**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ,   
МЕХАНИКИ И ОПТИКИ»**

***Изображение выглядит как черный, темнота

Автоматически созданное описание***

**Факультет Программной Инженерии и Компьютерной Техники**

**Дисциплина:**

**«*Архитектура Программных систем*»**

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 2**

**Выполнил:**

Студент гр. P32151 *Соловьев Артемий Александрович*

**Проверил:**

*Перл Иван Андреевич*

Санкт-Петербург

2023г.

**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.

Выбранные паттерны:

1. Singleton (GoF)
2. Observer (GoF)
3. Polymorphism (GRASP)
4. Controller (GRASP)

**Singleton**

Целью Singleton является гарантия того, что у класса есть только один экземпляр, и предоставление глобальной точки доступа к этому экземпляру.

**Сценарии использования**

1. **Управление конфигурацией приложения.**В представлении управления конфигурацией приложения может быть создан класс, использующий шаблон Singleton. Этот класс может содержать параметры конфигурации, такие как адреса баз данных, настройки безопасности и другие глобальные параметры. При использовании Singleton, мы можем гарантировать, что у нас есть единственный экземпляр в приложении, и все части приложения будут использовать одни и те же конфигурационные данные.

*Ограничения и недостатки:*

1. *Глобальный доступ*:   
   Использование Singleton для управления конфигурацией может привести к ситуации, где любая часть кода приложения имеет глобальный доступ к конфигурационным данным. Это создает трудности контролирования того, какие части кода могут читать и изменять конфигурацию, что в конечном итоге может повлиять на чистоту архитектуры.
2. *Управление жизненным циклом*:  
   Если изменения в конфигурации не частые, но требуют перезапуска приложения, использование Singleton может быть излишним, так как он создает объект, который живет на протяжении всего жизненного цикла приложения.
3. *Обратная сторона поддержки многих экземпляров*:  
   Если в будущем потребуется поддерживать несколько конфигураций для разных компонентов приложения, использование Singleton окажется нецелесообразным.
4. **Соединение с базой данных.**Рассмотрим класс, который управляет соединением с базой данных. С использованием шаблона Singleton, мы можем гарантировать, что в приложении есть только одно активное соединение, что важно для эффективного использования ресурсов и предотвращения конфликтов при одновременном доступе.

*Ограничения и недостатки:*

1. *Глобальный доступ*:   
   В случае управления соединением с базой данных глобальный доступ к экземпляру Singleton может сделать его сложным для тестирования, поскольку трудно предоставить замену Singleton при тестировании.
2. *Управление жизненным циклом*:  
   Если приложение требует активного соединения с базой данных только в определенных моментах пользования, использование Singleton может быть неоптимальным, так как соединение будет устанавливаться, даже если оно не используется.

**Observer**

Целью Observer является определять зависимость "один ко многим" между объектами так, чтобы при изменении состояния одного объекта все зависящие от него объекты автоматически оповещались и обновлялись.

1. **Графический интерфейс пользователя в мессенджере.**  
   Допустим у нас в мессенджере у пользователя есть список контактов, и когда один из контактов отправляет сообщение, это сообщение должно мгновенно отобразиться в интерфейсе всех открытых окон чата, связанных с этим контактом.  
   В этом примере есть обозреваемый объект -- чат с конкретным контактом, и наблюдатель, представляющий графическое окно чата. Когда в чате отправляется сообщение, все окна этого чата обновляют свой интерфейс.  
     
   *Ограничения и недостатки:*
2. *Синхронизация и потокобезопасность:*  
   Если множество наблюдателей одновременно реагирует на изменения (например, при одновременном поступлении нескольких сообщений), может возникнуть необходимость в синхронизации в потокобезопасном режиме.
3. *Зависимость между наблюдателями*:  
   При увеличении числа наблюдателей может возникнуть зависимость между ними, что усложнит поддержку и внесение изменений. Если изменится формат сообщений, это может потребовать изменений в каждом из наблюдателей.
4. *Неоптимальная реакция на изменения*: В случае, если один из наблюдателей требует больше времени на обработку изменений, это может замедлить общий процесс уведомления всех наблюдателей.
5. **Реакция на изменения в системе мониторинга ресурсов сервера**Представим систему мониторинга ресурсов сервера, где различные компоненты отображают, например текущую загрузку процессора, использование памяти и другие параметры. Когда один из компонентов получает новые данные о ресурсах, все остальные компоненты также должны обновить свои данные.  
     
   *Ограничения и недостатки*:
6. *Частые обновления*:  
   Если очень часто требуется обновлять состояние ресурсов, это может привести к избыточному использованию ресурсов системы. Наблюдатели должны быть готовы эффективно обрабатывать обновления и не приводить к дополнительным нагрузкам.
7. *Отсутствие гарантий доставки*:  
   Наблюдатель не гарантирует доставку уведомлений. Если наблюдатель временно недоступен или произошла ошибка при обновлении, уведомление может быть утрачено.
8. *Сложность отладки*:  
   Когда множество наблюдателей наблюдают за одним объектом, сложно отследить, какой именно наблюдатель реагирует на изменение. Это создаст трудности для отладки и выявления проблем.

**Polymorphism**

Polymorphism используется для достижения гибкости в обработке разных типов объектов.

1. **Обработка различных типов документов в текстовом редакторе**  
   Предположим, у нас есть текстовый редактор, который должен обрабатывать различные типы документов, такие как текстовые документы, изображения и таблицы. Используя полиморфизм, мы можем создать иерархию классов для различных типов документов. В этом случае, независимо от типа документа, мы можем обращаться к нему через общий интерфейс, что позволяет нам использовать полиморфизм при работе с коллекциями документов или в других сценариях.  
     
   *Ограничения и недостатки:*
2. *Усложнение иерархии*  
   С ростом числа типов документов может усложниться иерархия классов, особенно если типы документов имеют сложные взаимосвязи. Это усложнение может привести к потере ясности в структуре программы.
3. *Ограниченность типами*  
   Полиморфизм может быть ограничен в том случае, если различные типы документов имеют слишком разные интерфейсы, что затрудняет их обработку через общий интерфейс.
4. **Разработка плагинов в системе электронной коммерции**Представим систему электронной коммерции, в которой разработчики могут создавать плагины для добавления новой функциональности. Используя полиморфизм, мы можем определить общий интерфейс Plugin и позволить разработчикам создавать свои плагины, реализующие этот интерфейс.  
   В данном случае, система может динамически загружать и использовать плагины через их общий интерфейс Plugin, что позволяет достичь гибкости и расширяемости системы

*Ограничения и недостатки:*

1. *Сложность в поддержке*С ростом числа плагинов система может становиться сложной в поддержке, особенно если каждый плагин вносит свои уникальные изменения.
2. *Конфликты имен*Полиморфизм может привести к конфликтам имен, особенно если различные плагины пытаются использовать одни и те же имена классов или методов. Это может затруднить интеграцию и использование различных плагинов.

**Controller**

Controller относится к паттернам проектирования, связанным с архитектурой MVC. Он предоставляет централизованный контроль для обработки пользовательских запросов и управления потоком данных между моделью и представлением.

1. **Веб-приложение с использованием MVC**  
   Рассмотрим веб-приложение, где у нас есть база данных, представление HTML страницы и контроллер, который обрабатывает HTTP-запросы и управляет взаимодействием между моделью и представлением.  
   В этом примере контроллер обрабатывает запросы от пользователя, вызывая методы модели и представления для обновления данных и отображения соответствующей страницы.  
     
   *Ограничения и недостатки:*
2. *Сложность в управлении состоянием*  
   Когда приложение становится масштабным, и у нас есть множество контроллеров, управление состоянием между ними может стать сложным. Например, переход от одной страницы к другой может потребовать сохранения и восстановления большого объема данных.
3. *Поддержка асинхронных операций*  
   Обработка асинхронных операций и взаимодействие с ними (например, асинхронные запросы AJAX) может потребовать дополнительных усилий в контроллерах и представлениях.
4. **Мобильное приложение с архитектурой MVC**  
   Предположим, у нас есть мобильное приложение с архитектурой MVC, где пользователь взаимодействует с интерфейсом пользователя, который в свою очередь взаимодействует с контроллером для обработки действий пользователя и обновления модели.  
   В этом случае контроллер реагирует на действия пользователя, например нажатия на кнопки или ввод текста, и обновляет модель и представление соответственно.  
     
   *Ограничения и недостатки:*
5. *Ограниченность в навигации*  
   Когда приложение имеет сложную структуру навигации, использование одного центрального контроллера может привести к сложностям в управлении этой навигацией. Может потребоваться введение дополнительных механизмов для обработки навигационной логики.
6. *Проблемы с производительностью*  
   В случае больших и сложных мобильных приложений может возникнуть проблема производительности из-за избыточного использования контроллера для различных экранов и действий.

**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.