Java ReentrantLock
(Part 3)

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 the concept of mutual exclusion in concurrent programs
• Recognize how Java ReentrantLock provides mutual exclusion to concurrent programs
• Know the key methods defined by the Java ReentrantLock class
Overview of Key ReentrantLock Methods
Overview of Key ReentrantLock Methods

- It key methods acquire & release the lock

```java
class ReentrantLock implements Lock, java.io.Serializable {  
  
  public void lock() { sync.lock(); }  
  
  public void lockInterruptibly() throws InterruptedException {  
    sync.acquireInterruptibly(1);  
  }  
  
  public boolean tryLock() {  
    return sync.nonfairTryAcquire(1);  
  }  
  
  public void unlock() {  
    sync.release(1);  
  }  
  
  ...
}
```

See `src/share/classes/java/util/concurrent/locks/ReentrantLock.java`
Overview of Key ReentrantLock Methods

- It key methods acquire & release the lock

```java
public class ReentrantLock implements Lock, java.io.Serializable {
    ...
    public void lock() { sync.lock(); }

    public void lockInterruptibly()
        throws InterruptedException {
        sync.acquireInterruptibly(1);
    }

    public boolean tryLock() {
        return sync.nonfairTryAcquire(1);
    }

    public void unlock() {
        sync.release(1);
    }
    ...
}
```

These methods are defined in the Java Lock interface

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/locks/Lock.html](docs.oracle.com/javase/8/docs/api/java/util/concurrent/locks/Lock.html)
Overview of Key ReentrantLock Methods

• It key methods acquire & release the lock

```java
public class ReentrantLock implements Lock, java.io.Serializable {
    ...
    public void lock() { sync.lock(); }
    public void lockInterruptibly() throws InterruptedException {
        sync.acquireInterruptibly(1);
    }
    public boolean tryLock() {
        return sync.nonfairTryAcquire(1);
    }
    public void unlock() { sync.release(1); }
    ...
}
```

These methods simply forward to their implementor methods, which largely inherit from AbstractQueuedSynchronizer

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/locks/AbstractQueuedSynchronizer.html
Overview of Key ReentrantLock Methods

- It key methods acquire & release the lock
- lock() acquires the lock if it’s available

```java
public class ReentrantLock implements Lock, java.io.Serializable {
    ...
    public void lock() {
        sync.lock();
    }
    ...
}
```
Overview of Key ReentrantLock Methods

- It key methods acquire & release the lock
- lock() acquires the lock if it’s available
- If lock isn’t available its implementation depends on the “fairness” policy

```java
public class ReentrantLock implements Lock, java.io.Serializable {
    ...
    public void lock() {
        sync.lock();
    }
    ...
}
```
Overview of Key ReentrantLock Methods

- It key methods acquire & release the lock
- lock() acquires the lock if it’s available
  - If lock isn’t available its implementation depends on the “fairness” policy
  - Non-fair implementations are optimized in hardware

```java
public class ReentrantLock implements Lock, java.io.Serializable {

  public void lock() {
    sync.lock();
  }

  ...  
```

See en.wikipedia.org/wiki/Spinlock
Overview of Key ReentrantLock Methods

- It key methods acquire & release the lock
  - lock() acquires the lock if it’s available
  - If lock isn’t available its implementation depends on the “fairness” policy
- Non-fair implementations are optimized in hardware
- Fair implementations “park” themselves on a wait queue in FIFO order

```java
public class ReentrantLock implements Lock, java.io.Serializable {
    public class ReentrantLock implements Lock, java.io.Serializable {
        public void lock() {
            sync.lock();
        }
    }
    ...
```
Overview of Key ReentrantLock Methods

- It key methods acquire & release the lock
- `lock()` acquires the lock if it’s available
  - If lock isn’t available its implementation depends on the “fairness” policy
- `lock()` is not interruptible

```java
public class ReentrantLock implements Lock, java.io.Serializable {
    ... 
    public void lock() {
        sync.lock();
    }
    ... 
}
```
Overview of Key ReentrantLock Methods

• It key methods acquire & release the lock
  • `lock()` acquires the lock if it’s available
  • `lockInterruptibly()` acquires lock unless interrupted

```java
public class ReentrantLock
    implements Lock,
    java.io.Serializable {
...
    public void lockInterruptibly()
        throws InterruptedException {
        sync.acquireInterruptibly(1);
    }
...}
```

See lesson on “Managing the Java Thread Lifecycle”
Overview of Key ReentrantLock Methods

- It key methods acquire & release the lock
  - lock() acquires the lock if it’s available
  - lockInterruptibly() acquires lock unless interrupted
  - tryLock() acquires lock only if it’s not held by another thread at invocation time

```
public class ReentrantLock implements Lock,
    java.io.Serializable {
    ...
    public boolean tryLock() {
        sync.nonfairTryAcquire(1);
    }
    ...
```

Untimed tryLock() doesn’t honor fairness setting & can “barge”
Overview of Key ReentrantLock Methods

• It key methods acquire & release the lock
  • lock() acquires the lock if it’s available
  • lockInterruptibly() acquires lock unless interrupted
  • tryLock() acquires lock only if it’s not held by another thread at invocation time
  • unlock() attempts to release the lock

```java
class ReentrantLock implements Lock, java.io.Serializable {  
  ...  
  public void unlock() {  
    sync.release(1);  
  }  
  ...  
```
Overview of Key ReentrantLock Methods

- It key methods acquire & release the lock
  - lock() acquires the lock if it’s available
  - lockInterruptibly() acquires lock unless interrupted
  - tryLock() acquires lock only if it’s not held by another thread at invocation time
  - unlock() attempts to release the lock

```java
public class ReentrantLock implements Lock,
java.io.Serializable {

    ...

    public void unlock() {
        sync.release(1);
    }

    ...
}
```
Overview of Key ReentrantLock Methods

- It key methods acquire & release the lock
  - lock() acquires the lock if it’s available
  - lockInterruptibly() acquires lock unless interrupted
  - tryLock() acquires lock only if it’s not held by another thread at invocation time
  - unlock() attempts to release the lock
    - IllegalMonitorStateException is thrown if calling thread doesn’t hold lock

    ```java
    public class ReentrantLock implements Lock, java.io.Serializable {
        ...
        public void unlock() {
            sync.release(1);
        }
        ...
    }
    ```

i.e., a ReentrantLock is “fully bracketed”!
Overview of Key ReentrantLock Methods

- It key methods acquire & release the lock
  - `lock()` acquires the lock if it’s available
  - `lockInterruptibly()` acquires lock unless interrupted
  - `tryLock()` acquires lock only if it’s not held by another thread at invocation time
  - `unlock()` attempts to release the lock
    - `IllegalMonitorStateException` is thrown if calling thread doesn’t hold lock
    - If hold count > 1 then lock is not released

```java
public class ReentrantLock implements Lock, java.io.Serializable {

    public void unlock() {
        sync.release(1);
    }

    // Other methods...
}
```

See [en.wikipedia.org/wiki/Reentrant_mutex](en.wikipedia.org/wiki/Reentrant_mutex)
Overview of Other ReentrantLock Methods
Overview of Other ReentrantLock Methods

- There are many other ReentrantLock methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean tryLock(long timeout, TimeUnit unit)</td>
<td>Acquires the lock if it is not held by another thread within the given waiting time and the current thread has not been interrupted</td>
</tr>
<tr>
<td>boolean isFair()</td>
<td>Returns true if this lock has fairness set true</td>
</tr>
<tr>
<td>boolean isLocked()</td>
<td>Queries if this lock is held by any thread</td>
</tr>
<tr>
<td>Condition newCondition()</td>
<td>Returns a Condition instance for use with this Lock instance</td>
</tr>
</tbody>
</table>

These methods go above & beyond what’s available from Java’s synchronized statements/methods
Overview of Other ReentrantLock Methods

- There are many other ReentrantLock methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean tryLock(long timeout, TimeUnit unit)</td>
<td>Acquires the lock if it is not held by another thread within the given waiting time and the current thread has not been interrupted</td>
</tr>
<tr>
<td>boolean isFair()</td>
<td>Returns true if this lock has fairness set true</td>
</tr>
<tr>
<td>boolean isLocked()</td>
<td>Queries if this lock is held by any thread</td>
</tr>
<tr>
<td>Condition newCondition()</td>
<td>Returns a Condition instance for use with this Lock instance</td>
</tr>
</tbody>
</table>

Timed tryLock() *does* honor fairness setting & can’t “barge”
### Overview of Other ReentrantLock Methods

- There are many other ReentrantLock methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean tryLock(long timeout, TimeUnit unit)</td>
<td>Acquires the lock if it is not held by another thread within the given waiting time and the current thread has not been interrupted</td>
</tr>
<tr>
<td>boolean isFair()</td>
<td>Returns true if this lock has fairness set true</td>
</tr>
<tr>
<td>boolean isLocked()</td>
<td>Queries if this lock is held by any thread</td>
</tr>
<tr>
<td>Condition newCondition()</td>
<td>Returns a Condition instance for use with this Lock instance</td>
</tr>
</tbody>
</table>

...
Overview of Other ReentrantLock Methods

- There are many other ReentrantLock methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tryLock(long timeout, TimeUnit unit)</td>
<td>Acquires the lock if it is not held by another thread within the given waiting time and the current thread has not been interrupted</td>
</tr>
<tr>
<td>isFair()</td>
<td>Returns true if this lock has fairness set true</td>
</tr>
<tr>
<td>isLocked()</td>
<td>Queries if this lock is held by any thread</td>
</tr>
<tr>
<td>newCondition()</td>
<td>Returns a Condition instance for use with this Lock instance</td>
</tr>
</tbody>
</table>

... Not very useful due to non-determinism of concurrency..
Overview of Other ReentrantLock Methods

- There are many other ReentrantLock methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean tryLock(long timeout, TimeUnit unit)</td>
<td>Acquires the lock if it is not held by another thread within the given waiting time and the current thread has not been interrupted</td>
</tr>
<tr>
<td>boolean isFair()</td>
<td>Returns true if this lock has fairness set true</td>
</tr>
<tr>
<td>boolean isLocked()</td>
<td>Queries if this lock is held by any thread</td>
</tr>
<tr>
<td>Condition newCondition()</td>
<td>Returns a Condition instance for use with this Lock instance</td>
</tr>
</tbody>
</table>

See upcoming lesson on "Java ConditionObject"
End of Java
ReentrantLock (Part 3)