Паттерны проектирования

Часть I

Что такое паттерн?

Паттерн проектирования — это часто встречающееся решение определённой проблемы при проектировании архитектуры программ.

В отличие от готовых функций или библиотек, паттерн нельзя просто взять и скопировать в программу. Паттерн представляет собой не какой-то конкретный код, а общую концепцию решения той или иной проблемы, которую нужно будет ещё подстроить под нужды вашей программы.

Паттерны это не алгоритмы. И если алгоритм — это чёткий набор действий, то паттерн — это высокоуровневое описание решения, реализация которого может отличаться в двух разных программах.

Если привести аналогии, то алгоритм — это кулинарный рецепт с чёткими шагами, а паттерн — инженерный чертёж, на котором нарисовано решение, но не конкретные шаги его реализации.

Описание паттернов.

- проблема, которую решает паттерн;
- мотивации к решению проблемы способом, который предлагает паттерн;
- структуры классов, составляющих решение;
- пример на одном из языков программирования;
- особенностей реализации в различных контекстах;
- связей с другими паттернами.

Зачем знать паттерны?

Вы можете вполне успешно работать, не зная ни одного паттерна. Более того, вы могли уже не раз реализовать какой-то из паттернов, даже не подозревая об этом.

Осознанное владение инструментом как раз и отличает профессионала от любителя. Вы можете забить гвоздь молотком, а можете и дрелью, если сильно постараетесь. Но профессионал знает, что главная фишка дрели совсем не в этом. Итак, зачем же знать паттерны?

Проверенные решения. Вы тратите меньше времени, используя готовые решения, вместо повторного изобретения велосипеда. До некоторых решений вы смогли бы додуматься и сами, но многие могут быть для вас открытием.

Стандартизация кода. Вы делаете меньше просчётов при проектировании, используя типовые унифицированные решения, так как все скрытые проблемы в них уже давно найдены.

Общий программистский словарь. Вы произносите название паттерна, вместо того, чтобы час объяснять другим программистам, какой крутой дизайн вы придумали и какие классы для этого нужны.

Классификация паттернов

Самые низкоуровневые и простые паттерны — **идиомы**. Они не универсальны, поскольку применимы только в рамках одного языка программирования.

Самые универсальные — **архитектурные паттерны**, которые можно реализовать практически на любом языке. Они нужны для проектирования всей программы, а не отдельных её элементов.

Порождающие паттерны

Отвечают за удобное и безопасное создание новых объектов или даже целых семейств объектов.

Фабричный метод —определяет общий интерфейс для создания объектов в суперклассе, позволяя подклассам изменять тип создаваемых объектов.

Прототип - позволяет копировать объекты, не вдаваясь в подробности их реализации.

Абстрактная фабрика — позволяет создавать семейства связанных объектов, не привязываясь к конкретным классам создаваемых объектов.

Одиночка —гарантирует, что у класса есть только один экземпляр, и предоставляет к нему глобальную точку доступа.

Строитель — позволяет создавать сложные объекты пошагово. Строитель даёт возможность использовать один и тот же код строительства для получения разных представлений объектов.

Структурные паттерны

Отвечают за построение удобных в поддержке иерархий классов.

Адаптер - позволяет объектам с несовместимыми интерфейсами работать вместе.

Мост — разделяет один или несколько классов на две отдельные иерархии — абстракцию и реализацию, позволяя изменять их независимо друг от друга.

Компоновщик —позволяет сгруппировать множество объектов в древовидную структуру, а затем работать с ней так, как будто это единичный объект.

Декоратор —позволяет динамически добавлять объектам новую функциональность, оборачивая их в полезные «обёртки».

Фасад —предоставляет простой интерфейс к сложной системе классов, библиотеке или фреймворку.

Заместитель —позволяет подставлять вместо реальных объектов специальные объекты-заменители. Эти объекты перехватывают вызовы к оригинальному объекту, позволяя сделать что-то до или после передачи вызова оригиналу.

Легковес - позволяет вместить бóльшее количество объектов в отведённую оперативную память. Легковес экономит память, разделяя общее состояние объектов между собой, вместо хранения одинаковых данных в каждом объекте.

Поведенческие паттерны

Решают задачи эффективного и безопасного взаимодействия между объектами программы.

Посетитель —позволяет добавлять в программу новые операции, не изменяя классы объектов, над которыми эти операции могут выполняться.

Итератор —даёт возможность последовательно обходить элементы составных объектов, не раскрывая их внутреннего представления.

Снимок —позволяет сохранять и восстанавливать прошлые состояния объектов, не раскрывая подробностей их реализации.

Наблюдатель — это поведенческий паттерн проектирования, который создаёт механизм подписки, позволяющий одним объектам следить и реагировать на события, происходящие в других объектах.

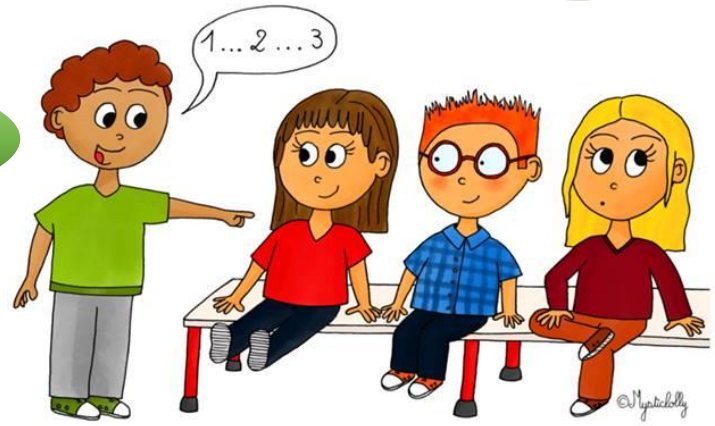
Наблюдатель

Снимок

Посетитель

Состояние

Итератор



Поведенческие паттерны

Решают задачи эффективного и безопасного взаимодействия между объектами программы.

Цепочка обязанностей —позволяет передавать запросы последовательно по цепочке обработчиков. Каждый последующий обработчик решает, может ли он обработать запрос сам и стоит ли передавать запрос дальше по цепи.

Команда —превращает запросы в объекты, позволяя передавать их как аргументы при вызове методов, ставить запросы в очередь, логировать их, а также поддерживать отмену операций.

Посредник —позволяет уменьшить связанность множества классов между собой, благодаря перемещению этих связей в один класс-посредник.

Стратегия —определяет семейство схожих алгоритмов и помещает каждый из них в собственный класс, после чего алгоритмы можно взаимозаменять прямо во время исполнения программы.

Команда

Стратегия

Цепочка обязанностей

Шаблонный метод

Посредник