Multi-threading

Introduction

A thread is a single sequential flow of control within a program. Threads are also called light weight processes. A program that has many threads running concurrently is called multithreading. *Java multi-threading is platform dependent. Thus, a multithreaded application could behave differently on different java implementations.*

How to create a thread?

Threads are implemented in the form of objects that contain the method called run( ) method. The run( ) method should be invoked by an object of the concerned thread. This can be achieved by creating the thread and initiating it with the help of another thread method called start( ) method. The run( ) method can be written in two ways.

- By extending Thread class
- By implementing Runnable interface

Thread class

The java.lang.Thread class is a concrete class, that is, it is not an abstract class, which encapsulates the behavior of a thread. To create a thread, the programmer must create a new class that should extends Thread class. The programmer must override the run( ) function of Thread to do useful work. This function is not called directly by the user, rather the user must call the start( ) method of thread, which in turn calls run( ). The following topic illustrate the use.

How to extends Thread class?

```
Class MyThread extends Thread {
    public void run() {
      // do some work
    }
}
class TestThread {
   public static void main(String[] args) {
      MyThread t1 = new MyThread();
      MyThread t2 = new MyThread();
      t1.start();
      t2.start();

   }
}
```

Runnable Interface

This interface has a single function run( ), which must be implemented by the class that implements the interface. The following topic shows the use of Runnable interface.

How to implements Runnable interface?

```
Class MyThread implements Runnable{
    public void run() {
      // do some work
    }
}
class TestThread {
   public static void main(String[] args) {
      MyThread a = new MyThread();
      Thread t1 = new Thread(a);
      MyThread b = new MyThread();
      Thread t2 = new Thread(b);
      t1.start();
      t2.start();
   }
}
```

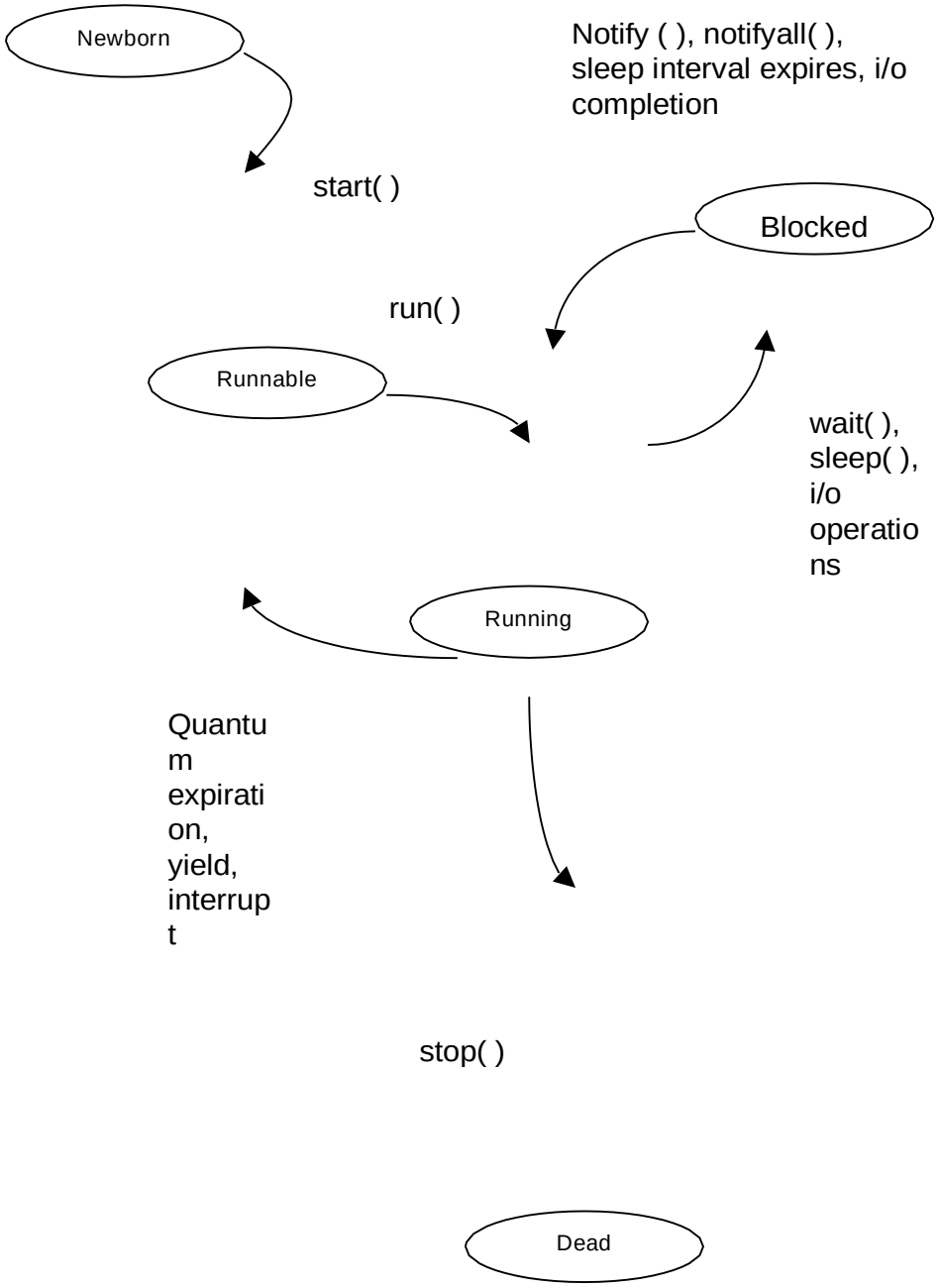
Transition diagram or state diagram of a thread

At any time, a thread is said to be in one of several thread states. They include:

- Newborn state
- Runnable state
- Running state
- Blocked state
- Dead state

When we create a thread object, the thread is said to be in the newborn state. The thread remains in this state until the thread's start method is called. This function causes the thread to enter the ready

or runnable state. The highest priority ready thread enters the running state when the system assigns a processor to the thread. A thread enters the dead state when its run method completes or terminates for any reason – a dead thread will eventually be disposed of by the system.



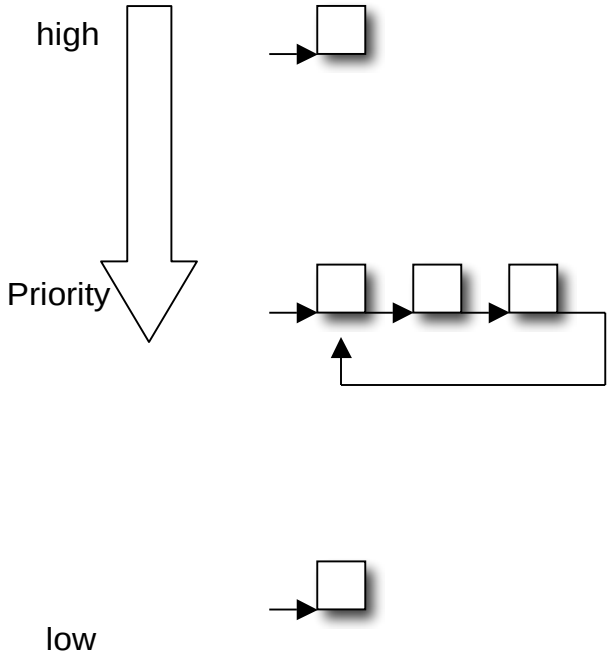
One way for a running thread to enter the blocked state is when the thread issues an input/output request. In this case, a blocked thread becomes ready when the I/O it is waiting for completes. A blocked thread cannot use processor if one is available. When a sleep method is called in a running thread, that thread enters the sleeping state. A sleeping thread becomes ready after the designated sleep time expires. When a running thread calls wait the thread enters a waiting state for the particular object on which wait was called. One thread in the waiting state for a particular object becomes ready on a call

to notify issued by another thread associated with that object.

Scheduling and Thread priority

Every thread has a priority value. It ranges from *Tread.MIN_PRIORIY* (1) to *Thread.MAX_PRIORITY* (10). By default, each thread is given priority *Thread.NORM_PRIORITY* (5). Each new thread inherits the priority of the thread that creates it.

If at any time a thread of a higher priority than the current thread becomes runnable, it preempts the lower-priority thread and begins executing. By default, threads at the same priority are scheduled round-robin, which means once a thread starts to run, it continues until it goes to blocked state by any reason.



The job of the java scheduler is to keep a highest priority thread running at all times, and If time-slicing is available, to ensure that several equally high-priority threads each execute for a quantum in round-robin fashion. A new entry of higher priority thread can postpone, possibly indefinitely, the execution of lower priority threads. Such indefinite postponement of lower priority thread is often referred as *starvation.*

Synchronization

Every thread has a life of its own. Normally, a thread goes about its business without any regard for what other threads in the application are doing. Synchronization is about coordinating the activities

of two or more threads, so they can work together and not collide in their use of the same address space. Java uses monitors (semaphore) to perform synchronization. Every object with the **synchronized** methods is a monitor. The monitor allows one thread at a time to execute a**synchronized** method on the object. The following program explains the use of **synchronized** keyword.

```
class CompleteName{
        synchronized void printName(String first, String second){
                System.out.print(first);
                try{
                        Thread.sleep(1000);
                }catch(InterruptedException e){
                        System.out.println("Interrupted");
                }
                System.out.println(" " + second);
        }
}

class SThread implements Runnable{
        String f,s;
        CompleteName cn;
        public SThread(CompleteName cn, String f, String s){
                this.cn = cn;
                this.f = f;
                this.s = s;
        }

        public void run(){
                cn.printName(f, s);
        }
}

class Synch{
  public static void main(String[] args){
    CompleteName cn = new CompleteName();
```

```
Thread t1 = new Thread( new SThread(cn,"Rupak","Shakya"));
Thread t2 = new Thread( new SThread(cn,"Sushil","Bajracharya"));
Thread t3 = new Thread( new SThread(cn,"Vivek","Agrawal"));
```

```
  t1.start();
  t2.start();
  t3.start();
}}
```