**Multithreading – Part-10 – Inter Thread Communication Part-02**

* **wait() and notify() example:**

class ThreadA{

pubic static void main(String[] args) throws InterruptedException{

ThreadB b = new ThreadB();

b.start();

synchronized(b){

System.out.println(“Main thread calling wait”); //1

b.wait();

System.out.println(“Main thread got notification”); //4

System.out.println(b.total); // 5

}

}

}

class ThreadB extends Thread{

int total = 0;

public void run(){

synchronized(this){

System.out.println(“Child thread started calculation”); //2

for(int i =1; i<= 100; i++){

total = total + i;

}

System.out.println(“Child thread giving notification”); //3

this.notify();

}

}

}

Output:

Main thread calling wait method

Child thread started calculation

Child thread giving notification

Main thread got notification

5050

Note:

If the child thread got the chance first in the above program, the main thread will wait notification for infinite amount of time.

To overcome this problem, use the overloaded wait(long ms), now the main thread wait for the time specified once the time expires the program will exit.

* **Producer Consumer Problem:**

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 then 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 its execution with updated items.

Example:

ProducerThread

class ProducerThread{

produce(){

synchronized(q){

produce items to the queue

q.notify();

}

}

}

Queue [ 0, 0, 0, 0, 0]

Consumer Thread:

class ConsumerThread{

consume(){

synchronized(q){

if( q is empty){

q.wait();

else

consume items

}

}

}

* **Difference between notify() and notifyAll():**

We can use notify method to give the notification for only one waiting thread, if multiple threads are waiting then only one thread will be notified and the remaining threads have to wait for further notifications.

Which thread will be notified we can’t expect it depends on JVM.

We can use notifyAll() to give the notification for all waiting threads of a particular object.

Even though multiple threads notified, but execution will be performed one by one, because threads require lock and only one lock is available.

Note: Recall about the bus notification for the travelers waiting at the bus stand and they get in one by one in to the bus.

Note:

On which object we are calling wait method thread requires the lock of that particular object. For example if we are calling wait() on s1 then we have to get lock of s1 object, but not s2 object.

Example:

Stack s1 = new Stack();

Stack s2 = new Stack();

synchronized(s1){

s2.wait();

}

RE: IllegalMonitorStateException

synchronized(s2){

s2.wait();

}

//Valid