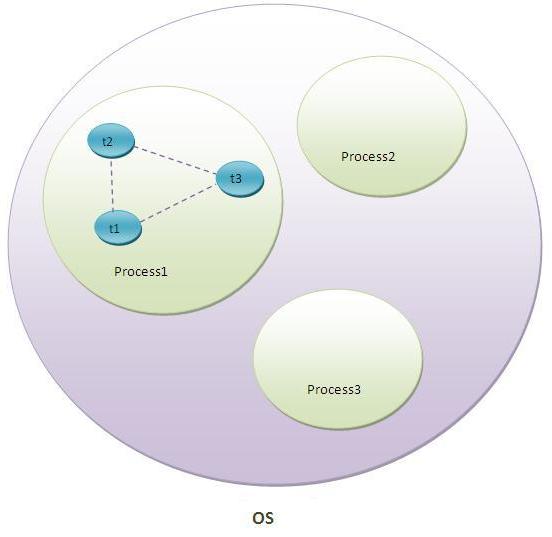
What is Thread in java

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.

## 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)

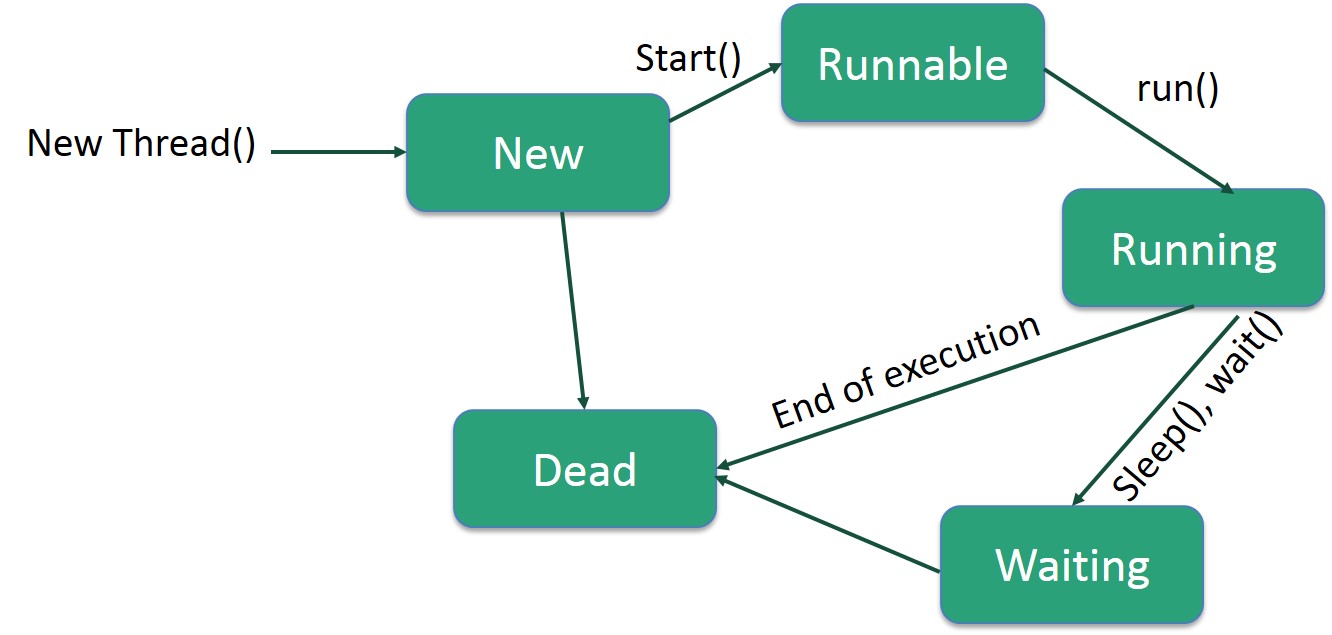
### 1) Process-based Multitasking (Multiprocessing)

* 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.

1. The main purpose of multithreading is to provide simultaneous execution of two or more parts of a program to maximum utilize the CPU time. A multithreaded program contains two or more parts that can run concurrently. Each such part of a program called thread.

* 2. Threads are lightweight sub-processes, they share the common memory space. In Multithreaded environment, programs that are benefited from multithreading, utilize the maximum CPU time so that the idle time can be kept to minimum.
* 3. A thread can be in one of the following states:  
  NEW – A thread that has not yet started is in this state.  
  RUNNABLE – A thread executing in the Java virtual machine is in this state.  
  BLOCKED – A thread that is blocked waiting for a monitor lock is in this state.  
  WAITING – A thread that is waiting indefinitely for another thread to perform a particular action is in this state.  
  TIMED\_WAITING – A thread that is waiting for another thread to perform an action for up to a specified waiting time is in this state.  
  TERMINATED – A thread that has exited is in this state.  
  A thread can be in only one state at a given point in time.
* Life cycle of a thread
* 

## Multitasking vs Multithreading vs Multiprocessing vs parallel processing

If you are new to java you may get confused among these terms as they are used quite frequently when we discuss multithreading. Let’s talk about them in brief.

**Multitasking:**Ability to execute more than one task at the same time is known as multitasking.

**Multithreading:**We already discussed about it. It is a process of executing multiple threads simultaneously. Multithreading is also known as Thread-based Multitasking.

**Multiprocessing:** It is same as multitasking, however in multiprocessing more than one CPUs are involved. On the other hand one CPU is involved in multitasking.

**Parallel Processing:** It refers to the utilization of multiple CPUs in a single computer system.

**Creating a thread in Java**

There are two ways to create a thread in Java:  
1) By extending Thread class.  
2) By implementing Runnable interface.

## Create a Thread by Extending a Thread Class

The first way to create a thread is to create a new class that extends **Thread** class using the following two simple steps. This approach provides more flexibility in handling multiple threads created using available methods in Thread class.

### Step 1

You will need to override **run( )** method available in Thread class. This method provides an entry point for the thread and you will put your complete business logic inside this method. Following is a simple syntax of run() method −

public void run( )

### Step 2

Once Thread object is created, you can start it by calling **start()** method, which executes a call to run( ) method. Following is a simple syntax of start() method −

void start( );

### 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 intgetPriority():**returns the priority of the thread. 7. **public intsetPriority(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 intgetId():**returns the id of the thread. 12. **public Thread.StategetState():**returns the state of the thread. 13. **public booleanisAlive():**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 booleanisDaemon():**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 booleanisInterrupted():**tests if the thread has been interrupted. 22. **public static booleaninterrupted():**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...

### 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. |

|  |
| --- |
| //create thread using Thread class  class thread1 extends Thread  {  public void run()  {  try  {  for(inti=1;i<=10;i++)  {  Thread.sleep(1000);  System.out.println("welcome java : " +i);  }  }  catch(InterruptedException e)  {  System.out.println("the error is : "+e);  }  } //end of run  } //end of thread1  class threaddemo1  {  public static void main(String args[])  {  thread1 obj=new thread1();  obj.start();  }  } |

Output

|  |
| --- |
| E:\javaprgs\threads>javac threaddemo1.java  E:\javaprgs\threads>java threaddemo1  welcome java : 1  welcome java : 2  welcome java : 3  welcome java : 4  welcome java : 5  welcome java : 6  welcome java : 7  welcome java : 8  welcome java : 9  welcome java : 10 |

In the above program , the class **thread1** is defined as a subclass of the **Thread** class. An object **obj** is created for this **thread1** class. Then the **start()** method is invoked. At this stage, the thread **obj** will be in the **ready** state. JVM will subsequently take it up for execution. The thread **obj** will now move to the **running** state.

At this stage, the **run()** method which implements the thread’s behaviour, will be executed.

The execution of **sleep()** method may at times result in the generation of a special type of exception, known as **InterruptedException**. As this is a **checked exception**, it has to be handled with **try-catch** as follows

try

{

------

Thread.sleep();

----

}

Catch(InterruptedException e)

{

}

Whenever the **sleep()** method which is specified in the **run()** method is executed, the thread **obj** will move to the **waiting** state. When the prescribed sleep time gets over, the thread will move to the **ready** state and then to the **running** state. When the ***run()*** method completes, the thread x will move to the **dead** state.

Example2:

|  |
| --- |
| //create thread using Thread class  class thread2 extends Thread  {  double sqrtvalue;  public void run()  {  try  {  for(inti=1;i<=10;i++)  {  Thread.sleep(1000);  sqrtvalue=Math.sqrt(i);  System.out.println("The sqrt of "+i +" is ; "+sqrtvalue);  }  }  catch(InterruptedException e)  {  System.out.println("the error is : "+e);  }  } //end of run  } //end of thread1  class threaddemo2  {  public static void main(String args[])  {  thread2 obj=new thread2();  obj.start();  }  } |

Output

|  |
| --- |
| E:\javaprgs\threads>javac threaddemo2.java  E:\javaprgs\threads>java threaddemo2  The sqrt of 1 is ; 1.0  The sqrt of 2 is ; 1.4142135623730951  The sqrt of 3 is ; 1.7320508075688772  The sqrt of 4 is ; 2.0  The sqrt of 5 is ; 2.23606797749979  The sqrt of 6 is ; 2.449489742783178  The sqrt of 7 is ; 2.6457513110645907  The sqrt of 8 is ; 2.8284271247461903  The sqrt of 9 is ; 3.0  The sqrt of 10 is ; 3.1622776601683795 |

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, intnanos)throws InterruptedException

Before we begin with the programs(code) of creating threads, let’s have a look at these methods of Thread class. We have used few of these methods in the example below.

* getName(): It is used for Obtaining a thread’s name
* getPriority(): Obtain a thread’s priority
* isAlive(): Determine if a thread is still running
* join(): Wait for a thread to terminate
* run(): Entry point for the thread
* sleep(): suspend a thread for a period of time
* start(): start a thread by calling its run() method

## Naming Thread and current thread

The Thread class provides methods to change and get the name of a thread. By default, each thread has a name i.e. thread-0, thread-1 and so on. By we can change the name of the thread by using setName() method. The syntax of setName() and getName() methods are given below:

1. **public String getName():** is used to return the name of a thread.

## **public void setName(String name):** is used to change the name of a thread.

## Naming thread

## Example of naming thread

|  |
| --- |
| class testmultnaming1 extends Thread{int fact=1;public void run(){for(inti=1;i<=5;i++){fact=fact\*i;}System.out.println(fact);} //end of runpublic static void main(String args[]){testmultnaming1 t1=new testmultnaming1();testmultnaming1 t2=new testmultnaming1();System.out.println("name of t1: "+t1.getName());System.out.println("name of t2: "+t2.getName());t1.start();t2.start();t1.setName("factorial");t2.setName("factorial two");System.out.println("After changing the name of t1: "+t1.getName());System.out.println("After changing the name of t2: "+t2.getName());} //end of main} // end of main class |

## Output

|  |
| --- |
| E:\javaprgs\thread>javac testmultnaming1.javaE:\javaprgs\thread>java -cp . testmultnaming1name of t1: Thread-0name of t2: Thread-1After changing the name of t1: factorial120120After changing the name of t2: factorial two |

## Current Thread

## This method returns a reference of currently executing thread

## Syntax

## Public static Thread currentThread()

## Example

|  |
| --- |
| class testsleepmethod1 extends Thread{public void run(){for (inti=1;i<5;i++){try{Thread.sleep(500);}catch(InterruptedException e){System.out.println(e);} // end of try catchSystem.out.println(Thread.currentThread().getName()+" "+i);} //end of for loop} //end of runpublic static void main(String args[]){testsleepmethod1 t1=new testsleepmethod1();testsleepmethod1 t2=new testsleepmethod1();t1.start();t2.start();}} |

## Output

|  |
| --- |
| E:\javaprgs\thread>javac testsleepmethod1.javaE:\javaprgs\thread>java -cp . testsleepmethod1Thread-0 1Thread-1 1Thread-0 2Thread-1 2Thread-0 3Thread-1 3Thread-0 4Thread-1 4 |

## Create a Thread by Implementing a Runnable Interface

If your class is intended to be executed as a thread then you can achieve this by implementing a **Runnable** interface. You will need to follow three basic steps −

### Step 1

As a first step, you need to implement a run() method provided by a **Runnable** interface. This method provides an entry point for the thread and you will put your complete business logic inside this method. Following is a simple syntax of the run() method −

public void run( )

### Step 2

As a second step, you will instantiate a **Thread** object using the following constructor −

Thread(Runnable threadObj, String threadName);

Where, *threadObj* is an instance of a class that implements the **Runnable**interface and **threadName** is the name given to the new thread.

### Step 3

Once a Thread object is created, you can start it by calling **start()** method, which executes a call to run( ) method. Following is a simple syntax of start() method −

void start();

|  |
| --- |
| // program to create a thread using the Runnable interfaceclass Thread3 implements Runnable{public void run(){try{for (inti=1;i<=10;i++){Thread.sleep(1000);System.out.println("welcome : "+i);}}catch(Exception e){System.out.println(e);}}}class threaddemo3{public static void main(String args[]){Thread3 rx=new Thread3();Thread t3=new Thread(rx);t3.start();}} |

## Output

|  |
| --- |
| E:\javaprgs\thread>javac threaddemo3.javaE:\javaprgs\thread>java -cp . threaddemo3welcome : 1welcome : 2welcome : 3welcome : 4welcome : 5welcome : 6welcome : 7welcome : 8welcome : 9welcome : 10 |

|  |
| --- |
| // program to create a thread using the Runnable interfaceclass Thread4 implements Runnable{public void run(){try{for (inti=1;i<=10;i++){intfactvalue=1;Thread.sleep(1000);for(int n=1;n<=i;n++){factvalue=factvalue\*n;}System.out.println("Factorial value of "+i+" is "+factvalue);}//end of outer for} //end of trycatch(Exception e){System.out.println(e);} //end of catch} //end of run} //end of Thread4 classclass threaddemo4{public static void main(String args[]){Thread4 rx=new Thread4();Thread t4=new Thread(rx);t4.start();}} |

## Output

|  |
| --- |
| E:\javaprgs\thread>javac threaddemo4.javaE:\javaprgs\thread>java -cp . threaddemo4Factorial value of 1 is 1Factorial value of 2 is 2Factorial value of 3 is 6Factorial value of 4 is 24Factorial value of 5 is 120Factorial value of 6 is 720Factorial value of 7 is 5040Factorial value of 8 is 40320Factorial value of 9 is 362880Factorial value of 10 is 3628800 |

|  |
| --- |
| class childthread implements Runnable{Thread t;childthread(){t=new Thread(this,"demo thread");System.out.println("Child thread : "+t);t.start();}public void run(){try{for(inti=5;i>0;i--){System.out.println("Child thread = "+i);Thread.sleep(500);} //for}//trycatch(InterruptedException e){System.out.println("Child interrupted");}//catchSystem.out.println("exiting Child thread ");} //run} //childthread classclass threaddemo1{public static void main(String args[]){childthreadrx=new childthread();try{for(inti=5;i>0;i--){System.out.println("Main thread = "+i);Thread.sleep(1000);} //for} //trycatch(InterruptedException e){System.out.println("Main thread interrupted");} //catchSystem.out.println("main thread exiting");}} |

## Output

|  |
| --- |
| E:\javaprgs\thread>javac threaddemo1.javaE:\javaprgs\thread>java -cp . threaddemo1Child thread : Thread[demo thread,5,main]Main thread = 5Child thread = 5Child thread = 4Main thread = 4Child thread = 3Child thread = 2Main thread = 3Child thread = 1exiting Child threadMain thread = 2Main thread = 1main thread exiting |

## Using extends Thread

|  |
| --- |
| class newthread extends Thread{newthread(){super("demo thread");System.out.println("Child thread : "+this);start();}public void run(){try{for(inti=5;i>0;i--){System.out.println("Child thread = "+i);Thread.sleep(500);} //for}//trycatch(InterruptedException e){System.out.println("Child interrupted");}//catchSystem.out.println("exiting Child thread ");} //run} //childthread classclass threaddemo2{public static void main(String args[]){new newthread();try{for(inti=5;i>0;i--){System.out.println("Main thread = "+i);Thread.sleep(1000);} //for} //trycatch(InterruptedException e){System.out.println("Main thread interrupted");} //catchSystem.out.println("main thread exiting");}} |

## The output will be the same as the previous program

## MULTITHREAD

|  |
| --- |
| class multithread implements Runnable{String name;Thread t;multithread(String tname){name=tname;t=new Thread(this,name);System.out.println("new thread : "+t);t.start();}public void run(){try{for(inti=5;i>0;i--){System.out.println("Child thread = "+i);Thread.sleep(500);} //for}//trycatch(InterruptedException e){System.out.println(name+" interrupted");}//catchSystem.out.println(name+ "exiting thread ");} //run} //childthread classclass threadmultidemo1{public static void main(String args[]){new multithread("one");new multithread("two");new multithread("three");try{Thread.sleep(1000);} //trycatch(InterruptedException e){System.out.println("Main thread interrupted");} //catchSystem.out.println("main thread exiting");}} |

## OUTPUT

|  |
| --- |
| E:\javaprgs\thread>java -cp . threadmultidemo1new thread : Thread[one,5,main]new thread : Thread[two,5,main]new thread : Thread[three,5,main]Child thread = 5Child thread = 5Child thread = 5Child thread = 4Child thread = 4Child thread = 4main thread exitingChild thread = 3Child thread = 3Child thread = 3Child thread = 2Child thread = 2Child thread = 2Child thread = 1Child thread = 1Child thread = 1oneexiting threadtwoexiting threadthreeexiting thread |

## Thread Priorities

Every Java thread has a priority that helps the operating system determine the order in which threads are scheduled.

Java thread priorities are in the range between MIN\_PRIORITY (a constant of 1) and MAX\_PRIORITY (a constant of 10). By default, every thread is given priority NORM\_PRIORITY (a constant of 5).

Threads with higher priority are more important to a program and should be allocated processor time before lower-priority threads. However, thread priorities cannot guarantee the order in which threads execute and are very much platform dependent.

|  |
| --- |
| class X extends Thread  {  public void run()  {  System.out.println("thread X started");  for(inti=0;i<5;i++)  {  System.out.println("\t value of i in thread X :"+i);  }  System.out.println("Thread X finished");  }  }  class Y extends Thread  {  public void run()  {  System.out.println("thread Y started");  for(inti=0;i<5;i++)  {  System.out.println("\t value of i in thread Y :"+i);  }  System.out.println("Thread Y finished");  }  }  class Z extends Thread  {  public void run()  {  System.out.println("thread Z started");  for(inti=0;i<5;i++)  {  System.out.println("\t value of i in thread Z :"+i);  }  System.out.println("Thread Z finished");  }  }  class threadpriority1  {  public static void main(String args[])  {  System.out.println("Main thread started");  X tx=new X();  Y ty=new Y();  Z tz=new Z();  tx.setPriority(Thread.MIN\_PRIORITY); //priority=1  ty.setPriority(tx.getPriority()+1); // priority=  tz.setPriority(Thread.MAX\_PRIORITY); // priority=10  tx.start();  ty.start();  tz.start();  }  } |

Output

|  |
| --- |
| D:\javaprgs>java threadpriority1  Main thread started  thread X started  thread Z started  value of i in thread Z :0  value of i in thread Z :1  value of i in thread Z :2  value of i in thread Z :3  value of i in thread Z :4  Thread Z finished  thread Y started  value of i in thread Y :0  value of i in thread Y :1  value of i in thread Y :2  value of i in thread Y :3  value of i in thread Y :4  Thread Y finished  value of i in thread X :0  value of i in thread X :1  value of i in thread X :2  value of i in thread X :3  value of i in thread X :4  Thread X finished |

# 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 |

|  |
| --- |
| class testjoinmethod1 extends Thread  {  public void run()  {  for(inti=1;i<=5;i++)  {  try  {  Thread.sleep(500);  }  catch(Exception e)  {  System.out.println(e);  }  System.out.println(i);  }  }  public static void main(String args[])  {  testjoinmethod1 t1=new testjoinmethod1();  testjoinmethod1 t2=new testjoinmethod1();  testjoinmethod1 t3=new testjoinmethod1();  t1.start();  try  {  t1.join();  }  catch(Exception e)  {  System.out.println(e);  }  t2.start();  t3.start();  }  } |

Output

|  |
| --- |
| D:\javaprgs>javac testjoinmethod1.java  D:\javaprgs>java testjoinmethod1  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.

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

|  |
| --- |
| class testjoinmethod2 extends Thread  {  public void run()  {  for(inti=1;i<=5;i++)  {  try  {  Thread.sleep(1000);  }  catch(Exception e)  {  System.out.println(e);  }  System.out.println("thread name is : "+Thread.currentThread().getName()+" "+i);  }  }  public static void main(String args[])  {  testjoinmethod2 t1=new testjoinmethod2();  testjoinmethod2 t2=new testjoinmethod2();  testjoinmethod2 t3=new testjoinmethod2();  t1.start();  t2.start();  t3.start();  }  } |

if we remove the join in the above program

|  |
| --- |
| D:\javaprgs\threads>javac testjoinmethod2.java  D:\javaprgs\threads>java testjoinmethod2  thread name is : Thread-2 1  thread name is : Thread-1 1  thread name is : Thread-0 1  thread name is : Thread-1 2  thread name is : Thread-0 2  thread name is : Thread-2 2  thread name is : Thread-2 3  thread name is : Thread-1 3  thread name is : Thread-0 3  thread name is : Thread-2 4  thread name is : Thread-0 4  thread name is : Thread-1 4  thread name is : Thread-1 5  thread name is : Thread-0 5  thread name is : Thread-2 5 |

Multiplication table

|  |
| --- |
| public class multiplication\_table implements Runnable  {  int number;  public multiplication\_table(int n)  {  this.number=n;  }  public void run()  {  for(inti=1;i<=10;i++)  {  System.out.println(Thread.currentThread().getName()+" "+ number+"\*"+i+"="+i\*number);  }  } //run  public static void main(String args[])  {  System.out.println("multiplication table");  for(int j=1;j<=3;j++)  {  multiplication\_tableobj=new multiplication\_table(j);  Thread t1=new Thread(obj);  t1.start();  }  }  } |

Output

|  |
| --- |
| D:\javaprgs>java multiplication\_table  multiplication table  Thread-0 1\*1=1  Thread-0 1\*2=2  Thread-0 1\*3=3  Thread-0 1\*4=4  Thread-0 1\*5=5  Thread-2 3\*1=3  Thread-0 1\*6=6  Thread-2 3\*2=6  Thread-0 1\*7=7  Thread-2 3\*3=9  Thread-0 1\*8=8  Thread-2 3\*4=12  Thread-0 1\*9=9  Thread-2 3\*5=15  Thread-0 1\*10=10  Thread-2 3\*6=18  Thread-2 3\*7=21  Thread-2 3\*8=24  Thread-2 3\*9=27  Thread-2 3\*10=30  Thread-1 2\*1=2  Thread-1 2\*2=4  Thread-1 2\*3=6  Thread-1 2\*4=8  Thread-1 2\*5=10  Thread-1 2\*6=12  Thread-1 2\*7=14  Thread-1 2\*8=16  Thread-1 2\*9=18  Thread-1 2\*10=20 |

Example for multithread

|  |
| --- |
| class evenThread implements Runnable  {  int a[]=new int[10];  String name;  Thread t;  evenThread(String name, int a[])  {  t=new Thread(this,name);  this.a=a;  t.start();  }  public void run()  {  try  {  for(inti=0;i<a.length;i++)  {  if(a[i]%2==0)  System.out.println("Even number of index ["+i+"] is " +a[i]);  }  }  catch(Exception e)  {  System.out.println(e);  }  } //run  } //evenThread  class oddThread implements Runnable  {  int a[]=new int[10];  String name;  Thread t;  oddThread(String name, int a[])  {  t=new Thread(this,name);  this.a=a;  t.start();  }  public void run()  {  try  {  for(inti=0;i<a.length;i++)  {  if(a[i]%2!=0)  System.out.println("odd number of index ["+i+"] is " +a[i]);  }  }  catch(Exception e)  {  System.out.println(e);  }  } //run  } //oddThread  class evenoddthread  {  public static void main(String args[])  {  int a[]={12,4,3,78,55,35,90,6,7,56};  evenThread e=new evenThread("even",a);  oddThread o=new oddThread("odd",a);  }  } |

Output

|  |
| --- |
| E:\javaprgs\threads>javac evenoddthread.java  E:\javaprgs\threads>java evenoddthread  Even number of index [0] is 12  Even number of index [1] is 4  Even number of index [3] is 78  Even number of index [6] is 90  Even number of index [7] is 6  Even number of index [9] is 56  odd number of index [2] is 3  odd number of index [4] is 55  odd number of index [5] is 35  odd number of index [8] is 7 |

isAlive() method

Sometimes one thread needs to know when other thread is terminating. In java, **isAlive()** and **join()** are two different methods that are used to check whether a thread has finished its execution or not.

syntax

final boolean isAlive( )

this method returns ture if the thread upon which it is called is still running. It returns false otherwise.

|  |
| --- |
| class jointhread1 extends Thread  {  public void run()  {  System.out.println("r1");  try  {  Thread.sleep(500);  } //try  catch(InterruptedException ex)  {  System.out.println(ex);  } //catch  System.out.println("r2");  } //run  public static void main(String args[])  {  jointhread1 t1=new jointhread1();  jointhread1 t2=new jointhread1();  t1.start();  t2.start();  System.out.println("thread 1 is "+t1.isAlive());  System.out.println("thread 2 is "+t2.isAlive());  }  } |

Output

|  |
| --- |
| E:\javaprgs\threads>java jointhread1  r1  r1  thread 1 is true  thread 2 is true  r2  r2 |

Another example of isAlive()

This example uses join( ) to ensure that the main thread is the last to stop. It also demonstrates the isAlive( ) method.

|  |
| --- |
| class newThread implements Runnable  {  String name;  Thread t;  newThread(String tname)  {  name=tname;  t=new Thread(this,name);  System.out.println("New Thread : "+t);  t.start();  }  public void run()  {  try  {  for(int i=5;i>0;i--)  {  System.out.println(name + ": "+i);  Thread.sleep(1000);  } //for  } //try  catch(InterruptedException e)  {  System.out.println(name+" interrupted");  } //catch  System.out.println(name+" exiting");  } //run  } //class newThread  class isaliveeg2  {  public static void main(String args[])  {  newThread ob1=new newThread("one");  newThread ob2=new newThread("Two");  newThread ob3=new newThread("Three");  System.out.println("Thread one is alive: "+ob1.t.isAlive());  System.out.println("Thread Two is alive: "+ob2.t.isAlive());  System.out.println("Thread Three is alive: "+ob3.t.isAlive());  try  {  System.out.println("waiting for threads to finish");  ob1.t.join();  ob2.t.join();  ob3.t.join();  }  catch(InterruptedException e)  {  System.out.println("Main thread interrupted");  }  System.out.println("Thread one is alive: "+ob1.t.isAlive());  System.out.println("Thread Two is alive: "+ob2.t.isAlive());  System.out.println("Thread Three is alive: "+ob3.t.isAlive());  System.out.println("Main thread exiting");  }  } |

Output

|  |
| --- |
| E:\javaprgs\threads>javac isaliveeg2.java  E:\javaprgs\threads>java isaliveeg2  New Thread : Thread[one,5,main]  New Thread : Thread[Two,5,main]  one: 5  New Thread : Thread[Three,5,main]  Two: 5  Three: 5  Thread one is alive: true  Thread Two is alive: true  Thread Three is alive: true  waiting for threads to finish  one: 4  Two: 4  Three: 4  one: 3  Two: 3  Three: 3  one: 2  Two: 2  Three: 2  one: 1  Two: 1  Three: 1  one exiting  Two exiting  Three exiting  Thread one is alive: false  Thread Two is alive: false  Thread Three is alive: false  Main thread exiting |

Synchronization

When two or more threads need access to a shared resource, they need some way to ensure that the resource will be used by only one thread at a time. The process by which this is achieved is called *synchronization*

Key to synchronization is the concept of the monitor (also called a *semaphore*). A *monitor*

is an object that is used as a mutually exclusive lock, or *mutex.* Only one thread can *own* a monitor at a given time. When a thread acquires a lock, it is said to have *entered* the monitor.

All other threads attempting to enter the locked monitor will be suspended until the first

thread *exits* the monitor. These other threads are said to be *waiting* for the monitor. A thread that owns a monitor can reenter the same monitor if it so desires.

Synchronization is easy in Java, because all objects have their own implicit monitor associated with them. To enter an object’s monitor, just call a method that has been modified with the **synchronized** keyword. While a thread is inside a synchronized method, all other threads that try to call it (or any other synchronized method) on the same instance have to wait. To exit the monitor and relinquish control of the object to the next waiting thread, the owner of the monitor simply returns from the synchronized method.

### 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.

Synchronized method

|  |
| --- |
| class Table  {  void printTable(int n)  {  for(inti=1;i<=5;i++)  {  System.out.println(n+"\*"+i+"= "+n\*i);  try  {  Thread.sleep(400);  }  catch(Exception e)  {  System.out.println(e);  } //catch  } //for  } //printTable  } // clas Table  class myThread1 extends Thread  {  Table t;  myThread1(Table t)  {  this.t=t;  }  public void run()  {  t.printTable(5);  } //run  } //myThread1  class myThread2 extends Thread  {  Table t;  myThread2(Table t)  {  this.t=t;  }  public void run()  {  t.printTable(100);  } //run  } //myThread2  class testsynchronization  {  public static void main(String args[])  {  Table obj=new Table();  myThread1 t1=new myThread1(obj);  myThread2 t2=new myThread2(obj);  t1.start();  t2.start();  }  } |

Output

|  |
| --- |
| E:\javaprgs\thread>javac testsynchronization.java  E:\javaprgs\thread>java -cp .testsynchronization  5\*1= 5  100\*1= 100  5\*2= 10  100\*2= 200  5\*3= 15  100\*3= 300  5\*4= 20  100\*4= 400  5\*5= 25  100\*5= 500 |

Using synchronization

|  |
| --- |
| class Table  {  synchronized void printTable(int n)  {  for(inti=1;i<=5;i++)  {  System.out.println(n+"\*"+i+"= "+n\*i);  try  {  Thread.sleep(400);  }  catch(Exception e)  {  System.out.println(e);  } //catch  } //for  } //printTable  } // clas Table  class myThread1 extends Thread  {  Table t;  myThread1(Table t)  {  this.t=t;  }  public void run()  {  t.printTable(5);  } //run  } //myThread1  class myThread2 extends Thread  {  Table t;  myThread2(Table t)  {  this.t=t;  }  public void run()  {  t.printTable(100);  } //run  } //myThread2  class testsynchronization1  {  public static void main(String args[])  {  Table obj=new Table();  myThread1 t1=new myThread1(obj);  myThread2 t2=new myThread2(obj);  t1.start();  t2.start();  }  } |

Output

|  |
| --- |
| E:\javaprgs\thread>javac testsynchronization1.java  E:\javaprgs\thread>java -cp . testsynchronization1  5\*1= 5  5\*2= 10  5\*3= 15  5\*4= 20  5\*5= 25  100\*1= 100  100\*2= 200  100\*3= 300  100\*4= 400  100\*5= 500 |

Another example for synchronization.

The below program demonstrates without synchronization what will be the output

|  |
| --- |
| class first  {  public void display(String msg)  {  System.out.print("["+msg);  try  {  Thread.sleep(1000);  } //try  catch(InterruptedException e)  {  e.printStackTrace();  } //catch  System.out.println("]");  } //display  } //class first  class second extends Thread  {  String msg;  first fobj;  second(first fp,Stringstr)  {  fobj=fp;  msg=str;  start();  }  public void run()  {  fobj.display(msg);  } //run  } // class second  public class testsynchronization3  {  public static void main(String args[])  {  first fnew=new first();  second s1=new second(fnew,"welcome");  second s2=new second(fnew,"to");  second s3=new second(fnew,"java");  }  } |

Output

|  |
| --- |
| E:\javaprgs\thread>javac testsynchronization3.java  E:\javaprgs\thread>java -cp . testsynchronization3  [welcome[to[java]  ]  ] |

In the above program, object **fnew** of class first is shared by all the three running threads(s1, s2 and s3) to call the shared method(*void***display**). Hence the result is unsynchronized and such situation is called **Race condition**.

#### Synchronized Keyword

To synchronize above program, we must *synchronize* access to the shared **display()** method, making it available to only one thread at a time. This is done by using keyword **synchronized** with display() method.

Synchronized void display (String msg)

#### Using Synchronized block

If you have to synchronize access to an object of a class or you only want a part of a method to be synchronized to an object then you can use synchronized block for it.

|  |
| --- |
| class first  {  public void display(String msg)  {  System.out.print("["+msg);  try  {  Thread.sleep(1000);  } //try  catch(InterruptedException e)  {  e.printStackTrace();  } //catch  System.out.println("]");  } //display  } //class first  class second extends Thread  {  String msg;  first fobj;  second(first fp,Stringstr)  {  fobj=fp;  msg=str;  start();  }  public void run()  {  synchronized(fobj)  {  fobj.display(msg);  }  } //run  } // class second  public class testsynchronization4  {  public static void main(String args[])  {  first fnew=new first();  second s1=new second(fnew,"welcome");  second s2=new second(fnew,"to");  second s3=new second(fnew,"java");  }  } |

Output

|  |
| --- |
| E:\javaprgs\thread>javac testsynchronization4.java  E:\javaprgs\thread>java -cp . testsynchronization4  [welcome]  [to]  [java] |

#### Difference between synchronized keyword and synchronized block

When we use synchronized keyword with a method, it acquires a lock in the object for the whole method. It means that no other thread can use any synchronized method until the current thread, which has invoked it's synchronized method, has finished its execution.

synchronized block acquires a lock in the object only between parentheses after the synchronized keyword. This means that no other thread can acquire a lock on the locked object until the synchronized block exits. But other threads can access the rest of the code of the method.

#### Which is more preferred - Synchronized method or Synchronized block?

In Java, synchronized keyword causes a performance cost. A synchronized method in Java is very slow and can degrade performance. So we must use synchronization keyword in java when it is necessary else, we should use Java synchronized block that is used for synchronizing critical section only.

<http://www.javahelps.com/2015/04/thread-synchronization-in-java.html>

<http://www.javahelps.com/2015/04/thread-synchronization-in-java.html>