Quiz # 1 CS361
Fall 2020 Concurrent Programming

Your name: Dennis George

Time: 25 minutes

1. We have two integer arrays T and B, both with a predetermined constant N number of entries. We want to figure out the ratio -- the sum of values in T divided by the sum of values in B. We can assume that we are given lines of code that look something like this

integer T[] = {10057, 3, 3, 2, 190365, …. } integer

B[] = {10058, 100058, , …. }

which we can just copy into the program. (If you think this is unrealistic for large N, you can just assume that there's some initialization procedure that fills T and B that work with the code you're asked to write here.)

Write a Java program that does this correctly, using the "give object that implements Runnable to thread constructor" method of creating threads. (You can just sketch out without writing fully the line of code that initializes the array). If you need to, you may use join() but no other synchronization methods (no locks, volatile variables, semaphores, monitors, use of sleep, yield, etc.). Both threads must contribute work towards computing the sum of both the values of T and the values of B. In other words, you can't split the work by having one thread just sum T and the other just B. [This is not an arbitrary constraint whose only purpose is to make you think harder about your answer. It would also allow your design to easily scale to handle situations where you want a lot more than two threads because the number of values to treat is truly titanic.]

The ground rules of this quiz are that you can use materials you have access to in order to construct your answer. But remember that the course attribution policy applies to this on-line quiz. We aren't going to spy on you. But if you get information about how to create your answer from any source other than the course textbook and lecture notes**, you must give a citation to it**. This includes internet sources, classmates in chat conversations, chegg, etc. For example, you are generating your answer by modifying code you got from another source (even your own notes), you are obligated to explain this in your submission. We do reserve the right to give only partial credit if it appears that too much of the thought that goes into construction of the answer was done by

someone other than you. But if you don't provide attribution, you will be penalized for an academic honesty violation. Be honest in what you turn in.

Scoring: 10 points – threads launched correctly, proper use of start and join, sharing data but not a lot of contention over shared data (a lot of concurrent writes to the same memory location), and of course correct computation without error-causing race conditions despite the ban on locks or other synchronization devices beyond the use of join. We care mainly about the part of your code that does the parallel computation and its synchronization.

9 points – slight flaw or oversight in any of these criteria.
8 points – significant but easily fixable flaw (or perhaps two or three)
7 points – one significant but not easily fixed flaw, such as not creating threads correctly, concurrency with error-causing race conditions, or coordination such that threads don't start until other finish.
6 points – several significant and some not easily fixed.
5 points – a significant attempt but far from working 0
points – little or no effort.

```
class WorkThread implements Runnable {
    int[] chunk;
    int sum = 0;
    public WorkThread(int[] chunk){
        this.chunk = chunk;
    }

    @Override
    public void run() {
       for (int i = 0; i < this.chunk.length; i++)
           this.sum += this.chunk[i];
    }
}
public class Main {

    public static void main(String[] args) {
        int[] T = {1,2,3,4,5};
        int[] B = {1,2,3,4,5};

        int numThreads = 2;
        int tSum = 0;
        int bSum = 0;
        WorkThread[] threads = new WorkThread[2];
        boolean finished = false;
        int chunkSize = 2;
        /*
            The main idea is to create a working thread that
knows how to sum up any chunk that is given to it
            The driver program is responsible for dividing the
work up until things are finished
            We will remove elements from T and B as they are sent
to threads. When both T and B are empty,
            we know our calculations are finished.
            We store the sum as a part of the thread
implementation
            Both threads can run on a chunk from either array.
         */
        while (!finished) {
            for(int i = 0; i < numThreads; i++){
                // get chunk for new thread
```

```
                // remove elements from T and B as they are
calculated so we can finish when T and B are empty
                threads[i] = new WorkThread(chunk);
            }

            for(WorkThread thread : threads) {
                Thread t = new Thread(thread);
                t.start();
                t.join();
            }
            tSum += threads[0].sum;
            bSum += threads[1].sum;
            if (B.length == 0 & T.length == 0) finished = true;
        }

        double ratio = tSum / bSum;
        System.out.println("Ratio: " + ratio);
    }
}
```

As it stands, this wouldn’t run. The code for creating the chunks is not finalized.