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:

Поленов Кирилл Александрович

Проверил:

Перл Иван Андреевич

г. Санкт-Петербург, 2025

Содержание

1. Задание

Из списка шаблонов проектирования GoF и GRASP требуется выбрать 3–4 шаблона и для каждого из них предложить 2–3 сценария применения.

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

В рамках данной работы были выбраны следующие шаблоны проектирования из каталога GoF:

- Prototype
- Adapter
- Proxy
- Strategy

2. Prototype (GoF)

2.1. Описание

Prototype (Прототип) — порождающий шаблон проектирования, предназначенный для создания новых объектов путём клонирования уже существующих экземпляров. В отличие от стандартного создания объектов через конструкторы, Prototype предполагает использование заранее подготовленного объекта-прототипа, состояние которого копируется в новый экземпляр.

Данный шаблон особенно полезен в ситуациях, когда:

- создание объекта является ресурсоемкой операцией;
- инициализация объекта требует большого количества параметров;
- необходимо создавать множество однотипных объектов с небольшими отличиями;
- структура объектов может изменяться во время выполнения программы.

Prototype снижает зависимость клиентского кода от конкретных классов создаваемых объектов, так как клиент работает с интерфейсом клонирования, а не с конструкторами.

2.2. Сценарии использования

2.2.1. Создание конфигураций сложных объектов

В графических редакторах и игровых движках Prototype часто применяется для создания множества однотипных объектов: графических элементов интерфейса, игровых сущностей, эффектов или уровней. Вместо создания каждого объекта с нуля используется заранее подготовленный прототип, который клонируется при необходимости.

Ограничения:

- Сложность реализации корректного клонирования для объектов с глубокими зависимостями.
- Ошибки поверхностного копирования могут приводить к совместному использованию состояния между объектами.

2.2.2. Кэширование и повторное использование шаблонов

В системах с высокой нагрузкой Prototype может использоваться совместно с кэшированием. Например, часто используемые объекты (шаблоны документов, отчёты, DTO с типовыми данными) создаются один раз, сохраняются как прототипы и затем клонируются по запросу.

Ограничения:

- Усложнение контроля жизненного цикла объектов.

- Усложняется контроль актуальности данных в прототипах.
- Возможен рост потребления памяти при большом количестве прототипов.

2.3. Общие ограничения Prototype

- Необходимость реализации интерфейса клонирования для всех поддерживаемых классов.
- Сложность корректной реализации глубокого копирования.
- Может приводить к неявному копированию состояний.

3. Adapter (GoF)

3.1. Описание

Adapter (Адаптер) — структурный шаблон проектирования, предназначенный для преобразования интерфейса одного класса в интерфейс, ожидаемый клиентом. Адаптер позволяет объектам с несовместимыми интерфейсами работать совместно без изменения их исходного кода.

Основная идея шаблона заключается во введении промежуточного слоя, который инкапсулирует логику преобразования данных и вызовов методов. Это особенно актуально при интеграции сторонних библиотек, устаревших компонентов или внешних сервисов.

3.2. Сценарии использования

3.2.1. Интеграция сторонней библиотеки

При подключении внешней библиотеки с неудобным или отличающимся API адаптер позволяет привести интерфейс библиотеки к формату, принятому в системе.

Ограничения:

- Увеличение количества классов и дополнительный уровень абстракции усложняют систему.
- Возможны накладные расходы на преобразование данных.

3.2.2. Поддержка устаревших интерфейсов

Адаптер может использоваться для поддержки старых версий API, позволяя новому коду работать с устаревшими компонентами.

Ограничения:

- Увеличение количества поддерживаемого кода.
- Рост технического долга при длительном использовании адаптеров.

3.2.3. Унификация интерфейсов внешних сервисов

В микросервисной архитектуре адаптеры могут использоваться для унификации взаимодействия с различными внешними API (платёжные системы, сервисы доставки, аналитика), предоставляя единый интерфейс для бизнес-логики.

Ограничения:

- Адаптер может скрывать особенности конкретных сервисов.
- При изменении внешнего API требуется обновление адаптера.

3.3. Общие ограничения Adapter

- Может скрывать архитектурные проблемы системы.
- Избыточен при возможности прямого изменения интерфейсов.

4. Proxy (GoF)

4.1. Описание

Proxy (Заместитель) — структурный шаблон проектирования, предоставляющий объект, который контролирует доступ к другому объекту. Прокси реализует тот же интерфейс, что и реальный объект, и может выполнять дополнительные действия до или после передачи вызова.

Прокси применяется для решения задач, связанных с контролем доступа, ленивой инициализацией, логированием, кэшированием и удалённым взаимодействием.

4.2. Сценарии использования

4.2.1. Ленивая загрузка ресурсов

В системе работы с медиафайлами прокси может загружать изображения или видео только при фактическом обращении к ним.

Ограничения:

- Дополнительные задержки при первом обращении.
- Усложнение отладки.

4.2.2. Контроль доступа

Прокси может проверять права пользователя перед передачей запроса к защищенному объекту.

Ограничения:

- Дублирование логики безопасности.
- Риск ошибок при неправильной настройке.

4.2.3. Кэширование результатов операций

Прокси может сохранять результаты вычислений или запросов и возвращать их при повторных обращениях, снижая нагрузку на систему.

Ограничения:

- Сложность управления актуальностью кэша.
- Дополнительное потребление памяти.

4.3. Общие ограничения Proxy

- Увеличение количества вспомогательных классов.
- Возможное снижение производительности.

5. Strategy (GoF)

5.1. Описание

Strategy (Стратегия) — поведенческий шаблон проектирования, который определяет семейство алгоритмов, инкапсулирует каждый из них и делает их взаимозаменяемыми. Контекст использует стратегию через общий интерфейс, не зная о конкретной реализации.

Шаблон Strategy позволяет динамически изменять поведение системы без изменения клиентского кода и способствует соблюдению принципа открытости/закрытости.

5.2. Сценарии использования

5.2.1. Расчёт стоимости доставки

В интернет-магазине могут использоваться разные стратегии расчета доставки: по расстоянию, весу или срочности. Стратегия позволяет динамически выбирать нужный алгоритм.

Ограничения:

- Увеличение числа классов.
- Необходимость управления выбором стратегии.

5.2.2. Алгоритмы сортировки

В системе обработки данных стратегия может применяться для выбора оптимального алгоритма сортировки в зависимости от объема и структуры данных.

Ограничения:

- Дополнительная сложность конфигурации.
- Возможность неправильного выбора стратегии.

5.2.3. Политики аутентификации и авторизации

В системах безопасности стратегия может использоваться для реализации различных механизмов аутентификации (JWT, OAuth2, LDAP).

Ограничения:

- Рост числа реализаций стратегий.
- Повышенные требования к тестированию.

5.3. Общие ограничения Strategy

- Усложнение архитектуры за счёт множества мелких классов.
- Снижение прозрачности логики выполнения.

6. Controller (GRASP)

6.1. Описание

Controller — шаблон GRASP, определяющий объект, ответственный за обработку системных событий и входящих запросов. Контроллер принимает запросы от пользовательского интерфейса или внешних систем и делегирует выполнение соответствующим объектам доменной модели или сервисного слоя.

Основная задача Controller — изолировать логику обработки запросов от представления и тем самым повысить модульность и управляемость системы.

6.2. Сценарии использования

6.2.1. Обработка HTTP-запросов в веб-приложении

В серверном веб-приложении контроллер принимает HTTP-запросы, выполняет базовую валидацию данных и передаёт управление сервисному слою.

Ограничения:

- Контроллер может стать перегруженным логикой.
- Усложняется модульное тестирование.

6.2.2. Координация пользовательских сценариев

Контроллер управляет последовательностью действий пользователя, например, процессом оформления заказа или регистрации.

Ограничения:

- Риск нарушения принципа единой ответственности.
- Рост связности с другими компонентами системы.

6.3. Общие ограничения Controller

- Возможность превращения контроллера в «божественный объект».
- Высокая зависимость от структуры приложения.

7. Information Expert (GRASP)

7.1. Описание

Information Expert — шаблон GRASP, согласно которому ответственность должна быть возложена на тот объект, который обладает необходимой информацией для ее выполнения. Это позволяет добиться высокой связности и логической целостности доменной модели.

7.2. Сценарии использования

7.2.1. Расчёт стоимости заказа

Объект заказа (Order) содержит информацию о позициях, количестве и ценах, поэтому именно он должен отвечать за расчёт итоговой стоимости.

Ограничения:

- Возможна перегрузка объекта логикой.
- Усложнение тестирования доменных классов.

7.2.2. Управление состоянием сущностей

Объект, владеющий состоянием (например, User или Account), отвечает за его изменение и проверку корректности.

Ограничения:

- Рост сложности доменных сущностей.
- Необходимость строгого контроля обязанностей.

7.3. Общие ограничения Information Expert

- Риск перегрузки доменных объектов.
- Требует аккуратного проектирования модели
- Нарушения принципа единой ответственности

8. Low Coupling (GRASP)

8.1. Описание

Low Coupling (Низкая связность) — шаблон GRASP, направленный на минимизацию зависимостей между объектами системы. Чем ниже связность между компонентами, тем проще систему изменять, тестировать и сопровождать. Данный паттерн не описывает конкретную структуру классов, а задаёт архитектурный принцип распределения ответственности.

Low Coupling тесно связан с принципами SOLID и является одной из ключевых эвристик при проектировании масштабируемых и гибких систем.

8.2. Сценарии использования

8.2.1. Взаимодействие сервисного и репозитория слоёв

В многоуровневой архитектуре бизнес-логика взаимодействует с базой данных через абстракции (интерфейсы репозиториев), а не напрямую с конкретными реализациями. Это снижает связность между слоями и упрощает замену реализации хранения данных.

Ограничения:

- Увеличение количества абстракций и интерфейсов.
- Повышение сложности навигации по коду.

8.2.2. Интеграция внешних сервисов

Для взаимодействия с внешними API используется отдельный слой или адаптер, что позволяет изолировать основную систему от изменений во внешних сервисах.

Ограничения:

- Дополнительные накладные расходы на проектирование.
- Возможность избыточной абстракции при простых интеграциях.

8.3. Общие ограничения Low Coupling

- Чрезмерное стремление к снижению связности может привести к over-engineering.
- Избыточное количество интерфейсов усложняет понимание системы.

6. ВЫВОД

В ходе лабораторной работы были рассмотрены шаблоны проектирования из каталогов GoF и GRASP. В качестве шаблонов GoF были проанализированы Prototype, Adapter, Proxy и Strategy. Были рассмотрены три GRASP-паттерна: Controller, Information Expert и Low Coupling, демонстрирующие принципы распределения ответственности и снижения связности в объектно-ориентированных системах.

Использование шаблонов проектирования позволяет создавать более гибкие, расширяемые и поддерживаемые программные системы, однако применение паттернов требует осознанного подхода, так как их избыточное или формальное использование может привести к усложнению архитектуры и снижению читаемости и сопровождаемости кода.