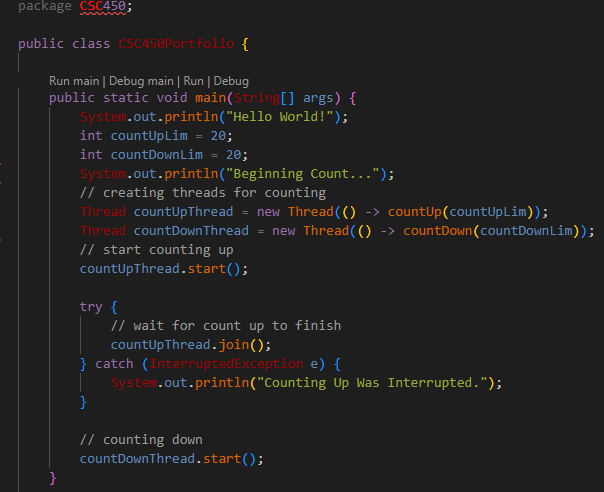
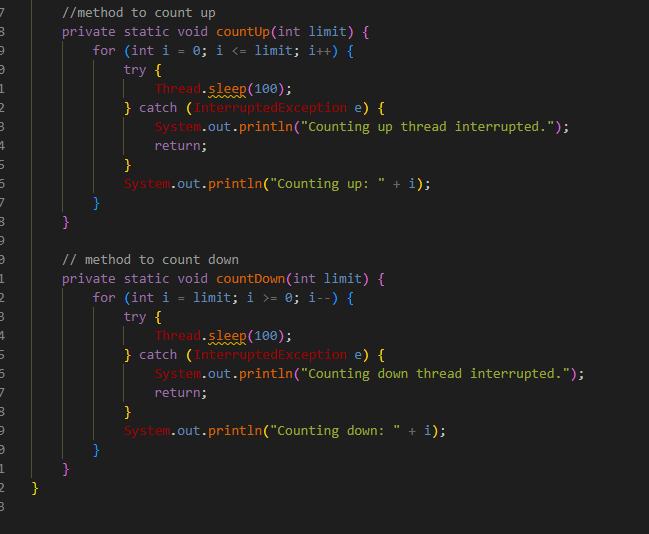
Anthony Le

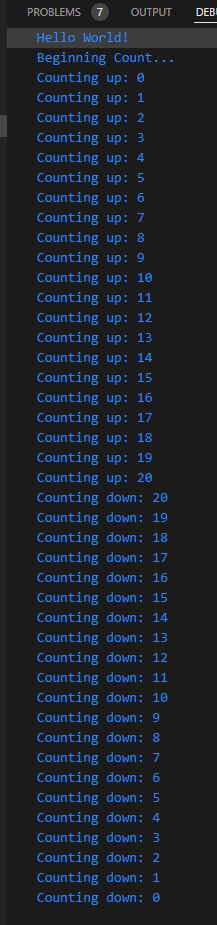
CSC450 - Portfolio

1. Screenshots of Source Code





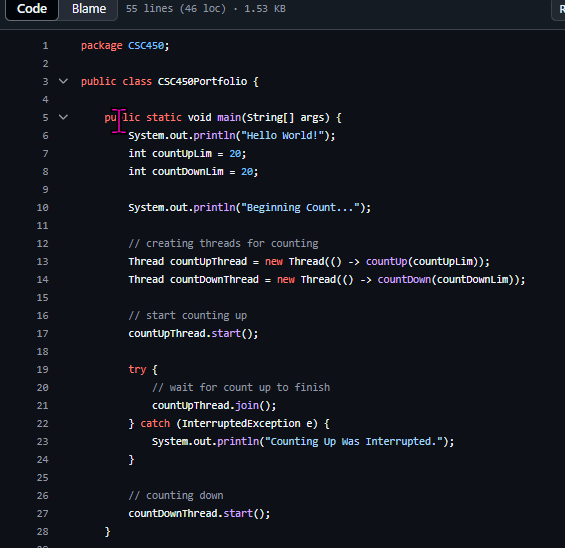
2. Screenshots of program executing

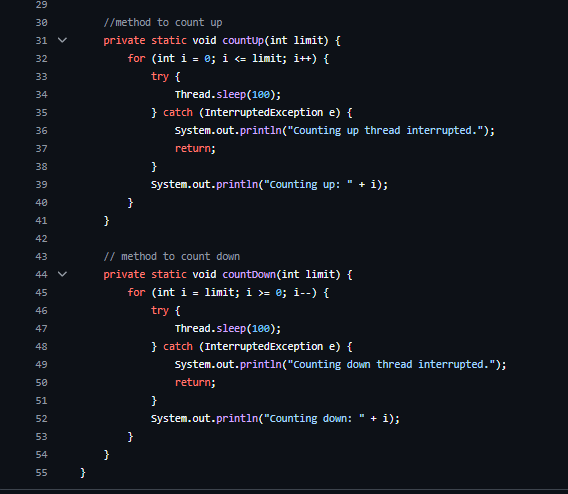


Gif of output will be attached in the zip file, if the gif doesn’t work, try this link:   
https://i.imgur.com/jaSs5UZ.gif

Git repository:  
https://github.com/aale12/CSC450\_Portfolio2

Screenshot of C++ source code on github





Vulnerability Analysis

1. Performance issues with concurrency

Java uses the “join()” method to make thread synchronization easier. This method blocks one thread while the working thread finishes. It requires no further input from the user and is straightforward to use, but may introduce minor amounts of overhead when the second thread is idle and waiting for the first thread to finish completing. However since there are no other tasks in the program, this does not cause any performance issues since there is nothing else for the waiting thread to do anyways. In more complex programs however, an idle thread could reduce efficiency when there are other tasks that need to be done.

2. Vulnerabilities exhibited with use of strings

This program does not use any user inputted strings, so there shouldn’t be any vulnerabilities. However if strings were introduced some vulnerabilities could arise. If the program used a “Scanner”, invalidated input could cause errors such as passing a string when the program expects an integer. A benefit of using Java strings over some other languages like C++ is that Java strings are immutable, meaning they cannot be altered after creation. This immutability protects against buffer overflow vulnerabilities like in languages like C++.

3. Security of the data types exhibited

In Java, integers are not inherently thread safe because there aren’t built in synchronization for primitives. Without proper synchronization for the variable, threads could operate on the data type all at the same time and run into race conditions. In this counting program, the threads are operating sequentially due to the use of the “join()” method. Because of this, there is minimal risk of race conditions in the counting program.

References

GeeksforGeeks. (2016, January 9). *Multithreading in Java - GeeksforGeeks*. GeeksforGeeks. https://www.geeksforgeeks.org/multithreading-in-java/

Group, I. S. R. (n.d.). *What is memory safety and why does it matter?* Prossimo. https://www.memorysafety.org/docs/memory-safety/

*java.util.concurrent Package*. (2020, October 31). GeeksforGeeks. https://www.geeksforgeeks.org/java-util-concurrent-package/

*Memory Management in Java - Javatpoint*. (2011). Www.javatpoint.com. https://www.javatpoint.com/memory-management-in-java

*Memory-safety in Java – You are likely to be eaten by a grue!* (2023, November 30). https://www.novatec-gmbh.de/en/insights/blog/mmory-safety-in-java/

*Strings in Java vs. Strings in CPP: A Comparative Analysis*. (2024, November 29). @Upgrad. https://www.upgrad.com/tutorials/software-engineering/java-tutorial/strings-in-java-vs-strings-in-cpp/