Multithreading in Java

**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.

### **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:

* Process-based Multitasking (Multiprocessing)
* 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](https://www.javatpoint.com/register-memory), 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.

#### **Note: At least one process is required for each thread.**

## **What is Thread in java**

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](https://www.javatpoint.com/os-tutorial), and one process can have multiple threads.

#### **Note: At a time only one thread is executed.**

## **Java Thread class**

Java provides **Thread class** to achieve thread programming. Thread class provides [constructors](https://www.javatpoint.com/java-constructor) and methods to create and perform operations on a thread. Thread class extends [Object class](https://www.javatpoint.com/object-class) and implements Runnable interface.

## **Java Thread Methods**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.N.** | **Modifier and Type** | **Method** | **Description** |
| 1) | void | [start()](https://www.javatpoint.com/java-thread-start-method) | It is used to start the execution of the thread. |
| 2) | void | [run()](https://www.javatpoint.com/java-thread-run-method) | It is used to do an action for a thread. |
| 3) | static void | [sleep()](https://www.javatpoint.com/java-thread-sleep-method) | It sleeps a thread for the specified amount of time. |
| 4) | static Thread | [currentThread()](https://www.javatpoint.com/java-thread-currentthread-method) | It returns a reference to the currently executing thread object. |
| 5) | void | [join()](https://www.javatpoint.com/java-thread-join-method) | It waits for a thread to die. |
| 6) | int | [getPriority()](https://www.javatpoint.com/java-thread-getpriority-method) | It returns the priority of the thread. |
| 7) | void | [setPriority()](https://www.javatpoint.com/java-thread-setpriority-method) | It changes the priority of the thread. |
| 8) | String | [getName()](https://www.javatpoint.com/java-thread-getname-method) | It returns the name of the thread. |
| 9) | void | [setName()](https://www.javatpoint.com/java-thread-setname-method) | It changes the name of the thread. |
| 10) | long | [getId()](https://www.javatpoint.com/java-thread-getid-method) | It returns the id of the thread. |
| 11) | boolean | [isAlive()](https://www.javatpoint.com/java-thread-isalive-method) | It tests if the thread is alive. |
| 12) | static void | [yield()](https://www.javatpoint.com/java-thread-yield-method) | It causes the currently executing thread object to pause and allow other threads to execute temporarily. |
| 13) | void | [suspend()](https://www.javatpoint.com/java-thread-suspend-method) | It is used to suspend the thread. |
| 14) | void | [resume()](https://www.javatpoint.com/java-thread-resume-method) | It is used to resume the suspended thread. |
| 15) | void | [stop()](https://www.javatpoint.com/java-thread-stop-method) | It is used to stop the thread. |
| 16) | void | [destroy()](https://www.javatpoint.com/java-thread-destroy-method) | It is used to destroy the thread group and all of its subgroups. |
| 17) | boolean | [isDaemon()](https://www.javatpoint.com/java-thread-isdaemon-method) | It tests if the thread is a daemon thread. |
| 18) | void | [setDaemon()](https://www.javatpoint.com/java-thread-setdaemon-method) | It marks the thread as daemon or user thread. |
| 19) | void | [interrupt()](https://www.javatpoint.com/java-thread-interrupt-method) | It interrupts the thread. |
| 20) | boolean | [isinterrupted()](https://www.javatpoint.com/java-thread-isinterrupted-method) | It tests whether the thread has been interrupted. |
| 21) | static boolean | [interrupted()](https://www.javatpoint.com/java-thread-interrupted-method) | It tests whether the current thread has been interrupted. |
| 22) | static int | [activeCount()](https://www.javatpoint.com/java-thread-activecount-method) | It returns the number of active threads in the current thread's thread group. |
| 23) | void | [checkAccess()](https://www.javatpoint.com/java-thread-checkaccess-method) | It determines if the currently running thread has permission to modify the thread. |
| 24) | static boolean | [holdLock()](https://www.javatpoint.com/java-thread-holdlock-method) | It returns true if and only if the current thread holds the monitor lock on the specified object. |
| 25) | static void | [dumpStack()](https://www.javatpoint.com/java-thread-dumpstack-method) | It is used to print a stack trace of the current thread to the standard error stream. |
| 26) | StackTraceElement[] | [getStackTrace()](https://www.javatpoint.com/java-thread-getstacktrace-method) | It returns an array of stack trace elements representing the stack dump of the thread. |
| 27) | static int | [enumerate()](https://www.javatpoint.com/java-thread-enumerate-method) | It is used to copy every active thread's thread group and its subgroup into the specified array. |
| 28) | Thread.State | [getState()](https://www.javatpoint.com/java-thread-getstate-method) | It is used to return the state of the thread. |
| 29) | ThreadGroup | [getThreadGroup()](https://www.javatpoint.com/java-thread-getthreadgroup-method) | It is used to return the thread group to which this thread belongs |
| 30) | String | [toString()](https://www.javatpoint.com/java-thread-tostring-method) | It is used to return a string representation of this thread, including the thread's name, priority, and thread group. |
| 31) | void | [notify()](https://www.javatpoint.com/java-thread-notify-method) | It is used to give the notification for only one thread which is waiting for a particular object. |
| 32) | void | [notifyAll()](https://www.javatpoint.com/java-thread-notifyall-method) | It is used to give the notification to all waiting threads of a particular object. |
| 33) | void | [setContextClassLoader()](https://www.javatpoint.com/java-thread-setcontextclassloader-method) | It sets the context ClassLoader for the Thread. |
| 34) | ClassLoader | [getContextClassLoader()](https://www.javatpoint.com/java-thread-getcontextclassloader-method) | It returns the context ClassLoader for the thread. |
| 35) | static Thread.UncaughtExceptionHandler | [getDefaultUncaughtExceptionHandler()](https://www.javatpoint.com/java-thread-getdefaultuncaughtexceptionhandler-method) | It returns the default handler invoked when a thread abruptly terminates due to an uncaught exception. |
| 36) | static void | [setDefaultUncaughtExceptionHandler()](https://www.javatpoint.com/java-thread-setdefaultuncaughtexceptionhandler-method) | It sets the default handler invoked when a thread abruptly terminates due to an uncaught exception. |

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:

1. New
2. Runnable
3. Running
4. Non-Runnable (Blocked)
5. Terminated



|  |
| --- |
| **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.

|  |  |  |  |
| --- | --- | --- | --- |
| How to create thread There are two ways to create a thread:   1. By extending Thread class 2. 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:**  |  | | --- | | 1. **public void run():**is used to perform action for a thread. 2. **public void start():**starts the execution of the thread. JVM calls the run() method on the thread. 3. **public void sleep(long miliseconds):**Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds. 4. **public void join():**waits for a thread to die. 5. **public void join(long miliseconds):**waits for a thread to die for the specified miliseconds. 6. **public int getPriority():**returns the priority of the thread. 7. **public int setPriority(int priority):**changes the priority of the thread. 8. **public String getName():**returns the name of the thread. 9. **public void setName(String name):**changes the name of the thread. 10. **public Thread currentThread():**returns the reference of currently executing thread. 11. **public int getId():**returns the id of the thread. 12. **public Thread.State getState():**returns the state of the thread. 13. **public boolean isAlive():**tests if the thread is alive. 14. **public void yield():**causes the currently executing thread object to temporarily pause and allow other threads to execute. 15. **public void suspend():**is used to suspend the thread(depricated). 16. **public void resume():**is used to resume the suspended thread(depricated). 17. **public void stop():**is used to stop the thread(depricated). 18. **public boolean isDaemon():**tests if the thread is a daemon thread. 19. **public void setDaemon(boolean b):**marks the thread as daemon or user thread. 20. **public void interrupt():**interrupts the thread. 21. **public boolean isInterrupted():**tests if the thread has been interrupted. 22. **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(). |

|  |
| --- |
| 1. **public 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**

1. **class** Multi **extends** Thread{
2. **public** **void** run(){
3. System.out.println("thread is running...");
4. }
5. **public** **static** **void** main(String args[]){
6. Multi t1=**new** Multi();
7. t1.start();
8. }
9. }

Output:thread is running...

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

1. **class** Multi3 **implements** Runnable{
2. **public** **void** run(){
3. System.out.println("thread is running...");
4. }
6. **public** **static** **void** main(String args[]){
7. Multi3 m1=**new** Multi3();
8. Thread t1 =**new** Thread(m1);
9. t1.start();
10. }
11. }

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 explicitly create Thread class object. We are passing the object of your class that implements Runnable so that your class run() method may execute. |