

```

t1.setName ("first person");
t2.setName ("Second person");
t3.setName ("Third person");

t1.start();
t2.start();
t3.start();
}
}

```

### Output:

available tickets = 1

1 berth reserved for first person

available tickets = 0

some berths are not available

available tickets = 0

some berths are not available

### Problem:

With only single object all persons

get message that available tickets are 1

only 1 ticket cannot be allocated for

3 persons.

Solution. To encounter this problem, we use the concept synchronization.

Synchronization. (locking of an object).

When multiple threads are acting on single object.

2 Types:

1) synchronized method

2) synchronized block.

→ Using synchronized method:

```
synchronized public void run {  
    }  
}
```

→ Using synchronized block,

```
public void run() {  
    synchronized (this) {  
        }  
}
```

```
Thread t1 = new Thread (obj1);
```

```
Thread t2 = new Thread (obj2);
```

```
Thread t3 = new Thread (obj3);
```

```
t1.start();
```

```
t2.start();
```

```
t3.start();
```

Q

Q

In booking tickets, there is only one ticket available. That ticket is wanted by 3 persons. If any person is booked the ticket then remaining should get the message not available.

Sol:

```
class Reservation extends Thread {
```

```
    int avail = 1;
```

```
    int wanted;
```

```
    public Reservation (int i) {
```

```
        wanted = i;
```

```
    }
```

synchronized

```
    public void run() {
```

```
        Syso ("available tickets: " + avail);
```

```
if (avail >= wanted) {
```

```
String name = Thread.currentThread().  
getname();
```

```
System.out.println(wanted + " berth reserved for  
+ name);
```

```
try {
```

```
Thread.sleep(1500);
```

```
avail = avail - wanted;
```

```
}  
catch (InterruptedException e) {
```

```
e.printStackTrace();
```

```
}  
else {
```

```
System.out.println("sorry berths are not  
available");
```

```
}  
}
```

```
class Singleobject {
```

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

```
Reservation obj = new Reservation(1);
```

```
Thread t1 = new Thread(first person obj);
```

```
Thread t2 = new Thread(obj);
```

```
Thread t3 = new Thread(obj);
```

Q, all tasks at once for multi persons

```
class Main Theater implements Runnable {
```

```
    String str;
```

```
    public Theater (String str) {
```

```
        this.str = str;
```

```
    }
```

```
    public void run () {
```

```
        for (int i=1; i<=10; i++) {
```

```
            System.out.println(str + " : " + i);
```

```
        }
```

```
    }
```

```
        Thread.sleep(1500);
```

```
    }
```

```
    catch (InterruptedException e) {
```

```
        e.printStackTrace(); // or e.printStackTrace();
```

```
    }
```

```
}
```

```
}
```

```
class Multitasking {
```

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

```
        Thread
```

```
        Theater obj1 = new Theater ("issueTicket");
```

```
        Theater obj2 = new Theater ("verifyTicket");
```

```
        Theater obj3 = new Theater ("allocateSeat");
```



t1.setPriority(1);

t2.setPriority(5);

t3.setPriority(9);

t1.start();

t2.start();

t3.start();

### → Methods of Thread Groups:

1. To create thread group

ThreadGroup tg1 = new ThreadGroup();

2. How to add a thread to thread group tg1:

Thread t1 = new Thread();

or

Thread t1 = new Thread(tg1, <sup>obj</sup>res, "first name");

"first thread"

Group: - ThreadGroup tg = new ThreadGroup();

-threads: - Thread t1 = new Thread(tg, res, "thread1");

3. To find the parent thread group name

tg.getParent();

4. To set priority to thread group (max)

tg.setMaxPriority();

5. To set least priority

tg.setMinPriority();

6. Get thread group name

t1.getThreadGroupName();

## → Methods of Thread class:

1. Creation of thread

```
Thread t1 = new Thread();
```

```
Thread t1 = new Thread(obj);
```

2. Naming of thread

```
t1.setName("name");
```

```
Thread t1 = new Thread(obj, "name of thread");
```

3. How to find current thread performs in.

```
Thread t = Thread.currentThread();
```

4. To stop execution of current thread.

```
t1.sleep(milliseconds);
```

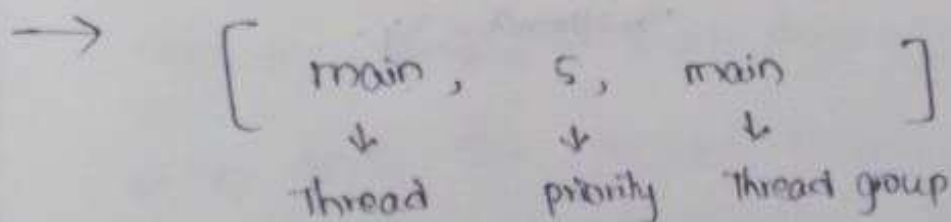
```
t1.wait(milliseconds);
```

5. To pause your execution

```
t1.yield(milliseconds);
```

6. How to get name

```
String name = t.getName();
```



## → Thread priorities methods:

1. Set priority to thread

```
t1.setPriority(int prio);
```

2. get priority

```
Integer priority-num = t1.getPriority();
```

3. To test whether thread is alive

boolean b = t3.isAlive(); true/false

∴ active :- certain thread is active or not.

Ex:

find current thread, priority number from 100 numbers by using `getPriority()`.

MyThread

class MyThread implements Runnable {

int count=1;

public void run() {

for (int i=1; i<=1000; i++) {

System.out.println("completed thread is: " + Thread.currentThread().getName());

System.out.println("priority number is: " + Thread.currentThread().getPriority());

}

}

}

class Demo {

public static void main (String args[]) {

Thread t1 = new Thread (obj, "first");

Thread t2 = new Thread (obj, "second");

Thread t3 = new Thread (obj, "third");