Java Readers/Writer Locks

(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 the benefits that readers-writer synchronizers provide to concurrent programs

• Know the key methods in Java 8 StampedLock

• Recognize how to apply Java 8 StampedLock in practice

class Point {
    void moveIfAtOrigin(double newX, double newY) {
        long stamp = sl.readLock();
        try {
            while (x == 0.0 && y == 0.0) {
                long ws = sl.tryConvertToWriteLock(stamp);
                if (ws != 0L) {
                    stamp = ws;
                    x = newX; y = newY;
                    break;
                } else {
                    sl.unlockRead(stamp);
                    stamp = sl.writeLock();
                }
            }
        } else {
            sl.unlockRead(stamp);
            stamp = sl.writeLock();
        }
    }
    ...
}
Simple Example of StampedLock
Simple Example of StampedLock

- The Point class shows how to program with StampedLock

```java
class Point {

    private double x;
    private double y;

    private final StampedLock sl =
        new StampedLock();

    ...  
```

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/locks/StampedLock.html](docs.oracle.com/javase/8/docs/api/java/util/concurrent/locks/StampedLock.html)
The Point class shows how to program with StampedLock

```java
class Point {
    private double x;
    private double y;

    private final StampedLock sl =
        new StampedLock();
    ...
}
```

Maintains two-dimensional points
The Point class shows how to program with StampedLock

```java
class Point {
    private double x;
    private double y;

    private final StampedLock sl =
        new StampedLock();

    ...
}
```

State that must be protected
Simple Example of StampedLock

• The Point class shows how to program with StampedLock

```java
class Point {
    private double x;
    private double y;
    private final StampedLock sl = new StampedLock();
    ...
```

StampedLock that will do the protecting
Simple Example of StampedLock

- The Point class shows how to program with StampedLock

```java
class Point {
    ...
    \[ An \textbf{exclusively locked method} \]
     void move(double deltaX, double deltaY) {
        long stamp = sl.writeLock();
        try {
            x += deltaX;
            y += deltaY;
        } finally {
            sl.unlockWrite(stamp);
        }
    }
    ...
}
```
Simple Example of StampedLock

- The Point class shows how to program with StampedLock

```java
class Point {
    ...

    void move(double deltaX,
               double deltaY) {
        long stamp = sl.writeLock();
        try {
            x += deltaX;
            y += deltaY;
        } finally {
            sl.unlockWrite(stamp);
        }
    }
    ...
```
The Point class shows how to program with StampedLock

Simple Example of StampedLock

class Point {
    ...

    void move(double deltaX, double deltaY) {
        long stamp = sl.writeLock();
        try {
            x += deltaX;  // Modify the state
            y += deltaY;
        } finally {
            sl.unlockWrite(stamp);
        }
    }
    ...
}
The Point class shows how to program with StampedLock

```java
class Point {
    ...

    void move(double deltaX, double deltaY) {
        long stamp = sl.writeLock();
        try {
            x += deltaX;
            y += deltaY;
        } finally {
            sl.unlockWrite(stamp);
        }
    }
    ...
}
```

Release the write lock
The Point class shows how to program with StampedLock

```java
class Point {
    ...

    double distanceFromOrigin() {
        long stamp = sl.tryOptimisticRead();
        double currX = x, currY = y;
        if (!sl.validate(stamp)) {
            stamp = sl.readLock();
            try {
                currX = x; currY = y;
            } finally {
                sl.unlockRead(stamp);
            }
        }
        return Math.sqrt(currX * currX + currY * currY);
    }
    ...
```

A read-only method
The Point class shows how to program with StampedLock

class Point {
    ...
    double distanceFromOrigin() {
        long stamp = sl.tryOptimisticRead();
        double currX = x, currY = y;
        if (!sl.validate(stamp)) {
            stamp = sl.readLock();
            try {
                currX = x; currY = y;
            } finally {
                sl.unlockRead(stamp);
            }
        }
        return Math.sqrt(currX * currX + currY * currY);
    }
    ...
}
The Point class shows how to program with StampedLock

```java
class Point {
    ...
    double distanceFromOrigin() {
        long stamp = sl.tryOptimisticRead();
        double currX = x, currY = y;
        if (!sl.validate(stamp)) {
            stamp = sl.readLock();
            try {
                currX = x; currY = y;
            } finally {
                sl.unlockRead(stamp);
            }
        }
        return Math.sqrt(currX * currX + currY * currY);
    }
    ...
}
```

"Optimistically" read state into local variables

Optimistic read sections typically read fields & hold them in local variables for later use after validation
Simple Example of StampedLock

• The Point class shows how to program with StampedLock

class Point {
    ... 
    double distanceFromOrigin() { 
        long stamp = sl.tryOptimisticRead();
        double currX = x, currY = y;
        if (!sl.validate(stamp)) {
            stamp = sl.readLock();
            try {
                currX = x; currY = y;
            } finally
            { sl.unlockRead(stamp); }
        }
        return Math.sqrt (currX * currX + currY * currY);
    }
    ...
The Point class shows how to program with StampedLock

```java
class Point {
    ...

double distanceFromOrigin() {
    long stamp = sl.tryOptimisticRead();
    double currX = x, currY = y;
    if (!sl.validate(stamp)) {
        stamp = sl.readLock();
        try {
            currX = x; currY = y;
        } finally {
            sl.unlockRead(stamp);
        }
    }
    return Math.sqrt(currX * currX + currY * currY);
}
...
The Point class shows how to program with StampedLock

```java
Simple Example of StampedLock

class Point {
    ...
    double distanceFromOrigin() {
        long stamp = sl.tryOptimisticRead();
        double currX = x, currY = y;
        if (!sl.validate(stamp)) {
            stamp = sl.readLock();
            try {
                currX = x; currY = y;     // Do “pessimistic” reads
            } finally {
                sl.unlockRead(stamp);
            }
        }
        return Math.sqrt(currX * currX + currY * currY);
    }
    ...
```
The Point class shows how to program with StampedLock

class Point {
    ...  
    double distanceFromOrigin() {
        long stamp = sl.tryOptimisticRead();
        double currX = x, currY = y;
        if (!sl.validate(stamp)) {
            stamp = sl.readLock();
            try {
                currX = x; currY = y;
            } finally {
                sl.unlockRead(stamp);
            }
        }
        return Math.sqrt(currX * currX + currY * currY);
    }
    ...  
}
The Point class shows how to program with StampedLock

class Point {
    ...
    double distanceFromOrigin() {
        long stamp = sl.tryOptimisticRead();
        double currX = x, currY = y;
        if (!sl.validate(stamp)) {
            stamp = sl.readLock();
            try {
                currX = x; currY = y;
            } finally {
                sl.unlockRead(stamp);
            }
        } No lock to release if validate() succeeded
        return Math.sqrt(currX * currX + currY * currY);
    }
    ...
}
The Point class shows how to program with StampedLock

class Point {

  ...

  double distanceFromOrigin() {
    long stamp = sl.tryOptimisticRead();
    double currX = x, currY = y;
    if (!sl.validate(stamp)) {
      stamp = sl.readLock();
      try {
        currX = x; currY = y;
      } finally {
        sl.unlockRead(stamp);
      }
    }
    return Math.sqrt(currX * currX + currY * currY);
  }

  ...

The Point class shows how to program with StampedLock

```java
class Point {
    ...
    void moveIfAtOrigin(double newX, double newY) {
        long stamp = sl.readLock();
        try {
            while (x == 0.0 && y == 0.0) {
                long ws = sl.tryConvertToWriteLock(stamp);
                if (ws != 0L) {
                    stamp = ws;
                    x = newX; y = newY;
                    break;
                } else {
                    sl.unlockRead(stamp);
                    stamp = sl.writeLock();
                }
            }
        } catch (Exception e) {
            // Handle exception
        }
    }
    ...
}
```

Conditional write
The Point class shows how to program with StampedLock

class Point {
    ...  
    void moveIfAtOrigin(double newX, double newY) {
        long stamp = sl.readLock();  // Acquire a read lock
        try {
            while (x == 0.0 && y == 0.0) {
                long ws = sl.tryConvertToWriteLock(stamp);
                if (ws != 0L) {
                    stamp = ws;
                    x = newX; y = newY;
                    break;
                } else {
                    sl.unlockRead(stamp);
                    stamp = sl.writeLock();
                }
            }
        } else {
            sl.unlockRead(stamp);
            stamp = sl.writeLock();
        }
    }
    ...
Simple Example of StampedLock

• The Point class shows how to program with StampedLock

```java
class Point {
    ... 
    void moveIfAtOrigin(double newX, double newY) {
        long stamp = sl.readLock();
        try {
            while (x == 0.0 && y == 0.0) {
                long ws = sl.tryConvertToWriteLock(stamp);
                if (ws != 0L) {
                    stamp = ws;
                    x = newX; y = newY;
                    break;
                } else {
                    sl.unlockRead(stamp);
                    stamp = sl.writeLock();
                }
            }
        } catch (InterruptedException e) {
            // Handle exception
        }
    }
    ... 
```
class Point {
    ...
    void moveIfAtOrigin(double newX, double newY) {
        long stamp = sl.readLock();
        try {
            while (x == 0.0 && y == 0.0) {
                long ws = sl.tryConvertToWriteLock(stamp);
                if (ws != 0L) {
                    stamp = ws;
                    x = newX; y = newY;
                    break;
                } else {
                    sl.unlockRead(stamp);
                    stamp = sl.writeLock();
                }
            }
        } catch (Exception e) {
            // Handle exception
        }
    }
    ...
}
The Point class shows how to program with StampedLock

class Point {
    ...
    void moveIfAtOrigin(double newX, double newY) {
        long stamp = sl.readLock();
        try {
            while (x == 0.0 && y == 0.0) {
                long ws = sl.tryConvertToWriteLock(stamp);
                if (ws != 0L) {
                    stamp = ws;
                    x = newX; y = newY;
                    break;
                } else {
                    sl.unlockRead(stamp);
                    stamp = sl.writeLock();
                }
            }
        } catch (IOException e) {
            ...  // handle exception
        }
    }
    ...
}
Simple Example of StampedLock

- The Point class shows how to program with StampedLock

```java
class Point {
    ... 
    void moveIfAtOrigin(double newX, double newY) {
        long stamp = sl.readLock();
        try {
            while (x == 0.0 && y == 0.0) {
                long ws = sl.tryConvertToWriteLock(stamp);
                if (ws != 0L) {
                    stamp = ws;
                    x = newX; y = newY; 
                    break;
                } else {
                    sl.unlockRead(stamp);
                    stamp = sl.writeLock();
                }
            }
        } finally {
            /* If lock was acquired and then released, return it.
             * This is used by acquireReadLock to recover from a race condition. 
             */
            if (stamp != sl.readLock())
                sl.unlockRead(stamp); 
        }
    }
    // ... 
```
Simple Example of StampedLock

- The Point class shows how to program with StampedLock

```java
class Point {
    ...
    void moveIfAtOrigin(double newX, double newY) {
        long stamp = sl.readLock();
        try {
            while (x == 0.0 && y == 0.0) {
                long ws = sl.tryConvertToWriteLock(stamp);
                if (ws != 0L) {
                    stamp = ws;
                    x = newX; y = newY;
                    break;  // Exit the loop
                } else {
                    sl.unlockRead(stamp);
                    stamp = sl.writeLock();
                }
            }
        } finally {
            ...
        }
    }
```
The Point class shows how to program with StampedLock

```java
class Point {
    ...
    void moveIfAtOrigin(double newX, double newY) {
        long stamp = sl.readLock();
        try {
            while (x == 0.0 && y == 0.0) {
                long ws = sl.tryConvertToWriteLock(stamp);
                if (ws != 0L) {
                    stamp = ws;
                    x = newX; y = newY;
                    break;
                } else {
                    sl.unlockRead(stamp);
                    stamp = sl.writeLock();
                }
            }
        } catch (
```
Simple Example of StampedLock

- The Point class shows how to program with StampedLock

```java
class Point {
    ...  
    void moveIfAtOrigin(double newX, double newY) {
        long stamp = sl.readLock();
        try {
            while (x == 0.0 && y == 0.0) {
                long ws = sl.tryConvertToWriteLock(stamp);
                if (ws != 0L) {
                    stamp = ws;
                    x = newX; y = newY;
                    break;
                } else {
                    sl.unlockRead(stamp);
                    stamp = sl.writeLock();
                }
            }
        } catch (InterruptedException e) {  
            // Handle interrupt exception
        }
    }
    ...  
```
Simple Example of StampedLock

- The Point class shows how to program with StampedLock

```java
class Point {
    ...
    void moveIfAtOrigin(double newX, double newY) {
        long stamp = sl.readLock();
        try {
            while (x == 0.0 && y == 0.0) {
                ...
                stamp = ws;
                ...
                stamp = sl.writeLock();
            }
        } finally { sl.unlock(stamp); }
    }
    ...
    Release the appropriate lock
```
End of Java Readers/Writer Locks (Part 3)
Java Readers/Writer Locks

(Part 4)

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 benefits that readers-writer synchronizers provide to concurrent programs
• Know the key methods in Java 8 StampedLock
• Recognize how to apply Java 8 StampedLock in practice
• Appreciate Java 8 StampedLock usage considerations
StampedLock Usage Considerations
# StampedLock Usage Conventions

- StampedLock often *much* faster than ReentrantReadWriteLock

<table>
<thead>
<tr>
<th>SYNCHRONIZED</th>
<th>OPTIMISTIC</th>
<th>RWLOCK</th>
<th>STAMPED</th>
</tr>
</thead>
<tbody>
<tr>
<td>1996.6</td>
<td>1174</td>
<td>116393</td>
<td>64077</td>
</tr>
<tr>
<td>2312.7</td>
<td>1174</td>
<td>116617</td>
<td>47897</td>
</tr>
<tr>
<td>2100.9</td>
<td>1122</td>
<td>117746</td>
<td>65921</td>
</tr>
<tr>
<td>2285.1</td>
<td>1182.9</td>
<td>115605</td>
<td>73500</td>
</tr>
<tr>
<td>2173.6</td>
<td>1184.9</td>
<td>118346</td>
<td>32857</td>
</tr>
<tr>
<td><strong>2173.78</strong></td>
<td><strong>1167.56</strong></td>
<td><strong>116941.4</strong></td>
<td><strong>56850.4</strong></td>
</tr>
</tbody>
</table>

19 readers & 1 writer

StampedLock Usage Conventions

- StampedLock often *much* faster than ReentrantReadWriteLock
- However, your mileage may vary!

![EPA Fuel Economy Estimates](image)

**EPA Fuel Economy Estimates**

These estimates reflect new EPA methods beginning with 2008 models.

<table>
<thead>
<tr>
<th>CITY MPG</th>
<th>18</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expected range for most drivers</td>
<td>15 to 21 MPG</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>HIGHWAY MPG</th>
<th>25</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expected range for most drivers</td>
<td>21 to 29 MPG</td>
</tr>
</tbody>
</table>

**Estimated Annual Fuel Cost**

$2,039

based on 15,000 miles at $2.80 per gallon

**Combined Fuel Economy**

This Vehicle

21

10 ▼ 31

All SUVs

Your actual mileage will vary depending on how you drive and maintain your vehicle.

See the FREE Fuel Economy Guide at dealers or [www.fueleconomy.gov](http://www.fueleconomy.gov)
StampedLock Usage Conventions

- StampedLock often *much* faster than ReentrantReadWriteLock
- However, your mileage may vary!

<table>
<thead>
<tr>
<th>RWLOCK</th>
<th>STAMPED</th>
<th>SYNCHRONIZED</th>
<th>OPTIMISTIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1960.8</td>
<td>165.1</td>
<td>177.4</td>
<td>387.9</td>
</tr>
<tr>
<td>1473.6</td>
<td>111.3</td>
<td>192.1</td>
<td>382.8</td>
</tr>
<tr>
<td>2119.7</td>
<td>216.8</td>
<td>173.3</td>
<td>403.6</td>
</tr>
<tr>
<td>2772.2</td>
<td>221.9</td>
<td>205.4</td>
<td>403.9</td>
</tr>
<tr>
<td>2721.4</td>
<td>189.3</td>
<td>181.2</td>
<td>394.2</td>
</tr>
<tr>
<td>2209.54</td>
<td>180.88</td>
<td>185.88</td>
<td>394.48</td>
</tr>
</tbody>
</table>

10 readers & 10 writers

Readers-writer lock speedups are only fully realized under certain conditions.
• Readers-writer lock speedups are only fully realized under certain conditions, e.g.
  • Frequency of reads to writes
    • Ideally, *many* more reads than writes
Readers-writer lock speedups are only fully realized under certain conditions, e.g.

- Frequency of reads to writes
- Duration of read & write operations
- Ideally, read operations should be non-trivial or else locking costs will dominate
Readers-writer lock speedups are only fully realized under certain conditions, e.g.:
- Frequency of reads to writes
- Duration of read & write operations
- Contention for the data
  - Ideally, many concurrent readers

StampedLock Usage Conventions
Readers-writer lock speedups are only fully realized under certain conditions, e.g.

- Frequency of reads to writes
- Duration of read & write operations
- Contention for the data
- Number of cores
  - Ideally, *many* cores
StampedLock Usage Conventions

- StampedLock is more complicated to use
StampedLock Usage Conventions

- StampedLock is more complicated to use
- Many more methods
StampedLock Usage Conventions

- StampedLock is more complicated to use
  - Many more methods
  - More intricate semantics & usage patterns

```java
void moveIfAtOrigin(double newX, double newY) {
    long stamp = sl.readLock();
    try {
        while (x == 0.0 && y == 0.0) {
            long ws =
                sl.tryConvertToWriteLock(stamp);
            if (ws != 0L) {
                stamp = ws;
                x = newX; y = newY;
                break;
            } else {
                sl.unlockRead(stamp);
                stamp = sl.writeLock();
            }
        }
    } finally {
        sl.unlock(stamp);
    }
    ...
}
```

See [www.techevents.online/using-java-8-lambdas-stampedlock-manage-thread-safety](http://www.techevents.online/using-java-8-lambdas-stampedlock-manage-thread-safety)
StampedLock Usage Conventions

- StampedLock is more complicated to use
  - Many more methods
  - More intricate semantics & usage patterns
- Invariants are tricky with optimistic read locks

```java
class Boooom {
    int mX = 0;
    int mY = 1;
    StampedLock mSL =
        new StampedLock();
}

while (true) {
    mSL.writeLock();
    mX++; mY++;
    mSL.writeUnlock();
}

Thread T₁

while (true) {
    mSL.writeLock();
    mX++; mY++;
    mSL.writeUnlock();
}

Thread T₂

do {
    stamp = mSL.tryOptimisticRead();
    z = 1 / (mX - mY);
} while (mSL.validate(stamp));

See concurrencyfreaks.blogspot.com/2013/11/stampedlocktryoptimisticread-and.html
StampedLock Usage Conventions

- StampedLock is more complicated to use
  - Many more methods
  - More intricate semantics & usage patterns
- Invariants are tricky with optimistic read locks
  - Fields read in optimistic mode may be wildly inconsistent

```
class Boooom {
    int mX = 0;
    int mY = 1;
    StampedLock mSL =
        new StampedLock();
}

while (true) {
    mSL.writeLock();
    mX++; mY++;
    mSL.writeUnlock();
}
```

```
Thread T_1
while (true) {
    mSL.writeLock();
    mX++; mY++;
    mSL.writeUnlock();
}
```

```
Thread T_2
do {
    stamp = mSL.tryOptimisticRead();
    z = 1 / (mX - mY);
} while (mSL.validate(stamp));
```

The values of mX & mY may be reordered & thus mX may not always == mY - 1
StampedLock Usage Conventions

- StampedLock is more complicated to use
  - Many more methods
  - More intricate semantics & usage patterns
  - Invariants are tricky with optimistic read locks
- Non-reentrant

```java
class SomeComponent {
  private StampedLock sl = new StampedLock();

  public void someMethod1() {
    long stamp = sl.readLock();
    someMethod2();
    ...
  }

  private void someMethod2() {
    long stamp = sl.readLock();
    ...
  }
```
Despite its complexity, StampedLock is the best choice for readers-writer locks in Java 8+!

Class StampedLock
java.lang.Object
java.util.concurrent.locks.StampedLock

All Implemented Interfaces:
Serializable

public class StampedLock
extends Object
implements Serializable

A capability-based lock with three modes for controlling read/write access. The state of a StampedLock consists of a version and mode. Lock acquisition methods return a stamp that represents and controls access with respect to a lock state; “try” versions of these methods may instead return the special value zero to represent failure to acquire access. Lock release and conversion methods require stamps as arguments, and fail if they do not match the state of the lock. The three modes are:

- **Writing.** Method writeLock() possibly blocks waiting for exclusive access, returning a stamp that can be used in method unlockWrite(long) to release the lock. Untimed and timed versions of tryWriteLock are also provided. When the lock is held in write mode, no read locks may be obtained, and all optimistic read validations will fail.

- **Reading.** Method readLock() possibly blocks waiting for non-exclusive access, returning a stamp that can be used in method unlockRead(long) to release the lock. Untimed and timed versions of tryReadLock are also provided.

- **Optimistic Reading.** Method tryOptimisticRead() returns a non-zero stamp only if the lock is not currently held in write mode. Method validate(long) returns true if the lock has not been acquired in write mode since obtaining a given stamp. This mode can be thought of as an extremely weak version of a read-lock, that can be broken by a writer at any time. The use of optimistic mode for short read-only code segments.

End of Java Readers/Writer Locks (Part 4)