Multithreading in Java

Multithreading in Java is a process of executing multiple threads simultaneously.

A thread is a lightweight sub-process, the smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.

However, we use multithreading than multiprocessing because threads use a shared memory area. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process.

Java Multithreading is mostly used in games, animation, etc.

Advantages of Java Multithreading

1) It **doesn't block the user** because threads are independent and you can perform multiple operations at the same time.

2) You **can perform many operations together, so it saves time**.

3) Threads are **independent**, so it doesn't affect other threads if an exception occurs in a single thread.

Multitasking

Multitasking is a process of executing multiple tasks simultaneously. We use multitasking to utilize the CPU. Multitasking can be achieved in two ways:

- o Process-based Multitasking (Multiprocessing)
- o Thread-based Multitasking (Multithreading)

1) Process-based Multitasking (Multiprocessing)

- o Each process has an address in memory. In other words, each process allocates a separate memory area.
- o A process is heavyweight.
- o Cost of communication between the process is high.
- o Switching from one process to another requires some time for saving and loading registers, memory maps, updating lists, etc.

2) Thread-based Multitasking (Multithreading)

- o Threads share the same address space.
- o A thread is lightweight.
- o Cost of communication between the thread is low.

What is Thread in java

A thread is a lightweight subprocess, the smallest unit of processing. It is a separate path of execution.

Threads are independent. If there occurs exception in one thread, it doesn't affect other threads. It uses a shared memory area.
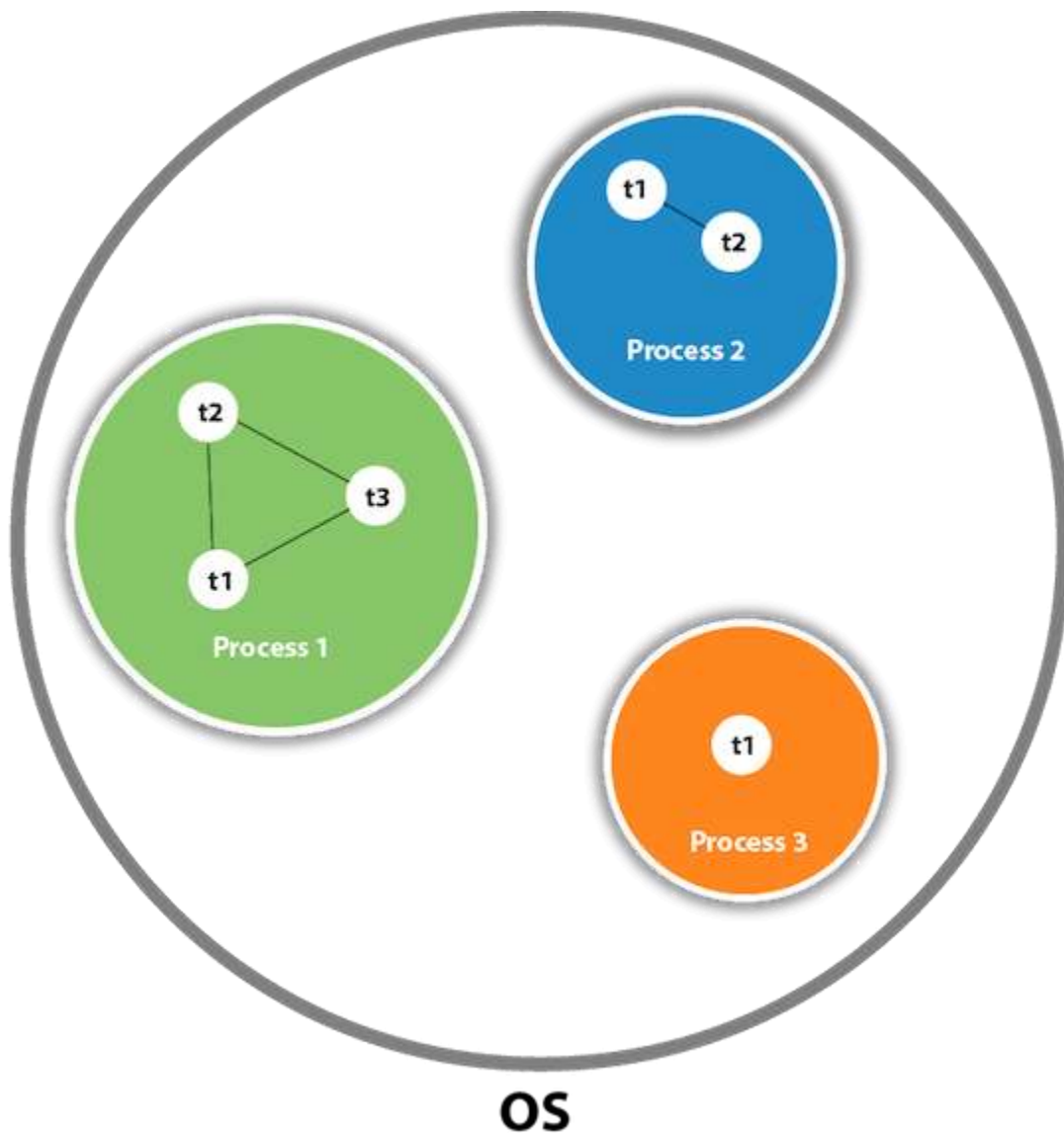
As shown in the above figure, a thread is executed inside the process. There is context-switching between the threads. There can be multiple processes inside the OS, and one process can have multiple threads.

Note: At a time one thread is executed only

Java Thread class

Java provides **Thread class** to achieve thread programming. Thread class provides constructors and methods to create and perform operations on a thread. Thread class extends Object class and implements Runnable interface.

Life Cycle of a Thread

A thread goes through various stages in its life cycle. For example, a thread is born, started, runs, and then dies. The following diagram shows the complete life cycle of a thread
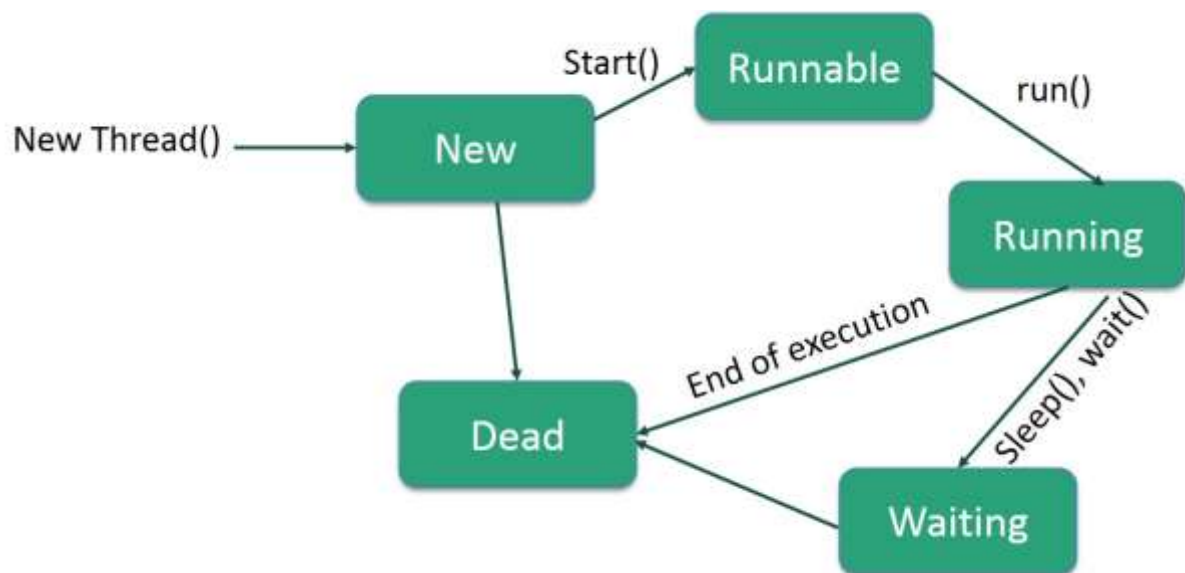
Following are the stages of the life cycle −

- **New** − A new thread begins its life cycle in the new state. It remains in this state until the program starts the thread. It is also referred to as a **born thread**.

- **Runnable** − After a newly born thread is started, the thread becomes runnable. A thread in this state is considered to be executing its task.

- **Waiting** − Sometimes, a thread transitions to the waiting state while the thread waits for another thread to perform a task. A thread transitions back to the runnable state only when another thread signals the waiting thread to continue executing.

- **Timed Waiting** − A runnable thread can enter the timed waiting state for a specified interval of time. A thread in this state transitions back to the runnable state when that time interval expires or when the event it is waiting for occurs.

- **Terminated (Dead)** − A runnable thread enters the terminated state when it completes its task or otherwise terminates.

Java Thread Methods

S.N.	Modifier and Type	Method	Description
1)	void	start()	It is used to start the execution of the thread.
2)	void	run()	It is used to do an action for a thread.
3)	static void	sleep()	It sleeps a thread for the specified amount of time.
4)	static Thread	currentThread()	It returns a reference to the currently executing thread object.