**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()**

There are three simple methods and a little trick which makes thread communication possible.

All the three methods are listed below −

|  |  |
| --- | --- |
| **Sr.No** | **Method & Description** |
| 1 | **public void wait()**  Causes the current thread to **wait until another thread invokes** the **notify()**. |
| 2 | **public void notify()**  **Wakes up a single thread** that is waiting on this object's monitor. |
| 3 | **public void notifyAll()**  **Wakes up all the threads** that called wait( ) on the same object. |

* These methods have been implemented as **final methods** in Object, so they are available in all the classes. All three **methods can be** **called only from within a synchronized context**.

## 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.

## 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()

* 1. **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 inter thread communication in java**

**The point to point explanation is as follows:**

* + Threads enter to acquire lock.
  + Lock is acquired by a thread.
  + Now thread goes to waiting state if you call wait() method on the object. Otherwise it releases the lock and exits.
  + If you call notify() or notifyAll() method, thread moves to the notified state (runnable state).
  + Now thread is available to acquire lock.
  + 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.

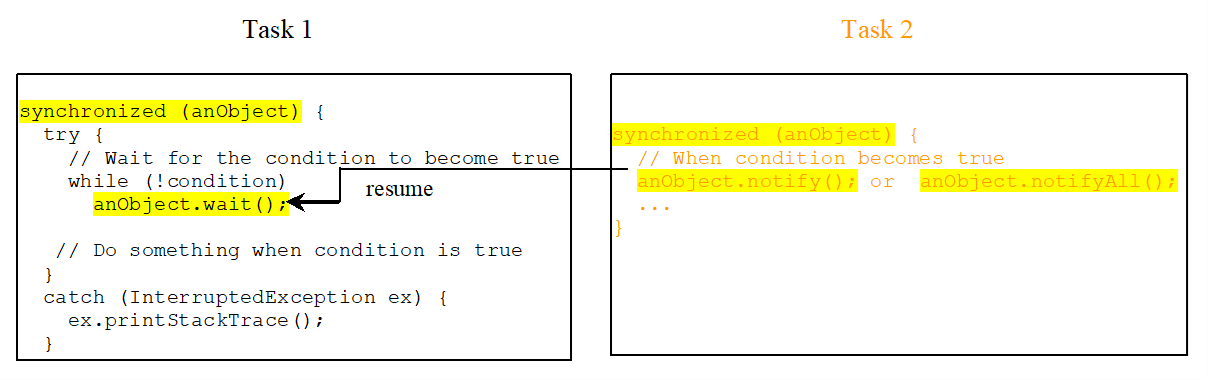
## wait() sleep()

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 after the specified amount of

notifyAll() methods time, sleep is completed.



Let's now work through an example that uses wait() and notify() method. To start, consider the below sample program that incorrectly implements a simple form of the producer/consumer problem.

It consists of the following four classes:

**Q** - the queue that you are trying to synchronize

**Producer** - the threaded object that is producing queue entries **Consumer** - the threaded object which is consuming queue entries

**PC** - the tiny class that creates the single Q, Producer

/\* Java Program Example - Java inter thread communication

\* A correct implementation of producer and consumer \*/

class Q

{ **int num**;

boolean **valueSet = false**; **synchronized** int get()

{ while(!valueSet) try

{ **wait();**

}

catch(InterruptedException e)

{ System.out.println("InterruptedException caught..!!");

}

System.out.println("Got : " + num); valueSet = false;

**notify();** return num;

}

**synchronized** void put(int **num**)

{ while(valueSet)

{ try

{ **wait();**

}

catch(InterruptedException e)

{System.out.println("InterruptedException caught..!!");

}

}

this.num = **num**; **valueSet = true;**

System.out.println("Put : " + num); **notify();**

} }

class Producer **implements Runnable**

{ Q que; Producer(Q que)

{ this.que = que;

**new Thread(this, "Producer").start();**

}

public void run()

{ int n = 0; while(true)

{ **que.put(n++);**

}

}}

class Consumer **implements Runnable**

{ Q que;

Consumer(Q que)

{ this.que = que;

**new Thread(this, "Consumer").start();**

}

public void run()

{ while(true)

{ **que.get();**

}

}

}

class PC

{ public static void main(String args[])

{ Q que = new Q();

**new Producer(que);**

**new Consumer(que);**

System.out.println("Press Control-C to stop...");

}}

Inside get(), wait() is called. This causes its execution to suspend until Producer notifies you that some data is ready. When this happens, execution inside the method named get() resumes.

* After the data has been got, get() calls notify(). This tells Producer that it is fine to put more data in the queue.
* Inside the method named put(), wait() suspends execution until Consumer has removed the item from the queue.
* When the execution resumes, the next item of data is put in the queue, and notify() is called. This tells Consumer that it should now remove it.

Below is some output from this program, which shows the clean synchronous behaviour :

Put : 1

Got : 1

Put : 2

Got : 2

Put : 3

Got : 3

Put : 4

Got : 4

Put : 5

Got : 5

## Thread class A

**{ void add(int b[])**

**{**

**for(int i=0;i<b.length;i++)**

**{b[i]=b[i]+5;}**

**}**

**void sub(int b[])**

**{**

**for(int i=0;i<b.length;i++)**

**{b[i]= b[i]-1;}**

**}**

**}**

**class thread1 extends Thread**

**{ A t;**

**thread1(A t)**

**{this.t=t;}**

**public void run(int a[])**

**{t.sub(a);**

**System.out.println("Result of array subtraction is:"); for(int i=0;i<a.length;i++)**

**{**

**System.out.print(a[i]+" ");**

**}**

**}**

**}**

**class thread2 extends Thread**

**{ A t;**

**thread2(A t)**

**{this.t=t;}**

**public void run(int a[])**

**{ t.add(a);**

**System.out.println("Result of array addtion is:"); for(int i=0;i<a.length;i++)**

**{**

**System.out.print(a[i]+" ");**

**}**

**}**

**}**

**class test**

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

**{ A t =new A();**

**thread1 t1=new thread1(t); thread2 t2=new thread2(t);**

**int [] a1= {1,2,3,4};**

**t1.run(a1);**

**t2.run(a1);**

**}**

**}**

**OUTPUT:**

**Result of array subtraction is:**

**0 1 2 3**

**Result of array addtion is:**

**5 6 7 8**

**Example2**

class thread2 extends Thread

{ public void run(**int b[])**

{int [] c =new int[b.length];

for(int i=0;i<c.length;i++)

{c[i]=5+b[i];

}

System.out.println("Result of array addtion is:"); for(int i=0;i<c.length;i++) { System.out.print(c[i]+" ");

}}

public static void main(String [] args)

{ thread2 t=new thread2();

int [] a1= {1,2,3,4};

t.run(a1);

}}

Result of array addtion is: 6

7

8

9

Example 3

class thread2 extends Thread

{ int a,b,c;

thread2(int a,int b)

{this.a=a; this.b=b;}

public void run()

{System.out.println("Before swap:"+"a:"+a+" "+"b:"+b);

c=a;

a=b;

b=c;

System.out.println("After swap:"+"a:"+a+" "+"b:"+b);}

public static void main(String [] args)

{ int a=6,b=5;

thread2 t=new thread2(a,b);

t.run();

}

}

Output

Before swap:a:6 b:5

After swap:a:5 b:6

**Example 4**

class thread2 extends Thread

{ public void run(**int a,int b)**

{int c ;

System.out.println("before swap:"+”a:”+a+” “+”b:”+b);

c=a;

a=b;

b=c;

}

System.out.println("before swap:"+”a:”+a+” “+”b:”+b);

public static void main(String [] args)

{thread2 t=new thread2();

int a=5,b=6;

}

class A{

t.run(a,b);

}

{ void add(int b[],int n)

{ for(int i=0;i<b.length;i++)

{b[i]=b[i]+n;}

}

void sub(int b[], int m)

{ for(int i=0;i<b.length;i++)

{b[i]= b[i]-m;}

}

}

class thread1 extends Thread

{ A t;

thread1(A t)

{this.t=t;}

public void run(int a[],int n)

{t.sub(a,n);

System.out.println("Result of array subtraction is:"); for(int i=0;i<a.length;i++)

{

System.out.print(a[i]+" ");

}

}

}

class thread2 extends Thread

{ A t;

thread2(A t)

{this.t=t;}

public void run(int a[],int m)

{ t.add(a,m);

System.out.println("Result of array addtion is:"); for(int i=0;i<a.length;i++)

{

System.out.print(a[i]+" ");

}

}

}

class test

{public static void main(String [] args)

{ A t =new A();

thread1 t1=new thread1(t); thread2 t2=new thread2(t);

int [] a1= {1,2,3,4};

t1.run(a1,1);

t2.run(a1,5);

}}

**Exchange two values using generic** public class Question2 {

static <T> T[] exchange(T a[],T elem1, T elem2){ int i1=0,i2=0;

for(int i = 0;i<a.length;i++){ if(a[I ]==elem1)

i1=i; if(a[i]==elem2) i2=i;

}

a[i1]=elem2; a[i2]=elem1;

return a;

}

}

public static void main(String args[]) { Integer a[] = {1,2,3,4,5};

Integer b[] = exchange(a,1,4); String a1[] = {"a","b","c"};

String b1[] = exchange(a1,"a","c"); for(int i = 0;i<b.length;i++)

{System.out.print(b[i]+" ");}

System.out.println();

for(int i = 0;i<b1.length;i++){ System.out.print(b1[i]+" ");

}

}

}

## Sorting an array using generics class Sort {

**public <T extends Number> void sort(T arr[]) { T temp;**

**int n = arr.length;**

**for (int i = 0; i < n-1; i++)**

**for (int j = 0; j < n-i-1; j++)**

**{ double a=arr[j].doubleValue(); double b=arr[j+1].doubleValue();**

**if (a>b)**

**{ temp = arr[j];**

**arr[j] = arr[j+1];**

**arr[j+1] = temp;**

**}**

**}**

for(int q=0;q<arr.length;q++)

**{**

**System.out.println(arr[q]);**

**}**

**}}**

**class A{**

**public static void main (String args[]) { Sort s=new Sort();**

**Integer[] a= {2,34,5,65,4};**

**Double [] b= {2.1,8.9,10.1,3.4,3.4,5.7,3.2,547.9,525.9};**

**System.out.println("First array A1"); s.sort(a);**

**System.out.println("Second array A1"); s.sort(b);**

**}}**

Output

odd:3 Palindrome:1 Prime:4

# public class Box<T>

{private T t;

public void put(T t)

{ this.t = t; } public T get()

{return t; }

public static void main(String[] args)

{Box<Integer> integerBox = new Box<Integer>(); Box<String> stringBox = new Box<String>(); integerBox.put(new Integer(10)); stringBox.add(new String("Hello World"));

System.out.printf("Integer Value :%d\n\n", integerBox.get());

System.out.printf("String Value :%s\n", stringBox.get());

}

}

This will produce the following result −

**Output**

# Integer Value :10

String Value : Hello World

class s1

{void checkeven(int a)

{if(a%2==0)

System.out.println("Even");

else

System.out.println("Not Even");

}

}

class s2

{

void checkodd(int a)

{if(a%2!=0)

System.out.println("odd");

else

System.out.println("Not odd");

}

}

class thread1 extends Thread

{ s1 t;

thread1(s1 t)

{this.t=t;}

public void run(int a)

{

t.checkeven(a);

}

}

class thread2 extends Thread

{ s2 t;

thread1(s2 t)

{this.t=t;}

public void run(int a)

{

t.checkodd(a);

}

}

public static void main(String [] args)

{ S1 t1=new S1();

S2 t2=new S2();

thread1 p=new thread1(t1);

thread2 q=new thread2(t2);

Scanner s=new Scanner(System.in);

int m=s.nextInt();

p.run(m);

q.run(m);

}}