Федеральное государственное автономное образовательное учреждение высшего образования

Университет ИТМО

Дисциплина: Архитектура программных систем

Лабораторная работа 2

Выполнил:
Кривоносов Егор Дмитриевич

Группа: P33111

Преподаватель:
Перл Иван Андреевич

2021 г.

г. Санкт-Петербург

Задание

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.

Выполнение

Выбранные паттерны:

Factory Method	(Фабричный метод - GoF)
Observer	(Наблюдатель - GoF)
Proxy	(Заместитель - GoF)
Protected variations	(Устойчивость к изменениям - GRASP)

Factory Method

Описание

Фабричный метод - порождающий шаблон проектирования, предоставляющий подклассам (дочерним классам) интерфейс для создания экземпляров некоторого класса.

Сценарии использования

1. **Генерация объектов**
   Создание множества схожих объектов является одним из стандартных способов использования фабрики. Когда нам нужно много «подобных», но не одинаковых объектов, фабрика может прийтись очень кстати. Например, в видеоиграх, если нам нужно создать большое количество зданий или детализировать стену до кирпичиков, нам идеально поможет такой паттерн, как фабрика. Так если не использовать паттерн, нам придется париться по поводу того, правильно ли мы генерируем клонов, или было отличие, а генерация по общему интерфейсу дает возможность не задумываться об этих вещах. Также генерация объектов по данному паттерну, позволяет добавлять новые объекты для клонирования без особых проблем (лёгкое расширение).

   Ограничения и недостатки:
   Ограниченность интерфейса и невозможность создания уникальных объектов без лишних хлопот. Фабрика создана для предоставления определенного интерфейса, который может не подразумевать под собой добавление уникальных объектов, например кабриолетов или половинчатых кирпичей, а в проекте из использования необходимо, приходится хардкодить, чтобы получить всего пару уникальных объектов.

2. **Заполнение баз данных**
   Для заполнения баз данных мы также можем использовать фабрику. Мы можем использовать данный паттерн, как посредника между базой данных и человеком. Она может условному пользователю выдать интерфейс по созданию объектов, которые будут заполняться в таблице базы данных. Использование данного

паттерна может облегчить работу с базой данных, позволить ему не лезть в код БД и отделить представление БД от представления пользовательского интерфейса.

Ограничения и недостатки:
Основные ограничения данного примера - невозможность изменения таблиц в базе данных, добавление или удаление столбцов, без изменения интерфейса фабрики. Если хочешь добавить сотрудникам пункт с зарплатой, придется пере-собирать фабрику так, чтобы она могла предоставить пользователю возможность добавлять зарплату.

Observer

Описание

Наблюдатель — это поведенческий паттерн проектирования, который создаёт механизм подписки, позволяющий одним объектам следить и реагировать на события, происходящие в других объектах. Паттерн Наблюдатель предлагает хранить внутри объекта издателя список ссылок на объекты подписчиков, причём издатель не должен вести список подписки самостоятельно. Он предоставит методы, с помощью которых подписчики могли бы добавлять или убирать себя из списка.

Сценарии использования

1. **Model-View-Controller**
   Модель MVC на данный момент находит огромное распространение в системах пользовательского интерфейса. Её суть заключатся в том, что логика программы делится на три части – Model – предоставляет данные и реагирует на команды контроллера, View – отвечает за отображение данных для пользователя, Controller – интерпретирует действия пользователя отображая необходимые изменения. Наблюдателя можно использовать в качестве View части. Он может позволить реализовать процесс наблюдения за различными кнопками/элементами интерфейса, и при их нажатии изменять отображение.

   Ограничения и недостатки:
   Один из главных недостатков Observer - является его случайный порядок действий. То есть если мы хотим реализовать View, нам нужно продумать этот момент, и реализовать все элементы интерфейса на одном уровне представления так, чтобы не возникало возможных конфликтов из-за работы разных частей интерфейса. А это уже само по себе огромное ограничение, если данный паттерн заставляет программиста писать все на одном условном уровне.

2. **Система издатель-подписчик**
   Благодаря паттерну наблюдателя можно реализовать систему издатель-подписчик. В данном случае Observer опять же будет выступать в роли связующего. После того, как некоторый издатель – выпустит продукт, об этом узнает Observer тем самым запустит рассылку для всего подписчиков данного продукта издателя. Реализация может быть и в обратную сторону. После того. Как подписчик подпишется на продукт издателя, Observer – оповестит издателя об увеличении продаж/необходимости увеличения поставок.

Ограничения и недостатки:
В случае с примером издатель-подписчик, мы не знаем кому и когда будут приходить уведомления/продукты первыми, оно будет отправляться всем в случайном порядке, что может быть неудобно, если издатель, например биржа, и им необходимо, чтобы клиенты получали своевременный равный отклик о прогнозах.

Proxy

Описание

Заместитель — это структурный шаблон проектирования, позволяющий создавать подобие Декораторов, но не на функции, а на методы (иногда поля) объектов.

Сценарии использования

1. **Аспектно-ориентированное программирование**
   Proxy классы позволяют легко реализовать некоторые концепции АОП на низком уровне. В частности, это применяется в Spring AOP, где для многих задач применяется декларативное управление сквозной логикой приложения с помощью аннотаций, которые в процессе загрузки бина в CDI считываются. В результате, вместо исходного объекта в контейнер помещается прокси-объект, применяющий сквозную логику к исходным методам там, где это необходимо.

   Ограничения и недостатки:
   Аспекты невозможно применять и отменять в процессе жизни приложения, хотя, это, наверное, больше ограничение АОП. Реализация АОП с помощью прокси-объектов неполноценна относительно общих идей АОП, например, в AspectJ АОП реализован лучше и позволяет определять больше различных точек соединения, нежели чем Spring AOP, ограничивающийся (в силу ограничений встроенных возможностей рефлексии Java) лишь вызовами методов.

2. **Object Relation Mapping**
   Прокси объекты позволяют удобным образом организовать работу с ORM. Например, так сделано в JPA/Hibernate. Благодаря использованию прокси-объектов возможно более натуральное взаимодействие с ORM за счёт возможности вызова методов сущностей так, будто они напрямую связаны и синхронизированы с состоянием БД, что обеспечивается замещением оригинальной логики методов логикой, нацеленной на взаимодействие с ORM. Более конкретно, при вызове геттеров и сеттеров объектов сущностей программист (в ряде случаев) может полагаться на синхронизацию с ORM, что значит, что при вызове геттера будет получено актуальное (по мнению ORM) состояние экземпляра сущности в БД, а при вызове сеттера, изменение будет (условно) мгновенно передано в БД.

   Ограничения и недостатки:
   Внутри приложения прокси-объекты могут быть восприняты программистом как оригинальные объекты сущностей, в связи с чем могут возникнуть неприятные проблемы, если программист не учитывает природу объектов.

Protected variations

Описание

Устойчивость к изменениям - один из фундаментальных шаблонов разработки программных систем. Принцип работы его заключается в том, что необходимо идентифицировать точки вариаций, эволюций или неустойчивости и обеспечить для них надежный, по возможности не изменяемый интерфейс. Использование шаблона позволяет архитекторам и программистам быть готовым к постоянно меняющимся требованиям. Многие принципы и шаблоны (Polymorphism, Low Coupling и так далее) являются частным случаем реализации этого шаблона.

Сценарии использования

1. **Docker, контейнеризация**
   Проектируя приложение, архитектор не может знать заранее какую операционную систему будет иметь конечный пользователь (точка вариации). Docker предоставляет интерфейс взаимодействия контейнеров с ресурсами компьютера и демонстрирует наглядную реализацию принципа Protected Variations.

   Ограничения и недостатки:
   Ограничением в данном случае будут накладные расходы на виртуализацию. Получая единый интерфейс, теряем в производительности.

2. **JNDI, службы именования**
   Служба предоставляет единый интерфейс взаимодействия с ресурсами, который инкапсулирует поиск необходимого ресурса и избавляет от необходимости знать физический адрес, который скорее всего будет отличаться в разных окружениях. Также использование службы именования защищает программиста от изменения расположения используемых ресурсов.

   Ограничения и недостатки:
   Дополнительные расходы на поддержание системы будут недостатком такого подхода.

Вывод

В ходу выполнения данной лабораторной работы я изучил множество материалов по паттернам, разобрался в принципах их работы, также проанализировал сценарии, где используются выбранные мною паттерны. Нашел ограничения и недостатки использования паттернов в моих сценариях, а также описал основы внедрения паттерна в сценарии.