**JAVA THREAD**

Typically, we can define threads as a subprocess with lightweight with the smallest unit of processes and also has separate paths of execution. These threads use shared memory but they act independently hence if there is an exception in threads that do not affect the working of other threads despite them sharing the same memory.

a thread runs inside the process and there will be context-based switching between threads there can be multiple processes running in OS, and each process again can have multiple threads running simultaneously. The Multithreading concept is popularly applied in games, animation…etc.

**The Concept Of Multitasking**

To help users Operating System accommodates users the privilege of multitasking, where users can perform multiple actions simultaneously on the machine. This Multitasking can be enabled in two ways:

1. **Process-Based Multitasking**
2. **Thread-Based Multitasking**

**1. Process-Based Multitasking (Multiprocessing)**

In this type of Multitasking, processes are heavyweight and each process was allocated by a separate memory area. And as the process is heavyweight the cost of communication between processes is high and it takes a long time for switching between processes as it involves actions such as loading, saving in registers, updating maps, lists, etc.

**2. Thread-Based Multitasking**

As we discussed above Threads are provided with lightweight nature and share the same address space, and the cost of communication between threads is also low.

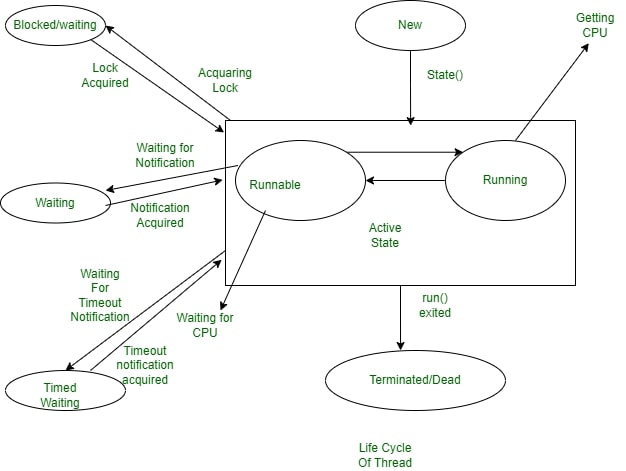
**Why Threads are used?**

Now, we can understand why threads are being used as they had the advantage of being lightweight and can provide communication between multiple threads at a Low Cost contributing to effective multi-tasking within a shared memory environment.

**Life Cycle Of Thread**

There are different states Thread transfers into during its lifetime, let us know about those states in the following lines: in its lifetime, a thread undergoes the following states, namely:

1. New State
2. Active State
3. Waiting/Blocked State
4. Timed Waiting State
5. Terminated State



We can see the working of  different states in a Thread in the above Diagram, let us know in detail each and every state:

**1. New State**

By Default, a Thread will be a new state,  in this state, code has not yet been run and the execution process is not yet initiated.

**2. Active State**

A Thread that is a new state by default gets transferred to Active state when it invokes the start() method, his Active state contains two sub-states namely :

* **Runnable State:**In This State, The Thread is ready to run at any given time and it’s the job of the Thread Scheduler to provide the thread time for the runnable state preserved threads. A program that has obtained Multithreading shares slices of time intervals which are shared between threads hence, these threads run for some short span of time and wait in the runnable state to get their schedules slice of a time interval.
* **Running State:**When The Thread Receives CPU allocated by Thread Scheduler, it transfers from “Runnable” state to “Running” state. and after the expiry of its given time slice session, it again moves back to the “Runnable” state and waits for its next time slice.

**3. Waiting/Blocked State**

If a Thread is inactive but on a temporary time, then either it is at waiting or blocked state, for example, if there are two threads, T1 and T2 where T1 need to communicate to the camera and other thread T2 already using a camera to scan then T1 waits until T2 Thread completes its work, at this state T1 is parked in waiting for the state, and in another scenario, the user called two Threads T2 and T3 with the same functionality and both had same time slice given by Thread Scheduler then both Threads T1, T2 is in a blocked state. When there are multiple threads parked in Blocked/Waiting state Thread Scheduler clears Queue by rejecting unwanted Threads and allocating CPU on a priority basis.

**4. Timed Waiting State**

Sometimes the longer duration of waiting for threads causes starvation, if we take an example like there are two threads T1, T2 waiting for CPU and T1 is undergoing Critical Coding operation and if it does not exit CPU until its operation gets executed then T2 will be exposed to longer waiting with undetermined certainty, In order to avoid this starvation situation, we had Timed Waiting for the state to avoid that kind of scenario as in Timed Waiting, each thread has a time period for which sleep() method is invoked and after the time expires the Threads starts executing its task.

**5. Terminated State**

A thread will be in Terminated State, due to the below reasons:

* Termination is achieved by a Thread when it finishes its task Normally.
* Sometimes Threads may be terminated due to unusual events like segmentation faults, exceptions…etc. and such kind of Termination can be called Abnormal Termination.
* A terminated Thread means it is dead and no longer available.

**What is Main Thread?**

As we are familiar that, we create Main Method in each and every Java Program, which acts as an entry point for the code to get executed by JVM, Similarly in this Multithreading Concept, Each Program has one Main Thread which was provided by default by JVM, hence whenever a program is being created in java, JVM provides the Main Thread for its Execution.

**How to Create Threads using Java Programming Language?**

We can create Threads in java using two ways, namely :

1. By extending Thread Class
2. By Implementing a Runnable interface

**1. By Extending Thread Class**

We can run Threads in Java by using Thread Class, which provides constructors and methods for creating and performing operations on a Thread, which extends a Thread class that can implement Runnable Interface. We use the following constructors for creating the Thread:

* Thread
* Thread(Runnable r)
* Thread(String name)
* Thread(String name, Runnable r)