In the Java program that shows concurrency, speed is mostly affected by the extra work that comes with switching between contexts, competing for locks, and threads that need to synchronize with each other. When synced blocks are present, threads have to wait for others to release the lock. This could cause CPU resources to be underused. Another thing is that one thread is waiting for another thread to tell it something, which might seem wasteful, especially when the wait time is much longer than the active working time. These errors aren't very big in our simple counter application, but they could get big in more complicated systems with a lot of threads running at the same time and a lot of synchronized blocks.

When it comes to strings and possible security holes, Java's design of "immutable strings" makes concurrent apps safer by default. Since strings can't be changed after they've been made, they don't cause race conditions, which is a common problem when shared data can be changed while it's being used. There are, however, problems with string interning. Having a lot of different strings could cause memory leaks, and private data stored in strings could be visible for longer than needed, which could cause security issues in memory dumps. Handling strings with private information should be done with care to avoid these kinds of problems.

Primitive types and secure object locking methods are used in Java programs to keep data types safe. The `int` primitive type for the counter is a secure choice, as primitives are kept on the stack and are not susceptible to external reference manipulation. The class protects the lock object that is used for syncing from unauthorized access that could cause deadlocks or states that don't match up. The app makes sure that threads are safe by controlling who can access the shared state and keeping the security of shared data between threads. It is very important to carefully control how threads interact with each other and shared resources in order to keep the performance and security of Java applications that use concurrency.

**Screenshot:**

A screenshot of a computer

Description automatically generated

**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();

}

}

**Github Repository:**

https://github.com/diriyeabdalla/CSC450-Portfolio