

Факультет программной инженерии и компьютерной техники

Архитектура программных систем

Лабораторная работа №2: Паттерны и их применение

Преподаватель: Перл Иван Андреевич

Выполнил: Кульбако Артемий Юрьевич Р33112

# Задание

Из списка шаблонов проектирования GoF и GRASP выбрать 3-4 шаблона и для каждого из них придумать 2-3 сценария, для решения которых могу применены выбранные шаблоны.

Сделать предположение о возможных ограничениях, к которым можем привести использование шаблона в каждом описанном случае. Обязательно выбрать шаблоны из обоих списков.

## Легковес (Flyweight, GOF)

**Легковес** — это структурный паттерн (отвечают за построение удобных в поддержке иерархий классов) проектирования, который 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. Использование общего «тяжёлого» объекта между другими объектами, а не создание новых экземпляров оного. Этим он похож на синглтон, но в отличии от него, легковес не является явно общим, так как данные объекта разделяются на внешние и внутренние (к примеру, за генерацию flyweight-объекта может отвечать фабричный метод, который при обращении хоть и будет возвращать новый объект, но часть его полей будет заполнена ссылками на старые данные, которые являются «тяжёлыми»).

**Недостатки:**

«Тяжёлые» данные должны быть неизменяемы, доступ для их модификации должен быть запрещён, так как они передаются по ссылке.

**Сценарии:**

1. Текстуры в видеоиграх: на экране может быть множество пуль или дыр от выстрелов, но так как они всегда одинаковы, нет смысла загружать для каждого объекта свою текстуру. Лично видел, как в одних играх при генерации множества одинаковых NPC через редакторы уровней, одни игры начинали тормозить, а другие нет.
2. При реализации систем, где по ajax подгружаются изображения из сети в галерею, в качестве изображения-заглушки используется одна общая для всех элементов галереи картинка. Реализовал подобное в Android-приложении.

## Строитель (Builder, GOF)

**Строитель** — это порождающий паттерн проектирования, который позволяет создавать сложные объекты пошагово. Строитель даёт возможность использовать один и тот же код строительства для получения разных представлений объектов.

**Недостатки:**

Строитель и создаваемый им продукт жестко связаны между собой, поэтому при внесении изменений в класс продукта скорее всего придется соответствующим образом изменять и класс строителя.

**Сценарии:**

1. Позволяет сильно сократить шаблонный код – убрать множество разных конструкторов, например. Тем не менее, в некоторых языках эта проблема решается поддержкой параметров по умолчанию у функций и введением специальной функции, позволяющей провести множественные операции над создаваемым объектов перед его возвратом. Пример на Kotlin:

*data class* Person(*var* name: String, *var* age: Int = 0, *var* city: String = "")  
  
*fun* main() {  
 *val* p = Person("Pupa").apply **{** age = 2  
 city = "Lupa"  
 *// здесь может быть любая логика* **}**}

1. Когда процесс создания нового объекта не должен зависеть от того, из каких частей этот объект состоит и как эти части связаны между собой. К примеру, класс должен приобретать дополнительную функциональность в течении своей жизни. Строитель может добавлять в существующий объект «модули» с новыми функциями. В некоторых языках это решено с помощью прототипного наследования или функций-расширений:

*fun* main() {  
 *fun* String.praiseVT() = *this*.plus(" ВТ - это круто")  
 "теперь все строки могут так: ".praiseVT() *// теперь все строки могут так: ВТ - это круто*}

## Контроллер (Controller, GRASP)

**Контроллер** - отвечает за операции, запросы на которые приходят от пользователя, и может выполнять сценарии одного или нескольких [вариантов использования](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B5%D1%86%D0%B5%D0%B4%D0%B5%D0%BD%D1%82_(UML)) (например, создание и удаление); не выполняет работу самостоятельно, а делегирует компетентным исполнителям.

**Недостатки:**

Порождает дублирование кода, т.к. для каждой сущность нужен свой контроллер. Устранить недостаток можно создав ещё больше контроллеров))) (т.е. контроллеров управляющих другими контроллерами).

**Сценарии:**

1. Активно используется в архитектурах MVC и MVVC, чтобы явно разделить сущности и всю логику вокруг них (включая управление).
2. Может быть единой точкой входа в систему, являясь самым высокоуровневым интерфейсом системы, называется Front Controller. Им, к примеру, является метод main программы.

## Чистая выдумка (Pure fabrication, GRASP)

**Чистая выдумка** – относится к предметной области, но не представляют ни одну из сущностей (отсюда и получил своё название). Необходим для уменьшения зацепления и увеличения связанности.

**Недостатки:**

Сложно вписать в предметную область (в языках, где можно писать функции, не являющиеся частью классов подобной проблемы нет).

**Сценарии:**

1. Есть некая класс, являющийся моделью. Согласно ООП, он сам должен сохранять себя в базу данных, но в таком случае увеличится зацепление, ведь теперь он будет зависеть от объекта, отвечающего за бд. К тому же, подобная реализация повлечёт за собой увеличение бойлерплейт-кода, т.к. сохранение в бд является частой и общей задачей. Поэтому есть смысл отделить модель, класс для операций с бд, и некий «помощник» для модели, который будет отвечать за специфичные операции при взаимодействии модели и бд-класса.
2. Для реализации шаблона Data Transfer Object, используемого в бэкенде для превращения «чистых» данных от клиента в пригодную для использования модель.

# Вывод

Паттерны проектирования – мощный, но сложный инструмент, требующий от программиста прежде всего опыта, в противном же случае, из-за применения неправильного шаблона код может стать куда сложнее чем вовсе без него. Важным пунктом здесь является выбор паттерна под задачу, а не задачи под понравившийся паттерн, порой даже бывает, что какой-либо паттерн реализуется сам по себе при правильном проектировании системы (в своё время, при написании командного интерпретатора я реализовал паттерн «Команда», ещё не имея представления о GOF и GRASP вообще). При этом, я бы не рекомендовал использовать паттерны для небольших программ, так как в таком случае написание всего окружающего бизнес-логику кода может занять больше времени чем сама логика.