Java Monitor Objects: Coordination Methods

Douglas C. Schmidt

d.schmidt@vanderbilt.edu

www.dre.vanderbilt.edu/~schmidt

Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA
Learning Objectives in this Part of the Lesson

- Understand how Java built-in monitor objects provide waiting & notification mechanisms that coordinate threads running in a concurrent program.

1. Enter monitor object
2. Acquire lock
3. wait()
4. notifyAll()
5. Release lock
6. Leave monitor object

Critical Section

Java Built-in Waiting & Notification Mechanisms
Java built-in waiting & notification mechanisms

Java synchronized methods & statements only provide a partial solution to concurrent programs.
Java Built-in Waiting & Notification Mechanisms

- Java monitor objects allow threads to coordinate their interactions.

Diagram:

- Acquire lock
- Critical Section
- Wait on condition
- Running Thread
- T1
- T2
- T3
- T4
Java Built-in Waiting & Notification Mechanisms

- Java monitor objects allow threads to coordinate their interactions
  - via the `wait()`, `notify()`, & `notifyAll()` methods

  | void `wait()` – Causes the current thread to wait until another thread invokes the `notify()` method or the `notifyAll()` method for this object |
  | void `notify()` – Wakes up a single thread that is waiting on this object's monitor |
  | void `notifyAll()` – Wakes up all threads that are waiting on this object's monitor |

See [docs.oracle.com/javase/8/docs/api/java/lang/Object.html](docs.oracle.com/javase/8/docs/api/java/lang/Object.html)
Java Built-in Waiting & Notification Mechanisms

- Java monitor objects allow threads to coordinate their interactions
  - via the `wait()`, `notify()`, & `notifyAll()` methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void <code>wait()</code></td>
<td>Causes the current thread to wait until another thread invokes the <code>notify()</code> method or the <code>notifyAll()</code> method for this object</td>
</tr>
<tr>
<td>void <code>notify()</code></td>
<td>Wakes up a single thread that is waiting on this object's monitor</td>
</tr>
<tr>
<td>void <code>notifyAll()</code></td>
<td>Wakes up all threads that are waiting on this object's monitor</td>
</tr>
</tbody>
</table>
Java Built-in Waiting & Notification Mechanisms

- Java monitor objects allow threads to coordinate their interactions via the `wait()`, `notify()`, & `notifyAll()` methods.

**void wait()** – Causes the current thread to wait until another thread invokes the `notify()` method or the `notifyAll()` method for this object.

**void notify()** – Wakes up a single thread that is waiting on this object's monitor.

**void notifyAll()** – Wakes up all threads that are waiting on this object's monitor.
Java Built-in Waiting & Notification Mechanisms

- Java monitor objects allow threads to coordinate their interactions via the `wait()`, `notify()`, & `notifyAll()` methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void wait()</code></td>
<td>Causes the current thread to wait until another thread invokes the <code>notify()</code> method or the <code>notifyAll()</code> method for this object</td>
</tr>
<tr>
<td><code>void notify()</code></td>
<td>Wakes up a single thread that is waiting on this object's monitor</td>
</tr>
<tr>
<td><code>void notifyAll()</code></td>
<td>Wakes up all threads that are waiting on this object's monitor</td>
</tr>
</tbody>
</table>
Java Built-in Waiting & Notification Mechanisms

- Java monitor objects allow threads to coordinate their interactions via the wait(), notify(), & notifyAll() methods

void wait() – Causes the current thread to wait until another thread invokes the notify() method or the notifyAll() method for this object

void notify() – Wakes up a single thread that is waiting on this object's monitor

void notifyAll() – Wakes up all threads that are waiting on this object's monitor

See [en.wikipedia.org/wiki/Thundering_herd_problem](en.wikipedia.org/wiki/Thundering_herd_problem)
Java Built-in Waiting & Notification Mechanisms

- Java built-in monitor objects have one entrance queue & one wait queue

See [en.wikipedia.org/wiki/Monitor_(synchronization)#Implicit_condition_variable_monitors](en.wikipedia.org/wiki/Monitor_(synchronization)#Implicit_condition_variable_monitors)
Java Built-in Waiting & Notification Mechanisms

• Java built-in monitor objects have one entrance queue & one wait queue

Serializes thread access to monitor object’s critical section
Java Built-in Waiting & Notification Mechanisms

• Java built-in monitor objects have one entrance queue & one wait queue

All threads that call wait() are parked on the wait queue
Java built-in monitor objects have one entrance queue & one wait queue.

All `notify()` & `notifyAll()` calls also apply to the wait queue.
Java Built-in Waiting & Notification Mechanisms

- Java built-in monitor objects have one entrance queue & one wait queue

```java
class SimpleBlockingBoundedQueue<E> implements BlockingQueue<E> {
    ...
    public void put(E msg){
        synchronized(this) {
            while (isFull()) wait();
            mList.add(msg);
            notifyAll();
        }
    }

    public E take() ... {
        synchronized(this) {
            while (isEmpty()) wait();
            notifyAll();
            return mList.poll();
        }
    }
    ...
}
```

Java Built-in Waiting & Notification Mechanisms

- Java built-in monitor objects have one entrance queue & one wait queue, e.g.
- put() calls wait() when the queue is full

```java
class SimpleBlockingBoundedQueue<E> implements BlockingQueue<E> {
    ...
    public void put(E msg) {
        synchronized(this) {
            while (isFull()) wait();
            mList.add(msg);
            notifyAll();
        }
    }
    ...
    public E take() ... {
        synchronized(this) {
            while (isEmpty()) wait();
            notifyAll();
            return mList.poll();
        }
    }
    ...
}
```

See en.wikipedia.org/wiki/Guarded_suspension

Atomically releases the intrinsic lock & sleeps on the wait queue
Java Built-in Waiting & Notification Mechanisms

- Java built-in monitor objects have one entrance queue & one wait queue, e.g.
- put() calls wait() when the queue is full
- It also calls notifyAll() after adding an item

```java
class SimpleBlockingBoundedQueue<E> implements BlockingQueue<E> {
    ...
    public void put(E msg) {
        synchronized(this) {
            while (isFull()) wait();
            mList.add(msg);
            notifyAll();
        }
    }
    
    public E take() { ... {
        synchronized(this) {
            while (isEmpty()) wait();
            notifyAll();
            return mList.poll();
        }
    }
    ...
}
```

Must wake up all the threads blocked on the wait queue since waiters are non-uniform

See upcoming lesson on “Java Monitor Objects: Usage Considerations"
Java Built-in Waiting & Notification Mechanisms

- Java built-in monitor objects have one entrance queue & one wait queue, e.g.
  - put() calls wait() when the queue is full
  - It also calls notifyAll() after adding an item

```java
class SimpleBlockingBoundedQueue<E> implements BlockingQueue<E> {
    ...
    public void put(E msg) {
        synchronized(this) {
            while (isFull()) wait();
            mList.add(msg);
            notifyAll();
        }
    }
    ...
    public E take() ... {
        synchronized(this) {
            while (isEmpty()) wait();
            notifyAll();
            return mList.poll();
        }
    }
    ...
}
```

`notifyAll()` is required due to a Java monitor object only having one wait queue

See [stackoverflow.com/questions/37026/java-notify-vs-notifyall-all-over-again/3186336#3186336](https://stackoverflow.com/questions/37026/java-notify-vs-notifyall-all-over-again/3186336#3186336)
Java Built-in Waiting & Notification Mechanisms

- Java built-in monitor objects have one entrance queue & one wait queue, e.g.
  - `put()` calls `wait()` when the queue is full
  - `take()` calls `wait()` when the queue is empty

```java
class SimpleBlockingBoundedQueue<E> implements BlockingQueue<E> {
    ...
    public void put(E msg) {
        synchronized(this) {
            while (isFull()) wait();
            mList.add(msg);
            notifyAll();
        }
    }
    ...
    public E take() {... {
        synchronized(this) {
            while (isEmpty()) wait();
            notifyAll();
            return mList.poll();
        }
    }
    ...
}
```

atomically releases the intrinsic lock & sleeps on the wait queue

See [en.wikipedia.org/wiki/Guarded_suspension](en.wikipedia.org/wiki/Guarded_suspension)
Java Built-in Waiting & Notification Mechanisms

- Java built-in monitor objects have one entrance queue & one wait queue, e.g.
  - put() calls wait() when the queue is full
  - take() calls wait() when the queue is empty
  - It also calls notifyAll() after removing an item

```java
class SimpleBlockingBoundedQueue<E> {
    implements BlockingQueue<E> {
        ...
        public void put(E msg) {
            synchronized (this) {
                while (isFull()) wait();
                mList.add(msg);
                notifyAll();
            }
        }
        ...
        public E take() ... {
            synchronized (this) {
                while (isEmpty()) wait();
                notifyAll();
                return mList.poll();
            }
        }
        ...
    }
}
```

Must wake up all the threads blocked on the wait queue since waiters are non-uniform

Again, notifyAll() is required here due to the limitations of Java monitor objects, which only have one wait queue
Java Built-in Waiting & Notification Mechanisms

- Java built-in monitor objects have one entrance queue & one wait queue

```java
class SimpleBlockingBoundedQueue<E> 
    implements BlockingQueue<E> {
...

    public void put(E msg) {
        synchronized(this) {
            while (isFull()) wait();
            mList.add(msg);
            notifyAll();
        }
    }

    public E take() {
        synchronized(this) {
            while (isEmpty()) wait();
            notifyAll();
            return mList.poll();
        }
    }

    ...
```

The put() & take() methods are examined later in this lesson

See upcoming lesson on “Java Monitor Objects: Coordination Example Implementation”
Java Built-in Waiting & Notification Mechanisms

- Java built-in monitor object synchronizers can be implemented w/POSIX-like synchronizers
Java Built-in Waiting & Notification Mechanisms

- Java built-in monitor object synchronizers can be implemented w/POSIX-like synchronizers, e.g.
  - Entrance queue is akin to a POSIX recursive mutex

See computing.llnl.gov/tutorials/pthreads/#Mutexes
Java Built-in Waiting & Notification Mechanisms

- Java built-in monitor object synchronizers can be implemented w/POSIX-like synchronizers, e.g.
  - Entrance queue is akin to a POSIX recursive mutex
  - Wait queue is akin to a POSIX condition variable

See computing.llnl.gov/tutorials/pthreads/#ConditionVariables
Java Built-in Waiting & Notification Mechanisms

- Java built-in monitor object synchronizers can be implemented w/POSIX-like synchronizers, e.g.
  - Entrance queue is akin to a POSIX recursive mutex
  - Wait queue is akin to a POSIX condition variable
  - Similar to Java ConditionObjects

See earlier lessons on "Java ConditionObjects"
Java Built-in Waiting & Notification Mechanisms

- Java built-in monitor object synchronizers can be implemented w/POSIX-like synchronizers, e.g.
  - Entrance queue is akin to a POSIX recursive mutex
  - Wait queue is akin to a POSIX condition variable
- The implementation in the Oracle JDK uses lower-level locking primitives

```cpp
199  bool   try_enter (TRAPS);
200  void   enter(TRAPS);
201  void   exit(bool not_suspended, TRAPS);
202  void   wait(jlong millis, bool interruptable, TRAPS);
203  void   notify(TRAPS);
204  void   notifyAll(TRAPS);
205
206  // Use the following at your own risk
207  intptr_t complete_exit(TRAPS);
208  void   reenter(intptr_t recursions, TRAPS);
209
210  private:
211  void   AddWaiter (ObjectWaiter * waiter);
212  static void DeferredInitialize();
```

End of Java Monitor Object: Coordination Methods