Іванчук Я. В., Яковчук П.Л.	ТРЕНДИ РОЗВИТКУ ВИСОКОНАВАНТАЖЕНИХ ІНФОРМАЦІЙНИХ СИСТЕМ	116
Кабанова Є.І., Мельничук В.Е.	DIGITAL TRANSFORMATION OF UKRAINE'S HEALTHCARE SECTOR UNDER WARTIME CONDITIONS: KEY ASPECTS, BARRIERS, AND ADAPTATION STRATEGIES	118
Катеринюк Г. Д.	ЕЛЕКТРОННІ ІНФОРМАЦІЙНІ РЕСУРСИ ЯК ЗАСІБ РЕАЛІЗАЦІЇ STEM-ОСВІТИ ТА РОЗВИТКУ ЦИФРОВОЇ КОМПЕТЕНТНОСТІ УЧНІВ У ПРОЦЕСІ НАВЧАННЯ МАТЕМАТИКИ	120
Кисельова О.Б.	СУЧАСНІ ЦИФРОВІ ОСВІТНІ ІНСТРУМЕНТИ В КОНТЕКСТІ РОЗВИТКУ ЕДУКАЦІЙНОГО СЕРЕДОВИЩА: ПРОБЛЕМИ ТА ПЕРСПЕКТИВИ	124
Кізін Г. О., Ткаченко О. М.	ГІБРИДНА РЕКОМЕНДАЦІЙНА СИСТЕМА ДЛЯ ЕЛЕКТРОННИХ БІБЛІОТЕК НА ОСНОВІ КОНТЕНТНОГО ТА ПОВЕДІНКОВОГО АНАЛІЗУ	128
Кізін Г. О., Ткаченко О. М.	ІНТЕЛЕКТУАЛЬНІ МЕТОДИ УПРАВЛІННЯ ЕЛЕКТРОННОЮ БІБЛІОТЕКОЮ ТА ОПТИМІЗАЦІЯ ПРОЦЕСІВ ОБРОБКИ ДАНИХ	129
Кіпоренко І.Є., Катєльніков Д.І.	АРХІТЕКТУРА СИСТЕМИ ОБРОБКИ КОМАНД У ПОДІЄВО-КЕРОВАНИХ ПРОГРАМНИХ КОМПЛЕКСАХ НА ОСНОВІ ІНВЕРСІЇ УПРАВЛІННЯ ТА МЕХАНІЗМІВ АВТОМАТИЧНОЇ КОНФІГУРАЦІЇ	130

КІПОРЕНКО І.Є., КАТЄЛЬНІКОВ Д.І.
Вінницький національний технічний університет (Україна)

АРХІТЕКТУРА СИСТЕМИ ОБРОБКИ КОМАНД У ПОДІЄВО-КЕРОВАНИХ ПРОГРАМНИХ КОМПЛЕКСАХ НА ОСНОВІ ІНВЕРСІЇ УПРАВЛІННЯ ТА МЕХАНІЗМІВ АВТОМАТИЧНОЇ КОНФІГУРАЦІЇ

*Анотація: Стаття присвячена розробці архітектурного підходу для проектування високоадаптивних систем обробки команд у подієво-керованих програмних комплексах. Досліджено проблему управління зростаючим функціоналом та необхідністю синхронізації з зовнішніми API. Запропоновано рішення на основі інтеграції патерну **Команда** з принципами **Інверсії Управління (IoC)** фреймворка Spring Boot. Обґрунтовано архітектуру, де IoC-контейнер автоматично збирає та інкапсулює логіку команд, а **патерн Слухач** забезпечує асинхронну диспетчеризацію. Ключовим результатом є реалізація механізму **автоматичної конфігурації** та пакетної реєстрації команд у зовнішньому API під час запуску системи, що гарантує її високу модульність, розширюваність та мінімізує операційні витрати на супровід.*

Ключові слова: принцип Інверсії Управління, патерн Команда, патерн Слухач, автоматична конфігурація, подієво-керовані програмні комплекси.

Постановка проблеми

Проектування програмних комплексів, що функціонують у подієво-керованих асинхронних середовищах, таких як комунікаційні платформи: Discord, Slack, Viber, Meta, стикається з проблемою управління зростаючою кількістю функціональних команд. Традиційні підходи, що базуються на централізованій обробці подій та послідовних умовних конструкціях (if-else або switch), призводять до створення монолітних, погано масштабованих та складних для підтримки диспетчерів команд. Виникає необхідність розробки архітектурного рішення, яке забезпечує високу модульність, гнучке додавання нового функціоналу та автоматичну синхронізацію команд із зовнішніми API платформи.

Вирішені завдання дослідження

Для подолання означених проблем було поставлено та успішно вирішено наступні завдання:

1. **Розробка архітектури для декомпозиції функціоналу команд** із використанням принципів об'єктно-орієнтованого проектування та IoC.
2. **Створення ефективного механізму диспетчеризації подій** (Dispatching) на основі патерну **Слухач**, що мінімізує зв'язаність компонентів.
3. **Імплементація механізму автоматичної конфігурації** та реєстрації команд у зовнішньому API під час запуску програмного комплексу.

Суть дослідження

Дослідження базується на інтеграції патерну **Команда (Command Pattern)** з фреймворком **Spring Boot** та його механізмом **Інверсії Управління (IoC)** [1].

1. Модульна архітектура та IoC-контейнер

Функціонал був декомпонований шляхом визначення універсальних інтерфейсів команд (PrefixCommand, SlashCommand). Завдяки механізму **Component Scanning** фреймворка Spring IoC-контейнер автоматично ідентифікує та збирає всі реалізації цих інтерфейсів у відповідні колекції. Це перетворює додавання нового функціоналу з процедури модифікації центрального диспетчера на процедуру створення та розміщення нового класу, що значно підвищує модульність та підтримує принцип розширюваності.

2. Асинхронна диспетчеризація на базі патерну "Слухач"

Центральна обробка вхідних подій реалізована через спеціалізовані **Слухачі (CommandListener)**, які виконують функцію первинного фільтру та парсера. Слухач, після ідентифікації команди, використовує інжектований **Менеджер команд** (універсальний реєстр) для пошуку відповідного об'єкта-команди, після чого делегує йому виконання методу execute(Event) [2]. Цей механізм забезпечує суворе дотримання принципу **єдиної відповідальності (SRP)**, оскільки слухач займається виключно трафіком та маршрутизацією, а обробка бізнес-логіки інкапсульована у відповідному об'єкті команди (рисунок 1) [3].

3. Автоматична конфігурація зовнішнього API та інкапсуляція метаданих

Для вирішення проблеми синхронізації між програмним комплексом та зовнішнім API платформи розроблено механізм автоматичної реєстрації. Кожен клас-команда інкапсулює не лише свою логіку, а й метадані, необхідні для реєстрації у зовнішньому середовищі (наприклад, метод getCommandData()) (рисунок 2).

Сервіс реєстрації автоматично отримує колекцію всіх необхідних команд від IoC-контейнера і використовуючи інкапсульовані метадані ініціює пакетну конфігурацію зовнішнього API платформи під час запуску системи [4]. Це забезпечує актуальність зовнішнього інтерфейсу та мінімізує операційні витрати.

Такий підхід дозволяє уникнути дублювання конфігурацій та гарантує узгодженість між внутрішньою структурою програми та зовнішнім середовищем. Крім того, автоматизована реєстрація зменшує ризик помилок, пов'язаних із ручним оновленням параметрів команд. У результаті система зберігає високу гнучкість і здатність до еволюційного розширення без необхідності втручання у вже існуючі модулі.

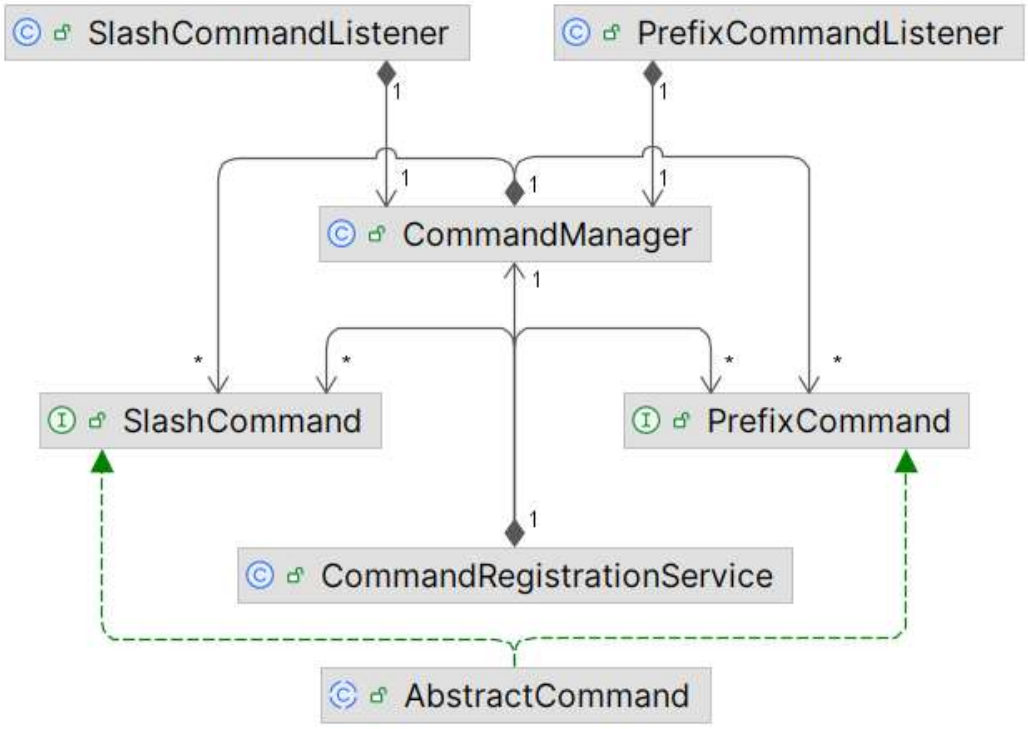


Рисунок 1 – Структурна схема організації команд

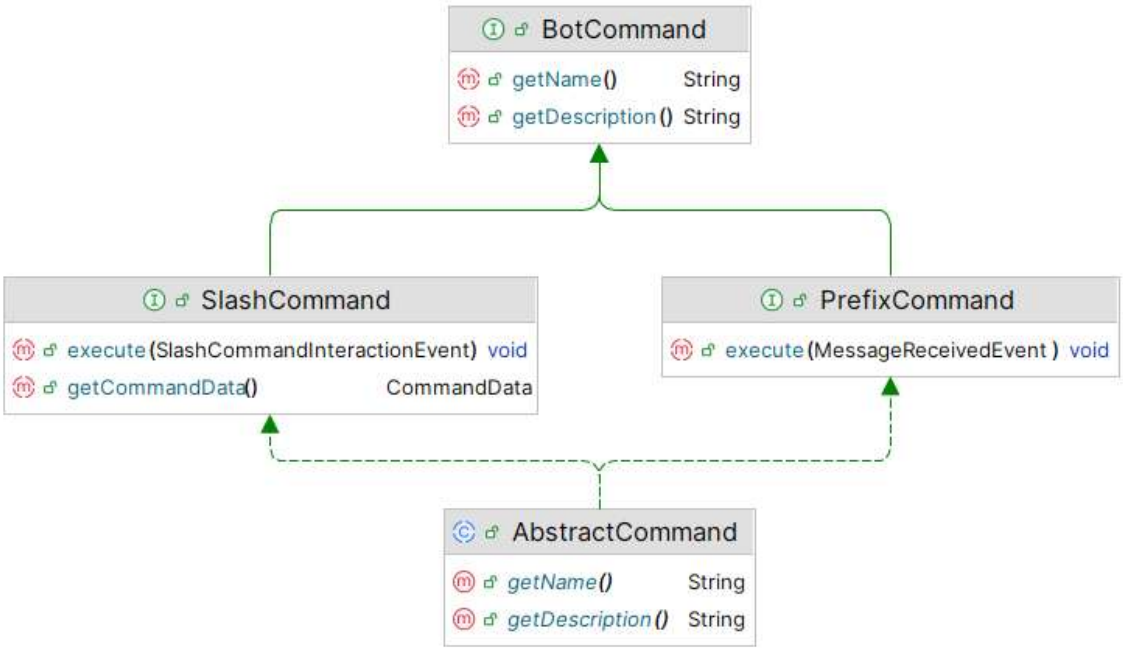


Рисунок 2 – Структура виконуваних команд

Висновки

Розроблене архітектурне рішення забезпечує високу ефективність у проектуванні подієво-керованих програмних комплексів, відповідаючи поставленій задачі:

1. **Підтверджено можливість створення високомодульної системи обробки команд**, де додавання функціоналу не вимагає модифікації існуючих диспетчерів.
2. **Досягнуто архітектурної гнучкості** шляхом успішної інтеграції патерну "Команда" та "Слухач" з принципами IoC, що значно покращує підтримуваність та тестовність.
3. **Імплементовано надійний механізм автоматичної конфігурації**, який гарантує постійну синхронізацію між внутрішнім функціоналом системи та її зовнішнім інтерфейсом, що є критично важливим для промислової експлуатації.

Список використаних джерел

[1] “Spring Framework Documentation :: Spring Framework”. Spring | Home. [Online]. Available: https://docs.spring.io/spring-framework/reference/index.html. [Accessed: Nov. 18, 2025].

[2] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, *Design Patterns. Elements of Reusable Object-oriented Software*. Boston, USA:Addsion Wesley, 1994. 395 p.

[3] R. C. Martin, *Agile Software Development, Principles, Patterns, and Practices*, 2d ed. London: Pearson Education, 2002. 552 p.

[4] R. Martin, *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Hoboken, NJ, USA: Prentice Hall, 2017. 352 p.