МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г.ЧЕРНЫШЕВСКОГО»

Кафедра математического обеспечения вычислительных комплексов и информационных систем

СОЗДАНИЕ ПАТТЕРНА И ФРЕЙМВОРКА ДЛЯ ОПТИМИЗАЦИИ АРХИТЕКТУРЫ И РАЗРАБОТКИ КОРПОРАТИВНЫХ ПРИЛОЖЕНИЙ

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студента 4 курса 441 группы
направления 02.03.03 Математическое обеспечение и администрирование информационных систем
факультета компьютерных наук и информационных технологий
Степаняна Унана Рубиковича

Научный руководитель:
Профессор _____ Андрейченко Д. К.

Зав. кафедрой:
Профессор _____ Андрейченко Д. К.

Саратов 2021

ВВЕДЕНИЕ

Современная разработка крупных корпоративных приложений крайне сложный и процесс требующий исполинских человеческих и финансовых ресурсов. Технологический базис разработки программного обеспечения сформирован не в полном объеме и имеет множество недоработок в области построения рабочего процесса, что дает пространство для оптимизации процесса разработки в целях экономии ресурсов.

В данной работе, объектом исследования является разработка приложений для взаимодействий с пользователем. Примером таких приложений являются программы голосового управления, голосовые ассистенты, чат боты и другие приложения взаимодействующие с клиентом, которые стали крайне популярны в современное время. В связи с небольшим набором инструментов для разработки подобных приложений, каждая компания принимается за разработку собственного приложения практически с нуля.

Цель работы заключается в формировании паттерна и внутреннего фреймворка на его основе, упрощающего процесс разработки вышеописанной группы приложений, снижающий требования к техническим специалистам и сокращение затрат на поддержку и доработку приложения.

Задача, которую необходимо решить для достижения поставленной цели, состоит в описании паттерна и разработке внутреннего фреймворка, задающего процесс разработки основного пайплайна. Результатом должен быть интерфейс, который можно использовать для моделирования сценария диалога с пользователем.

Для достижения этой цели нужно решить следующие задачи:

- определиться с выбором концепции фреймворка или библиотеки
- описать новый паттерн проектирования
- определить набор основных интерфейсов
- разработать внутренний фреймворк

- сформировать ответы клиенту без фреймворка
- использовать фреймворк для формирования ответов
- провести анализ процесса разработки до и после внедрения фреймворка

Теоретическая и/или практическая значимость бакалаврской работы. В теоретической части был описан новый паттерн, который описывает требования к интерфейсам и их реализации. В практической части был спроектирован и реализован фреймворк на основе описанного паттерна. Фреймворк предоставляет возможность строить ответ на запрос только в рамках возможных сценариев.

Структура и объем работы. Бакалаврская работа состоит из введения, теоретического, аналитического и практического разделов, заключения, списка использованных источников и приложения. Общий объем работы – 84 страницы, из них 49 страниц – основное содержание, включая 13 рисунков, список использованных источников информации – 29 наименований.

КРАТКОЕ СОДЕРЖАНИЕ РАБОТЫ

Первая глава «ФРЕЙМВОРК И БИБЛИОТЕКА**»** посвящена сравнению концепции библиотеки и фреймворка.

Исходя из описаний, библиотека и фреймворк выполняют одни и те же задачи: оптимизируют процесс разработки и архитектуру, а также позволяют переиспользовать код. Однако, если разобраться детальнее, библиотека используется для решения конкретной проблемы или добавления определенной функции в вашу программу. Фреймворк предоставляет нечто гораздо более общее и многоразовое.

Фреймворк от библиотеки отделяют три вещи.

Инверсия контроля

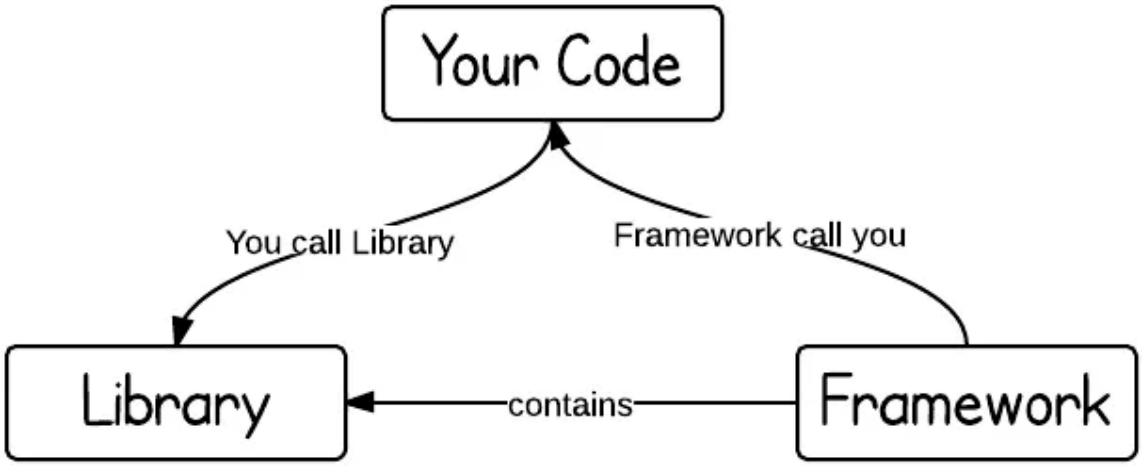

Рис. 3 – диаграмма отношений фреймворка, библиотеки и пользовательского кода.

На рисунке 3 представлена диаграмма отношений фреймворка, библиотеки и пользовательского кода. Эта диаграмма показывает, что библиотека может быть частью фреймворка, так же как и частью приложения. Это свойство говорит о том, фреймворк по своим свойствам ближе к приложению, а не к библиотеке.

Первое важное различие между фреймворком и библиотекой заключается в том, кто контролирует процесс разработки. С библиотекой кода разработчик обычно обращается к ней всякий раз, когда считает, что это уместно. Во фреймворке весь поток управления уже существует, и есть множество предопределенных пробелов, которые необходимо заполнить пользовательским кодом. Фреймворк обычно более сложный. Он определяет каркас, в котором приложение определяет свои собственные функции для заполнения каркаса. Таким образом, клиентский код будет вызываться фреймворком при необходимости. В результате часто возникает ощущение, что процесс разработки контролируется фреймворком, а не разработчиком. Это инверсия управления. Обычно это упрощается как некоторая вариация следующего:

Библиотека: Позвоните нам, чтобы выполнить работу.

Фреймворк: вы не звоните нам, мы вам позвоним.

Именно из-за этой инверсии контроля фреймворки становятся более самостоятельными, и сильно расширяют инструментарий разработчика.

Расширяемость

Важное понятие в объектно-ориентированном программировании - принцип открытости/закрытости. Принцип открытости/закрытости гласит, что поведение кода должно быть открыто для изменения / расширения без его непосредственного изменения. Простым примером этого является наследование в классах, которое позволяет подклассу добавлять необходимые ему функции и функции без необходимости напрямую изменять код в родительском классе. Расширяемость позволяет разработчику добавлять новые функции в структуру или адаптировать поведение существующих функций в соответствии со своими потребностями без изменения исходного кода. Хорошие фреймворки создаются с учетом расширяемости. Они предоставляют общие функции, необходимые для разработки универсального приложения, но оставляют себя открытыми для конкретных дополнений и изменений, необходимых для конкретного приложения. Если бы фреймворк не был расширяемым, он был бы довольно ограничен в своих функциональных возможностях, и это сделало бы его менее привлекательным для изучения. Разница с библиотеками здесь гораздо менее заметна, однако, поскольку фреймворки обеспечивают общую функциональность, они должны быть построены с учетом расширяемости, чтобы можно было реализовать специфические для приложения функции. Библиотеку не обязательно строить с учетом расширяемости, ее основная цель - выполнить конкретную задачу.

Немодифицируемый код

Фреймворки генерируют много кода, и по большей части, это неприкасаемый код, поскольку фреймворк проектируется с намерением расширить его во время пользовательской разработки, без изменения исходного кода фреймворка. В зависимости от специфики задач, пользователь

расширяет фреймворк под нужды своего приложения, наследуясь или реализуя интерфейсы, с которыми фреймворк может работать.

Как итог, имеем две концепции.

Библиотека предоставляет набор инструментов для решения конкретных задач, не зависит а так же не влияет на структуру приложения, на его бизнес логику и процесс разработки. Абсолютная безразлична к окружению, подключается только с помощью агрегации и забирает на себя часть утильного функционала.

Фреймворк не смотря на свои сходства с библиотекой, противоположен ей в вопросе участия в структуре и окружения приложения. Его можно агрегировать или унаследовать, однако это не влияет на тот факт, фреймворк диктует свои правила во время разработки или планирования архитектуры. Накладывает свои ограничения такое свойство как расширение приложения своими рамками тех же свойств. Именно эта возможность позволить ограничить разработчика в действиях и сфокусировать решение в рамках, которые заданы фреймворком и выступить окрестратором решений и скорректировать процесс разработки в большой команде.

Из вышесказанного, очевидно, что для решения поставленной задачи, концепция фреймворка более предпочтительна.

Вторая глава «АНАЛИЗ ШАБЛОНОВ ПРОЕКТИРОВАНИЯ» посвящена анализу шаблонов проектирования и описанию нового паттерна.

Шаблоны проектирования относятся к многократно используемым или повторяемым решениям, которые направлены на решение аналогичных проблем проектирования в процессе разработки. Для поддержки процесса разработки доступны различные шаблоны проектирования. Каждый шаблон дает основное решение конкретной проблемы. С точки зрения разработчика, шаблоны проектирования создают более удобный дизайн. В то время как с точки зрения пользователя решения, предоставляемые шаблонами проектирования, улучшают удобство использования веб-приложений.

Обычно разработчики используют свой собственный дизайн для данного требования к программному обеспечению, которое может быть новым каждый раз, когда они проектируют для аналогичного требования. Такой метод требует много времени и затрудняет обслуживание программного обеспечения. Адаптация программного приложения к будущим изменениям требований будет затруднена, если во время проектирования не будут приняты соответствующие меры.

Использование шаблона дизайна при разработке веб-приложений может способствовать повторному использованию и согласованности веб-приложения. Без принятия паттернов проектирования или их неправильного выбора разработка веб-приложений станет более сложной, и ее будет трудно поддерживать. Таким образом, знание шаблонов проектирования необходимо для выбора наиболее подходящих шаблонов в веб-приложении. К сожалению, способность разработчиков применять шаблон проектирования не определена. Поощряется хорошая практика использования шаблонов проектирования, и необходимо прилагать усилия для расширения знаний о шаблонах проектирования.

Шаблоны проектирования создания абстрагируют процесс создания экземпляра. Они помогают сделать систему независимой от того, как ее объекты созданы, составлены и представлены. Шаблон создания класса использует наследование для изменения экземпляра класса, при этом шаблон создания объекта делегирует создание экземпляра другим объектам. Шаблоны создания становятся важными, поскольку системы развиваются и больше зависят от композиции объектов, чем от наследования классов. По мере того, как это происходит, акцент смещается с жесткого кодирования фиксированного набора поведений на определение меньшего набора фундаментальных поведений, которые можно объединить в любое количество более сложных. Таким образом, создание объектов с определенным поведением требует большего, чем просто создание экземпляра класса.

В случае веб-приложений иногда необходимо динамически добавлять дополнительные обязанности к веб-странице, чтобы удовлетворить меняющиеся требования. Такие дополнения должны быть прозрачными, не затрагивая другие объекты. Может быть возможность, когда эти обязанности могут быть сняты. В таком сценарии можно использовать шаблон декоратора для добавления дополнительных обязанностей к веб-странице по умолчанию (часть содержимого которой всегда постоянна). Например, если в приложении есть разные пользователи, которые могут войти в него (например, администратор, менеджер программы и заинтересованное лицо). У администратора будет другой пользовательский интерфейс по сравнению с другими пользователями, но некоторые элементы пользовательского интерфейса могут быть одинаковыми для всех пользователей. В таком сценарии можно использовать шаблон декоратора для украшения изменяющихся элементов пользовательского интерфейса в зависимости от типа пользователя, сохраняя при этом оставшиеся элементы постоянными для всех типов пользователей.

В таких приложениях объекты Decorator Pattern могут быть созданы с использованием шаблона создания Abstract Factory. В этой работе конкретные объекты-декораторы создаются с использованием такого шаблона абстрактной фабрики без указания их конкретных классов. Это инкапсулирует процесс создания экземпляра объекта от клиента, что делает систему более гибкой для будущих изменений требований. Дополнительные новые пользователи могут быть добавлены в систему без внесения изменений в код на стороне клиента. Код на стороне клиента скрыт от типа текущего пользователя, созданного шаблоном абстрактной фабрики, что делает систему более многоразовой и простой в обслуживании. Поскольку приложению обычно требуется только один экземпляр конкретной фабрики в абстрактном шаблоне фабрики, использовался шаблон проектирования Singleton для реализации конкретного фабричного объекта.

Шаблоны проектирования программного обеспечения - это проверенные методы проектирования, которые при осторожном использовании при реализации разработки программного обеспечения дают несколько преимуществ. Повторное использование, инкапсуляция и простота обслуживания системы - вот некоторые из преимуществ, которые можно включить в программные системы, если эти проверенные шаблоны проектирования используются в системе. Шаблоны создания относятся к той категории шаблонов проектирования, которые помогают абстрагироваться от процесса создания экземпляров объекта. Они помогают сделать систему независимой от того, как объекты созданы, составлены и представлены. Шаблоны Abstract Factory и Singleton относятся к категории шаблонов создания, которые используются в этой работе для инкапсуляции процесса создания экземпляра объекта шаблона Decorator. Объекты Concrete Decorator создаются соответствующими объектами Concrete Factory в зависимости от типа пользователя, вошедшего в систему. Это привело к повышению производительности приложения при добавлении в систему дополнительных типов пользователей и облегчению поддержки кода приложения для будущих изменений требований.

Были проанализированы такие паттерны как Абстрактная Фабрика и Строитель. Для описания собственного паттерна в первую очередь стоит определить проблемы, которые он будет решать. Так как область применения это диалог, то можно выделить следующие проблемы при стандартной разработке:

- единообразное создание ответов
- ограничение на создание не предусмотренных ответов

Для решения вышеуказанных проблем определим список требований к реализации паттерна:

- делегировать создание ответов в диалоге Фабрике, которая будет отвечать за их создание
- реализовать декларативное создание ответа

- ограничить возможность создания не предусмотренного ответа с помощью иерархии интерфейсов
- инкапсулировать код отвечающий за создание объекта

Фабрика Действий - Порождающий шаблон проектирования представляющий подклассам набор интерфейсов для декларативного определения поведения приложения в ответ на действие клиента. Другими словами данный шаблон определяет набор инструментов для взаимодействия с клиентом.

Третья глава «ПРИМЕНЯЕМЫЕ ИНСТРУМЕНТЫ» посвящена инструментам, которые использовались при разработке и проектирование. Хочется выделить функциональное программирование на языке Java. Новый способ программирования на Java существует уже несколько десятилетий на других языках. С помощью этих возможностей в Java можно писать краткий, элегантный и выразительный код с меньшим количеством ошибок, а также чтобы легко применять и реализовывать общие шаблоны проектирования с меньшим количеством строк кода.

Применив этот инструмент имеем следующие преимущества:

• Не нужно возиться с изменяемыми переменными.

• Шаги итерации скрыты под капотом

• Меньше беспорядка

• Лучшая ясность; сохраняет наше внимание

• Меньшее сопротивление; код четко отслеживает бизнес-намерения

• Меньше подверженности ошибкам

• Легче понять и поддерживать

Четвертая глава «ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ФРЕЙМВОРКА» посвящена реализации фреймворка по описанному во втором разделе паттерну. В этом разделе были представлены иерархии основных интерфейсов отвечающих за действия, конфигурации отвечающие за отношения между

действиями, основной интерфейс DialogFactory, реализация всех вышеописанных интерфейсов и применение фреймворка для формирования ответов на запросы клиента, а так же было приведено сравнение реализации с фреймворком и без.

ЗАКЛЮЧЕНИЕ

В результате данной работы был разработан фреймворк, который сокращает трудозатраты в процессе разработки. Подобный подход будет полезен в долгоиграющей перспективе, так как нужно время, чтобы окупить ресурсы, затраченные на его разработку.

Программа написана на языке Java в среде IntelliJ IDEA. Она не требует конфигурации и готова к компиляции и запуску. В качестве точки входа в приложение, используется rest api. Основными классами, демонстрирующими результат работы являются InteractionService и ResponseService, в которых очевидно преимущество использования фреймворка.

Проанализировав затраты на проектирование и реализацию фреймворка можно смело утверждать, что в крупных, долгоиграющих проектах с большими командами разработчиков, где есть необходимость поддержки кода и демонстрация его заказчику, подобный фреймворк решает множество проблем от архитектуры, до взаимодействия с заказчиком. Таким образом, все задачи, поставленные для достижения цели, решены, и цель достигнута.

Основные источники информации:

1. Фреймворк - важный инструмент программиста. [Электронный ресурс] URL: https://fructcode.com/ru/blog/features-of-popular-frameworks-html-css-php-and-python-frameworks/ Дата обращения: 07.05.21
2. Инверсия и внедрение зависимостей [Электронный ресурс] URL: https://webdevblog.ru/inversiya-i-vnedrenie-zavisimostej/ Дата обращения: 07.05.21

3. Эрик Фримен, Элизабет Фримен, Кэтти Сьерра, Берт Бейтс, Паттерны проектирования, издание. “Питер”, 2018.
4. Ставим объекты на поток, паттерн фабрика объектов [Электронный ресурс] URL: https://habr.com/ru/post/129202/ Дата обращения: 07.05.21
5. Элегантный Builder на Java [Электронный ресурс] URL: https://habr.com/ru/post/244521/ Дата обращения: 07.05.21
6. Функциональное программирование в Java, [Электронный ресурс] URL: https://habr.com/ru/post/122919/ Дата обращения 07.05.21.
7. Мартин Роберт К., Чистый код. Создание, анализ и рефакторинг , издание “Питер”, 2010.