**Score: \_\_\_\_\_**

**Lab09 – Multithreaded Programs**

COMP256 – Computing Abstractions

Dickinson College

Spring 2023

Prof. Grant Braught

**Name(s):**

**Introduction:**

In this lab you will gain some practice writing multithreaded programs using the common patterns of *parallelization* and *specialization*. You will also gain some practice using locks to ensure mutually exclusive execution of critical sections to prevent race conditions.

**Setup:**

1. This lab will used the Comp256Silli container that you created and used in Lab05. If you still have that container in your Docker Desktop, start it and connect to it using the Tiger VNC viewer. If you do not have that container, you can create it from a command prompt on your machine (MacOS or WSL in Windows) using the following command:

docker create --name Comp256Silli --publish 5901:5901 --publish 6901:6901 braughtg/comp256-silli:1.0.0

There is no answer required here, but you must be able to start and connect to the Comp256Silli container.

2. In the Comp256Silli container, create a new directory named Lab09. You should store all of the programs that you create for this lab in your Lab09 directory.

There is no answer required here. But you must have a Lab09 directory because will compress and submit it to Moodle at the end of the lab.

**Thread Parallelization:**

3. Summarize in a sentence or two of your own words what it means to use threads for parallelization.

4. Consider the Parallelization program given in Appendix A of the lab when answering the following questions:

a. What static field is declared in the Parallelization class?

b. Describe in a sentence what the value of this static field will be after it is initialized.

c. Find the nested SumIt class inside of the Parallelization class.

i. What is the super class of the SumIt class?

ii. What are names of the parameters to the constructor of the SumIt class?

iii. Briefly explain what the SumIt thread computes when its run method is executed. Be sure that your explanation includes how the values passed to the SumIt constructor affect this computation.

d. The main method creates an Object of type SumIt, calls its start method, calls its getSum method and prints the result.

i. Without running the program describe what this program should output. You are not expected to give a specific value, but rather describe in words what the output should be.

ii. Copy and paste the Parallelization program from Appendix A into a file named Parallelization.java in your Lab09 directory.

Compile the program and run it multiple times. What value does the program output? Does that output seem correct? Why not?

iii. Why would the program output this value rather than the correct value?

5. Fix the Parallelization program so that it computes the correct result. Hint: main needs to wait for SumIt to do its work before asking it for the result.

No answer is required here, you will turn in your program at the end of the lab. However, as a check, if your program is working correctly, it should generate the output:

Sum: 49996307142

Note: It might be surprising that we could know the exact output of this program, given that it uses the random() method to fill the values array. However, notice that the Random object is created using the statement new Random(183) instead of what you might be more used to seeing, new Random(). When a parameter is passed to the constructor for the Random class, this initializes the random number generator with a *seed*. Every Random object created with the same seed will always generate the same sequence of numbers. While not useful in practice for generating random numbers, it is a very useful feature for testing programs.

It is not required reading, but if you are curious about random number generation and seeds, you can learn more here:

* <https://mathbits.com/JavaBitsNotebook/LibraryMethods/RandomGeneration.html>

6. Notice, that while your Parallelization program now computes the correct result it does not actually do any parallelization since one thread processes all of the data.

Modify the main method so that it uses four threads. Each thread should sum up ¼ of the values. For example, the first thread could sum the first ¼ of the values. The main thread should wait until all four threads are complete, retrieve the individual results and then add them up to get the final result. When your program works correctly the output should be the same as before you added the parallelization. Note: You do not need to add any additional Thread sub-classes. But you will need to create additional SumIt objects.

No answer is required here, you will turn in your program at the end of the lab.

**Thread Specialization:**

7. Summarize in a sentence or two of your own words what it means to use threads for specialization.

8. Copy the Specialization program from Appendix B into a file named Specialization.java in your Lab09 directory.

Modify the Specialization program so that it uses thread specialization to perform several different computations on an array of data concurrently. Your modified program should find and print the maximum and minimum of the values array.

Your program must:

* use two specialized threads, one to find the max and one to find the min.
* ensure that these specialized threads run concurrently. That is neither one of them should be forced to wait for the other to complete.
* The max and min values must be printed by main, as in the given code.

Note: Because a seed was again used for the Random class, your answer should always be the same. The max value that should be found is 999997 and the min value that should be found is 11.

No answer is required here, you will turn in your program at the end of the lab.

**Race Conditions:**

9. Summarize in a sentence or two of your own words what it means for a program to contain a race condition and what about a program causes race conditions.

10. Consider the RaceConditions program given in Appendix C in answering the following questions:

a. What three threads exist in this program?

b. What two shared variables exist in this program?

c. If the program did not contain a race condition what should final length and contents of the variable s1 be?

d. If the program did not contain a race condition what should final length and contents of the variable s2 be?

11. Copy the RaceConditions program from Appendix B into a file named RaceConditions.java in your Lab09 directory.

Modify the RaceConditions program so that it no longer contains any race conditions. Your modified program must:

* Use the two shared variables s1 and s2 and the three threads AppendA, AppendB and AppendC.
* Use synchronized blocks with shared Object variables as locks to ensure mutually exclusive execution that eliminates the race conditions.
* Ensure that the three threads AppendA, AppendB and AppendC:
  + All run concurrently.
  + Only make other threads wait when necessary to prevent race conditions.
* Print the length and value of s1 and s2 from main as in the given code.

No answer is required here, you will turn in your program at the end of the lab.

**Submitting:**

12. Submit this lab as follows:

a. Use the following commands in a terminal window in the container:

cd /home/student

tar -zcvf Lab09.tar.gz Lab09

b. Use the FireFox browser in the container to go to the course Moodle site and submit the Lab09.tar.gz file to the Lab09 Code assignment.

c. Convert this activity sheet to a PDF and submit it to the Lab09 Activity Sheet assignment on Moodle.

Optional: To help me improve and scope these activities for future semesters please consider providing the following feedback.

a. Approximately how much time did you spend on this activity outside of class time?

b. Please comment on any particular challenges you faced in completing this activity.

**Appendix A: Parallelization.java**

import java.util.Random;

class Parallelization {

private static int[] values = getArray();

private static int[] getArray() {

Random rnd = new Random(183);

int[] arr = new int[10000000];

for (int i=0; i<arr.length; i++) {

arr[i] = rnd.nextInt(10000);

}

return arr;

}

public static void main(String[] args) {

Thread t1 = new SumIt(0, values.length);

t1.start();

long total = ((SumIt)t1).getSum();

System.out.println("Sum: " + total);

}

static class SumIt extends Thread {

private int start;

private int end;

private long sum;

public SumIt(int start, int end) {

this.start = start;

this.end = end;

this.sum = 0;

}

public long getSum() {

return sum;

}

public void run() {

for (int i=start; i<end; i++) {

sum = sum + values[i];

}

}

}

}

**Appendix B: Specialization.java**

import java.util.Random;

public class Specialization {

private static int[] values = getArray();

private static int[] getArray() {

Random rnd = new Random(183);

int[] arr = new int[100000];

for (int i=0; i<arr.length; i++) {

arr[i] = rnd.nextInt(1000000);

}

return arr;

}

public static void main(String[] args) {

values = getArray();

// Create and start threads

// Wait for threads to complete

// Get and print results

}

// Add classes here for the threads.

}

**Appendix C: RaceConditions.java**

public class RaceConditions {

private static String s1 = new String();

private static String s2 = new String();

public static void main(String[] args) {

Thread ta = new AppendA();

ta.start();

Thread tb = new AppendB();

tb.start();

Thread tc = new AppendC();

tc.start();

System.out.println("s1.length: " + s1.length());

System.out.println("s1: " + s1);

System.out.println();

System.out.println("s2.length: " + s2.length());

System.out.println("s2: " + s2);

}

static class AppendA extends Thread {

public void run() {

for (int i = 0; i < 10; i++) {

s1 = s1 + "A";

}

}

}

static class AppendB extends Thread {

public void run() {

for (int i = 0; i < 10; i++) {

s2 = s2 + "B";

}

}

}

static class AppendC extends Thread {

public void run() {

for (int i = 0; i < 10; i++) {

s1 = s1 + "C";

s2 = s2 + "C";

}

}

}

}