**Multithreading Enhancements – Part-02**

* **java.util.concurrent package:**

**The problems with traditional synchronized keyword**

1. We are not having any flexibility to try for a lock without waiting.
2. There is no way to specify maximum waiting time for a thread to get lock so that thread will wait until getting the lock which may create performance problems, which may cause deadlock.
3. If a thread releases the lock then which waiting thread will get that lock we are not having any control on this.
4. There is not API to list out all waiting threads for a lock.
5. The synchronized compulsory we have to use either at method level or within a method and it is not possible to use across multiple methods.

To overcome the above problems sun people introduced java.util.concurrent.locks package in 1.5 version

It also provides several enhancements to the programmer to provide more control on concurrency.

* **Lock Interface:**

Lock object is similar to implicit lock acquired by a thread to execute synchronized method or synchronized block.

Lock implementations provide more extensive operations than traditional implicit locks.

**Important methods for Lock Interface:**

void lock()

We can use this method to acquire a lock. If lock is already available then immediately current thread will get that lock.

If the lock is not already available then it will wait until getting the lock. It is exactly same behavior of traditional synchronized keyword.

boolean tryLock()

To acquire the lock without waiting. If the lock is available then then thread acquires that lock and returns true.

If the lock is not available then this method returns false and can continue its execution without waiting.

In this case thread never be entered into waiting state.

if(l.tryLock(){

//Perform safe operations

} else{

// Perform alternative operations.

}

boolean tryLock(long time, TimeUnit unit)

If lock is available then the thread will get the lock and continue its execution.

If the lock is not available then the thread will wait until specified amount of time. Still if the lock is not available then thread can continue its execution.

**TimeUnit:**

TImeUnit is an ENUM present in java.util.concurrent package.

enum TimeUnit{

NANOSECONDS,

MICROSECONDS,

MILLISECONDS,

SECONDS,

MINUTES,

HOURS,

DAYS;

}

**Example:**

if(l.tryLock(1000, TimeUnit.MILLISECONDS)

void lockInterruptibly()

Acquires a lock if it is available and returns immediately, if the lock is not available then it will wait.

While waiting if the thread is interrupted, then thread won’t get the lock.

void unlock()

To release the lock.

To call this method compulsory current thread should be owner of the lock, otherwise we will get runtime exception saying:

IllegalMonitorStateException