Java Programming (TCS408)

Module 3

Multithreading

Inter Thread Communication

- ❑ Inter-thread communication or Co-operation allow synchronized threads to communicate with each other.

- ❑ It is a mechanism in which a thread is paused running in its critical section and another thread is allowed to enter (or lock) in the same critical section to be executed.

- ❑ Methods used are:

- wait()
- notify()
- notifyAll()

Need for Inter Thread Communication

- Pooling is usually implemented by loop to check some condition repeatedly.

- Once condition is true appropriate action is taken. This waste CPU time.

- To avoid pooling Inter Thread Communication is used.

- For example: In e-commerce website: It happens sometimes that we like a product so much which is out of scope. We waste our effort and resources to check that item again and again that it is back in stock or not.

- In place of checking again and again, we can use notify me method so that we will get the notification on my email id when the product will be back in stock.

Inter Thread Communication

- ❑ These method are implemented as **final** methods in Object, so that all classes have them.

- ❑ All the three method can be called only from within a **synchronized** context.

wait( ) : Caused current thread to release lock and wait till specified time elapsed or another thread calls notify() or notifyAll() method.

Syntax: public void wait() throws InterruptedException

Should be written inside try block.

notify( ) : wakes up a single thread that called wait() on same object.

Syntax: public void notify( )

notifyAll( ): wakes up all the thread that called wait() on same object.

Syntax: public void notifyAll( )

Inter Thread Communication Example

Producer-Consumer Problem

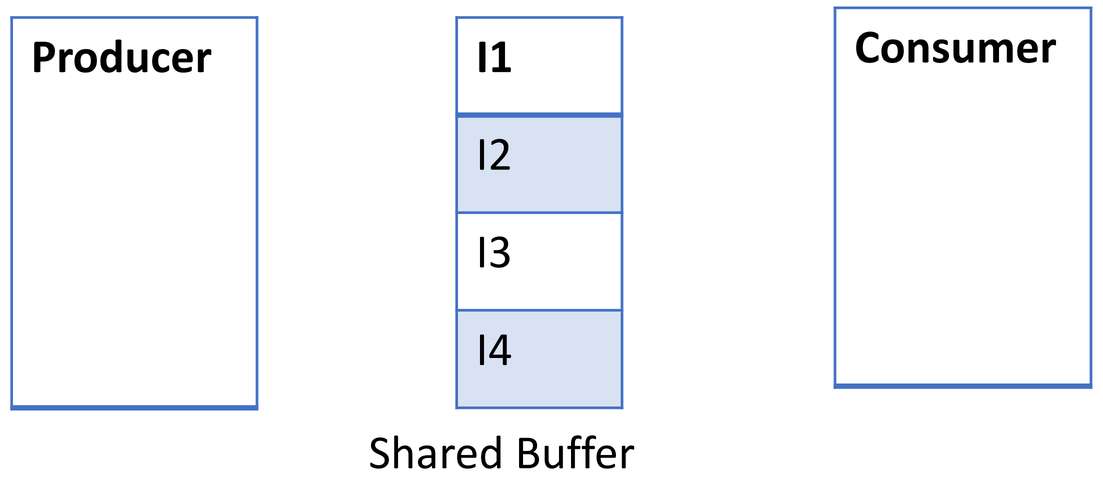
- Common problem in multiprocessor synchronized system.
- If two different processes(threads) are there(producer and consumer).
- Job of producer is to produce the items and put these items in buffer pool(common shared memory).
- Job of consumer is consumer items from buffer memory.
- Producer can’t add items in buffer if it is full and consumer can’t consume items if buffer is empty.
- If there is no communication between consumer and producer then such problems may arise:

Inter Thread Communication Example

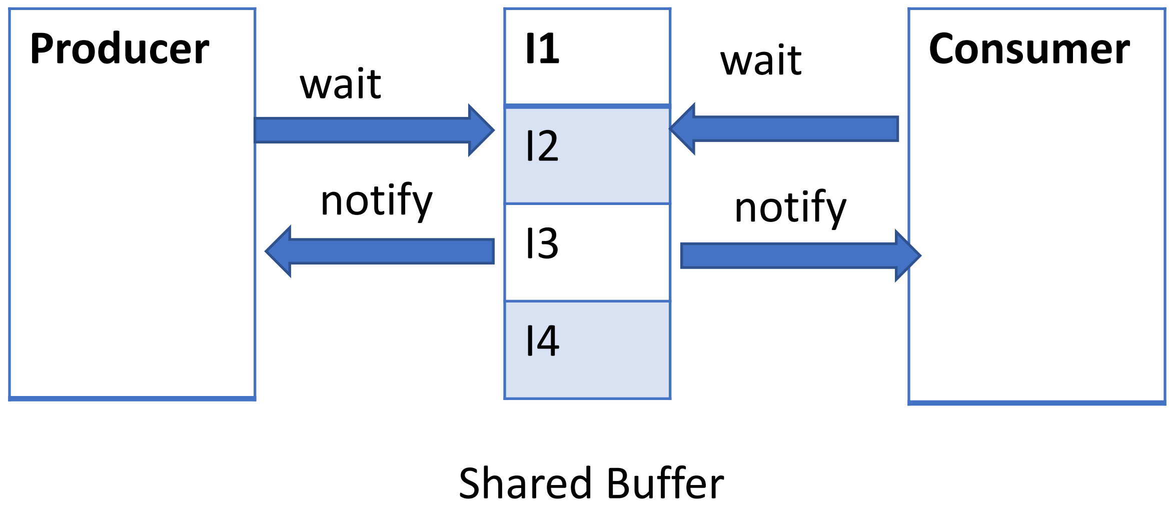
- As consumer will consume items, situation may arise when buffer will empty and consumer does not know about it, because this buffer is shred with producer also and it may be possible that producer is adding items in buffer .

- Similarly, it may also be possible that buffer is full and still producer is trying to add items in buffer pool.

- In this case, communication is required between producer and consumer.



Inter Thread Communication Example



```
class Customer
{
int amount=10000;
  synchronized void withdraw(int money)
  {
System.out.println(" withdraw");

if(amount<money){
System.out.println("Less balance.Waiting for deposit");
try
{
   wait();

}
catch(Exception e){}
}
amount=amount-money;
System.out.println("withdraw completed");
System.out.println("updated balance="+amount);
}
```

```
synchronized void deposit(int money){
System.out.println("Deposit");
amount=amount+money;
System.out.println("deposit completed");
notify();
}
}

class T1 extends Thread
{
 Customer c;
T1(Customer c)
{
this.c=c;
}

   public void run()
{

   c.withdraw(15000);

}
}
```

```
class T2 extends Thread
{
 Customer c;
T2(Customer c)
{
this.c=c;
}
public void run()
{
c.deposit(10000);
}
}
class Main{
public static void main(String args[]){
 Customer c=new Customer();
T1 obj1=new T1(c);
T2 obj2=new T2(c);
obj1.start();
obj2.start();
}}
```

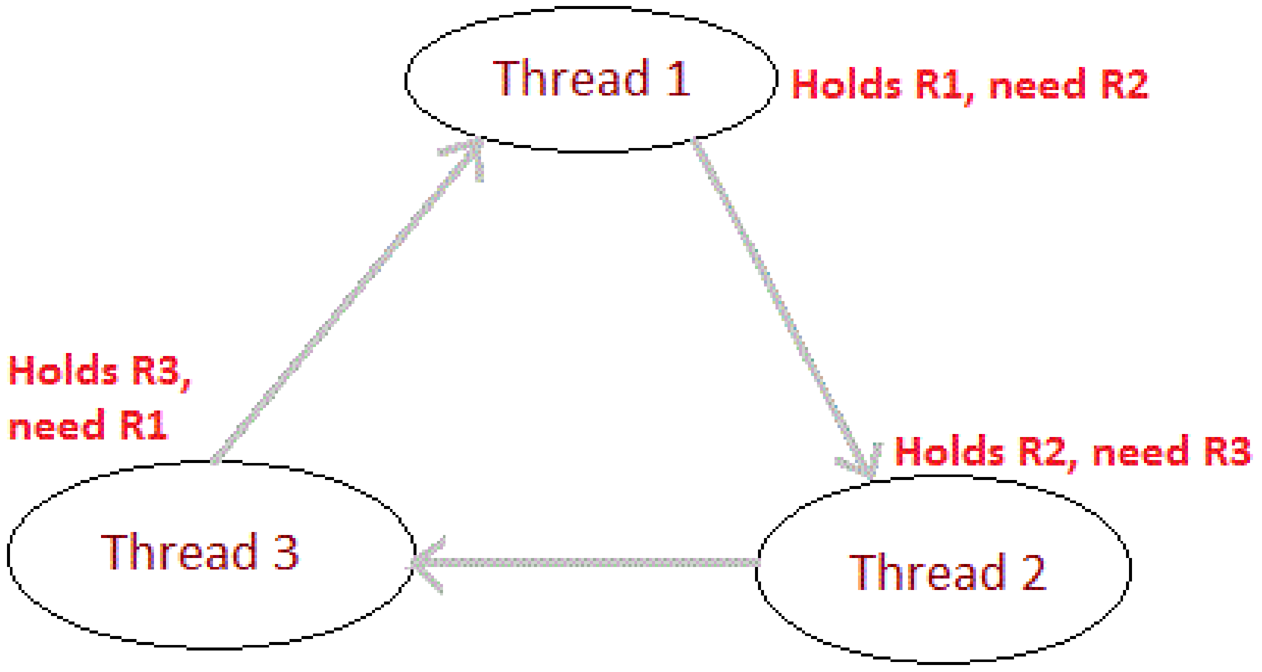
wait()	**sleep()**
called from synchronised block	no such requirement
monitor is released	monitor is not released
gets awake when notify() or notifyAll() method is called.	does not get awake when notify() or notifyAll() method is called
not a static method	static method
wait() is generaly used on condition	sleep() method is simply used to put your thread on sleep.

Concurrency Problem

- threads run at the same time as other parts of the program, there is no way to know in which order the code will run.
- When the threads and main program are reading and writing the same variables, the values are unpredictable.
- The problems that result from this are called concurrency problems.
- To avoid concurrency problems, it is best to share as few attributes between threads.

Deadlock in Threads

- When two or more threads are blocked forever, waiting for availability of resources occupied by them.
- Occurs when synchronized keyword causes currently executing thread to lock the resources.
-



Deadlock Condition

```
public class Main
{
  public static void main(String[] args) {
   final String  a="First Thread";
   final String b="Second String";

    // t1 tries to lock a then b
    Thread t1 = new Thread() {
      public void run() {
          synchronized (a) {
           System.out.println("Thread 1: locked String a");

           try { Thread.sleep(2000);}
           catch (Exception e) {}
}

           synchronized (b) {
            System.out.println("Thread 1: locked String b");
           }
          }
      }  };
```

```
    // t2 tries to lock b then  a
    Thread t2 = new Thread() {
      public void run() {
        synchronized (b) {
          System.out.println("Thread 2: locked String b");

          try { Thread.sleep(20000);} catch (Exception e) {}

          synchronized (a) {
            System.out.println("Thread 2: locked String a");
          }
        }
      }
    };
    t1.start();
    t2.start();
  }  }
```

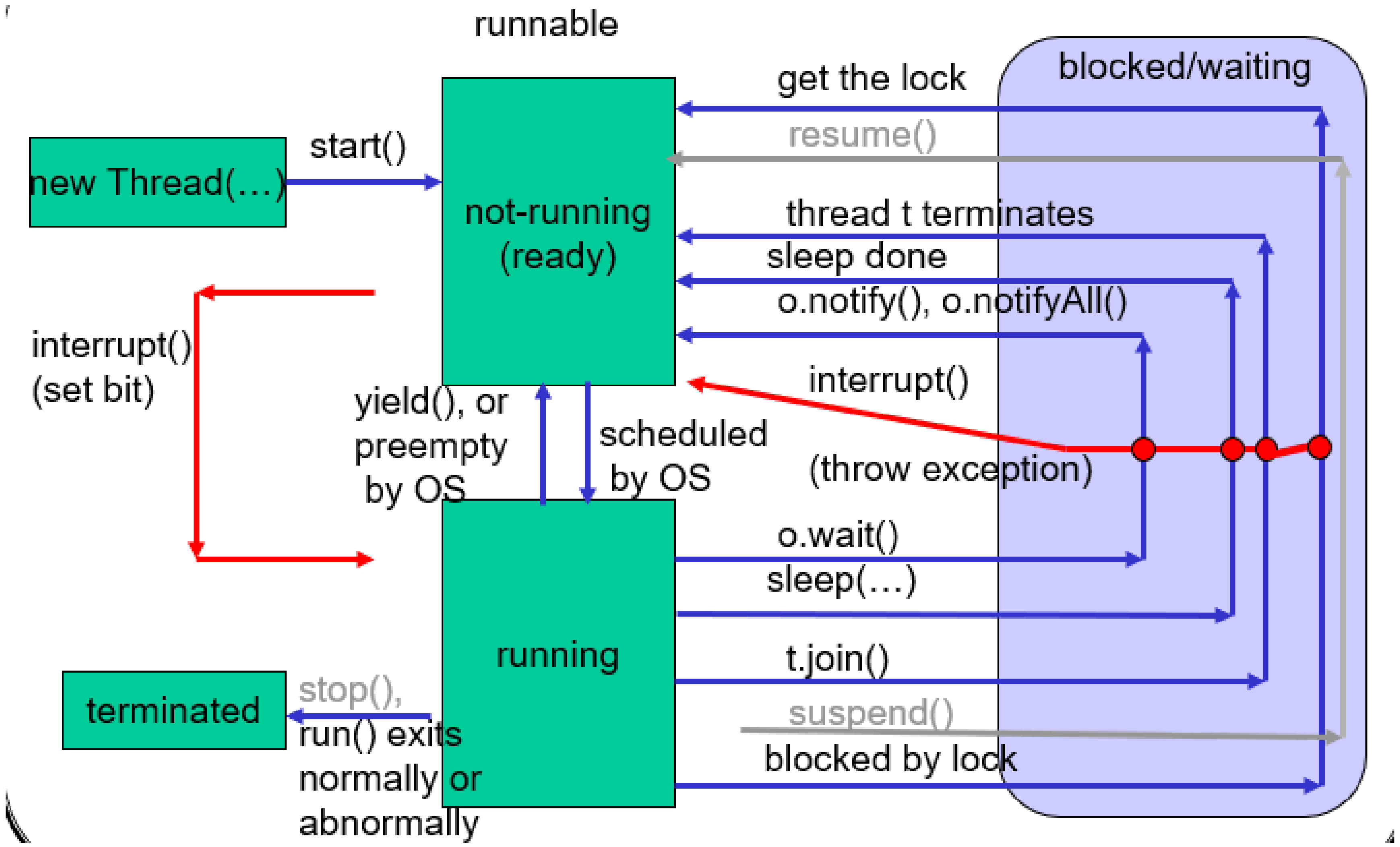
Soultion

Change the order of lock in any one thread. When thread 1 is sleeping then second thread can have lock on the first resource(a).

```
public class Main
{
  public static void main(String[] args) {
   final String  a="First Thread";
   final String b="Second String";

    // t1 tries to lock a then b
    Thread t1 = new Thread() {
     public void run() {
          synchronized (a) {
           System.out.println("Thread 1: locked String a");

           try { Thread.sleep(2000);}
           catch (Exception e) {}
      synchronized (b) {
System.out.println("Thread 1: locked String b");
           }  }  } };
```

```
 // t2 tries to lock a then b
   Thread t2 = new Thread() {
    public void run() {
      synchronized (a) {
        System.out.println("Thread 2: locked String b");

        try { Thread.sleep(20000);} catch (Exception e) {}

        synchronized (b) {
          System.out.println("Thread 2: locked String a");
        }
      } } };
   t1.start();
   t2.start();
  } }
```

Life Cycle of a Java Thread


runnable
blocked/waiting
new Thread(…)
start()
not-running (ready)
get the lock
resume()
thread t terminates
sleep done
o.notify(), o.notifyAll()
interrupt()
(throw exception)
interrupt() (set bit)
yield(), or preempty by OS
scheduled by OS
running
o.wait()
sleep(…)
t.join()
suspend()
blocked by lock
terminated
stop(), run() exits normally or abnormally