Top of Form

#### ****Multithreading in Java****

Multithreading is a Java feature that allows concurrent execution of two or more parts of a program for maximum utilization of CPU. Each part of such a program is called a thread. So, threads are light-weight processes within a process.

or

**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.

**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.

#### ****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.

#### ****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) NewThe 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.

Bottom of Form

Top of Form

#### ****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:****

**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 void join():** waits for a thread to die.

**public void join(long miliseconds):** waits for a thread to die for the specified milliseconds.

**public int getPriority():** returns the priority of the thread.

**public int setPriority(int priority):** changes the priority of the thread.

**public String getName():** returns the name of the thread.

**public void setName(String name):**changes the name of the thread.

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

**public int getId():** returns the id of the thread.

**public Thread.State getState():** returns the state of the thread.

**public boolean isAlive():** tests if the thread is alive.

**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 setDaemon(boolean b):** marks the thread as daemon or user 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.

#### ****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...

#### ****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().

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.

#### ****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 a Thread class object. We are passing the object of your class that implements Runnable so that your class run() method may execute.

**Thread Class vs Runnable Interface**

1. If we extend the Thread class, our class cannot extend any other class because Java doesn’t support multiple inheritance. But, if we implement the Runnable interface, our class can still extend other base classes.

2. We can achieve basic functionality of a thread by extending Thread class because it provides some inbuilt methods like yield(), interrupt() etc. that are not available in Runnable interface.

#### ****Sleep method in java****

The sleep() method of Thread class is used to sleep a thread for the specified amount of time.

#### ****Syntax of sleep() method in java****

The Thread class provides two methods for sleeping a thread:

* public static void sleep(long miliseconds)throws InterruptedException
* public static void sleep(long miliseconds, int nanos)throws InterruptedException

#### ****Example of sleep method in java****

1. **class** TestSleepMethod1 **extends** Thread{
2. **public** **void** run(){
3. **for**(**int** i=1;i<5;i++){
4. **try**{Thread.sleep(500);}**catch**(InterruptedException e){System.out.println(e);}
5. System.out.println(i);
6. }
7. }
8. **public** **static** **void** main(String args[]){
9. TestSleepMethod1 t1=**new** TestSleepMethod1();
10. TestSleepMethod1 t2=**new** TestSleepMethod1();
12. t1.start();
13. t2.start();
14. }
15. }

**Output:**

1

1

2

2

3

3

4

4

As you know well that at a time only one thread is executed. If you sleep a thread for the specified time, the thread scheduler picks up another thread, and so on.

#### ****Can we start a thread twice****

No. After starting a thread, it can never be started again. If you does so, an IllegalThreadStateException is thrown. In such case, thread will run once but for second time, it will throw exception.

Let's understand it by the example given below:

1. **public** **class** TestThreadTwice1 **extends** Thread{
2. **public** **void** run(){
3. System.out.println("running...");
4. }
5. **public** **static** **void** main(String args[]){
6. TestThreadTwice1 t1=**new** TestThreadTwice1();
7. t1.start();
8. t1.start();
9. }
10. }

running

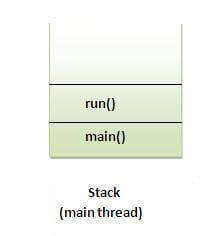
Exception in thread "main" java.lang.IllegalThreadStateException

#### ****What if we call run() method directly instead start() method?****

* Each thread starts in a separate call stack.
* Invoking the run() method from main thread, the run() method goes onto the current call stack rather than at the beginning of a new call stack.

1. **class** TestCallRun1 **extends** Thread{
2. **public** **void** run(){
3. System.out.println("running...");
4. }
5. **public** **static** **void** main(String args[]){
6. TestCallRun1 t1=**new** TestCallRun1();
7. t1.run();//fine, but does not start a separate call stack
8. }
9. }

**Output:** running...



**Problem if you direct call run() method**

1. **class** TestCallRun2 **extends** Thread{
2. **public** **void** run(){
3. **for**(**int** i=1;i<5;i++){
4. **try**{Thread.sleep(500);}**catch**(InterruptedException e){System.out.println(e);}
5. System.out.println(i);
6. }
7. }
8. **public** **static** **void** main(String args[]){
9. TestCallRun2 t1=**new** TestCallRun2();
10. TestCallRun2 t2=**new** TestCallRun2();
12. t1.run();
13. t2.run();
14. }
15. }

**Output:**

1

2

3

4

5

1

2

3

4

5

As you can see in the above program that there is no context-switching because here t1 and t2 will be treated as normal object not thread object.

#### ****The join() method****

The join() method waits for a thread to die. In other words, it causes the currently running threads to stop executing until the thread it joins with completes its task.

#### ****Syntax:****

public void join()throws InterruptedException

public void join(long milliseconds)throws InterruptedException

**Example of join() method**

1. **class** TestJoinMethod1 **extends** Thread{
2. **public** **void** run(){
3. **for**(**int** i=1;i<=5;i++){
4. **try**{
5. Thread.sleep(500);
6. }**catch**(Exception e){System.out.println(e);}
7. System.out.println(i);
8. }
9. }
10. **public** **static** **void** main(String args[]){
11. TestJoinMethod1 t1=**new** TestJoinMethod1();
12. TestJoinMethod1 t2=**new** TestJoinMethod1();
13. TestJoinMethod1 t3=**new** TestJoinMethod1();
14. t1.start();
15. **try**{
16. t1.join();
17. }**catch**(Exception e){System.out.println(e);}
19. t2.start();
20. t3.start();
21. }
22. }

**Output:**

1

2

3

4

5

1

1

2

2

3

3

4

4

5

5

As you can see in the above example, when t1 completes its task then t2 and t3 starts executing.

**Example of join(long miliseconds) method**

1. **class** TestJoinMethod2 **extends** Thread{
2. **public** **void** run(){
3. **for**(**int** i=1;i<=5;i++){
4. **try**{
5. Thread.sleep(500);
6. }**catch**(Exception e){System.out.println(e);}
7. System.out.println(i);
8. }
9. }
10. **public** **static** **void** main(String args[]){
11. TestJoinMethod2 t1=**new** TestJoinMethod2();
12. TestJoinMethod2 t2=**new** TestJoinMethod2();
13. TestJoinMethod2 t3=**new** TestJoinMethod2();
14. t1.start();
15. **try**{
16. t1.join(1500);
17. }**catch**(Exception e){System.out.println(e);}
19. t2.start();
20. t3.start();
21. }
22. }

**Output:**

1

2

3

1

4

1

2

5

2

3

3

4

4

5

5

In the above example, when t1 is completes its task for 1500 milliseconds (3 times) then t2 and t3 starts executing.

#### ****getName(),setName(String) and getId() method:****

public String getName()

public void setName(String name)

public long getId()

1. **class** TestJoinMethod3 **extends** Thread{
2. **public** **void** run(){
3. System.out.println("running...");
4. }
5. **public** **static** **void** main(String args[]){
6. TestJoinMethod3 t1=**new** TestJoinMethod3();
7. TestJoinMethod3 t2=**new** TestJoinMethod3();
8. System.out.println("Name of t1:"+t1.getName());
9. System.out.println("Name of t2:"+t2.getName());
10. System.out.println("id of t1:"+t1.getId());
12. t1.start();
13. t2.start();
15. t1.setName("CSE");
16. System.out.println("After changing name of t1: "+t1.getName());
17. }
18. }

**Output:**

Name of t1:Thread-0

Name of t2:Thread-1

id of t1:8

running...

After changing name of t1: CSE

running...

Bottom of Form

Top of Form

<https://youtu.be/JceAHRlQsqc>

<https://youtu.be/5kTh-dMnBWo>

**Reference Links:**

<https://www.geeksforgeeks.org/lifecycle-and-states-of-a-thread-in-java/>

<https://www.journaldev.com/1044/thread-life-cycle-in-java-thread-states-in-java>

<https://www.tutorialspoint.com/java/java_multithreading.htm>

<https://www.javatpoint.com/multithreading-in-java>

<https://www.journaldev.com/1079/multithreading-in-java>

<https://www.javatpoint.com/join()-method>

Bottom of Form