**Multithreading in**[**Java**](https://www.javatpoint.com/java-tutorial) 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.

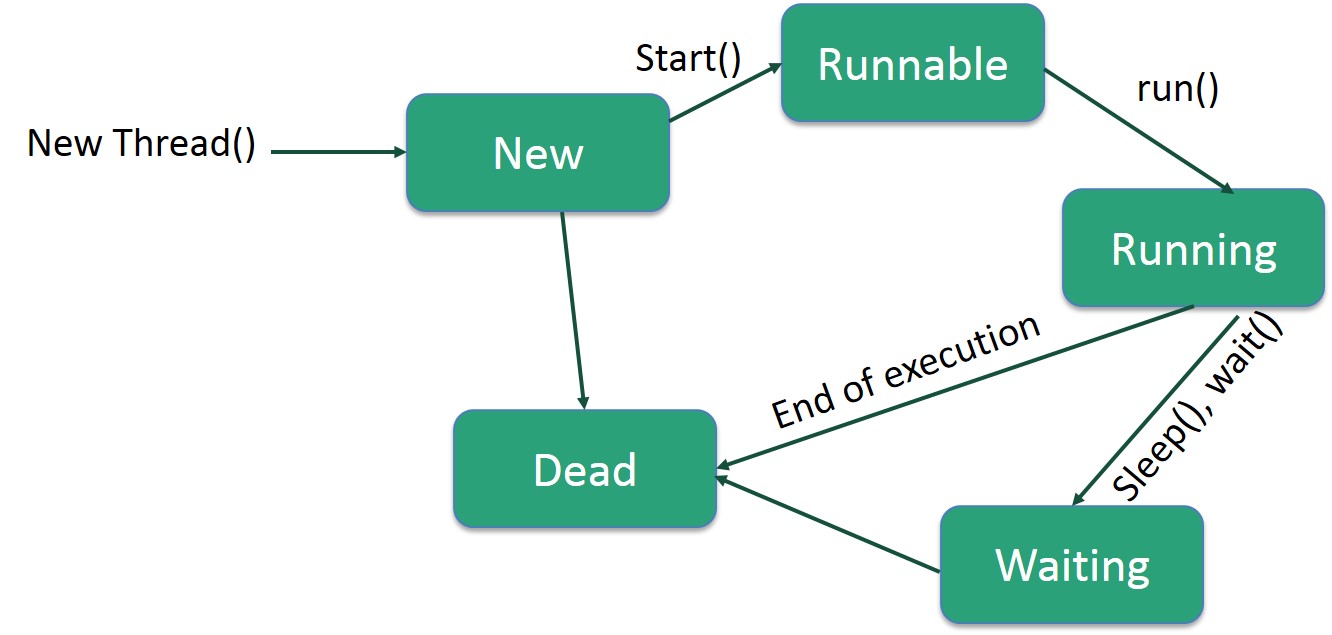
However, we use multithreading than multiprocessing because threads use a shared memory area. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process.

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

A program can have more than one thread of execution. The threads can share process global objects, like global variables ( which they usually do via controlled access - e.g. using objects that can only be locked by one thread at a time.)

Threads allow for a number of cool features-

* A thread can block on some condition without blocking the other threads
* It allows your program to take advantage of multiple processors on the computer and/or processors with multiple cores
* It allows for independent logical sections of the program to run with minimal effort; for example, imagine you want your program to spit out some statistics every minute; you could create a thread to do this easily; doing it manually by inserting some logic in your program’s main loop could be very cumbersome
* New
* Active
* Blocked / Waiting
* Timed Waiting



Example 1

class RunnableDemo implements Runnable {

private Thread t;

private String threadName;

RunnableDemo( String name) {

threadName = name;

System.out.println("Creating " + threadName );

}

public void run() {

System.out.println("Running " + threadName );

try {

for(int i = 4; i > 0; i--) {

System.out.println("Thread: " + threadName + ", " + i);

// Let the thread sleep for a while.

Thread.sleep(50);

}

} catch (InterruptedException e) {

System.out.println("Thread " + threadName + " interrupted.");

}

System.out.println("Thread " + threadName + " exiting.");

}

public void start () {

System.out.println("Starting " + threadName );

if (t == null) {

t = new Thread (this, threadName);

t.start ();

}

}

}

public class TestThread {

public static void main(String args[]) {

RunnableDemo R1 = new RunnableDemo( "Thread-1");

R1.start();

RunnableDemo R2 = new RunnableDemo( "Thread-2");

R2.start();

}

}

Output:

Creating Thread-1

Starting Thread-1

Creating Thread-2

Starting Thread-2

Running Thread-1

Thread: Thread-1, 4

Running Thread-2

Thread: Thread-2, 4

Thread: Thread-2, 3

Thread: Thread-1, 3

Thread: Thread-2, 2

Thread: Thread-1, 2

Thread: Thread-2, 1

Thread: Thread-1, 1

Thread Thread-2 exiting.

Thread Thread-1 exiting.

Example 2

**package** hello;

//Java code for thread creation by extending

//the Thread class

**class** MultithreadingDemo **extends** Thread {

**public** **void** run()

{

**try** {

// Displaying the thread that is running

System.***out***.println(

"Thread " + Thread.*currentThread*().getId()

+ " is running");

}

**catch** (Exception e) {

// Throwing an exception

System.***out***.println("Exception is caught");

}

}

}

//Main Class

**public** **class** Multithread {

**public** **static** **void** main(String[] args)

{

**int** n = 8; // Number of threads

**for** (**int** i = 0; i < n; i++) {

MultithreadingDemo object

= **new** MultithreadingDemo();

object.start();

}

}

}

Output:

Thread 15 is running

Thread 21 is running

Thread 20 is running

Thread 19 is running

Thread 16 is running

Thread 17 is running

Thread 22 is running

Thread 18 is running