использование требует определенных навыков. Инкапсуляция работы с ними в одном месте позволяет корректно использовать их всеми разработчиками независимо от их опыта

Переход на новую версию библиотеки. При выходе новой версии библиотеки достаточно будет протестировать лишь фасад, чтобы принять решение, стоит на нее переходить или нет.

Переход с одной библиотеки на другую. Благодаря фасаду приложение не так сильно завязано на библиотеку, так что переход на другую библиотеку потребует лишь создание еще одного фасада. А использование адаптера сделает этот переход менее болезненным.

Фасад не нужно применять, когда в повышении уровня абстракции нет никакого смысла. В большинстве случаев нет смысла в фасаде для библиотеки логирования, достаточно выбрать одно из решений и использовать его в коде приложения. Фасады бесполезны, когда имеют громоздкий интерфейс, и пользоваться ими сложнее, чем исходной библиотекой.