Portfolio Project – Part 2

Concurrency Concepts

Arianna Martinez

CSU – Global

CSC450

Professor Bari

November 5, 2023

Source Code

package cscAM;

public class ConcurrencyConcepts {

public static void main(String[] args) {

Thread threadOne = new Thread(() -> { // module 5

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

System.out.println("Thread-1: " + i);

try {

Thread.sleep(500); // basic delay

} catch (InterruptedException e) {

e.printStackTrace();

}

}

});

Thread threadTwo = new Thread(() -> {

try {

threadOne.join(); // the join will make it so that it waits for Thread-1 to finish.

} catch (InterruptedException e) {

e.printStackTrace();

}

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

System.out.println("Thread-2: " + i);

try {

Thread.sleep(500);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

});

threadOne.setName("Thread-1");

threadTwo.setName("Thread-2");

threadOne.start();

threadTwo.start();

}

}

Output

A screenshot of a computer

Description automatically generated

Program Analysis

Performance issues with concurrency –

For programmers, it is very fundamental to understand the concurrency of work/data flow. If it is not effectively managed, applications may experience performance problems or shutdowns. Two threads are used in the provided program to count up to 20 and then down to 0. Even if the counting process doesn't use a lot of resources, it is good to look out for any possible speed bottlenecks. To increase processing speed and prevent needless delays in such situations, it is very important to optimize the code, reduce resource contention, and take into account the use of thread pooling, multilevel threading, or other concurrency control methods.

Vulnerabilities exhibited with the use of strings –

On its own, the application's use of strings, especially when reporting thread names and count values, does not pose a serious risk. But when working with user input or outside data that is used directly as strings, it's particularly important to take security risks into account. Poor validation and cleanup of user-supplied or external data by the program may result in vulnerabilities like cross-site scripting or injection attacks. Implementing input validation, sanitization, and appropriate techniques are key for reducing these risks and guaranteeing that user-supplied or external data utilized as strings do not pose as dangers to the security of all that information.

Security of the data types exhibited –

The way that different data types are handled and utilized by the software determines how secure they are when used in the application. Although there are no overt security issues with the data types utilized in the provided code, it is just as important to take into account any potential security dangers that may arise in real-world situations. Security flaws can be introduced, for instance, thanks to buffer overflows, string manipulation, or data type inconsistencies. It is going to be good to know how to validate and sanitize that user input, manage data conversions, and type casting correctly, and adhere to safe coding techniques in order to guarantee the security of data types. So, developers can guarantee the overall security of the program and avoid typical vulnerabilities related to data types by putting these preventative measures to work.