**Multithreading in Java (Concurrency 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.

## **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.

## **Java Thread class**

A thread is of two types: a **user thread** or a **daemon thread**.

* 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. |

## **Thread States/Lifecycle of a thread**

A thread has different states and traverses through each state in different stages as described below:

* **New**: When we create a new thread, it is in a new state before we call the start() method.
* **Runnable:** A thread takes this state when it calls the start() method and waits for the scheduler to pick up the thread.
* **Running:**When the thread is in execution, it is in a running state.
* **Waiting:**Thread is in waiting state when it has to wait for other threads to complete execution as part of the synchronization process.
* **Dead:**When the thread is terminated, it is in the dead state.



## **Creating a thread**

We can create a thread in two different ways. Both the methods override the run() method.

* Implementing Runnable interface
* Extending Thread class

## **Runnable interface**

To create a runnable thread, we can implement the Runnable interface and override the public run() method. To access this method, we need to create a Thread object and invoke the start() method. We normally use the Runnable interface when we want to implement more functionality in a class.

## **Thread class**

Another way of creating a runnable thread is to extend the Thread class and implement the public run() method. Using the Thread object, we can invoke the start() method to execute the run() method and run the thread. We use the Thread class when we want to implement only the Thread functionality.

### **Commonly used Constructors of Thread class:**

* Thread()
* Thread(String name)
* Thread(Runnable r)
* Thread(Runnable r,String name)

## **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...

**Java Thread Example by implementing Runnable interface**

**class** Multi **implements** Runnable

{

**public** **void** run()

{

System.out.println("thread is running...");

}

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

{

Multi m1=**new** Multi();

Thread t1 =**new** Thread(m1);   // Using the constructor Thread(Runnable r)

t1.start();

 }

}

**Output:**

thread is running...

**Using the Thread Class: Thread(Runnable r, String name)**

**public** **class** MyThread2 **implements** Runnable

{

**public** **void** run()

{

System.out.println("Now the thread is running ...");

}

// main method

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

{

// creating an object of the class MyThread2

Runnable r1 = **new** MyThread2();

// creating an object of the class Thread using Thread(Runnable r, String name)

Thread th1 = **new** Thread(r1, "My new thread");

// the start() method moves the thread to the active state

th1.start();

// getting the thread name by invoking the getName() method

String str = th1.getName();

System.out.println(str);

}

}

**Output:**

My new thread

Now the thread is running ...

* **Thread Scheduler in Java**

A component of Java that decides which thread to run or execute and which thread to wait is called a **thread scheduler in Java**. In Java, a thread is only chosen by a thread scheduler if it is in the runnable state. However, if there is more than one thread in the runnable state, it is up to the thread scheduler to pick one of the threads and ignore the other ones. There are some criteria that decide which thread will execute first. There are two factors for scheduling a thread i.e. **Priority** and **Time of arrival**.

**Priority:** Priority of each thread lies between 1 to 10. If a thread has a higher priority, it means that thread has got a better chance of getting picked up by the thread scheduler.

**Time of Arrival:** Suppose two threads of the same priority enter the runnable state, then priority cannot be the factor to pick a thread from these two threads. In such a case, **arrival time** of thread is considered by the thread scheduler. A thread that arrived first gets the preference over the other threads.

# Priority of a Thread (Thread Priority)

Each thread has a priority. Priorities are represented by a number between 1 and 10. In most cases, the thread scheduler 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. Note that not only JVM a Java programmer can also assign the priorities of a thread explicitly in a Java program.

# Setter & Getter Method of Thread Priority

Let's discuss the setter and getter method of the thread priority.

**public final int getPriority():** The java.lang.Thread.getPriority() method returns the priority of the given thread.

**public final void setPriority(int newPriority):** The java.lang.Thread.setPriority() method updates or assign the priority of the thread to newPriority. The method throws IllegalArgumentException if the value newPriority goes out of the range, which is 1 (minimum) to 10 (maximum).

# 3 constants defined in Thread class:

1. public static int MIN\_PRIORITY
2. public static int NORM\_PRIORITY
3. 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.

# Java Thread join() method

The **join()** method of thread class waits for a thread to die. It is used when you want one thread to wait for completion of another.

## **Syntax**

1. **public** **final** **void** join()**throws** InterruptedException
2. **public** **void** join(**long** millis)throwsInterruptedException
3. **public** **final** **void** join(**long** millis, **int** nanos)**throws** InterruptedException

## **Parameter**

1. millis: It defines the time to wait in milliseconds
2. nanos: 0-999999 additional nanoseconds to wait

## **Return**

It does not **return** any value.

**Example:**

**public** **class** JoinExample1 **extends** Thread

{

**public** **void** run()

    {

**for**(**int** i=1; i<=4; i++)

        {

**try**

            {

                Thread.sleep(500);

            }**catch**(Exception e){System.out.println(e);}

            System.out.println(i);

        }

     }

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

    {

        // creating three threads

        JoinExample1 t1 = **new** JoinExample1();

        JoinExample1 t2 = **new** JoinExample1();

        JoinExample1 t3 = **new** JoinExample1();

        // thread t1 starts

        t1.start();

        // starts second thread when first thread t1 is died.

**try**

        {

            t1.join();

        }**catch**(Exception e){System.out.println(e);}

        // start t2 and t3 thread

        t2.start();

        t3.start();

    }

}

**Output:**

1

2

3

4

1

1

2

2

3

3

4

4

# Write a program, which can run two-thread simultaneously. One thread will calculate the factorial of a number and another thread will calculate the factors of that number. Make sure main thread finishes last.

**class** Factor **implements** Runnable

{

    Thread t;

**int** n;

    Factor(**int** x)

    {

        n = x;

        t=**new** Thread(**this**,"Factor Thread");

        System.out.println("\nFactor thread created : "+t);

        t.start();

    }

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

    {

**try**

        {

            System.out.print("Factors of "+n+" are : ");

**for**(**int** i = 1;i<=n;i++)

            {

**if**(n%i == 0 )

                    System.out.print(i+" ");

            }

        }

**catch** (Exception e)

        {

            System.out.println(e);

        }

        System.out.println("\nExiting "+t);

    }

}

**class** Factorial **implements** Runnable

{

    Thread t;

**int** n;

    Factorial(**int** x)

    {

        n = x;

        t=**new** Thread(**this**,"Factorial Thread");

        System.out.println("\nFactorial thread created : "+t);

        t.start();

    }

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

    {

**try**

        {

**long** f = 1;

**for**(**int** i = 2;i<=n;i++)

            {

                f\*=i;

            }

            System.out.print("Factorial of "+n+" is : "+f);

        }

**catch** (Exception e)

        {

            System.out.println(e);

        }

        System.out.println("\nExiting "+t);

    }

}

**class** Main

{

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

    {

        Factor obj = **new** Factor(12);

        Factorial obj2 = **new** Factorial(5);

**try**

        {

            obj2.t.join();

            obj.t.join();

        }

**catch** (InterruptedException e)

        {

            System.out.println(e);

        }

        System.out.println("Exiting main thread");

    }

}

***Sample Output:***

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | Factor thread created : Thread[Factor Thread,5,main]  Factors of 12 are : 1 2 3 4 6 12  Factorial thrsead created : Thread[Factorial Thread,5,main]  Exiting Thread[Factor Thread,5,main]  Factorial of 5 is : 120  Exiting Thread[Factorial Thread,5,main]  Exiting main thread |