# Parallel and MultiThreaded Programming

# CSYE 7215

# Homework 1

# Due: September 19, 2020

Put all your java, compiled class files and documentation into a zip file named Homework1.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. **Describe the followings:**

**Process?**

Process are execution environment provided by operating system that has its own set of resources such as memory

**Thread? Provide five examples, Advantages?**

Threads are processes that live within the process and shares their resources.

Threads are also called as lightweight processes.

Examples

1. Microsoft word or any other text editors where we are writing and another thread is used to check the spelling
2. Integrated development environment (IDE such as IntelliJ or Android Studio) which runs the auto complete or suggest syntax feature in separate thread.
3. Web Browser where we can work simultaneously with multiple tabs and we can download multiple things concurrently.
4. Video Players which process the multiple frame and audio script using multiple threads
5. Games are another example of multithreading where one thread using for background music another is used for visuals and other stuff.

**Advantages:**

Improve development and Maintenance cost

Improve performance of complex application

Using Multithreading we can increase the throughput on single process system.

Threads are very useful for GUI application to improve the responsiveness of the user interface.

**Task?**

Task is set of work that we can perform using single or multiple thread

**Yield?**

Yield allow us to temporarily pause the execution of current thread and allow other thread to proceed.

**States of Java Threads**

1. **NEW**: A thread that has not yet started is in this state.
2. **RUNNABLE**: A thread executing in the Java virtual machine is in this state.
3. **BLOCKED**: A thread that is blocked waiting for **a monitor lock** is in this state.
4. **WAITING**: A thread that is waiting indefinitely for another thread to perform a **particular action is in this state**.
5. **TIMED\_WAITING**: A thread that is waiting for another thread to perform an action for up to a **specified waiting time** is in this state.
6. **TERMINATED**: A thread that has exited is in this state.

**Java Object class**.

* Java object class is the **parent class of all the classes** (Root of the class hierarchy) in java. Aka Topmost class in java.
* It’s helpful when you don’t know type of the object.

**What is the difference between Concurrency and Parallelism?**

**Concurrency :**

* Concurrency means running multiple task at the same time and share same resources
* For example if we have different task to perform such as Task1, Task2 and so on we can perform these task concurrently by assigning each task to different thread.

**Parallelism**

* Splits up task in subtask and execute all task parallelly in different core of CPU.
* Parallelism requires hardware with multiple processing units.
* In **single core CPU** you might get concurrency **not parallelism**.
* For example we want to perform big task assuming that we can not perform that in single core of CPU then we can splits up this task into subtask and let run each individual task in different core of the CPU.

**Why MultiThreading is important**?

* Multithreading increase the Responsiveness of application. Single application will use the multiple thread to accomplish various task.
* Multithreading improve the throughput of the applications .
* Minimized system resource usage. Threads impose minimal impact on system resources. Threads require less overhead to create, maintain, and manage than a traditional process.

**What are similarities and differences between Java and C++**

* Both the languages support Object oriented programming.
* Java compiler compile code in bytecode and JVM convert that code to native machine code using interpreter. While C++ compiler compile and convert the source code. Into machine code.
* C++ allows multiple inheritance using concrete class using virtual inheritance while java support multiple inheritance using interface.
* Java supports interfaces and packages while C++ doesn’t.
* C++ is platform dependent while java is platform independent.

**What is diamond problem in C++.**

* C++ supports the multiple inheritance using concrete classes.
* Consider the following diagram Let’s assume that class A and B inherits the method of Class Z and class C inherits the class A and B. class A and B override the same method of class Z with different Implementation. Now when class C uses the same method compiler did not understand which implementation it should use. This problem is known as classic diamond problem.

Z

B

A

C

*Diamond problem in C++*

**JVM, JRE, explain what they do?**

* JRE stands for Java runtime Environment
  + Minimum Environment requires to run java application.
  + JRE Contains java virtual machine and deployment tools

* JVM stands for Java virtual Environment
  + JVM interpret the compile code into native machine language which can be understood by OS. and underlying hardware platform.
  + It resides inside the RAM

**What is a Class Loader?**

* Class loader helps to bring the calls file on to RAM also known as Dynamic class loading functionality.
* It’s load, link and initialize the class file when it refers to class for the for the first time at runtime.
* There are three class loaders available in loading task
  + Bootstrap class loader
  + Extension Class loader
  + Application Class Loader

\**Apart from these three major class loader we can also defined user defined class loader as well.*

2. Write a Java program to define a Student class with instance variables name, id, homework, midterm, and final. Name is a string whereas others are all integers. Add a static variable nextId, which is an integer and statically initialized to 1. Have some overLoaded constructors. In each constructor, id should be assigned to the next available id given by nextId. The default constructor should set the name of the student object to “Student\_X” where X is the next id. Add calculateGrade() method which returns a string for the letter grade of the student, like “A”, “B”, “C”, “D” or “F”, based on the overall score. Overall score should be calculated as (50% homework, 30% midterm + 20% final).

Write TestDriver to test your program. It should create 20 student objects with default constructor and invoke the setter methods for homework, midterm, and final with random numbers ranging from 70 to 100 inclusive. Then it should print the student information via the toString() method. Student information provided by toString() should include name, homework, midterm, and final and the letter grade given by the calculateGrade() method.

3. Create file data.txt and add this line: “Welcome to class CSYE712 Parallelism, Concurrency, MultiThreading, Fall 2020”. Write a program that reads text line from file using BufferedReader and FileReader:

a) print it on the console

b) add it to a list

c) read it from the list and change it to upper case

d) count the number of strings

e) count the number of letters

e) add it to hashMap with key/value (string, frequency)

Note: first use HashSet to get frequencies,

f) sort the map using TreeMap

g) iterate through the map and print and provide Catch clause

for FileNotFoundException (file not found), and IOException (invalid file).

4. Create interface I that extends interface I1 with m1() and m2() methods, and interface I2 with m3() and m4() methods. Create class A that implements interface I. The implementation of each method must print the name of method. Create a Test class that instantiates an object of class A but uses interface I as its type. The Test class must execute all the methods of class A. Compile and run the code.

5. Compile and Run this code. Explain what and how it does it Step-by-Step?

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){

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

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();

}

}

Output

A screenshot of a computer

Description automatically generated

Explanation:

In caller class, we have created the object of the Input class and lock instance of the object class. 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.

6. What is the result of this Lambda function?

Runnable task = () -> {

String threadName = Thread.currentThread().getName();

System.out.println("Hello " + threadName);

};

task.run();

Thread thread = new Thread(task);

thread.start();

System.out.println(“Done!");

Output :

A screenshot of a cell phone

Description automatically generated

Explanation

The given program creates the runnable task using the lambda function inside the main thread and call the run method using the main thread and also create a separate thread and pass the runnable task to that thread object using the constructor and starts that thread. We can observe from the output screenshot that the last print statement “Done” executes before the Thread2.start() it shows the asynchronous behavior of multiple concurrent threads. The output may vary it depends on OS how it allocates the resources to each thread

7. Consider the following programs Example1 and Example2. You have no control over the order of running Threads, the operating system manages scheduling of running threads. However the code in example1 and example2 have differences that are observable if you look closely when running the programs.

a) Compile example1 and example2

b) Run each program

d) Read the code for both examples

b) Run each program

e) Identify the different behaviors between the two programs and Explain.

Example-1:

class Job implements Runnable {

private static Thread [] jobs = new Thread[4];

private int threadID;

public Job(int ID) {

threadID = ID;

}

public void run() { *do something* }

public static void main(String [] args) {

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

jobs[i] = new Thread(new Job(i));

jobs[i].start();

}

try {

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

jobs[i].join();

}

} catch(InterruptedException e) { System.out.println(e); }

}

}

Output:

A screenshot of a cell phone

Description automatically generated

Explanation:

In given program inside the main method using for loop we have create

4 different threads and assign job to each thread and start each thread

using **thread.start()** method. After that we used another loop and use

**thread.join()** method to makes sure that main thread is running until all

the threads finish their jobs. Here the **order of thread execution may**

**vary depend on how OS allocate the resources**

Example-2:

class Schedule implements Runnable {

private static Thread [] jobs = new Thread[4];

private int threadID;

public Schedule(int ID) {

threadID = ID;

}

public void run() { *do something* }

public static void main(String [] args) {

int nextThread = 0;

setPriority(Thread.MAX\_PRIORITY);

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

jobs[i] = new Thread(new Job(i));

jobs[i].setPriority(Thread.MIN\_PRIORITY);

jobs[i].start();

}

try {

for(;;) {

jobs[nextThread].setPriority(Thread.NORM\_PRIORITY);

Thread.sleep(1000);

jobs[nextThread].setPriority(Thread.MIN\_PRIORITY);

nextThread = (nextThread + 1) % jobs.length;

}

} catch(InterruptedException e) { System.out.println(e); }

}

}

Output:

A screenshot of a cell phone

Description automatically generated

**Explanation**:

In given program we have created 4 different threads and set the priority of the each thread to Min\_Priority (1), so now all threads have same priority so when we run the program order of threads execution are depend on OS because we have set the same priority to each thread. After that that we have use infinite for loop and in each iteration we are changing the priority of thread to NORM\_PRIORITY(5) in order of threadID like 0,1,2,3 then again 0,1,2,3 for 1 second and after that we again change the priority of that thread to 1. We have used the infinite loops so we need to terminate the program manually.

So in conclusion if we compare the behavior of the both example we can clearly see that in first example1 **order of thread execution is depend upon how OS allocate the resources to our program** while in second example we **tried to control the order of thread execution using setPriority method**.