Assignment no: 4

Q.1) What is Multithreading?

→
- Multithreading in java is a process of executing two or more threads simultaneously to maximum utilization of CPU.
- Multithread applications are where two or more thread runs concurrently. Hence it is also known as Concurrency in Java.
- This multitasking is done, when multiple processes share common resources like CPU, memory, etc.
- Each thread runs parallel to each other.
- Threads don't allocate seperate memory area; hence it saves memory. Also, context switching between threads takes less time.
- Advantages of Multithreading :-

a) The users are not blocked because threads are independent, and we can perform multiple operations at a times.

b) As such the threads are independent, the other threads won't get affected if one thread meets an exception.

- Example of Multithreading :-

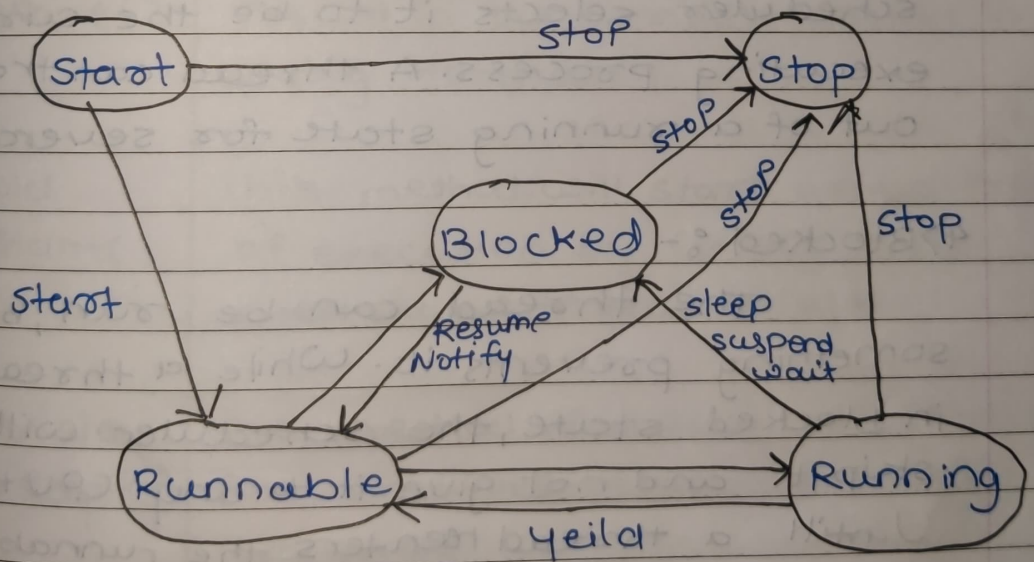
```
package demotest;
public class GuruThread1 implements Runnable
{
public static void main (String args[]){
Thread gurethread1 = new Thread ("Guru1");
```

```
Thread guruthread2 = new Thread ("Guru2");
    guruThread1.start();
    guruThread2.start();
System.out.println("Threads names are following.");
System.out.println("guruthread1.getName());
System.out.println("guruthread2.getName());
  }
  public void run(){
    }
  }
```

Q2) What are lifecycle of Thread?

→ • A thread is a lightweight subprocess within a process. It is an smallest unit of processing.

• Thread life cycle:-



1) New/start :-

This is the state the thread is in after the Thread instance has been created, but the start() method has not

been invoked on the thread. It is a live Thread object, but not yet a thread of execution. At this point, the thread is considered not alive.

2) Runnable:-

This means that a thread can be run when the time slicing mechanisms has CPU cycles available for the thread. Thus, thread might or might not be running at any movement.

3) Running:-

This state is important state where the action is performed. This is the state a thread is in when the thread scheduler selects it to be the currently executing process. A thread can transition out of a running state for several reasons

4) Blocked :-

The thread can be run, but something prevents it. While a thread is in blocked state, the scheduler will simply skip it and not give it any CPU time. Until a thread reenters the runnable state, it won't perform any operations.

.Blocked on I/O:-

The thread waits for completion of blocking operation. A thread can enter this state because of waiting

I/O resource.

- Blocked for join completion :-
  This thread can come in the state because of waiting for the completion of another thread.
- Blocked for lock acquistion :-
  The thread can come in this state because of waiting for the acquire the lock an object.

Q.3) What are different Thread methods which are used in Java?

→

method signature	Description
String getName()	Retrieves the name of the running thread in the current context in String format.
void start()	This method will start a new thread of execution by calling run() method of thread object.
void sleep (int sleeptime)	This method suspend the thread for mentioned time duration argument.
void run()	This method is the entay point of thread. Execution of thread starts from this method.

method signature	Description
void yeild()	By invoking this method the current thread pause its execution temporarily and allow other threads to execute.
void join()	This method used to queue up a thread in execution. Once called on thread, current thread will wait till calling thread completes its execution.
boolean isAlive()	This method will check if thread is alive or dead.

Q.4) Explain thread priorities and thread scheduling.

Q.5) How to create and execute threads? Write down a simple program for executing threads?

-> There are two ways to create threads :-

a) By extending thread class :-

We create a class that extends the java.lang.thread class. This class overrides run() method available in thread class.

Ex.

```
class MultithreadDemo extends Thread{
    public void run(){
        try{
        System.out.println("Thread" + Thread.
```

```
currentThread().getId() + "is running");
    }
  }
}
catch (Exception e){
  System.out.println ("Exception is caught");
  }
  }
}
public class Multithread{
public static void main (String args[]){
  int n=8;
  for (int i=0; i<8; i++){
  MultithreadingDemo obj = new MultithreadDemo();
    obj.start();
  }
}
}
```

·Output:-

```
Thread 0 is running
Thread 1 is running
Thread 2 is running
Thread 3 is running
Thread 4 is running
Thread 5 is running
Thread 6 is running
Thread 7 is running
```

b) by implementing Runnable Interface :-

We create new class which implements java.lang.Runnable interface and ~~over~~ overriden run() method. Then we instantiate Thread object and call start method on this object.

. Ex.

```
class Multithreaddemo implements Runnable {
  public void run(){
    try {
System.out.println("Thread"+Thread.currentThread().getId+"is running");
    }
catch(Exception e){
system.out.println("Exception is caught");
    }
  }
}
  Class Multithread {
public static void main(String args[]){
  int n=8;
  for(int i=0; i<8; i++){
Thread obj=new Thread(new multithreaddemo());
  Obj.start();
 }
}
```

. Output :-

```
Thread 0 is running
Thread 1 is running
Thread 2 is running
Thread 3 is running
```

Thread 4 is running
Thread 5 is running
Thread 6 is running
Thread 7 is running

Q.6) Explain Thread Synchronization.

→ • When we start two or more threads within program, there may be a situation when multiple threads to access the same resource and finally they can produce unforseen results due to concurrency issues.

• For example, if multiple threads to write within a same file then they may corrupt the data because one of the threads can override data or while one thread is opening same file at the same time another might be closing the same file.

• So there is a need to synchronize the action of multiple threads and make sure that only one thread can access the resource at a given point in time.

• Java programming language provides a very handy way of creating thread and synchronizing their task by ~~the~~ using synchronized blocks.

• Syntax:-

```
synchronized (objectidentifier){
//Access shared variables and other
//shared resources
}
```

- Here, the objectidentifier is a reference to an object whose lock associates with the monitor that the synchronized statement represents.

Q. → Write down a program for thread synchronization.

→

```
class PrintDemo {
  public void printCount() {
    try {
      for (int i = 5; i > 0; i--) {
        System.out.println("Counter---" + i);
      }
    } catch (Exception e) {
      System.out.println("Thread interrupted");
    }
  }
}

class ThreadDemo extends Thread {
  private Thread t;
  private String threadName;
  PrintDemo PD;
  ThreadDemo (String name, PrintDemo pd) {
    threadName = name;
    PD = pd;
  }
  public void run() {
    PD.printCount();
    System.out.println("Thread" + threadName + "existing");
  }
```

```
    public void start (){
    System.out.println ("starting"+ threadName);
     if(t==null){
    t=new Thread (this, threadName);
     t.start();
     }
   }
  }
public class TestThread{
public class void main (string args[]){
PrintDemo PD= new PrintDemo();
ThreadDemo T1 =new ThreadDemo("Thread-1", PD);
ThreadDemo T2=new ThreadDemo("Thread-2", PD);
 T1.start();
 T2.start();
 try{
 ~~try~~ T1.join();
      T2.join();
    } catch (Exception e){
    System.out.println("Interrupted");
    }
  }
 }
```

•Output :- Starting Thread-1
Starting Thread-2
Counter --5
Counter -4
Counter--3
Counter --5
Counter--2
Counter --1

Counter--4
~~Count~~ Thread Thread-1 existing
~~Tart~~ Counter--3
Counter--2
Counter--1
Thread Thread-2 existing

Q.8) Write down a program that implements Runnable Interface.

→
```
public class RunnableDemo{
public static void main (String args[]){
System.Out.println("main thread is-"
        +Thread.currentthread().getName());
Thread t1 = new Thread (new RunnableDemo().
                    new RunnableImpl());
    t1.start();
    }
private class RunnableImpl implements Runnable{
    public void run(){
    System.out.println(Thread.currentThread().getName()
            +", executing run() method!");
      }
    }
  }
```

.Output :-

Main thread is - main
Thread-0, executing run() method!.

Q.9) Explain following in details.

1. Race Condition:-

→ • Race condition in Java is type of concurrency bug or issue that is introduced in your program because of parallel execution of your program by multiple threads at same time. Since, java is multi-threaded programming language, hence, the risk of race condition is higher in which demands are understanding of what causes a race condition and how to avoid that.

• Anyway, race conditions are just one of the hazards or risks presented by the use of multiple-threading in Java just like deadlock in Java.

• Race condition occur when two internal operate on the same object without proper synchronization and their operation intervals on each others.

2. Thread class :-

The java.lang.Thread class is a thread of execution in a program. The JVM allows an application to have multiple thread of execution running concurrently.

• Every thread has a priority thread with higher priority are executed in preferences to threads with lower priority.

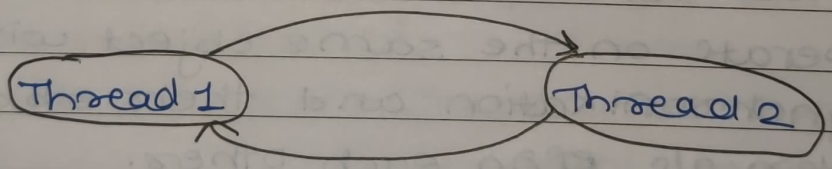
• Each thread may or may not also marked

as a daemon.

- There are two ways to create a threads:-
  a) Extending thread class.
  b) implementing Runnable Interface.

3. Deadlock:-

- Deadlock in java is a part of multithreading deadlock can occur in a situation when a thread is ~~writting~~ waiting for an object lock.
- That is acquired by the another thread and second thread is waiting for the object lock, that is acquired by another thread and second thread is acquired by first thread.
- Since both threads are waiting for each other to release lock the condition is called deadlock.



4) Explain thread priorities and thread scheduling.

→ • Thread Priorities in Multithreading:-

In multithreading environment, thread scheduler assigns processes or a thread based on priority of thread. Whenever we create a thread in Java it always has some priority assigned to it. Priority can either be given by JVM

while creating the thread or it can be given by programmer explicitly.

Accepted value of priority for a thread is in range of 1 to 10. There are 3 static variables defined in Thread class priority.

a) public static int MIN_PRIORITY :- This is minimum priority that a thread can have. value for this is 1.

b) public static int NORM_PRIORITY :- This is default priority of thread if donot explicitly define it. Value of this 5.

c) public static int MAX_PRIORITY :- This is maximum priority of thread. Value for this is 10.

1. public final int getPriority () :- It returns priority of given thread.
2. public final int setPriority (int newPriority ()) :- It changes the priority of thread to the value of newPriority.

- Thread scheduling :-

1) Most computer configuration have single CPU. Hence, thread runs one at a time in such a way to provide an illusion of concurrency.
2) Execution of multiple thread on single CPU in some order is called scheduling
3) Java runtime environment supports a very simple scheduling algorithm called fixed priority algorithm.
4) This algorithm schedules threads on the

basis of their priority relative to other Runnable threads.

5) When a thread is created it inherits its priority from thread that created it.

6) The Java Runtime System's thread scheduling algorithm is also preemptive. If at any time a thread with higher priority than all other Runnable threads becomes Runnable, the runtime chooses the new higher priority thread for execution.