УДК 004.45

ПРЕИМУЩЕСТВО АБСТРАКЦИЙ В ЯЗЫКЕ JAVA НАД НАСЛЕДОВАНИЕМ, КОНКРЕТНЫМИ КЛАССАМИ И СТАТИЧЕСКИМИ МЕТОДАМИ

Казак А. В., студ., Бизюк А. Н., ст. преп.
Витебский государственный технологический университет,
г. Витебск, Республика Беларусь

Реферат. *Данная работа посвящена анализу подходов в проектировании программного обеспечения с использованием композиции и абстракции, которые предлагают более гибкие и масштабируемые решения по сравнению с традиционным наследованием. Рассматриваются преимущества и недостатки композиции и абстракции, а также их влияние на поддержку, тестируемость и гибкость кода. В исследовании проводится сравнительный анализ этих подходов с классическим наследованием и приводятся примеры из практики, такие как фреймворки Spring, JavaFX и библиотека Mockito.*

Ключевые слова: композиция, абстракция, наследование, ООП, проектирование программного обеспечения, гибкость, тестируемость.

Введение

С развитием программных технологий, сложность программного обеспечения постоянно растёт, и, соответственно, возникает необходимость искать новые инструменты для упрощения разработки и выбора наиболее эффективных подходов. Одним из таких решений стала композиция и абстракция, которые постепенно вытесняют более традиционные методы, такие как наследование.

ООП

Организация классов и их взаимодействие в ООП регулируются основными принципами: композиция, абстракция, наследование, полиморфизм и инкапсуляция. Эти принципы обеспечивают создание гибких и масштабируемых систем. В данном исследовании будут рассмотрены композиция и абстракция, противопоставленные наследованию, как единый подход.

Краткое описание принципов.

– Наследование в Java: наследование позволяет создавать иерархии классов, где дочерние классы унаследуют поведение и свойства родительского.

– Абстракция в Java: абстракция реализуется через абстрактные классы и интерфейсы, служащие основой для обобщения поведения и повышения гибкости системы. Абстрактные классы создают каркас для будущих классов, определяя обязательные для реализации методы, но не реализуя их. Интерфейс – это контракт, который объявляет, что нужно сделать имплементирующему его классу.

– Композиция как принцип связи классов: композиция представляет собой подход, при котором один класс включает в себя экземпляры других классов в качестве своих свойств, что даёт большую гибкость в изменении поведения системы.

Вводная часть к доказательству

Доказательство тезиса о преимуществах композиции и абстракции в проектировании программного обеспечения. В ходе исследования будет проведён сравнительный анализ наследования, композиции и абстракции с учётом их влияния на гибкость, масштабируемость и поддержку кода.

Негативные свойства наследования

Главные причины появления наследования – это необходимость улучшения связей между классами.

1. Наследование выражает связь "is-a" (есть некто).
2. Иерархическая структура.

В небольших проектах наследование работает удовлетворительно, однако по мере масштабирования возникают проблемы.

Основные проблемы наследования.

1. Отсутствие множественного наследования. Пример: водное млекопитающее не может унаследовать одновременно «Водяной» и «Млекопитающее», что приводит к дублированию кода.
2. Ненужные методы в дочерних классах. В больших архитектурах дочерние классы могут наследовать методы, которые им не нужны. Это нарушает принцип подстановки Лисков (LSP), например, если в методе void может вызвать исключение. Такое может происходить также при изменении родительского класса, от которого уже тянется крупная архитектура. Пример: у дельфина появляется метод «спать» из базового класса, хотя у него другой режим сна.
3. Усложнение масштабирования кода. В сервисной архитектуре код быстро разрастается, а наследование не решает проблему различий в поведении.
4. Проблемы с тестированием. Если родительский класс имеет сложную логику, тестирование наследников становится сложнее из-за их высокой зависимости от него. Изменения в родительском классе могут сломать тесты дочерних классов.

Негативные свойства композиции и абстракции

Главные причины появления композиции и абстракции – это необходимость улучшения связей между классами без ущерба гибкости кода. В этом контексте:

– композиция выражает связь "has-a" (имеет что-то).
– абстракция выражает связь "makes-a" (делает что-то).

Композиция и абстракция являются мощными инструментами в Java, позволяя управлять поведением и составными частями без жёстких связей между компонентами. Однако они также имеют свои недостатки.

Основные проблемы композиции и абстракции.

1. Сложность и громоздкость. Часто возникает дублирование кода из-за отсутствия связей с общими реализациями. Если объект содержит множество других объектов и поведений, структура становится трудно отслеживаемой. Пример: разделение поведения на (3–4 интерфейса на класс).
2. Затруднённая отладка. Из-за многочисленных уровней абстракции сложно определить источник ошибки. Информация о реализации скрыта, что усложняет поиск проблем.
3. Высокая сложность проектирования. Абстракция требует глубокого понимания, какие детали реализации следует скрывать и какие интерфейсы предоставлять пользователю, а также объявлять общие черты без конкретных реализаций.
4. Связность с другими классами. Изменения в коде других классов могут негативно сказаться на классе, использующем композицию. Пример: если реализация «Плавник» для класса «Угорь» изменится, это может сломать сам класс, так как он требует строгой структуры «Плавник».
5. Дополнительные сложности в тестировании. Композиция и абстракция увеличивают количество тестов, необходимых для проверки взаимодействия компонентов.

Фреймворки и библиотеки, демонстрирующие переход

1. Spring Framework. Spring изначально использовал подход с наследованием в ряде своих компонентов. Однако в более поздних версиях активно использует инъекцию зависимостей (DI), которая позволяет объединять различные компоненты через композицию, а не наследование.
2. JavaFX. JavaFX, фреймворк для создания графических интерфейсов в Java, также со временем отошёл от наследования и больше ориентируется на композицию и абстракцию. В старых версиях JavaFX часто использовались сложные иерархии классов, но современный подход основан на более простой композиции компонентов.
3. Mockito. Mockito – популярная библиотека для тестирования в Java, активно использующая композицию и абстракцию для создания моков и stubs. Mockito позволяет заменять зависимые компоненты с помощью интерфейсов и абстракций, что даёт большую гибкость.

Вывод

При использовании наследования можно столкнуться с трудностями при проектировании классов. Абстракция, несмотря на необходимость писать больше кода, оказывается более гибким инструментом, что в конечном итоге позволяет сделать систему более масштабируемой и удобной для дальнейших изменений. Современные проекты требуют гибкости и простоты в поддержке, и именно эти качества предоставляет абстракция и композиция. Наследование

накладывает жёсткие ограничения, что делает изменения в коде более сложными и менее безопасными. В большинстве случаев использование композиции и абстракции предпочтительнее, особенно при создании масштабируемых решений. Оптимальный подход – это комбинирование этих методов в зависимости от контекста, с фокусом на снижение связности и улучшение тестируемости кода.

Список использованных источников

1. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гамма [и др.]. – Санкт-Петербург : Питер, 2022. – 366 с.
2. Мартин Р. Чистый код: создание, анализ и рефакторинг / Р. Мартин. – М.: Питер, 2021. – 464 с.
3. Мартин Р. Чистая архитектура. Искусство разработки программного обеспечения / Р. Мартин. – М.: Питер, 2020. – 352 с.

УДК 004

РАЗРАБОТКА FULL-STACK ПРИЛОЖЕНИЙ НА NEXT.JS

Казаков В. Е., к.т.н., доц.
Витебский государственный технологический университет,
г. Витебск, Республика Беларусь

Реферат. *В статье представлен обзор современного фреймворка для разработки веб-приложений. Представлено краткое описание основных особенностей данного фреймворка, позволяющих реализовывать full-stack web-приложение.*

Ключевые слова: full-stack, Next.JS, React, File Based Routing, SSG, SSR.

Next.js – это популярный фреймворк с открытым исходным кодом, построенный на основе React, который позволяет создавать серверные и статические веб-приложения более эффективно. Он расширяет возможности React, добавляя такие функции, как маршрутизация, генерация статического контента (SSG), серверный рендеринг (SSR), API-роуты и другие удобства для разработки.

Рассмотрим основные нововведения более подробно.

Серверный рендеринг означает, что HTML-страница генерируется на сервере при каждом запросе клиента. То есть браузер получает уже готовый HTML с содержимым страницы. При этом пользователю не нужно ждать выполнения JavaScript в своём браузере.

Процесс генерации происходит следующим образом: пользователь делает запрос к серверу; сервер выполняет логику (например, обращается к БД), генерирует HTML и отправляет его клиенту; клиент получает готовую страницу.

Плюсы:

– хорошее SEO – поисковики сразу видят содержимое;
– актуальные данные при каждом запросе;
– отличная производительность: страницы хранятся на CDN и загружаются очень быстро.

Минусы:

– высокая нагрузка на сервер при большом количестве запросов;
– медленнее, чем SSG, так как требуется обработка на лету.

Такой вид генерации применяется в сайтах с динамическим или часто меняющимся контентом; в приложениях, где важны актуальные данные (например, погода, акции, чаты). Способ хорошо работает для статичного или редко обновляемого контента (блоги, документация, портфолио).

Статическая генерация предусматривает генерацию HTML во время сборки проекта и хранится как статический файл, который пересылается клиенту без участия сервера.

При сборке Next.js-приложения предварительно генерирует HTML для всех указанных