МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное автономное
образовательное учреждение высшего образования
“Национальный исследовательский университет ИТМО”

**ФАКУЛЬТЕТ ПРОГРАММНОЙ ИНЖЕНЕРИИ
И КОМПЬЮТЕРНОЙ ТЕХНИКИ**

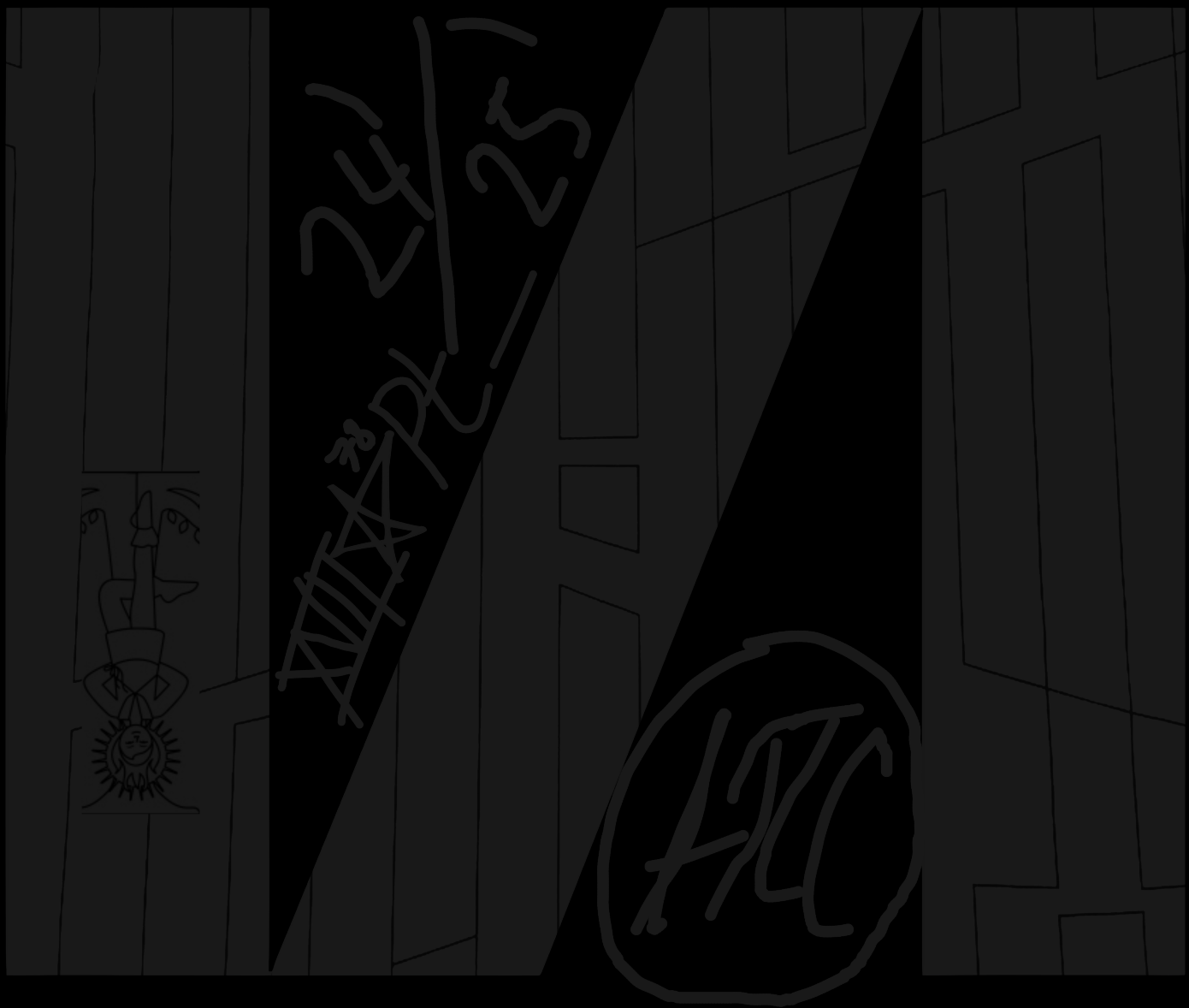
ЛАБОРАТОРНАЯ РАБОТА №2

по дисциплине
“Архитектура Программных Систем”

Работу выполнил:
Студент группы P3311
Болорболд Аригуун
Преподаватель:
Перл Иван Андреевич

г. Санкт-Петербург
2025 г.

Содержимое



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 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.

2 Выполнение

2.1 GoF шаблон «Команда» (Command)

Сценарии использования.

- **Настройки управления в играх.**
  В каждой игре где-то есть фрагмент кода, который отвечает за считывание пользовательского ввода: нажатия кнопок, события клавиатуры, щелчки мыши — что угодно. Он берет каждый ввод и преобразует его в осмысленное действие в игре. Обычно эти функции привязаны определенные кнопки к определенному действию. Но иногда нам хочется поменять кнопки, вызывающий функцию. Для этого нам нужен объект, который мы можем использовать для представления игрового действия. Это мы решаем через шаблон «Команда». Благодаря ему мы можем добавить дополнительные функциональности, например отмена и повтор действия.

- **Настройка кнопки в мультиплатформенных программах.**
  Представим, что мы разрабатываем мультиплатформенную файловую систему. У каждой операционной системы —- свои правила взаимоотношения с файлами, и свои механизмы работы. Чтобы реализовать такую систему нам нужен два типа класса вызывателя и получателя. Пришел черед паттерна «Команда». В получателях реализуем классы под каждой операционной системой реализованы работы с файлом. Мы будем вызывать эти классы в наших Командах, а эти команды в классах вызывателя.

- **Ограничения:**
  В играх не которые команды представляют собой просто действия бек состояния. В этих случаях наш шаблон занимает память без необходимости. Паттерн Flyweight справился бы с этим ограничением.

2.2 GoF шаблон «Прототип» (Prototype)

Сценарии использования.

- **Спавнер в гейм-дизайне.** Представим ситуацию, когда игровое окружение процедурно генерируется. Это может быть сделано для увеличения размера уровня или для разнообразия. При процедурное генерации использоваться так называемый «спавнер» почти всегда необходимо. В этом и помогает шаблон проектирования «прототип». Допустим у нас есть город, в котором существуют жители. У них есть простой искусственный интеллект, который нужен только для того, чтобы была иллюзия живого мира. Прописывать поведения каждому невозможно, в такой ситуации создается оригинальный, изначальный прототип, которому задается поведение, и этот объект просто копируется, ведь разнообразие в данном случае не так важно.

- **Автоматизация создания базы данных производителя.** Представим производство по сборке, к примеру, моторов для Rolls Royce. Для создания нового двигателя конструктор получает прототип двигателя для разных типов машин и просто меняет серийный номер. Не нужно будет создавать новый подкласс каждый раз, а при изменении типов двигателя нужно просто передать другие объекты. При необходимости нужно будет переписывать всего несколько методов, когда остальные остаются без изменений.

- **Ограничения:** Такой подход накладывает серьезные ограничения – каждый объект должен реализовывать операцию клонирования, но при необходимости глубокого клонирования,

когда используются ссылки, указатели, задача такого клонирования сильно усложняется. К первому примеру выше, если мы хотим в будущем поменять поведения только некоторых объектов, задача станет намного сложнее.

2.3 GoF шаблон «Состояние» (State)

Сценарии использования.

- **Инструмент проверки и рецензирования научных статей.** Возьмем пример сайт для публикации научных статей и исследований. Процесс создания статьи можно разделить на несколько состояний – публикация на рассмотрении, на модерации и рецензировании, и готова к публикации для всех. Во время каждого из состояний инструменты редактирования, просмотра и другие функции отличаются. Паттерн «состояние» позволит не создавать отдельные разделы или страницы сайта под каждый функционал, а менять его в зависимости от состояния.
- **Приложение музыкального сервиса – плеера** Рассмотрим приложение Spotify для прослушивания музыки. Здесь мы так же можем выделить несколько состояний прослушивания. Просушивание альбома, своего плейлиста, подкаста или радио. В каждом случае функционал будет частично отличаться. Для своих плейлистов нужны функции рекомендаций и авто воспроизведения новых треков, для радио требуется возможность выбора жанра или исполнителя, для подкастов нужны комментарии или оценки. Но в то же время все состояния имеют общий базовый функционал паузы, уровня громкости и т. д.
- **Приложение YouTube для телевизора** YouTube имеет возможность отправить выбранное видео на телевизор, где так же установлено приложение. В этим случает смартфон становится «пультом управления» воспроизведения видео. Остается весь функционал приложения, включая очередь воспроизведения, лайки и громкость, хотя само видео показывается на другом экране. Здесь явно видно разделение на два разных состояния внутри одного приложения.
- **Ограничения:** С одной стороны, паттерн освобождает нас от большого количество условных операторов state-машины, но помимо этого происходит сильное усложнение и уменьшение гибкости кода, особенно если самих состояний мало и меняются они редко.

2.4 GRASP шаблон «Информационный эксперт» (Information Expert)

Сценарии использования.

- **Везде, где ООП.**
  Шаблон Информационный Эксперт определяет базовый принцип распределения ответственности. Этот шаблон – самый очевидный и самый важный из девяти. Есди его не учесть – то получится код, с которым далеко не удобно работать. Применение шаблона информационный эксперт повышает связность модулей и не противоречит свойству инкапсуляции. Суть данного паттерна можно выразить так: “информация должна обрабатываться там, где она содержится”.
- **Управление объектами в информационной системе.**
  Рассмотрим сценарий, в котором вы проектируете систему управления библиотекой. В этой системе при добавлении новой книги в библиотеку необходимо создать объект ‘Book‘. Ответственность за создание объектов ‘Book‘ должна лежать на классе типа ‘Library‘ или отдельном классе ‘BookFactory‘. Это гарантирует, что логика создания объектов ‘Book‘ централизована и инкапсулирована, что упрощает управление.
- **Ограничения:** Если ViewModel состоит из множества представлении и при этом имеет сложную логику, то оно разрастает до такой степени, что станет невозможным написать и поддердивать юнит-тесты. Также в некоторых местах данный принцип приводит к дублированию кода.

2.5 GRASP шаблон «Устойчивый к изменениям» (Protected Variations)

Один из самых важных показателей хорошего кода – простота изменений и расширений функционала. Паттерн предполагает нахождений «неустойчивых» мест – фрагменты кода, которые чаще всего редактируются при новых изменениях, и устранения их, путем создания, к примеру, интерфейсов вокруг них.

Сценарии использования.

Привести конкретный пример сложно, так как паттерн касается любого написания кода, но представим приложение, реализующее оплату по банковской карте. Реализована платежная система, передача банковский данных и защита данных. При портировании приложения на Android и IOS помимо оплаты по банковской карте нужны функции Google Pay и Apple Pay. Придется сильно переписывать код, чтобы добавить новый функционал. Вместо этого можно разделить общую реализацию оплаты на составляющие, чтобы выбор способа оплаты не менял реализацию полностью.

Ограничения.

Иногда полностью изменить модель невозможно, изменения все равно будут. Кроме того, предугадать изменения тоже сложно, на это влияют различные особенности и способы взаимодействия частей приложения. Для хорошей реализации шаблона и написания качественного, соответствующего ему кода нежно много усилий и времени, необходимо учитывать множество факторов как фреймворки, платформы и различный особенности используемого языка.

3 Вывод

В ходе лабораторной работы я изучил паттерны программирования GoF и GRASP. Я понял, что паттерны программирования напутствуют на эффективный и чистый код.