Lab 2

Jan Salama 213769151

Mias Ghantous 213461692

Question 1:

The method run() simply executes the run method on the current thread whereas the method start() starts a new thread and executes run on it

If we wanted to run a thread parallel to the main thread we need to use start().

Question 2:

1.
   ```
   Hello world from thread number5
   Hello world from thread number8
   Hello world from thread number9
   Hello world from thread number2
   Hello world from thread number1
   Hello world from thread number6
   Hello world from thread number3
   Hello world from thread number0
   Hello world from thread number4
   Hello world from thread number7
   That's all, folks
   ```

2.
   ```
   That's all, folks
   Hello world from thread number2
   Hello world from thread number6
   Hello world from thread number1
   Hello world from thread number8
   Hello world from thread number7
   ```

```
Hello world from thread number3
Hello world from thread number9
Hello world from thread number0
Hello world from thread number5
Hello world from thread number4
```

After removing the loop that joined the threads, the main thread did not wait for the other threads to terminate and immediately continued, that is why we see “That’s all, folks” at the start this time.

3.

   ```
   Hello world from thread number0
   Hello world from thread number1
   Hello world from thread number2
   Hello world from thread number3
   Hello world from thread number4
   Hello world from thread number5
   Hello world from thread number6
   Hello world from thread number7
   Hello world from thread number8
   Hello world from thread number9
   That's all, folks
   ```

   Since we joined after every start(), that means the main thread waited for each thread it started to terminate before ending the loop iteration, that’s why we see the outputs in order.
   No thread was started before the one that was created earlier finished running.

4. Thread.currentThread.join() is an error, essentially we are making a thread wait for itself to stop running, resulting in endless waiting.

Question 3:

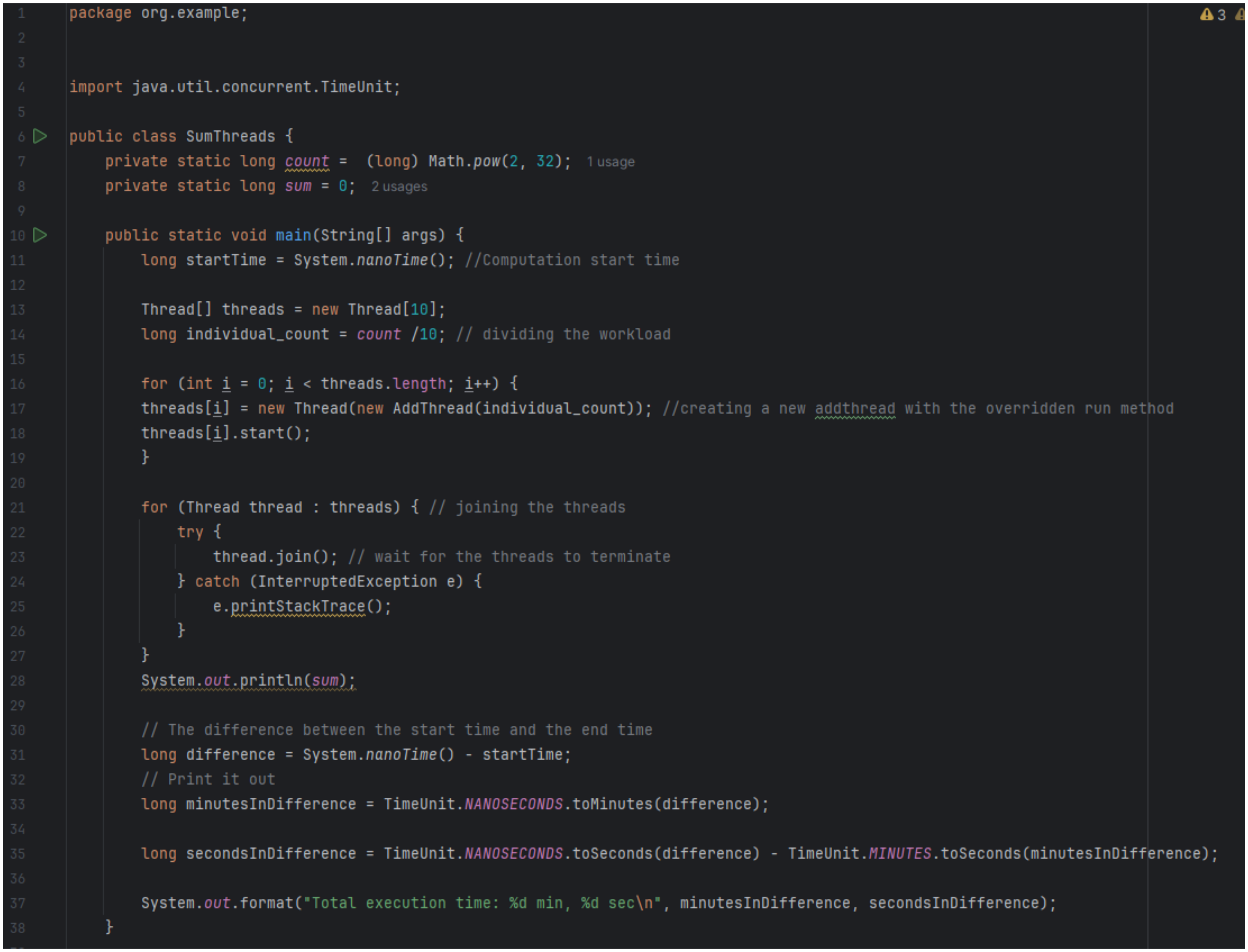
1. The output of the normal Sum class:
   ```
   4294967296
   Total execution time: 0 min, 1 sec
   ```

   The output of SumThreads:
   ```
   4294967290
   Total execution time: 0 min, 0 sec
   ```

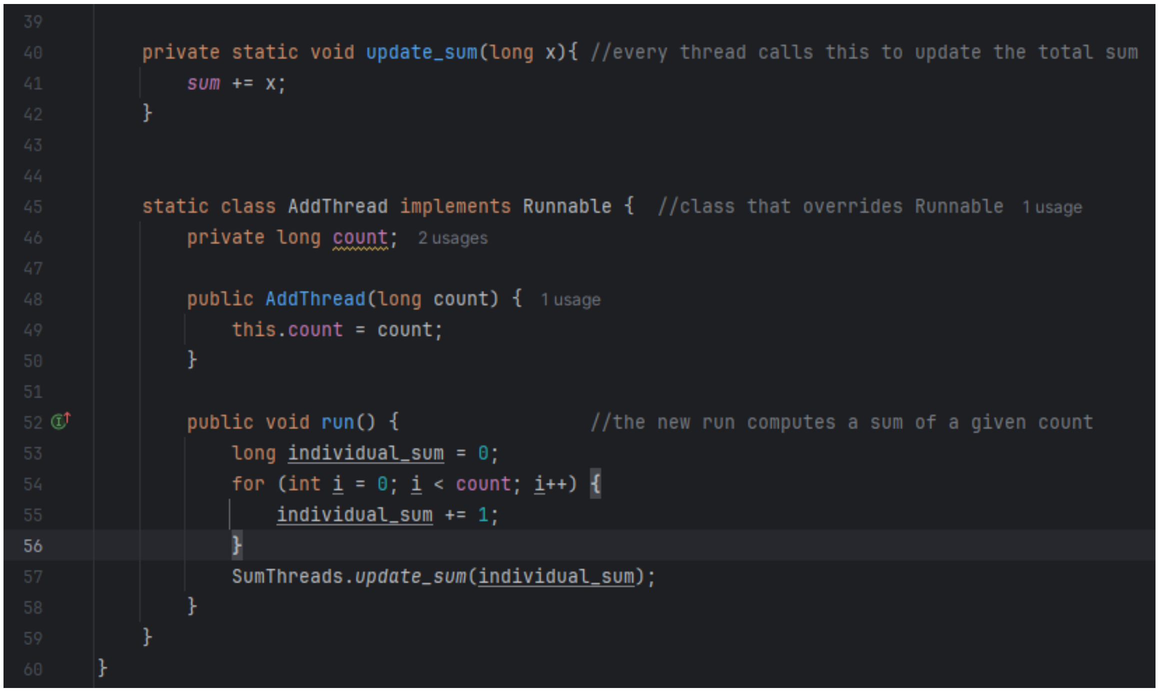
   These results were consistent over numerous running attempts, showing clearly that the multi-threader approach was advantageous.

2. Since the execution time showed 0.0 consistently, it appears as though the run time was consistent.
   However, this is because the run time is so low.
   In general when working with multiple threads run time usually differ based on how the cpu allocated runtime to each thread, the amount of overhead from context switching, and many other differences in execution.
   This is clear after completing question 2, as every run of the original code yielded a different order of thread outputs.

3. The code of SumThreads:



```java
package org.example;


import java.util.concurrent.TimeUnit;

public class SumThreads {
    private static long count =  (long) Math.pow(2, 32);
    private static long sum = 0;

    public static void main(String[] args) {
        long startTime = System.nanoTime(); //Computation start time

        Thread[] threads = new Thread[10];
        long individual_count = count /10; // dividing the workload

        for (int i = 0; i < threads.length; i++) {
        threads[i] = new Thread(new AddThread(individual_count)); //creating a new addthread with the overridden run method
        threads[i].start();
        }

        for (Thread thread : threads) { // joining the threads
            try {
                thread.join(); // wait for the threads to terminate
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        System.out.println(sum);

        // The difference between the start time and the end time
        long difference = System.nanoTime() - startTime;
        // Print it out
        long minutesInDifference = TimeUnit.NANOSECONDS.toMinutes(difference);

        long secondsInDifference = TimeUnit.NANOSECONDS.toSeconds(difference) - TimeUnit.MINUTES.toSeconds(minutesInDifference);

        System.out.format("Total execution time: %d min, %d sec\n", minutesInDifference, secondsInDifference);
    }
```

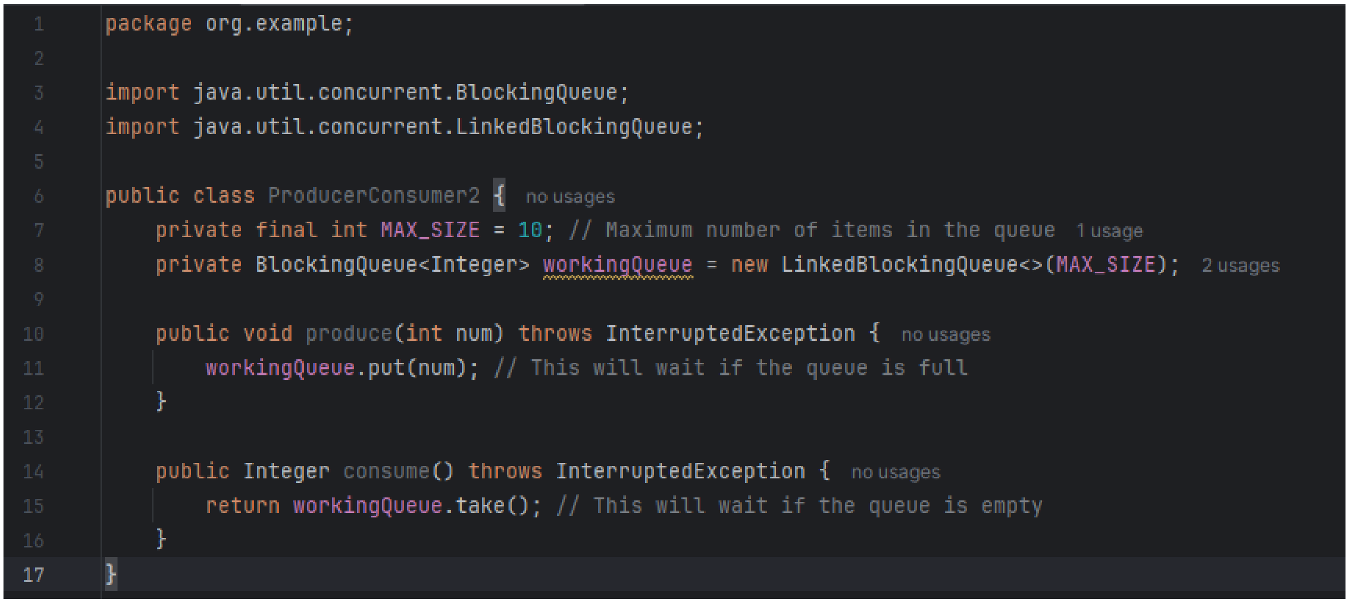


```

    private static void update_sum(long x){ //every thread calls this to update the total sum
        sum += x;
    }


    static class AddThread implements Runnable {  //class that overrides Runnable  1 usage
        private long count;  2 usages

        public AddThread(long count) {  1 usage
            this.count = count;
        }

        public void run() {                 //the new run computes a sum of a given count
            long individual_sum = 0;
            for (int i = 0; i < count; i++) {
                individual_sum += 1;
            }
            SumThreads.update_sum(individual_sum);
        }
    }
}
```

Question 4:

After reading part 7 in the lab pdf file I decided to read into and use the BlockingQueue collection.

This is my new code:



```java
package org.example;

import java.util.concurrent.BlockingQueue;
import java.util.concurrent.LinkedBlockingQueue;

public class ProducerConsumer2 {
    private final int MAX_SIZE = 10; // Maximum number of items in the queue
    private BlockingQueue<Integer> workingQueue = new LinkedBlockingQueue<>(MAX_SIZE);

    public void produce(int num) throws InterruptedException {
        workingQueue.put(num); // This will wait if the queue is full
    }

    public Integer consume() throws InterruptedException {
        return workingQueue.take(); // This will wait if the queue is empty
    }
}
```

Lab Exercise 2:

After writing both the normal sum and the threaded sum classes, and measuring the time via the calendar utility as instructed.

Both classes shows a run time of 0 seconds.

Even when I tried adding milliseconds to the calendar, they show 0-millisecond runtime.

**Side-Note: Since this exercise resembles question 3, I used the same classes I used there and modified them according to the new demands.