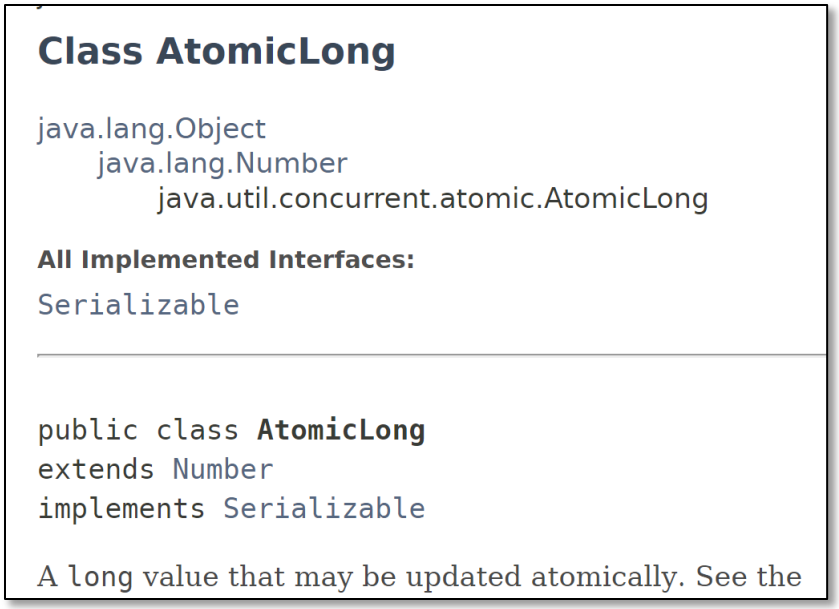
Implementing Java AtomicLong

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 how Java atomic classes & operations provide concurrent programs with lock-free, thread-safe mechanisms to read from & write to single variables
- Note a human known use of atomic operations
- Know how Java atomic operations are implemented & applied
- Recognize how Java atomics classes are implemented
  - Java AtomicLong using the Unsafe class



Class AtomicLong

java.lang.Object
  java.lang.Number
    java.util.concurrent.atomic.AtomicLong

All Implemented Interfaces:

Serializable

```
public class AtomicLong
extends Number
implements Serializable
```

A long value that may be updated atomically. See the

Implementing Java AtomicLong

Implementing Java AtomicLong

- AtomicLong contains a value that is updated atomically

Class AtomicLong

java.lang.Object
    java.lang.Number
        java.util.concurrent.atomic.AtomicLong

All Implemented Interfaces:

Serializable

```
public class AtomicLong
extends Number
implements Serializable
```

A `long` value that may be updated atomically. See the `java.util.concurrent.atomic` package specification for description of the properties of atomic variables. An `AtomicLong` is used in applications such as atomically incremented sequence numbers, and cannot be used as a replacement for a `Long`. However, this class does extend `Number` to allow uniform access by tools and utilities that deal with numerically-based classes.

Since:

1.5

See Also:

Serialized Form

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/atomic/AtomicLong.html

Implementing Java AtomicLong

- AtomicLong uses the method Unsafe.compareAndSwapLong() in Java 8 & before

```
10: /**
11:  * A <tt>long</tt> value that may be updated atomically.  See the
12:  * {@link java.util.concurrent.atomic} package specification for
13:  * description of the properties of atomic variables. An
14:  * <tt>AtomicLong</tt> is used in applications such as atomically
15:  * incremented sequence numbers, and cannot be used as a replacement
16:  * for a {@link java.lang.Long}. However, this class does extend
17:  * <tt>Number</tt> to allow uniform access by tools and utilities that
18:  * deal with numerically-based classes.
19:  *
20:  * @since 1.5
21:  * @author Doug Lea
22:  */
23: public class AtomicLong extends Number implements java.io.Serializable {
24:     private static final long serialVersionUID = 1927816293512124184L;
25:
26:     // setup to use Unsafe.compareAndSwapLong for updates
27:     private static final Unsafe unsafe = Unsafe.getUnsafe();
28:     private static final long valueOffset;
29:
30:     /**
31:      * Records whether the underlying JVM supports lockless
32:      * CompareAndSet for longs. While the unsafe.CompareAndSetLong
33:      * method works in either case, some constructions should be
34:      * handled at Java level to avoid locking user-visible locks.
35:      */
36:     static final boolean VM_SUPPORTS_LONG_CAS = VMSupportsCS8();
37:
38:     /**
39:      * Returns whether underlying JVM supports lockless CompareAndSet
40:      * for longs. Called only once and cached in VM_SUPPORTS_LONG_CAS.
41:      */
42:     private static native boolean VMSupportsCS8();
43:
44:     static {
45:       try {
46:         valueOffset = unsafe.objectFieldOffset
47:             (AtomicLong.class.getDeclaredField("value"));
48:       } catch (Exception ex) { throw new Error(ex); }
49:     }
50:
51:     private volatile long value;
```

See java/util/concurrent/atomic/AtomicLong.java

Implementing Java AtomicLong

- AtomicLong uses the method Unsafe.compareAndSwapLong() in Java 8 & before

This volatile field will be read from & written to atomically via CAS operations

```
public class AtomicLong
  ... {
  private volatile long value;
  ...
  private static final Unsafe unsafe
    = Unsafe.getUnsafe();

  private static final long
    valueOffset;

  static {
    ...
    valueOffset = unsafe.
      objectFieldOffset
        (AtomicLong.class.
           getDeclaredField("value"));
    ...
  }
...
```

See en.wikipedia.org/wiki/Volatile_(computer_programming)#In_Java

Implementing Java AtomicLong

- AtomicLong uses the method Unsafe.compareAndSwapLong() in Java 8 & before

```
public class AtomicLong
   ... {
   private volatile long value;
   ...
   private static final Unsafe unsafe
     = Unsafe.getUnsafe();

   private static final long
     valueOffset;

   static {
     ...
     valueOffset = unsafe.
       objectFieldOffset
         (AtomicLong.class.
           getDeclaredField("value"));
     ...
   }
...
```

Java reflection is used to determine & store the offset of volatile 'value'

See docs.oracle.com/javase/tutorial/reflect

Implementing Java AtomicLong

- AtomicLong uses the method Unsafe.compareAndSwapLong() in Java 8 & before

```
public final class Unsafe {
  public final long getAndAddLong
                        (Object o,
                         long offset,
                         long delta) {
    long v;
    do {
        v = getIntVolatile
                (o, offset);
    } while (!compareAndSwapLong
                (o, offset,
                 v, v + delta));
    return v;
  }
```

Unsafe.getAndAddLong() atomically updates a value at an offset in the object

See jdk8/jdk/file/687fd7c7986d/src/share/classes/sun/misc/Unsafe.java

Implementing Java AtomicLong

- AtomicLong uses the method Unsafe.compareAndSwapLong() in Java 8 & before

```
public final class Unsafe {
  public final long getAndAddLong
                              (Object o,
                               long offset,
                               long delta) {
    long v;
    do {
        v = getIntVolatile
                (o, offset);
    } while (!compareAndSwapLong
                (o, offset,
                 v, v + delta));
    return v;
  }
```

This "lock-free" call runs atomically

Implementing Java AtomicLong

- AtomicLong uses the method Unsafe.compareAndSwapLong() in Java 8 & before

```
public final class Unsafe {
  public final long getAndAddLong
                              (Object o,
                               long offset,
                               long delta) {
    long v;
    do {
      v = getIntVolatile
            (o, offset);
    } while (!compareAndSwapLong
                  (o, offset,
                   v, v + delta));
    return v;
  }
```

The 'offset' is relative to the start of object 'o'

Implementing Java AtomicLong

- AtomicLong uses the method Unsafe.compareAndSwapLong() in Java 8 & before

```
public final class Unsafe {
  public final long getAndAddLong
                              (Object o,
                               long offset,
                               long delta) {
      long v;
      do {
          v = getIntVolatile
                 (o, offset);
      } while (!compareAndSwapLong
                  (o, offset,
                   v, v + delta));
      return v;
  }
```

'v' is the value at 'offset' into object 'o'

Implementing Java AtomicLong

- AtomicLong uses the method Unsafe.compareAndSwapLong() in Java 8 & before

```
public final class Unsafe {
  public final long getAndAddLong
                              (Object o,
                               long offset,
                               long delta) {
    long v;
    do {
      v = getIntVolatile
                 (o, offset);
    } while (!compareAndSwapLong
                  (o, offset,
                  v, v + delta));
    return v;
  }
```

'delta' is atomically added to value 'v' iff 'v' hasn't changed since it was read

Implementing Java AtomicLong

- AtomicLong uses the method Unsafe.compareAndSwapLong() in Java 8 & before

```
public class AtomicLong
  ... {

  private volatile long value;
  ...
  public final long getAndIncrement(){
    return unsafe
      .getAndAddLong(this,
                     valueOffset,
                     1L);
  }

  public final long getAndDecrement(){
     return unsafe
       .getAndAddLong(this,
                      valueOffset,
                      -1L);
  }
```

The Unsafe.getAndAddLong() method is used to increment & decrement values atomically!

Implementing Java AtomicLong

- AtomicLong uses the method Unsafe.compareAndSwapLong() in Java 8 & before

```
public class AtomicLong
  ... {

  private volatile long value;
  ...
  public final boolean compareAndSet
   (long expect, long update) {
     return unsafe
      .compareAndSwapLong
        (this,
         valueOffset,
         expect,
         update);
  }
  ...
```

Atomically sets value to given updated value if current value equals the expected value

Implementing Java AtomicLong

- AtomicLong uses the method Unsafe.compareAndSwapLong() in Java 8 & before

```
public class AtomicLong
  ... {

  private volatile long value;
  ...
  public final boolean compareAndSet
   (long expect, long update) {
     return unsafe
      .compareAndSwapLong
        (this,
         valueOffset,
         expect,
         update);
  }
  ...
```

Unsafe.compareAndSwapLong() attempts to update value atomically!

Implementing Java AtomicLong

- AtomicLong getAndUpdate() uses compareAndSet()

Atomically update current value with results of applying the given function, returning previous value

```
public class AtomicLong
   ... {
   ...

   public final long getAndUpdate
        (LongUnaryOperator updateFunc) {
     long prev, next;
     do {
       prev = get();
       next = updateFunction
                 .applyAsLong(prev);
     } while (!compareAndSet(prev,
                             next));
     return prev;
   }
   ...
```

Implementing Java AtomicLong

- AtomicLong getAndUpdate() uses compareAndSet()

```
public class AtomicLong
   ... {
   ...

   public final long getAndUpdate
        (LongUnaryOperator updateFunc) {
     long prev, next;
     do {
       prev = get();
       next = updateFunction
                 .applyAsLong(prev);
     } while (!compareAndSet(prev,
                             next));
     return prev;
   }
   ...
```

Get the current value

Implementing Java AtomicLong

- AtomicLong getAndUpdate() uses compareAndSet()

```
public class AtomicLong
   ... {
   ...

   public final long getAndUpdate
        (LongUnaryOperator updateFunc) {
     long prev, next;
     do {
       prev = get();
       next = updateFunction
                 .applyAsLong(prev);
     } while (!compareAndSet(prev,
                             next));
     return prev;
   }
   ...
```

Get results of function

Implementing Java AtomicLong

- AtomicLong getAndUpdate() uses compareAndSet()

```
public class AtomicLong
   ... {
   ...

   public final long getAndUpdate
        (LongUnaryOperator updateFunc) {
     long prev, next;
     do {
       prev = get();
       next = updateFunction
                 .applyAsLong(prev);
     } while (!compareAndSet(prev,
                             next));
     return prev;
   }
   ...
```

Atomically update current value with results of applying the given function

Implementing Java AtomicLong

- AtomicLong getAndUpdate() uses compareAndSet()

```
public class AtomicLong
   ... {
   ...

   public final long getAndUpdate
        (LongUnaryOperator updateFunc) {
     long prev, next;
     do {
       prev = get();
       next = updateFunction
                 .applyAsLong(prev);
     } while (!compareAndSet(prev,
                             next));
     return prev;
   }
   ...
```

Return the previous value

Implementing Java AtomicLong

- AtomicLong updateAndGet() also uses compareAndSet()

```
public class AtomicLong
   ... {
   ...

   public final long updateAndGet
        (LongUnaryOperator updateFunc) {
     long prev, next;
     do {
       prev = get();
       next = updateFunction
              .applyAsLong(prev);
     } while (!compareAndSet(prev,
                             next));
     return next;
   }
```

Similar to getAndUpdate()

Implementing Java AtomicLong

- AtomicLong updateAndGet() also uses compareAndSet()

```
public class AtomicLong
   ... {
   ...

   public final long updateAndGet
        (LongUnaryOperator updateFunc) {
     long prev, next;
     do {
       prev = get();
       next = updateFunction
              .applyAsLong(prev);
     } while (!compareAndSet(prev,
                             next));
     return next;
   }
```

However, it returns the **`next`** *value*

End of Implementing Java AtomicLong