Университет ИТМО

Факультет программной инженерии и компьютерной техники
Направление подготовки 09.03.04 Программная инженерия
Дисциплина «Архитектура программных систем»

Отчет

По лабораторной работе №2

Выполнил:
Кузнецов М. А.
P33131
Преподаватель:
Перл И. А.

Санкт-Петербург, 2022 г.

Задание:

Из списка шаблонов проектирования GoF и GRASP выбрать 3–4 шаблона и для каждого из них придумать 2–3 сценария, для решения которых могут быть применены выбранные шаблоны.
Сделать предположение о возможных ограничениях, к которым можем привести использование шаблона в каждом описанном случае. Обязательно выбрать шаблоны из обоих списков.

Выполнение:

Я разберу следующие шаблоны:

- GoF-шаблон Decorator
- GoF-шаблон Cor (Chain of Responsibility)
- GRASP-шаблон Controller

1. Decorator (GoF)

Что это такое:

Decorator (Декоратор) – структурный паттерн проектирования, который позволяет динамически добавлять объектам новую функциональность, оборачивая их в полезные «обёртки».

Сценарии использования:

- Например, у нас есть персонаж **простой** игры, который может носить разные вещи, которые тем или иным образом изменяют его игровые характеристики. Чтобы не порождать огромное обилие классов всех возможных вариантов носки вещей, мы можем просто «оборачивать» каждую последующую вещь над имеющимся набором вещей.
- Другой пример: у нас есть элемент интерфейса, который мы хотим видоизменять. Например, кнопка, к которой можно добавить круглые края, фон, поведение для различных состояний, иконку и так далее. Аналогично, чтобы не создавать все возможные комбинации классов, нужно просто добавить возможность «оборачивать» нашу кнопку во что-то новое, тем самым создавая уже изменённую кнопку.

Минусы и ограничения:

- Объекты, обернутые большое количество раз, становится впоследствии трудно конфигурировать и обслуживать.
- Мы также создаем большое количество мелких объектов, выполняющих соответственно примерно схожую работу.

- Декоратор и его компонент не совсем идентичные -- следует помнить, что "прозрачность" использования декоратора всё же является условной и зависит от реализации проброса запросов самим декоратором.

2. CoR (Chain of Responsibility) (GoF)

Что это такое:

Cor (Chain of Responsibility (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.

Сценарии использования:

- Реальный пример: в языке программирования Swift (для разработки под iOS) есть так называемый Responder Chain, который при тапе на экран помогает определить, что сейчас было нажато. То есть, мы нажимаем на экран, а данная цепочка обязанностей пытается определить элемент, по которому мы нажали, иначе, если текущий элемент не может обработать его, то он передает запрос нажатия дальше по иерархии элементов.
- Другой пример: мы хотим в нашем приложении добавить возможность покупки товара. Для этого, при потенциальном совершении платежа, мы можем проверять необходимые условия по такой цепи: «пользователь авторизован»→«у пользователя достаточно средств»→«у пользователя указан адрес доставки» и так далее. В данном случае при текущем промахе мы **останавливаем** выполнение.
- А вот такой пример: допустим, мы хотим отправить сообщение человеку в нашем приложении, зная только его никнейм. Система начинает проверять возможные каналы доступа пользователя, и при первом удачном, отправляет ему письмо через данный подходящий сервис. Что-то вроде: «чат приложения»→«Telegram»→«почта Gmail». В данном случае при текущем промахе мы **продолжаем** выполнение.

Минусы и ограничения:

- Может случиться такое, что при «прохождении» по нашей цепи запрос в итоге, может быть, не обработан никем. Такой случай может возникнуть из-за неправильно построения структуры цепи.

3. Controller (GRASP)

Что это такое:

Controller — это объект, который отвечает за обработку системных событий, и при этом не относится к интерфейсу пользователя. Controller определяет методы для выполнения системных операций.

Сценарии использования:

- Реальный пример: в среде web- и mobile-разработки есть так называемый шаблон MVC (Model-View-Controller). Здесь четко прослеживается аналогия, так как Controller в MVC соответствует назначению GRASP-овского Controller'а – он принимает запросы с UI (View), обрабатывает их, взаимодействует с моделью (Model), и определяет конечные методы для выполнения операций. Важно: в обоих случаях (GRASP и MVC) Controller никаким образом не относится к интерфейсу! По сути, это сущность, которая не представлена предметной областью.
- Другой пример: теперь представим, что мы делаем игру, в которой персонаж может принимать различные облики\костюмы, и в зависимости от текущего облика, у персонажа открываются разные способности. Для решения данного вопроса можно поступить следующим образом: есть общий Controller, который принимает пользовательские нажатия по интерфейсу, и есть Controller'ы (или обработчики, здесь это по сути то же самое) каждого из костюмов. Соответственно, наш главный и общий Controller обрабатывает поступающие действия-нажатия пользователя и делегирует оставшуюся работу одному из этих обработчиков.

Минусы и ограничения:

- Есть один важный момент, который заключается в следующем: иногда, (особенно актуально в среде web- и mobile-разработки) Controller может «раздуваться» до больших размеров. Например, в разработке под iOS, в шутку иногда шаблон MVC интерпретируют не как Model-View-Controller, а Massive-View-Controller. Это происходит из-за того, что Controller берет на себя слишком много обязанностей и\или слабо делегирует задачи своим «помощникам». Как следствие, такой код становится тяжело поддерживать и тестировать. Для решения данной проблемы зачастую в профессиональной разработке используют другие шаблоны проектирования, которые помогают грамотно и правильно разграничить обязанности классов и объектов в системе, такие как MVVM (Model-View-ViewModel), VIPER(View-Interactor-Presenter-Entity-Routing) и другие. Но это совершенно не значит, что Controller является плохим связующим звеном в работе системы – просто он подходит либо для небольших

проектов, где не имеет смысла такое жесткое разграничение обязанностей, либо требует создания дополнительных классов-помощников, которым можно будет делегировать свою работу.

Выводы:

В ходе выполнения лабораторной работы я:

- Укрепил имеющиеся знания в среде GoF-шаблонов. Познакомился для себя с новыми, выявил преимущества и недостатки каждого. Теперь, при анализе ситуации, мне будет легче обратиться к существующему шаблону, чтобы решить поставленную задачу, не изобретая велосипед :)
- Познакомился с принципами проектирования GRASP (или же GRASP-шаблонами). Узнал, какие методики и приемы можно и нужно использовать в разработке, чтобы грамотно распределять и назначать задачи обязанности и задачи объектам и классам в системе.