МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«Национальный исследовательский университет
ИТМО»

ФАКУЛЬТЕТ ПРОГРАММНОЙ ИНЖЕНЕРИИ И КОМПЬЮТЕРНОЙ
ТЕХНИКИ

ЛАБОРАТОРНАЯ РАБОТА №2

по дисциплине

«АРХИТЕКТУРА ПРОГРАММНЫХ СИСТЕМ»

Выполнил:
Студент группы P3318
Рамеев Тимур
Ильгизович
Преподаватель:
Перл Иван Андреевич

Санкт-Петербург 2025

Содержание

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.

State

Позволяет объекту изменять свое поведение при изменении его внутреннего состояния.

Сценарии использования:

Сценарий 1: Управление состоянием заказа на маркетплейсе

Описание:

Маркетплейс, где заказ может находиться в различных состояниях, таких как "Ожидание оплаты", "Обработка", "Отправлен", "Доставлен" и т.д.

Применение шаблона State:

Вы можете использовать шаблон State для представления каждого состояния заказа в виде отдельного класса. Класс заказа будет иметь ссылку на текущее состояние, и в зависимости от событий, таких как оплата или отправка, состояние заказа будет динамически изменяться.

Ограничения:

Дополнение новых состояний может потребовать изменений в коде класса заказа.

При большом количестве состояний система может стать сложной для понимания и поддержки.

Сценарий 2: Разработка редактора графических объектов

Описание:

Допустим, у вас есть графический редактор, где пользователь может создавать, редактировать и перемещать различные графические объекты, такие как круги, квадраты и треугольники.

Применение шаблона State:

Каждый тип графического объекта может иметь свои собственные состояния редактирования, например "Создание", "Редактирование", "Перемещение". Шаблон State позволяет объекту изменять свое поведение в зависимости от текущего состояния, что делает его более гибким для изменений.

Ограничения:

- Добавление новых типов графических объектов может потребовать создания новых классов состояний.
- Необходимо внимательно обрабатывать переходы между состояниями, чтобы избежать непредсказуемого поведения.

Flyweight

Использует разделение для обеспечения эффективного использования памяти или любого другого ресурса.

Сценарий 1: Графический редактор с большим количеством изображений:

Описание:
Графический редактор, который позволяет пользователям создавать и редактировать изображения. Редактор имеет большое количество точек, линий, форм и других графических элементов.

Применение шаблона Flyweight:
Используя паттерн Flyweight, можно создать разделяемые объекты для общих атрибутов элементов, таких как цвет, текстура и т.д. Например, все красные линии могут использовать один и тот же объект Flyweight для хранения информации о цвете.

Ограничения и недостатки:

- **Ограничение применимости:** Эффективность шаблона Flyweight зависит от того, насколько много объектов можно разделить между собой. Если у объектов слишком много уникальных характеристик, использование Flyweight может не дать значительного выигрыша в производительности.

- **Усложнение кода:** Разделение внутреннего и внешнего состояния может усложнить код и его понимание. Разработчикам необходимо внимательно управлять разделяемым и неразделяемым состоянием.

Сценарий 2: Система управления шаблонами документов:

Описание:
Система для создания документов с использованием шаблонов. Каждый документ может содержать общие элементы, такие как логотип компании, даты и другие общие фрагменты текста.

Применение шаблона Flyweight:

Шаблон Flyweight может быть использован для представления общих элементов документов. Например, объект Flyweight может содержать информацию о логотипе компании, который встречается в разных документах.

Ограничения и недостатки:

- **Потеря некоторой гибкости:** Применение Flyweight может привести к потере гибкости, если для некоторых элементов все равно требуется индивидуальная настройка.

- **Необходимость внимательного управления состоянием:** Необходимо внимательно управлять состоянием Flyweight-объектов, чтобы избежать неожиданных изменений в одном месте, затрагивающих другие части системы.

Low coupling

Минимизировать зависимости между классами и объектами, чтобы система оставалась гибкой и изменения в одной части не влияли на другие.

Сценарий 1: Разработка расширяемого плагинового приложения

Описание:
Приложение, которое может легко расширяться за счет подключаемых плагинов. Каждый плагин предоставляет определенную функциональность, и вы хотите, чтобы добавление, удаление или обновление плагинов не требовало значительных изменений в основной части приложения.

Применение "low coupling":
Разделяйте интерфейсы плагинов от их реализации, используйте абстракции, чтобы основное приложение не зависело от конкретных классов плагинов.

Ограничения и недостатки:

- **Сложность взаимодействия:** Увеличивается сложность взаимодействия между модулями из-за использования абстракций, что может потребовать больше усилий для понимания и отладки.
- **Дополнительный слой абстракции:** Введение слоя абстракции может увеличить сложность кода и требовать дополнительной работы для его поддержки.

Сценарий 2: Разработка многоплатформенного приложения

Описание:
Приложение должно работать на разных платформах (например, Windows, Linux, macOS), и вы хотите минимизировать зависимость кода, специфичного для каждой платформы.

Применение "low coupling":
Выделяйте платформозависимый код в отдельные модули, предоставляя общие интерфейсы для взаимодействия с этим кодом. Таким образом, код для каждой платформы остается минимальным.

Ограничения и недостатки:

- **Дополнительная абстракция:** Использование дополнительных слоев абстракции может усложнить структуру кода.
- **Ограниченные возможности платформы:** Некоторые платформозависимые возможности могут потребовать уровня связанности, который сложно избежать.

Сценарий 3: Разработка микросервисного приложения

Описание:
Микросервисная архитектура, где каждый сервис предоставляет определенный функционал. Хотите, чтобы изменения в одном сервисе не требовали изменений в других сервисах.

Применение "low coupling":
Определите четкие API для взаимодействия между сервисами, минимизируя зависимости от внутренней реализации каждого сервиса.

Ограничения и недостатки:

- **Сложность взаимодействия:** Введение API для каждого сервиса может увеличить сложность взаимодействия между ними, особенно при необходимости соблюдения целостности данных.
- **Управление версиями:** Изменение API может потребовать управления версиями, чтобы сохранить обратную совместимость.

Indirection

Ответственность за выполнение операции должна быть передана посреднику или промежуточному объекту для уменьшения прямых связей между объектами.

Сценарий 1: Работа с различными источниками данных

Описание:
Система, которая получает данные из различных источников: базы данных, внешних API и локальных файлов. Каждый источник данных имеет свой уникальный формат и интерфейс. Используя шаблон Indirection, можно ввести промежуточный слой, который абстрагирует работу с каждым источником данных и предоставляет единый интерфейс для работы с данными в приложении.

Применение шаблона Indirection:

Создание класса DataManager, который служит в качестве индиректора между компонентами системы и различными источниками данных.

Каждый конкретный источник данных (например, DatabaseProvider, APIProvider, FileProvider) реализует общий интерфейс через DataManager.

Компоненты приложения взаимодействуют только с DataManager, что позволяет легко добавлять или изменять источники данных без изменения клиентского кода.

Ограничения и недостатки:

- **Дополнительная сложность:** Введение промежуточного слоя может добавить дополнительную сложность в систему, особенно если количество источников данных невелико.
- **Производительность:** Индирекция может повлечь за собой небольшую дополнительную нагрузку на производительность из-за дополнительных уровней абстракции.

Сценарий 2: Управление зависимостями в большом проекте

Описание:
Крупный проект с множеством компонентов, взаимодействующих друг с другом. Использование прямых зависимостей между компонентами может привести к жестким связям, усложняя поддержку и изменения.

Применение шаблона Indirection:

Введение слоя DependencyManager, который управляет зависимостями между компонентами.
Каждый компонент регистрирует свои зависимости через DependencyManager, и они получают доступ к друг другу через этот слой.

Клиентский код не имеет прямых зависимостей между компонентами, что облегчает замену или обновление компонентов.

Ограничения и недостатки:

- **Дополнительная сложность конфигурации:** Необходимость регистрации зависимостей может сделать конфигурацию проекта более сложной.
- **Потеря производительности:** Индирекция может влиять на производительность из-за дополнительных уровней абстракции.

Вывод

В рамках данной лабораторной работы были рассмотрены несколько шаблонов проектирования GoF и GRASP. Каждый из них предоставляет рекомендации по организации кода, повышению его гибкости, повторному использованию и обеспечению легкости сопровождения. Применение шаблонов проектирования — это инструмент, который помогает разработчикам создавать гибкий, поддерживаемый и масштабируемый код. Но важно учитывать контекст проекта и применять шаблоны там, где они наиболее уместны, а также постоянно оценивать их эффективность в рамках развивающегося проекта.