# Parallel and MultiThreaded Programming

# CSYE 7215

# Homework 3

# Due: February 2, 2020

Put all your java, compiled class files and documentation into a zip file named Homework3.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:

Basic components of computer architecture?

Memory Heap?

When you compile and execute your program (eg: Hello.java),

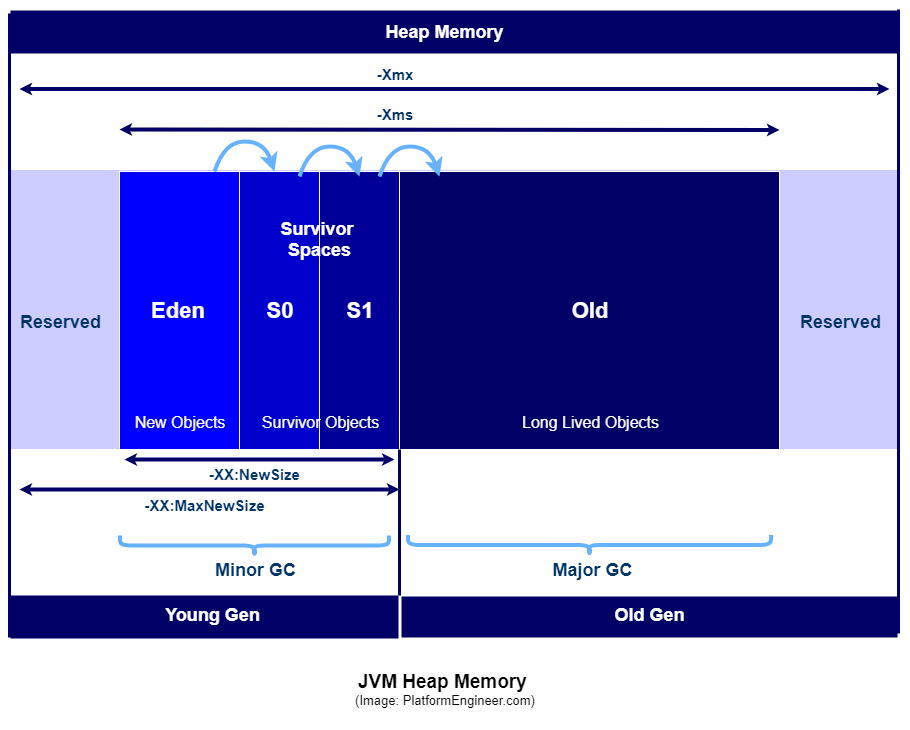
what happens? Provide details of execution environment.

JVM memory architecture components?

JVM Heap, JVM Non-Heap, Cache?

Native Cache?

Java Native Interface (JNI)?

2. Explain Garbage Collection, How does it work? Young Generation, Old Generation?

How does this Figure works, Eden Space, Survivor Space, Old Space, MinorGC,

MajorGC, Explain

3. In Homework2, you defined Student class, and created 32 student threads and created one GraderThread. Change previous homework, and have GraderThread to write final grade to file “FinalGrades”. This is how things going to work:

a) Create 50 student threads, each thread records 4 scores (homework, quiz, midterm-exam, final-exam), and each score is randomly generated between 70 to 100 with incremental 1second interval. Use a List to store scores and is set to zeros initially. Use Map with key/value, (List, threadId).

b) Each student thread first generates homework score within 1 second, stores score in List [0], and then writes List to map. Next, it generates quiz score within the next second, stores to List[1] and writes List to map. Next, it generates midterm-exam score within the next second, stores score in List[2] and writes List to map. And then generates final-exam score within the next second, stores score in List[3], and writes List to map.

c) GraderThread randomly reads scores from Map, calculates final grade for Letter

A, B, C, D, F, and writes the final grade to file “FinalGrades”.

d) Each student thread, Reads final grade from file “FinalGrades” and reports its grade,

Notes: Things to think about: 1) How to protect Map and File? 2) student Threads writing to Map and Reading from File, 3) GraderThread is reading Map and writing to File. 4) student threads write List to Map with incremental List update for each score. Grader thread reads Map for each student scores, the Grader can move to calculate letter grade calculation and write the grade to the file?, the Grader thread does not need to wait for all student threads, once each student completes scores, grader thread can move to calculate grade. 5) Student threads often read file to see if final grade is submitted by the grader thread, think thread safety for all reads and writes. Carefully read steps described above.

Note: Because there are many read and write operations to the Map and File, between student threads and grader thread, there is possibility for **Race Condition**, How does that is possible?, Explain.

4. Provide Call Stack, Stack Memory, and Heap Space for the following code:

class MyThread implements Runnable {

String name;

Thread t;

MyThread String thread){

name = threadname;

t = new Thread(this, name);

System.out.println("New thread: " + t);

t.start();

}

public void run() {

try {

for(int i = 5; i > 0; i--) {

System.out.println(name + ": " + i);

Thread.sleep(1000);

}

}catch (InterruptedException e) {

System.out.println(name + "Interrupted");

}

System.out.println(name + " exiting.");

}

}

class MultiThread {

public static void main(String args[]) {

new MyThread("One");

new MyThread("Two");

new NewThread("Three");

try {

Thread.sleep(10000);

} catch (InterruptedException e) {

System.out.println("Main thread Interrupted");

}

System.out.println("Main thread exiting.");

}

}

5. A deadlock is when two or more threads are blocked waiting to obtain locks that some of the other threads in the deadlock are holding. Deadlock can occur when multiple threads need the same locks, at the same time, but obtain them in different order. For instance, if thread-1 locks A, and tries to lock B, and thread-2 has already locked B, and tries to lock A, a deadlock arises. Thread-1 can never get B, and thread 2 can never get A. In addition, neither of them will ever know. They will remain blocked on each their object, A and B, forever. This situation is a deadlock.

Thread-1 locks A, waits for B

Thread-2 locks B, waits for A

<http://tutorials.jenkov.com/java-concurrency/deadlock.html>

A) Explain As why this code Deadlocks?

B) Compile and Run this code.

C) Is Race Condition possible in this code, Yes/No, Why?

public class TreeNode {

TreeNode parent = null;

List children = new ArrayList();

public synchronized void addChild(TreeNode child){

if(!this.children.contains(child)) {

this.children.add(child);

child.setParentOnly(this);

}

}

public synchronized void addChildOnly(TreeNode child){

if(!this.children.contains(child){

this.children.add(child);

}

}

public synchronized void setParent(TreeNode parent){

this.parent = parent;

parent.addChildOnly(this);

}

public synchronized void setParentOnly(TreeNode parent){

this.parent = parent;

}

}

6. Explain every element in this JVM architecture, and how element layers are tied to each other?