Contents

[**Thread** 2](#_Toc34777924)

[**2 ways to create Thread in Java** 2](#_Toc34777925)

[**Implementing thread using Runnable Interface** 2](#_Toc34777926)

[**Important points about thread** 2](#_Toc34777927)

[**Thread scheduler** 2](#_Toc34777928)

[**Thread Priorities** 3](#_Toc34777929)

[**Thread States** 3](#_Toc34777930)

# **Thread**

Thread of execution is an individual Process that has its own call stack

## **2 ways to create Thread in Java**

1. Implement the Runnable Interface
2. Extend java.lang.Thread class

### **Implementing thread using Runnable Interface**

Public class ExampleClass implements Runnable{

@Override

public void run(){

System.out.println("Thread has ended");

}

Public static void main(String[] args){

ExampleClass test = new ExampleClass();

Thread t1 = new Thread(test);

t1.start();

System.out.println(“End of Main thread”);

}

}

## **Important points about thread**

* Starting a new thread can be done using thread.start() method
* When thread gets a chance to execute, its thread.run() method will run
* Current thread name can be accessed by using Thread.currentThread().getName()
* Prior to calling thread.start() method the thread is in “New” State
* Call start() on thread instance and not on runnable instance

## **Thread scheduler**

Thread scheduler in java is the part of the JVM that decides which thread should run.

There is no guarantee that which runnable thread will be chosen to run by the thread scheduler.

Only one thread at a time can run in a single process.

Difference between preemptive scheduling and time slicing

Under preemptive scheduling, the highest priority task executes until it enters the waiting or

dead states or a higher priority task comes into existence. Under time slicing, a task executes

for a predefined slice of time and then reenters the pool of ready tasks. The scheduler then

determines which task should execute next, based on priority and other factors.

## **Thread Priorities**

Each thread has a priority. Priorities are represented by a number between 1 and 10. In most cases, thread schedular schedules the threads according to their priority (known as preemptive scheduling). But it is not guaranteed because it depends on JVM specification that which scheduling it chooses.

**3 constants defined in Thread class:**

public static int MIN\_PRIORITY

public static int NORM\_PRIORITY

public static int MAX\_PRIORITY

Default priority of a thread is 5 (NORM\_PRIORITY). The value of MIN\_PRIORITY is 1 and the value of MAX\_PRIORITY is 10.

# **Thread States**

New – Thread is in new state if you create an instance of Thread class but before the invocation of start() method

Runnable – This is the state a thread is in if it is eligible to run but the scheduler has not selected it to be the running thread. A thread enters the Runnable state when the start() method is invoked but a thread can also return to the runnable state after either running or coming back from a blocked, waiting, or sleeping state. When the thread is in the runnable state, it is considered *alive*.

Running – This is the state a thread is in when the thread scheduler selects it (from the runnable pool) to

be the currently executing process. A thread can transition out of a running state for several reasons

Waiting/blocked/sleeping – This is the state a thread is in when it's not eligible to run.

Dead - A thread is considered dead when its run() method completes. It may still be a viable Thread object, but it is no longer a separate thread of execution. Once a thread is dead, it can never be brought back to life