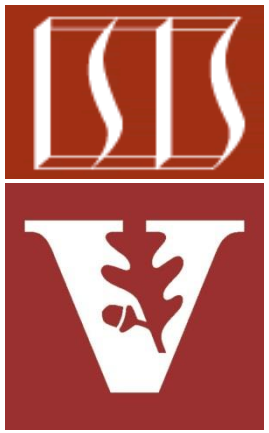
Java ReentrantReadWriteLock: Example Application



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
- Recognize how to apply Java ReentrantReadWriteLock in practice

```
class SimpleAtomicLong {
  private long mValue;

  private ReentrantReadWriteLock
          mRWLock = new
             ReentrantReadWriteLock();


  ...
```

Applying the Java ReentrantReadWriteLock

Applying the Java ReentrantReadWriteLock

- The SimpleAtomicLong class shows how to program with ReentrantReadWriteLock

```
class SimpleAtomicLong {
  private long mValue;


  private ReentrantReadWriteLock
          mRWLock = new
             ReentrantReadWriteLock();
  ...
```

Applying the Java ReentrantReadWriteLock

- The SimpleAtomicLong class shows how to program with ReentrantReadWriteLock

```
class SimpleAtomicLong {
  private long mValue;

  private ReentrantReadWriteLock
          mRWLock = new
             ReentrantReadWriteLock();
  ...
```

Java AtomicLong actually uses "compare-and-swap"

See src/share/classes/java/util/concurrent/atomic/AtomicLong.java

Applying the Java ReentrantReadWriteLock

- The SimpleAtomicLong class shows how to program with ReentrantReadWriteLock

```
class SimpleAtomicLong {
  private long mValue;

  private ReentrantReadWriteLock
          mRWLock = new
             ReentrantReadWriteLock();
  ...
```

The value written to & read from (which is not atomic by default)

See dzone.com/articles/longdouble-are-not-atomic-in-java

Applying the Java ReentrantReadWriteLock

- The SimpleAtomicLong class shows how to program with ReentrantReadWriteLock

```
class SimpleAtomicLong {
  private long mValue;

  private ReentrantReadWriteLock
  mRWLock = new
     ReentrantReadWriteLock();

  ...
```

The ReentrantReadWriteLock that serializes access to mValue

There's no need to use "fair" lock semantics here

Applying the Java ReentrantReadWriteLock

- The SimpleAtomicLong class shows how to program with ReentrantReadWriteLock

```
class SimpleAtomicLong {
  private long mValue;
  ...

  public SimpleAtomicLong(long init) {
    mValue = init;
  }
  ...
```

Constructor initializes the mValue field

This constructor needs no lock since it's only called once in a single thread!

Applying the Java ReentrantReadWriteLock

- The SimpleAtomicLong class shows how to program with ReentrantReadWriteLock

```
class SimpleAtomicLong {
  private long mValue;
  ...

  public long incrementAndGet() {
    mRWLock.writeLock().lock();
    try {
      return ++mValue;
    } finally {
      mRWLock.writeLock().unlock();
    }
  }
  ...
```

This method writes mValue atomically

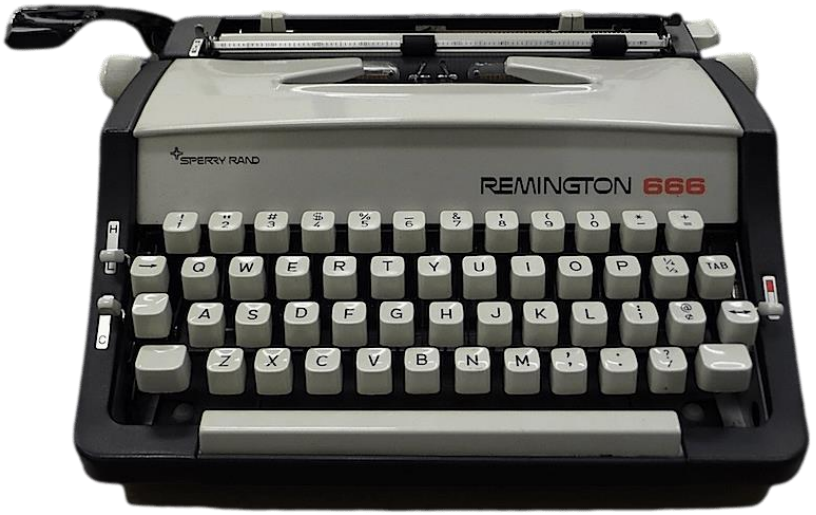
Applying the Java ReentrantReadWriteLock

- The SimpleAtomicLong class shows how to program with ReentrantReadWriteLock

Atomically acquire the write-lock (blocking if necessary) & increment the current mValue

```
class SimpleAtomicLong {
  private long mValue;
  ...

  public long incrementAndGet() {
    mRWLock.writeLock().lock();
    try {
      return ++mValue;
    } finally {
      mRWLock.writeLock().unlock();
    }
  }
  ...
```

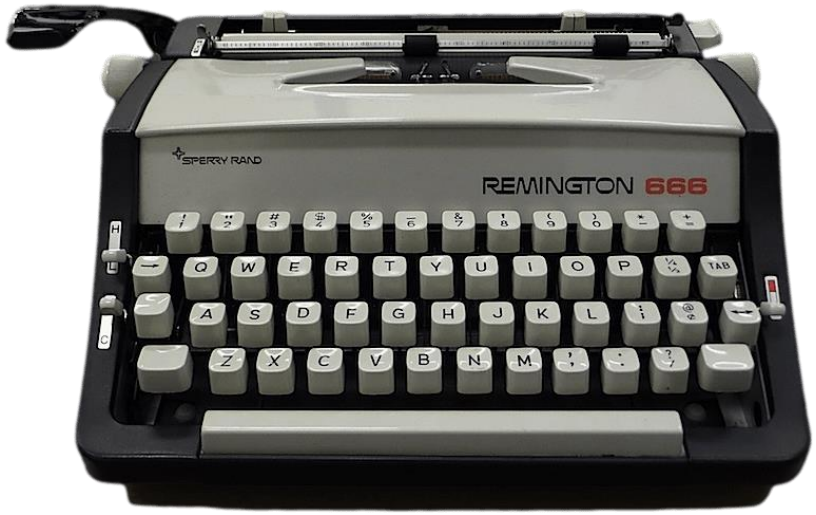


Applying the Java ReentrantReadWriteLock

- The SimpleAtomicLong class shows how to program with ReentrantReadWriteLock


```
class SimpleAtomicLong {
  private long mValue;
  ...

  public long incrementAndGet() {
    mRWLock.writeLock().lock();
    try {
      return ++mValue;
    } finally {
      mRWLock.writeLock().unlock();
    }
  }
  ...
```



A write-lock is "pessimistic" since it assumes contention may occur, so no other thread can acquire the lock while it's held, i.e., a write lock is "exclusive"

Applying the Java ReentrantReadWriteLock

- The SimpleAtomicLong class shows how to program with ReentrantReadWriteLock

```
class SimpleAtomicLong {
  private long mValue;
  ...

  public long incrementAndGet() {
    mRWLock.writeLock().lock();
    try {
      return ++mValue;
    } finally {
      mRWLock.writeLock().unlock();
    }
  }
  ...
```

The "try/finally" idiom ensures the lock is always released

See docs.oracle.com/javase/tutorial/essential/exceptions/finally.html

Applying the Java ReentrantReadWriteLock

- The SimpleAtomicLong class shows how to program with ReentrantReadWriteLock

```
class SimpleAtomicLong {
  private long mValue;
  ...

  public long get() {
    mRWLock.readLock().lock();
    try {
      return mValue;
    } finally {
      mRWLock.readLock().unlock();
    }
  }
  ...
```

This method reads mValue atomically

Applying the Java ReentrantReadWriteLock

- The SimpleAtomicLong class shows how to program with ReentrantReadWriteLock

Atomically acquire the read-lock (blocking if necessary) & return current mValue

```
class SimpleAtomicLong {
  private long mValue;
  ...

  public long get() {
    mRWLock.readLock().lock();
    try {
      return mValue;
    } finally {
      mRWLock.readLock().unlock();
    }
  }
  ...
```

Applying the Java ReentrantReadWriteLock

- The SimpleAtomicLong class shows how to program with ReentrantReadWriteLock


```
class SimpleAtomicLong {
  private long mValue;
  ...

  public long get() {
    mRWLock.readLock().lock();
    try {
      return mValue;
    } finally {
      mRWLock.readLock().unlock();
    }
  }
  ...
```

A read-lock is also "pessimistic" since it assumes contention may occur, though other threads can acquire the lock for reading, i.e., a read lock is "shared"

Applying the Java ReentrantReadWriteLock

- The SimpleAtomicLong class shows how to program with ReentrantReadWriteLock

```
class SimpleAtomicLong {
  private long mValue;
  ...

  public long get() {
    mRWLock.readLock().lock();
    try {
      return mValue;
    } finally {
      mRWLock.readLock().unlock();
    }
  }
  ...
```

The "try/finally" idiom ensures the lock is always released

See docs.oracle.com/javase/tutorial/essential/exceptions/finally.html

Applying the Java ReentrantReadWriteLock

- The SimpleAtomicLong class shows how to program with ReentrantReadWriteLock
  - "Lock downgrading" example

```
class SimpleAtomicLong {
  ...
  public long getAndIncrement() {
    long value = 0;
    Lock lock = mRWLock.writeLock();
    lock.lock();
    try {
      mValue++;
      final Lock readLock =
        mRWLock.readLock();
      readLock.lock();
      try {
        lock.unlock();
        value = mValue;
      } finally { lock = readLock; }
    } finally {
      lock.unlock();
    }
    return value - 1;
  }
```

See word-bits.flurg.com/safely-downgrading-a-write-lock-with-readwritelock

Applying the Java ReentrantReadWriteLock

- The SimpleAtomicLong class shows how to program with ReentrantReadWriteLock
  - "Lock downgrading" example

First obtain a write-lock

```
class SimpleAtomicLong {
  ...
  public long getAndIncrement() {
    long value = 0;
    Lock lock = mRWLock.writeLock();
    lock.lock();
    try {
      mValue++;
      final Lock readLock =
        mRWLock.readLock();
      readLock.lock();
      try {
        lock.unlock();
        value = mValue;
      } finally { lock = readLock; }
    } finally {
      lock.unlock();
    }
    return value - 1;
  }
```

Applying the Java ReentrantReadWriteLock

- The SimpleAtomicLong class shows how to program with ReentrantReadWriteLock
  - "Lock downgrading" example

Atomically increment mValue with the write-lock held

```
class SimpleAtomicLong {
  ...
  public long getAndIncrement() {
    long value = 0;
    Lock lock = mRWLock.writeLock();
    lock.lock();
    try {
      mValue++;
      final Lock readLock =
        mRWLock.readLock();
      readLock.lock();
      try {
        lock.unlock();
        value = mValue;
      } finally { lock = readLock; }
    } finally {
      lock.unlock();
    }
    return value - 1;
  }
```

Applying the Java ReentrantReadWriteLock

- The SimpleAtomicLong class shows how to program with ReentrantReadWriteLock
  - "Lock downgrading" example

```
class SimpleAtomicLong {
  ...
  public long getAndIncrement() {
    long value = 0;
    Lock lock = mRWLock.writeLock();
    lock.lock();
    try {
      mValue++;
      final Lock readLock =
        mRWLock.readLock();
      readLock.lock();
      try {
        lock.unlock();
        value = mValue;
      } finally { lock = readLock; }
    } finally {
      lock.unlock();
    }
    return value - 1;
  }
```

Next downgrade the write-lock to a read-lock

Applying the Java ReentrantReadWriteLock

- The SimpleAtomicLong class shows how to program with ReentrantReadWriteLock
  - "Lock downgrading" example

```
class SimpleAtomicLong {
  ...
  public long getAndIncrement() {
    long value = 0;
    Lock lock = mRWLock.writeLock();
    lock.lock();
    try {
      mValue++;
      final Lock readLock =
        mRWLock.readLock();
      readLock.lock();
      try {
        lock.unlock();
        value = mValue;
      } finally { lock = readLock; }
    } finally {
      lock.unlock();
    }
    return value - 1;
  }
```

Unlock write-lock & read the mValue with read-lock still held

Other readers threads can now access this value, but any writer threads must wait

Applying the Java ReentrantReadWriteLock

- The SimpleAtomicLong class shows how to program with ReentrantReadWriteLock
  - "Lock downgrading" example

```
class SimpleAtomicLong {
  ...
  public long getAndIncrement() {
    long value = 0;
    Lock lock = mRWLock.writeLock();
    lock.lock();
    try {
      mValue++;
      final Lock readLock =
        mRWLock.readLock();
      readLock.lock();
      try {
        lock.unlock();
        value = mValue;
      } finally { lock = readLock; }
    } finally {
      lock.unlock();
    }
    return value - 1;
  }
```

Release the proper lock

Applying the Java ReentrantReadWriteLock

- The SimpleAtomicLong class shows how to program with ReentrantReadWriteLock
  - "Lock downgrading" example

```
class SimpleAtomicLong {
  ...
  public long getAndIncrement() {
    long value = 0;
    Lock lock = mRWLock.writeLock();
    lock.lock();
    try {
      mValue++;
      final Lock readLock =
        mRWLock.readLock();
      readLock.lock();
      try {
        lock.unlock();
        value = mValue;
      } finally { lock = readLock; }
    } finally {
      lock.unlock();
    }
    return value - 1;
  }
```

Return the original (non-incremented) value

No need to lock 'value' since it's local to the activation record of the thread's stack!

Applying the Java ReentrantReadWriteLock

- The SimpleAtomicLong class shows how to program with ReentrantReadWriteLock
  - "Lock downgrading" example


```
class SimpleAtomicLong {
  ...
  public long getAndIncrement() {
    long value = 0;
    Lock lock = mRWLock.writeLock();
    lock.lock();
    try {
      mValue++;
      final Lock readLock =
        mRWLock.readLock();
      readLock.lock();
      try {
        lock.unlock();
        value = mValue;
      } finally { lock = readLock; }
    } finally {
      lock.unlock();
    }
    return value - 1;
  }
```

Lock downgrading is overkill for the SimpleAtomicLong!

End of Java ReentrantReadWriteLock: Example Application