Thread class, Runnable interface, thread objects, thread object life cycle, thread scheduling, about thread methods start(), run(), sleep(), join() methods and examples.

-K.L.MADHAVI

Thread class

A **Thread class** has several methods and constructors which allow us to perform various operations on a thread. The Thread class extends the **Object** class. The **Object** class implements the **Runnable** interface. The thread class has the following methods that are used to perform various operations,they are:

- start(): The thread execution starts from this method.
- run(): Implicitly called by start().
- sleep(milliseconds): It suspends the thread for given milliseconds of time.
- join(): It waits the thread to complete its process. It is used in multithreading.
- getId(): It gives the id of a thread.
- getName(): It gives thread name and always starts from thread-0.
- setName(string): Thread name will be replaced with given string.
- getPriority(): Priority ranges from 1 to 10.{MIN_PRIORITY-1,NORM_PRIORITY-5,MAX_PRIORITY-10

Continue…

- setPriority(integer): Used to change priority of thread.
- isAlive(): -true: If thread is still running.

  -false: If thread completes its execution.

Example Program

```
Class Th extends Thread
{
    public void run()
    {
          Thread t=currentThread();
           for(int i=0;i<=3;i++)
           {
                  try
                  {
                           t.sleep(1000);
                   }
                  catch(Exception e)
                  {
                  }
                System.out.println(i);
         }
}

Class Tmethods
{
      public static void main(String args[])
       {
            Th t1=new Th();
            System.out.println(“ID=“+t1.getID()); //8
            System.out.println(“Name of the thread is :”+t1.getName());  //Thread-0
            t1.setName(“Madhu”);
            System.out.println(“Name of the thread after changing its name is :”+t1.getName());  //Madhu
            System.out.println(“Priority of the thread is :”+t1.getPriority()); //5
            t1.setPriority(10);
            System.out.println(“Priority of the thread after changing its priority  is :”+t1.getPriority()); //10
            t.start();
      }
}
```

```
Output:
ID=8
Name of the Thread is :Thread-0
Name of the thread after changing its name is :Madhu
Priority of the thread is :5
Priority of the thread after changing its priority  is :10
0
1
2
3
```

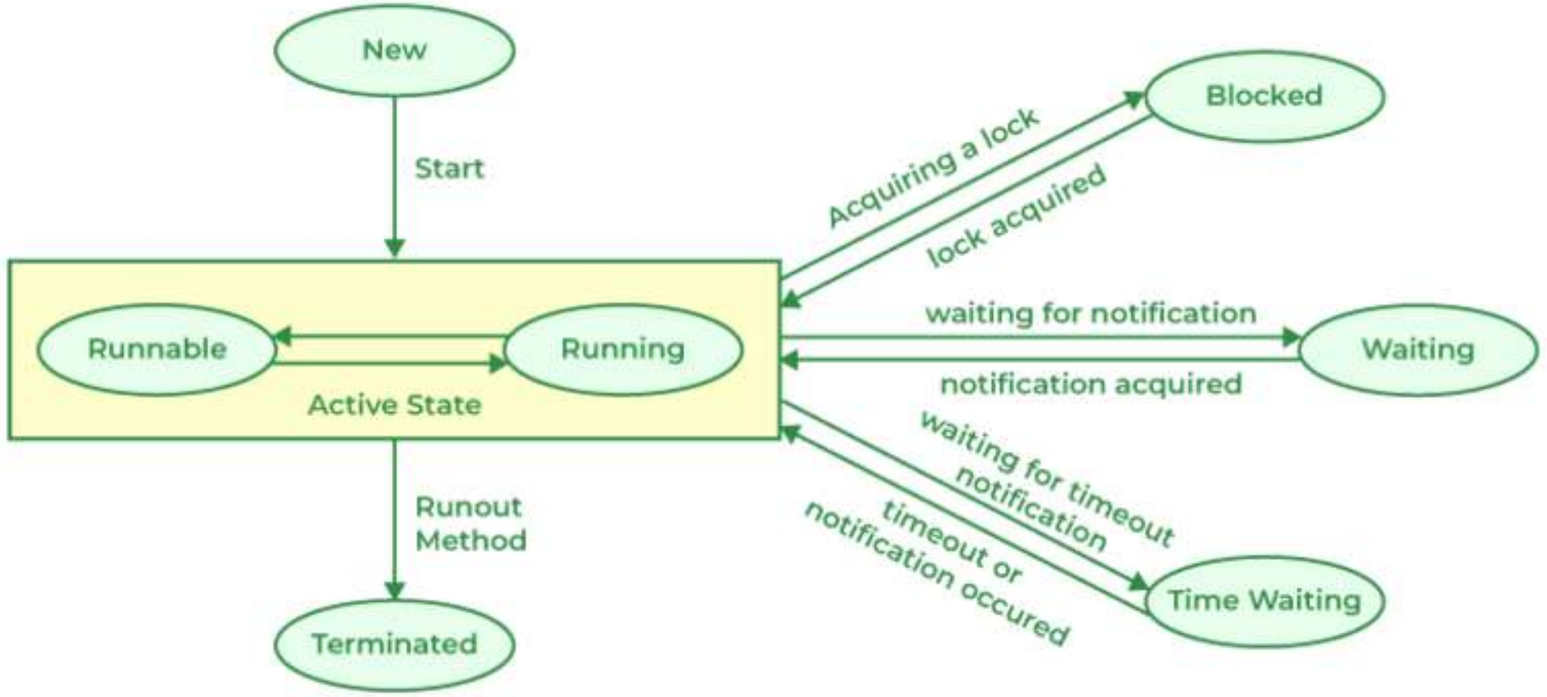
Implementing thread using Runnable Interface

The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread. Runnable interface have only one method named run(). This approach is preferred when we want to separate the task from the thread itself, promoting better encapsulation and flexibility.

public void run(): is used to perform action for a thread.

```
class  SingleThread implements Runnable
{
        public void run()
         {
                  System.out.println(“thread”);
         }
}
Class ST
{
     public static void main(String args[])
    {
          SingleThread s=new SingleThread();
          Thread t=new Thread(s);
          t.start(); //it implicitly calls the run() function
      }
}
```

Thread lifecycle



Continue…

There are multiple states of the thread in a lifecycle as mentioned below:

- **New Thread:** whenever the thread is created it will be in new state. The thread has not yet started to run when the thread is in this state. When a thread lies in the new state, its code is yet to be run and hasn’t started to execute.
- **Runnable state and running state:** If the program is ready to execute its task then it will be in Runnable state. In this state, a thread might actually be running or it might be ready to run at any instant of time. It is the responsibility of the thread scheduler to give the thread, time to run. A multi-threaded program allocates a fixed amount of time to each individual thread.
- **Blocked:** The thread will be in blocked state when it is trying to acquire a lock but currently the lock is acquired by the other thread. The thread will move from the blocked state to runnable state when it acquires the lock.
- **Waiting state:** The thread will be in waiting state when it calls wait() method or join() method. It will move to the runnable state when other thread will notify or that thread will be terminated.

Continue…

- **Timed Waiting:** A thread lies in a timed waiting state when it calls a method with a time-out parameter. A thread lies in this state until the timeout is completed or until a notification is received.

- **Terminated state:** A thread reaches the termination state because of the following reasons:

  i. When a thread has finished its job, then it exists or terminates normally.

  ii. Abnormal termination: It occurs when some unusual events such as an unhandled exception or segmentation fault.

Thread Scheduling

- Thread Scheduling refers to the process of determining which thread should run at a given time.
- The JVM and the OS work together to handle thread scheduling,ensuring fair and efficient use of CPU resources.

Java uses **preemptive priority-based scheduling model**,which works as follows:

- **Preemptive:** The scheduler can interrupt a running thread to allow another thread to run.
- **Priority-based:** Threads with higher priority are given preference over lower-priority threads.