Федеральное государственное автономное образовательное учреждение высшего образования
Университет ИТМО

Кафедра Вычислительной Техники

Дисциплина: Архитектура программных систем

Лабораторная работа №2

Выполнил: **Патутин Владимир Михайлович**

Группа: **P33101**

Преподаватель: **Перл Иван Андреевич**

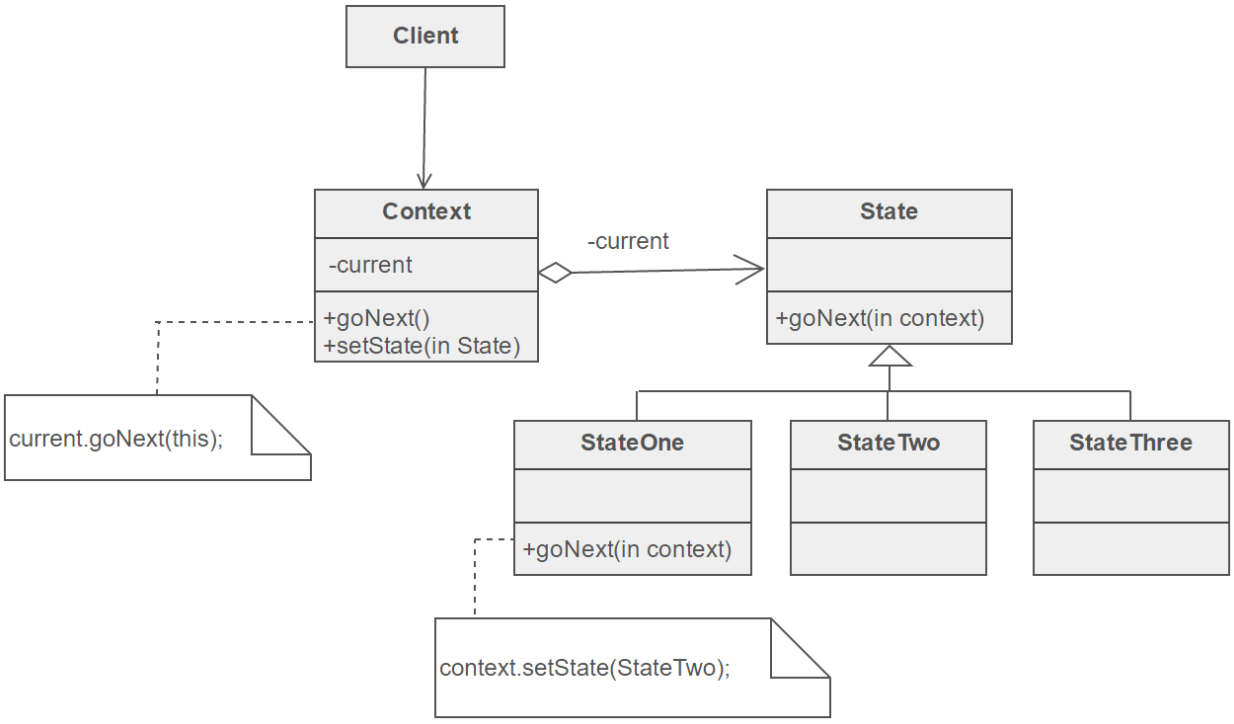
2020г

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.

Паттерн Состояние (GOF):

Позволяет объекту варьировать свое поведение в зависимости от внутреннего состояния. Извне создается впечатление, что изменился класс объекта.



Сценарий использования:

- Написание конечных автоматов для telegram ботов. Конечные автоматы (Finite State Machine) реализуются при помощи такого паттерна. Мы можем задать каждому пользователю свое состояние, которое будет хранится в контексте. В зависимости от этого состояния пользователь может получить разный функционал от бота.
- Запуск программного сценария рендера с различной конфигурацией. У нас есть такие программы, как Блендер и Майя, которые предоставляют возможность запуска рендера с использованием скриптов Python. В зависимости от сцены, которую нужно отрендерить, могут запускаться различные функции (выбор ассетов, включение нужного движка и т.д). Именно такая логика может передаваться через класс-состояние или функцию-состояние, которое описывает необходимый набор действий для текущего случая.

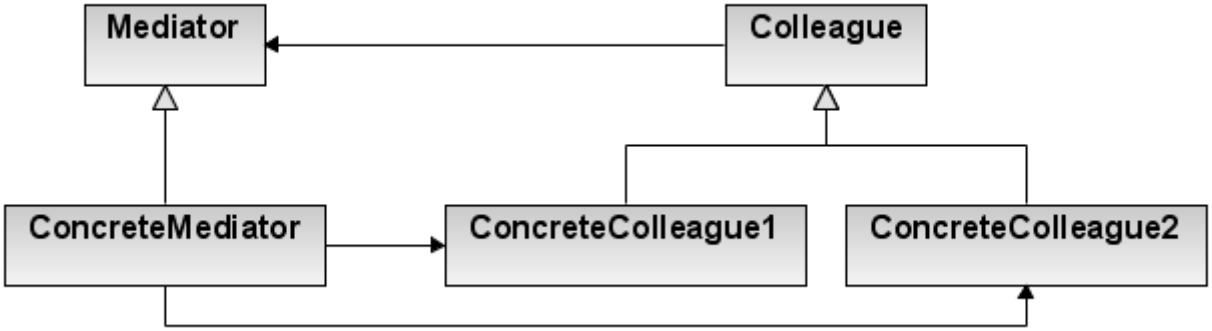
Ограничения:

1. Сложность системы: если в системе всего несколько состояний, то не рекомендуется использовать метод состояний, так как в конечном итоге вы добавите ненужный код.
2. Каждый производный класс состояния связан со своим братом, который прямо или косвенно вводит зависимости между подклассами.

3. Класс, который использует паттерн состояние, должен знать, в какой момент времени отдать выполнение своему состоянию. У нас может быть большое количество различных мест, где это может происходить, что в последствии приводит к нагромождению проверок на необходимость обращения. В свою очередь это снижает гибкость и прозрачность кода.

Паттерн Посредник (GOF):

Обеспечивает взаимодействие множества объектов без необходимости ссылаться друг на друга. Тем самым достигается слабосвязанность взаимодействующих объектов.



Сценарий использования:

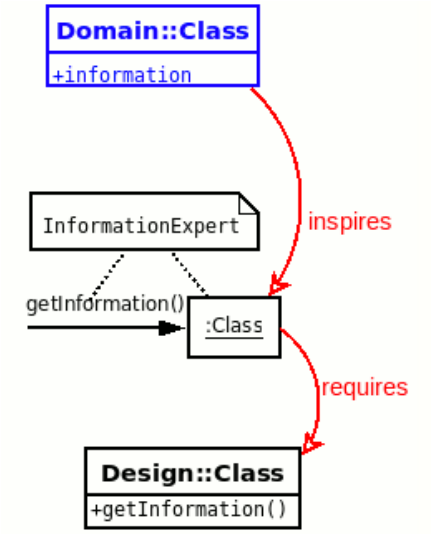
- Представим обычный сайт, который предоставляет запись на курсы повышения квалификации (openedu, stepik, etc). У нас существует огромное количество курсов и огромное количество студентов. Если мы не опытный программист и не знает паттерна посредника, то нам придется обрабатывать огромное количество подклассов, отвечающих за взаимосвязь, что приведет к огромному количеству дополнительного кода. А при добавлении посредника мы сможем предотвратить появление неупорядоченного кода, что приводит к его лучшей читаемости, но и нам будет легко новых посредников, не нарушая существующий клиентский код.
- Отличным примером будет доставка еды «Сбермаркет». Каждый человек в данный момент знает о доставке еды, и почти каждая крупная сеть магазинов предоставляет возможность доставки еды. Если бы нам надо было написать софт для СберМаркета, то использование паттерна «Посредник» отлично бы подошло для этого.

Ограничения:

1. Посредник может быть преобразован в объект, который слишком много знает или делает слишком много.
2. Структура объекта-посредника может стать слишком сложной, если мы вложим в нее слишком много логики.
3. Он полностью централизует управление, потому что шаблон посредника меняет сложность взаимодействия на сложность посредника.

Информационный эксперт (GRASP):

Это принцип, используемый для определения того, кому делегировать обязанности. Эти обязанности включают методы, вычисляемые поля и так далее.



Сценарий использования:

- Сбор статистики из классов. Для сбора статистики у существующих классов стоит вспомнить про это паттерн. Ведь согласно ему, лучшим способом реализоваться сбор статистики – это возложить сбор на сами классы, добавив в них методы для этого, ведь эти классы знают о себе больше других.
- Создание логирования. У нас часто есть два варианта при создании логирования: 1. Создание логирования на более высоком уровне и в одном месте. **2.** Создание логирования на более низком уровне, во всех подклассах. И данный паттерн гласит, что лучше выбрать 2 вариант, потому что подкласс располагает большими сведениями о своем поведении, из чего следует что он может более подробно описать ситуации, которые возникли с ним.

Ограничения:

1. Возлагая сбор статистики на сами классы это требует от нас написания большого объема кода, который ведет к худшей читаемости кода.
2. Слабое сцепление.

Выводы:

Шаблоны проектирования – неотъемлемая часть жизни архитектора. Разумеется, их знание необходимо и рядовому программисту, однако именно архитекторам ПО приходится сталкиваться с ними каждый день. Многие шаблоны очевидны, если долго пишешь код. Однако знание их названий имеет некоторую пользу, потому что при этом можно проще коммуницировать с другими разработчиками, не объясняя каждому свои мысли, которые уже были ранее сформулированы и придуманы. Однако не стоит воспринимать их как панацею от всех проблем и пытаться реализовать шаблон конкретно. Стоит использовать их с умом.

В ходе выполнения работы я познакомился с шаблонами GRASP (о шаблонах GoF я слышал давно). Могу подметить, что принципы, не шаблоны, а именно принципы, входящие в GRASP напрямую вытекают из принципов ООП. Шаблоны проектирования GoF, это именно шаблоны, которые стоит применять с умом там, где это уместно, с пониманием зачем тот или иной шаблон там применяется