DeadLock vs starvation

- Long waiting of a thread, where waiting never ends is termed "deadlock".
- Long waiting for a thread, where waiting ends at a certain point is called "starvation".

eg:: Assume we have 1cr threads, where all 1cr threads have priority is 10,but one thread is there which has priority 0,now the thread with a priority-0 has to wait for long time but still
it gets a chance,but it has to wait for a long time, this scenario is called "Starvation".

Note: Low priority thread has to wait until completing all priority threads but ends at a certain point which is nothing but starvation.

Daemon Threads

The thread which is executing in the background is called "DaemonThread".
eg: AttachListener,SignalDispatcher,GarbageCollector,....

remember the example of movie
1. producer
2. director
3. music director
4. ....
5. ....
6. ....

MainObjective of DaemonThread

The main objective of DaemonThread, to provide support for Non-Daemon threads(main thread).
eg:: if main threads run with low memory then jvm will call GarbageCollector thread, to destroy the useless objects,so that no of bytes of free memory will be improved with this free memory the main thread can continue its execution.

Usually Daemon threads have low priority,but based on our requirement daemon threads can run with high priority also.
JVM => creates 2 threads
a. Daemon Thread(priority=1,priority=10)
b. main (priority=5)
while executing the main code, if there is a shortage of memory then immediately jvm will change the priority of Daemon thread to 10, so Garbage collector activates Daemon thread and it frees the memory after doing it immediately it changes the priority to 1, so the main thread will continue.

How to check whether the Thread is Daemon or not?

public boolean isDaemon() => To check whether the thread is "Daemon"
public void setDaemon(boolean b) throws IllegalThreadStateException
b=> true,means the thread will become Daemon,before starting the Thread we need to make the thread as "Daemon" otherwise it would result in "IllegalThreadStateException".

What is the default nature of the Thread?

Ans. By default the main thread is "NonDaemon ".for all remaining thread Daemon nature is inherited from Parent to child, that is If the parent thread is "Daemon" then the child thread will become "Daemon" and if the parent thread is "NonDaemon" then automatically the child thread is also "NonDaemon".

Is it possible to change the NonDameon nature of Main Thread?

Ans. Not possible, becoz the main thread starting is not in our hands, it will be started by "JVM".

How to stop a thread ?

As for how to start a method we have a method called t.start().
Similarly to stop a thread we have a method called t.stop().
This method will kill a thread and it makes the thread enter into a dead state.
This is not a good approach to kill a thread, so this method is "deprecated".

CompleteLifeCycle of a Thread

```
MyThread t=new MyThread();//"new or born state"
t.start();//"ready or runnable state"
```

a. if T.S allocates CPU time, then thread will be in "running state".
b. if running thread calls
    1. Thread.yield() it enters into ready/runnable state(it is just pausing)
    2. t2.join()
        t2.join(1000)
        t2.join(1000,100);//Thread enters into waiting state
            a. if t2 finishes the execution
            b. if time expires
            c. if thread gets interruption [it comes back to ready/runnable state]
    3. Thread.sleep(1000,100);
        Thread.sleep(100);//Thread enters into sleeping state
        a. if time expires
            b. if thread gets interruption [it comes back to ready/runnable state]
    4. obj.wait()
        obj.wait(1000);
        obj.wait(1000,100); //Thread enters into waiting state
            a. if thread gets notification
            b. if time expires
            c. if thread gets interruption [it comes back to ready/runnable state]
    5. t.suspend()
            a. it enters into suspended state
                a. if it calls t.resume()
                    it enters into ready/runnable state
    6. t.stop()
            a. it enters into dead state
b. if run() is completed by a thread, then the thread enters into "deadstate".

InterThreadCommunication

Two threads can communicate each other with the help of
a. notify()
b. notifyAll()
c. wait()

notify()=> Thread which is performing updates should call notify(),so the waiting thread will get notification so it will continue with its execution with the updated items.
wait() => Thread which is expecting notification/updation should call
wait(), immediately the Thread will enter into a waiting state.

wait(),notifyAll(),notify() is present in Object class, but not in Thread class why?

- Thread will call wait(),notify(),notifyall() on any type of objects like Student,Customer,Engineer.
- If a thread wants to call wait(),notify()/notifyall() then compulsorily the thread should be the owner of the object otherwise it would result in "IllegalMonitorStateException".
- We say thread to be owner of that object if thread has lock of that object.
- It means these methods are part of synchronized block or synchronized method, if we try to use outside the synchronized area then it would result in a RunTimeException called "IllegalMonitorStateException".
- if a thread calls wait() on any object, then first it immediately releases the lock on that object and it enters into waiting state.
- if a thread calls notify() on any object,then he may or may not release the lock on that object immediately.

Method prototype of wait(),notify(),notifyAll()

1. public final void wait()throws InterruptedException
2. public final native void wait(long ms)throws InterruptedException
3. public final void wait(long ms,int ns)throws InterruptedException
4. public final native void notify()
5. public final void notifyAll()

Difference b/w notify and notifyAll()

notify() => To give notification only for one waiting thread
notifyAll() => To give notification for many waiting thread

=> We can use notify() method to give notification for only one Thread. If multiple Threads are waiting then only one Thread will get the chance and remaining Threads have to wait for further notification. But which Thread will be notified(inform) we can't expect exactly it depends on JVM.
=> We can use notifyAll() method to give the notification for all waiting Threads of a particular object. All waiting Threads will be notified and will be executed one by one, because they require lock.

Note: On which object we are calling wait(), notify() and notifyAll() methods that corresponding object lock we have to get but not other object locks.