**Multithreading Enhancements – Part-03**

* **Reentrant Lock:**

It is the implementation class of Lock interface and it is the direct child class of Object.

Reentrant means a thread can acquire same lock multiple times without any issue.

Internally reentrant lock increments threads personal count whenever we call lock method and decrements count value whenever thread calls unlock method and lock will be released whenever count reaches 0.

* **Constructors:**

ReentrantLock lock = new ReentrantLock();

Creates an instance of ReentrantLock.

ReentrantLock lock = new ReentrantLock(boolean fairness);

Creates ReentrantLock with the given fairness policy.

If the fairness is true, then the longest waiting thread can acquire the lock, if it available. That is it follows first come first served policy.

If the fairness is false, then which waiting thread will get the chance we can’t expect.

Note:

The default value for fairness is false.

Which of the following declarations are equal?

ReentrantLock l = new ReentrantLock();

ReentrantLock l = new ReentrantLock(true);

ReentrantLock l = new ReentrantLock(false);

All the above

1 and 3rd are equal.

* **Important methods of Reentrant Lock:**

void lock();

boolean tryLock();

boolean tryLock(long l, TimeUnit t)

void lockInterruptibly();

void unlock();

int getHoldCount()

Returns number of holds on this lock by current thread.

boolean isHeldByCurrentThread()

Returns true if and only if lock is hold by current thread.

int getQueueLength()

Returns number of threads waiting for the lock.

Collection getQueuedThreads()

It returns a collection of threads which are waiting to get the lock.

boolean hasQueuedThreads()

Returns true if any thread waiting to get the lock.

boolean isLocked()

Returns true if the lock is acquired by some thread.

boolean isFair()

Returns true if the fairness policy is set with true value.

Thread getOwner()

Returns the thread which acquires the lock.

Example:

import java.util.concurrent.\*;

class ReentrantLockDemo{

public static void main(String[] args){

ReentrantLock l = new ReentrantLock();

l.lock();

l.lock();

System.out.println(l.isLocked()); // true

System.out.println(l.isHeldByCurrentTh read()); //true

System.out.println(l.getQueuedLength()); //0

l.unlock();

System.out.println(l.getHoldCount()); //1

System.out.println(l.isLocked()); //true

l.unlock();

System.out.println(l.isLocked()); // false

System.out.println(l.isFair()); //false

}

}

class Display {

ReentrantLock l = new ReentrantLock();

public void wish(String name){

l.lock(); // Line-1

for(int i = 0; i<10; i++){

System.out.println(“Good morning:”);

try{

}catch(InterruptedException ie){}

System.out.println(name);

}

l.unlock(); // Line-2

}

}

class MyThread extends Thread{

Display d;

String name;

MyThread(Display d, String name){

this.d = d;

this.name = name;

}

public void run(){

d.wish();

}

}

class ReentrantLockDemo1{

public static void main(String[] args){

Display d = new Display();

MyThread t1 = new MyThread(d, “Dhoni”);

MyThread t2 = new MyThread(d, “YuvRaj”);

MyThread t3 = new MyThread(d, “Kohli”);

t1.start();

t2.start();

t3.start();

}

}

Note: If we comment Line-1 & 2, then threads will be executed simultaneously and we will get irregular output.

If we are not commenting Lines 1 & 2, then threads will be executed one by one and we will get regular output.

* **Demo program for try lock method:**

Import java.util.concurrent.locks.\*;

class MyThread extends Thread{

static ReentrantLock l = new ReentrantLock();

MyThread(String name){

super(name);

}

public void run(){

if(l.tryLock()){

System.out.println(Thread.currentThread().getName()+”…got the lock and performing safe operations”);

try{

Thread.sleep();

} catch(InterruptedException ie){}

l.unlock();

}

else{

System.out.println(Thread.currentThread().getName()+”…unable to get lock and hence performing alternative operations”);

}

}

}

class ReentrantDemo3{

public static void main(String[] args){

MyThread t1 = new MyThread(“First Thread”);

MyThread t2 = new MyThread(“Second Thread”);

t1.start();

t1.start();

}

}

Output:

First Thread …got lock and performing safe operations

Second Thread… unable to get lock and hence performing alternate operations.

* **Demo program for tryLock(long ms, TimeUnit):**

import java.util.concurrent.locks.\*;

import java.util.concurrent.\*;

class MyThread extends Thread{

static ReentrantLock l = new ReentrantLock();

MyThread(String name){

super(name);

}

public void run(){

do{

try{

if(l.tryLock(5000, TimeUnit.MILLISECONDS)){

System.out.println(Thread.currentThread().getName()+“…got the lock”);

Thread.sleep(30000);

l.unlock();

System.out.println(Thread.currentThread().getName()+”…releases lock”);

break;

} else{

System.out.println(Thread.currentThread().getName()+”…unable to get lock and will try again”);

}

} catch(InterruptedException ie){}

}while(true);

}

}

class ReentrantLockDemo4{

public static void main(String[] args){

MyThread t1 = new MyThread(“First Thread);

MyThread t2 = new MyThread(“First Thread);

t1.start();

t2.start();

}

}

Output:

