SNS COLLEGE OF TECHNOLOGY
(AN AUTONOMOUS INSTITUTION)
COIMBATORE – 35
DEPARTMENT OF COMPUTER SIENCE AND ENGINEERING

UNIT IV
MULTITHREADING IN JAVA

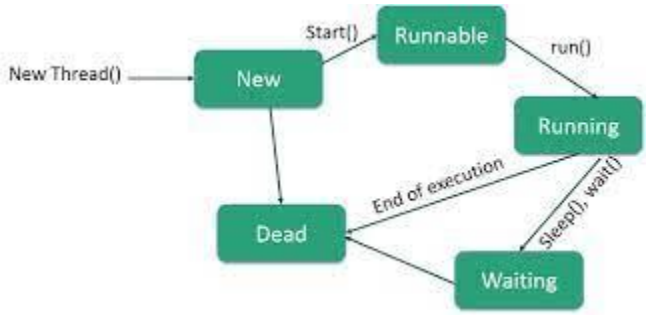
Thread Life Cycle

Introduction

Multithreading is a programming concept in which the application can create a small unit of tasks to execute in parallel.
example of a multithreaded program that we are all familiar with is a word processor. While you are typing, multiple threads are used to display your document, asynchronously check the spelling and grammar of your document, generate a PDF version of the document.

Java supports multithreading through Thread class. Java Thread allows us to create a lightweight process that executes some tasks. We can create multiple threads in our program and start them. Java runtime will take care of creating machine-level instructions and work with OS to execute them in parallel.
There are two types of threads in an application - **user thread** and **daemon thread**.

Life Cycle of a Thread



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

1. **New Thread:** When a new thread is created, it is in the 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.
2. **Runnable State:** A thread that is ready to run is moved to a 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. Each and every thread runs for a short while and then pauses and relinquishes the CPU to another thread so that other threads can get a chance to run. When this happens, all such threads that are ready to run, waiting for the CPU and the currently running thread lie in a runnable state.
3. **Blocked/Waiting state:** When a thread is temporarily inactive, then it's in one of the following states:
   - Blocked
   - Waiting
4. **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. For example, when a thread calls sleep or a conditional wait, it is moved to a timed waiting state.
5. **Terminated State:** A thread terminates because of either of the following reasons:
   - Because it exits normally. This happens when the code of the thread has been entirely executed by the program.
   - Because there occurred some unusual erroneous event, like a segmentation fault or an unhandled exception.

There are two ways to create a thread:

- By extending Thread class
- By implementing Runnable interface.

Thread class:

Thread class provide constructors and methods to create and perform operations on a thread.Thread class extends Object class and implements Runnable interface.

Commonly used Constructors of Thread class:
Thread()
Thread(String name)
Thread(Runnable r)
Thread(Runnable r,String name)

Commonly used methods of Thread class:

public void run(): is used to perform action for a thread.
public void start(): starts the execution of the thread.JVM calls the run() method on the thread.
public void sleep(long miliseconds): Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds.
public void join(): waits for a thread to die.
public void join(long miliseconds): waits for a thread to die for the specified miliseconds.
public int getPriority(): returns the priority of the thread.
public int setPriority(int priority): changes the priority of the thread.
public String getName(): returns the name of the thread.
public void setName(String name): changes the name of the thread.
public Thread currentThread(): returns the reference of currently executing thread.
public int getId(): returns the id of the thread.
public Thread.State getState(): returns the state of the thread.
public boolean isAlive(): tests if the thread is alive.
public void yield(): causes the currently executing thread object to temporarily pause and allow other threads to execute.
public void suspend(): is used to suspend the thread(depricated).
public void resume(): is used to resume the suspended thread(depricated).
public void stop(): is used to stop the thread(depricated).
public boolean isDaemon(): tests if the thread is a daemon thread.
public void setDaemon(boolean b): marks the thread as daemon or user thread.
public void interrupt(): interrupts the thread.
public boolean isInterrupted(): tests if the thread has been interrupted.
public static boolean interrupted(): tests if the current thread has been interrupted.

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().

public void run(): is used to perform action for a thread.
Starting a thread:
The start() method of Thread class is used to start a newly created thread. It performs the following tasks:

A new thread starts(with new callstack).
The thread moves from New state to the Runnable state.
When the thread gets a chance to execute, its target run() method will run.
1) Java Thread Example by extending Thread class

```
class Multi extends Thread{
public void run(){
System.out.println("thread is running...");
}
public static void main(String args[]){
Multi t1=new Multi();
t1.start();
 }
}
```

Output:

```
thread is running...
```

2) Java Thread Example by implementing Runnable interface
FileName: Multi3.java

```
class Multi3 implements Runnable{
public void run(){
System.out.println("thread is running...");
}
public static void main(String args[]){
Multi3 m1=new Multi3();
Thread t1 =new Thread(m1);   // Using the constructor Thread(Runnable r)
t1.start();
```

```
 }
}
```

Output:

```
thread is running...
```