# Parallel and MultiThreaded Programming

# CSYE 7215

# Homework 4

# Due: February 8, 2020

Put all your java, compiled class files and documentation into a zip file named Homework4.zip and submit it via the dropbox on the blackboard before the END of due date. Put your name on all .java files. There will be a short quiz on this assignment.

**1.** Explain:

Implicit lock versus Explicit lock

Class lock versus Object lock

Call Stack

Stack Memory

Heap Space

String Pool, give example

Deadlock, Starvation, Race condition, provide examples

**2**. Search internet to find ALL Threadsafe and NotThreadsafe Java collections.

For each collection:

a) List three methods in each collection describing what is the intent of collection,

when it is useful to use this collection, and describe the selected methods.

b) What is Collections class? Most methods in this class are static, name five methods.

**3**. Suppose you have a two dimensional array input data;

int[][] arr = { { 9, 12, 6, 14, 10, 21, 13}, { 3, 5, 41, 16, 14, 10, 21 },

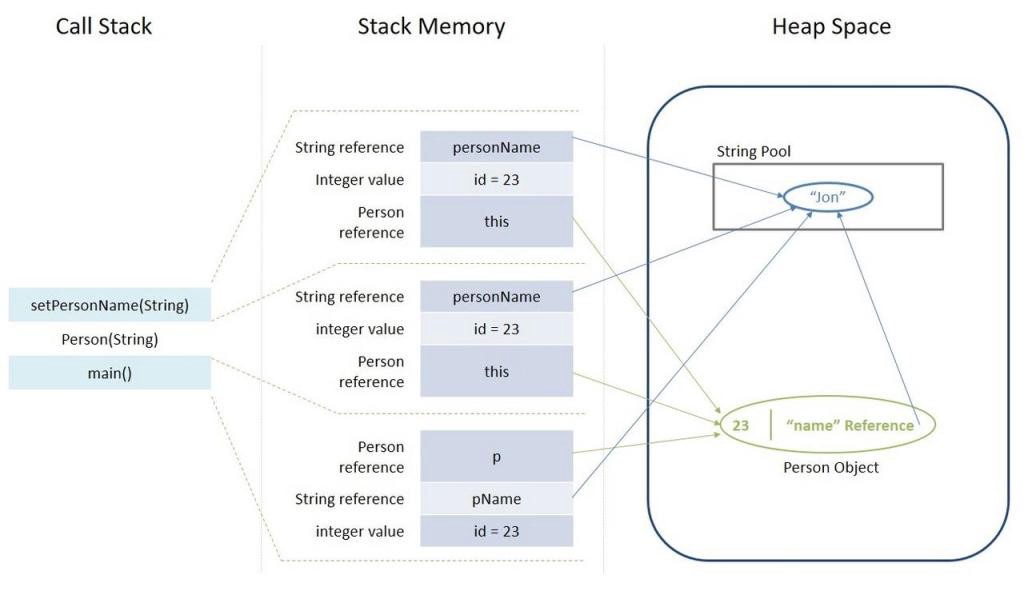
{ 3, 15, 41, 17, 11, 10, 51 }, { 3, 15, 41, 17, 11, 10, 51 } };

Write Java program: a) Create four Threads where each thread-id corresponds to a row in array input data, for example, (tid1, row1), (tid2, row2), (tid3, row3), (tid4, row4),

b) Write code for each thread to sort its row of array data using sort method in Java Collections library, c) Write code to update array data with sorted values, d) Sort all rows in array data using HeapSort, e) Write code for each thread to print the sorted data.

Notes: You need to protect Array data. How do you protect array data? First you sort using Collections class, and then for Heapsort. How lock mechanism works in (b) and (d)?

**4.** In Homework3, you created 50 student threads and one GraderThread. Change the program to use Explicit locking instead of implicit locking. Note: see problem description in hw3, problem-3 (a) (b) (c) (d), and all the requirements for that problem must be implemented in this problem using explicit locking.

**5**. Reverse engineer the following JVM memory model explain what object it represents in detail?

**6.**  Synchronized blocks in Java are Reentrant. That is if a Java thread enters a synchronized block code, and thereby takes the lock on the monitor object the block is synchronized on, the thread can enter other Java code blocks synchronized on the same monitor object. a) Explain how the following code works? b) Provide different Test scenarios that can successfully execute this code.

public class Reentrant2 {

Lock lock = new Lock();

public outer() {

lock.lock();

inner();

lock.unlock();

}

public synchronized inner(){

lock.lock();

//do something

lock.unlock();

} }

**7**. Consider the following code using Reentrance Lock. How does the Lock work in the following program. Compile and Run.

**class** PrintingJob **implements** Runnable

{

**private** PrinterQueue printerQueue;

**public** PrintingJob(PrinterQueue printerQueue)

   {

**this**.printerQueue = printerQueue;

   }

   @Override

**public** **void** run()

   {

    System.out.printf("%s: Going to print a document\n”,

Thread.currentThread().getName());

      printerQueue.printJob(**new** Object());

   }

}

**class** PrinterQueue

{

**private** **final** Lock queueLock = **new** ReentrantLock();

**public** **void** printJob(Object document)

   {

      queueLock.lock();

**try**

      {

         Long duration = (**long**) (Math.random() \* 10000);

         System.out.println(Thread.currentThread().getName() +

": PrintQueue: Printing a Job during " + (duration / 1000) +

" seconds :: Time - " + **new** Date());

         Thread.sleep(duration);

      } **catch** (InterruptedException e)

      {

         e.printStackTrace();

      } **finally**

      {

         System.out.printf("%s: The document has been printed\n”,

Thread.currentThread().getName());

         queueLock.unlock(); } } }

**public** **class** LockExample

{

**public** **static** **void** main(String[] args)

   {

      PrinterQueue printerQueue = **new** PrinterQueue();

      Thread thread[] = **new** Thread[10];

**for** (**int** i = 0; i < 10; i++)

      {

         thread[i] = **new** Thread(**new** PrintingJob(printerQueue), "Thread " + i);

      }

**for** (**int** i = 0; i < 10; i++)

      {

         thread[i].start();

      }

   }

}