# Module 8 Critical Thinking: Portfolio Project

Fadumo Abdalla

Colorado State University Global

CSC450-1

Professor Reginald Haseltine

2/11/24

**Source code:**

public class CounterApplication {

private static final Object lock = new Object();

private static boolean reachedTwenty = false;

static class CountUpThread extends Thread {

public void run() {

for (int i = 1; i <= 20; i++) {

synchronized (lock) {

System.out.println("Count Up: " + i);

if (i == 20) {

reachedTwenty = true;

lock.notifyAll();

}

}

try {

Thread.sleep(100);

} catch (InterruptedException e) {

Thread.currentThread().interrupt();

}

}

}

}

static class CountDownThread extends Thread {

public void run() {

synchronized (lock) {

while (!reachedTwenty) {

try {

lock.wait();

} catch (InterruptedException e) {

Thread.currentThread().interrupt();

}

}

}

for (int i = 20; i >= 0; i--) {

System.out.println("Count Down: " + i);

try {

Thread.sleep(100);

} catch (InterruptedException e) {

Thread.currentThread().interrupt();

}

}

}

}

public static void main(String[] args) {

Thread countUpThread = new CountUpThread();

Thread countDownThread = new CountDownThread();

countUpThread.start();

countDownThread.start();

}

}

Pseudocode

Define CounterApplication

Initialize lock object for synchronization

Initialize reachedTwenty flag as false

Define CountUpThread as Thread

Run method:

For i = 1 to 20

Synchronize on lock

Print "Count Up: " + i

If i equals 20

Set reachedTwenty to true

Notify all threads waiting on lock

Sleep for 100 milliseconds

Define CountDownThread as Thread

Run method:

Synchronize on lock

Wait while not reachedTwenty

Wait on lock

For i = 20 down to 0

Print "Count Down: " + i

Sleep for 100 milliseconds

Main program:

Start CountUpThread

Start CountDownThread

Github:

Performance issues with concurrency:

Concurrency in Java applications can lead to performance issues due to contention for shared resources, context switching, and synchronization overhead. In the provided code, synchronized blocks are used to ensure mutual exclusion and coordination between the two threads. However, excessive synchronization can lead to contention and reduced parallelism. To mitigate performance issues, developers can use finer-grained locking strategies, such as using Lock objects with explicit locking and unlocking, to minimize the duration of critical sections and reduce contention (Goetz, 2006).

Vulnerabilities exhibited with the use of strings:

The provided code does not directly involve string manipulation, so there are no vulnerabilities associated with strings in this context. However, when dealing with strings in Java applications, developers should be mindful of potential vulnerabilities such as buffer overflows, injection attacks, and format string vulnerabilities. Secure coding practices, such as input validation, proper sanitization, and using parameterized queries for database interactions, can help mitigate these vulnerabilities (Seacord, 2011).

Security of the data types exhibited:

In this application, the primary data type used is int for counting. Java's built-in data types, including int, are generally secure from a data type perspective. However, developers should be cautious when dealing with user inputs and external data sources to prevent potential security vulnerabilities such as integer overflow, data truncation, and injection attacks. Proper input validation, data validation, and sanitation techniques should be employed to ensure the security and integrity of the application (Java Secure Coding Guidelines, 2019).

Conclusion:

While the provided Java application effectively demonstrates concurrency concepts, it is essential to address potential performance issues, vulnerabilities associated with string manipulation, and ensure the security of data types when developing concurrent applications in Java.

References:

Goetz, B. (2006). Java Concurrency in Practice. Addison-Wesley.

Seacord, R. C. (2011). Secure Coding in Java: Java Security. Addison-Wesley.

Java Secure Coding Guidelines. (2019). Retrieved from https://cwe.mitre.org/data/definitions/477.html