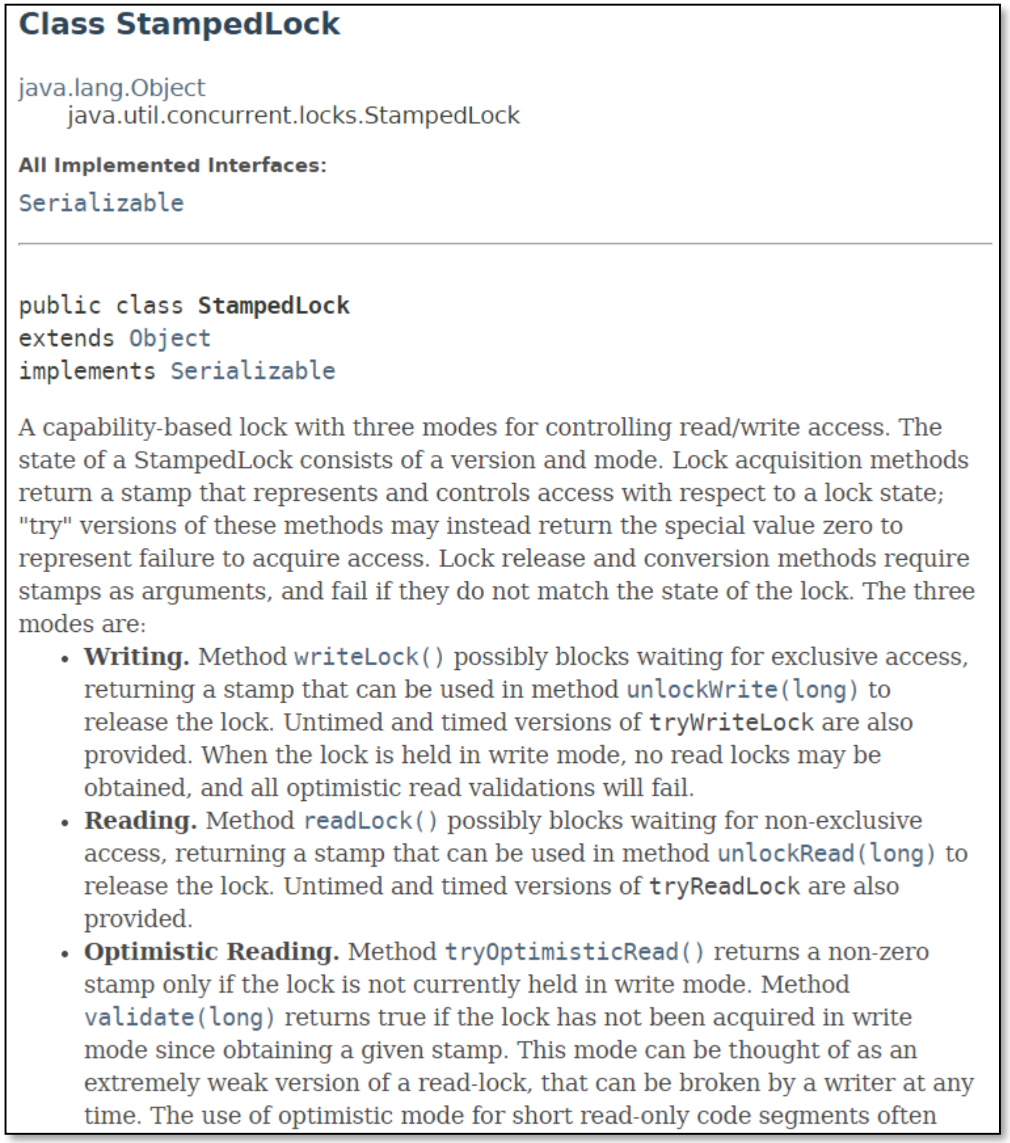
Java StampedLock: Structure & Functionality

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 Stamped Lock class



Class StampedLock

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

All Implemented Interfaces:

`Serializable`

```
public class StampedLock
extends Object
implements Serializable
```

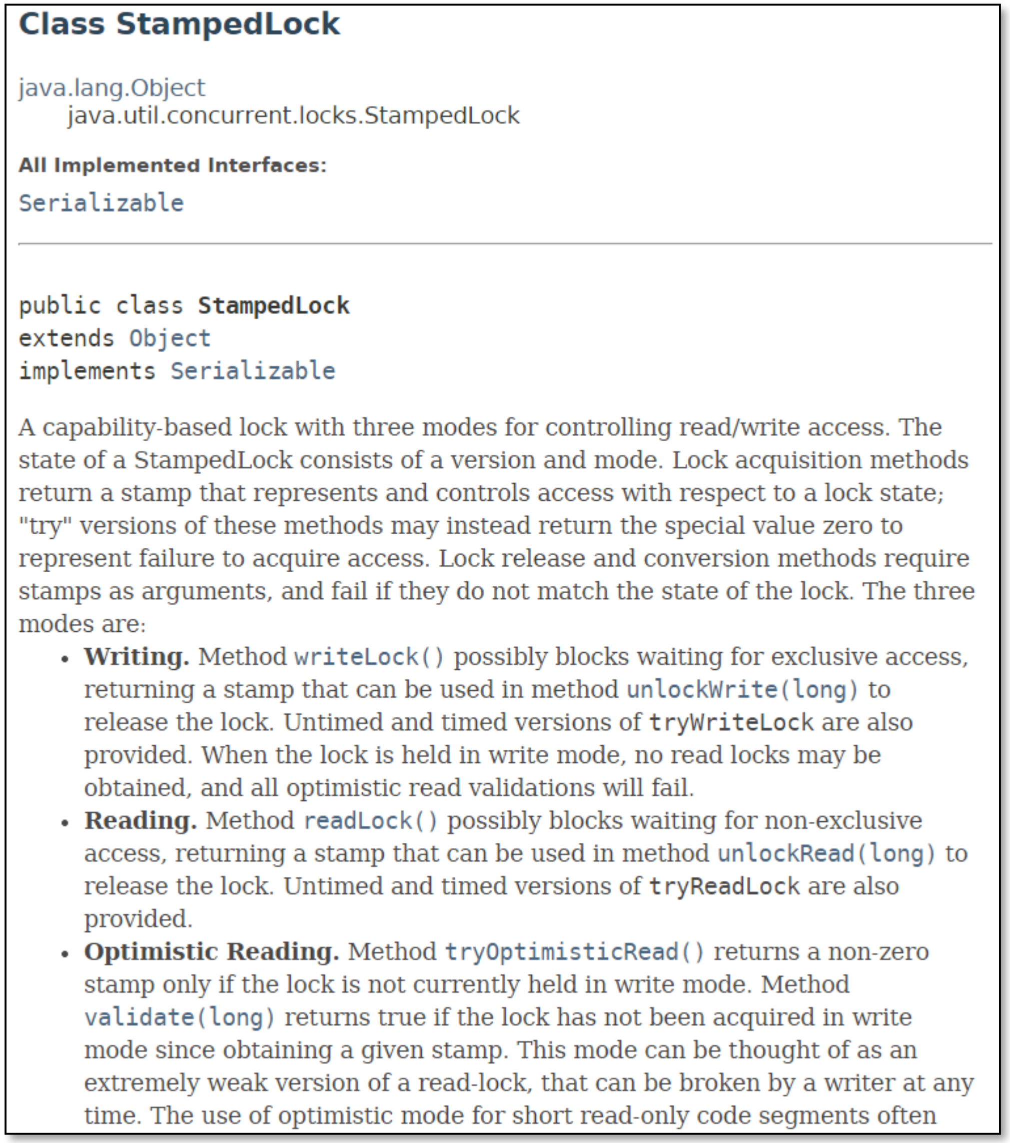
A capability-based lock with three modes for controlling read/write access. The state of a StampedLock consists of a version and mode. Lock acquisition methods return a stamp that represents and controls access with respect to a lock state; "try" versions of these methods may instead return the special 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 often

Overview of Java StampedLock

Overview of Java StampedLock

- Provides a readers-writer implementation in Java 8+



Class StampedLock

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

All Implemented Interfaces:

`Serializable`

```
public class StampedLock
extends Object
implements Serializable
```

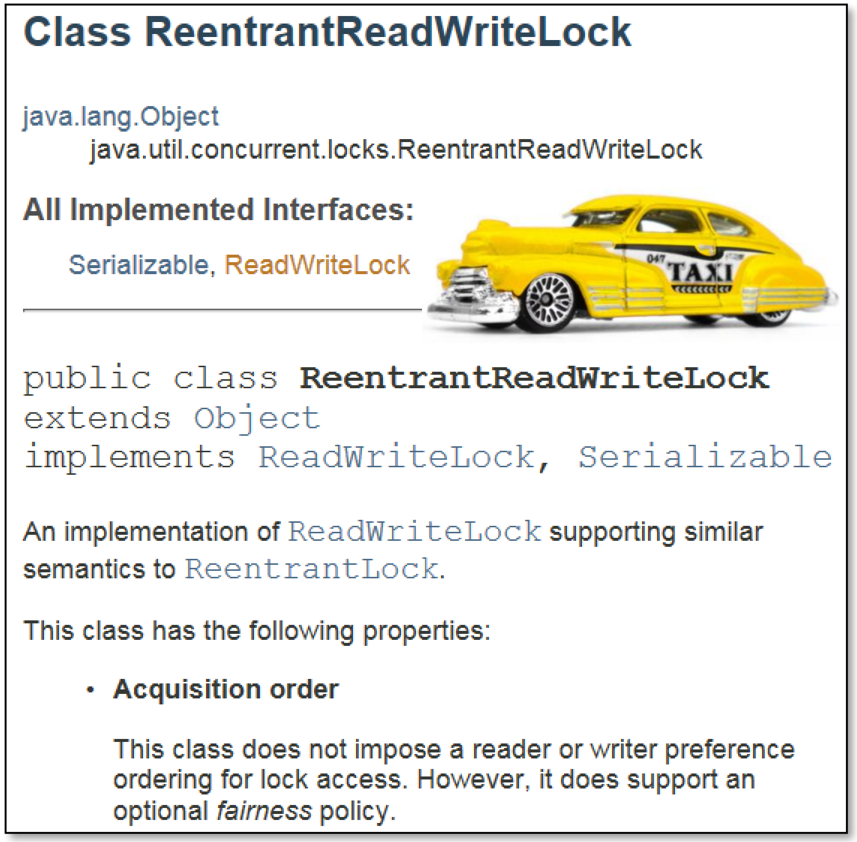
A capability-based lock with three modes for controlling read/write access. The state of a StampedLock consists of a version and mode. Lock acquisition methods return a stamp that represents and controls access with respect to a lock state; "try" versions of these methods may instead return the special 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 often

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/locks/StampedLock.html

Overview of Java StampedLock

- Provides a readers-writer implementation in Java 8+
  - Much more efficient & scalable than ReentrantReadWriteLock



Class ReentrantReadWriteLock

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

All Implemented Interfaces:

Serializable, ReadWriteLock

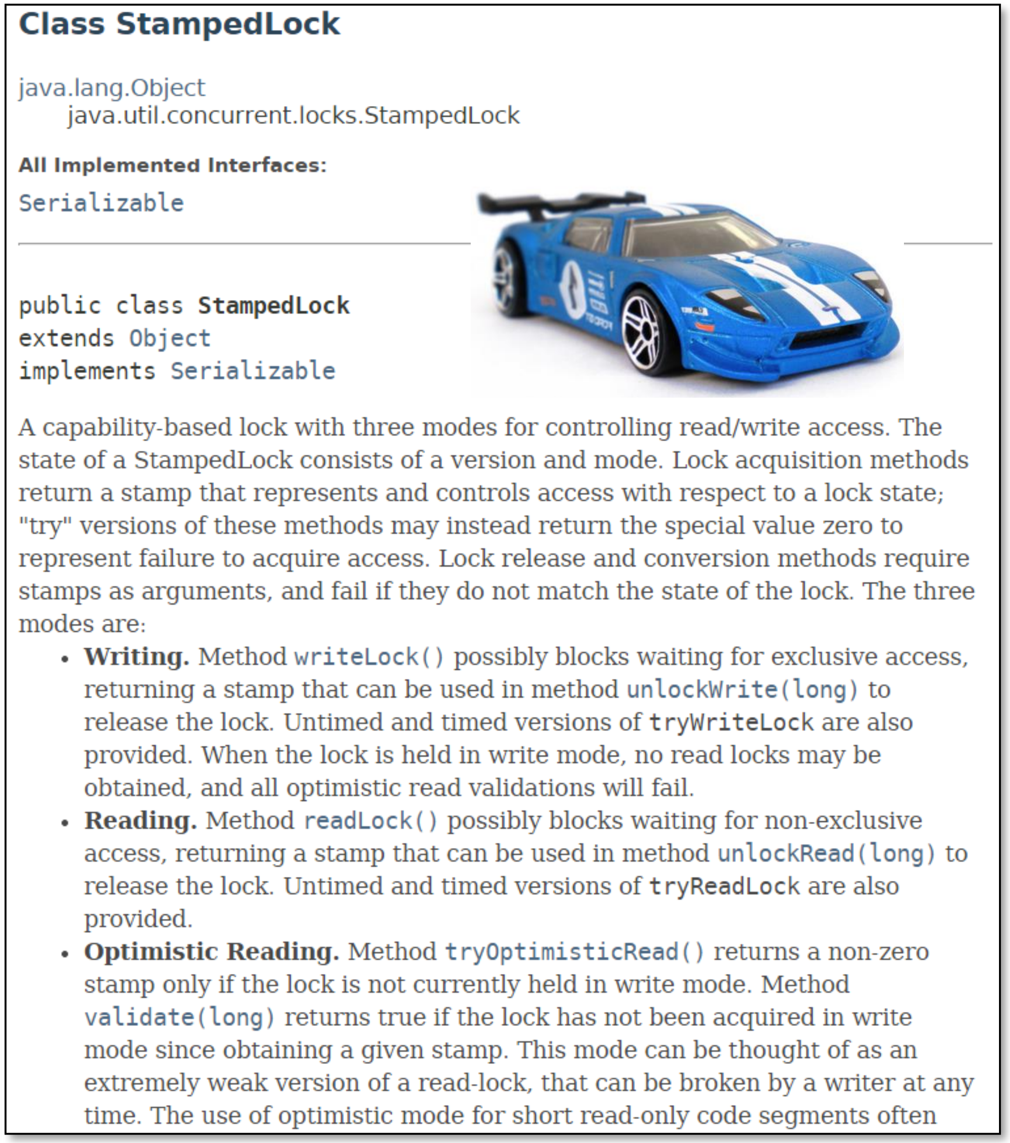
```
public class ReentrantReadWriteLock
extends Object
implements ReadWriteLock, Serializable
```

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.



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 often

Overview of Java StampedLock

- Provides a readers-writer implementation in Java 8+

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

Does not implement ReadWriteLock interface, does not use the AbstractQueuedSynchronizer framework, & does not apply Bridge pattern

Overview of Java StampedLock

- Provides three locking modes

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

These modes go above & beyond what's supported in ReentrantReadWriteLock

Overview of Java StampedLock

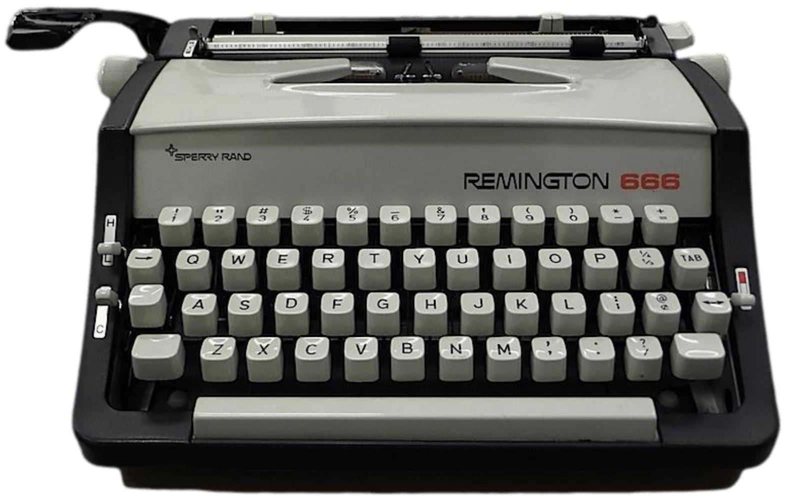
- Provides three locking modes
  - Writing

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

  public long tryWriteLock() { ... }

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

  ...
```



Writing mode 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"

Overview of Java StampedLock

- Provides three locking modes
  - Writing
  - Reading

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

  public long tryReadLock() { ... }

  public long tryReadLock
                (long time,
                 TimeUnit unit) {...}
  ...
```

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

Overview of Java StampedLock

- Provides three locking modes
  - Writing
  - Reading
  - Optimistic reading

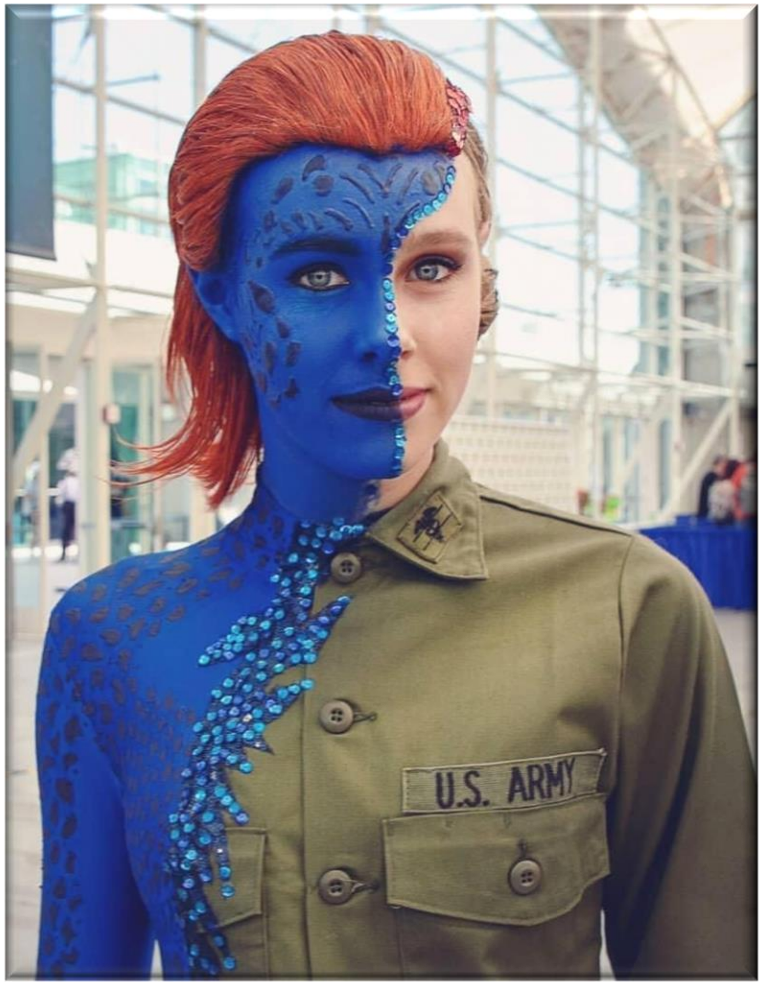
```
class StampedLock
  implements java.io.Serializable {
  ...
  public long tryOptimisticRead()
    { ... }

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

This reading mode is "optimistic" since it assumes contention will not occur, so other threads can obtain the lock optimistically, i.e., the lock is "probabilistic"

Overview of Java StampedLock

- It's also possible to convert a lock from one mode to another



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

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

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

End of Java StampedLock: Structure & Functionality