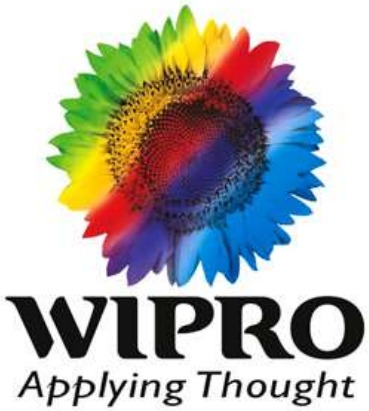


Multithreading

Inter Thread Communication

© 2012 WIPRO LTD | WWW.WIPRO.COM

Agenda

1. Inter Thread Communication
2.
3.
4.

© 2012 WIPRO LTD | WWW.WIPRO.COM

Objectives

At the end of this module, you will be able to:

- Inter Thread Communication

© 2012 WIPRO LTD | WWW.WIPRO.COM

Inter Thread Communication

© 2012 WIPRO LTD | WWW.WIPRO.COM

Thread Messaging

- In Java, you need not depend on the OS to establish communication between threads
- All objects have predefined methods, which can be called to provide inter-thread communication
- Threads are often interdependent - one thread depends on another thread to complete an operation, or to service a request.

© 2012 WIPRO LTD | WWW.WIPRO.COM

Inter-Thread Communication

- The words wait and notify encapsulate the two central concepts to thread communication
  - A thread waits for some condition or event to occur.
  - You notify a waiting thread that a condition or event has occurred.
- To avoid polling, Java's elegant inter-thread communication mechanism uses:
  - **wait( )**
  - **notify( ), and notifyAll( )**

© 2012 WIPRO LTD | WWW.WIPRO.COM

Inter-Thread Communication (Contd.).

- wait( ), notify( ) and notifyAll( ) are:
  - Declared as final in Object
  - Hence, these methods are available to all classes
  - These methods can only be called from a synchronized context

- wait( ) directs the calling thread to surrender the monitor, and go to sleep until some other thread enters the monitor of the same object, and calls notify( )

- notify( ) wakes up the other thread which was waiting on the same object(*that had called wait() previously on the same object*)

© 2012 WIPRO LTD | WWW.WIPRO.COM

The Problem

- Consider the situation, where one thread(Developer) is producing some data, and another thread(Client) is consuming it.

- Suppose that the Developer has to wait until the Client has completed reading before it produces more data. In a polling system, the Client would waste many CPU cycles while it waits for the Developer to produce.

- Once the Developer has completed its job, the Developer would start polling, wasting more CPU cycles, waiting for the Client to complete its work and this will go on.

© 2012 WIPRO LTD | WWW.WIPRO.COM

The Problem (Contd.).

- The following sample program **incorrectly** implements a simple form of the Developer/Client problem

- It consists of four classes namely:
  - QueueClass, the queue that you are trying to synchronize
  - Developer, the threaded object that is producing queue entries
  - Client, the threaded object that is consuming queue entries
  - Caller, the class that creates the single Queue, Developer, and Client

© 2012 WIPRO LTD | WWW.WIPRO.COM

The Problem (Contd.).

```
public class QueueClass {

int number;
synchronized int get( ) {
System.out.println( "Got: " + number);
return number;
}


synchronized void put( int number) {
this.number = number;
System.out.println( "Put: " + number);
}
}
```

Method which gets the number

Method which puts the number

© 2012 WIPRO LTD | WWW.WIPRO.COM

The Problem (Contd.).

```
public class Developer implements Runnable {

QueueClass queueClass;
Developer ( QueueClass queueClass) {
this.queueClass = queueClass;
new Thread( this, "Developer").start( );
}
public void run( ) {
int i = 0;
while (true) {
queueClass.put (i++);
}
}

}
```


© 2012 WIPRO LTD | WWW.WIPRO.COM

The Problem (Contd.).

```
public class Client implements Runnable {


QueueClass queueClass;
Client (QueueClass queueClass) {
this.queueClass = queueClass;
new Thread (this, "Client").start( );
}
public void run( ) {
while (true) {
queueClass.get( );
}
}

}
```


© 2012 WIPRO LTD | WWW.WIPRO.COM

The Problem (Contd.).

```
public class Caller {

public static void main(String args[]){
QueueClass queueClass = new QueueClass( );
new Developer (queueClass);
new Client (queueClass);
System.out.println("Press Control-C to stop");
}
}
```


© 2012 WIPRO LTD | WWW.WIPRO.COM

Output

- A portion of the output:

```
Put: 52948
Put: 52949
Put: 52950
Put: 52951
Put: 52952
Put: 52953
Put: 52954
Put: 52955
Put: 52956
Put: 52957
Put: 52958
Put: 52959
Got: 52959
Got: 52959
Got: 52959
Got: 52959
Got: 52959
Got: 52959
```

As you can see, the process of producing numbers and reading numbers is not proper. Only after the put() method had executed many times, the got() method read the number. This is not a proper output.

© 2012 WIPRO LTD | WWW.WIPRO.COM

Solution using wait() and notify()

- *Modifying the QueueClass in the previous example using* **wait()** *and* **notify()** *methods:*

```
public class QueueClass {
int number;
boolean valueset = false;

synchronized int get( ) {
if (!valueset)
    try {
          wait( );
     }
    catch (InterruptedException e) {
       System.out.println("InterruptedException caught");
    }
System.out.println( "Got: " + number);
valueset = false;
notify( );
return number;
}
```


© 2012 WIPRO LTD | WWW.WIPRO.COM

Inter-Thread Communication (Contd.).

```
synchronized void put( int number) {
if (valueset)
    try {
          wait( );
    }
    catch (InterruptedException e) {
          System.out.println("InterruptedException
caught");
    }
this.number = number;
valueset = true;
 System.out.println( "Put: " + number);
notify( );}}
```

Note: Rest of the code remains the same

© 2012 WIPRO LTD | WWW.WIPRO.COM

Output

- Put: 197376
- Got: 197376
- Put: 197377
- Got: 197377
- Put: 197378
- Got: 197378
- Put: 197379
- Got: 197379
- Put: 197380
- Got: 197380
- Put: 197381
- Got: 197381
- Put: 197382
- Got: 197382
- Put: 197383
- Got: 197383
- Put: 197384
- Got: 197384

Now we can see that only after each put() method, the get() method is called.

© 2012 WIPRO LTD | WWW.WIPRO.COM

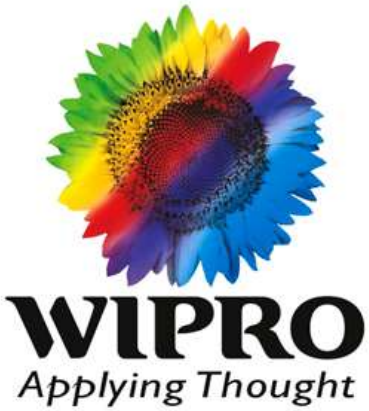
Assignment

© 2012 WIPRO LTD | WWW.WIPRO.COM

Summary

- Inter-thread Communication

© 2012 WIPRO LTD | WWW.WIPRO.COM



Thank You

© 2012 WIPRO LTD | WWW.WIPRO.COM