Слайд 2

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

В отличие от готовых функций или библиотек,паттерн представляет собой не какой-то конкретный код, а общую концепцию решения той или иной проблемы.

Паттерны часто путают с алгоритмами, ведь оба понятия описывают типовые решения каких-то известных проблем. Но если алгоритм — это чёткий набор действий, то паттерн — это лишь описание решения, реализация которого может отличаться в от программы к программе..

Слайд 3

### **Порождающие -** Отвечают за удобное и безопасное создание новых объектов или даже целых семейств объектов.

### **Структурные -** Отвечают за построение удобных в поддержке иерархий классов.

### **Поведенческие -** Решают задачи эффективного и безопасного взаимодействия между объектами программы.

Слайд 4

**Фабричный метод** — Предлагает объединить схожие сущности под один интерфейс и создавать их не напрямую, а через отдельный конструктор.

**Абстрактная фабрика** — В свою очередь предлагает поднять уровень абстракции еще выше и вынести под общий интерфейс уже сами конструкторы.

**Одиночка** — Предлагает создать для класса единую точку входа, в которой будет храниться не изменяемый экземпляр класса.

**Прототип** — Предлагает вынести весь функционал по копированию объекта в сам объект, таким образом мы сможем получить копию нужного нам объекта не вдаваясь в его внутреннюю реализацию. Как пример «http.Request»

Слайд 5

Фабричный метод и абстрактная фабрика очень похожи между собой. По своей сути абстрактная фабрика это продолжение фабричного метода и новым уровнем абстракции.

Слайд 6

Одиночка полезный паттерн в тех моментах, когда изменения класса, могут привести к нежелательным последствиям. Как допустим в логгере, нам не хотелось бы, чтобы один сервис внезапно стал слать логги с измененным именем

Слайд 7

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

**Мост** — паттерн, который разделяет один или несколько классов на две отдельные иерархии — как говориться в книге «банда четырех» абстракцию и реализацию, позволяя изменять их независимо друг от друга.

Допустим на нашем сайте, это может быть любая кнопка, ее вид может быть любым в зависимости от клиента, но под капотом она будет делать одно и тоже.

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

**Декоратор** — это структурный паттерн проектирования, который позволяет динамически добавлять объектам новую функциональность, оборачивая их в полезные «обёртки». Как пример `middleware` в обработке запроса.

**Адаптер** — это структурный паттерн проектирования, который позволяет объектам с несовместимыми интерфейсами работать вместе.

**Легковес** — это структурный паттерн проектирования, который позволяет вместить большее количество объектов в отведённую оперативную память. Легковес экономит память, разделяя общее состояние объектов между собой, вместо хранения одинаковых данных в каждом объекте.

Слайд 8

В наших проектах можно часто встретить обертки над сложными библиотеками соединения с базами данных, как пример Postgres. Мы создаем класс Provider, который хранит в себе пул соединений и выполняет порой очень сложные запросы, при этом разработчику не нужно задумываться над этими самыми запросами, главное что они возвращают.

Слайд 9

**Цепочка обязанностей** — это поведенческий паттерн проектирования, который 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** — это поведенческий паттерн проектирования, который превращает запросы в объекты, позволяя передавать их как аргументы при вызове методов, ставить запросы в очередь, логировать их, а также поддерживать отмену операций.

Как мне кажется эта концепция очень схожа с паттерном «Мост»

**Посредник** — это поведенческий паттерн проектирования, который позволяет уменьшить связанность множества классов между собой, благодаря перемещению этих связей в один класс-посредник.

**Снимок** — это поведенческий паттерн проектирования, который 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.

**Состояние** — это поведенческий паттерн проектирования, который позволяет объектам менять поведение в зависимости от своего состояния. Извне создаётся впечатление, что изменился класс объекта.

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

**Шаблонный метод** — это поведенческий паттерн проектирования, который определяет скелет алгоритма, перекладывая ответственность за некоторые его шаги на подклассы. Паттерн позволяет подклассам переопределять шаги алгоритма, не меняя его общей структуры.

**Посетитель** — это поведенческий паттерн проектирования, который позволяет добавлять в программу новые операции, не изменяя классы объектов, над которыми эти операции могут выполняться.

Слайд 10

Основной принцип цепочки обязанностей, это последовательное выполнение действий над объектом запроса. В данном конкретном случае, мы проверяем к какому типу относится логин.