Thread Inter Communication

Objectives

On completion, you would be able to learn

- Inter thread communication methods
- wait() method
- notify() method
- Producer-Consumer problem
- Solution to Producer-Consumer problem

Recap

You have learnt

- Synchronization
- Unsynchronized examples
- Locking objects
- Examples on synchronization

Inter Thread Communication Methods

- When more than one thread uses a shared resource they need to synchronize with each other
- While using a shared resource the threads need to communicate with each other, to get the expected behavior of the application
- Java provides some methods for the threads to communicate

Inter Thread Communication Methods Contd..

Methods for Interthread Communication
`public final void wait()`
Causes this thread to wait until some other thread calls the *notify* or *notifyAll* method on this object. May throw *InterruptedException*.
`public final void notify()`
Wakes up a thread that called the *wait* method on the same object.
`public final void notifyAll()`
Wakes up all threads that called the *wait* method on the same object.

wait() Method

- wait() method causes a thread to release the lock it is holding on an object; allowing another thread to run
- wait() method is defined in the Object class
- wait() can only be invoked from within synchronized code
- It should always be wrapped in a try block as it throws IOExceptions
- wait() can only invoked by the thread that owns the lock on the object

wait() Method

Contd . . .

- When wait() is called, the thread becomes disabled for scheduling and lies dormant until one of four things occur:
  - another thread invokes the notify() method for this object and the scheduler arbitrarily chooses to run the thread
  - another thread invokes the notifyAll() method for this object
  - another thread interrupts this thread
  - the specified wait() time elapses

wait() Method

Contd . . .

- When one of the above occurs, the thread becomes re-available to the thread scheduler and competes for a lock on the object

- Once it regains the lock on the object, everything resumes as if no suspension had occurred

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
- Can only be used within synchronized code
- The awakened thread will not be able to proceed until the current thread relinquishes the lock on this object

Producer-Consumer Problem

- Producing thread may write to buffer (shared memory)
- Consuming thread reads from buffer
- If not synchronized, data can become corrupted
  - Producer may write before consumer read last data
    - Data lost
  - Consumer may read before producer writes new data
    - Data "doubled"

Producer-Consumer Problem

Contd . . .

- Using synchronization
  - If producer knows that consumer has not read last data, calls wait (awaits a notify command from consumer)
  - If consumer knows producer has not updated data, calls wait (awaits notify command from producer)

Incorrect Implementation

```
class Q {
  int n;
  synchronized int get() {
    System.out.println("Got: " + n);
    return n;
  }
  synchronized void put(int n) {
    this.n = n;
    System.out.println("Put: " + n);
  }
}
class Producer implements Runnable {
  Q q;
  Producer(Q q) {
    this.q = q;
    new Thread(this, "Producer").start();
  }
  public void run() {
    int i = 0;
    while(true) {
      q.put(i++);
    }
  }
}

class Consumer implements Runnable {
  Q q;
  Consumer(Q q) {
    this.q = q;
    new Thread(this, "Consumer").start();
  }
  public void run() {
    while(true) {
      q.get();
    }
  }
}
```

class Q is treated as Buffer

Producer class

Consumer class

Incorrect Implementation

Contd . . .

```
class PC {
  public static void main(String args[]) {
    Q q = new Q();
    new Producer(q);
    new Consumer(q);
    System.out.println("Press Control-C to stop.");
  }
}
```

Incorrect Implementation

Contd . . .

Output

```
C:\workshop\test>C:\java\jdk1.5.0\bin\java -classpath .;c:\workshop\test;c:\java\jdk1.5.0\lib\dt.jar;c:\java\jdk1.5.0\lib\tools.jar; PC
Put: 0
Put: 1
Put: 2
Put: 3
Put: 4
Put: 5
Put: 6
Put: 7
Put: 8
Put: 9
Put: 10
Put: 11
Put: 12
Put: 13
Put: 14
Put: 15
Put: 16
Put: 17
Put: 18
Put: 19
Put: 20
Put: 21
Put: 22
Put: 23
Put: 24
```

Only Producer is doing work

```
Put: 164
Put: 165
Put: 166
Put: 167
Put: 168
Put: 169
Put: 170
Put: 171
Got: 171
Put: 172
Got: 172
Put: 173
Got: 173
Put: 174
Got: 174
Put: 175
Got: 175
Got: 175
Got: 175
Got: 175
Got: 175
Got: 175
Got: 175
```

Though producer and consumer are working, there is no sync between them

Correct Implementation

```
class Q {
  int n;
  boolean valueSet = false;
  synchronized int get() {
    if(!valueSet)
      try {
        wait();
      } catch(InterruptedException e)
    {     System.out.println("InterruptedExcep
    tion caught");
      }
      System.out.println("Got: " + n);
      valueSet = false;
      notify();
      return n;
  }
```

```
  synchronized void put(int n) {
  if(valueSet)
    try {
      wait();
    } catch(InterruptedException e) {

  System.out.println("InterruptedException
  caught");
    }

    this.n = n;
    valueSet = true;
    System.out.println("Put: " + n);
    notify();
  }
}
```

class Q is treated as Buffer

Use of wait and notify

Use of wait and notify

get() is used by Consumer and put () is used by Producer

Correct Implementation

Contd . . .

```
class Producer implements Runnable {
  Q q;

  Producer(Q q) {
    this.q = q;
    new Thread(this, "Producer").start();
  }

  public void run() {
    int i = 0;

    while(true) {
      q.put(i++);
    }
  }
}
```

Producer class

```
class Consumer implements Runnable {
  Q q;

  Consumer(Q q) {
    this.q = q;
    new Thread(this, "Consumer").start();
  }

  public void run() {
    while(true) {
      q.get();
    }
  }
}
```

Consumer class

Correct Implementation

Contd . . .

Output

```
C:\workshop\test>C:\java\jdk1.5.0\bin\java -classpath .;c:\workshop\test;c:\java\jdk1.5.0\lib\dt.jar;c:\java\jdk1.5.0\lib\tools.jar; PC
Press Control-C to stop.
Put: 0
Got: 0
Put: 1
Got: 1
Put: 2
Got: 2
Put: 3
Got: 3
Put: 4
Got: 4
Put: 5
Got: 5
Put: 6
Got: 6
Put: 7
Got: 7
Put: 8
Got: 8
Put: 9
Got: 9
Put: 10
Got: 10
Put: 11
Got: 11
Put: 12
Got: 12
Put: 13
Got: 13
Put: 14
Got: 14
```

put and get are in synch

Quiz

1. Which is NOT a method useful for interThread communication ?

   A. wait()
   B. notify()
   C. suspend()
   D. notifyAll()

Quiz Contd..

2. During wait(), which method wakes up the thread ?

A. resume()
B. sleep()
C. notify()
D. yield()

Frequently Asked Questions

1. Which methods are useful for inter thread communication ?
2. Explain the wait() method
3. Explain the notify() method
4. Define and explain Producer-Consumer problem