```
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
  sorry berths are not available
 available tickets =0
  Sorry 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→ 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  Reservaton (int i) {
            wanted=i;
     }
synchronized
   Public void run(){
      Syso("available tickets: + avail);
```

```
if (avail >= wanted){
  String name = Thread.currentThread().
                              getName();
  Sysout.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( [illegible] obj)
     Thread t2= new Thread (obj);
      Thread t3= new Thread (obj);
```

Q→ all tasks at once for multi persons

```
class ~~Multi~~ Theater implements Runnable {
        private
          String str;
        Public Theater (String str){
            this. str = str;
        }
    public void run (){
          for(int i=1; i<=20; i++){
           syso (str+":"+i);
          }
    try {
       Thread.sleep(1500);
      }
    catch(InterruptedException e) {
         e.printStackTrace(); // or s.op.(e);
      }
  }
}}
 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();
```

→ <u>Methods of Thread Groups:</u>

1. To create thread group

```
Thread Group tg1=new Thread Group();
```

2. How to add t1 thread to thread group tg1:

```
Thread t1=new Thread();
        ⇓
                            obj
Thread t1=new Thread(tg1, 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();
```

→ <u>Methods of Thread class</u> :

1. Creation of thread

   Thread t_1 = new Thread();

   Thread t_1 = new Thread(obj);

2. Naming of thread

   t_1.setName("name");

   Thread t_1=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.

   t_1.sleep(milliseconds);

   t_1.wait (milliseconds);

5. To pause your execution

   t_1.yield (milliseconds);

6. How to get name

   String name = t.getName();

→
$$\begin{array}{ccc} [\ \text{main}, & 5, & \text{main}\ ] \\ \downarrow & \downarrow & \downarrow \\ \text{thread} & \text{priority} & \text{thread group} \end{array}$$

→ <u>Thread priorities methods</u>:

1. Set priority to thread

   t_1.setPriority(int prio);

2. get priority

   Integer priority-num = t_1.getPriority();

3. To test whether thread is alive

boolean b3= t3.isAlive(); true/false

∵ alive :- certain thread is active or not.

Ex:

Find current thread, prioritynumber from 1000 numbers by using getPriority().

MyThread.

```
class Mythread implements Runnable {
        int count=1;
        public void run(){
      for (int i=1;i<=1000;i++){
        syso(" completed thread is: "+ Thread.currentT
                                 -read().getName());
      syso(" priority number is: " + Thread.currentthread
                                 .getPriority());
        }
    }
}

class Demo{
      psvm(){
    Thread t1= new Thread (obj, "first");
    Thread t2= new Thread (obj, "second");
    Thread t3= new Thread (obj, "third");
```