**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:

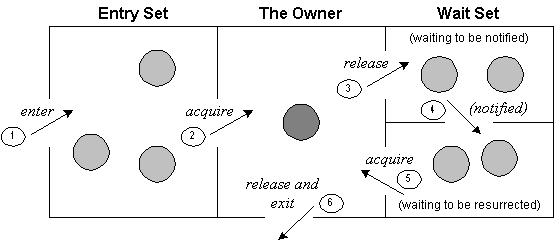
public final void notify()

**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.

1. class Customer{
2. int amount=10000;
4. synchronized void withdraw(int amount){
5. System.out.println("going to withdraw...");
7. if(this.amount<amount){
8. System.out.println("Less balance; waiting for deposit...");
9. try{wait();}catch(Exception e){}
10. }
11. this.amount-=amount;
12. System.out.println("withdraw completed...");
13. }
15. synchronized void deposit(int amount){
16. System.out.println("going to deposit...");
17. this.amount+=amount;
18. System.out.println("deposit completed... ");
19. notify();
20. }
21. }
23. class Test{
24. public static void main(String args[]){
25. final Customer c=new Customer();
26. new Thread(){
27. public void run(){c.withdraw(15000);}
28. }.start();
29. new Thread(){
30. public void run(){c.deposit(10000);}
31. }.start();
33. }}

Output: going to withdraw...

Less balance; waiting for deposit...

going to deposit...

deposit completed...

withdraw 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{
2. public void run(){
3. for(int i=1;i<=2;i++){
4. if(Thread.interrupted()){
5. System.out.println("code for interrupted thread");
6. }
7. else{
8. System.out.println("code for normal thread");
9. }
11. }//end of for loop
12. }
14. public static void main(String args[]){
16. TestInterruptingThread4 t1=new TestInterruptingThread4();
17. TestInterruptingThread4 t2=new TestInterruptingThread4();
19. t1.start();
20. t1.interrupt();
22. t2.start();
24. }
25. }

[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