## Multithreading

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:

1. Process-based Multitasking (Multiprocessing)
2. Thread-based Multitasking (Multithreading)

## 1) Process-based Multitasking (Multiprocessing)

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

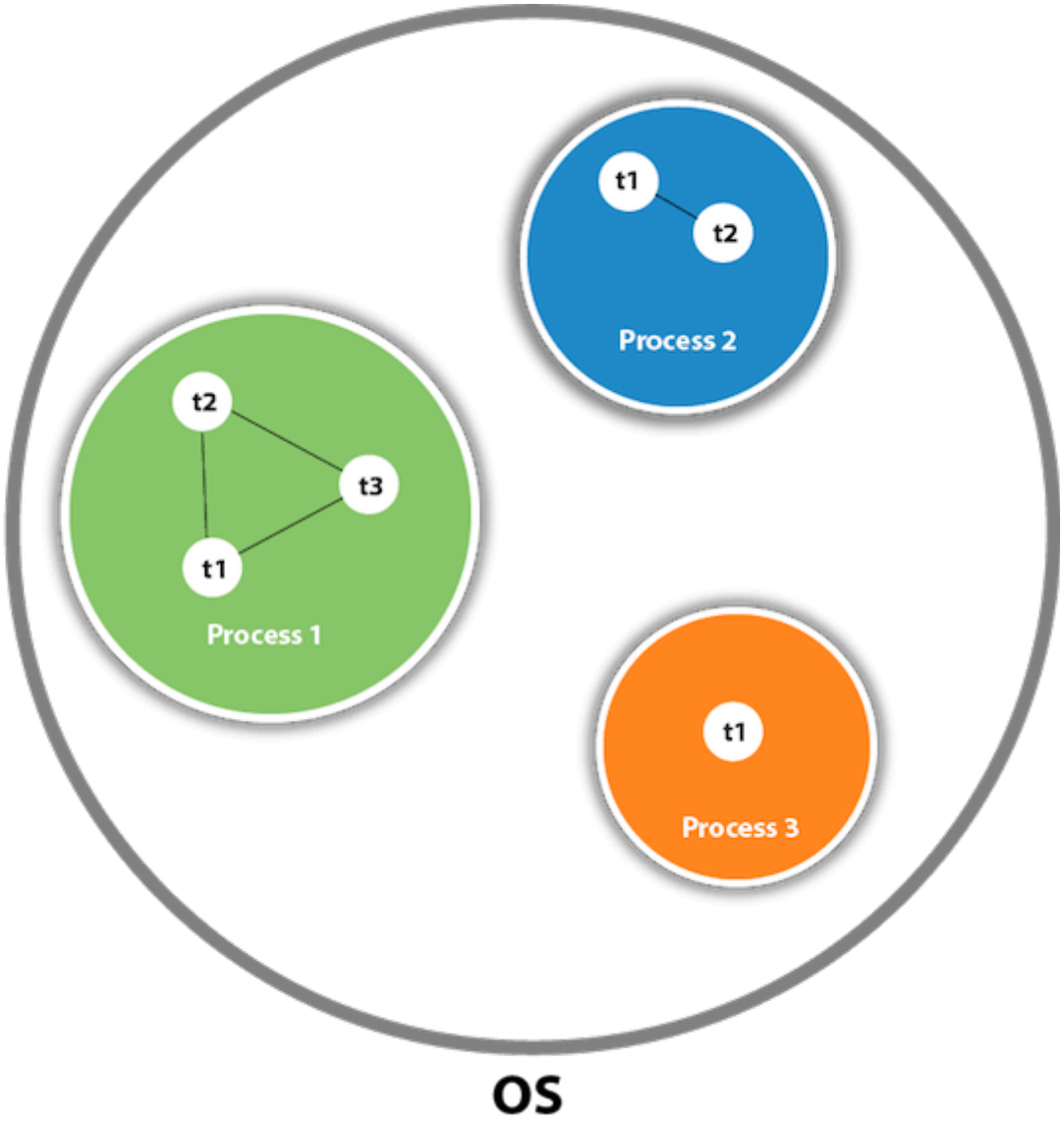
## 2) Thread-based Multitasking (Multithreading)

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

## Thread

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.

## 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 (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.

But for better understanding the threads, we are explaining it in the 5 states.

The life cycle of the thread in java is controlled by JVM. The java thread states are as follows:

- New
- Runnable
- Running
- Non-Runnable (Blocked)
- Terminated

## 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 Thread currentThread():** returns the reference of currently executing thread.
- **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 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().

**void run():** is used to perform action for a thread.

## **Starting a thread:**

**start() method** of Thread class is used to start a newly created thread. It performs 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();
 }
}
```

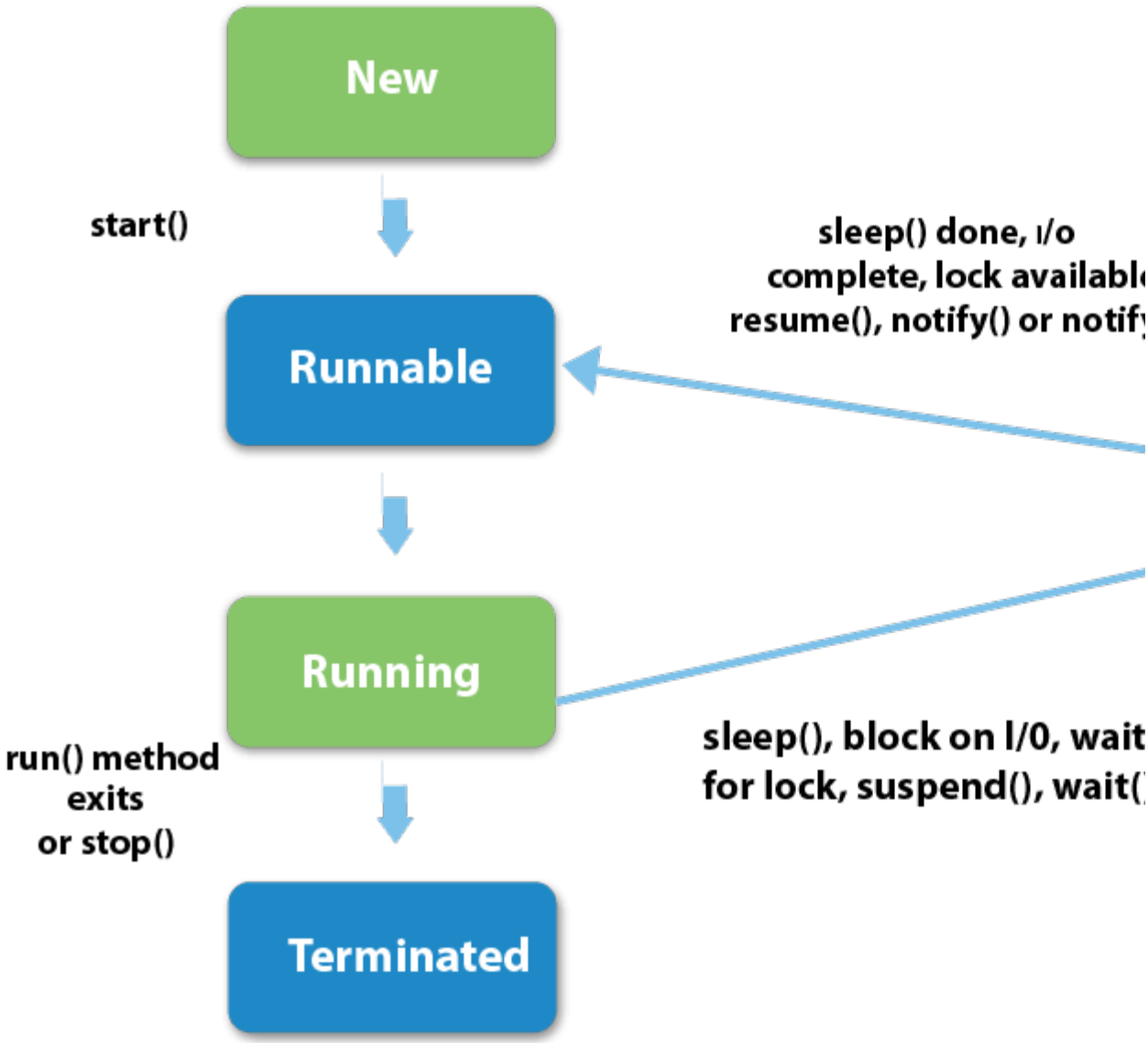
Output:thread is running...

**2) Java Thread Example by implementing Runnable interface**

```
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);
t1.start();
 }
}
```

**Output**:thread is running...

If you are not extending the Thread class,your class object would not be treated as a thread object.So you need to explicitely create Thread class object.We are passing the object of your class that implements Runnable so that your class run() method may execute.




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