Inter thread communication

Two threads can communicate with each other by using
- wait()
- notify()
- notifyAll()

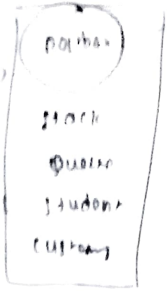
- The thread which is expecting updation is responsible to call wait() method then immediately the thread will enters into waiting state

- The thread which is responsible to perform updation, after performing updation it is responsible to call notify method then waiting thread will get that notification and continue its execution with the updated items

- wait, notify, notifyAll methods present in object class

Why: Thread can call (wait) (notify) - (notifyAll) method on any java object

for every object a method is required
∴ it must be object class and not in thread class



wait, notify, notifyAll() methods present in object class but not in thread class because thread can call these methods on any java object

t1 -> lock(obj)

obj.wait()

To call wait, notify, notifyAll methods on any object, thread should be owner of that object i.e thread should has lock of that object

i.e thread should be inside synchronized area

We can call wait, notify, notifyAll only from synchronized area

Other wise we will get runtime exception saying

IllegalMonitorStateException

- If a thread calls wait method on any object it immediately releases the lock of that particular object and enters into waiting state.

- If a thread calls notify method on any object it releases the lock of that object but may not immediately
- except

wait, notify, notifyAll

there is no other method where thread releases the lock

method	if thread releases Lock?
yield()	NO
join()	NO
sleep()	NO
wait()	yes
notify()	yes
notifyAll()	yes

Which of the following is valid ?

① If a thread calls wait method immediately it enters into waiting state without releasing any lock ✗

② If a thread calls wait method it releases the lock of that object but may not immediately ✗

③ If a thread calls wait method on any object it releases all locks acquired by that thread and immediately enters in to waiting state ✗

④ If a thread calls wait method on any object it immediately releases the lock of that particular object and enters into waiting state ✓

⑤ If a thread calls notify method on any object it immediately releases the lock of that particular object ✗

⑥ If a thread call notify method on any object it releases the lock of that object but may not immediately ✓

public final void wait() throws InterruptedException

public final native void wait(long millisecond) throws IE

public final void wait(long ms, int ns) throws IE

public final native void notify()

public final native void notifyAll()

Note: Every wait method throws InterruptedException which is checked exception

hence when ever we are using wait method compulsory we should handle this interrupted exception

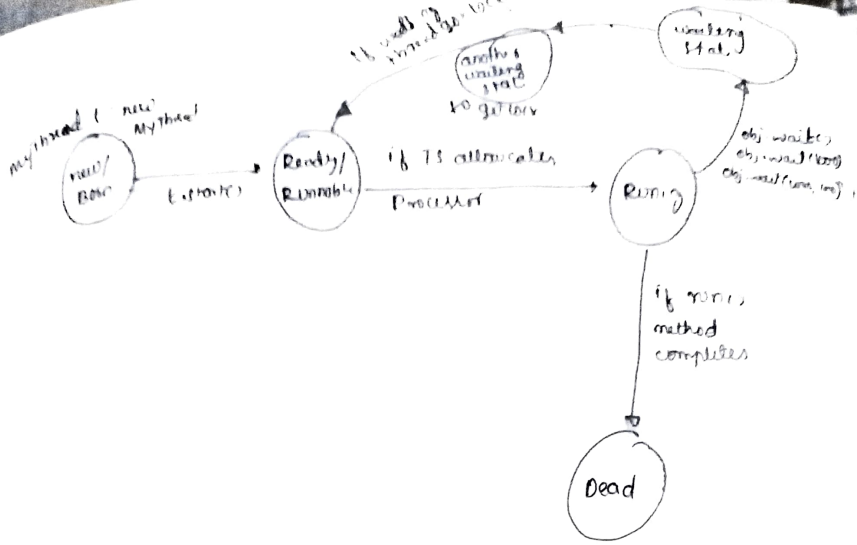
either by

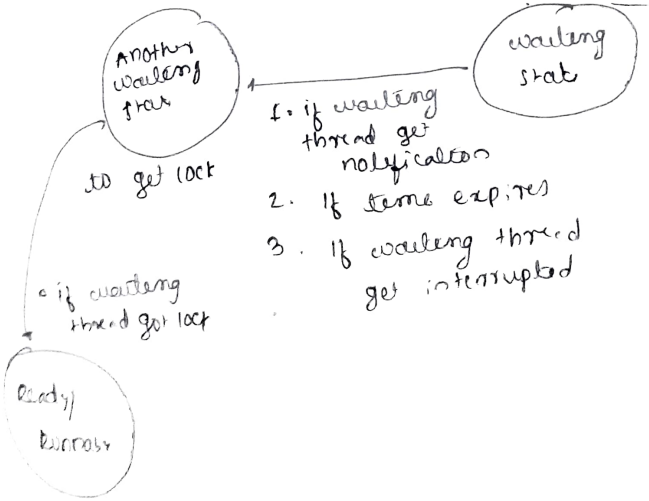
try catch

or by

throws keyword

Otherwise we will get compile time error


MyThread t = new MyThread();
new/ Born
t.start()
Ready/ Runnable
if TS allocates Processor
Running
if run() method completes
Dead
obj.wait()
obj.wait(long)
obj.wait(long, int)
waiting stat
another waiting stat
to get lock


Another waiting thread
waiting stat
1. if waiting thread get notification
2. If time expires
3. If waiting thread get interrupted
to get lock
4. if waiting thread got lock
Ready/ Runnable

```
class ThreadA {                                  // throws InterruptedException
    public static void main(String[] args) {
        ThreadB b = new ThreadB();
        b.start();                               // Thread.sleep(10000);
        System.out.println(b.total);
    }
}

class ThreadB extends Thread {
    int total = 0;
    public void run() {
        for (int i = 1; i <= 100; i++) {
            total = total + i;
        }
    }
}
```

0

5050

$0 \le x$

$x <= 10$

$\frac{n * (n+1)}{2}$

$\frac{100 * (101)}{2}$

50 50

```
threadA {
    psvm(String[] args) throws IE {
        threadB b = new threadB()
        b.start()
        b.join();          ← b.wait()
        sop(b.total);
    }
}

class ThreadB extends Thread {
    int total=0,
    public void run() {
        for(int i=1; i<=100; i++){
            total = total+i,
        }                  ← this.notify();
        // 1 crore lines of code
    }
}
```

IllegalMonitorStateException

wait, notify, notifyAll

Should only be called in synchronized Area

```
class ThreadOne {
    psvm(String[] args) throws InterruptedException {
        ThreadTwo t = new ThreadTwo();
        t.start();
        Synchronized(t){
(1)         SOP("main thread trying to call
                wait method");
            t.wait();
(4)         SOP("main thread got notification");
(5)         SOP(t.total);
        }
    }
}

class ThreadTwo extends Thread {
    int total = 0;
    public void run() {
        Synchronized (this) {
(2)         SOP("child thread starts
                calculation");
            for(int i=0; i<=100; i++){
                total = total+i;
            }
(3)         SOP("child thread trying to give
                notification");
            this.notify();
        }
    }
}
```

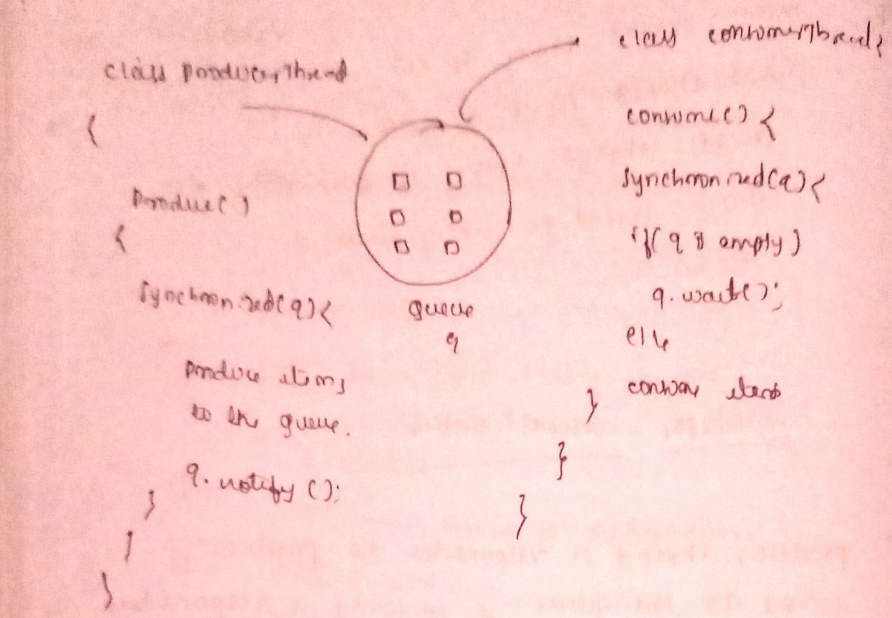
o/p: main Thread trying to call wait method
child thread start calculation
child thread trying to give notification
main thread got notification
5050

Producer - consumer problem

- producer thread is responsible to produce items to the queue & consumer is responsible to consume items from the queue

- If queue is empty then consumer thread will call wait method and enters into waiting state.

After producing item to the queue, producer thread is responsible to call notify method then waiting consumer will get the notification, and continue its execution with updated items



- Diff notify v/s notifyAll

- We can use notify() to give notification to for only one waiting thread

If multiple threads are waiting then only one thread will be notified and remaining threads have to wait for further notifications

- which thread will be notified we cannot exactly. that depends on JVM

we can use notify/all to give the notify calls for all waiting threads of a particular object

now only even though multiple threads notified but execution starts the performed because

threads require lock and only one lock is available

```
Stock S1 = new Stock();
Stock S2 = new Stock();
```

```
Synchronized(S1){
    S2.wait();
}
```

```
Synchronized(S1){
    S1.wait()
}
```

then it :
RuntimeException
IllegalMonitorStateException

valid

For which object we are calling wait method thread required lock of that particular object

if we are calling wait method on S1 then we have to call ~~wait method~~ lock on S1 object but not on S2 object