Portfolio Project

Sarah LeFlore

Colorado State University Global

CSC450: Programming III

Farhad Bari

03/09/2025

Portfolio Project

For this project, I wrote two different programs, one in Java, and one in C++. Both programs demonstrate multithreading, featuring one loop that counts up, and one loop that counts down. I will be comparing the security and performance of each program. Java is a high-level, memory safe programming language which offers robust security measures. C++ is a mid-level programming language which requires manual memory management, and additional security precautions. Java's memory is managed automatically through its garbage collection system.

Big-O Complexity

Big-O Complexity can be used to approximate the time complexity of a program, it is however not an exact measurement because it does not account for factors such as difference in programming language, network speed, concurrency, or device differences. The C++ and Java programs both feature two loops with fixed variables, so the Big O for each loop is O(1). The C++ program has four additional variables, each equal to O(1). The Java program features four variables and a try and catch statement, all of which evaluate to O(1). So, both programs have a Big O complexity of O(1) constant, meaning that we should expect a fast performance from each.

Latency

For a precise performance comparison, I ran a latency test on each program. For this test, I will be using the same device for consistent results. The program will start counting in microseconds at the start of each program, and then print the final results at the end. For the C++ program, I measured the latency using std::chrono::duration, and the program returned a latency

of 3500.04 microseconds. For the Java program, I tested the latency using System.nanotime(), which returned a latency of 4257 microseconds. So, while both programs performed efficiently, the C++ program performs approximately 17% faster than the Java program.

Performance Comparison

C++ is compiled directly from machine code, whereas Java must first be interpreted (Fetisov, 2023) Because of this difference, C++ is able to execute more efficiently than Java. However, there are other factors which determine the efficiency of a program. A poorly designed C++ program will perform less efficiently than a well designed Java program. For example, a C++ program which has a memory leak will be less efficient than a Java program that doesn't. Also, running either program with concurrent threads would increase the efficiency.

Thread Safety

Threads can be implemented in Java by extending the Thread class. Similarly, threads are implemented in C++ through its thread class. Some of the concurrency issues that can arise in either language are race conditions and dead lock. A race condition occurs when one or more thread attempts to access the same resource at the same time. To avoid race conditions in Java, a ReentrantLock can be used. Alternatively, a thread can be set to wait for the other to finish by using join(); this will prevent race conditions because the threads will not execute concurrently. Similarly in C++, race conditions can be avoided through the use of a lock guard by using std::lock_guard, or by using join().

Dead Lock

Dead lock occurs when one thread is stuck waiting on another thread to unlock. To prevent a deadlock in either program, it is important to release any locks that are declared. This should not be an issue in the C++ program because is uses std::lock_guard, which releases automatically. Conversely, a ReentrantLock must be manually released.

Buffer Overflow

A buffer overflow occurs when a buffer is not able to contain the size of data that is written to it. A buffer overflow can result in a memory leak, a crash, or in the execution arbitrary code (Ballman, 2016). While C++ is vulnerable to buffer overflows, Java is not because it uses automatic bounds checking (Conklin, 2017). To avoid a buffer overflow in C++, a string should be used instead of a char array, because a string is automatically resized. Alternatively, if a char array must be used, then error handling should be implemented to prevent buffer overflow.

Uninitialized Variables

Variables in C++ must be declared before accessing them to prevent undefined behavior (Ballman, 2016). Uninitialized variables don't present a security vulnerability in Java because they will either return an error, or return as null. (Peris, 2025) In spite of this, it is still best practice to always initialize variables in any programming languages to avoid errors.

Conclusion

C++ offers faster performance speed, whereas Java offers more security. Both programs perform efficiently, with C++ outperforming Java by ~17%. Java offers more security features by providing safe guards for buffer overflow, uninitialized variables, and memory management.

So, Java should be used for programs where security is essential, and C++ should be used for programs where speed is essential.

References

Ballman, A. (2016). *SEI CERT C++ Coding Standard*. https://resources.sei.cmu.edu/downloads/secure-coding/assets/sei-cert-cpp-coding-standard-2016-v01.pdf

Conklin, L. (2017). *OWASP Code Review Guide*. https://owasp.org/www-project-code-review-guide/assets/OWASP_Code_Review_Guide_v2.pdf

Fetisov, E. (2023, July 12). *Making the right choice: Java vs. C++ for your 2023 project*. JayDevs. https://jaydevs.com/differences-between-java-and-cpp/

Peris, L. (2025). *Java Error "variable might not have been initialized."* Baeldung. https://www.baeldung.com/java-error-variable-initialized