More on Java Synchronization

D. Thiebaut
CSC352 Fall 2013
Outline

- Basics of Thread Operations (new stuff)
- Thread States
- Thread Scheduling
- Threads and I/O
- Producer/Consumer Pattern
- wait()/notify()
- Thread-Safe Data Structures
- Processing Lab
The Basics

• Threads
  • run() / start()
  • yield()
  • sleep()
  • join()
  • wait() and notify(), and also notifyAll()
States of a Thread

New thread → running → Terminated

blocked

waiting on an object
sleeping
blocking on I/O
blocked on a lock
How to get the state?

http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/Thread.State.html

- NEW
- RUNNABLE
- BLOCKED
- WAITING
- TIME_WAITING
- TERMINATED
Thread Scheduling

• What is the policy?

• Java doc says: Implemented in the JVM, preemptive, based on priority. (No mention of time-slices.)

• 1 = low priority, 5 = main, 10 = high priority

• getPriority() & setPriority()

• However, most OS implement time-slices (quanta), roughly 1ms, preemptive, and round-robin ==> JVMs do the same
Threads good not only for speedup

App

T1

T2
Threads good not only for speedup
But also to simplify code
Important Concepts

• CPU Bound Processes/Threads
• I/O Bound Processes/Threads
Time Scale

- Why I/O recognizing I/O-bound process is important
  - CPU cycle: 1 ns
  - RAM cycle: 100-500 ns
  - Disk access = seek + latency
    - seek = 1 ms
    - latency = 1/2 rotation, at 7,000 RPM
- Question: How long does the processor wait for data from disk?
// producer code
synchronized (lock) {
    while (!container.isEmpty()) {
        try {
            lock.wait();
        } catch (InterruptedException e) {
        }
    }
    container.put(newItem);
}

// consumer code
synchronized (lock) {
    if (!container.isEmpty()) {
        item = container.getItem();
        consume(item);
        lock.notify();
    }
}
Beware of Deadlocks!

- The Dining-Philosophers Problem

- The Applet
  http://elvis.rowan.edu/~hartley/ConcProglJava/Applets/diningPhilosophers.html
Rule #1 for Preventing Deadlocks

- Grab all the data-structures that you need first
- If you can’t, release them all
- Wait a random amount of time and try again
Thread-Safe Data Structures

Blocking Queues

Non-Blocking Queues

ConcurrentLinkedQueues

ArrayBlockingQueues
DelayQueues
LinkedBlockingQueues
LinkedBlockingDeques
PriorityBlockingQueues
SynchronousQueues
## ConcurrentLinkedQueue

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean add(E e)</td>
<td>Inserts the specified element at the tail of this queue.</td>
</tr>
<tr>
<td>boolean contains(Object o)</td>
<td>Returns true if this queue contains the specified element.</td>
</tr>
<tr>
<td>boolean isEmpty()</td>
<td>Returns true if this queue contains no elements.</td>
</tr>
<tr>
<td>iterator()</td>
<td>Returns an iterator over the elements in this queue in proper sequence.</td>
</tr>
<tr>
<td>boolean offer(E e)</td>
<td>Inserts the specified element at the tail of this queue.</td>
</tr>
<tr>
<td>Object peek()</td>
<td>Retrieves, but does not remove, the head of this queue, or returns null if this queue is empty.</td>
</tr>
<tr>
<td>Object poll()</td>
<td>Retrieves and removes the head of this queue, or returns null if this queue is empty.</td>
</tr>
<tr>
<td>boolean remove(Object o)</td>
<td>Removes a single instance of the specified element from this queue, if it is present.</td>
</tr>
<tr>
<td>int size()</td>
<td>Returns the number of elements in this queue.</td>
</tr>
<tr>
<td>Object[] toArray()</td>
<td>Returns an array containing all of the elements in this queue, in proper sequence.</td>
</tr>
</tbody>
</table>
### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>clear()</code></td>
<td>Removes all of the elements from this collection.</td>
</tr>
<tr>
<td><code>contains(Object o)</code></td>
<td>Returns true if this collection contains the specified element.</td>
</tr>
<tr>
<td><code>drainTo(Collection&lt;? super E&gt; c)</code></td>
<td>Removes all available elements from this queue and adds them into the given collection.</td>
</tr>
<tr>
<td><code>drainTo(Collection&lt;? super E&gt; c, int maxElements)</code></td>
<td>Removes at most the given number of available elements from this queue and adds them into the given collection.</td>
</tr>
<tr>
<td><code>iterator()</code></td>
<td>Returns an iterator over the elements in this queue in proper sequence.</td>
</tr>
<tr>
<td><code>offer(E o)</code></td>
<td>Inserts the specified element at the tail of this queue if possible, returning immediately if this queue is full.</td>
</tr>
<tr>
<td><code>offer(E o, long timeout, TimeUnit unit)</code></td>
<td>Inserts the specified element at the tail of this queue, waiting if necessary up to the specified wait time for space to become available.</td>
</tr>
<tr>
<td><code>peek()</code></td>
<td>Retrieves, but does not remove, the head of this queue, returning <code>null</code> if this queue is empty.</td>
</tr>
<tr>
<td><code>poll()</code></td>
<td>Retrieves and removes the head of this queue, or <code>null</code> if this queue is empty.</td>
</tr>
<tr>
<td><code>poll(long timeout, TimeUnit unit)</code></td>
<td>Retrieves and removes the head of this queue, waiting if necessary up to the specified wait time if no elements are present on this queue.</td>
</tr>
<tr>
<td><code>put(E o)</code></td>
<td>Adds the specified element to the tail of this queue, waiting if necessary for space to become available.</td>
</tr>
<tr>
<td><code>remainingCapacity()</code></td>
<td>Returns the number of elements that this queue can ideally (in the absence of memory or resource constraints) accept without blocking.</td>
</tr>
</tbody>
</table>
ArrayBlockingQueue (cont’d)

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>remainingCapacity()</code></td>
<td>Returns the number of elements that this queue can ideally accept without blocking.</td>
</tr>
<tr>
<td><code>remove(Object o)</code></td>
<td>Removes a single instance of the specified element from this collection, if it is present (optional operation).</td>
</tr>
<tr>
<td><code>size()</code></td>
<td>Returns the number of elements in this queue.</td>
</tr>
<tr>
<td><code>take()</code></td>
<td>Retrieves and removes the head of this queue, waiting if no elements are present on this queue.</td>
</tr>
<tr>
<td><code>toArray()</code></td>
<td>Returns an array containing all of the elements in this collection.</td>
</tr>
<tr>
<td><code>toArray(T[] a)</code></td>
<td>Returns an array containing all of the elements in this collection; the runtime type of the returned array is that of the specified array.</td>
</tr>
<tr>
<td><code>toString()</code></td>
<td>Returns a string representation of this collection.</td>
</tr>
</tbody>
</table>
Processing Lab

Processing App

setup()

draw()

1/framerate
Processing Lab: Version 1

Processing App

setup()
Processing Lab: Version 1

setup()

Producer

object

object

object

object

draw()

Consumer

object

object

object

object

object

object

object

object

object
Processing Lab: Version 2

setup()

Producer

draw()

Consumer