Custom Thread class

```
class CustomThread extends Thread
 {
 Shared s;

 public CustomThread (Shared s, String str)
  {
  super(str);
  this.s=s;
  start();
 }

 public void run()
 {
 s.show (Thread.currentThread().getName(), 10);
 // Sop ("Thread1 Sum of  10, 20 = "+s.add(10,20));

 }
}
```

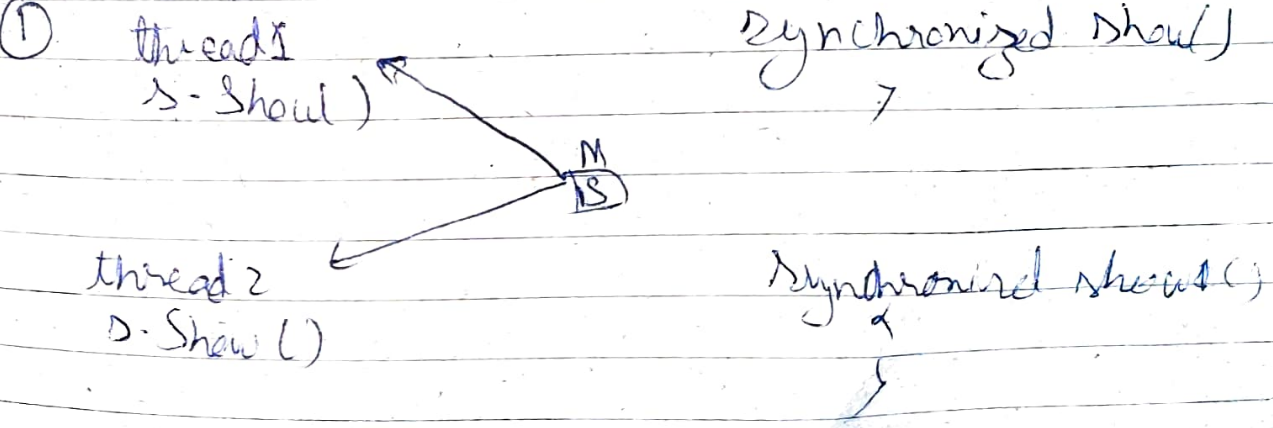
```
class CustomThread1 extend Thread
  {
  Shared s;

  public CustomThread1(Shared s, String str)
    {
     super(str);
     this.s=s;
     start();
    }

 public void run()
   {
    s.show(Thread.CurrentThread().getName(),20);
//Sop ("Thread 2 Sum of 100, 200 -"+s.add
                                  (100, 200.));

 }
 };

clas customThread2 extend Thred
 {
  Shareed s;

  public custamThread2(Shared s, String str)
  {
   super(str);
   this.s=s;
   start();
   }
~~void~~ public void run()
{
  s.show(Thread.currentThread().getName(),30);
//sop("Thread3 sum of 1000,2000 ="+s.add(1000,2000));
  }
}
```

<u>RunSync.class</u>

```
class RunSync
{
  psvm()
{
 Shared st = new Shared();
CustomThread t1 = new CustomThread(st, "one");
Custom Thread1 t2 = new CustomThread1(st, "two");
Custom Thread2 t3 = new customThread2(st, "three");
}
```

<u>Interview Question on above implement^n of Sync</u>

①



• StringBuffer is a Synchronised class or Thread-
StringBuilder " " non- " " class.

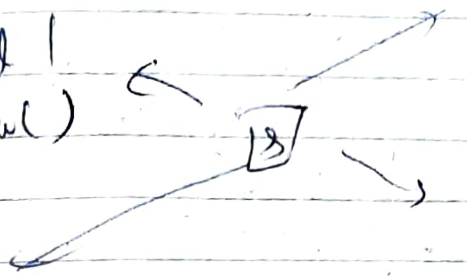
• If a class is synchronized that means at a time only one thread can access that the object of that class if all the threads are having same object of that class.

How to make our own class Synchronised

If you want to make your own class syn-
-chronized then make all the methods of that class Synchronized.

Condition-2

thread 1
s.show()



show

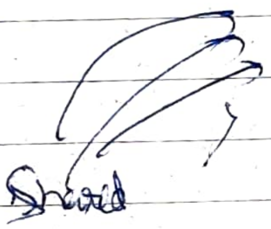
thread 2
s.show1() → show1()

• Both can run at sametime.

~~Imp~~ Condition-3

thread1
Shared.show()

static synchronised show()
{



thread2
Shared.show()

~~Void s~~

}

thread 3
Shared.show()

• ~~Only one Executes a time~~ [illegible]

• One by One

Rule

In Java every class is also having a One implicit ~~ob~~ lock on it.

~~Com~~

Condition 04

thread1
Shared.show()
↳ class Name

static synchronised show()

thread2
s.show()
↳ object

- One by One entry

Rule

In case of static Synchronised method, lock is always achived on a

class,
Hardly . matters, whether it is called by
the class ~~MO~~ name or by the
object

Condition-5

• thread 1 → static synchronised show()
~~st~~ Shared.show()
{
}

Thread 2 → ~~sta~~ synchronized void show()
s.Show()
{
}

• Both execute together.

Anather Method to Achive Synchronizaten
(OR by
Synchronized Block

→

```
synchronized Vaid show)
    {
      ≡
      ≡
      ≡
    }
```

large no. of lines

```
~~synchronized~~ Vaid show()
   {
     ≡
T1                          T2
   synchronized( t )
   {
     ≡
   }
```

↓ object of any class

Diff b/w sync method & block.

Sync. Method	Sync block
① In case of synced method, we make whole method synchronized.	① In case of synced block, we make particular portion of the method synced rather than whole method.
~~① Lock can be achived only on object who~~ on	~~② Lock can be achived on any object~~
② In case of synchronized method, lock is always achived on a current object.	② In case of sync block, lock can be achived on any object.
	③ Sync block is also used to make object of any class sync.

```
void show()
{
  ===
  t = new Temp();
synchronised(t)
  {
  t.display();
  }
  ==
}
```

```
class Temp
{
void display()
 {
  ___
 }
}
```

Here we have made display() sync.. via sync. block.

Program (insert in Shared-class)

```
void show 2(String s, int a)
{
~~Synchr~~
 sop("starting in method" +s);
 synchronized (this)
 {
  x=a;
 System.out.println("starting in block"+
                  + s+" "+ x);

 try
 {
 Thread.sleep(2000);
 }
 catch(Exceptione){}
  Sop("exit from block"+ s+"  "+x);
 }
 }

Temp t =new Temp();
Void show 3(String s, int a)
{
Sop("starting in method"+s);
synchronized (t)
{
t.dilTohPagalHai(s);
}
}
```

```
class Temp
  {
  void dilToPagalHai(String s)
 {
 Sop("starting haa mai hoon in
        dilTohPagalHai+" "+s);

  try
  {
 Thread.sleep(2000);
 }
 catch(Exception e){}
 Sop("ending from dilTohPagalHai"+" "+
                                s);

 }
 }
```

suspend() & ~~R~~ resume() {~~To be~~ Depricated}

- If we use suspend() method on a running thread it goes to pool for a infinite time.

Why suspend() method has been depricated

Whenever we call suspend() method on any thread from the synchronized method or synchronized block then one dead-lock will be created.

Program: (~~To be~~ Add in Shared.class)

```
synchronized void show(String s, int a)
{
x=a;
System.out.println("starting in method"
                     +s+"  "+x);
Thread.currentThread().suspend();
                      .resume();
 Sop("exit from method"+s+"  "+x);
}
}
```

Now,
We have alternate for Suspend() 4 resume
ie. wait() 4 notify()

• wait() in addition to ~~remove~~ releases thread from processer cycle also removes lock from

• Suspend() method only releases a process-or cycle from any thread

• wait method releases both processor cycle and a lock from any thread.

wait() / notify() / notifyAll {methods of Object class}

Program ~~New~~ (Shared.java)

```
class Shared
{
int flag=0;
int data=0;
synchronized public void submit()
{
 flag=1
 try
 {
 Thread.sleep(1000);
 }
 catch (Excepn e){}
```

```
data = 10;
sop("value submitted;
  ~~[illegible]~~ notify();
}

synchronized int withdraw()
  {
  if(flag==0)
  {
  try
  {
  Sop ("sending into wait block");
   wait();
  }

 catch(Exception e){}
 }
 return data;
 }
}

Thread1.java

class    Thread1 extends Thread
{
Shared s;
Thread1(Shared s, String str)
{
Super (str);
this.s = s;
```

```
    start();
  }

  public void run()
  {
  Sop (s.withdraw());
  }
  }

class Thread2 extends Thread
   {
   Shared s;
   Thread2(Shared s, String str)
    {
     super(str);
     this.s=s;
     start();
    }

  public void run()
   {
   s.submit();
  }
```

RunSync.java

```
class RunSync
 {
 psvm (String s[])
  {
  Shared st = new Shared();

Thread1 t1 = new Thread1(st, "one");
 Thread2 t2 = new Thread2(st, "two");

}
}
```

Deadlock

Copy from previous class

Another Type of Deadlock

T1

```
synchronized(o1)
 {
 Thread.sleep(1000);
  synchronized(o2)
  {
  }
 }
```

T2

```
synchronized(o2)
 {
 Thread.sleep(1000);
synchronized(o1)
{
}
}
```

Deadlock