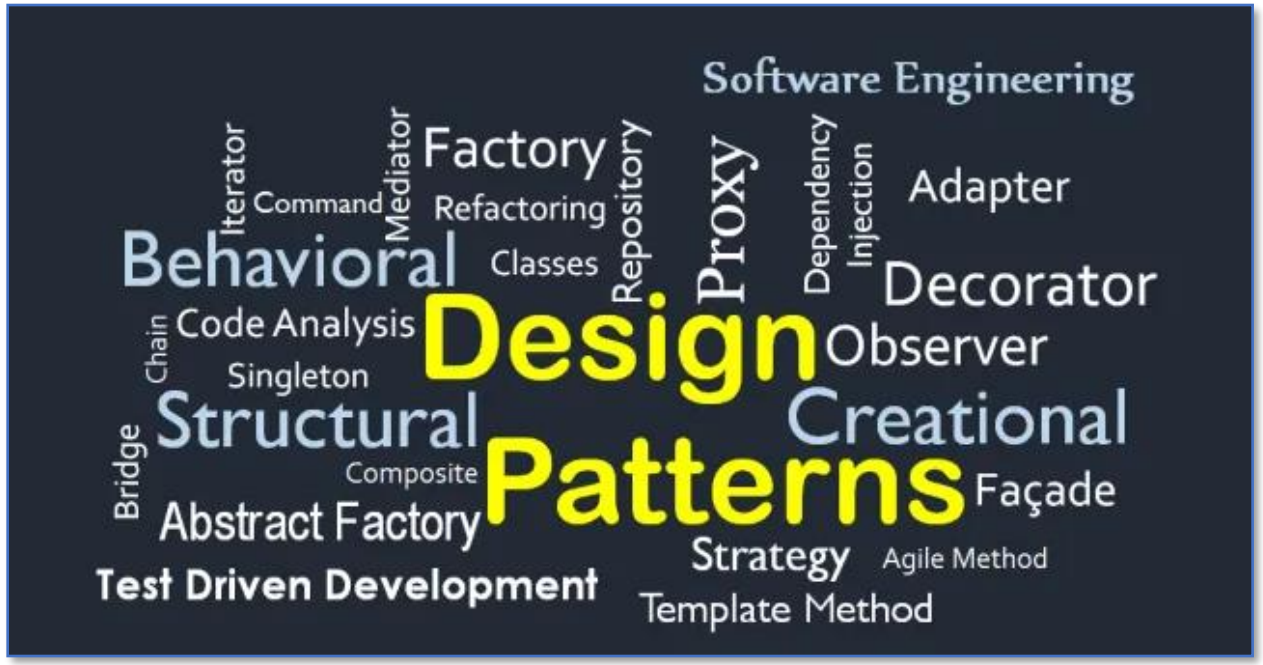


Lesson #1

Introduction to Design Patterns

Course structure

Объектно-ориентированные подходы и приёмы к разработке программного обеспечения / Object-Orientated Software Development Approaches and Design Patterns

Practical Lessons (36 hrs.) + Test

Bibliography

Agenda

1. *What are Design Patterns?*
2. *Why use a Design Patterns?*
3. *Categories of Design Patterns.*

Christopher Alexander

"Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice"

Кристофер Александер

«Любой паттерн описывает задачу, которая снова и снова возникает в нашей работе, а также принцип ее решения, причем таким образом, что это решение можно потом использовать миллион раз, и при этом никакие две реализации не будут полностью одинаковыми»

Кристофер Вольфганг Александер
(1936 - 2022)

Профессор Калифорнийского университета. Архитектор и дизайнер, создатель более 200 архитектурных проектов в Калифорнии, Японии, Мексике и в др. частях мира. Создал и внедрял на практике «язык шаблонов» в архитектуре.
Книга "Notes on the Synthesis of Form" стала обязательной к прочтению для исследователей в области информатики в 1960-х. Она оказала заметное влияние на разработку языков программирования, объектно-ориентированное программирование. Книга "A Pattern Language" оказала влияние на движение за использование языка шаблонов в разработке программного обеспечения. Книга «The Nature of Order» оказала влияние на объектно-ориентированное программирование, особенно в языке C++.

What are Design Patterns?

Design pattern is a general reusable solution to a commonly occurring problem is software design within a given context.

Design patterns are formalized best practices that the programmer can use to solve common problems when designing an application or system.

Что такое паттерны проектирования?

Паттерн проектирования – общее решение часто встречающейся проблемы, возникающей при разработке ПО (в заданном контексте).

Паттерны проектирования – это формализованные лучшие практики, которые программист может использовать для решение часто встречающихся проблем при разработке ПО или систем.

What are Design Patterns?

The **design pattern** describes communicating objects and classes customized to solve a general design problem in a particular context.

Что такое паттерны проектирования?

Далее под **паттерном проектирования** будем понимать описание взаимодействия объектов и классов, адаптированных для решения общей задачи в конкретном контексте.

Comment

A design pattern names, abstracts, and identifies the key aspects of a common design structure that make it useful for creating a reusable object-oriented design. The design pattern identifies the participating classes and instances, their roles and collaborations, and the distribution of responsibilities. Each design pattern focuses on a particular object-oriented design problem or issue.

Замечание

Паттерн проектирования именует, абстрагирует и идентифицирует ключевые аспекты структуры общего решения, которые и позволяют применить его для создания повторно используемого дизайна. Он выделяет участвующие классы и экземпляры, их роли и отношения, а также распределение обязанностей. Каждый паттерн предназначен для решения конкретной задачи объектно-ориентированного проектиров-я.

Why use a Design Patterns?

Design patterns offer numerous advantages in software development. They can simplify the coding process, enhance code maintainability, and promote code reuse.

They also help developers write code that is more efficient, scalable, and adaptable. And they're incredibly beneficial when working on a project with multiple contributors. This is because design patterns provide a shared framework of best practices that can ensure consistency across the codebase.

Зачем использовать паттерны проектирования?

Использование ПП обеспечивает преимущества при разработке ПО. Они упрощают процесс кодирования, позволяют проще сопровождать код, способствуют его повторному использованию.

Они также помогают разработчикам писать более эффективный, масштабируемый и адаптируемый код. Паттерны полезны при работе над проектом с несколькими участниками. Это связано с тем, что шаблоны проектирования предоставляют общую структуру лучших практик, которая может обеспечить согласованность всей базы кода.

Основные элементы паттерна

ИМЯ: Указывая имя, мы сразу описываем проблему проектирования, ее решения и их последствия (одним-двумя словами). Присваивание паттернам имен расширяет наш «словарный запас» проектирования и позволяет проектировать на более высоком уровне абстракции. Наличие словаря паттернов позволяет обсуждать их с коллегами, в документации. Имена позволяют анализировать дизайн системы, обсуждать его достоинства и недостатки с другими.

ЗАДАЧА: Описание того, когда следует применять паттерн. Описание объясняет задачу и ее контекст. Может описываться конкретная проблема проектирования, например способ представления алгоритмов в виде объектов. В нем могут быть отмечены структуры классов или объектов, типичные для негибкого дизайна. Также может включаться перечень условий, при выполнении которых имеет смысл применять данный паттерн.

Основные элементы паттерна

РЕШЕНИЕ: Описание элементов дизайна, отношений между ними, их обязанностей и правил взаимодействия. В решении не описывается конкретный дизайн или реализация, поскольку паттерн – это шаблон, применимый в разных ситуациях. Вместо этого дается абстрактное описание задачи проектирования и ее возможного решения с помощью классов и объектов.

РЕЗУЛЬТАТЫ: Следствия применения паттерна, его вероятные плюсы и минусы. Позволяют оценить варианты использования и оценить преимущества и недостатки данного паттерна. Учитывают баланс затрат времени и памяти, подробности реализации. Поскольку в объектно-ориентированном проектировании повторное использование зачастую является важным фактором, то к результатам следует относить и влияние на степень гибкости, расширяемости и переносимости системы.

Правила описания паттернов проектирования

Название и классификация паттерна

Название паттерна должно быть компактным и четко отражающим его назначение. Выбор названия чрезвычайно важен, потому что оно станет частью вашего словаря проектирования.

Назначение

Краткие ответы на следующие вопросы: что делает паттерн? Почему и для чего он был создан? Какую конкретную задачу проектирования можно решить с его помощью?

Правила описания паттернов проектирования

Другие названия

Другие распространенные названия паттерна, если таковые имеются.

Мотивация

Сценарий, иллюстрирующий задачу проектирования и то, как она решается данной структурой класса или объекта. Благодаря мотивации можно лучше понять последующее, более абстрактное описание паттерна.

Применимость

Описание ситуаций, в которых можно применять данный паттерн. Примеры неудачного проектирования, которые можно улучшить с его помощью. Распознавание таких ситуаций.

Правила описания паттернов проектирования

Структура

Графическое представление классов в паттерне с использованием нотации, основанной на методике Object Modeling Technique (OMT). Также используется диаграмма взаимодействий для иллюстрации последовательностей запросов и отношений между объектами.

Участники

Классы или объекты, задействованные в данном паттерне проектирования, и их обязанности.

Отношения

Взаимодействие участников для выполнения своих обязанностей.

Правила описания паттернов проектирования

Результаты

Насколько паттерн удовлетворяет поставленным требованиям? К каким результатам приводит паттерн, на какие компромиссы приходится идти? Какие аспекты структуры системы можно независимо изменять при использовании данного паттерна?

Реализация

О каких сложностях и подводных камнях следует помнить при реализации паттерна? Существуют ли какие-либо советы и рекомендуемые приемы? Есть ли у данного паттерна зависимость от языка программирования?

Правила описания паттернов проектирования

Пример кода

Фрагменты кода, демонстрирующие возможную реализацию на языке C++.

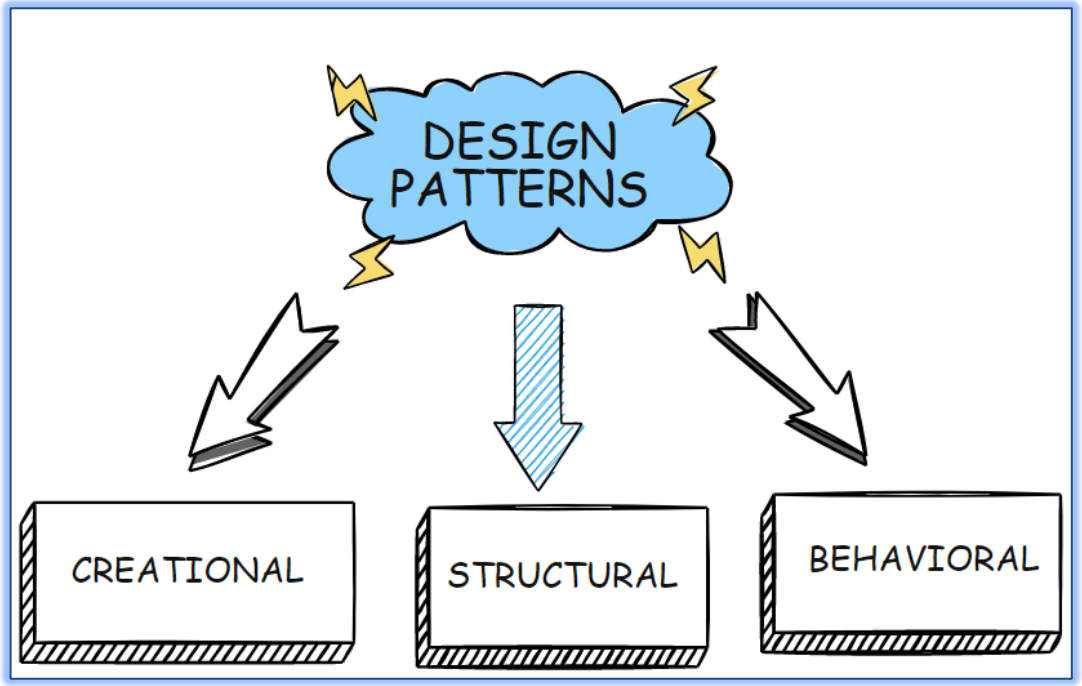
Известные применения

Возможности применения паттерна в реальных системах.

Родственные паттерны

Какие паттерны проектирования тесно связаны с данным? Какие важные различия существуют между ними? С какими другими паттернами хорошо сочетается данный паттерн?

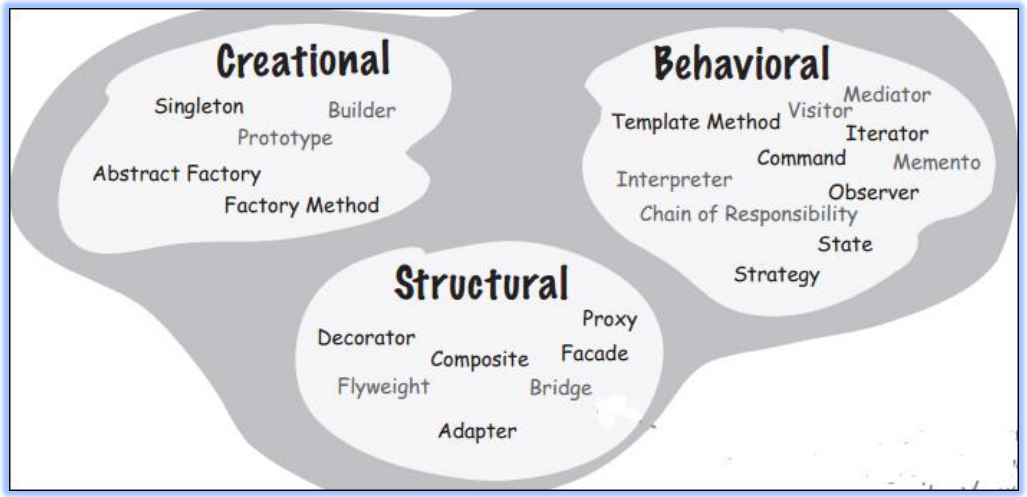
Категории паттернов проектирования



Выделяют три типа паттернов (по цели – назначению паттерна):

Creational (Порождающие)
Structural (Структурные)
Behavioral (Поведенческие)

Категории паттернов проектирования



Creational (Порождающие) – паттерны, связанные с процессом создания объектов.

Structural (Структурные) – паттерны, связанные с композицией объектов и классов.

Behavioral (Поведенческие) – паттерны, определяющие как классы и объекты взаимодействуют между собой.

Категории паттернов проектирования

Первый критерий – *цель* – отражает назначение паттерна (см. выше).
Второй критерий – *уровень* – сообщает, к чему применяется паттерн (к объектам или к классам).

Паттерны ***уровня классов*** описывают отношения между классами и их подклассами. Такие отношения выражаются с помощью наследования, поэтому они статичны, то есть зафиксированы на этапе компиляции.

Паттерны ***уровня объектов*** описывают отношения между объектами, которые могут изменяться во время выполнения и потому более динамичны. Почти все паттерны в какой-то мере используют наследование. Поэтому к категории «паттерны классов» отнесены только те, что концентрируются лишь на отношениях между классами.

Большинство паттернов действует на уровне объектов (см. ниже).

Категории паттернов проектирования

Цель / Уровень	Порождающие паттерны	Структурные паттерны	Паттерны поведения
Класс	Фабричный метод	Адаптер	Интерпретатор Шаблонный метод
Объект	Абстрактная фабрика Одиночка Прототип Строитель	Адаптер Декоратор Заместитель Компоновщик Мост Приспособленец Фасад	Итератор Команда Наблюдатель Посетитель Посредник Состояние Стратегия Хранитель Цепочка обязанностей

Каталог паттернов проектирования

Abstract Factory (абстрактная фабрика)
Предоставляет интерфейс для создания семейств связанных между собой или зависимых объектов без указания их конкретных классов.

Adapter (адаптер)
Преобразует интерфейс класса в другой интерфейс, ожидаемый клиентами.
Обеспечивает совместную работу классов, которая была бы невозможна без данного паттерна из-за несовместимости интерфейсов.

Bridge (мост)
Отделяет абстракцию от реализации, чтобы их можно было изменять независимо друг от друга.

Builder (строитель)
Отделяет конструирование сложного объекта от его представления, чтобы один процесс конструирования мог использоваться для создания различных представлений.

Каталог паттернов проектирования

Chain of Responsibility (цепочка обязанностей)
Можно избежать формирования жесткой связи между отправителем запроса и его получателем, для чего возможность обработки запроса предоставляется нескольким объектам. Объекты-получатели объединяются в цепочку, и запрос передается по цепочке, пока не будет обработан каким-либо объектом.

Command (команда)
Инкапсулирует запрос в виде объекта, позволяя тем самым параметризовывать клиентов по типу запроса, ставить запросы в очередь, протоколировать их и поддерживать отмену выполнения операций.

Composite (компоновщик)
Группирует объекты в древовидные структуры для представления иерархий типа «часть – целое». Позволяет клиентам работать с единичными объектами так же, как с группами объектов.

Каталог паттернов проектирования

Decorator (декоратор)
Динамически наделяет объект новыми обязанностями. Декораторы применяются для расширения существующей функциональности и являются гибкой альтернативой порождению подклассов.

Facade (фасад)
Предоставляет унифицированный интерфейс к набору интерфейсов в некоторой подсистеме. Определяет интерфейс более высокого уровня, облегчающий работу с подсистемой.

Factory Method (фабричный метод)
Определяет интерфейс для создания объектов, позволяя подклассам решить, экземпляр какого класса следует создать. Позволяет классу передать ответственность за создание экземпляра в подклассы.

Каталог паттернов проектирования

Flyweight (приспособленец)
Применяет механизм совместного использования для эффективной поддержки большого числа мелких объектов.

Interpreter (интерпретатор)
Для заданного языка определяет представление его грамматики вместе с интерпретатором, который использует представление для интерпретации предложений языка.

Iterator (итератор)
Дает возможность последовательно обойти все элементы составного объекта, не раскрывая его внутреннего представления.

Каталог паттернов проектирования

Mediator (посредник)
Определяет объект, в котором инкапсулирована информация о взаимодействии объектов из некоторого множества. Способствует ослаблению связей между объектами, позволяя им работать без явных ссылок друг на друга. Это, в свою очередь, дает возможность независимо изменять схему взаимодействия.

Memento (хранитель)
Позволяет без нарушения инкапсуляции получать и сохранять во внешней памяти внутреннее состояние объекта, чтобы позже объект можно было восстановить в точно таком же состоянии.

Observer (наблюдатель)
Определяет между объектами зависимость типа «один-ко-многим», так что при изменении состояния одного объекта все зависящие от него получают уведомление и автоматически обновляются.

Каталог паттернов проектирования

Prototype (прототип)
Описывает виды создаваемых объектов с помощью прототипа и создает новые объекты путем его копирования.

Proxy (заместитель)
Подменяет другой объект для контроля доступа к нему.

Singleton (одиночка)
Гарантирует, что некоторый класс может существовать только в одном экземпляре, и предоставляет глобальную точку доступа к нему.

State (состояние)
Позволяет объекту изменять свое поведение при модификации внутреннего состояния. При этом все выглядит так, словно поменялся класс объекта.

Каталог паттернов проектирования

Strategy (стратегия)
Определяет семейство алгоритмов, инкапсулируя их все и позволяя подставлять один вместо другого. Позволяет менять алгоритм независимо от клиента, который им пользуется.

Template Method (шаблонный метод)
Определяет скелет алгоритма, перекладывая ответственность за некоторые его шаги на подклассы. Позволяет подклассам переопределять отдельные шаги алгоритма, не меняя его общей структуры.

Visitor (посетитель)
Представляет операцию, которую надо выполнить над элементами объектной структуры. Позволяет определить новую операцию без изменения классов элементов, к которым он применяется.