# 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**

* A monitor is mechanism to control concurrent access to an object.
* **mutual exclusion** – only one thread can execute the method at a certain point in time, using *locks*
* ***cooperation*** – the ability to make threads wait for certain conditions to be met, using *wait-set*

**Lock**

* A lock is kind of data which is logically part of an object’s header on the heap memory.
* Each object in a JVM has this lock (or mutex) that any program can use to coordinate multi-threaded access to the object.
* If any thread want to access instance variables of that object; then thread must “own” the object’s lock.
* All other threads that attempt to access the object’s variables have to wait until the owning thread releases the object’s lock (unset the flag).

**What are Java Object Monitors?**

Java associates a monitor with each object. The monitor enforces mutual exclusive access to synchronized methods invoked on the associated object. When a thread calls a synchronized method on an object, the JVM checks the monitor for that object

* If the monitor is *unowned*, ownership is assigned to the calling thread, which is then allowed to proceed with the method call
* if the monitor is *owned* by another thread, the calling thread will be put on hold until the monitor becomes available

**What is the difference between Lock and Monitor? Synchronization**

* A lock is kind of data **which is logically part of an object’s** header on the heap memory on the other hand A monitor is **mechanism to control concurrent acc**ess to an object using mutual exclusion (**using lock**) and cooperation
* Mutual exclusive lock we can achieve using synchronized block while cooperation can be done using wait and notify methods.

**Object Level Lock**

* Every object in java has a unique lock. Whenever we are using synchronized keyword, then only lock concept will come in the picture.
* If a thread wants to execute synchronized method on the given object. First, it has to get lock of that object. Once thread got the lock then it is allowed to execute any synchronized method on that object. Once method execution completes automatically thread releases the lock.

**Class Level Lock**

* Every class in java has a unique lock which is nothing but class level lock.
* If a thread wants to execute a static synchronized method, then thread requires class level lock. Once a thread got the class level lock, then it is allowed to execute any static synchronized method of that class.
* Once method execution completes automatically thread releases the lock

**StackThread**

* Thread stack is memory area which can be accessible by only that particular thread that owns the thread stack.
* We cannot share the data between two thread via thread stack.
* Every time we are creating new thread. Each threads create the new thread stack which store the local variable and the pointer to reference variable apart from this it contains the frame of method execution in LIFO manner.
* Thread stack is available inside the run time memory area of JVM.

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?**

Suppose I have following class Hello and it has reference of 5 classes class1 to class5

Public class Hello{

Class1 c1= new Class1();

Class2 c2= new Class2();

Class3 c3= new Class3();

Class4 c4= new Class4();

Class5 c5= new Class5();

}

Public class main(){

Public static void main(String args[])

{

Hello h = new Hello();

}

}

When we first compile the program java compiler convert this code to **bytecode** and create **class file** for example **Hello.class and main.class.** this bytecode acts as a platform-independent intermediary state which is portable among any JVM regardless of underlying OS and hardware architecture.

So once the code is compiled by java compiler it will go into Java runtime environment. JRE contains the Java virtual machine which has different components like

* Class Loader 🡪 Dynamic class loading Functionality
* Runtime Data area🡪 Memory Area where class loader generates the corresponding binary data and save the method and class metadata
* Execution Engine 🡪Execute the instruction in byte code line by line.

Please find the following figure for better understanding

Source Code

Hello.java

Main.java

Java Bytecode

Hello.class

Main.class

In HexaDecimal format

Java Compiler

JavaC

Java Virtual machine

Class Loader

Initializing

Loading

Linking

Runtime data Area

Native Stack

Program Counter

ThreadStack

Heap

Method Area

Execution Engine

Interpreter

JIT Compiler

**Execution Steps**

* Source code compiled into byte code in .class file in hexadecimal format by java compiler
* Then this class file goes into Java runtime environment 🡪java virtual machine
* This class file will load into main memory by class loader which is known as dynamic class loading
* Runtime Data Areas are the memory areas assigned when the JVM program runs on the OS. In addition to reading .class files, the Class Loader subsystem generates corresponding binary data and save the following information in the Method area for each class separately.
  + Fully qualified name of the loaded class and its immediate parent class
  + Whether .class file is related to a Class/Interface/Enum
  + Modifiers, static variables, and method information etc.
* Then, for every loaded .class file, it creates exactly one object of Class to represent the file in the Heap memory as defined in java.lang package. This Class object can be used to read class level information (class name, parent name, methods, variable information, static variables etc.) later in our code.
* At the end the actual execution of the bytecode occurs. Execution Engine executes the instructions in the bytecode line-by-line by reading the data assigned to above runtime data areas using interpreter and JIT.

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.

Output:

Text

Description automatically generated

Text

Description automatically generated

JVM Model

* For given program I have created two class which implements the runnable interface 1) StudentThread and 2)Grader Thread. Apart from this to perform the reading , writing and updating the Grade.txt file I have created one Utils classs.
* So I am creating the 25 different threads for each student and then assign the id , name, homework, midterm and final score randomly between 70 to 100.
* So Inside the JVM run time memory area it will create the 25 different thread stack for each thread such as thread-1 to thread-25.

Thread-25

Thread-1

Local variable

Frame stack

…Upto 25 thread stack for each thread.

Heap Memory Area

Instance of Utils class

* Each thread stack contains the all local variables and reference to class variable which is pointing to heap memory.
* Apart from this it will also contains the frame stack of method execution so every time the local method return something the frame will popped out from that stack in LIFO manner.
* In my solution I have used the Utils class which is performing data reading and writing task. During the creation of each StudentThread I passed the instance of utils class.
* Inside the util class I have created two three synchronized method
  + writeData 🡪 to write data in Grade.txt file
  + readFile🡪 Read the data from Grade.txt file
  + updateFile🡪 Update the letter grade in grade.txt file
* I have used the object level lock mechanism so I am acquiring the lock on instance of utils class.
* So Inside the main method I am creating new thread named as GraderThread and start that thread. So after every 30 seconds ,this thread will read the Grade.txt file and calculate the letter grade.
* Initially this thread will wait so StudentThread can write into Grade.txt file
* After the 30 seconds this thread start reading the grade.txt file and calculate the grade and update the grade in grade.txt file.
* To synchronized the letter grade with each thread I have created ConcurrentHashMap object which store the student thread id as key and letter grade as value.

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

// **Obj2 initialization is missing**

MyThread1 t1=new MyThread1(obj1);

MyThread2 t2=new MyThread2(obj2);

t1.start(); t2.start(); }}

Answer:

In above program synchronization will not work properly because we are passing two different object of Table class so here the object level lock will not work.

Output snap:

A screenshot of a cell phone

Description automatically generated

Now we have two option to fix this problem

1. we can pass the same table object to each thread so we can utilize the object level lock functionality

MyThread1 t1=new MyThread1(obj1);

**MyThread2 t2=new MyThread2(obj1);** *<------* ***this allow us to use object level lock***

1. we can make the **printable** method as static method so we can acquire lock on class object itself.

synchronized **static** 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);}

}

}

In above case we are making printable as static method so we can make sure that multiple thread with different table instance execute this method in synchronized manner.

MyThread1 t1=new MyThread1(obj1);

**MyThread2 t2=new MyThread2(obj2);** <------ ***it will work as we have used class level lock.***

pasted-image.tiff5. Explain Thread State Diagram

* when we creates a new thread and use the **thread.start()** method thread will go to in Runnable state.
* After that scheduler if scheduler pick that thread and run it then it will be in running state.
* If the Running thread use the any sleep method such as sleep(), wait() or join() then it will go into time waiting state for the **particular time period which mentioned in sleep methods**.
* If the thread in Running state and use the **thread.wait()** method then thread will go into waiting state until other thread notify the current thread using **notify() or notifyAll**() method.
* If the current thread trying to access the guarded resources such as synchronized block and that resources is already in use by other threads, then current thread will go into Blocked state until it obtained the resource.
* At the end, once the run method returns the value current thread gets terminated.

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) { // Tansfer is false means receiver yet not received the packet so wait for that

**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) { // True that means Sender is sending a packet

**try** {

                wait();

            } **catch** (InterruptedException e)  {

                Thread.currentThread().interrupt();

                Log.error(**"Thread interrupted"**, e);

            }

        }

        transfer = **true**;

        notifyAll();

**return** packet;

    } }

***Output :***

A screenshot of a cell phone

Description automatically generated

***How Programs works?***

* Packet variable denotes as the data that is being transferred over the network.
* We have used the transfer flag to make sure that sender will send the packet only after receiver received previous packet otherwise sender should wait vice versa receiver should wait while sender is sending a new packet.
* The Sender will use the send() method to send the data to receiver.
  + If transfer = false means sender should wait
  + If transfer = true then sender should notify all other thread to wake up using notifyAll() method.
* Receiver will use Receiver Method to receive data
  + If transfer = false thread will proceed the data and notify all other threads using notifyAll() method
  + If transfer = true then thread will wait

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 method block make sure that at a time only one thread can execute this method**

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) {

**used object lock approach to makes sure that single instance of an object can access this block**

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();

}

}

Output:

A screenshot of a cell phone

Description automatically generated

Explanation:

In caller class, we have created the **object of the Input class** and **lock instance of the object class which will monitor the lock on current instance method**. We have passed the same lock object to thread t1 and t2 to make sure that our synchronized code block(inside the MyThread class run method) execute by on one thread at the same time. similar way we have passed the **same input object to thread t1 and t2**. We used that object to call **the getIndex method** which is again **synchronized method** so it makes sure that **it is executed by a single thread at a given time**. We can confirm this by checking the output screenshot which clearly shows that at a given time each thread gets the different value form the **getIndex()** method. At the end of the main method, we have used the **thread.Join()** method which makes sure that **the main thread not get terminated before thread t1 and t2**.

**What are object monitors?**

Object monitor is a **mechanism** that **allow thread to have mutual** **exclusive lock** means only **one thread** can **execute** the method at certain **point in time** using **lock** and **cooperation** means **ability to make threads** **wait** for certain condition to be met using **wait and set**

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](https://www.journaldev.com/1162/java-multithreading-concurrency-interview-questions-answers);

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");

}

}

Explanation

* In given program we have created waiter class and notifier class which implements the runnable interface and we have message class which is simple contains message.
* In main class we are creating multiple objects of the waiter class waiter and wiater1 and passing message objects and start that threads using thread.start() method.
* In Run method of waiter class we are printing the waiting message and hold the execution of that thread using thread.wait() method.
* Meanwhile we are also creating instance of notifier class and start the thread.In Notifier class run method we are printing the thread started message and after that inside the synchronized block we are setting the message value and then we notify the other thread (waiter / waiter1) using thread.notify() method.
* As we have used the notify method on of the waiter thread start their execution while other will still in waiting state.
* If we use the thread.notifyAll() method then both the waiter thread start their execution.
* We can verify the behavior of notify and notifyAll method by output screenshots

Output with notify() method

Text

Description automatically generated

Output with notifyAll() method

Text

Description automatically generated