Multithreading

Two types of Multitasking

- Process-based Multitasking(Multiprocessing)
- Thread-based Multitasking(Multithreading)

- **Multiprocessing**: **A process is a running instance of a program. R**unning multiple processes at a time known as multiprocessing. Process is heavyweight.

- **Multithreading**: **Threads execute under the process.** Running multiple threads known as multithreading. Thread is basically a lightweight sub-process.

Advantage of Java Multithreading

1) It **doesn't block the user** because threads are independent and you can perform multiple operations at same time.

2) You **can perform many operations together so it saves time**.

3) Threads are **independent** so it doesn't affect other threads if exception occur in a single thread.

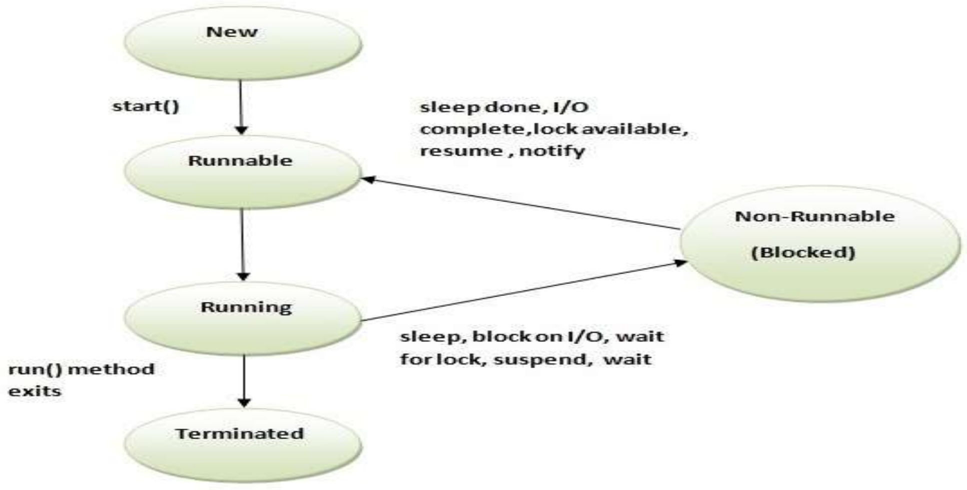
In Java there are two ways to create Threads:

- extends Thread or
- implements Runnable

```
Class Demo implements Runnable{

Public void run(){

}

}

Class Demo extends Thread{

Public void run(){

}

}
```

<u>Notes:</u> If our class already extends from some other class then we need to implement Runnable

Thread Lifecycle Methods


New
start()
Runnable
sleep done, I/O complete, lock available, resume , notify
Non-Runnable (Blocked)
Running
sleep, block on I/O, wait for lock, suspend, wait
run() method exits
Terminated

- Start(): Ready to Run
- Run(): We have to override run method and we need to provide business logic in run method
- Sleep(): We have to provide sleep time. It will pause execution. When sleep time expires it will automatically go to Ready to Run i.e Start method
- Stop(): Dead state (Deprecated now)

Born -> Ready to Start - > Running - > Sleep - > Dead state (Now it is deprecated)

Demo: Explain Life cycle of Thread

Questions?

1)Can we start a thread twice?

No. After starting a thread, it can never be started again. If you does so, an *IllegalThreadStateException* is thrown. **In such case, thread will run once but for second time, it will throw exception.**

2) What if we call run() method directly instead start() method?

- Each thread starts in a separate call stack.
- Invoking the run() method from main thread, the run() method goes onto the current call stack rather than at the beginning of a new call stack.

Methods from Thread Class:

currentThread(): to check current thread in the run method

setPriority(): Setting the priority of the thread (It's just a request we can't force)

join(): Join waits until thread to terminate

isAlive(): Returns Boolean value. To check weather thread is alive or not. If alive it will return true else false.

Yield()- give away to next thread of same priority or Higher priority(Again it is request)

Synchronization Thread safety

- Objective should be common(We required shared resources)
- To avoid race condition
- **Synchronization ensures that only one thread can access common object at a time**
- There are two ways to synchronize: Synchronize method or synchronize block

Concept of Lock in Java

Synchronization is built around an internal entity known as the lock or monitor. Every object has lock associated with it. By convention, a thread that needs consistent access to an object's fields has to acquire the object's lock before accessing them, and then release the lock when it's done with them.

Q: Which one is better Synchronized method or block?

Ans: a)Synchronized block is better because it synchronized only required method to be synchronized not all the methods within the run method b)We are using any third party jar file in our project source file of which is not with us and we want to use certain methods as synchronized we can put it under synchronized block.)

In Java 1.5 renetrant lock is introduced which provides same synchronized concept but we can lock and unlock explicitly (R.lock() and R.unlock())

Now let’s see how we can use Lock API without using synchronized keyword.

ConcurrencyLockExample.java

```java
package com.journaldev.threads.lock;

import java.util.concurrent.TimeUnit;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

public class ConcurrencyLockExample implements
Runnable{

    private Resource resource;
    private Lock lock;

    public ConcurrencyLockExample(Resource r){
        this.resource = r;
        this.lock = new ReentrantLock();
    }

    @Override
    public void run() {
        try {
            if(lock.tryLock(10, TimeUnit.SECONDS)){
            resource.doSomething();
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }finally{
            //release lock
            lock.unlock();
        }
        resource.doLogging();
    }

}
```
As you can see that, I am using tryLock() method to make sure my thread waits **only for definite time and i**f it’s not getting the lock on object, it’s just logging and exiting. Another important point to note is the use of try-finally block to make sure lock is released even if doSomething() method call throws any exception.

Lock vs synchronized

Based on above details and program, we can easily conclude following differences between Lock and synchronization.

1. Lock provides more visibility and options for locking, unlike synchronized where a thread might end up waiting indefinitely for the lock, **we can use tryLock() to make sure thread waits for specific time only.**
2. Synchronization code is much cleaner and easy to maintain whereas with **Lock we are forced to have try-finally block to make sure Lock is released even if some exception is thrown between lock() and unlock() method calls**.
3. synchronization blocks or methods can cover only one method whereas we can acquire the lock in one method and release it in another method with Lock API.

Inter thread Communication

Inter-thread communication or **Co-operation** is all about allowing synchronized threads to communicate with each other.

Cooperation (Inter-thread communication) is a mechanism in which a thread is paused running in its critical section and another thread is allowed to enter (or lock) in the same critical section to be executed. It is implemented by following methods of **Object class**:

- wait()
- notify()
- notifyAll()

wait():Causes current thread to release the lock and wait will pause the execution until another threads gives notification. We have parameterized wait method which will allow the thread to come out once the waiting period is over.

notify(): notify will give notification to single thread. It will not release lock

notifyAll():notify will give notification to all the threads which are in wait condition . It will also not release lock.

The awakened thread can not run until the code which called notify releases its lock.