Федеральное государственное автономное образовательное учреждение высшего образования "Национальный Исследовательский Университет ИТМО"
Мегафакультет Компьютерных Технологий и Управления
Факультет Программной Инженерии и Компьютерной Техники

Лабораторная работа №2
по дисциплине
'Архитектура программных систем'

Выполнил Студент группы P33102
Лапин Алексей Александрович
Преподаватель:
Перл Иван Андреевич

г. Санкт-Петербург
2023г.

Содержание

1 Текст задания

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.
Keyword: Шаблоны проектирования, GoF, GRASP

2 Шаблоны проектирования GoF

2.1 Легковес (Flyweight)

Описание: Это структурный паттерн 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.

Ограничения/Недостатки:

- **Неизменяемость.** Разделяемые объекты не могут иметь изменяемого состояния, поэтому они могут быть менее гибкими и адаптивными к разным ситуациям.
- **Сложность.** Код программы становится сложнее, так как он теперь работает с разделяемыми и неразделяемыми данными.
- **Дополнительные издержки.** Возникают дополнительные издержки на поиск и управление разделяемыми объектами.
- **Неоправданное использование.** Паттерн можно использовать только тогда, когда в приложении есть определённые повторяющиеся объекты, которые действительно можно разделить. В некоторых случаях разделение объектов может привести к увеличению количества используемой памяти.

Сценарии:

- **Сценарий 1: Разработка компьютерной игры**
  Шаблон проектирования Flyweight может быть применен при разработке компьютерной игры, где требуется управлять большим количеством объектов. Например, если в игре есть множество объектов, представляющих врагов, и эти объекты имеют общие характеристики (такие, как модель, текстуры, звуки), то можно использовать шаблон Flyweight (хранить неизменяемые общие данные во внутреннем состоянии, которое расшаривается между объектами) для сокращения использования памяти и улучшения производительности.
- **Сценарий 2: Система управления заказами**
  В системе управления заказами, где каждый заказ содержит список товаров, шаблон Приспособленец может быть использован для сокращения использования памяти. Вместо хранения каждого товара как уникального объекта, можно использовать общие данные для товаров с аналогичными характеристиками, такими как название, цена и описание

- **Сценарий 3: Векторный редактор:**
  Векторный редактор, который умеет работать с графическими примитивами: Квадрат (Square), Окружность (Circle), Точка (Point). Вместо того, чтобы рисовать примитивы по точкам, мы каждый примитив сделаем легковесом, которые различаются по внутреннему состоянию: у квадрата — размеры сторон (height, width), у окружности — радиус (radius). Таким образом, одинаковые примитивы будут разделять один и тот же легковес.

2.2 Команда (Command)

Описание: 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.

Ограничения/Недостатки:

- **Сложность.** Усложняет код программы из-за введения множества дополнительных классов.
- **Требования к ресурсам.** Паттерн Команда может потребовать больше ресурсов, так как каждая команда является отдельным объектом. Это может привести к увеличению использования памяти, особенно в случае большого количества команд.

Сценарии:

- **Сценарий 1: Операции отмены (Undo) и повтора (Redo)**
  Паттерн Command идеально подходит для реализации этих функций в приложениях. Команда, которую вы хотите отменить или повторить, может быть представлена в виде объекта, который можно хранить для последующего использования. Когда пользователь выбирает отмену, вы просто вызываете метод отмены на соответствующем объекте команды. Для повтора команды вы вызываете метод выполнения на этом же объекте.
- **Сценарий 2: Макросы**
  Паттерн Command можно использовать для создания макросов, т.е. последовательностей команд, которые выполняются вместе как одна операция. Каждая отдельная команда представляется в виде объекта, и эти объекты команд затем группируются вместе для создания макроса. Это может быть полезно в сценариях, где необходимо автоматизировать сложные или повторяющиеся задачи.
- **Сценарий 3: Отложенные операции**
  Когда нужно отложить выполнение операции или выполнить ее в отдельном потоке выполнения, можно использовать паттерн Command. Команда инкапсулирует все необходимые детали операции в объекте, который затем может быть помещен в очередь или передан в другой поток для последующего выполнения. Это можно использовать в сценариях, где операции требуют значительного времени для выполнения и не должны блокировать основной поток выполнения, например, в определенных видах сетевых операций или операциях ввода-вывода.

2.3 Заместитель (Proxy)

Описание: Это структурный паттерн проектирования, который 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.

Ограничения/Недостатки:

- **Сложность кода.** Паттерн Заместитель может сделать код программы сложнее. Клиенты должны работать как с реальными, так и с заместителями, что может сделать код более сложным.
- **Производительность.** Паттерн Proxy может влиять на производительность, так как он добавляет дополнительный уровень абстракции. Это может привести к увеличению времени отклика и замедлению работы приложения.
- **Проблема 'черного ящика'.** В некоторых случаях, прокси может вести себя как 'черный ящик', скрывая от клиента детали реализации и внутреннее состояние реального объекта

Сценарии:

- **Сценарий 1: Ленивая инициализация.**
  Когда есть тяжёлый объект, загружающий данные из файловой системы или базы данных, можно отложить его инициализацию до момента, когда он действительно нужен. Заместитель может управлять созданием этого объекта, обеспечивая его создание только при необходимости.
- **Сценарий 2: Защита доступа.**
  Когда нужно разграничить доступ к объекту в зависимости от прав доступа, можно использовать заместитель для проверки этих прав. Например, если у объекта есть методы, которые должны быть доступны только определённому кругу лиц, то заместитель может проверять, имеет ли вызывающий объект необходимые для вызова метода права.
- **Сценарий 3: Кеширование объектов.**
  Заместитель может кешировать результаты запросов клиентов и управлять их жизненным циклом. Это может быть полезно, когда нужно кешировать результаты тяжелых запросов и управлять их жизненным циклом.
- **Сценарий 4: История запросов.**
  Заместитель может сохранять историю обращений к реальному объекту, позволяя клиенту управлять и повторять их по необходимости.

3 Шаблоны проектирования GRASP

3.1 Контроллер (Controller)

Описание: отвечает за прием запросов от пользователя и делегирование их исполнения соответствующим компетентным исполнителям.

Ограничения/Недостатки:

- **Сложность управления.** Контроллеры могут стать сложными и трудными для управления, если они начинают обрабатывать слишком много системных событий. Это может привести к тому, что код станет сложным для понимания и поддержки.
- **Риск нарушения SRP (Single Responsibility Principle).** Если контроллер начинает обрабатывать слишком много задач, он может нарушить принцип единственной ответственности (SRP), что делает его более подверженным к изменениям и более сложным для тестирования.
- **Связность с пользовательским интерфейсом.** Поскольку контроллеры обычно тесно связаны с пользовательским интерфейсом, любые изменения в пользовательском интерфейсе могут потребовать изменений в контроллере. Это может привести к частым изменениям кода.

Сценарии:

- **Сценарий 1: Обработка пользовательского ввода.**
  Controller может быть использован для обработки входящих запросов от пользователей в веб-приложении. Например, в модели MVC (Model-View-Controller), контроллеры обычно обрабатывают пользовательский ввод, выполняют соответствующие действия в модели и обновляют представление.
- **Сценарий 2: Оркестрация между компонентами системы.**
  Controller может быть использован для координации и последовательного выполнения действий между различными компонентами системы. Это может быть полезно в сложных системах, где нужно координировать действия между множеством различных объектов или классов.
- **Сценарий 3: Управление внутренними событиями системы.**
  Controller может быть использован для обработки внутренних событий системы, таких как таймеры, исключения или изменения состояния. Это позволяет централизовать логику обработки этих событий и упрощает управление ими.

4 Вывод

В ходе выполнения лабораторной работы были изучены шаблоны проектирования GoF и GRASP, а также были рассмотрены сценарии их использования.