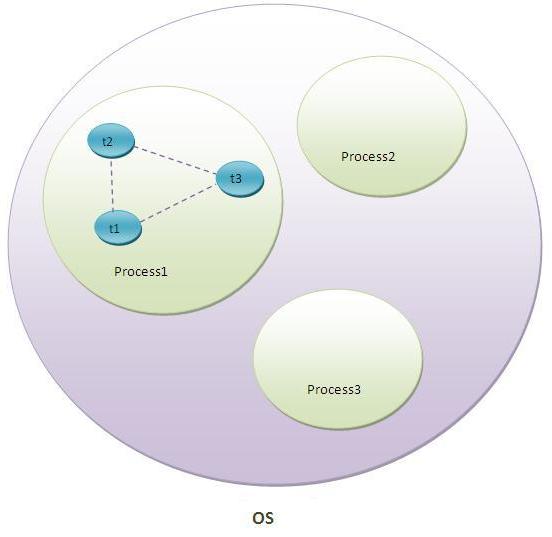
## What is Thread in java

A thread is a lightweight sub process, a 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 shares a common memory area.



As shown in the above figure, thread is executed inside the process. There is context-switching between the threads. There can be multiple processes inside the OS and one process can have multiple threads.

|  |
| --- |
| Life cycle of a Thread (Thread States)   1. [Life cycle of a thread](https://www.javatpoint.com/life-cycle-of-a-thread)    1. [New](https://www.javatpoint.com/life-cycle-of-a-thread#threadstatenew)    2. [Runnable](https://www.javatpoint.com/life-cycle-of-a-thread#threadstaterunnable)    3. [Running](https://www.javatpoint.com/life-cycle-of-a-thread#threadstaterunning)    4. [Non-Runnable (Blocked)](https://www.javatpoint.com/life-cycle-of-a-thread#threadstateblocked)    5. [Terminated](https://www.javatpoint.com/life-cycle-of-a-thread#threadstateterminated)   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   thread life cycle in java  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.

# Multithreading in Java

1. [Multithreading](https://www.javatpoint.com/multithreading-in-java)
2. [Multitasking](https://www.javatpoint.com/multithreading-in-java#multitasing)
3. [Process-based multitasking](https://www.javatpoint.com/multithreading-in-java#multiprocessing)
4. [Thread-based multitasking](https://www.javatpoint.com/multithreading-in-java#multithreading)
5. [What is Thread](https://www.javatpoint.com/multithreading-in-java#thread)

**Multithreading in java** is a process of executing multiple threads simultaneously.

Thread is basically a lightweight sub-process, a smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.

But we use multithreading than multiprocessing because threads share a common 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 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 exception occur 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 by two ways:

* Process-based Multitasking(Multiprocessing)
* Thread-based Multitasking(Multithreading)

### 1) Process-based Multitasking (Multiprocessing)

* Each process have its own address in memory i.e. each process allocates separate memory area.
* Process is heavyweight.
* Cost of communication between the process is high.
* Switching from one process to another require some time for saving and loading registers, memory maps, updating lists etc.

### 2) Thread-based Multitasking (Multithreading)

* Threads share the same address space.
* Thread is lightweight.
* Cost of communication between the thread is low.

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

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

# Priority of a Thread (Thread Priority):

|  |
| --- |
| Each thread have 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:

|  |
| --- |
| 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. |

### Example of priority of a Thread:

1. **class** TestMultiPriority1 **extends** Thread{
2. **public** **void** run(){
3. System.out.println("running thread name is:"+Thread.currentThread().getName());
4. System.out.println("running thread priority is:"+Thread.currentThread().getPriority());
6. }
7. **public** **static** **void** main(String args[]){
8. TestMultiPriority1 m1=**new** TestMultiPriority1();
9. TestMultiPriority1 m2=**new** TestMultiPriority1();
10. m1.setPriority(Thread.MIN\_PRIORITY);
11. m2.setPriority(Thread.MAX\_PRIORITY);
12. m1.start();
13. m2.start();
15. }
16. }

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=TestMultiPriority1)

Output:running thread name is:Thread-0

running thread priority is:10

running thread name is:Thread-1

running thread priority is:1

Synchronization in Java

Synchronization in java is the capability *to control the access of multiple threads to any shared resource*.

Java Synchronization is better option where we want to allow only one thread to access the shared resource.

Why use Synchronization

The synchronization is mainly used to

1. To prevent thread interference.
2. To prevent consistency problem.

Types of Synchronization

There are two types of synchronization

1. Process Synchronization
2. Thread Synchronization

Here, we will discuss only thread synchronization.

Thread Synchronization

There are two types of thread synchronization mutual exclusive and inter-thread communication.

1. Mutual Exclusive
   1. Synchronized method.
   2. Synchronized block.
   3. static synchronization.
2. Cooperation (Inter-thread communication in java)

Mutual Exclusive

Mutual Exclusive helps keep threads from interfering with one another while sharing data. This can be done by three ways in java:

1. by synchronized method
2. by synchronized block
3. by static synchronization

Concept of Lock in Java

Synchronization is built around an internal entity known as the lock or monitor. Every object has an lock associated with it. By convention, a thread that needs consistent access to an object's fields has to acquire the object's lock before accessing them, and then release the lock when it's done with them.

From Java 5 the package java.util.concurrent.locks contains several lock implementations.

Understanding the problem without Synchronization

In this example, there is no synchronization, so output is inconsistent. Let's see the example:

1. Class Table{
3. **void** printTable(**int** n){//method not synchronized
4. **for**(**int** i=1;i<=5;i++){
5. System.out.println(n\*i);
6. **try**{
7. Thread.sleep(400);
8. }**catch**(Exception e){System.out.println(e);}
9. }
11. }
12. }
14. **class** MyThread1 **extends** Thread{
15. Table t;
16. MyThread1(Table t){
17. **this**.t=t;
18. }
19. **public** **void** run(){
20. t.printTable(5);
21. }
23. }
24. **class** MyThread2 **extends** Thread{
25. Table t;
26. MyThread2(Table t){
27. **this**.t=t;
28. }
29. **public** **void** run(){
30. t.printTable(100);
31. }
32. }
34. **class** TestSynchronization1{
35. **public** **static** **void** main(String args[]){
36. Table obj = **new** Table();//only one object
37. MyThread1 t1=**new** MyThread1(obj);
38. MyThread2 t2=**new** MyThread2(obj);
39. t1.start();
40. t2.start();
41. }
42. }

Output: 5

100

10

200

15

300

20

400

25

500

Java synchronized method

If you declare any method as synchronized, it is known as synchronized method.

Synchronized method is used to lock an object for any shared resource.

When a thread invokes a synchronized method, it automatically acquires the lock for that object and releases it when the thread completes its task.

1. //example of java synchronized method
2. **class** Table{
3. **synchronized** **void** printTable(**int** n){//synchronized method
4. **for**(**int** i=1;i<=5;i++){
5. System.out.println(n\*i);
6. **try**{
7. Thread.sleep(400);
8. }**catch**(Exception e){System.out.println(e);}
9. }
11. }
12. }
14. **class** MyThread1 **extends** Thread{
15. Table t;
16. MyThread1(Table t){
17. **this**.t=t;
18. }
19. **public** **void** run(){
20. t.printTable(5);
21. }
23. }
24. **class** MyThread2 **extends** Thread{
25. Table t;
26. MyThread2(Table t){
27. **this**.t=t;
28. }
29. **public** **void** run(){
30. t.printTable(100);
31. }
32. }

1. **public** **class** TestSynchronization2{
2. **public** **static** **void** main(String args[]){
3. Table obj = **new** Table();//only one object
4. MyThread1 t1=**new** MyThread1(obj);
5. MyThread2 t2=**new** MyThread2(obj);
6. t1.start();
7. t2.start();
8. }
9. }

Output: 5

10

15

20

25

100

200

300

400

500

Example of synchronized method by using annonymous class

In this program, we have created the two threads by annonymous class, so less coding is required.

1. //Program of synchronized method by using annonymous class
2. **class** Table{
3. **synchronized** **void** printTable(**int** n){//synchronized method
4. **for**(**int** i=1;i<=5;i++){
5. System.out.println(n\*i);
6. **try**{
7. Thread.sleep(400);
8. }**catch**(Exception e){System.out.println(e);}
9. }
11. }
12. }
14. **public** **class** TestSynchronization3{
15. **public** **static** **void** main(String args[]){
16. **final** Table obj = **new** Table();//only one object
18. Thread t1=**new** Thread(){
19. **public** **void** run(){
20. obj.printTable(5);
21. }
22. };
23. Thread t2=**new** Thread(){
24. **public** **void** run(){
25. obj.printTable(100);
26. }
27. };
29. t1.start();
30. t2.start();
31. }
32. }

Output: 5

10

15

20

25

100

200

300

400

500

Inter-thread communication in Java

**Inter-thread communication** or **Co-operation** is all about allowing synchronized threads to communicate with each other.

Cooperation (Inter-thread communication) is a mechanism in which a thread is paused running in its critical section and another thread is allowed to enter (or lock) in the same critical section to be executed.It is implemented by following methods of **Object class**:

* wait()
* notify()
* notifyAll()

1) wait() method

Causes current thread to release the lock and wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed.

The current thread must own this object's monitor, so it must be called from the synchronized method only otherwise it will throw exception.

|  |  |
| --- | --- |
| **Method** | **Description** |
| public final void wait()throws InterruptedException | waits until object is notified. |
| public final void wait(long timeout)throws InterruptedException | waits for the specified amount of time. |

2) notify() method

Wakes up a single thread that is waiting on this object's monitor. If any threads are waiting on this object, one of them is chosen to be awakened. The choice is arbitrary and occurs at the discretion of the implementation. Syntax:

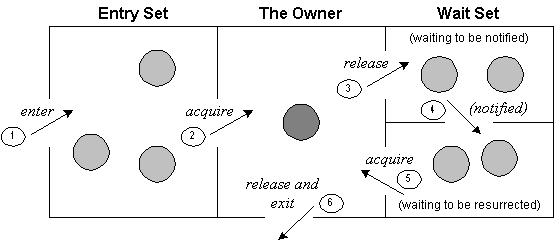
public final void notify()

3) notifyAll() method

Wakes up all threads that are waiting on this object's monitor. Syntax:

public final void notifyAll()

Understanding the process of inter-thread communication



The point to point explanation of the above diagram is as follows:

1. Threads enter to acquire lock.
2. Lock is acquired by on thread.
3. Now thread goes to waiting state if you call wait() method on the object. Otherwise it releases the lock and exits.
4. If you call notify() or notifyAll() method, thread moves to the notified state (runnable state).
5. Now thread is available to acquire lock.
6. After completion of the task, thread releases the lock and exits the monitor state of the object.

Why wait(), notify() and notifyAll() methods are defined in Object class not Thread class?

It is because they are related to lock and object has a lock.

Difference between wait and sleep?

Let's see the important differences between wait and sleep methods.

|  |  |
| --- | --- |
| **wait()** | **sleep()** |
| wait() method releases the lock | sleep() method doesn't release the lock. |
| is the method of Object class | is the method of Thread class |
| is the non-static method | is the static method |
|  |  |
| should be notified by notify() or notifyAll() methods | after the specified amount of time, sleep is completed. |

Example of inter thread communication in java

Let's see the simple example of inter thread communication.

**class** Customer{

**int** amount=10000;

**synchronized** **void** withdraw(**int** amount){

System.out.println("going to withdraw...");

**if**(**this**.amount<amount){

System.out.println("Less balance; waiting for deposit...");

**try**{wait();}**catch**(Exception e){}

}

**this**.amount-=amount;

System.out.println("withdraw completed...");

}

**synchronized** **void** deposit(**int** amount){

System.out.println("going to deposit...");

**this**.amount+=amount;

System.out.println("deposit completed... ");

notify();

}

}

**class** Test{

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

**final** Customer c=**new** Customer();

**new** Thread(){

**public** **void** run(){c.withdraw(15000);}

}.start();

**new** Thread(){

**public** **void** run(){c.deposit(10000);}

}.start();

}}

Output: going to withdraw...

Less balance; waiting for deposit...

going to deposit...

deposit completed...

withdraw completed

Java - Thread Control

Advertisements

[Previous Page](https://www.tutorialspoint.com/java/java_multithreading.htm)

[Next Page](https://www.tutorialspoint.com/java/java_multithreading.htm)

Core Java provides complete control over multithreaded program. You can develop a multithreaded program which can be suspended, resumed, or stopped completely based on your requirements. There are various static methods which you can use on thread objects to control their behavior. Following table lists down those methods –

|  |  |
| --- | --- |
| **Sr.No.** | **Method & Description** |
| 1 | **public void suspend()**  This method puts a thread in the suspended state and can be resumed using resume() method. |
| 2 | **public void stop()**  This method stops a thread completely. |
| 3 | **public void resume()**  This method resumes a thread, which was suspended using suspend() method. |
| 4 | **public void wait()**  Causes the current thread to wait until another thread invokes the notify(). |
| 5 | **public void notify()**  Wakes up a single thread that is waiting on this object's monitor. |

Be aware that the latest versions of Java has deprecated the usage of suspend( ), resume( ), and stop( ) methods and so you need to use available alternatives.

Example

class RunnableDemo implements Runnable {

public Thread t;

private String threadName;

boolean suspended = false;

RunnableDemo( String name) {

threadName = name;

System.out.println("Creating " + threadName );

}

public void run() {

System.out.println("Running " + threadName );

try {

for(int i = 10; i > 0; i--) {

System.out.println("Thread: " + threadName + ", " + i);

// Let the thread sleep for a while.

Thread.sleep(300);

synchronized(this) {

while(suspended) {

wait();

}

}

}

}catch (InterruptedException e) {

System.out.println("Thread " + threadName + " interrupted.");

}

System.out.println("Thread " + threadName + " exiting.");

}

public void start () {

System.out.println("Starting " + threadName );

if (t == null) {

t = new Thread (this, threadName);

t.start ();

}

}

void suspend() {

suspended = true;

}

synchronized void resume() {

suspended = false;

notify();

}

}

public class TestThread {

public static void main(String args[]) {

RunnableDemo R1 = new RunnableDemo( "Thread-1");

R1.start();

RunnableDemo R2 = new RunnableDemo( "Thread-2");

R2.start();

try {

Thread.sleep(1000);

R1.suspend();

System.out.println("Suspending First Thread");

Thread.sleep(1000);

R1.resume();

System.out.println("Resuming First Thread");

R2.suspend();

System.out.println("Suspending thread Two");

Thread.sleep(1000);

R2.resume();

System.out.println("Resuming thread Two");

}catch (InterruptedException e) {

System.out.println("Main thread Interrupted");

}try {

System.out.println("Waiting for threads to finish.");

R1.t.join();

R2.t.join();

}catch (InterruptedException e) {

System.out.println("Main thread Interrupted");

}

System.out.println("Main thread exiting.");

}

}

The above program produces the following output −

Output

Creating Thread-1

Starting Thread-1

Creating Thread-2

Starting Thread-2

Running Thread-1

Thread: Thread-1, 10

Running Thread-2

Thread: Thread-2, 10

Thread: Thread-1, 9

Thread: Thread-2, 9

Thread: Thread-1, 8

Thread: Thread-2, 8

Thread: Thread-1, 7

Thread: Thread-2, 7

Suspending First Thread

Thread: Thread-2, 6

Thread: Thread-2, 5

Thread: Thread-2, 4

Resuming First Thread

Suspending thread Two

Thread: Thread-1, 6

Thread: Thread-1, 5

Thread: Thread-1, 4

Thread: Thread-1, 3

Resuming thread Two

Thread: Thread-2, 3

Waiting for threads to finish.

Thread: Thread-1, 2

Thread: Thread-2, 2

Thread: Thread-1, 1

Thread: Thread-2, 1

Thread Thread-1 exiting.

Thread Thread-2 exiting.

Main thread exiting.

# Java Networking

Java Networking is a concept of connecting two or more computing devices together so that we can share resources.

Java socket programming provides facility to share data between different computing devices.

### Advantage of Java Networking

1. sharing resources
2. centralize software management

Do You Know ?

* How to perform connection-oriented Socket Programming in networking ?
* How to display the data of any online web page ?
* How to get the IP address of any host name e.g. www.google.com ?
* How to perform connection-less socket programming in networking ?

## Java Networking Terminology

The widely used java networking terminologies are given below:

1. IP Address
2. Protocol
3. Port Number
4. MAC Address
5. Connection-oriented and connection-less protocol
6. Socket

### 1) IP Address

IP address is a unique number assigned to a node of a network e.g. 192.168.0.1 . It is composed of octets that range from 0 to 255.

It is a logical address that can be changed.

### 2) Protocol

A protocol is a set of rules basically that is followed for communication. For example:

* TCP
* FTP
* Telnet
* SMTP
* POP etc.

### 3) Port Number

The port number is used to uniquely identify different applications. It acts as a communication endpoint between applications.

The port number is associated with the IP address for communication between two applications.

### 4) MAC Address

MAC (Media Access Control) Address is a unique identifier of NIC (Network Interface Controller). A network node can have multiple NIC but each with unique MAC.

### 5) Connection-oriented and connection-less protocol

In connection-oriented protocol, acknowledgement is sent by the receiver. So it is reliable but slow. The example of connection-oriented protocol is TCP.

But, in connection-less protocol, acknowledgement is not sent by the receiver. So it is not reliable but fast. The example of connection-less protocol is UDP.

### 6) Socket

A socket is an endpoint between two way communication.

Visit next page for java socket programming.

What we will learn in Networking Tutorial

* Networking and Networking Terminology
* Socket Programming (Connection-oriented)
* URL class
* Displaying data of a webpage by URLConnection class
* InetAddress class
* DatagramSocket and DatagramPacket (Connection-less)