CSC 450 Portfolio Project

Gregory Norris

CSC 450

Colorado State University Global Campus

CSC 450 Portfolio Project

**Portfolio 1 Program Analysis**

For my portfolio project, I have created a project in Java (1.8) which implements multiple threads to count up to 20 on one thread, then count from 20 to zero on a second thread after the first once has completed. To ensure that the program works as requested, each thread calls upon a function which either counts up, or counts down, and prints both the thread number and the current counter number. To ensure that each thread runs sequentially instead of concurrently, the join() command is used after the start of each thread to ensure that the program pauses until the previous thread has been executed. For the sake of simplicity, I decided to use individually declared threads and runnable method calls vs using an executor service to manage the threads. If I were writing a program which needed lots of parallel processing, I would instead think about using a thread pool and an executor service to schedule tasks out of the thread pool.

**Performance Issues with Concurrency**

As mentioned in my Module 7 milestone, there is a minor performance hit that any program takes when using threads. This is caused by the additional overhead of managing simultaneous running processes and is even more costly in the JVM vs running in C/C++. Where performance increases again is when processes are optimized for multi-thread operation, such as image processing where multiple threads can work on different parts of an image simultaneously without having to wait for any one thread to complete or running into race conditions. With more complex programs, it is possible to run into race conditions or deadlock conditions which can be difficult to troubleshoot and correct. To minimize the potential for a race condition or deadlock, I chose not to share variables between the threads, and encased the threads within a try block to catch Interrupted Exception in the event that one of the threads stalls for any reason. If I had chosen to share data or methods between the threads, I might have chosen to synchronize parts of the code to prevent race or lock conditions. I chose not to due to the partitioned nature of the program, and the additional performance loss of locking and unlocking code using mutex operations.

**Vulnerabilities Exhibited with the Use of Strings**

Thankfully, Java is a lot less vulnerable to exceptions and exploits caused by the use of strings within a program. Java does support the use of C printf style string as of Java 1.5. The use of using formatted strings is not as dangerous as it’s C++ counterpart since most string errors will result in a thrown exception. Burch and Seacord (2007) state that it might be possible for an attacker to take advantage of unhandled or improperly handled exceptions to perform a denial of service attack, but since Java operates within a virtual machine, it does not suffer from the same types of exploits as formatted strings in C/C++ such as printing memory locations or stack information.

The Java program submitted for this portfolio project has very little attack surface in regards to strings since it does not accept input from the user in any form. This program utilizes integer values primarily, although, it does print a string value to the IDE console as proof that the program is functioning as intended.

**Security of the Data Types Exhibited**

This program mostly uses integer data types to perform the requested operations. Since the program does not use input from the user, there is practically no risk of an unhandled integer overflow or truncation, although if the program were altered to include a number larger than max int, there are no specific handlers in place to deal with this issue. Since the program does not utilize a formatted output, there is little danger of an unhandled exception arising from the program output through System.out.print().

**Comparison Between Java and C++ Implementations**

In both implementations, two threads are created which call upon two methods that print out the value of an increasing or decreasing number, then the program quits. I think that both implementations are relatively secure in that they do not accept user data, although only the Java version includes a try / catch block around the threads to handle any unforeseen exceptions regarding thread execution. I tried to make the structure of each as similar as possible, although with the Java version I decided to run and close on thread before running and closing the next thread. In the C++ version, both threads are started simultaneously, and the second thread is prevented from running until the first thread completes. In the C++ program, I chose to add a mutex value around the shared print statement just to see if it was possible, and to show that I could force the program threads to access a program resource in sequence rather than in parallel. This differs from the Java program where the second thread is only started after the first thread has closed.

**Security Analysis**

In general, Java programs are usually more secure than C++ programs due to Java running on the Java Virtual Machine which effectively prevents executable code from being able to access lower level system resources. Also, in Java, the JVM handles the use of pointer functions, which prevents many of the pointer-related exploits found in C/C++. I would consider both programs to be relatively secure since they do not accept external arguments or user input, and while neither program is likely to be completely free of exploits, I would assume that the Java program is slightly more protected due to the JVM.

**Performance Analysis**

When it comes to a simple program like this, I would assume that both Java and C++ are similar in their performance. The main difference would be the hardware resources required for each program. Since Java runs on top of the JVM, there is more overhead and more memory used to initialize the program than would be with C++. Java’s Just-in-time compiler may also require slightly more time at the program start than C++’s pre-compiled executable. Judging from the fastest code benchmarks from The Computer Language Benchmarks Game (n.d.), it is apparent that in almost all cases, C++ is significantly faster than Java and uses much less memory. C++, since it is “closer to the metal” has a lot less overhead and is therefore able to utilize computing hardware more effectively in some cases. The game results shows that C++ is 5x faster than Java when rendering the Mandelbrot set, and approximately twice as fast as java in performing binary tree calculations, while utilizing less than ¼ of the system memory required by Java. From these results, and given the similar project structure between the programs, I can reasonably assume that the C++ application executes faster than the Java application. When it comes to programmer performance, I found that it is generally faster and easier to write code in Java than it is in C++, although the ratio is roughly proportional to the number of the Java programming courses I have taken compared to C++ (5/2).

**Conclusion**

In conclusion, when considering if a program should be written in Java or C++, there are a number of concerns which will guide the final decision. C++ seems to be much faster, and have a smaller footprint than Java, while Java is generally easier to program with and includes more security features due to the separation between the executable code and the Java Virtual Machine. I think that any large performance-minded program is likely to utilize a different number of languages to best leverage the strengths of each where convenient while avoiding their weaknesses when it is crucial to do so.

References

Burch, H., Seaford, R. (2007). Programing language format string vulnerabilities. Web. Retrieved 11/29/19 from: <https://www.drdobbs.com/security/programming-language-format-string-vulne/197002914?pgno=3>

Debian Benchmarks Game Team. Java versus C g fastest programs. (n.d.). Retrieved November 29,2019 from: <https://benchmarksgame-team.pages.debian.net/benchmarksgame/fastest/java-gpp.html>