

# Java StampedLock: Example Application



**Douglas C. Schmidt**  
**[d.schmidt@vanderbilt.edu](mailto:d.schmidt@vanderbilt.edu)**  
**[www.dre.vanderbilt.edu/~schmidt](http://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
- Recognize how to apply Java 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();  
                }  
            } ...  
    }
```

---

# Applying Java Stamped Lock in Practice

# Applying Java StampedLock in Practice

---

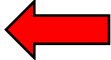
- The Point class shows how to program with StampedLock

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

# Applying Java StampedLock in Practice

---

- The Point class shows how to program with StampedLock

`class Point {`  **Maintains two-dimensional points**

```
    private double x;
```

```
    private double y;
```

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

```
    ...
```

# Applying Java StampedLock in Practice

---

- The Point class shows how to program with StampedLock

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

 **State that must be protected**

# Applying Java StampedLock in Practice

---

- The Point class shows how to program with StampedLock

```
class Point {
```

```
    private double x;
```

```
    private double y;
```

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

```
    ...
```

 **StampedLock that does  
the protecting**

---

# Applying Java StampedLock: Writing Mode

# Applying Java StampedLock: Writing Mode

- Performing an exclusive write with a StampedLock

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

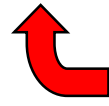
**This method atomically moves  
a point to a new location**



# Applying Java StampedLock: Writing Mode

- Performing an exclusive write with a StampedLock

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

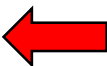
 Acquire a write lock



# Applying Java StampedLock: Writing Mode

---

- Performing an exclusive write with a 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);  
        }  
    }  
    ...  
}
```

# Applying Java StampedLock: Writing Mode

---

- Performing an exclusive write with a StampedLock

```
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**

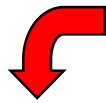
---

# Applying Java StampedLock: Optimistic & Reading Mode

# Applying Java StampedLock: Optimistic & Reading Mode

- Performing a optimistic read with a 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);  
    }  
    ...  
}
```



**A read-only method**



# Applying Java StampedLock: Optimistic & Reading Mode

- Performing a optimistic read with a 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);  
    }  
    ...  
}
```

 Attempt to get an  
"observation" stamp

# Applying Java StampedLock: Optimistic & Reading Mode

- Performing a optimistic read with a 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);  
    }  
    ...  
}
```



**“Optimistically” read  
state into local variables**

Code using optimistic reading mode typically copies the values of fields & holds them in local variables for use after they are validated

# Applying Java StampedLock: Optimisitic & Reading Mode

- Performing a optimistic read with a 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);  
    }  
    ...  
}
```



Check if another thread acquired the lock for writing after earlier call to tryOptimisticRead()

# Applying Java StampedLock: Optimisitic & Reading Mode

- Performing a optimistic read with a 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);  
    }  
    ...  
}
```

If write lock occurred then  
acquire a read lock (blocking  
as long as the write lock is  
held by another thread)

# Applying Java StampedLock: Optimistic & Reading Mode

- Performing a optimistic read with a 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;  Make copies of x & y  
                via "pessimistic" reads  
            } finally  
            { sl.unlockRead(stamp); }  
        }  
        return Math.sqrt (currX * currX + currY * currY);  
    }  
    ...  
}
```

# Applying Java StampedLock: Optimistic & Reading Mode

- Performing a optimistic read with a 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);  
    }  
    ...  
}
```



# Applying Java StampedLock: Optimisitic & Reading Mode

- Performing a optimistic read with a 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);  
    }  
    ...  
}
```

# Applying Java StampedLock: Optimisitic & Reading Mode

- Performing a optimistic read with a 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);  
    }  
    ...  
}
```



Do computation with the copied values

---

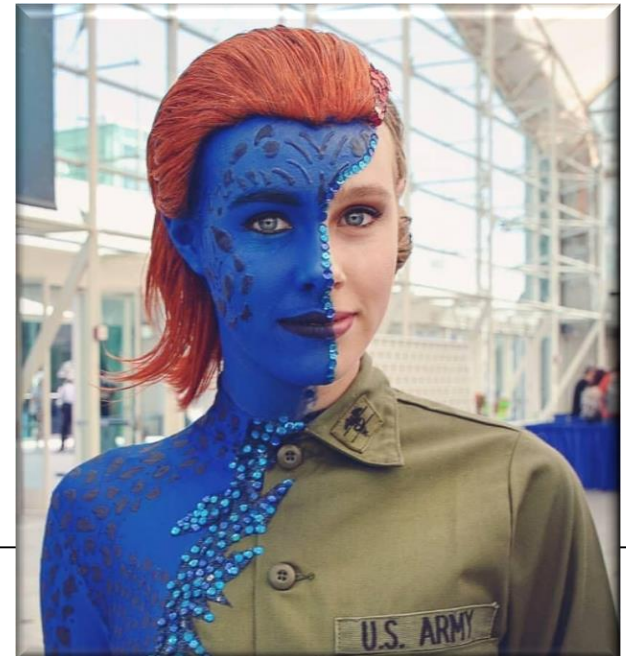
# Applying Java Stamped Lock: Conditional Write

# Applying Java StampedLock: Conditional Write

- Performing a conditional write with a 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();  
                }  
            }  
        }  
        ...  
    }  
}
```

**Move a point only if it's current at the origin**



# Applying Java StampedLock: Conditional Write

- Performing a conditional write with a 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();  
                }  
            }  
        ...  
    }  
}
```

# Applying Java StampedLock: Conditional Write

- Performing a conditional write with a 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();  
                }  
            }  
        ...  
    }  
}
```

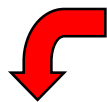
← Check whether x & y are at the origin

This loop only executes at most twice!

# Applying Java StampedLock: Conditional Write

- Performing a conditional write with a 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();  
                }  
            }  
        ...  
    }  
}
```

 Try to upgrade  
to a write lock  
w/out blocking

tryConvertToWriteLock() atomically releases the read lock  
& acquires the write lock if there are no other readers

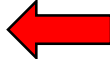
# Applying Java StampedLock: Conditional Write

- Performing a conditional write with a 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) {  Upgrade succeeded w/out blocking!  
                    stamp = ws;  
                    x = newX; y = newY;  
                    break;  
                } else {  
                    sl.unlockRead(stamp);  
                    stamp = sl.writeLock();  
                }  
            }  
        ...  
    }  
}
```

# Applying Java StampedLock: Conditional Write

- Performing a conditional write with a 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;  Update stamp &  
                    break;                                     modify Point's state  
                } else {  
                    sl.unlockRead(stamp);  
                    stamp = sl.writeLock();  
                }  
            }  
        ...  
    }  
}
```

# Applying Java StampedLock: Conditional Write

---

- Performing a conditional write with a StampedLock

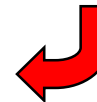
```
class Point {  
    ...  
    void moveIfAtOrigin(double newX, double newY) {  
        long stamp = sl.readLock();  
        try  
            while (x == 0.0 && y == 0.0) {  
                long ws = sl.tryConvertToWriteLock(stamp);  
                if (ws != 0L) {  
                    stamp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

# Applying Java StampedLock: Conditional Write

- Performing a conditional write with a 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();  
                }  
            }  
        ...  
    }  
}
```

Upgrade failed, so release the  
read lock & block until the write  
lock acquired exclusively



The **x** & **y** field values may change between `unlockRead()` & `writeLock()`!

# Applying Java StampedLock: Conditional Write

- Performing a conditional write with a 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();  
                }  
            }  
        }  
        ...  
    }  
}
```



**Must retest loop condition since x & y field values may change between unlockRead() & writeLock()!**

# Applying Java StampedLock: Conditional Write

- Performing a conditional write with a 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();  
                }  
            }  
        ...  
    }  
}
```

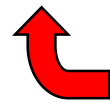


**This conversion will always succeed since stamp is now a write lock**

# Applying Java StampedLock: Conditional Write

- Performing a conditional write with a StampedLock

```
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 Stamped Lock: Example Application