# Introduction to Multithreading in Java

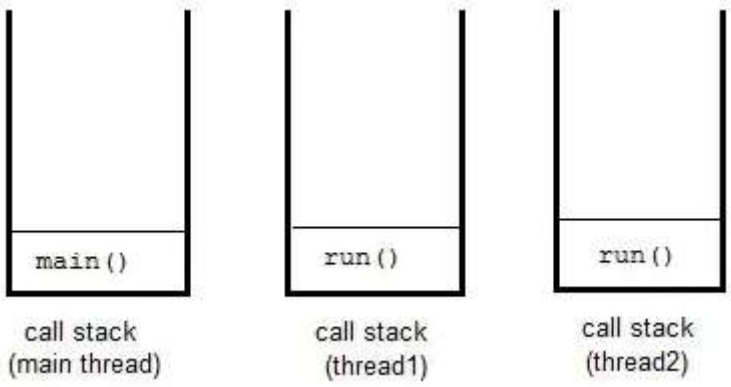
Thread is a lightweight unit of a process that executes in multithreading environment. Multithreading is a concept of running multiple threads simultaneously.

A program can be divided into a number of small processes. Each small process can be addressed as a single thread (a lightweight process). You can think of a lightweight process as a virtual CPU that executes code or system calls. Multithreaded programs contain two or more threads that can run concurrently and each thread defines a separate path of execution. This means that a single program can perform two or more tasks simultaneously. For example, one thread is writing content on a file at the same time another thread is performing spelling check.

In Java, the word thread means two different things.

**An instance of Thread class** or, A thread of execution.

An instance of Thread class is just an object, like any other object in java. But a thread of execution means an individual "lightweight" process that has its own call stack. In java each thread has it's own call stack.




**Advantage of Multithreading**

Multithreading reduces the CPU idle time that increase overall performance of the system. Since thread is lightweight process then it takes less memory and perform context switching as well that helps to share the memory and reduce time of switching between threads.

**Multitasking**

Multitasking is a process of performing multiple tasks simultaneously. We can understand it by computer system that perform multiple tasks like: writing data to a file, playing music, downloading file from remote server at the same time.

Multitasking can be achieved either by using multiprocessing or multithreading. Multitasking by using multiprocessing involves multiple processes to execute multiple tasks simultaneously whereas Multithreading involves multiple threads to executes multiple tasks.

**Why Multithreading ?**

Thread has many advantages over the process to perform multitasking. Process is heavy weight, takes more memory and occupy CPU for longer time that may lead to performance issue with the system. To overcome this issue of process, it is broken into small units of independent sub-process. These sub-processes are called threads that can perform independent task efficiently. So nowadays computer systems prefer to use thread over the process and use multithreading to perform multitasking.

**How to Create Thread ?**

To create a thread, Java provides a class Thread and an interface Runnable both are located into java.lang package.

We can create thread either by extending Thread class or implementing Runnable interface. Both includes a run method that must be override to provide thread implementation.

It is recommended to use Runnable interface if you just want to create a thread but can use Thread class for implementation of other thread functionalities as well.

**The main thread**

When we run any java program, the program begins to execute its code starting from the main method. Therefore, the JVM creates a thread to start executing the code present in main method. This thread is called as main thread. Although the main thread is automatically created, you can control it by obtaining a reference to it by calling currentThread() method.

Two important things to know about main thread are,

It is the thread from which other threads will be produced.

It must be always the last thread to finish execution.

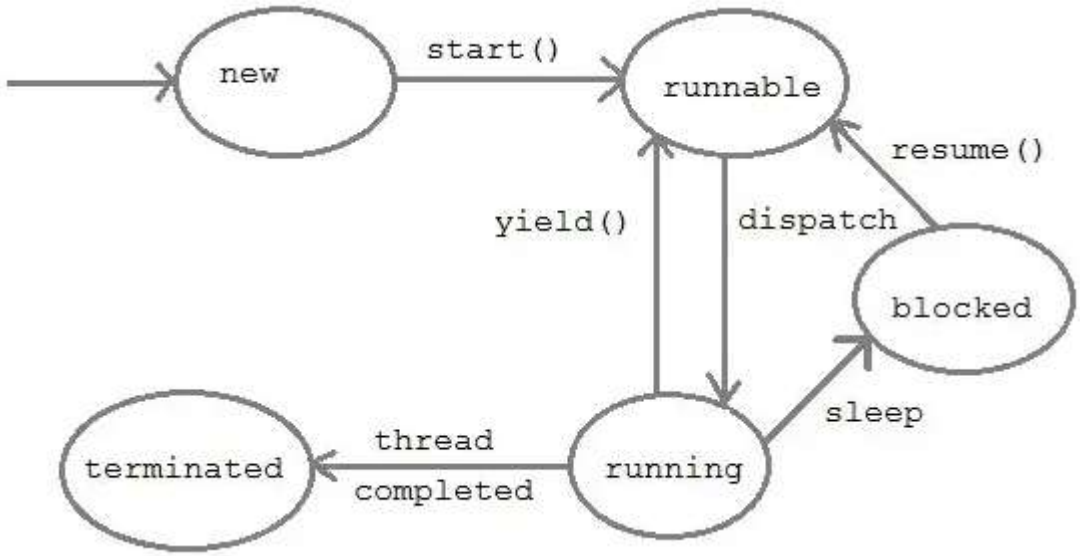
**Example:**

```
public class MainThread {
   public static void main(String[] args)
   {
      Thread t = Thread.currentThread();
      t.setName("MainThread");
      System.out.println(Thread.currentThread().getName());
   }
}
```

Output: MainThread

**Life cycle of a Thread**

Like process, thread have its life cycle that includes various phases like: new, running, terminated etc. we have described it using the below image.



1. New: A thread begins its life cycle in the new state. It remains in this state until the start() method is called on it.

2. Runnable: After invocation of start() method on new thread, the thread becomes runnable.

3. Running: A thread is in running state if the thread scheduler has selected it.

4. Waiting: A thread is in waiting state if it waits for another thread to perform a task. In this stage the thread is still alive.

5. Terminated: A thread enters the terminated state when it completes it's task.

**Thread Priorities**

In Java, when we create a thread, always a priority is assigned to it. In a Multithreading environment, the processor assigns a priority to a thread scheduler. The priority is given by the JVM or by the programmer itself explicitly. The range of the priority is between 1 to 10 and there are three variables which are static to define priority in a Thread Class.

Note: Thread priorities cannot guarantee that a higher priority thread will always be executed first than the lower priority thread. The selection of the threads for execution depends upon the thread scheduler.