Inter Thread communication

- **Inter-thread communication – Thread lar o’rtasidagi aloqa(bog’lanish).**

dasturlash.uz

Inter Thread communication 1

- **Inter-thread communication is** all about allowing threads to communicate with each other.
- Two threads can communicate witch each other by using wait(),notify(),notifyAll() methods.
- **Inter-thread communication** - bu thread larning bir-biri bilan aloqa qilishiga imkon berishdir.
- Ikkita thread bir biribilan wait(), notify(), notifyAll() metodlari orqali komunikatsiya/aloqa qilishi mumkin.

dasturlash.uz

Inter Thread communication 2

- The thread which is expecting updation is responsible to call wait() method, Then immediately the thread will enter into waiting state.
- The thread which is responsible to perform updation, after perform updation it is responsible to call notify method. Then waiting thread will get that notification and continue its execution with those updated items.
- Update kutayotgan Thread wait() metodini chaqiradi. Shundan so'ng Thread darhol waiting state/holat ga o'tadi.
- Update qilish kerak bo’lgan Thread , Update qilganidan keyin notify metodini chaqirish kerak. Shundan keyin kutayotgan thread habarni oladi va waiting state dan chiqadi va yangi malumotlar bilan o’z ishini davom ettiradi.

Inter Thread communication 3

- Cooperation (Inter-thread communication) 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.
- Hamkorlik (Thread larning bir birlari bilan muloqati) bu shunday mexanizm kiy, u Thread larga synchronized maydonda ishlab turgan holida o’z ishini to’xtatib turadi va u lock qilgan ob’ektini unlock qiladi. Boshqa Thread shu ob’ect ni lock qilishi mumkin bo’ladi.
- Boshqa Thread ishini tugatib bo’lib ob’ekctni unlock qilsa boynagi thread yana o’z ishini davom ettirishi mumkin bo’ladi.

dasturlash.uz

Inter Thread communication wait()

- The wait() method causes current thread to release the lock and wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed.
- The current thread must own this object's monitor, so it must be called from the synchronized method only otherwise it will throw exception.
- wait() metodi berilgan/(o’zi ishlayotgan) ob’ekt ning lock ni bo’shatadi va boshqa Thread shu ob’ekct uchun notify() , notifyAll() metodini chaqirishini yoki malum bir vaqt kutadi.
- Hozirgi Thread ob’kectni monitoriga (qulfiga) egasi bo’lishi kerak. Shu sababdan wiat method synchronized maydondan chaqirilishi kerak, bo’lmasa Exception sodir bo’ladi.

dasturlash.uz

Inter Thread communication notify()

- The notify() method wakes up a single thread that is waiting on this object's monitor. If any threads are waiting on this object, one of them is chosen to be awakened. The choice is arbitrary and occurs at the discretion of the implementation.
- notify() – xabar berish metodi, shu ob’ektni kutayotgan boshqa Thread ga habar berish uchun ishlatiladi.Natijada o’sha Thread uyg’otiladi. Agar Thread lar shu o’bektni kutayotgan bo’sla ulardan biri uyg’otish uchun tanlanadi. Tanlov ixtiyoriy tartibda bo’ladi.
- Yani birnechta Thread kutayotgan bo’lsa faqat bittasiga xabar beriladi (uyg’otiladi).

dasturlash.uz

Inter Thread communication notifyAll()

- Wakes up all threads that are waiting on this object's monitor.
- notifyAll() – barchasiga habar berish metodi ob’ektni kutayotgan barcha thread larga habar beradi.

Inter Thread communication Important

- On which object we are calling wait() method Thread required the lock of that particular Object.

- Biz qaysi ob’ektni wait() methodini chaqirmoqchi bo’lsa Therad o’sha ob’ektni lock / qulfuni talab qiladi.
- Yani ob’kect oldin quluflanadi (synchronized ga olinadi) keyin uni wait() methodini chaqirsak bo’ladi.

```
s1 = new Stack()
s2 = new Stack()

synchronized(s1){
   s1.wait()
}
```

dasturlash.uz

Inter Thread communication

- .wait()/.notify()/.notifyAll()
- Methods present in object class, but not in thread class. Because thread can call this methods an any java object.
- To call .wait()/.notify()/.notifyAll() methods an any object, thread should be owner that object . That is the thread should has lock of that object. That is the thread should be inside in synchronized area.
- Hence we can call .wait()/.notify()/.notifyAll() methods only from synchronized area. Other ways we will get runtime Exception saying RE: IllegalMonitorStateException.

Inter Thread communication Example 1

- Main Thread starts a child thread and will wait for child thread
- When a child Thread gives notification of deid child Main continues its execution
- Main Thread chidl Thread ni ishga tushuradi va uni kutib turadi.
- Child o’z ishini tugatganidan keyin Main ga habar beradi.

```
public static void main(String[] args) throws InterruptedException {
        MyThreadB myThreadB = new MyThreadB();
        myThreadB.start();
        synchronized (myThreadB) {
            System.out.println(" Main Thread Calling waite method "); //< 1
            myThreadB.wait();
            System.out.println(" Main Thread got notification "); //< 4
            System.out.println(myThreadB.total);
        }
    }
```

dasturlash.uz

Inter Thread communication Example 2

```
public class MyThreadB extends Thread {
    public int total = 0;
    @Override
    public void run() {
        synchronized (this) {
            System.out.println(" Child Thread start running "); //< 2
            for (int i = 0; i <= 100; i++) {
                total += i;
            }
            try {
                Thread.sleep(2000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            System.out.println(" Child Thread trying to give notification "); //< 3
            this.notify();
        }
    }
}
```

- When child thread fished its work. Child sends notification to other objects for it.
- Chidl Thread o’z ishini tugatganidan keyin, uni kutayotgan thread larga habat jo’natadi.

dasturlash.uz

Difference between wait and sleep?

- Let's see the important differences between wait and sleep methods.
- sleep() va wait() methodlarini muhim farqlari.
- **Wait**
  - The wait() method releases the lock. - lock ni bo'shatadi.
  - It is a method of Object class – Object class ning methodi.
  - It is the non-static method   - non static method
  - It should be notified by notify() or notifyAll() methods – Thread uyg'onishi uchun u notify() va notifyAll() methodlaridan habar kelishi kerak.
- Sleep
  - The sleep() method doesn't release the lock. – ob'ektni qulfuni bo'shatmaydi.
  - It is a method of Thread class – Thread class ning methodi.
  - It is the static method – static method.
  - After the specified amount of time, sleep is completed. Malum bir vaqtdan keyin sleep tugaydi.

dasturlash.uz

Difference between wait and join?

- The wait() and join() methods are used to pause the current thread.
- The wait() is used in with notify() and notifyAll() methods, but join() is used in Java to wait until one thread finishes its execution.
- wait() is mainly used for shared resources, a thread notifies other waiting thread when a resource becomes free.
- On the other hand join() is used for waiting a thread to die.
- wait() va join() methodlari hozirgi Thread ni to’xtatish uchun ishlatiladi.
- wiat() methodi notify() va notifyAll() methodlari bilan ishlatiladi, ammo join() methodi Javada boshqa bir thread ni o’z ishini tugatishini kutadi.
- wait() umumiy manbalar uchun ishlatiladi, resource bo’sh bo’lganida u resource kutayotgan boshqa thread ga habar beradi.
- Join boshqa Thread ni died/o’lishini bo’lishini kutadi.

dasturlash.uz

Consumer and Producer Problem

- Iste'molchi va ishlab chiqaruvchi muammosi
- Producer – ishlab chiqaruvchi
- Consumer - Istemolchi

dasturlash.uz

Consumer and Producer Problem 1

- Producer thread is responsible to produce items to the queue and Consumer thread is responsible to consume items from the queue.
- If queue is empty than consumer thread will call wait() method and entered into waiting state.
- After producing items to the queue producer thread is responsible to call notify method.
- Then waiting consumer will get that notification and continue it is execution with updated items.
- Producer thread item ishlab chiqarib ulanri queue ga joylash uchun javobgar , Consumer esa queue dan item larni qabul qilib olish uchun javobgar.
- Agar queue bo’sh bo’lsa Consumer thread wait() methodni chaqiradi va waiting state ga o’tadi.
- Producer Thread item larni ishlab chiqib queue ga joylashtirganidan keyin notify() methodni chaqirishi kerak.
- Undan keyin kutayotgan Consumer Thread habar oladi va itemlarni queue dan olib ishlashni davom ettiradi.

Consumer and Producer Problem implementation

- There are many ways of implementing Consumer and Producer problems. We introduced one of them.

Consumer example 1

- Inside Producer Class

```
public class Producer extends Thread {
    private List<Integer> sharedQueue;
   //maximum number of products which sharedQueue can hold at a time.
    private int maxSize = 2;
    private Integer counter = 0;

   public Producer(List<Integer> sharedQueue) {
        this.sharedQueue = sharedQueue;
    }
}
```

dasturlash.uz

Consumer example 2

- Inside Producer Class

```
@Override
  public void run() {

        synchronized (sharedQueue) {
           while (true) {
              while (sharedQueue.size() == maxSize) {
                 Thread.sleep(1000);
                 sharedQueue.wait();
              }
              counter++;
              System.out.println("Produced : " + counter);
              sharedQueue.add(counter);
              sharedQueue.notify();
           }
        }
  }
```

dasturlash.uz

Consumer example 1

- Inside Consumer class

```
public class Consumer extends Thread {
    private List<Integer> sharedQueue;

    public Consumer(List<Integer> sharedQueue) {
        this.sharedQueue = sharedQueue;
    }
}
```

Consumer example 2

- Inside Consumer class

```
@Override
    public void run() {
            synchronized (sharedQueue) {
                while (true) {
                    while (sharedQueue.size() == 0) {
                        System.out.println("Queue is empty, consumerThread is waiting for "
                                + "producerThread to produce, sharedQueue's size= 0");
                        sharedQueue.wait();
                    }
                    System.out.println("CONSUMED : " + sharedQueue.remove(0));
                    sharedQueue.notify();
                }
            }
    }
```

dasturlash.uz

Consumer Producer Main Class

```
public class PCSolutionMain {
    public static void main(String[] args) {
        List<Integer> sharedQueue = new LinkedList<Integer>(); //Creating shared object

        Producer producer = new Producer(sharedQueue);
        Consumer consumer = new Consumer(sharedQueue);
        producer.start();
        consumer.start();
    }
}
```

Consumer Producer with BlockingQueue

dasturlash.uz

BlockingQueue - Producer

```
public class Producer implements Runnable {
    private final BlockingQueue<Integer> sharedQueue;
    public Producer(BlockingQueue<Integer> sharedQueue) {
        this.sharedQueue = sharedQueue;
    }
    @Override
    public void run() {
        for (int i = 1; i <= 10; i++) {
            try {
                System.out.println("Produced : " + i);
                //put/produce into sharedQueue.
                sharedQueue.put(i);
            } catch (InterruptedException ex) {

            }
        }
    }
}
```

BlockingQueue - Consumer

```
public class Consumer implements Runnable {
    private BlockingQueue<Integer> sharedQueue;
    public Consumer(BlockingQueue<Integer> sharedQueue) {
        this.sharedQueue = sharedQueue;
    }
    @Override
    public void run() {
        while (true) {
            try {
                //take/consume from sharedQueue.
                System.out.println("CONSUMED : " + sharedQueue.take());
            } catch (InterruptedException ex) {

            }
        }
    }
}
```

BlockingQueue Main

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

        //Creating shared object
        BlockingQueue<Integer> sharedQueue = new LinkedBlockingQueue<Integer>();

        Producer producer = new Producer(sharedQueue);
        Consumer consumer = new Consumer(sharedQueue);

        Thread producerThread = new Thread(producer, "ProducerThread");
        Thread consumerThread = new Thread(consumer, "ConsumerThread");
        producerThread.start();
        consumerThread.start();

    }
}
```

Deadlock

- O'lik qulf

dasturlash.uz

Deadlock 1

- If 2 thread waiting to each other forever such type of infinite waiting is called deadlock.
- Synchronized keyword is only reason deadlock situation.
- By using synchronized keyword we have to separate care. There are no resolution technique for deadlock but prevential technique a available.

- Agar 2ta Thread bir birnini abadiy kutayotgan bo’lsa , bunday cheksiz kutish jarayoni Deadlock deyiladi.
- Synchronized deadlock lekib chiqishining yagona sababi dir.
- Synchronized kalit so’zidan foydalanayotgan biz alohida extiyot bo’lishimiz kerak. Deadlockni hal qilish uchun texnik usuli yo’q. Ammo faqat oldini olishimiz mumkin.

dasturlash.uz

Deadlock example 1

- We have 2 resources

```
public class TestDeadlockExample1 {

  public static void main(String[] args) {
      final String resource1 = "ratan jaiswal";
      final String resource2 = "vimal jaiswal";

      …..
      …..
    t1.start();
    t2.start();

    }
}
```

Deadlock example 2

- Thread 1

```
Thread t1 = new Thread() {
        public void run() {
            synchronized (resource1) {
                System.out.println("Thread 1: locked resource 1");
                try {
                    Thread.sleep(1000);
                } catch (Exception e) {
                }
                synchronized (resource2) {
                    System.out.println("Thread 1: locked resource 2");
                }
            }
        }
    };
```

Deadlock example 3

- Thread 2

```
Thread t2 = new Thread() {
        public void run() {
            synchronized (resource2) {
                System.out.println("Thread 2: locked resource 2");

                try {
                    Thread.sleep(1000);
                } catch (Exception e) {
                }

                synchronized (resource1) {
                    System.out.println("Thread 2: locked resource 1");
                }
            }
        }
    };
```

Deadlock example explanation

- Thread 1 locks resource1 and sleeps
- At the same time Thread 2 locks resource 2
- Thread 1 also required lock of resource 2 and waits until Thread 2 releases lock of resource 2
- At the same time Thread2 also required lock of resource 1 and waits until Thread 1 releases lock of resource 1.
- Mazgi.

More Complicated Deadlocks

- A deadlock may also include more than two threads. The reason is that it can be difficult to detect a deadlock. Here is an example in which four threads have deadlocked.
- Thread 1 locks A, waits for B
- Thread 2 locks B, waits for C
- Thread 3 locks C, waits for D
- Thread 4 locks D, waits for A
- Thread 1 waits for thread 2, thread 2 waits for thread 3, thread 3 waits for thread 4, and thread 4 waits for thread 1.

How to avoid deadlock?

- Deadlocks cannot be completely resolved.
- But we can avoid them in our example:
  - A solution for a problem is found at its roots. In deadlock it is the pattern of accessing the resources A and B, is the main issue. To solve the issue we will have to simply re-order the statements where the code is accessing shared resources.
  - Bizni misolda deadlock ni oldini olish uchun resource larni block qilishni tartibini o’agartirishimiz kerak. Namuna keyingi slide da.

Avoid deadlock 1

- Thread 1

```
Thread t1 = new Thread() {
        public void run() {
            synchronized (resource1) {
                System.out.println("Thread 1: locked resource 1");
                Thread.sleep(1000);
                synchronized (resource2) {
                    System.out.println("Thread 1: locked resource 2");
                }
            }
        }
    };
```

Avoid deadlock 2

- Thread 2

```
Thread t2 = new Thread() {
         public void run() {
            synchronized (resource1) {
               System.out.println("Thread 2: locked resource 2");
               Thread.sleep(1000);
               synchronized (resource2) {
                  System.out.println("Thread 2: locked resource 1");
               }
            }
         }
      };
```

dasturlash.uz

Deadlock Links

- https://www.javatpoint.com/deadlock-in-java
- https://www.tutorialspoint.com/java/java_thread_deadlock.htm