

Thread Safety with Phaser, StampedLock and VarHandle

Dr Heinz M. Kabutz

Last updated 2019-05-24

© 2018-2019 Heinz Kabutz – All Rights Reserved



Whois

tinyurl.com/techweek19

- **Not ketchup**
- **Will come train your team on Java**
 - Design Patterns
 - Concurrency
 - Refactoring
 - Advanced Java tricks
 - Contact me heinz@kabutz.net
 - (Only if your employer loves you)



Phaser



Phasers

- **Allows threads to coordinate by phases**
 - Similar to `CountDownLatch` and `CyclicBarrier`, but more flexible
- **Registration**
 - Number of parties *registered* may vary over time
 - Same as *count* in `CountDownLatch`, *parties* in `CyclicBarrier`
 - A party can register/deregister itself at any time
- **ManagedBlocker**
 - Can be used in the `ForkJoinPool`

Demo of Cojoining Approaches

github.com/kabutz/modern-synchronizers



StampedLock



What is StampedLock?

- **Java 8 synchronizer**
- **Allows optimistic reads**
 - ReentrantReadWriteLock only has pessimistic reads
- **Not reentrant**
 - This is not a feature
- **Use to enforce invariants across multiple fields**
 - For simple classes, synchronized/volatile is easier and faster
- **Can split locking and unlocking between threads**

Pessimistic Exclusive Lock (write)

```
public class StampedLock {  
    long writeLock() // never returns 0, might block  
  
    // returns new write stamp if successful; otherwise 0  
    long tryConvertToWriteLock(long stamp)  
  
    void unlockWrite(long stamp) // needs write stamp  
  
    // and a bunch of other methods left out for brevity
```


Pessimistic Non-Exclusive Lock (read)

```
public class StampedLock { // continued ...  
    long readLock() // never returns 0, might block  
  
    // returns new read stamp if successful; otherwise 0  
    long tryConvertToReadLock(long stamp)  
  
    void unlockRead(long stamp) // needs read stamp  
  
    void unlock(long stamp) // unlocks read or write
```

Optimistic Non-Exclusive Read (No Lock)

```
public class StampedLock { // continued ...  
    // could return 0 if a write stamp has been issued  
    long tryOptimisticRead()  
  
    // return true if stamp was non-zero and no write  
    // lock has been requested by another thread since  
    // the call to tryOptimisticRead()  
    boolean validate(long stamp)
```

Code Idiom for Optimistic Read

```
public double optimisticRead() {
    long stamp = sl.tryOptimisticRead();
    double currentState1 = state1,
           currentState2 = state2, ... etc.;
    if (!sl.validate(stamp)) {
        stamp = sl.readLock();
        try {
            currentState1 = state1;
            currentState2 = state2, ... etc.;
        } finally {
            sl.unlockRead(stamp);
        }
    }
    return calculateSomething(currentState1, currentState2);
}
```

Code Idiom for Optimistic Read

```
public double optimisticRead() {  
    long stamp = sl.tryOptimisticRead();  
    double currentState1 = state1,  
           currentState2 = state2, ... etc.;  
    if (!sl.validate(stamp)) {  
        stamp = sl.readLock();  
        try {  
            currentState1 = state1;  
            currentState2 = state2, ... etc.;  
        } finally {  
            sl.unlockRead(stamp);  
        }  
    }  
    return calculateSomething(currentState1, currentState2);  
}
```

We get a stamp to use for the optimistic read

Code Idiom for Optimistic Read

```
public double optimisticRead() {  
    long stamp = sl.tryOptimisticRead();  
    double currentState1 = state1,  
           currentState2 = state2, ... etc.;  
    if (!sl.validate(stamp)) {  
        stamp = sl.readLock();  
        try {  
            currentState1 = state1;  
            currentState2 = state2, ... etc.;  
        } finally {  
            sl.unlockRead(stamp);  
        }  
    }  
    return calculateSomething(currentState1, currentState2);  
}
```

We read field values into local fields

Code Idiom for Optimistic Read

```
public double optimisticRead() {
    long stamp = sl.tryOptimisticRead();
    double currentState1 = state1,
           currentState2 = state2, ... etc.;
    if (!sl.validate(stamp)) {
        stamp = sl.readLock();
        try {
            currentState1 = state1;
            currentState2 = state2, ... etc.;
        } finally {
            sl.unlockRead(stamp);
        }
    }
    return calculateSomething(currentState1, currentState2);
}
```

Next we validate that no write locks have been issued in the meanwhile

Code Idiom for Optimistic Read

```
public double optimisticRead() {  
    long stamp = sl.tryOptimisticRead();  
    double currentState1 = state1,  
           currentState2 = state2, ... etc.;  
    if (!sl.validate(stamp)) {  
        stamp = sl.readLock();  
        try {  
            currentState1 = state1;  
            currentState2 = state2, ... etc.;  
        } finally {  
            sl.unlockRead(stamp);  
        }  
    }  
    return calculateSomething(currentState1, cur  
}
```

If they have,
then we don't
know if our
state is clean

Thus we acquire a
pessimistic read
lock and read the
state into local
fields

Code Idiom for Optimistic Read

```
public double optimisticRead() {
    long stamp = sl.tryOptimisticRead();
    double currentState1 = state1,
           currentState2 = state2, ... etc.;
    if (!sl.validate(stamp)) {
        stamp = sl.readLock();
        try {
            currentState1 = state1;
            currentState2 = state2, ... etc.;
        } finally {
            sl.unlockRead(stamp);
        }
    }
    return calculateSomething(currentState1, currentState2);
}
```

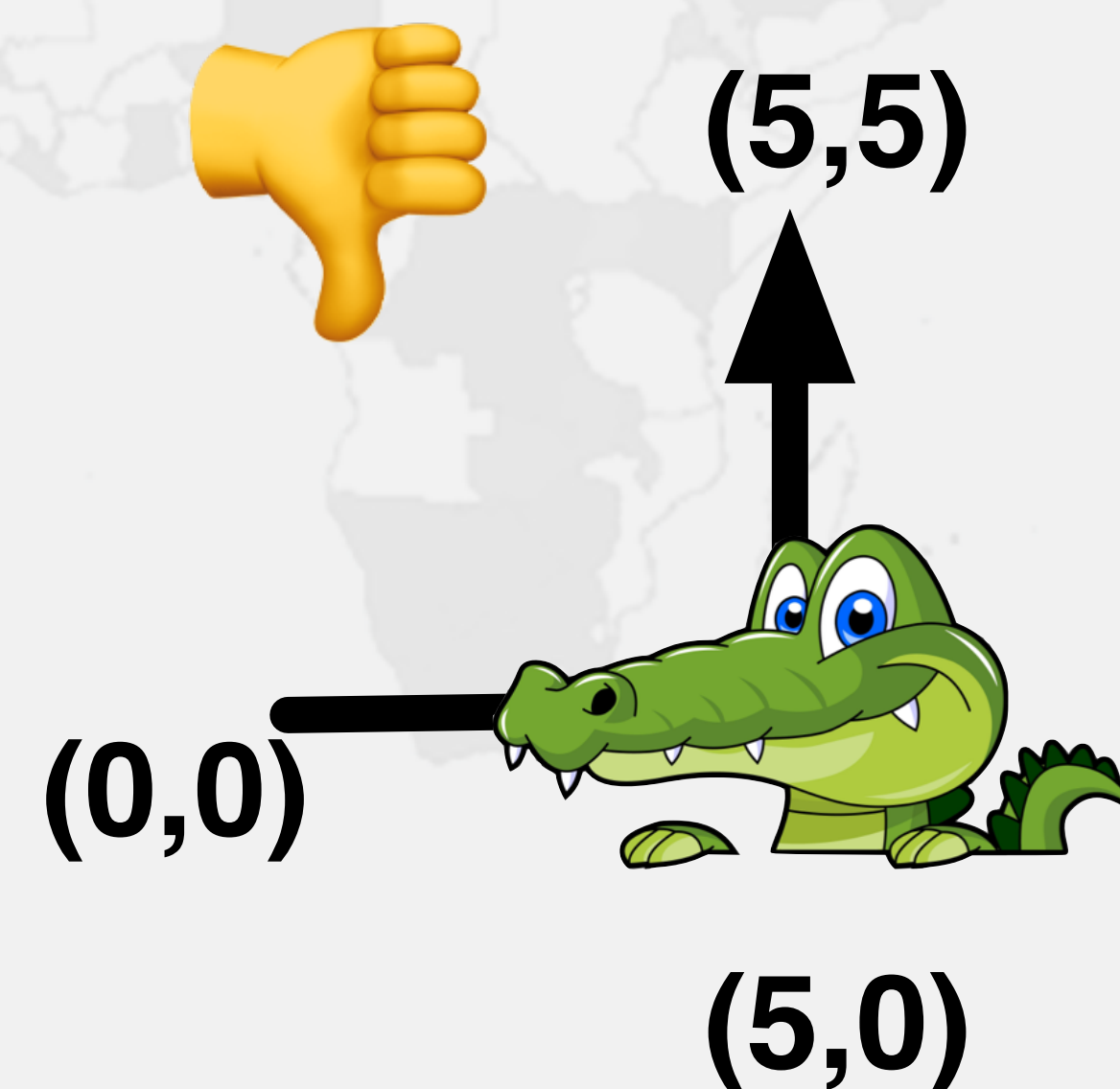
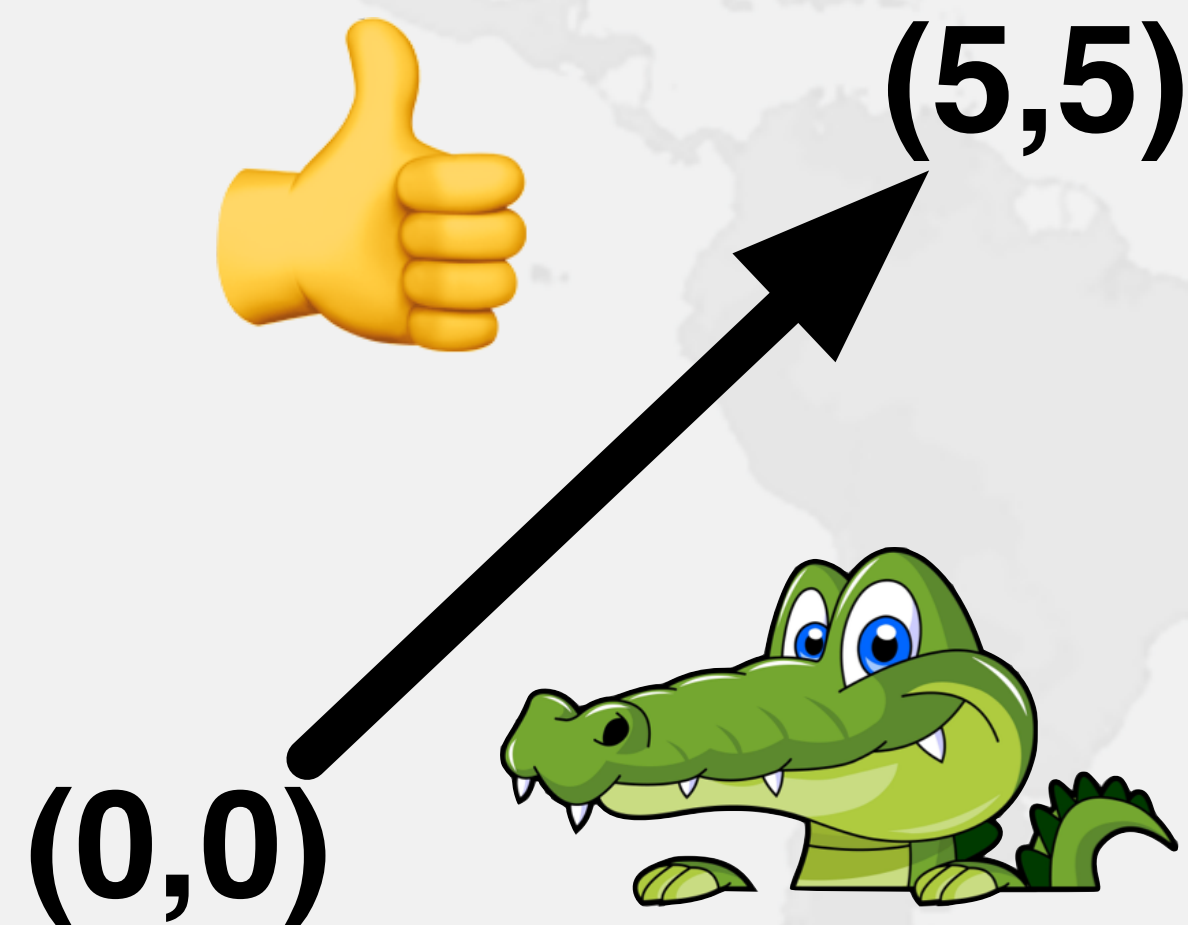

Sifis the Cretan Crocodile (RIP)

- **Poor critter was roaming around Crete**
 - The pet became too big
 - Or hungry
- **Eventually died in our cold winter months**



Introducing the Position Class

- When moving from $(0,0)$ to $(5,5)$, we want to travel in a diagonal line
 - We don't want to ever see our position at $(0,5)$ or especially $(5,0)$



Refactoring Position and IntList

github.com/kabutz/modern-synchronizers



VarHandle



Java 9 VarHandles Instead of Unsafe

- **VarHandles remove biggest temptation to use Unsafe**
 - As fast as Unsafe
- **Can read and write fields of class**
 - `getVolatile() / setVolatile()`
 - `getAcquire() / setRelease()`
 - `getOpaque() / setOpaque()`
 - `get() / set()` - plain
 - `compareAndSet()`, returning boolean
 - `compareAndExchange()`, returning found value always

Refactoring Position from StampedLock to VarHandle

tinyurl.com/techweek19



Surprise

- Remember, tinyurl.com/techweek19 before 17:15

