Anthony Borza

Professor Tarquinio, Michael

Due Date: September 3, 2017

Homework 2

Table of Contents

[Objective: 3](#_Toc491790291)

[IOIntensive Class: 3](#_Toc491790292)

[Computationally Class: 3](#_Toc491790293)

[Controller Class: 3](#_Toc491790294)

[Screenshots: 4](#_Toc491790295)

[First: 4](#_Toc491790296)

[Second: 5](#_Toc491790297)

[Lessons Learned: 5](#_Toc491790298)

# **Objective:**

The purpose of this assignment is to demonstrate an understanding for how to create and use threads effectively. The following program that I created includes 4 classes: IOIntensive class, Computationally class, Controller class, and a Time class. I will describe the functionality of each class below.

## **IOIntensive Class:**

The IOIntensive class is designed to iterate through five threads 100 times, starting at 0 and ending at 100. It also allows for the ability to suspend, wait, and resume threads. For example, if theread1 is suspended at 10, all the other threads: thread2, thread3, thread4, thread5 will continue, but thread1 will not continue until it is told to resume. When thread1 resumes, it will pick up at number 11. This class also prints the time each thread started and finished.

## **Computationally Class:**

The Computationally class has 4 additional inner classes called: Sum10, Sum25, Sum50, and Sum75, and they were created to perform many computational operations. The first one, Sum10 performs the sum of all numbers 1 to 10. The second one, Sum25 performs the sum of all numbers from 0 to 25. The third one, Sum50 performs the sum of all numbers from 0 to 50. The fourth one, Sum75 performs the sum of all numbers from 0 to 75. The fifth and final one that is defined in the outer main class, Computationally performs the sum of all numbers from 0 to 100. For the sum to be calculated, each thread: thread1, thread2, thread3, thread4, and thread 5 must reach 100 first. The first five threads are used in the IOIntensive class. Once each thread reaches 100, thread6, thread7, thread8, thread9, and thread10 of the Computationally class are started, and display the sums. Therefore, there is a total of 10 threads.

## **Controller Class:**

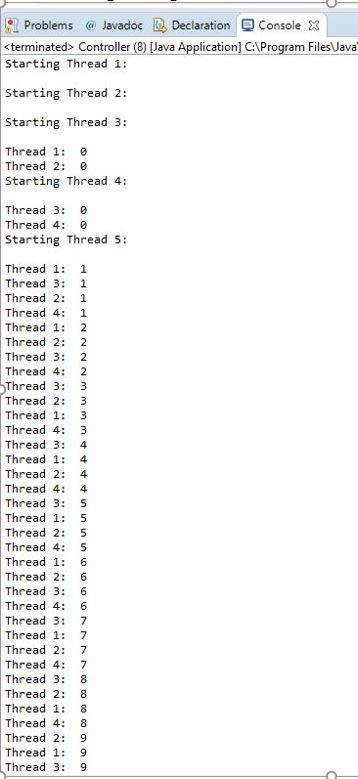
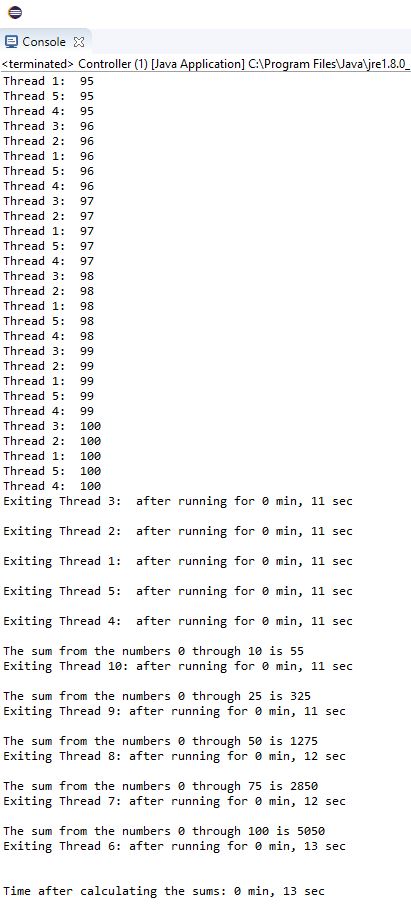
The Controller class is designed to run all the threads, and perform synchronized operations on them. The operations include: deciding when a thread should start, be suspended, resume, and sleep. The class also calls the Time class to display the time each thread started, was suspended, and when it resumed. The time is displayed in minutes and seconds. For example, 0 min, 5 sec. **Time Class:**

The Time class is designed to provide execution times for each thread. This class uses a TimeUnit, which is a library offered by java. This library is referenced in the Time.java class. The execution time, and ending time for each thread is converted into minutes and seconds.

# **Screenshots:**

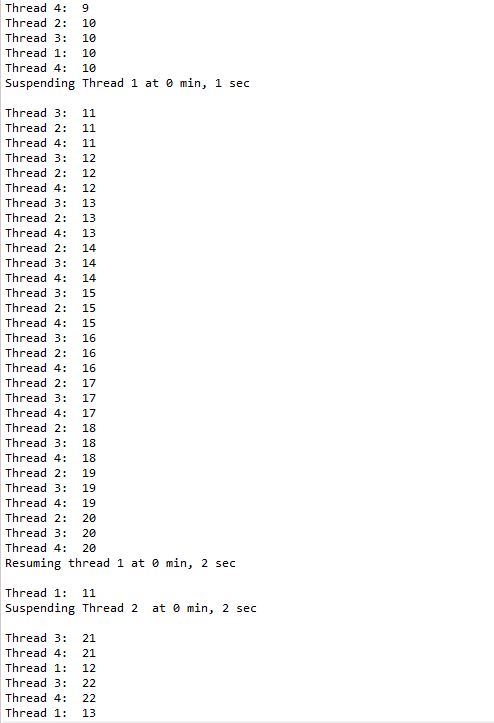
## **First:**

The program goes all the way to 100, where each thread must reach 100 before the program is finished. The following screenshot shows the beginning and the end of the programming.

## **Second:**

The screenshot below shows the time for when a thread has been suspended, and then resumed. When a thread is suspended and then resumed, it picks up where it left off.



Resumed at 11

Suspended at 10

# **Lessons Learned:**

There are many lessons to be learned after completing homework 2. This assignment allowed me to demonstrate my understanding for how to create and use threads. The first thing I did, and what I do for every programing assignment I am given, is to make sure that I understand what the assignment is asking for. The assignment wanted us to create three classes that would print to the system, perform a mathematical computation using threads, and a controller class to manage the threads. The hardest part for me during this assignment was figuring out how to use both classes: IOIntensive, and Computationally together. What made this difficult at first, was the inability to get all threads from each class to synchronize in the order I wanted it to. The problem was the sums from each class, and each thread were printing before the threads that were created to list the numbers 0 through 100 even started. The way I fixed this was by adjusting the sleep time in the run methods of each class, so that it would print the numbers first, and then the sum of all numbers. Overall, i am satisfied with my program, and my ability to create and use threads.