Java ReentrantReadWriteLock:

Key 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 the structure & functionality of the Java ReentrantReadWriteLock class

• Know the key methods in Java ReentrantReadWriteLock
Key Methods in Java
ReentrantReadWriteLock
• writeLock() & readLock() are the key (factory) methods defined by this class

```java
public class ReentrantReadWriteLock
    implements ReadWriteLock ...
{
    ...

    public ReentrantReadWriteLock.
        WriteLock

        writeLock() {
            return writerLock;
        }

    public ReentrantReadWriteLock.
        ReadLock

        readLock() {
            return readerLock;
        }

    ...
```

See en.wikipedia.org/wiki/Factory_method_pattern
writeLock() & readLock() are the key (factory) methods defined by this class

• Returns lock used by clients that want exclusive write access to the lock

```java
public class ReentrantReadWriteLock implements ReadWriteLock {
    ...
    public ReentrantReadWriteLock.
        WriteLock
    writeLock() {
        return writerLock;
    }

    public ReentrantReadWriteLock.
        ReadLock
    readLock() {
        return readerLock;
    }
    ...
```
Key Methods in Java ReentrantReadWriteLock

- `writeLock()` & `readLock()` are the key (factory) methods defined by this class
- Returns lock used by clients that want exclusive write access to the lock
- Returns lock used by clients that want shared read-only access to the lock

```java
public class ReentrantReadWriteLock implements ReadWriteLock {
    ...,
    public ReentrantReadWriteLock.
            WriteLock
        writeLock() {
            return writerLock;
        }
        
    public ReentrantReadWriteLock.
            ReadLock
        readLock() {
            return readerLock;
        }
    ...,
}
writeLock() & readLock() are the key (factory) methods defined by this class

- Returns lock used by clients that want exclusive write access to the lock
- Returns lock used by clients that want shared read-only access to the lock

These objects are initialized by the class constructor
• Locks returned by `writeLock()` & `readLock()` implement the Java Lock interface

```
public class ReentrantReadWriteLock implements ReadWriteLock ...
{
...

public ReentrantReadWriteLock.
  WriteLock

  writeLock() {
    return writerLock;
  }

public ReentrantReadWriteLock.
  ReadLock

  readLock() {
    return readerLock;
  }

...
```

*Readers vs. writer semantics are enforced internally by the class implementation using the Lock API*

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/locks/Lock.html](http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/locks/Lock.html)
**Key Methods in Java ReentrantReadWriteLock**

- It’s methods support a number of properties

<table>
<thead>
<tr>
<th>Key Methods</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reentrancy</td>
<td>This lock allows both readers and writers to reacquire read or write locks in the style of a ReentrantLock. Non-reentrant readers are not allowed until all write locks held by the writing thread have been released. Additionally, a writer can acquire the read lock, but not vice-versa. Among other applications, reentrancy can be useful when write locks are held during calls or callbacks to methods that perform reads under read locks. If a reader tries to acquire the write lock it will never succeed.</td>
</tr>
<tr>
<td>Lock downgrading</td>
<td>Reentrancy also allows downgrading from the write lock to a read lock, by acquiring the write lock, then the read lock and then releasing the write lock. However, upgrading from a read lock to the write lock is not possible.</td>
</tr>
<tr>
<td>Interruption of lock acquisition</td>
<td>The read lock and write lock both support interruption during lock acquisition.</td>
</tr>
<tr>
<td>Condition support</td>
<td>The write lock provides a Condition implementation that behaves in the same way, with respect to the write lock, as the Condition implementation provided by newCondition() does for ReentrantLock. This Condition can, of course, only be used with the write lock. The read lock does not support a Condition and readLock().newCondition() throws UnsupportedOperationException.</td>
</tr>
</tbody>
</table>
Key Methods in Java ReentrantReadWriteLock

- It's methods support a number of properties
  - Reentrancy
    - Enables "recursive lock" semantics for reader-writer locks
  - Lock downgrading
    - Reentrancy also allows downgrading from the write lock to a read lock, by acquiring the write lock, then the read lock and then releasing the write lock. However, upgrading from a read lock to the write lock is not possible.
  - Interruption of lock acquisition
    - The read lock and write lock both support interruption during lock acquisition.
  - Condition support
    - The write lock provides a Condition implementation that behaves in the same way, with respect to the write lock, as the Condition implementation provided by newCondition() does for ReentrantLock. This Condition can, of course, only be used with the write lock.
    - The read lock does not support a Condition and readLock().newCondition() throws UnsupportedOperationException.

This property is not supported by Java StampedLock
Key Methods in Java ReentrantReadWriteLock

- It’s methods support a number of properties
  - Reentrancy
  - Lock downgrading
    - Enables atomic downgrading of a write lock to a read lock

- Reentrancy
  This lock allows both readers and writers to reacquire read or write locks in the style of a ReentrantLock. Non-reentrant readers are not allowed until all write locks held by the writing thread have been released.

  Additionally, a writer can acquire the read lock, but not vice-versa. Among other applications, reentrancy can be useful when write locks are held during calls or callbacks to methods that perform reads under read locks. If a reader tries to acquire the write lock it will never succeed.

- Lock downgrading
  Reentrancy also allows downgrading from the write lock to a read lock, by acquiring the write lock, then the read lock and then releasing the write lock. However, upgrading from a read lock to the write lock is not possible.

- Interruption of lock acquisition
  The read lock and write lock both support interruption during lock acquisition.

- Condition support
  The write lock provides a Condition implementation that behaves in the same way, with respect to the write lock, as the Condition implementation provided by newCondition() does for ReentrantLock. This Condition can, of course, only be used with the write lock.

  The read lock does not support a Condition and readLock().newCondition() throws UnsupportedOperationException.
Key Methods in Java ReentrantReadWriteLock

- It’s methods support a number of properties
  - Reentrancy
  - Lock downgrading
  - Interruption of lock acquisition
  - Conventional Java interrupt requests are supported

- **Reentrancy**
  This lock allows both readers and writers to reacquire read or write locks in the style of a `ReentrantLock`. Non-reentrant readers are not allowed until all write locks held by the writing thread have been released.

  Additionally, a writer can acquire the read lock, but not vice-versa. Among other applications, reentrancy can be useful when write locks are held during calls or callbacks to methods that perform reads under read locks. If a reader tries to acquire the write lock it will never succeed.

- **Lock downgrading**
  Reentrancy also allows downgrading from the write lock to a read lock, by acquiring the write lock, then the read lock and then releasing the write lock. However, upgrading from a read lock to the write lock is not possible.

- **Interruption of lock acquisition**
  The read lock and write lock both support interruption during lock acquisition.

- **Condition support**
  The write lock provides a `Condition` implementation that behaves in the same way, with respect to the write lock, as the `Condition` implementation provided by `newCondition()` does for `ReentrantLock`. This `Condition` can, of course, only be used with the write lock.

  The read lock does not support a `Condition` and `readLock().newCondition()` throws `UnsupportedOperationException`.

This property is supported by Java StampedLock
Key Methods in Java ReentrantReadWriteLock

- It’s methods support a number of properties
  - Reentrancy
  - Lock downgrading
  - Interruption of lock acquisition
- Condition support
  - Enables the use of Java ReentrantReadWriteLocks with Java ConditionObjects only for write locks

Reentrancy
This lock allows both readers and writers to reacquire read or write locks in the style of a ReentrantLock. Non-reentrant readers are not allowed until all write locks held by the writing thread have been released.

Additionally, a writer can acquire the read lock, but not vice-versa. Among other applications, reentrancy can be useful when write locks are held during calls or callbacks to methods that perform reads under read locks. If a reader tries to acquire the write lock it will never succeed.

Lock downgrading
Reentrancy also allows downgrading from the write lock to a read lock, by acquiring the write lock, then the read lock and then releasing the write lock. However, upgrading from a read lock to the write lock is not possible.

Interruption of lock acquisition
The read lock and write lock both support interruption during lock acquisition.

Condition support
The write lock provides a Condition implementation that behaves in the same way, with respect to the write lock, as the Condition implementation provided by newCondition() does for ReentrantLock. This Condition can, of course, only be used with the write lock.

The read lock does not support a Condition and readLock().newCondition() throws UnsupportedOperationException.

This property is not supported by Java StampedLock
Key Methods in Java ReentrantReadWriteLock

- It’s methods support a number of properties

Reentrancy

This lock allows both readers and writers to reacquire read or write locks in the style of a ReentrantLock. Non-reentrant readers are not allowed until all write locks held by the writing thread have been released.

Additionally, a writer can acquire the read lock, but not vice-versa. Among other applications, reentrancy can be useful when write locks are held during calls or callbacks to methods that perform reads under read locks. If a reader tries to acquire the write lock it will never succeed.

Lock downgrading

Reentrancy also allows downgrading from the write lock to a read lock, by acquiring the write lock, then the read lock and then releasing the write lock. However, upgrading from a read lock to the write lock is not possible.

Interruption of lock acquisition

The read lock and write lock both support interruption during lock acquisition.

Condition support

The write lock provides a Condition implementation that behaves in the same way, with respect to the write lock, as the Condition implementation provided by newCondition() does for ReentrantLock. This Condition can, of course, only be used with the write lock.

The read lock does not support a Condition and readLock().newCondition() throws UnsupportedOperationException.

These properties make optimizing ReentrantReadWriteLock hard (& motivates the need for Java StampedLock)
End of Java ReentrantReadWriteLock: Key Methods