java.util.concurrent.locks 1.5

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 untill getting the lock, which may creates performance problems, which may cause deadlock.

(3) If a thread releases lock then which waiting thread will get that lock, we are not having any control on this.

(4) There is no API to list out all waiting threads for ~~the~~ a lock,

(5) The synchronized keyword compulsary we have to use either at method level or within the method and it is not possible to use across multiple methods

• To overcome these 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 implementation provides more extensive operation than traditional implicit locks

important methods of lock interface

(1) void lock()

we can use this method to acquired a lock if the lock is already available then immediately current thread will get that lock

if the lock is not available then it will wait until get the lock

it exactly same behavior of traditional synchronized keyword

(2) boolean tryLock()

- to acquire the lock without waiting if the lock is available then the 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 enters into waiting state

```
if( l.tryLock() ) {

        perform safe operation

    }
    else {
        perform Alternative operations
    }
```

(3) 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 untill specified amount of time
- Still if the lock is not available then thread can continue its execution

TimeUnit : is an enum present in java.util.concurrent package

```
enum    TimeUnit {

        NANOSECONDS,
        MICROSECONDS,
        MILLISECONDS,
        SECONDS,
        MINUTES,
        HOURS,
        DAYS ;
    }
```

ex:

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

}
```

(4) void lockInterruptibly()

- acquires the 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

(5) void unlock()

to releases a lock.

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