Parallel and MultiThreaded Programming
CSYE 7215
Homework 2
Due: September 27, 2020

Put all your java, compiled class files and documentation into a zip file named Homework2.zip and submit it via the dropbox on the Canvas before the END of due date. Put your name on all .java files. There will be a short quiz on this assignment.

1. Explain:

Monitor
Lock
What are Java Object Monitors?
What is the difference between Lock and Monitor?
Synchronization
Object Level Lock
Class Level Lock
StackThread

2. Your program Hello references five objects, Class1, Class2, Class3, Class4, Class5. What happens when you start your program? Describe your answers in terms of Process, JVM, JRE, ClassLoader, Stack Thread, how does your program gets started?

3. Define a Student class with instance variables name, id, homework, midterm, and final. Name is a string whereas others are all integers. Also add a static variable nextId which is an integer and statically initialized to 1. In each of them, the id should be assigned to the next available id given by nextId. The default constructor should set the name of the student object to "StudentX" where X is the next id.

a) Your program is to create 25 Student Threads each to be identified with name-<Thread-nextId>. The default constructor for each thread calls a method to randomly generate grades for homework, midterm, and final-exam ranging between 70 to 100 inclusive. You need to consider 1 second wait-time between each score generation for homework, midterm, and final. Each student thread writes the grade scores to "Grades" file in this format: name, nextId, ThreadId, homework, midterm, and final. All student threads share this file and you need to protect it.

b) Create GraderThread that checks "Grades" file periodically up to 30 seconds to retrieve submitted grades by all student threads. How do you protect the file? The GraderThread reads the file (format described above) and validates name, id, threadId, scores for all 25 student threads submitted scores. For any missing grade, the student will receives zero score. The GraderThread does calculateGrade() (50% homework + 30% midterm+ 20% final) and returns a letter grade like "A", "B", "C", "D" or "F", based on the overall score.

c) In your StudentThread and GraderThread, you wrote several methods to handle processing of various functions in your program. Provide JVM model for your program with details as how to handle stack threads with their methods, local variables, array, heap objects.

Note: you need to show only example data for one StudentThread. There are other single threads.

d) Create GradesDriver class to create 25 StudentThreads and GraderThread to test your program. Compile and Run hour program.

Notes: You need to consider a number of protection mechanisms to protect the "Grades" file to avoid threads over-stepping each other. You need to think about what data structures to use to hold the FinalGrade (ie: use HashMap for key/value as threadId/FinalGrade). You need to think about how to protect HashMap. You need to think about how GraderThread to notify each student with finalGrade. Do you want to update the file with new column "FinalGrade" and have student threads get the final grade by reading the file? How does that going to work? You need to think how this mechanism can work. OR do you want to do another Wait/Notify where the GraderThread will be the notifier to each student thread? All in all, you need to be creative to solve this problem.

4. Does Thread synchronization works correctly with the following code? Why or Why not? If Not, how do you fix it?

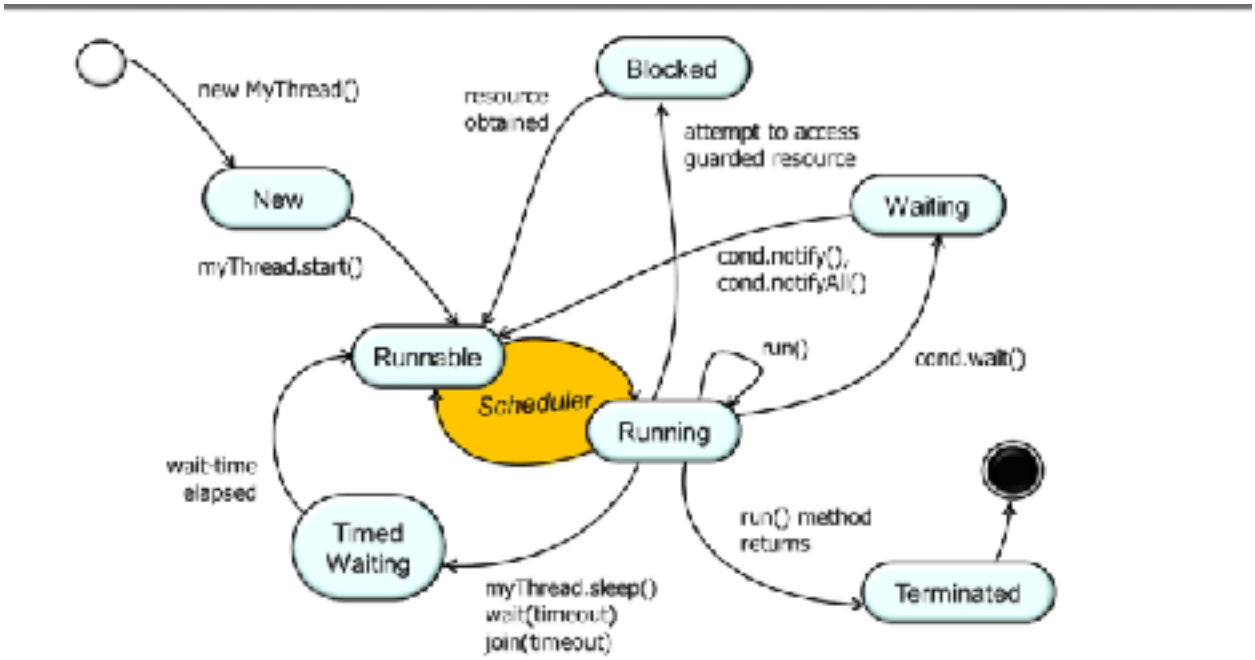
```
//example of java synchronized method
class Table{
 synchronized void printTable(int n){//synchronized method
   for(int i=1;i<=5;i++){
     System.out.println(n*i);
     try{
      Thread.sleep(400);
     }catch(Exception e){System.out.println(e);}
   }
 }
}

class MyThread1 extends Thread{
Table t;
MyThread1(Table t){
this.t=t;
}
public void run(){
t.printTable(5);
}
}
class MyThread2 extends Thread{
Table t;
MyThread2(Table t){
this.t=t;
}
public void run(){
```

```
        t.printTable(100);  } }
public class TestSynchronization2{
public static void main(String args[]){
Table obj1 = new Table();//only one object

MyThread1 t1=new MyThread1(obj1);
MyThread2 t2=new MyThread2(obj2=1);
t1.start();  t2.start();  }}
```

5. Explain Thread State Diagram



6. Consider the following code segments:

A) Compile and Run the following code segments

B) Explain the code and how Wait/Notify synchronization works with transfer flag?

```
public class Data {
    private String packet;

    // True if receiver should wait
```

```
// False if sender should wait
private boolean transfer = true;

public synchronized void send(String packet) {
    while (!transfer) {
        try {
            wait();
        } catch (InterruptedException e)  {
            Thread.currentThread().interrupt();
            Log.error("Thread interrupted", e);
        }
    }
    transfer = false;

    this.packet = packet;
    notifyAll();
}
public synchronized String receive() {
    while (transfer) {
        try {
            wait();
        } catch (InterruptedException e)  {
            Thread.currentThread().interrupt();
            Log.error("Thread interrupted", e);
        }
    }
    transfer = true;

    notifyAll();
    return packet;
} }
```

7. Consider the following Thread example:

A) Compile and Run the following code segments

B) Explain the code, What are object monitors?

```
public class Input {
   int index;
   int[] input = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};

   public Input(){
      index = 0;
   }

   public void print(int index){
      System.out.println(input[index]);
   }
```

```
    synchronized public int getIndex(){
        if(index == 15)
            return -1;
        return index++;
    }
}

public class MyThread implements Runnable{

    Input ip;
    Object lock;

    public MyThread(Input ip, Object lock){
        this.ip = ip;
        this.lock = lock;
    }

    @Override
    public void run() {
        int index = -1;
        while((index=ip.getIndex())!=-1){
            synchronized(lock) {
                System.out.println(Thread.currentThread().getName());
                ip.print(index);
            }
        }
    }
}
public class Caller {

    public static void main(String[] args) throws InterruptedException {
        Input ip = new Input();
        Object lock = new Object();
        Thread t1 = new Thread(new MyThread(ip, lock), "Thread1");
        Thread t2 = new Thread(new MyThread(ip, lock), "Thread2");
        t1.start();
        t2.start();
        t1.join();
        t2.join();
    }
}
```

8. Consider the following code segments:

A) Compile and Run the following code segments

B) Explain the code and how the synchronization of code works?

```
package com.journaldev.concurrency;

public class Message {
```

```
    private String msg;

    public Message(String str){
        this.msg=str;
    }

    public String getMsg() {
        return msg;
    }

    public void setMsg(String str) {
        this.msg=str;
    }
}

package com.journaldev.concurrency;

public class Notifier implements Runnable {
    private Message msg;
    public Notifier(Message msg) {
        this.msg = msg;
    }

    @Override
    public void run() {
        String name = Thread.currentThread().getName();
        System.out.println(name+" started");
        try {
            Thread.sleep(1000);
            synchronized (msg) {
                msg.setMsg(name+" Notifier work done");
                msg.notify();
                // msg.notifyAll();
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}}
```

```
package com.journaldev.concurrency;

public class WaitNotifyTest {

    public static void main(String[] args) {
        Message msg = new Message("process it");
        Waiter waiter = new Waiter(msg);
        new Thread(waiter,"waiter").start();

        Waiter waiter1 = new Waiter(msg);
        new Thread(waiter1, "waiter1").start();

        Notifier notifier = new Notifier(msg);
        new Thread(notifier, "notifier").start();
        System.out.println("All the threads are started");
    }

}
```