Java StampedLock: Key Methods



Douglas C. Schmidt <u>d.schmidt@vanderbilt.edu</u> 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 StampedLock class
- Know the key methods in Java StampedLock

<<Java Class>> G StampedLock StampedLock() writeLock():long tryWriteLock():long tryWriteLock(long,TimeUnit):long writeLockInterruptibly():long readLock():long tryReadLock():long tryReadLock(long,TimeUnit):long readLockInterruptibly():long tryOptimisticRead():long validate(long):boolean unlockWrite(long):void ounlockRead(long):void unlock(long):void tryConvertToWriteLock(long):long tryConvertToReadLock(long):long tryConvertToOptimisticRead(long):long tryUnlockWrite():boolean tryUnlockRead():boolean isWriteLocked():boolean isReadLocked():boolean getReadLockCount():int toString() asReadLock():Lock asWriteLock():Lock asReadWriteLock():ReadWriteLock

• Writing mode methods, which acquire the lock exclusively



Only one thread at a time can acquire a lock exclusively

public class StampedLock implements java.io.Serializable { public long writeLock() { ... } public long tryWriteLock() { ... } public long tryWriteLock (long time, TimeUnit unit) {...}



 Writing mode methods, which acquire the lock exclusively





These methods are "pessimistic", i.e., they assume contention can occur

Writing mode methods, which acquire the lock exclusively

All methods return a "stamp" value, which is a long that contains a version & a mode public class StampedLock
 implements java.io.Serializable {

public long writeLock() { ... }

public long tryWriteLock() { ... }

public long tryWriteLock

(long time, TimeUnit unit) {...}



See dzone.com/articles/a-look-at-stampedlock

- Writing mode methods, which acquire the lock exclusively
 - Acquires lock exclusively, blocking until available
 - This method can't be interrupted



public cl impleme	ass S ents j	StampedLock java.io.Serializable {
 public	long	<pre>writeLock() { }</pre>
public	long	<pre>tryWriteLock() { }</pre>
public	long	<pre>tryWriteLock (long time, TimeUnit unit) {}</pre>

There's also a writeLockInterruptibly() method that can be interrupted

- Writing mode methods, which acquire the lock exclusively
 - Acquires lock exclusively, blocking until available
 - Acquires lock exclusively if it's immediately available
 - Otherwise, it returns 0

- Writing mode methods, which acquire the lock exclusively
 - Acquires lock exclusively, blocking until available
 - Acquires lock exclusively if it's immediately available
 - Acquires lock exclusively if available within given time
 - Otherwise, it returns 0

```
public class StampedLock
implements java.io.Serializable {
....
public long writeLock() { ... }
public long tryWriteLock() { ... }
public long tryWriteLock
(long time,
TimeUnit unit) {...}
```

- Writing mode methods, which acquire the lock exclusively
 - Acquires lock exclusively, blocking until available
 - Acquires lock exclusively if it's immediately available
 - Acquires lock exclusively if available within given time
 - Otherwise, it returns 0
 - This method throws Interrupted Exception if it's interrupted



 Reading mode methods, which acquire the lock non-exclusively public class StampedLock
 implements java.io.Serializable {
 ...

public long readLock() { ... }

public long tryReadLock() { ... }

public long tryReadLock

(long time, TimeUnit unit) {...}



Many threads at a time can acquire a lock non-exclusively

 Reading mode methods, which acquire the lock non-exclusively



public class StampedLock implements java.io.Serializable { public long readLock() { ... } public long tryReadLock() { ... } public long tryReadLock (long time, TimeUnit unit) {...}



These methods are "pessimistic", i.e., they assume contention can occur

 Reading mode methods, which acquire the lock non-exclusively public class StampedLock
 implements java.io.Serializable {

public long readLock() { ... }

Again, all methods return a "stamp", which is a long that contains a version & a mode

public long tryReadLock() { ... }



- Reading mode methods, which acquire the lock non-exclusively
 - Acquires lock non-exclusively, blocking until available
 - This method can't be interrupted



<pre>public class StampedLock implements java.io.Serializable</pre>	{
<pre> public long readLock() { }</pre>	
<pre>public long tryReadLock() {</pre>	}
<pre>public long tryReadLock (long time, TimeUnit unit) {</pre>	. }
• • •	

There's also a readLockInterruptibly() method that can be interrupted

- Reading mode methods, which acquire the lock non-exclusively
 - Acquires lock non-exclusively, blocking until available
 - Acquires lock non-exclusively if immediately available
 - Otherwise, it returns 0

```
public class StampedLock
implements java.io.Serializable {
...
public long readLock() { ... }
public long tryReadLock() { ... }
public long tryReadLock
        (long time,
        TimeUnit unit) {...}
```

- Reading mode methods, which acquire the lock non-exclusively
 - Acquires lock non-exclusively, blocking until available
 - Acquires lock non-exclusively if immediately available
 - Acquires lock non-exclusively if it is available within given time
 - Otherwise, it returns 0

```
public class StampedLock
implements java.io.Serializable {
....
public long readLock() { ... }
public long tryReadLock() { ... }
public long tryReadLock
(long time,
TimeUnit unit) {...}
```

- Reading mode methods, which acquire the lock non-exclusively
 - Acquires lock non-exclusively, blocking until available
 - Acquires lock non-exclusively if immediately available
 - Acquires lock non-exclusively if it is available within given time
 - Otherwise, it returns 0
 - This method throws Interrupted Exception if it's interrupted



 Optimistic reading mode methods, which acquire the lock non-exclusively public class StampedLock
implements java.io.Serializable {
...
public long tryOptimisticRead()
{ ... }

public boolean validate

(long stamp) { ... }

Many threads at a time can acquire a lock non-exclusively



 Optimistic reading mode methods, which acquire the lock non-exclusively





```
(long stamp) { ... }
```



These methods are "optimistic", i.e., they assume contention may not occur

- Optimistic reading mode methods, which acquire the lock non-exclusively
 - Returns an "observation stamp" for later validation or 0 if the lock is currently held exclusively

```
public class StampedLock
implements java.io.Serializable {
....
public long tryOptimisticRead()
        { ... }
public boolean validate
```

```
(long stamp) { ... }
```

- Optimistic reading mode methods, which acquire the lock non-exclusively
 - Returns an "observation stamp" for later validation or 0 if the lock is currently held exclusively
 - Code using this mode reads the value of fields & holds them in local variables for use after they are "validated"

```
public class StampedLock
  implements java.io.Serializable {
    ...
    public long tryOptimisticRead()
        { ... }
```

```
public boolean validate
    (long stamp) { ... }
```

See upcoming part of this lesson on "Java StampedLock: Example Application"

- Optimistic reading mode methods, which acquire the lock non-exclusively
 - Returns an "observation stamp" for later validation or 0 if the lock is currently held exclusively
 - Code using this mode reads the value of fields & holds them in local variables for use after they are "validated"
 - tryOptimisticRead() internally does a volatile read on a field inside of StampedLock to ensure "fresh" values of fields are observed

```
public class StampedLock
  implements java.io.Serializable {
    ...
    public long tryOptimisticRead()
        { ... }
```

```
public boolean validate
    (long stamp) { ... }
```

See www.javaspecialists.eu/archive/Issue242.html

- Optimistic reading mode methods, which acquire the lock non-exclusively
 - Returns an "observation stamp" for later validation or 0 if the lock is currently held exclusively
 - Returns true if lock hasn't been acquired exclusively since stamp was issued, else false

```
public class StampedLock
implements java.io.Serializable {
    ...
    public long tryOptimisticRead()
        { ... }
```

```
public boolean validate
  (long stamp) { ... }
```

- Optimistic reading mode methods, which acquire the lock non-exclusively
 - Returns an "observation stamp" for later validation or 0 if the lock is currently held exclusively
 - Returns true if lock hasn't been acquired exclusively since stamp was issued, else false
 - If validate() succeeds (i.e., returns true) synchronization overhead is very low & there's no need to unlock the "lock"

```
public class StampedLock
  implements java.io.Serializable {
    ...
    public long tryOptimisticRead()
        { ... }
```

```
public boolean validate
  (long stamp) { ... }
```

 Conditionally convert a stamp across lock modes



public class StampedLock
 implements java.io.Serializable {

```
public long
```

tryToConvertToWriteLock

```
(long stamp) { ... }
```

public long

tryToConvertToReadLock

(long stamp) { ... }

```
public long
   tryToConvertToOptimisticRead
   (long stamp) { ... }
```

These calls perform work *atomically* (despite lack of documentation ;-))

 Conditionally convert a stamp to a write lock public class StampedLock implements java.io.Serializable { public long tryToConvertToWriteLock (long stamp) { ... } public long tryToConvertToReadLock (long stamp) { ... } public long tryToConvertToOptimisticRead

```
(long stamp) { ... }
```

- Conditionally convert a stamp to a write lock
 - If lock state matches stamp, performs one following action

```
public long
   tryToConvertToOptimisticRead
        (long stamp) { ... }
```

- Conditionally convert a stamp to a write lock
 - If lock state matches stamp, performs one following action
 - If stamp represents holding a write lock, return it

```
public long
   tryToConvertToOptimisticRead
        (long stamp) { ... }
```

- Conditionally convert a stamp to a write lock
 - If lock state matches stamp, performs one following action
 - If stamp represents holding a write lock, return it
 - If stamp represents holding a read lock—& a write lock is available—atomically release read lock & return write stamp



```
public class StampedLock
  implements java.io.Serializable {
  public long
           tryToConvertToWriteLock
              (long stamp) { ... }
  public long
           tryToConvertToReadLock
              (long stamp) { ... }
  public long
       tryToConvertToOptimisticRead
```

```
(long stamp) { ... }
```

This represents a "lock upgrade"

- Conditionally convert a stamp to a write lock
 - If lock state matches stamp, performs one following action
 - If stamp represents holding a write lock, return it
 - If stamp represents holding a read lock—& a write lock is available—atomically release read lock & return write stamp
 - If stamp represents a read that's optimistic, return a write stamp if immediately available

```
tryToConvertToReadLock
```

```
(long stamp) { ... }
```

```
public long
   tryToConvertToOptimisticRead
   (long stamp) { ... }
```

- Conditionally convert a stamp to a write lock
 - If lock state matches stamp, performs one following action
 - If stamp represents holding a write lock, return it
 - If stamp represents holding a read lock—& a write lock is available—atomically release read lock & return write stamp
 - If stamp represents a read that's optimistic, return a write stamp if immediately available
 - Else return zero

```
(long stamp) { ... }
```

```
public long
   tryToConvertToOptimisticRead
   (long stamp) { ... }
```

 Conditionally convert a stamp to a read lock tryToConvertToOptimisticRead

(long stamp) { ... }

- Conditionally convert a stamp to a read lock
 - If lock state matches stamp, performs one following action

```
public class StampedLock
implements java.io.Serializable {
...
public long
tryToConvertToWriteLock
(long stamp) { ... }
```

public long

```
tryToConvertToReadLock
```

```
(long stamp) { ... }
```

public long
 tryToConvertToOptimisticRead
 (long stamp) { ... }

- Conditionally convert a stamp to a read lock
 - If lock state matches stamp, performs one following action
 - If stamp represents holding a write lock atomically release it & obtain read lock



```
public class StampedLock
implements java.io.Serializable {
...
public long
tryToConvertToWriteLock
(long stamp) { ... }
```

```
public long
```

tryToConvertToReadLock

(long stamp) { ... }

```
public long
   tryToConvertToOptimisticRead
      (long stamp) { ... }
```

This represents a "lock downgrade"

- Conditionally convert a stamp to a read lock
 - If lock state matches stamp, performs one following action
 - If stamp represents holding a write lock atomically release it & obtain read lock
 - If stamp represents holding a read lock, return it

```
public class StampedLock
  implements java.io.Serializable {
    ...
    public long
```

```
tryToConvertToWriteLock
```

```
(long stamp) { ... }
```

```
public long
```

tryToConvertToReadLock

```
(long stamp) { ... }
```

```
public long
```

```
{\tt try To Convert To Optimistic Read}
```

```
(long stamp) { ... }
```

- Conditionally convert a stamp to a read lock
 - If lock state matches stamp, performs one following action
 - If stamp represents holding a write lock atomically release it & obtain read lock
 - If stamp represents holding a read lock, return it
 - If stamp represents holding an optimistic read, return read stamp only if available

public class StampedLock
implements java.io.Serializable {
 ...
 public long
 tryToConvertToWriteLock
 (long stamp) { ... }

public long

tryToConvertToReadLock

(long stamp) { ... }

```
public long
   tryToConvertToOptimisticRead
        (long stamp) { ... }
```

- Conditionally convert a stamp to a read lock
 - If lock state matches stamp, performs one following action
 - If stamp represents holding a write lock atomically release it & obtain read lock
 - If stamp represents holding a read lock, return it
 - If stamp represents holding an optimistic read, return read stamp only if available
 - Else return zero

```
public class StampedLock
implements java.io.Serializable {
    ...
    public long
        tryToConvertToWriteLock
        (long stamp) { ... }
```

```
public long
```

tryToConvertToReadLock

```
(long stamp) { ... }
```

```
public long
   tryToConvertToOptimisticRead
      (long stamp) { ... }
```

 Conditionally convert a stamp to an optimistic read lock public long

tryToConvertToOptimisticRead

```
(long stamp) { ... }
```

- Conditionally convert a stamp to an optimistic read lock
 - If lock state matches stamp, performs one following action

```
public long
```

```
tryToConvertToReadLock
```

```
(long stamp) { ... }
```

public long

```
tryToConvertToOptimisticRead
```

```
(long stamp) { ... }
```

- Conditionally convert a stamp to an optimistic read lock
 - If lock state matches stamp, performs one following action
 - If stamp represents holding a lock release it & return an observation stamp



```
public class StampedLock
implements java.io.Serializable {
...
public long
tryToConvertToWriteLock
(long stamp) { ... }
```

```
public long
```

```
tryToConvertToReadLock
```

```
(long stamp) { ... }
```

```
public long
```

```
tryToConvertToOptimisticRead
```

```
(long stamp) { ... }
```

This represents a "lock downgrade"

- Conditionally convert a stamp to an optimistic read lock
 - If lock state matches stamp, performs one following action
 - If stamp represents holding a lock release it & return an observation stamp
 - If stamp represents holding an optimistic read, return it if it's validated

```
public class StampedLock
implements java.io.Serializable {
...
public long
tryToConvertToWriteLock
(long stamp) { ... }
```

```
public long
```

```
tryToConvertToReadLock
```

```
(long stamp) { ... }
```

```
public long
```

```
tryToConvertToOptimisticRead
```

```
(long stamp) { ... }
```

- Conditionally convert a stamp to an optimistic read lock
 - If lock state matches stamp, performs one following action
 - If stamp represents holding a lock release it & return an observation stamp
 - If stamp represents holding an optimistic read, return it if it's validated
 - Else return zero

```
public class StampedLock
implements java.io.Serializable {
...
public long
    tryToConvertToWriteLock
        (long stamp) { ... }
```

```
public long
```

```
tryToConvertToReadLock
```

```
(long stamp) { ... }
```

```
public long
```

```
tryToConvertToOptimisticRead
```

```
(long stamp) { ... }
```

 There are several ways to unlock a StampedLock



public class StampedLock
 implements java.io.Serializable {

public void unlockWrite
 (long stamp) { ... }

public void unlockRead
 (long stamp) { ... }

public void unlock
 (long stamp) { ... }

- There are several ways to unlock a StampedLock
 - Releases exclusive lock if the state matches given stamp

public class StampedLock
 implements java.io.Serializable {

public void unlockWrite
 (long stamp) { ... }

public void unlockRead
 (long stamp) { ... }

public void unlock
 (long stamp) { ... }

- There are several ways to unlock a StampedLock
 - Releases exclusive lock if the state matches given stamp
 - Releases non-exclusive lock if the state matches given stamp

public class StampedLock
 implements java.io.Serializable {

public void unlockWrite
 (long stamp) { ... }

public void unlockRead
 (long stamp) { ... }

public void unlock
 (long stamp) { ... }

- There are several ways to unlock a StampedLock
 - Releases exclusive lock if the state matches given stamp
 - Releases non-exclusive lock if the state matches given stamp
 - Releases lock if the lock state matches given stamp

```
public class StampedLock
   implements java.io.Serializable {
```

public void unlockWrite
 (long stamp) { ... }

```
public void unlockRead
 (long stamp) { ... }
```

```
public void unlock
  (long stamp) { ... }
```

unlock() is slightly less efficient than unlockWrite() & unlockRead()

End of Java StampedLock: Key Methods