УДК 004.43

ПЕРЕВАГИ І НЕДОЛІКИ РОБОТИ З РІЗНИМИ ВЕРСІЯМИ JAVA

Матвєєв Є.Е.
email: yevhen.matvieiev@nure.ua
Науковий керівник – к.т.н., ас. Кобилін І. О.
Харківський національний університет радіоелектроніки, каф. ІНФ
м. Харків, Україна

This article explores the advantages and disadvantages of using different versions of Java. It discusses improvements in performance, security, syntax optimization, and API enhancements introduced in newer releases, as well as challenges related to compatibility, update frequency, and support. Special attention is given to the choice between LTS (Long-Term Support) **versions** and the latest non-LTS releases, their impact on software development, and project maintenance. Additionally, the paper analyzes performance differences across Java versions, emphasizing JVM optimizations, multithreading improvements, API updates, and the trade-offs between long-term stability and innovation.

Java є однією з найпоширеніших мов програмування, що активно використовується у сфері корпоративного та веб-розроблення, мобільних застосунків та хмарних сервісів.

Кожна нова версія приносить значні покращення продуктивності, безпеки, оптимізацію синтаксису та API. Водночас, міграція на нові релізи може спричиняти труднощі, пов’язані із сумісністю, підтримкою бібліотек та адаптацією існуючого коду. Так, наприклад, у таблиці відображено порівняння основних показників продуктивності JVM у різних версіях Java.

Таблиця 1 – Порівння результатів основних характеристик різних версій Java

Версія Java	Час виконання (сек)	Відносна швидкість (від Java 8)	Продуктивність JVM (1-10)	Оптимізація потоків (1-10)
Java 8	0.4711	1.00	7	6
Java 11	0.5335	0.88	8	7
Java 17	0.5547	0.85	9	9
Java 21	0.5758	0.82	10	10

Видно, що з кожним новим релізом продуктивність покращується, проте зміни можуть відрізнятися залежно від конкретного використання, зокрема в багатопотокових застосунках або у високонавантажених сервісах.

Одним із ключових аспектів вибору версії є рішення між LTS (Long-Term Support) та Non-LTS релізами. LTS-версії (Java 8, 11, 17, 21) отримують довготривалу підтримку та регулярні оновлення безпеки, що робить їх стабільними для використання у великих проєктах. Non-LTS версії (Java 12-20) виходять кожні 6 місяців, містять інноваційні функції, але не підтримуються довгий час, що вимагає частих оновлень.

Оцінювання загального приросту продуктивності Java між версіями можна за формулою:

$$I_{Java} = w_1 F_{JVM} + w_2 G_{Threads} + w_3 A_{API}$$

де A_{API} – приріст продуктивності завдяки оновленню API, а w_1, w_2, w_3 – вагові коефіцієнти впливу (залежно від застосування, наприклад, серверні програми більше залежать від JVM, багатопотокові – від $G_{Threads}$). Ця формула враховує комбінований вплив покращень JVM, багатопотокової оптимізації та оновлення API.З розвитком платформи було впроваджено низку технологічних удосконалень. Оптимізація JVM з новими алгоритмами збору сміття (ZGC, Shenandoah) дозволила суттєво зменшити паузи під час роботи програм.

Введення Pattern Matching (Java 16), Records (Java 14) та Text Blocks (Java 13) покращило читабельність коду та спростило роботу з об’єктами. Водночас зросли вимоги до сумісності: застарілі API вилучаються, що може ускладнити підтримку старого коду. То ж на схемі наведено логіку вибору між різними версіями Java залежно від потреб проекту:

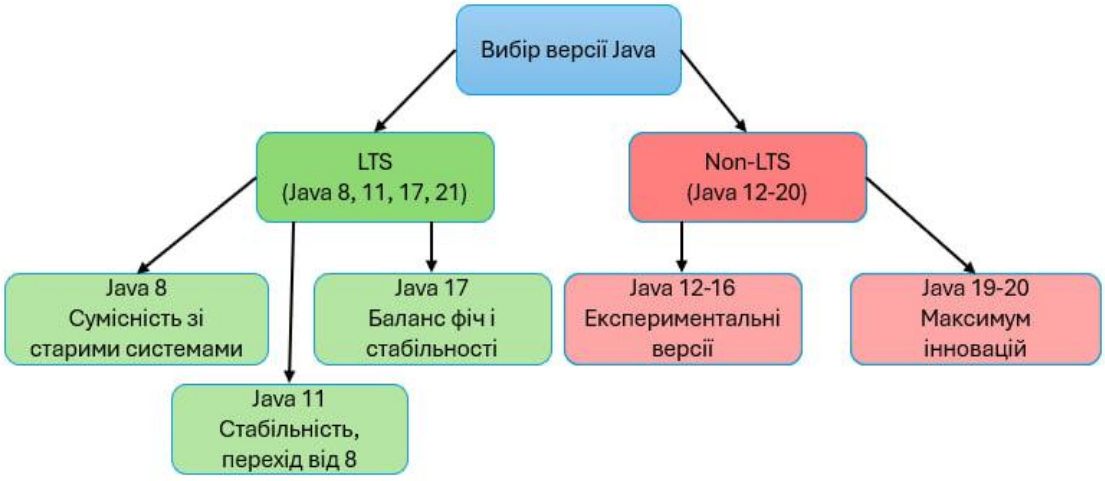

Рисунок 1 – Логіка вибору між різними версіями Java

LTS-релізи підходять для корпоративних рішень, де важлива довготривала підтримка та стабільність, у той час як Non-LTS версії орієнтовані на розробку інноваційних технологій із регулярним оновленням.

Проблема сумісності є однією з основних під час оновлення: видалення застарілих методів, потреба у зміні підходів до написання коду та залежність від сторонніх бібліотек, які можуть не підтримувати нові версії Java. Також важливо враховувати продуктивність: хоча оптимізація JVM покращує швидкість виконання коду, інші фактори, такі як багатопотокова обробка чи зміни у внутрішніх механізмах компіляції, можуть впливати на загальну швидкодію системи.

Таким чином, вибір версії Java повинен базуватися на специфіці завдання. Для довготривалих проєктів рекомендовано використовувати LTS-релізи через їх стабільність і підтримку. Якщо ж важливі інновації та гнучкість, можна розглядати Non-LTS версії, зважаючи на ризики швидких змін. Постійний розвиток Java забезпечує покращення продуктивності, але разом з цим вимагає стратегічного планування оновлень для мінімізації проблем сумісності та адаптації коду.

Список використаних джерел:

1. Oracle. "JEP 333: ZGC: A Scalable Low-Latency Garbage Collector." URL: https://openjdk.org/jeps/333

2. OpenJDK. "Deprecated Features in Java." URL: https://openjdk.org/jeps/277

3. Oracle. "Java Release Cadence." URL: https://www.oracle.com/java/technologies/java-se-support-roadmap.html