Java ReentrantReadWriteLock: Structure & Functionality

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.
Overview of ReentrantReadWriteLock
Overview of Java ReentrantReadWriteLock

• Provide a Java readers-writer lock implementation

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

### Class ReentrantReadWriteLock

```java
java.lang.Object
  java.util.concurrent.locks.ReentrantReadWriteLock

All Implemented Interfaces:
  Serializable, ReadWriteLock
```

```java
public class ReentrantReadWriteLock
extends Object
implements Object ReadWriteLock, Serializable
```

An implementation of `ReadWriteLock` supporting similar semantics to `ReentrantLock`.

This class has the following properties:

- Acquisition order

  This class does not impose a reader or writer preference ordering for lock access. However, it does support an optional fairness policy.

**Non-fair mode (default)**

When constructed as non-fair (the default), the order of entry to the read and write lock is unspecified, subject to reentrancy constraints. A nonfair lock that is continuously contended may indefinitely postpone one or more reader or writer threads, but will normally have higher throughput than a fair lock.

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/locks/ReentrantReadWriteLock.html](docs.oracle.com/javase/8/docs/api/java/util/concurrent/locks/ReentrantReadWriteLock.html)
Overview of Java ReentrantReadWriteLock

- Provide a Java readers-writer lock implementation
- Implements the ReadWriteLock interface

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

### Interface ReadWriteLock

**All Known Implementing Classes:**
- ReentrantReadWriteLock

**public interface ReadWriteLock**

A `ReadWriteLock` maintains a pair of associated `locks`, one for read-only operations and one for writing. The read lock may be held simultaneously by multiple reader threads, so long as there are no writers. The write lock is exclusive.

All `ReadWriteLock` implementations must guarantee that the memory synchronization effects of `writeLock` operations (as specified in the `Lock` interface) also hold with respect to the associated `readLock`. That is, a thread successfully acquiring the read lock will see all updates made upon previous release of the write lock.

A read-write lock allows for a greater level of concurrency in accessing shared data than that permitted by a mutual exclusion lock. It exploits the fact that while only a single thread at a time (a `writer` thread) can modify the shared data, in many cases any number of threads can concurrently read the data (hence `reader` threads). In theory, the increase in concurrency permitted by the use of a read-write lock will lead to performance improvements over the use of a mutual exclusion lock. In practice this increase in concurrency will only be fully realized on a multi-processor, and then only if the access patterns for the shared data are suitable.

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/locks/ReadWriteLock.html](docs.oracle.com/javase/8/docs/api/java/util/concurrent/locks/ReadWriteLock.html)
Overview of Java ReentrantReadWriteLock

- Provide a Java readers-writer lock implementation
- Implements the ReadWriteLock interface
- Nested ReadLock & WriteLock classes implement Lock interface

```java
public class ReentrantReadWriteLock implements ReadWriteLock {
    ...
    /** Inner class providing readlock */
    ReentrantReadWriteLock.ReadLock readerLock;
    ...
    /** Inner class providing writelock */
    ReentrantReadWriteLock.WriteLock writerLock;
    ...
}
```

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 Java ReentrantReadWriteLock

- Implements readers-writer semantics

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

    /** Inner class providing readlock */
    ReentrantReadWriteLock.ReadLock readerLock;

    /** Inner class providing writelock */
    ReentrantReadWriteLock.WriteLock writerLock;

    ...

    Multiple reader threads can run concurrently within a critical section
```
Overview of Java ReentrantReadWriteLock

- Implements readers-writer semantics

```java
public class ReentrantReadWriteLock implements ReadWriteLock ... {
  ...
  /** Inner class providing readlock */
  ReentrantReadWriteLock.ReadLock readerLock;

  /** Inner class providing writelock */
  ReentrantReadWriteLock.WriteLock writerLock;
  ...
```

Only one writer thread can run at a time within a critical section
Overview of Java ReentrantReadWriteLock

- Implements readers-writer semantics

```java
public class ReentrantReadWriteLock implements ReadWriteLock ...
{
    ...
    /** Inner class providing readlock */
    ReentrantReadWriteLock.ReadLock readerLock;
    
    /** Inner class providing writelock */
    ReentrantReadWriteLock.WriteLock writerLock;
    ...
}
```

ReentrantReadWriteLock is “pessimistic”, i.e., it assumes contention may occur
Overview of Java ReentrantReadWriteLock

- Applies the Bridge pattern

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

Decouple interface from implementation so that fair & non-fair readers-writer semantics can be supported uniformly

See [en.wikipedia.org/wiki/Bridge_pattern](en.wikipedia.org/wiki/Bridge_pattern)
Overview of Java ReentrantReadWriteLock

- Applies the *Bridge* pattern
- Locking handled by Sync implementor hierarchy

```java
public class ReentrantReadWriteLock
    implements ReadWriteLock ... {
    ...
    /** Performs sync mechanics */
    final Sync sync;
    ...
```
Overview of Java ReentrantReadWriteLock

- Applies the *Bridge* pattern
- Locking handled by Sync implementor hierarchy
- Inherits functionality from AbstractQueuedSynchronizer

```
public class ReentrantReadWriteLock
    implements ReadWriteLock ... {
    ...
    /** Performs sync mechanics */
    final Sync sync;

    /** Sync implementation for ReentrantReadWriteLock */
    abstract static class Sync extends AbstractQueuedSynchronizer
    {
        ... }
    ... }
```

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/locks/AbstractQueuedSynchronizer.html](docs.oracle.com/javase/8/docs/api/java/util/concurrent/locks/AbstractQueuedSynchronizer.html)
Overview of Java ReentrantReadWriteLock

- Applies the *Bridge* pattern
- Locking handled by Sync implementor hierarchy
- Inherits functionality from AbstractQueuedSynchronizer
- Many Java synchronizers based on FIFO wait queues use this framework

```
public class ReentrantReadWriteLock
    implements ReadWriteLock ... {
...
    /** Performs sync mechanics */
    final Sync sync;

    /** Sync implementation for ReentrantReadWriteLock */
    abstract static class Sync extends
        AbstractQueuedSynchronizer
    {
        ... 
    }
...
```

See [gee.cs.oswego.edu/dl/papers/aqs.pdf](gee.cs.oswego.edu/dl/papers/aqs.pdf)
Applies the *Bridge* pattern

Locking handled by Sync implementor hierarchy

Inherits functionality from `AbstractQueuedSynchronizer`

Defines `NonFairSync` & `FairSync` subclasses with non-FIFO & FIFO semantics

```java
public class ReentrantReadWriteLock
    implements ReadWriteLock ...
{
    ...
    /** Performs sync mechanics */
    final Sync sync;

    /** Sync implementation for ReentrantReadWriteLock */
    abstract static class Sync extends 
        AbstractQueuedSynchronizer
    {
        ...
    }

    static final class NonFairSync
        extends Sync { ...
    }

    static final class FairSync
        extends Sync { ...
    }

    See src/share/classes/java/util/concurrent/locks/ReentrantReadWriteLock.java
```
Overview of Java ReentrantReadWriteLock

- Applies the *Bridge* pattern
- Locking handled by Sync implementor hierarchy
- Constructor enables fair vs. non-fair lock acquisition model
- These models apply the same pattern used by ReentrantLock & Semaphore

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

    public ReentrantReadWriteLock (boolean fair) {
        sync = fair ? new FairSync()
                : new NonfairSync();
        readerLock =
            new ReadLock(this);
        writerLock =
            new WriteLock(this);
    }

    ...
```

See earlier lessons on "Java ReentrantLock" & "Java Semaphore"
Overview of Java ReentrantReadWriteLock

- Applies the *Bridge* pattern
- Locking handled by Sync implementor hierarchy
- Constructor enables fair vs. non-fair lock acquisition model
- These models apply the same pattern used by ReentrantLock & Semaphore

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

    public ReentrantReadWriteLock
        (boolean fair) {
        sync = fair ? new FairSync()
                    : new NonfairSync();
        readerLock =
            new ReadLock(this);
        writerLock =
            new WriteLock(this);
    }
    ...
    
    This param determines whether FairSync or NonfairSync is used
```

See earlier lessons on “Java Semaphore” & “Java ReentrantLock”
Ensures strict "FIFO" fairness, at the expense of performance

• Applies the Bridge pattern
• Locking handled by Sync implementor hierarchy
• Constructor enables fair vs. non-fair lock acquisition model
• These models apply the same pattern used by ReentrantLock & Semaphore

```java
public class ReentrantReadWriteLock
    implements ReadWriteLock ... {
    ...
    public ReentrantReadWriteLock
        (boolean fair) {
        sync = fair ? new FairSync()
                        : new NonfairSync();
        readerLock =
            new ReadLock(this);
        writerLock =
            new WriteLock(this);
    }
    ...
```

Overview of Java ReentrantReadWriteLock
• Applies the *Bridge* pattern
• Locking handled by Sync implementor hierarchy
• Constructor enables fair vs. non-fair lock acquisition model
• These models apply the same pattern used by ReentrantLock & Semaphore

```
public class ReentrantReadWriteLock
    implements ReadWriteLock ... {
    ...,
    public ReentrantReadWriteLock
        (boolean fair) {
        sync = fair ? new FairSync()
                     : new NonfairSync();
        readerLock = new ReadLock(this);
        writerLock = new WriteLock(this);
    }
    ...
```
Overview of Java ReentrantReadWriteLock

- Applies the *Bridge* pattern
- Locking handled by Sync implementor hierarchy
- Constructor enables fair vs. non-fair lock acquisition model
- These models apply the same pattern used by ReentrantLock & Semaphore

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

    public ReentrantReadWriteLock (boolean fair) {
        sync = fair ? new FairSync() : new NonfairSync();
        readerLock = new ReadLock(this);
        writerLock = new WriteLock(this);
    }

    ... // Further code
```

*FairSync is generally much slower than NonfairSync, so use it accordingly*
The default constructor therefore uses the faster non-fair semantics.

The Bridge pattern

- Locking handled by Sync implementor hierarchy
- Constructor enables fair vs. non-fair lock acquisition model
- These models apply the same pattern used by ReentrantLock & Semaphore

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

    public ReentrantReadWriteLock( boolean fair ) {
        sync = fair ? new FairSync() :
                   new NonfairSync();
        readerLock = new ReadLock(this);
        writerLock = new WriteLock(this);
    }

    public ReentrantReadWriteLock() {
        sync = new NonfairSync();
    }
}
```
Overview of Java ReentrantReadWriteLock

- Applies the *Bridge* pattern
- Locking handled by Sync implementor hierarchy
- Constructor enables fair vs. non-fair lock acquisition model
- These models apply the same pattern used by ReentrantLock & Semaphore
- Initializes the readerLock & writerLock field

```java
class ReentrantReadWriteLock
    implements ReadWriteLock ...
{
    ...
    public ReentrantReadWriteLock
    (boolean fair) {
        sync = fair ? new FairSync()
                   : new NonfairSync();
        readerLock =
                     new ReadLock(this);
        writerLock =
                      new WriteLock(this);
    }
    ...
```
Overview of Java ReentrantReadWriteLock

- Applies the *Bridge* pattern
- Locking handled by Sync Implementor hierarchy
- Constructor enables fair vs. non-fair lock acquisition model
- These models apply the same pattern used by ReentrantLock & Semaphore
- Initializes the readerLock & writerLock field
- WriteLock & ReadLock use “shared” mode of Abstract QueuedSynchronizer

```java
class ReentrantReadWriteLock implements ReadWriteLock {
    ...
    public ReentrantReadWriteLock (boolean fair) {
        sync = fair ? new FairSync () :
                    new NonfairSync ();
        readerLock =
                    new ReadLock (this);
        writerLock =
                    new WriteLock (this);
    }
    ...

    public static class WriteLock implements Lock {
        ...
    }

    public static class ReadLock implements Lock {
        ...
    }
```
• Applies the *Bridge* pattern
• Locking handled by Sync Implementor hierarchy
• Constructor enables fair vs. non-fair lock acquisition model
• These models apply the same pattern used by ReentrantLock & Semaphore
• Initializes the readerLock & writerLock field
• WriteLock & ReadLock use “shared” mode of Abstract QueuedSynchronizer
• Also implement the Lock interface w/methods like lock(), tryLock(), & unlock()

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

    public ReentrantReadWriteLock (boolean fair) {
        sync = fair ? new FairSync() : new NonfairSync();
        readerLock =
            new ReadLock(this);
        writerLock =
            new WriteLock(this);
    }
    ...

    public static class WriteLock implements Lock ... { ... }

    public static class ReadLock implements Lock ... { ... }
```

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/locks/Lock.html](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/locks/Lock.html)
End of Java ReentrantRead
WriteLock: Structure & Functionality