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

Кафедра информатики и программирования

СОЗДАНИЕ КОНТЕЙНЕРА ВНЕДРЕНИЯ ЗАВИСИМОСТЕЙ ДЛЯ JAVA 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
Локтева Ильи Константиновича

Научный руководитель:
Старший преподаватель
кафедры ИиП _____ Лапшева Е.Е.

подпись, дата

Зав. кафедрой ИиП,
к.ф.-м.н., доцент _____ Огнева М.В.

подпись, дата

Саратов 2022

ВВЕДЕНИЕ

На сегодняшний день разработка приложений, в частности больших корпоративных приложений, представляет собой достаточно масштабный и многоступенчатый процесс, требующий проявления высокого уровня ответственности всеми участниками проекта. Приложения, построенные с использованием объектно-ориентированной парадигмы, расширяются в процессе разработки, при этом увеличивается количество классов, взаимосвязанных друг с другом. Ввиду этого, создание экземпляров классов и предоставление им необходимых зависимостей становится достаточно непростой задачей, с которой разработчику может быть достаточно сложно справиться вручную. Решение этой проблемы предлагает принцип «Инверсия управления» и основанный на этом принципе контейнер внедрения зависимостей — фреймворк, которому передается поток управления объектами приложения и их жизненным циклом.

Актуальность настоящей работы подтверждается тем, что в настоящее время тенденция к использованию данного подхода только растет, многие проекты перерабатываются под его использование, либо используют его с самого начала разработки. Также использование этого подхода позволяет обеспечить хорошую архитектуру приложения, упростить понятность и поддержку кода. Также разработанный фреймворк может предоставить альтернативу существующим реализациям, будучи более легковесным, простым в подключении и использовании.

Цель бакалаврской работы: разработка контейнера внедрения зависимостей и демонстрация его использования в работе прикладного Java приложения.

Поставленная цель определила **следующие задачи**:

1. Изучить принцип «Инверсия управления» и его имплементации.
2. Изучить паттерн «Внедрение зависимостей».
3. Изучить теоретическую базу трехслойного приложения.

4. Изучить необходимый для работы инструментарий языка Java, фреймворки и библиотеки.
5. Разработать фреймворк контейнера внедрения зависимостей.
6. Разработать приложение, использующее разработанный фреймворк.
7. На основе приложения провести общее сравнение возможностей разработанного фреймворка с существующей имплементацией Spring Framework.

Методологические основы принципов инверсии управления и использования инструментов языка Java представлены в работах Р. Мартина, Б. Эккеля, Т. Дауни, С. Узайр.

Практическая значимость бакалаврской работы состоит в том, что разработанный фреймворк контейнера внедрения зависимостей может быть использован как альтернатива существующим имплементациям для Java приложений, поскольку он является достаточно легковесным фреймворком, а также прост в изучении и использовании.

Структура и объём работы. Бакалаврская работа состоит из введения, трех разделов, заключения, списка использованных источников и восьми приложений. Общий объем работы – 123 страницы, из них 58 страниц – основное содержание, включая 16 рисунков и 5 таблиц, список использованных источников информации – 27 наименований.

КРАТКОЕ СОДЕРЖАНИЕ РАБОТЫ

Первый раздел «Сведения об используемых принципах и архитектурах» посвящен описанию принципов инверсии управления и внедрения зависимостей. Рассматривается понятие контейнера внедрения зависимостей, его преимущества и проблемы, которые позволяет решить его использование. Также в этом разделе рассматривается трехслойная архитектура приложения.

В пункте 1.1 приводятся необходимые определения объектно-ориентированного программирования, а также рассматривается проблема сильной связанности.

Связанность (англ. coupling) — это мера, в которой модуль (компонент, класс) программы зависит от другого модуля программы (использует какую-то информацию о нем). Связанность может быть слабой и сильной. Слабая связанность часто является признаком хорошей архитектуры приложения.

При разработке классов сильная связанность проявляется в двух основных проблемах:

1) Код методов класса зависит от использования конкретного класса в качестве зависимости.
2) Класс сам ответственен за создание объектов своих зависимостей.

В пункте 1.2 рассматривается способ решения проблем сильной связанности с помощью **инверсии управления** — принципа объектно-ориентированного программирования, при котором поток управления объектами передается некоторой сущности (фреймворку), что позволяет достичь слабой связанности между объектами в процессе жизненного цикла приложения.

Основные особенности приложения, использующего инверсию управления:

- каждый модуль приложения отвечает за одну конкретную задачу;
- каждый модуль максимально независим от других;
- каждый модуль не должен зависеть от конкретной реализации других модулей.

Важную роль в реализации принципа инверсии управления занимает **инверсия зависимостей**, которая подразумевает использование абстракций, например, интерфейсов или абстрактных классов в качестве зависимостей, что позволяет решить первую проблему сильной связанности.

Решение проблемы ответственности класса за создание зависимостей предлагают несколько шаблонов проектирования, среди которых: шаблон «Фабрика»; локатор служб; шаблон «внедрение зависимостей»; контекстуальный поиск зависимостей.

В пункте 1.3 рассматривается **шаблон проектирования «внедрение зависимостей»** и понятие **контейнера внедрения зависимостей**.

«Внедрение зависимостей» — шаблон проектирования, который является одной из наиболее популярных реализаций принципа IoC и предлагает использование внешнего кода для внедрения объектов необходимых зависимостей, в чем заключается его основное преимущество перед другими шаблонами. Остальные шаблоны подразумевают, что класс сам ответственен за получение зависимостей и становится зависим от использования фабрики, локатора служб или интерфейса, что также порождает сильную связанность.

Однако при использовании внедрения зависимостей вручную в программном коде возникают определенные сложности. Самая главная проблема заключается в том, что разработчику приходится самостоятельно следить за передачей необходимых зависимостей в классы.

Для решения этой проблемы внедрение зависимостей поручается **контейнеру внедрения зависимостей** (IoC контейнеру) — настраиваемому объекту, который на основе информации о классах и об их зависимостях самостоятельно создает требуемые объекты, хранит и внедряет нужные зависимости.

Основные преимущества использования IoC контейнера: удобно использовать при юнит-тестировании; уменьшение повторяемости кода, связанного с созданием объектов; упрощается возможность расширения приложения; позволяет достигнуть слабой связанности объектов.

В пункте 1.4 рассматривается понятие трехслойной архитектуры. Каждый слой также описывается подробнее в подпунктах.

Трехслойная архитектура (от англ. three layer architecture) — архитектура, при которой приложение разделяется на 3 отдельных слоя: слой представления (от англ. presentation layer), слой логики (от англ. business logic layer), слой данных (от англ. data layer). Слой представления ответственен за предоставление пользователю интерфейса для работы с приложением. Слой логики обрабатывает пользовательские запросы и работает с данными, получаемыми из слоя данных, а также возвращает их на слой представления в требуемом виде. Слой данных представляет собой некоторое хранилище данных и интерфейс для работы с ним.

Итоги. Изученные в 1 разделе принципы инверсии управления и шаблон проектирования «внедрение зависимостей» являются популярными принципами и позволяют достичь написания грамотно построенного, переиспользуемого программного кода. При этом использование трехслойной архитектуры позволяет логически разграничивать слои приложения, упростить и упорядочить архитектуру классов приложения. Изученные принципы ложатся в основу приложений, разработка которых описана в практической части работы.

Второй раздел «Выбор инструментария для реализации задачи» посвящен обзору инструментов, с помощью которых будут разрабатываться приложения. К ним относятся:

1. **Язык программирования Java** — строго типизированный кросплатформенный объектно-ориентированный язык. Является языком общего значения, использование которого варьируется от написания кода для бытовой электроники до построения больших клиент-серверных приложений.
2. **Java Collections Framework** — унифицированная архитектура для представления коллекций и управления ими независимо от деталей реализации.

3. **Java аннотации —** предоставляют дополнительную информацию о классах и могут рассматриваться как особый вид комментариев, но основное их отличие заключается в том, что аннотации способны изменять поведение программы.
4. **Java Reflection API** — инструмент, предоставляющий возможность исследовать или изменять поведение приложений, работающих на виртуальной машине Java, в процессе их выполнения. Рефлексия является мощным механизмом, позволяющим приложениям выполнять операции, которые были бы невозможны без ее использования.
5. **JDBC (от англ. Java Database Connectivity) —** стандарт взаимодействия Java-приложений с различными СУБД. JDBC API предоставляет программный доступ к реляционным базам данных из программ, написанных на Java, является частью платформы Java и входит в Java SE и Java EE.
6. **Apache Maven —** инструмент для автоматической сборки проектов на основе описания их структуры в специальных файлах на языке POM (Project Object Model) — подмножестве XML.
7. **Apache log4j** — быстрый и гибкий фреймворк для логирования данных, распространяемый под лицензией Apache Software. Данный фреймворк является гибко настраиваемым через внешние конфигурационные файлы. Он использует различные уровни логирования и предоставляет механизмы для вывода логируемой информации в различные базы данных, файлы, консоль и т.д.
8. **Spring Framework** — обеспечивает комплексную программную и конфигурационную модель для современных Java-enterprise приложений. На его базе построено множество вспомогательных фреймворков, которые в сумме составляют единую экосистему. Ключевое значение в рамках данной работы представляет собой Spring IoC контейнер, поскольку он может быть использован как опорная точка для сравнения с создаваемой в рамках данной работы реализацией.

Итоги. Изученные в рамках 2 раздела инструменты позволяют в полной мере решить задачи разработки фреймворка контейнера внедрения зависимостей и разработки трехслойного приложения для демонстрации его работы. Spring Framework позволяет сравнить возможности разработанного фреймворка с собственной имплементацией.

Третий раздел «Создание и использование контейнера внедрения зависимостей» посвящен реализации фреймворка контейнера внедрения зависимостей и приложения для демонстрации его работы. Таким образом, в этом разделе описана разработка двух проектов.

Первый проект, DIProject, представляет собой фреймворк контейнера внедрения зависимостей.

Второй проект, JavaTaskManagerDI подключает DIProject как зависимость с помощью Maven. JavaTaskManagerDI представляет собой таск-менеджер — приложение, позволяющее пользователю создавать, редактировать и удалять задачи, устанавливать дату, до которой задачу требуется выполнить, объединять задачи в группы и проводить поиск задач по различным критериям. Пользователь может зарегистрироваться в приложении, при входе в систему приложение информирует его о задачах, срок выполнения которых скоро истечет.

Проект DIProject использует часть общепринятых терминов из Spring Framework, например, конфигурация, бин и т.д. Для разработки контейнера внедрения зависимостей в проекте DIProject созданы три основных пакета: container, annotations, exceptions. Пакет container содержит все классы, отвечающие за работу IoC контейнера. Пакет annotations содержит аннотации, которые используются для настройки контейнера. Пакет exceptions содержит набор исключений, которые контейнер может выбрасывать разработчику в случае возникновения ошибок, связанных с его работой.

В рамках фреймворка реализованы два основных вида контейнера: ConfigurationClassDrivenContainer и AnnotationDrivenContainer. Класс ConfigurationClassDrivenContainer представляет собой реализацию контейнера,

основанную на конфигурациях. В качестве аргументов конструктора он принимает список объектов типа Class, и с помощью Reflection API просматривает их методы. Класс AnnotationDrivenContainer представляет собой реализацию контейнера, основанную на аннотациях. В конструктор этого класса передается путь к пакету, который необходимо просканировать. Отличительной особенностью AnnotationDrivenContainer является возможность подключения конфигурационных классов, и тем самым, совмещения двух видов контейнеров.

Проект JavaTaskManagerDI построен на основе трехслойной архитектуры. Для хранения данных используется база данных PostgreSQL. Проект состоит из нескольких основных пакетов. Пакеты presentation_layer, business_layer, data_layer содержат классы соответствующих слоев приложения. Пакет exceptions содержит классы исключений приложения. Пакет config содержит конфигурации для работы фреймворка. Пакет utils содержит вспомогательные классы, например, класс для шифрования паролей пользователя.

Пакеты приложения JavaTaskManagerDI содержат множество зависимостей между классами и позволяют использовать его для тестирования и демонстрации работы разработанного фреймворка контейнера внедрения зависимостей.

В пункте 3.4 было проведено сравнение разработанного контейнера с существующей реализацией контейнера в Spring Framework. В рамках базовой функциональности, контейнер может служить альтернативой контейнеру Spring. При этом в разработанном фреймворке инициализация контейнера и операция извлечения объекта выполняются быстрее в силу простоты архитектуры.

Итоги. В рамках третьего раздела разработаны фреймворк контейнера внедрения зависимостей и приложения для демонстрации его работы. Созданный фреймворк обладает преимуществами и недостатками и также был сравнен с существующей реализацией контейнера, поставляемого Spring Framework.

ЗАКЛЮЧЕНИЕ

В результате выполнения работы были решены поставленные задачи и достигнута основная цель. Был изучен принцип «инверсия управления» и способы его имплементации, в частности, шаблон «внедрение зависимостей». Также было проведено ознакомление с трехслойной архитектурой приложения, и инструментами языка Java, среди которых Java Collections Framework, Java Reflection API, JDBC, Java аннотации. Также были рассмотрены инструмент для сборки проектов Apache Maven, библиотека Apache log4j и проведено общее знакомство с Spring Framework.

С использованием изученных принципов и технологий был разработан фреймворк DIProject, представляющий собой контейнер внедрения зависимостей, и проект JavaTaskManagerDI, на котором был протестирован разработанный фреймворк.

Созданный контейнер внедрения зависимостей позволяет улучшить архитектуру приложения, решить проблему сильной связанности между классами, улучшить расширяемость и читаемость программного кода. Он является достаточно простым в изучении и использовании и может быть использован в качестве альтернативы существующим реализациям, например, контейнерам, предоставляемым Spring Framework и Google Guice.

При этом контейнер имеет недостатки, которые могут быть устранены путем дальнейшей разработки и улучшений. Среди них: невозможность использования singleton зависимостей внутри prototype бина, невозможность модификации метаданных бина после его создания, невозможность оптимизации процесса инициализации контейнера, сложность внедрения в веб-приложения.

Основные источники информации

1. Что такое объектно-ориентированное программирование? [Электронный ресурс]. — URL: https://timeweb.com/ru/community/articles/obektno-orientirovannoe-programmirovanie (Дата обращения 18.05.2022)
2. Зацепление, связанность и связность в объектно-ориентированном программировании [Электронный ресурс]. — URL: https://intellect.icu/zatseplenie-zavisimost-i-svyazannost-svyaznost-v-obektno-orientirovannom-programmirovanii-oop-7010 (Дата обращения 18.05.2022)
3. Martin R. Design Principles and Design Patterns / Robert C. Martin // Object Mentor. — 2000. — Vol. 2 — Pp. 4-16.
4. Мартин Р. Чистый код. Создание анализ и рефакторинг / Р. Мартин; пер. с англ. — СПб.: Питер, 2020 — 464 с.: ил. — (Серия «Библиотека программиста»)
5. Эккель Б. Философия Java. 4-е изд / Б. Эккель. — СПб.: Питер, 2015. — 1168 с.: ил. — (Серия «Java»)
6. Uzayr S. Mastering Java / Sufyan bin Uzayr. — Boca Raton, CRC Press, 2022. — 518 p.
7. Downey T. Guide to Web Development with Java. 2nd edn. / T. Downey. — Miami, Springer, 2022. — 517 p.