Java Multithreading Tutorial for Beginners

[Java Tutorials](https://www.techbeamers.com/java-tutorials/) | By Meenakshi Agarwal

Greetings friends, in this post we are going to explain **Java multithreading** with examples. We’ll also cover the pros & cons of threading in Java and explore the thread lifecycle as well.

After that, we’ll give an overview of the thread class and its methods. And, finally, you’ll know how to create threads using the Thread class and the Runnable interface.

Also, we’ve supplemented the thread concepts with the help of classic examples and code samples. Now, before moving forward, let’s learn the generic **meaning of a process and thread**.

**What does a Process mean in Computer Science?**

A process consists of a memory space allocated by the operating system that can contain one or more threads. The thread cannot exist on its own; it must be a part of a process. A process remains running until all of the non-daemon threads are done executing.

**What does a Thread mean in Computer Science?**

* The thread is a lightweight sub-process and the smallest unit of execution.
* Each thread has a separate path of execution.
* It shares the memory area of the process.

Let’s now get started and explore the world of multithreading.

Java Multithreading Tutorial

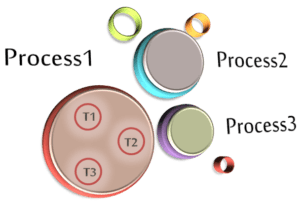
**Contents**[[hide](https://www.techbeamers.com/java-multithreading-with-examples/)]

* [1 What is multithreading in Java?](https://www.techbeamers.com/java-multithreading-with-examples/#what-is-multithreading-in-java)
* [2 What are the Pros and Cons of Java multithreading?](https://www.techbeamers.com/java-multithreading-with-examples/#what-are-the-pros-and-cons-of-java-multithreading)
  + [2.1 Pros:](https://www.techbeamers.com/java-multithreading-with-examples/#pros)
  + [2.2 Cons:](https://www.techbeamers.com/java-multithreading-with-examples/#cons)
* [3 What is Thread Lifecycle in Java?](https://www.techbeamers.com/java-multithreading-with-examples/#what-is-thread-lifecyclein-java)
* [4 Java Thread Class methods, usage, and examples](https://www.techbeamers.com/java-multithreading-with-examples/#java-thread-class-methods-usage-and-examples)
  + [4.1 List of Thread class Constructors.](https://www.techbeamers.com/java-multithreading-with-examples/#list-of-thread-class-constructors)
  + [4.2 General methods to manage the Threads.](https://www.techbeamers.com/java-multithreading-with-examples/#general-methods-to-manage-the-threads)
    - [4.2.1 start():](https://www.techbeamers.com/java-multithreading-with-examples/#start)
    - [4.2.2 run():](https://www.techbeamers.com/java-multithreading-with-examples/#run)
    - [4.2.3 String getName():](https://www.techbeamers.com/java-multithreading-with-examples/#string-getname)
    - [4.2.4 setName(String name):](https://www.techbeamers.com/java-multithreading-with-examples/#setnamestring-name)
      * [4.2.4.1 Sample code:](https://www.techbeamers.com/java-multithreading-with-examples/#sample-code)
      * [4.2.4.2 Sample output.](https://www.techbeamers.com/java-multithreading-with-examples/#sample-output)
    - [4.2.5 Sleep (long milliseconds):](https://www.techbeamers.com/java-multithreading-with-examples/#sleep-long-milliseconds)
    - [4.2.6 boolean isAlive():](https://www.techbeamers.com/java-multithreading-with-examples/#boolean-isalive)
      * [4.2.6.1 Check out the below sample code:](https://www.techbeamers.com/java-multithreading-with-examples/#check-out-the-below-sample-code)
    - [4.2.7 join(long milliseconds):](https://www.techbeamers.com/java-multithreading-with-examples/#joinlong-milliseconds)
      * [4.2.7.1 Check out the below sample code:](https://www.techbeamers.com/java-multithreading-with-examples/#check-out-the-below-sample-code-2)
    - [4.2.8 Set Thread Types.](https://www.techbeamers.com/java-multithreading-with-examples/#set-thread-types)
      * [4.2.8.1 Check out the below sample code:](https://www.techbeamers.com/java-multithreading-with-examples/#check-out-the-below-sample-code-3)
    - [4.2.9 Thread currentThread():](https://www.techbeamers.com/java-multithreading-with-examples/#thread-currentthread)
    - [4.2.10 Thread.State getState():](https://www.techbeamers.com/java-multithreading-with-examples/#threadstate-getstate)
      * [4.2.10.1 Check out the below sample code:](https://www.techbeamers.com/java-multithreading-with-examples/#check-out-the-below-sample-code-4)
      * [4.2.10.2 Sample output:](https://www.techbeamers.com/java-multithreading-with-examples/#sample-output-2)
    - [4.2.11 yield():](https://www.techbeamers.com/java-multithreading-with-examples/#yield)
    - [4.2.12 final int getPriority():](https://www.techbeamers.com/java-multithreading-with-examples/#final-int-getpriority)
    - [4.2.13 final void setPriority(int priority):](https://www.techbeamers.com/java-multithreading-with-examples/#final-void-setpriorityint-priority)
    - [4.2.14 int getId():](https://www.techbeamers.com/java-multithreading-with-examples/#int-getid)
    - [4.2.15 interrupt():](https://www.techbeamers.com/java-multithreading-with-examples/#interrupt)
    - [4.2.16 boolean isInterrupted():](https://www.techbeamers.com/java-multithreading-with-examples/#boolean-isinterrupted)
    - [4.2.17 boolean interrupted():](https://www.techbeamers.com/java-multithreading-with-examples/#boolean-interrupted)
    - [4.2.18 suspend():](https://www.techbeamers.com/java-multithreading-with-examples/#suspend)
    - [4.2.19 resume():](https://www.techbeamers.com/java-multithreading-with-examples/#resume)
    - [4.2.20 stop():](https://www.techbeamers.com/java-multithreading-with-examples/#stop)
* [5 Java Thread Creation Methods and Examples](https://www.techbeamers.com/java-multithreading-with-examples/#java-thread-creation-methods-and-examples)
  + [5.1 Create thread by extending the Thread class](https://www.techbeamers.com/java-multithreading-with-examples/#create-thread-by-extending-the-thread-class)
  + [5.2 Create a thread using the Runnable interface](https://www.techbeamers.com/java-multithreading-with-examples/#create-a-thread-using-the-runnable-interface)
* [6 Multithreading quiz – Must attempt](https://www.techbeamers.com/java-multithreading-with-examples/#multithreading-quiz-8211-must-attempt)
* [7 Download Java multithreading code samples](https://www.techbeamers.com/java-multithreading-with-examples/#download-java-multithreading-code-samples)

What Is Multithreading In Java?

* Java multithreading enables the execution of multiple threads simultaneously.
* It allows a process to run its tasks in parallel mode on a single processor system.

Now one question you may ask that what made this concept so famous. To get clarity, let’s see some of the benefits of using multithreading.



**Java Process and Threads**

What Are The Pros And Cons Of Java Multithreading?

Pros:

* Better use of system resources.
* Parallel execution of tasks and thus less execution time
* Enhanced performance on multi-processor machines
* Improved GUI responsiveness
* Independent threads (don’t impact other threads of the same process if an exception occurs)

Multithreading doesn’t always yield benefits. It comes with its disadvantages too. Let us go through some of the typical after results.

Cons:

* It results in the increased complexity of the code.
* Synchronization of shared resources (objects, data) is CPU/memory intensive.
* Debugging is hard because sometimes you can’t predict the results.
* Increased potential for deadlock occurrence.
* “Starvation” some of the threads may not be served with due to poor design.

What Is Thread Lifecycle In Java?

**Java Thread Life Cycle**

Before moving on to creating a thread, we’ll first see the various phases through which the thread has to pass to complete its lifespan.

* New
* Runnable
* Running
* Non-Runnable (blocked)
* Terminated

**New-**A thread is in New state when you create the instance of thread class, but the start() method is yet to get called.

**Runnable-**The thread is in runnable after executing the start() method. In this stage, it waits for the thread scheduler to run it.

**Running-**A thread picked by the thread scheduler for execution remains in running state.

**Non-Runnable (blocked)-**In this state, the thread remains alive, but it is not eligible to run. It may be due to some sleep operation, waiting for any File I/O to finish or in the locked state, etc.

**Terminated-**A thread is said to be in the terminated state when it’s Run() method exits.

So far, you’ve read the basic concepts of Java multithreading, let’s now move to learn the different ways to create threads.

How To Create Threads In Java?

In Java, we’ve got the following two ways to create threads.

* **Extend the Thread class, or**
* **Implement the Runnable interface.**

It is essential to learn that Java has supplied most of the thread functionality into the Thread class. So, before we jump to the thread creation part, let’s first understand the different features and methods available in the Thread class.

Java Thread Class Methods, Usage, And Examples

Java accomplishes multithreading through its java.lang.Thread class. To become efficient in writing the multithreaded code you must know about the constructors and the methods of thread class before starting to write multithreading programs in Java.

List Of Thread Class Constructors.

Thread class has the following eight constructors.

**Thread():** It creates a Thread object with a default name.

**Thread(String name):** It creates a Thread object with a name that the name argument specifies.

**Thread (Runnable target):** This method constructs Thread with a parameter of the Runnable object that defines the run() method.

**Thread (Runnable target, String name):** This method creates Thread with a name and a Runnable object parameter to set the run() method.

**Thread(ThreadGroup group, Runnable target):** It creates a Thread object with a Runnable object and the group to which it belongs.

**Thread (ThreadGroup group, Runnable target, String name):** It creates Thread object with a Runnable object that defines the run() method, specified name as its name and the Thread object belongs to the ThreadGroup referred to by the group.

**Thread(ThreadGroup group, String name):** It creates a thread that has the specified name and associates to the ThreadGroup given as the first parameter.

**Thread(ThreadGroup group, Runnable target, String name, long stackSize):** this constructor specifies the ThreadGroup parameter, the size of the thread’s method-call stack.

General Methods To Manage The Threads.

Start():

When program calls start() method a new Thread is created, and code inside run() method gets executed in a new Thread.

Run():

This method is the Entry point for the thread.

Let’s see a sample code that shows how to use the start() and run() methods.

package com.techbeamers.multithreading;

class MultithreadDemo extends Thread{

public void run(){

System.out.println("My thread is in running state.");

}

public static void main(String args[]){

MultithreadDemo obj=new MultithreadDemo();

obj.start();

}

}

Output:

My thread is in running state.

String GetName():

You can use this method to get the name of the current thread.

SetName(String Name):

You can use to set the name of the thread.

During a debugging session, it’s helpful as we can distinguish between threads. By default, Java also sets the label for each Thread. Alternatively, you can also give a name to the Thread using the Thread constructor that takes the name as an argument. The method setName(String name) can also update the Thread name.

Let’s see a sample code that shows how to give a name to the Thread using the Thread(String name) constructor, setName(String name) method and retrieve that name in the run() function using getName() method.

Sample Code:

package com.techbeamers.multithreading;

class NameMyThread

{

public static void main (String [] args)

{

MyThread mt;

if (args.length == 0)

mt = new MyThread ();

else

mt = new MyThread (args [0]);

mt.start ();

}

}

class MyThread extends Thread

{

MyThread ()

{

// The compiler creates the byte code equivalent of super ();

}

MyThread (String name)

{

setName (name); // Pass name to Thread superclass

}

public void run ()

{

System.out.println ("My name is: " + getName ());

}

}

Sample Output.

If you forget to provide any name while running the code, you’ll see the following output.

Java NameMyThread

Output:

My name is: Thread-0

If you give a name to a thread as “DemoThread,” then the output will be.

Java NameMyThread DemoThread

Output:

My name is: DemoThread

Sleep (Long Milliseconds):

It suspends a thread for the specified period. If any other thread interrupts this sleeping thread, it will throw an InterruptedException. So it is suggested to enclose the sleep() method in the try block. Alternatively, the code’s method must include InterruptedException in its throws clause.

Boolean IsAlive():

It determines if a thread is still running. JVM considers a thread to be alive immediately before calling to thread’s run() method, during the execution of thread’s run() and immediately after return from run().

During that interval, the isAlive() method returns a “true” Boolean value. Otherwise, it returns false.

This method is useful in the situations when one thread needs to wait for another thread to finish its run() method.

Check Out The Below Sample Code:

package com.techbeamers.multithreading;

class MyThread extends Thread{

public void run(){

System.out.println("My thread is in running state.");

}

}

class ThreadSleepDemo{

public static void main(String args[]){

MyThread obj=new MyThread();

obj.start();

while(obj.isAlive())

{

try

{

obj.sleep(10);

}

catch(InterruptedException e)

{

System.out.println(“Sleeping thread interrupted”);

}

System.out.println(“Thread-Sleep Demo Complete”);

}

}

}

Output:

My thread is in running state

Thread-Sleep Demo Complete

Join(Long Milliseconds):

This method gets called when a thread wants to wait for another thread to terminate. Let’s see a sample code for the **<join()>** method.

Check Out The Below Sample Code:

package com.techbeamers.multithreading;

class MyThread extends Thread{

public void run(){

System.out.println("My thread is in running state.");

}

}

class ThreadJoinDemo{

public static void main(String args[]){

MyThread obj=new MyThread();

obj.start();

try

{

obj.join();

}

catch(InterruptedException e)

{

}

System.out.println(“Thread-Join Demo Complete”);

}

}

}

Output:

My thread is in running state

Thread-Join Demo Complete

Set Thread Types.

Threads fall under into following two categories.

* User, and
* daemon.

A user thread performs critical tasks that must finish before the application terminates. However, a daemon thread performs garbage collection and other background tasks. When an application’s starting thread (a user thread) ends, JVM checks whether any other user thread is running. If there is some running user thread, then JVM prevents the application from terminating. If there is no user thread running, then JVM will end the program without bothering for the running daemon thread.

* **Boolean isDaemon():** checks if the thread is a daemon thread.
* **setDaemon(boolean b):** marks the thread as daemon or user thread. To start a daemon thread setDaemon(Boolean b) method must be called with an argument “**true**.”

Check Out The Below Sample Code:

package com.techbeamers.multithreading;

class DaemonThreadDemo

{

public static void main (String [] args)

{

MyThread mt = new MyThread ();

mt.setDaemon (true);

mt.start ();

try

{

Thread.sleep (100);

}

catch (InterruptedException e)

{

}

}

}

class MyThread extends Thread

{

public void run ()

{

System.out.println ("Daemon is " + isDaemon ());

}

}

Output:

Daemon is true

Thread CurrentThread():

It returns the instance reference of the currently executing thread.

Thread.State GetState():

It returns the state of the thread.

Check Out The Below Sample Code:

package com.techbeamers.multithreading;

class ThreadStateTest

{

public static void main (String [] args)

{

Thread currentThread = Thread.currentThread();

System.out.println(currentThread);

MyThread mt1 = new MyThread ();

mt1.setName("MyThread1");

MyThread mt2 = new MyThread();

mt1.setName("MyThread2");

System.out.println("Thread State of MyThread1 before calling start: "+mt1.getState());

mt1.start ();

mt2.start();

System.out.println("Thread State of MyThread1 in Main method before Sleep: " + mt1.getState());

System.out.println("Thread State of MyThread2 in Main method before Sleep: " + mt2.getState());

try

{

Thread.sleep (1000);

}

catch (InterruptedException e)

{

}

System.out.println("Thread State of MyThread1 in Main method after Sleep: " + mt1.getState());

System.out.println("Thread State of MyThread2 in Main method after Sleep: " + mt2.getState());

}

}

class MyThread extends Thread

{

public void run ()

{

System.out.println ("Run by " + Thread.currentThread().getName());

try

{

Thread.sleep (100);

}

catch (InterruptedException e)

{

}

System.out.println("Thread State of: "+ Thread.currentThread().getName()+ " - "+Thread.currentThread().getState());

System.out.println("Exit of Thread: " + Thread.currentThread().getName());

}

}

Sample Output:

Output:

Thread[main,5,main]

Thread State of MyThread1 before calling start: NEW

Run by MyThread2

Thread State of MyThread1 in Main method before Sleep: RUNNABLE

Run by Thread-1

Thread State of MyThread2 in Main method before Sleep: RUNNABLE

Thread State of: MyThread2 - RUNNABLE

Exit of Thread: MyThread2

Thread State of: Thread-1 - RUNNABLE

Exit of Thread: Thread-1

Thread State of MyThread1 in Main method after Sleep: TERMINATED

Thread State of MyThread2 in Main method after Sleep: TERMINATED

Yield():

This method causes the currently executing thread object to pause temporarily and allow other threads to run.

Check out the below sample code:

package com.techbeamers.multithreading;

public class ThreadTest extends Thread

{

public void run()

{

System.out.println("In run");

yield();

System.out.println("Leaving run");

}

public static void main(String []argv)

{

(new ThreadTest()).start();

}

}

The thread will halt after printing “In Run” due to the yield() method; since there is no other thread to execute so this thread will resume and print “Leaving run.”

Output:

In Run

Leaving Run

Final Int GetPriority():

It returns the priority of the thread.

Final Void SetPriority(Int Priority):

This function is used to change the priority of a thread.

Check out the below sample code:

package com.techbeamers.multithreading;

public class ThreadDemo {

public static void main(String[] args) {

Thread t = Thread.currentThread();

t.setName("Admin Thread");

// set thread priority to 1

t.setPriority(1);

// prints the current thread

System.out.println("Thread = " + t);

int priority= t.getPriority();

System.out.println("Thread priority= " + priority);

int count = Thread.activeCount();

System.out.println("currently active threads = " + count);

}

}

Output:

Thread = Thread[Admin Thread,1,main]

Thread priority= 1

currently active threads = 1

Int GetId():

It returns the id of the thread.

Interrupt():

It interrupts the thread.

Boolean IsInterrupted():

tests if the thread has been interrupted and returns the interrupted flag either true or false.

Boolean Interrupted():

The static interrupted() method tests if the thread has been interrupted. This method returns the interrupted flag after that it sets the flag to false if it is true.

Check out the below sample code:

package com.techbeamers.multithreading;

public class TestThreadInterrupt extends Thread {

public void run() {

for (int i = 1; i <= 2; i++) {

if (Thread.interrupted()) {

System.out.println("code for interrupted thread");

} else {

System.out.println("code for normal thread");

}

} //end of for loop

}

public static void main(String args[]) {

TestThreadInterrupt t1 = new TestThreadInterrupt();

TestThreadInterrupt t2 = new TestThreadInterrupt();

t1.start();

t1.interrupt();

t2.start();

}

}

Output:

code for interrupted thread

code for normal thread

code for normal thread

code for normal thread

Suspend():

You can use it to suspend the thread. *[deprecated]*

Resume():

You can use it to resume the suspended thread. *[deprecated]*

Stop():

You can use it to halt the thread. *[deprecated]*

*Note: Sun has deprecated a variety of Thread methods, such as suspend(), resume() and stop() because they can lock up your programs or damage objects. As a result, you should not call them in your code.*

We’ve covered almost all the important areas of the Java Thread class in the above section, hope it would help. In the next segment, you’ll see the two methods to create threads in Java.

Java Thread Creation Methods And Examples

**Create Thread By Extending The Thread Class**

In this case, you need to complete the following steps to spawn a thread in a Java program.

* Add a new class that extends the Thread class.
* This newly created class should override the Run() method which is the entry point for the new thread.
* Invokes the start() method to initiate the execution of the thread.

Check out the below sample code:

package com.techbeamers.multithreading;

class MultithreadDemo extends Thread{

public void run(){

System.out.println("My thread is in running state.");

}

public static void main(String args[]){

MultithreadDemo obj=new MultithreadDemo();

obj.start();

}

}

Output:

My thread is in running state

**Create A Thread Using The Runnable Interface**

For this approach, you need to follow the below steps to create a thread.

* Create a class that does the following.
  + Implements the Runnable interface
  + Provides the implementation of the run() method
* The “run()” function is an entry point for the thread, and it remains alive until the “run()” function finishes its execution.
* Once you create the thread, bring it to the Running state by calling the start() method.
  + **Note:** The start() method calls the run() method implicitly.

Check out the below sample code:

package com.techbeamers.multithreading;

class MultithreadDemo implements Runnable{

public void run(){

System.out.println("My thread is in running state.");

}

public static void main(String args[]){

MultithreadDemo obj=new MultithreadDemo();

Thread threadobj = new Thread(obj);

threadobj.start();

}

}

Output:

My thread is in running state

Multithreading Quiz – Must Attempt

Here is a Java multithreading quiz empowered with 20 interesting and challenging questions for programmers. Have a look:

[**Multithreading Quiz – Java**](https://www.techbeamers.com/java-multithreading-quiz-with-20-interview-questions/)

**Download Java Multithreading Code Samples**

Now it’s time to download the sample projects so that you can easily understand the Java multithreading code snippets specified in this post. Please use the below link to start your download.

[**⇓ Download Multithreading Project**](https://www.techbeamers.com/wp-content/uploads/2016/02/JavaMultithreading.zip)

**Summary – Java Multithreading**

We wish that the above Java multithreading tutorial would have helped you in walking a step further on the Java learning scale. In the next article of Java programming, we’ll give insights on different ways to implement synchronization in Java applications.