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. }

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=TestJoinMethod1)

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. }

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=TestJoinMethod2)

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 miliseconds(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("Sonoo Jaiswal");
16. System.out.println("After changing name of t1:"+t1.getName());
17. }
18. }

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=TestJoinMethod3)

Output:Name of t1:Thread-0

Name of t2:Thread-1

id of t1:8

running...

After changling name of t1:Sonoo Jaiswal

running...

The currentThread() method:

|  |
| --- |
| The currentThread() method returns a reference to the currently executing thread object. |

Syntax:

|  |
| --- |
| public static Thread currentThread() |

***Example of currentThread() method***

1. **class** TestJoinMethod4 **extends** Thread{
2. **public** **void** run(){
3. System.out.println(Thread.currentThread().getName());
4. }
5. }
6. **public** **static** **void** main(String args[]){
7. TestJoinMethod4 t1=**new** TestJoinMethod4();
8. TestJoinMethod4 t2=**new** TestJoinMethod4();
10. t1.start();
11. t2.start();
12. }
13. }

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=TestJoinMethod4)

Output:Thread-0

Thread-1

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. }

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=TestThreadTwice1)

running

Exception in thread "main" java.lang.IllegalThreadStateException

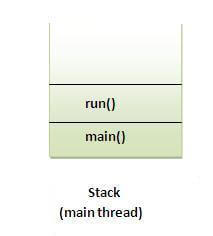
What if we call run() method directly instead start() method?

|  |
| --- |
| * 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. }

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=TestCallRun1)

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. }

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=TestCallRun2)

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. |

# Daemon Thread in Java

**Daemon thread in java** is a service provider thread that provides services to the user thread. Its life depend on the mercy of user threads i.e. when all the user threads dies, JVM terminates this thread automatically.

There are many java daemon threads running automatically e.g. gc, finalizer etc.

You can see all the detail by typing the jconsole in the command prompt. The jconsole tool provides information about the loaded classes, memory usage, running threads etc.

## Points to remember for Daemon Thread in Java

* It provides services to user threads for background supporting tasks. It has no role in life than to serve user threads.
* Its life depends on user threads.
* It is a low priority thread.

### Why JVM terminates the daemon thread if there is no user thread?

The sole purpose of the daemon thread is that it provides services to user thread for background supporting task. If there is no user thread, why should JVM keep running this thread. That is why JVM terminates the daemon thread if there is no user thread.

### Methods for Java Daemon thread by Thread class

The java.lang.Thread class provides two methods for java daemon thread.

|  |  |  |
| --- | --- | --- |
| **No.** | **Method** | **Description** |
| 1) | public void setDaemon(boolean status) | is used to mark the current thread as daemon thread or user thread. |
| 2) | public boolean isDaemon() | is used to check that current is daemon. |

### Simple example of Daemon thread in java

*File: MyThread.java*

1. **public** **class** TestDaemonThread1 **extends** Thread{
2. **public** **void** run(){
3. **if**(Thread.currentThread().isDaemon()){//checking for daemon thread
4. System.out.println("daemon thread work");
5. }
6. **else**{
7. System.out.println("user thread work");
8. }
9. }
10. **public** **static** **void** main(String[] args){
11. TestDaemonThread1 t1=**new** TestDaemonThread1();//creating thread
12. TestDaemonThread1 t2=**new** TestDaemonThread1();
13. TestDaemonThread1 t3=**new** TestDaemonThread1();
15. t1.setDaemon(**true**);//now t1 is daemon thread
17. t1.start();//starting threads
18. t2.start();
19. t3.start();
20. }
21. }

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=TestDaemonThread1)

#### Output

daemon thread work

user thread work

user thread work

#### Note: If you want to make a user thread as Daemon, it must not be started otherwise it will throw IllegalThreadStateException.

*File: MyThread.java*

1. **class** TestDaemonThread2 **extends** Thread{
2. **public** **void** run(){
3. System.out.println("Name: "+Thread.currentThread().getName());
4. System.out.println("Daemon: "+Thread.currentThread().isDaemon());
5. }
7. **public** **static** **void** main(String[] args){
8. TestDaemonThread2 t1=**new** TestDaemonThread2();
9. TestDaemonThread2 t2=**new** TestDaemonThread2();
10. t1.start();
11. t1.setDaemon(**true**);//will throw exception here
12. t2.start();
13. }
14. }

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=TestDaemonThread2)

Output:exception in thread main: java.lang.IllegalThreadStateException

# Java Shutdown Hook

The shutdown hook can be used to perform cleanup resource or save the state when JVM shuts down normally or abruptly. Performing clean resource means closing log file, sending some alerts or something else. So if you want to execute some code before JVM shuts down, use shutdown hook.

### When does the JVM shut down?

The JVM shuts down when:

* user presses ctrl+c on the command prompt
* System.exit(int) method is invoked
* user logoff
* user shutdown etc.

#### The addShutdownHook(Thread hook) method

The addShutdownHook() method of Runtime class is used to register the thread with the Virtual Machine. Syntax:

1. **public** **void** addShutdownHook(Thread hook){}

The object of Runtime class can be obtained by calling the static factory method getRuntime(). For example:

Runtime r = Runtime.getRuntime();

#### Factory method

The method that returns the instance of a class is known as factory method.

### Simple example of Shutdown Hook Continuee

1. **class** MyThread **extends** Thread{
2. **public** **void** run(){
3. System.out.println("shut down hook task completed..");
4. }
5. }
7. **public** **class** TestShutdown1{
8. **public** **static** **void** main(String[] args)**throws** Exception {
10. Runtime r=Runtime.getRuntime();
11. r.addShutdownHook(**new** MyThread());
13. System.out.println("Now main sleeping... press ctrl+c to exit");
14. **try**{Thread.sleep(3000);}**catch** (Exception e) {}
15. }
16. }

Output:Now main sleeping... press ctrl+c to exit

shut down hook task completed..

#### Note: The shutdown sequence can be stopped by invoking the halt(int) method of Runtime class.

### Same example of Shutdown Hook by annonymous class:

1. **public** **class** TestShutdown2{
2. **public** **static** **void** main(String[] args)**throws** Exception {
4. Runtime r=Runtime.getRuntime();
6. r.addShutdownHook(**new** Thread(){
7. **public** **void** run(){
8. System.out.println("shut down hook task completed..");
9. }
10. }
11. );
13. System.out.println("Now main sleeping... press ctrl+c to exit");
14. **try**{Thread.sleep(3000);}**catch** (Exception e) {}
15. }
16. }

**Output: Now main sleeping... press ctrl+c to exit**

**Shut down hook task completed.**

# Interrupting a Thread:

|  |
| --- |
| If any thread is in sleeping or waiting state (i.e. sleep() or wait() is invoked), calling the interrupt() method on the thread, breaks out the sleeping or waiting state throwing InterruptedException. If the thread is not in the sleeping or waiting state, calling the interrupt() method performs normal behaviour and doesn't interrupt the thread but sets the interrupt flag to true. Let's first see the methods provided by the Thread class for thread interruption. |

## The 3 methods provided by the Thread class for interrupting a thread

|  |
| --- |
| * **public void interrupt()** * **public static boolean interrupted()** * **public boolean isInterrupted()** |

## Example of interrupting a thread that stops working

|  |
| --- |
| In this example, after interrupting the thread, we are propagating it, so it will stop working. If we don't want to stop the thread, we can handle it where sleep() or wait() method is invoked. Let's first see the example where we are propagating the exception. |

1. **class** TestInterruptingThread1 **extends** Thread{
2. **public** **void** run(){
3. **try**{
4. Thread.sleep(1000);
5. System.out.println("task");
6. }**catch**(InterruptedException e){
7. **throw** **new** RuntimeException("Thread interrupted..."+e);
8. }
10. }
12. **public** **static** **void** main(String args[]){
13. TestInterruptingThread1 t1=**new** TestInterruptingThread1();
14. t1.start();
15. **try**{
16. t1.interrupt();
17. }**catch**(Exception e){System.out.println("Exception handled "+e);}
19. }
20. }

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=TestInterruptingThread1)

[download this example](https://www.javatpoint.com/src/multi/interrupt1.zip)

Output:Exception in thread-0

java.lang.RuntimeException: Thread interrupted...

java.lang.InterruptedException: sleep interrupted

at A.run(A.java:7)

## Example of interrupting a thread that doesn't stop working

|  |
| --- |
| In this example, after interrupting the thread, we handle the exception, so it will break out the sleeping but will not stop working. |

1. **class** TestInterruptingThread2 **extends** Thread{
2. **public** **void** run(){
3. **try**{
4. Thread.sleep(1000);
5. System.out.println("task");
6. }**catch**(InterruptedException e){
7. System.out.println("Exception handled "+e);
8. }
9. System.out.println("thread is running...");
10. }
12. **public** **static** **void** main(String args[]){
13. TestInterruptingThread2 t1=**new** TestInterruptingThread2();
14. t1.start();
16. t1.interrupt();
18. }
19. }

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=TestInterruptingThread2)

[download this example](https://www.javatpoint.com/src/multi/interrupt2.zip)

Output:Exception handled

java.lang.InterruptedException: sleep interrupted

thread is running...

## Example of interrupting thread that behaves normally

|  |
| --- |
| If thread is not in sleeping or waiting state, calling the interrupt() method sets the interrupted flag to true that can be used to stop the thread by the java programmer later. |

1. **class** TestInterruptingThread3 **extends** Thread{
3. **public** **void** run(){
4. **for**(**int** i=1;i<=5;i++)
5. System.out.println(i);
6. }
8. **public** **static** **void** main(String args[]){
9. TestInterruptingThread3 t1=**new** TestInterruptingThread3();
10. t1.start();
12. t1.interrupt();
14. }
15. }

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=TestInterruptingThread3)

Output:1

2

3

4

5

## What about isInterrupted and interrupted method?

|  |
| --- |
| The isInterrupted() method returns the interrupted flag either true or false. The static interrupted() method returns the interrupted flag afterthat it sets the flag to false if it is true. |

1. **public** **class** TestInterruptingThread4 **extends** Thread{
3. **public** **void** run(){
4. **for**(**int** i=1;i<=2;i++){
5. **if**(Thread.interrupted()){
6. System.out.println("code for interrupted thread");
7. }
8. **else**{
9. System.out.println("code for normal thread");
10. }
12. }//end of for loop
13. }
15. **public** **static** **void** main(String args[]){
17. TestInterruptingThread4 t1=**new** TestInterruptingThread4();
18. TestInterruptingThread4 t2=**new** TestInterruptingThread4();
20. t1.start();
21. t1.interrupt();
23. t2.start();
25. }
26. }

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=TestInterruptingThread4)

Output:Code for interrupted thread

code for normal thread

code for normal thread

code for normal thread

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{
2. **void** printTable(**int** n){//method not synchronized
3. **for**(**int** i=1;i<=5;i++){
4. System.out.println(n\*i);
5. **try**{
6. Thread.sleep(400);
7. }**catch**(Exception e){System.out.println(e);}
8. }
10. }
11. }
13. **class** MyThread1 **extends** Thread{
14. Table t;
15. MyThread1(Table t){
16. **this**.t=t;
17. }
18. **public** **void** run(){
19. t.printTable(5);
20. }
22. }
23. **class** MyThread2 **extends** Thread{
24. Table t;
25. MyThread2(Table t){
26. **this**.t=t;
27. }
28. **public** **void** run(){
29. t.printTable(100);
30. }
31. }
33. **class** TestSynchronization1{
34. **public** **static** **void** main(String args[]){
35. Table obj = **new** Table();//only one object
36. MyThread1 t1=**new** MyThread1(obj);
37. MyThread2 t2=**new** MyThread2(obj);
38. t1.start();
39. t2.start();
40. }
41. }

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. }
34. **public** **class** TestSynchronization2{
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

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

Synchronized block in java

Synchronized block can be used to perform synchronization on any specific resource of the method.

Suppose you have 50 lines of code in your method, but you want to synchronize only 5 lines, you can use synchronized block.

If you put all the codes of the method in the synchronized block, it will work same as the synchronized method.

**Points to remember for Synchronized block**

* Synchronized block is used to lock an object for any shared resource.
* Scope of synchronized block is smaller than the method.

**Syntax to use synchronized block**

1. **synchronized** (object reference expression) {
2. //code block
3. }

Example of synchronized block

Let's see the simple example of synchronized block.

***Program of synchronized block***

1. **class** Table{
3. **void** printTable(**int** n){
4. **synchronized**(**this**){//synchronized block
5. **for**(**int** i=1;i<=5;i++){
6. System.out.println(n\*i);
7. **try**{
8. Thread.sleep(400);
9. }**catch**(Exception e){System.out.println(e);}
10. }
11. }
12. }//end of the method
13. }
15. **class** MyThread1 **extends** Thread{
16. Table t;
17. MyThread1(Table t){
18. **this**.t=t;
19. }
20. **public** **void** run(){
21. t.printTable(5);
22. }
24. }
25. **class** MyThread2 **extends** Thread{
26. Table t;
27. MyThread2(Table t){
28. **this**.t=t;
29. }
30. **public** **void** run(){
31. t.printTable(100);
32. }
33. }
35. **public** **class** TestSynchronizedBlock1{
36. **public** **static** **void** main(String args[]){
37. Table obj = **new** Table();//only one object
38. MyThread1 t1=**new** MyThread1(obj);
39. MyThread2 t2=**new** MyThread2(obj);
40. t1.start();
41. t2.start();
42. }
43. }

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=TestSynchronizedBlock1)

Output:5

10

15

20

25

100

200

300

400

500

Same Example of synchronized block by using annonymous class:

***//Program of synchronized block by using annonymous class***

1. **class** Table{
3. **void** printTable(**int** n){
4. **synchronized**(**this**){//synchronized block
5. **for**(**int** i=1;i<=5;i++){
6. System.out.println(n\*i);
7. **try**{
8. Thread.sleep(400);
9. }**catch**(Exception e){System.out.println(e);}
10. }
11. }
12. }//end of the method
13. }
15. **public** **class** TestSynchronizedBlock2{
16. **public** **static** **void** main(String args[]){
17. **final** Table obj = **new** Table();//only one object
19. Thread t1=**new** Thread(){
20. **public** **void** run(){
21. obj.printTable(5);
22. }
23. };
24. Thread t2=**new** Thread(){
25. **public** **void** run(){
26. obj.printTable(100);
27. }
28. };
30. t1.start();
31. t2.start();
32. }
33. }

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=TestSynchronizedBlock2)

Output:5

10

15

20

25

100

200

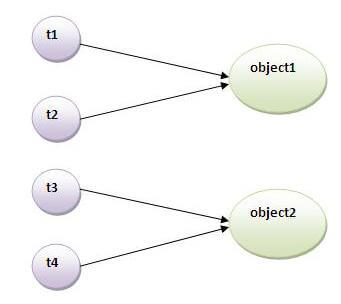
300

400

500

Static synchronization

If you make any static method as synchronized, the lock will be on the class not on object.



Problem without static synchronization

Suppose there are two objects of a shared class(e.g. Table) named object1 and object2.In case of synchronized method and synchronized block there cannot be interference between t1 and t2 or t3 and t4 because t1 and t2 both refers to a common object that have a single lock.But there can be interference between t1 and t3 or t2 and t4 because t1 acquires another lock and t3 acquires another lock.I want no interference between t1 and t3 or t2 and t4.Static synchronization solves this problem.

Example of static synchronization

In this example we are applying synchronized keyword on the static method to perform static synchronization.

1. **class** Table{
3. **synchronized** **static** **void** printTable(**int** n){
4. **for**(**int** i=1;i<=10;i++){
5. System.out.println(n\*i);
6. **try**{
7. Thread.sleep(400);
8. }**catch**(Exception e){}
9. }
10. }
11. }
13. **class** MyThread1 **extends** Thread{
14. **public** **void** run(){
15. Table.printTable(1);
16. }
17. }
19. **class** MyThread2 **extends** Thread{
20. **public** **void** run(){
21. Table.printTable(10);
22. }
23. }
25. **class** MyThread3 **extends** Thread{
26. **public** **void** run(){
27. Table.printTable(100);
28. }
29. }



34. **class** MyThread4 **extends** Thread{
35. **public** **void** run(){
36. Table.printTable(1000);
37. }
38. }
40. **public** **class** TestSynchronization4{
41. **public** **static** **void** main(String t[]){
42. MyThread1 t1=**new** MyThread1();
43. MyThread2 t2=**new** MyThread2();
44. MyThread3 t3=**new** MyThread3();
45. MyThread4 t4=**new** MyThread4();
46. t1.start();
47. t2.start();
48. t3.start();
49. t4.start();
50. }
51. }

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=TestSynchronization4)

Output: 1

2

3

4

5

6

7

8

9

10

10

20

30

40

50

60

70

80

90

100

100

200

300

400

500

600

700

800

900

1000

1000

2000

3000

4000

5000

6000

7000

8000

9000

10000

Same example of static synchronization by annonymous class

In this example, we are using annonymous class to create the threads.

1. **class** Table{
3. **synchronized** **static**  **void** printTable(**int** n){
4. **for**(**int** i=1;i<=10;i++){
5. System.out.println(n\*i);
6. **try**{
7. Thread.sleep(400);
8. }**catch**(Exception e){}
9. }
10. }
11. }
13. **public** **class** TestSynchronization5 {
14. **public** **static** **void** main(String[] args) {
16. Thread t1=**new** Thread(){
17. **public** **void** run(){
18. Table.printTable(1);
19. }
20. };
22. Thread t2=**new** Thread(){
23. **public** **void** run(){
24. Table.printTable(10);
25. }
26. };
28. Thread t3=**new** Thread(){
29. **public** **void** run(){
30. Table.printTable(100);
31. }
32. };
34. Thread t4=**new** Thread(){
35. **public** **void** run(){
36. Table.printTable(1000);
37. }
38. };
39. t1.start();
40. t2.start();
41. t3.start();
42. t4.start();
44. }
45. }

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=TestSynchronization5)

Output: 1

2

3

4

5

6

7

8

9

10

10

20

30

40

50

60

70

80

90

100

100

200

300

400

500

600

700

800

900

1000

1000

2000

3000

4000

5000

6000

7000

8000

9000

10000

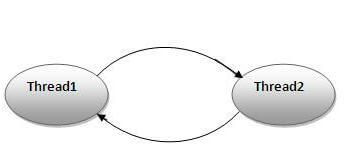
Synchronized block on a class lock:

The block synchronizes on the lock of the object denoted by the reference .class name .class. A static synchronized method printTable(int n) in class Table is equivalent to the following declaration:

1. **static** **void** printTable(**int** n) {
2. **synchronized** (Table.**class**) {       // Synchronized block on class A
3. // ...
4. }
5. }

Deadlock in java

Deadlock in java is a part of multithreading. Deadlock can occur in a situation when a thread is waiting for an object lock, that is acquired by another thread and second thread is waiting for an object lock that is acquired by first thread. Since, both threads are waiting for each other to release the lock, the condition is called deadlock.



Example of Deadlock in java

1. **public** **class** TestDeadlockExample1 {
2. **public** **static** **void** main(String[] args) {
3. **final** String resource1 = "ratan jaiswal";
4. **final** String resource2 = "vimal jaiswal";
5. // t1 tries to lock resource1 then resource2
6. Thread t1 = **new** Thread() {
7. **public** **void** run() {
8. **synchronized** (resource1) {
9. System.out.println("Thread 1: locked resource 1");
11. **try** { Thread.sleep(100);} **catch** (Exception e) {}
13. **synchronized** (resource2) {
14. System.out.println("Thread 1: locked resource 2");
15. }
16. }
17. }
18. };
20. // t2 tries to lock resource2 then resource1
21. Thread t2 = **new** Thread() {
22. **public** **void** run() {
23. **synchronized** (resource2) {
24. System.out.println("Thread 2: locked resource 2");
26. **try** { Thread.sleep(100);} **catch** (Exception e) {}
28. **synchronized** (resource1) {
29. System.out.println("Thread 2: locked resource 1");
30. }
31. }
32. }
33. };

36. t1.start();
37. t2.start();
38. }
39. }

Output: Thread 1: locked resource 1

Thread 2: locked resource 2