**Program Analysis:**

This program is sequential in nature and therefore does not need to be executed on multiple threads. This will cause the JVM to hog up more CPU on the users device un-neccessarily meaning this program is highly inefficient. Therefor it would make more sense to execute the logic on of this application from a single thread.

There are no vulnerabilities exhibited with the use of strings in this application. The reason being all strings are hard coded, and have no easy means to manipulate. The data type we use is an int. It is iterated through the means of a for loop, one incrementing and the other decrementing. This program accepts no inputs and only writes the count to the std out. Therefor its data is not vulnerable to user input, and it would be impractical to attack this program. We also create two threads to execute each count thread. The iteration of the for loops in each thread use locks to synchronize the execution of the counts, so there are no issues with concurrency in this application.

**Pseudo Code:**

class Main {

final int MAX = 20;

Lock lock = new Lock();

Thread CountUpThread {

run() {

lock.lock();

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

print("Counting up: " + i);

}

lock.unlock();

}

}

Thread CountDownThread{

run() {

lock.lock();

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

System.out.println("Counting down: " + i);

}

lock.unlock();

}

}

main() {

CountUpThread t1 = new CountUpThread();

CountDownThread t2 = new CountDownThread();

t1.start();

t2.start();

}

}

**Src Code:**

/\*\*

\* Alec Mcdaugale

\* CSC450-mod8-concurrency

\* Brief:

\* This application should create two threads that will act as counters. One thread should count up to 20. Once thread

\* one reaches 20, then a second thread should be used to count down to 0.

\*/

import java.util.concurrent.locks.ReentrantLock;

/\*\*

\* Main Class.

\*/

public class Main {

/\*\*

\* Max value to count up to.

\*/

private static int MAX = 20;

/\*\*

\* Lock prevent threads from counting concurrently.

\*/

private static final ReentrantLock lock = new ReentrantLock();

/\*\*

\* A class to count up on a thread.

\*/

private static class CountUpThread extends Thread {

public void run() {

// Hold lock to block another thread from executing.

lock.lock();

try {

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

System.out.println("Counting up: " + i);

}

} finally {

// Release thread to allow the next thread to execute.

lock.unlock();

}

}

}

/\*\*

\* A class to count down on a thread.

\*/

private static class CountDownThread extends Thread {

public void run() {

lock.lock();

try {

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

System.out.println("Counting down: " + i);

}

} finally {

lock.unlock();

}

}

}

public static void main(String[] args) {

CountUpThread t1 = new CountUpThread();

CountDownThread t2 = new CountDownThread();

// These threads will execute concurrently, this will test if locks work.

t1.start();

t2.start();

}

}

**Screen Shot:**



**GitHub:**

<https://github.com/amcdaugale/Programming3/tree/main/module8>

