Join():

If a thread wants to wait for another thread to complete its task,then we should use join() method.

Methods:

Public final void join() throws interrupted exception{ }

Public final snychronized void join(long milisecond) throws interrupted exception{ }

Public final snychronized void join( long milisecond,int nanosecond) throws interrupted exception{ }

Example:

class Find extends Thread  
{  
 public void run()  
 {  
 for(int i=0;i<=5;i++) {  
 try {  
 System.*out*.println(Thread.*currentThread*().getName() + " " + i);  
 Thread.*sleep*(1000);  
 } catch (Exception e)  
 {  
 System.*out*.println(e);  
 }  
 }  
 }  
}  
public class Joinss {  
 public static void main(String[] args) {  
 Find f = new Find();  
 f.start();  
  
 for(int i=0;i<=6;i++)  
 { try{  
 //f.join();  
 System.*out*.println(Thread.*currentThread*().getName()+" "+i);  
 Thread.*sleep*(1000);  
  
 }catch (Exception e)  
 {  
 System.*out*.println(e);  
 }  
 }  
 }  
}

Output:

Thread-0 0

Thread-0 1

Thread-0 2

Thread-0 3

Thread-0 4

Thread-0 5

main 0

main 1

main 2

main 3

main 4

main 5

main 6

2nd way

class Find extends Thread  
{  
 public void run()  
 {  
 for(int i=0;i<=5;i++) {  
 try {  
 System.*out*.println(Thread.*currentThread*().getName() + " " + i);  
 Thread.*sleep*(1000);  
 } catch (Exception e)  
 {  
 System.*out*.println(e);  
 }  
 }  
 }  
}  
public class Joinss {  
 public static void main(String[] args)throws InterruptedException {  
 Find f = new Find();  
 f.start();  
 f.join();  
 for(int i=0;i<=6;i++)  
 { try{  
 //f.join();  
 System.*out*.println(Thread.*currentThread*().getName()+" "+i);  
 Thread.*sleep*(1000);  
  
 }catch (Exception e)  
 {  
 System.*out*.println(e);  
 }  
 }  
 }  
}

Output: same output

public class Find extends Thread  
{  
 static Thread *mainthread*;  
 public void run() {  
 for (int i = 0; i <= 5; i++) {  
 try {  
 *mainthread*.join();  
 System.*out*.println(Thread.*currentThread*().getName() + " " + i);  
 Thread.*sleep*(1000);  
 } catch (Exception e) {  
 System.*out*.println(e);  
 }  
 }  
 }  
  
  
 public static void main(String[] args) {  
  
 *mainthread* = Thread.*currentThread*();  
 Find f = new Find();  
 f.start();  
 // f.join();  
 for (int i = 0; i <= 6; i++) {  
 try {  
 //f.join();  
 System.*out*.println(Thread.*currentThread*().getName() + " " + i);  
 Thread.*sleep*(1000);  
  
 } catch (Exception e) {  
 System.*out*.println(e);  
 }  
 }  
  
 }  
  
}

Output:

main 0

main 1

main 2

main 3

main 4

main 5

main 6

Thread-0 0

Thread-0 1

Thread-0 2

Thread-0 3

Thread-0 4

Thread-0 5

class mylicens extends Thread{  
 public void run()  
 {  
 try  
 {  
 System.*out*.println("Medical test");  
 Thread.*sleep*(2000);  
 System.*out*.println("Medical completed:");  
  
 }  
 catch (Exception e)  
 {  
 System.*out*.println(e);  
 }  
 }  
}  
class officer extends Thread  
{  
 public void run()  
 {  
 try{  
 System.*out*.println("Check the officer");  
 Thread.*sleep*(2000);  
 System.*out*.println("Verify Completed:");  
  
 }catch (Exception e)  
 {  
 System.*out*.println(e);  
 }  
 }  
}  
class Sign extends Thread{  
 public void run()  
 {  
 try{  
 System.*out*.println("Check the all paper of head officer");  
 Thread.*sleep*(2000);  
 System.*out*.println("Officer verify all process has done:");  
   
 }  
 catch (Exception e)  
 {  
 System.*out*.println(e);  
 }  
 }  
}  
public class License {  
 public static void main(String[] args) throws InterruptedException {  
 mylicens m = new mylicens();  
 m.start();  
 m.join();  
  
 officer o = new officer();  
 o.start();  
 o.join();  
  
 Sign s = new Sign();  
 s.start();  
 s.join();  
 }  
}

Output:

Medical test

Medical completed:

Check the officer

Verify Completed:

Check the all paper of head officer

Officer verify all process has done:

Diffrence between sleep(), yield() and join().

These all three method temprory stop the thread execution.

|  |  |  |
| --- | --- | --- |
| Sleep() | Yield() | Join() |
| If any thread does not want to perform any operation for particular time. | It stop the current executing thread & provide chance to another threads of same or higher priority to execute. | If a thread wants to wait for another thread to complete its task. |
| Ex-Timer,Ppt,Blinking bulbs | Ex-Shopping mall | Ex-License Department. |
| It execute automatically after provided time period.  If thread is interupted. | Automatically execute  By thread sheduler. | Automatically execute  After completion another task.  After completion of time period.If thread is interrupted. |
| There used two method:  1.Sleep(long milisecond)  2.Sleep(long milisecong,int nanosecond) | Only one method:  1.yield() | Three methods:  1.Join()  2.Join(longmilisecond)  3.join(long ms,int ns) |
| Method overloaded | No | Yes |
| Sleep() is static method | yes | No |
| Sleep() method is not final | No | Join method is final. |
| Sleep() method:  1.native  2.non-native | yes | No |
| Exception throw in sleep method.  (Interrupted Exception) | No | Yes(Interrupted Exception) |

Interrupting thread methods:

1.Interupted Method:

It is used to interrupt an executing method.

Interrupted() method is used to check whether a thread is interrupted or not.

Interrupt() method only works when the thread is in sleeping or waiting state(sleep() or wait()).

If a thread is not in sleeping or waiting state then calling or interrupt () method will perform normal behaviour.

When we use an interrupt() method then if throws InterruptedException.

Syntax:public void interrupt(){ }

Example:

class Inrterput extends Thread{  
  
 public void run() {  
 try {  
 for (int i = 0; i <= 5; i++) {  
 System.*out*.println(i);  
 Thread.*sleep*(1000);  
 }  
 } catch (Exception e) {  
 System.*out*.println(e);  
 }  
 }  
  
 public static void main(String[] args) {  
 Inrterput i = new Inrterput();  
 i.start();  
 i.interrupt();  
 }  
  
}

Output:

0

java.lang.InterruptedException: sleep interrupted

Interrupt method without sleep:

class Inrterput extends Thread{  
  
 public void run() {  
 try {  
 for (int i = 0; i <= 5; i++) {  
 System.*out*.println(i);  
 //Thread.sleep(1000);  
 }  
 } catch (Exception e) {  
 System.*out*.println(e);  
 }  
 }  
  
 public static void main(String[] args) {  
 Inrterput i = new Inrterput();  
 i.start();  
 i.interrupt();  
 }  
  
}

Output:

0

1

2

3

4

5

2. Isinterrupted method():

Interupted() method clearrs the interrupted status from true to false if thread is interrupted.

Interrupted() metod will change the reuslt is called twice.

Syntax:

Public static boolean interrupted(){ }

isInterrupted() method is used to check whether a thread is interrupted or not.

isInterrupted() method does not clear the interrupted status.

isInterrupted() metod will produce same reuslt is called twice.

Syntax:

Public boolean isinterrupted(){ }

class Inrterput extends Thread{  
  
 public void run() {  
 try {  
 for (int i = 0; i <= 5; i++) {  
 System.*out*.println(i);  
 Thread.*sleep*(1000);  
 }  
 } catch (Exception e) {  
 System.*out*.println(e+"Thread interrupted:");  
 }  
 }  
  
 public static void main(String[] args) {  
 Inrterput i = new Inrterput();  
 i.start();  
 i.interrupt();  
 }  
}

Output:

0

java.lang.InterruptedException: sleep interruptedThread interrupted:

class Inrterput extends Thread{  
  
 public void run()  
 {  
 System.*out*.println(Thread.*interrupted*());  
 try {  
 for (int i = 0; i <= 5; i++) {  
 System.*out*.println(i);  
 Thread.*sleep*(1000);  
 }  
 } catch (Exception e) {  
 System.*out*.println(e+"Thread interrupted:");  
 }  
 }  
  
 public static void main(String[] args) {  
 Inrterput i = new Inrterput();  
 i.start();  
 i.interrupt();  
 }  
}

Output:

true

0

1

2

3

4

5

class Inrterput extends Thread{  
  
 public void run()  
 {  
 System.*out*.println(Thread.*interrupted*());  
 System.*out*.println(Thread.*currentThread*().isInterrupted());  
 try {  
 for (int i = 0; i <= 5; i++) {  
 System.*out*.println(i);  
 Thread.*sleep*(1000);  
 }  
 } catch (Exception e) {  
 System.*out*.println(e+"Thread interrupted:");  
 }  
 }  
  
 public static void main(String[] args) {  
 Inrterput i = new Inrterput();  
 i.start();  
 i.interrupt();  
 }  
}

Output:

true

false

0

1

2

3

4

5

class Inrterput extends Thread{  
  
 public void run()  
 {  
 System.*out*.println(Thread.*interrupted*());  
 System.*out*.println(Thread.*currentThread*().isInterrupted());  
 try {  
 for (int i = 0; i <= 5; i++) {  
 System.*out*.println(i);  
 Thread.*sleep*(1000);  
 }  
 } catch (Exception e) {  
 System.*out*.println(e+"Thread interrupted:");  
 }  
 }  
  
 public static void main(String[] args) {  
 Inrterput i = new Inrterput();  
 i.start();  
 //i.interrupt();  
 }  
}

Output:

false

false

0

1

2

3

4

5

class Inrterput extends Thread{  
  
 public void run()  
 {  
 System.*out*.println("A:"+Thread.*interrupted*());  
 //System.out.println(Thread.currentThread().isInterrupted());  
 try {  
 for (int i = 0; i <= 5; i++) {  
 System.*out*.println(i);  
 Thread.*sleep*(1000);  
 System.*out*.println("B:"+Thread.*interrupted*());  
 }  
 } catch (Exception e) {  
 System.*out*.println(e+"Thread interrupted:");  
 }  
 }  
  
 public static void main(String[] args) {  
 Inrterput i = new Inrterput();  
 i.start();  
 i.interrupt();  
 }  
}

Output:

A:true

0

B:false

1

B:false

2

B:false

3

B:false

4

B:false

5

B:false

class Inrterput extends Thread{  
  
 public void run()  
 {  
 //System.out.println("A:"+Thread.interrupted());  
 System.*out*.println(Thread.*currentThread*().isInterrupted());  
 try {  
 for (int i = 0; i <= 5; i++) {  
 System.*out*.println(i);  
 Thread.*sleep*(1000);  
 // System.out.println("B:"+Thread.interrupted());  
 }  
 } catch (Exception e) {  
 System.*out*.println(e+"Thread interrupted:");  
 }  
 }  
  
 public static void main(String[] args) {  
 Inrterput i = new Inrterput();  
 i.start();  
 i.interrupt();  
 }  
}

Output:

0

java.lang.InterruptedException: sleep interruptedThread interrupted:

Synchronization:

In this processs control the access of multiple threads to any shared resources.

Without use synchronization:

class Booktheaterseats{  
 int totalseat =10;  
 void bookseat(int seats)  
 {  
  
  
  
 if (totalseat >= seats) {  
 System.*out*.println("Seat is booked" + " " + seats);  
 totalseat = totalseat - seats;  
 System.*out*.println("Available seat" + " " + totalseat);  
 } else {  
 System.*out*.println("Sorry seats cannot be booked:");  
 System.*out*.println("Left seats" + totalseat);  
 }  
  
 }  
}  
public class Movieticketbook extends Thread  
{  
 static Booktheaterseats *b*;  
 int seats;  
 public void run(){  
 *b*.bookseat(seats);  
 }  
 public static void main(String[] args) {  
 *b* = new Booktheaterseats();  
 Movieticketbook m = new Movieticketbook();  
 m.seats =8;  
 m.start();  
  
 Movieticketbook m1 = new Movieticketbook();  
 m1.seats = 6;  
 m1.start();  
 }  
}

Output:

Seat is booked 6

Available seat 4

Seat is booked 8

Available seat -4

Why use synchronization?

1.Data Inconsistency

2.Thread interference.

Multiple thread interfare with each other,around which there can be occur many problems like deadlock problem can also occur.

Advantage:

1.No data inconsisteny problem

2.No thread interference.

Disadvantage:

1.Increase the waiting time period of threads.

2.Create performance problem.

Types of Synchronization

* Process Synchronization

Not present in java multithreading.

* Thread Synchronization

Thread Synchronization has two types:

(1)Mutual Exclusive

By Using Synchronized Method

By Using Synchronized Block

By Using Static Synchronization

(2)Cooperation

wait()

notify()

notifyAll()

1.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.

Interal working:

1.All object everyone has a lock.

2.All object inside two area non-synchronized and synchronized.

Inside synchronized area to consider synchroniozed method,block.

Any thread accesesd synchronize method one by one by one, if first thread completed task another thread are waiting ,if complete the task of first thread then next thread go to synchonized area are start the working.

class Booktheaterseats{  
 int totalseat =10;  
 synchronized void bookseat(int seats)  
 {  
  
  
  
 if (totalseat >= seats) {  
 System.*out*.println("Seat is booked:" + " " + seats);  
 totalseat = totalseat - seats;  
 System.*out*.println("Available seat:" + " " + totalseat);  
 } else {  
 System.*out*.println("Sorry seats cannot be booked:");  
 System.*out*.println("Left seats:" +" "+ totalseat);  
 }  
  
 }  
}  
public class Movieticketbook extends Thread  
{  
 static Booktheaterseats *b*;  
 int seats;  
 public void run(){  
 *b*.bookseat(seats);  
 }  
 public static void main(String[] args) {  
 *b* = new Booktheaterseats();  
 Movieticketbook m = new Movieticketbook();  
 m.seats =8;  
 m.start();  
  
 Movieticketbook m1 = new Movieticketbook();  
 m1.seats = 6;  
 m1.start();  
 }  
}

Output:

Seat is booked: 8

Available seat: 2

Sorry seats cannot be booked:

Left seats: 2

Synchronized Block:

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

Suppose we have 50 lines of code in our method, but we want to synchronize only 5 lines, in such cases, we can use synchronized block.

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

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

Syntax

**synchronized** (object reference expression) {

//code block

}

class Table  
{  
 void printTable( int n)  
 {  
 synchronized (this) {  
 try {  
 for (int i = 1; i <= 5; i++) {  
 System.*out*.println(n \* i);  
 Thread.*sleep*(1000);  
 }  
 } catch (Exception e) {  
 System.*out*.println(e);  
 }  
 }  
 }  
}  
  
class Mythreads1 extends Thread{  
 Table t;  
 Mythreads1(Table t)  
 {  
 this.t=t;  
 }  
 public void run(){  
 t.printTable(5);  
 }  
}  
class Mythreads2 extends Thread{  
 Table t;  
 Mythreads2(Table t)  
 {  
 this.t=t;  
 }  
  
 @Override  
 public void run() {  
 t.printTable(100);  
 }  
}  
public class SynchroBlock {  
 public static void main(String[] args) {  
 Table t = new Table();  
 Mythreads1 m = new Mythreads1(t);  
 m.start();  
 Mythreads2 m1 = new Mythreads2(t);  
 m1.start();  
 }  
}

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. We don't want interference between t1 and t3 or t2 and t4. Static synchronization solves this problem.

Example:

class Bookthreaterapp{  
 int Totalseat = 20;  
 synchronized void bookSeats(int seats)  
 {  
 if(Totalseat>=seats)  
 {  
 System.*out*.println("Seat booked successfully");  
 Totalseat = Totalseat-seats;  
 System.*out*.println("Seat left"+" "+Totalseat);  
 }else{  
 System.*out*.println("Seats cannot be booked");  
 System.*out*.println("Seats left:"+Totalseat);  
 }  
 }  
}  
class Myseats extends Thread{  
 Bookthreaterapp b;  
 int seats;  
 Myseats(Bookthreaterapp b,int seats){  
 this.b = b;  
 this.seats=seats;  
  
 }  
 public void run()  
 {  
 b.bookSeats(seats);  
 }  
}  
class Myseats2 extends Thread{  
 Bookthreaterapp b;  
 int seats;  
 Myseats2(Bookthreaterapp b,int seats)  
 {  
 this.b=b;  
 this.seats=seats;  
 }  
 public void run()  
 {  
 b.bookSeats(seats);  
 }  
}  
public class Twoobj {  
 public static void main(String[] args) {  
 Bookthreaterapp b1 = new Bookthreaterapp();  
 Myseats m1 = new Myseats(b1,8);  
 m1.start();  
  
 Myseats2 m2 = new Myseats2(b1,6);  
 m2.start();  
  
  
 Bookthreaterapp b2 = new Bookthreaterapp();  
 Myseats m3 = new Myseats(b2,4);  
 m3.start();  
  
 Myseats2 m4 = new Myseats2(b2,6);  
 m4.start();  
 }  
}

Output:

Seat booked successfully

Seat booked successfully

Seat left 12

Seat left 16

Seat booked successfully

Seat left 6

Seat booked successfully

Seat left 10

Example using static synchronization:

class Bookthreaterapp{  
 static int *Totalseat* = 20;  
 static synchronized void bookSeats(int seats)  
 {  
 if(*Totalseat*>=seats)  
 {  
 System.*out*.println(seats+" "+"Seat booked successfully");  
 *Totalseat* = *Totalseat*-seats;  
 System.*out*.println("Seat left"+" "+*Totalseat*);  
 }else{  
 System.*out*.println("Seats cannot be booked");  
 System.*out*.println("Seats left:"+*Totalseat*);  
 }  
 }  
}  
class Myseats extends Thread{  
 Bookthreaterapp b;  
 int seats;  
 Myseats(Bookthreaterapp b,int seats){  
 this.b = b;  
 this.seats=seats;  
  
 }  
 public void run()  
 {  
 b.*bookSeats*(seats);  
 }  
}  
class Myseats2 extends Thread{  
 Bookthreaterapp b;  
 int seats;  
 Myseats2(Bookthreaterapp b,int seats)  
 {  
 this.b=b;  
 this.seats=seats;  
 }  
 public void run()  
 {  
 b.*bookSeats*(seats);  
 }  
}  
public class Twoobj {  
 public static void main(String[] args) {  
 Bookthreaterapp b1 = new Bookthreaterapp();  
 Myseats m1 = new Myseats(b1,8);  
 m1.start();  
  
 Myseats2 m2 = new Myseats2(b1,6);  
 m2.start();  
  
  
 Bookthreaterapp b2 = new Bookthreaterapp();  
 Myseats m3 = new Myseats(b2,4);  
 m3.start();  
  
 Myseats2 m4 = new Myseats2(b2,6);  
 m4.start();  
 }  
}

Output:

8 Seat booked successfully

Seat left 12

6 Seat booked successfully

Seat left 6

6 Seat booked successfully

Seat left 0

Seats cannot be booked

Seats left:0

Inter Thread Communication:

Object class method not thread class method:

Inter-thread communication is a mechanism in which a thread releases the lock and enter to paused state and another thread acquires the lock and continue to executed.

1.Wait()

If any threads calls the wait() method, The wait method causes the current thread to wait until another thread invokes the notify or notifyAll method on the same object.

2.notify()

This method is used to wake up a single thread and release the object lock.

3.This method is used to wake up all threads that are in waiting state.

Example:

class Totalearning extends Thread{  
 int total=0;  
 public void run()  
 {  
 synchronized (this) {  
 for (int i = 1; i <= 10; i++) {  
 total = total + 100;  
 }  
 this.notify();  
 }  
 }  
}  
public class Bookticket {  
 public static void main(String[] args) {  
 Totalearning t = new Totalearning();  
 t.start();  
 synchronized (t) {  
 try {  
 t.wait();  
 System.*out*.println("Total earning : " + t.total + " " + "Rs");  
 }catch (Exception e)  
 {  
 System.*out*.println(e);  
 }  
 }  
 }  
}

Output:

Total earning : 1000 Rs

Without wait()and Notify()

class Totalearning extends Thread{  
 int total=0;  
 public void run()  
 {  
  
 for (int i = 1; i <= 10; i++) {  
 total = total + 100;  
 }  
  
  
 }  
}  
public class Bookticket {  
 public static void main(String[] args) {  
 Totalearning t = new Totalearning();  
 t.start();  
 System.*out*.println("Total earning : " + t.total + " " + "Rs");  
 }  
}

Output:

Total earning : 0 Rs

Note:

To call wait(),Notify() or notifyAll() method of any Object,thread should own the lock of the object i.e the thread should be inside synchronized area.

Method vs Block Synchronization

Features

Method Synchronization

Block Synchronization

Scope

It synchronizes the entire method.

Synchronizes only block code.

Performance

This may lead to unnecessary synchronization for non-critical sections.

More efficiently, synchronizes only critical sections.

Lock

Locks the method object.

Locks the object or class specified in the block.

Flexibility

Less flexible, synchronize the entire method

More flexible, allows partial synchronization