**Multithreading in Java**

**Multithreading in java** is a process of executing multiple threads simultaneously.

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

But we use multithreading than multiprocessing because threads share a common 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 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 exception occur 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 by two ways:

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

**1) Process-based Multitasking (Multiprocessing)**

* Each process have its own address in memory i.e. each process allocates separate memory area.
* Process is heavyweight.
* Cost of communication between the process is high.
* Switching from one process to another require some time for saving and loading registers, memory maps, updating lists etc.

**2) Thread-based Multitasking (Multithreading)**

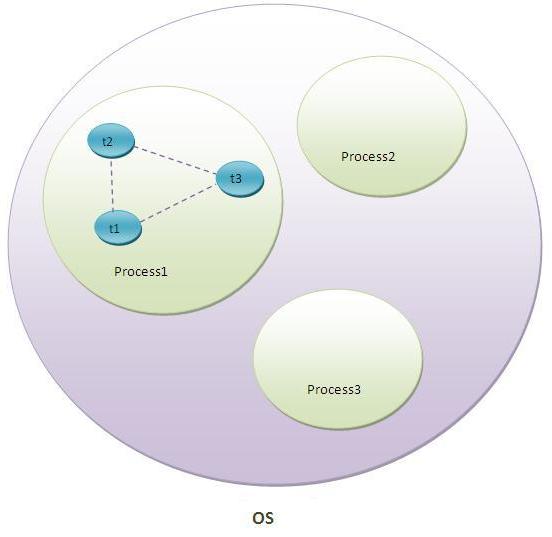
* Threads share the same address space.
* Thread is lightweight.
* Cost of communication between the thread is low.

**Note: At least one process is required for each thread.**

**What is Thread in java**

A thread is a lightweight sub process, a 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 shares a common memory area.



As shown in the above figure, 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.**

**Do You Know**

* How to perform two tasks by two threads ?
* How to perform multithreading by annonymous class ?
* What is the Thread Schedular and what is the difference between preemptive scheduling and time slicing ?
* What happens if we start a thread twice ?
* What happens if we call the run() method instead of start() method ?
* What is the purpose of join method ?
* Why JVM terminates the daemon thread if there is no user threads remaining ?
* What is the shutdown hook?
* What is garbage collection ?
* What is the purpose of finalize() method ?
* What does gc() method ?
* What is synchronization and why use synchronization ?
* What is the difference between synchronized method and synchronized block ?
* What are the two ways to perform static synchronization ?
* What is deadlock and when it can occur ?
* What is interthread-communication or cooperation ?

|  |
| --- |
| **Life cycle of a Thread (Thread States)**  A thread can be in one of the five states. According to sun, there is only 4 states in **thread life cycle in java** new, runnable, non-runnable and terminated. There is no running state.  The life cycle of the thread in java is controlled by JVM. The java thread states are as follows:   1. New 2. Runnable 3. Running 4. Non-Runnable (Blocked) 5. Terminated   thread life cycle in java  **1) New**  The thread is in new state if you create an instance of Thread class but before the invocation of start() method. |

**2) Runnable**

The thread is in runnable state after invocation of start() method, but the thread scheduler has not selected it to be the running thread.

**3) Running**

The thread is in running state if the thread scheduler has selected it.

**4) Non-Runnable (Blocked)**

This is the state when the thread is still alive, but is currently not eligible to run.

**5) Terminated**

A thread is in terminated or dead state when its run() method exits.