Java Atomic Classes & Operations: Implementing Java AtomicLong & AtomicBoolean

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
- Recognize how the Java AtomicLong & AtomicBoolean classes are implemented

Class AtomicBoolean

java.lang.Object
    java.util.concurrent.atomic.AtomicBoolean

All Implemented Interfaces:

Serializable

```
public class AtomicBoolean
extends Object
implements Serializable
```

A boolean value that may be updated atomically. See the

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 method Unsafe.compareAndSwapLong()

```java
/**
 * A <tt>long</tt> value that may be updated atomically.  See the
 * {@link java.util.concurrent.atomic} package specification for
 * description of the properties of atomic variables. An
 * <tt>AtomicLong</tt> is used in applications such as atomically
 * incremented sequence numbers, and cannot be used as a replacement
 * for a {@link java.lang.Long}. However, this class does extend
 * <tt>Number</tt> to allow uniform access by tools and utilities that
 * deal with numerically-based classes.
 *
 * @since 1.5
 * @author Doug Lea
 */
public class AtomicLong extends Number implements java.io.Serializable {
    private static final long serialVersionUID = 1927816293512124184L;

    // setup to use Unsafe.compareAndSwapLong for updates
    private static final Unsafe unsafe = Unsafe.getUnsafe();
    private static final long valueOffset;

    /**
     * Records whether the underlying JVM supports lockless
     * CompareAndSet for longs. While the unsafe.CompareAndSetLong
     * method works in either case, some constructions should be
     * handled at Java level to avoid locking user-visible locks.
     */
    static final boolean VM_SUPPORTS_LONG_CAS = VMSupportsCS8();

    /**
     * Returns whether underlying JVM supports lockless CompareAndSet
     * for longs. Called only once and cached in VM_SUPPORTS_LONG_CAS.
     */
    private static native boolean VMSupportsCS8();

    static {
      try {
        valueOffset = unsafe.objectFieldOffset
            (AtomicLong.class.getDeclaredField("value"));
      } catch (Exception ex) { throw new Error(ex); }
    }

    private volatile long value;
```

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

Implementing Java AtomicLong

- AtomicLong uses method Unsafe.compareAndSwapLong()

```
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"));
    ...
  }
...
```

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

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

Implementing Java AtomicLong

- AtomicLong uses method Unsafe.compareAndSwapLong()

```
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 method Unsafe.compareAndSwapLong()

```
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 www.docjar.com/html/api/sun/misc/Unsafe.java.html

Implementing Java AtomicLong

- AtomicLong uses method Unsafe.compareAndSwapLong()

```
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 method Unsafe.compareAndSwapLong()

```
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 method Unsafe.compareAndSwapLong()

```
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 method Unsafe.compareAndSwapLong()

```
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 method Unsafe.compareAndSwapLong()

```
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 method Unsafe.compareAndSwapLong()

```
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 method Unsafe.compareAndSwapLong()

```
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 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 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 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 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 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 AtomicBoolean

Implementing Java AtomicBoolean

- AtomicBoolean contains a value field that is updated atomically

Class AtomicBoolean

java.lang.Object
    java.util.concurrent.atomic.AtomicBoolean

All Implemented Interfaces:

Serializable

```
public class AtomicBoolean
extends Object
implements Serializable
```

A `boolean` value that may be updated atomically. See the `java.util.concurrent.atomic` package specification for description of the properties of atomic variables. An `AtomicBoolean` is used in applications such as atomically updated flags, and cannot be used as a replacement for a `Boolean`.

Since:

1.5

See Also:

Serialized Form

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

Implementing Java AtomicBoolean

- AtomicBoolean uses method Unsafe.compareAndSwapInt()

```java
/**
 * A {@code boolean} value that may be updated atomically. See the
 * {@link java.util.concurrent.atomic} package specification for
 * description of the properties of atomic variables. An
 * {@code AtomicBoolean} is used in applications such as atomically
 * updated flags, and cannot be used as a replacement for a
 * {@link java.lang.Boolean}.
 *
 * @since 1.5
 * @author Doug Lea
 */
public class AtomicBoolean implements java.io.Serializable {
    private static final long serialVersionUID = 4654671469794556979L;
    // setup to use Unsafe.compareAndSwapInt for updates
    private static final Unsafe unsafe = Unsafe.getUnsafe();
    private static final long valueOffset;

    static {
        try {
            valueOffset = unsafe.objectFieldOffset
                (AtomicBoolean.class.getDeclaredField("value"));
        } catch (Exception ex) { throw new Error(ex); }
    }

    private volatile int value;

    /**
     * Creates a new {@code AtomicBoolean} with the given initial value.
     *
     * @param initialValue the initial value
     */
    public AtomicBoolean(boolean initialValue) {
        value = initialValue ? 1 : 0;
    }
```

See src/share/classes/java/util/concurrent/atomic/AtomicBoolean.java

Implementing Java AtomicBoolean

- AtomicBoolean uses method Unsafe.compareAndSwapInt()

```
public class AtomicBoolean ... {
  private static final Unsafe unsafe
    = ...;

  private static final long
    valueOffset;

  private volatile int value;

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

Compute the offset of the 'value' field from the beginning of the object

See www.docjar.com/docs/api/sun/misc/Unsafe.html#objectFieldOffset

Implementing Java AtomicBoolean

- AtomicBoolean uses method Unsafe.compareAndSwapInt()

```
public class AtomicBoolean ... {
  private static final Unsafe unsafe
    = ...;

  private static final long
    valueOffset;

  private volatile int value;

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

Uses the Java reflection API

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

Implementing Java AtomicBoolean

- AtomicBoolean uses method Unsafe.compareAndSwapInt()

```
public class AtomicBoolean ... {
  private static final Unsafe unsafe
    = ...;

  private static final long
    valueOffset;

  private volatile int value;

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

Note the "value" field is volatile

See en.wikipedia.org/wiki/Volatile_variable#In_Java

Implementing Java AtomicBoolean

- AtomicBoolean uses method Unsafe.compareAndSwapInt()

```
public class AtomicBoolean ... {
  ...
  public final boolean compareAndSet
                 (boolean expected,
                  boolean updated){
    int e = expected ? 1 : 0;
    int u = updated ? 1 : 0;
    return unsafe.compareAndSwapInt
        (this, valueOffset, e, u);
  }
  ...
```

See www.docjar.com/docs/api/sun/misc/Unsafe.html#compareAndSwapInt

Implementing Java AtomicBoolean

- AtomicBoolean uses method Unsafe.compareAndSwapInt()

```
public class AtomicBoolean ... {
  ...
  public final boolean compareAndSet
                 (boolean expected,
                  boolean updated){
    int e = expected ? 1 : 0;
    int u = updated ? 1 : 0;
    return unsafe.compareAndSwapInt
        (this, valueOffset, e, u);
  }
  ...
```

Atomically update field at valueOffset to 'updated' iff it's currently holding 'expected'

See www.docjar.com/docs/api/sun/misc/Unsafe.html#compareAndSwapInt

Implementing Java AtomicBoolean

- AtomicBoolean uses method Unsafe.compareAndSwapInt()

```
public class AtomicBoolean ... {
  ...
  public final boolean compareAndSet
                 (boolean expected,
                  boolean updated){
    int e = expected ? 1 : 0;
    int u = updated ? 1 : 0;
    return unsafe.compareAndSwapInt
        (this, valueOffset, e, u);
  }
  ...
```

Returns true if successful, whereas false indicates that the actual value was not equal to the expected value

See www.docjar.com/docs/api/sun/misc/Unsafe.html#compareAndSwapInt

Implementing Java AtomicBoolean

- AtomicBoolean uses method Unsafe.compareAndSwapInt()

```
public class AtomicBoolean ... {
  ...
  public final boolean compareAndSet
                 (boolean expected,
                  boolean updated){
    int e = expected ? 1 : 0;
    int u = updated ? 1 : 0;
    return unsafe.compareAndSwapInt
        (this, valueOffset,
         e, u);
  }

  public final void set(boolean
                        newValue) {
    value = newValue ? 1 : 0;
  }
  ...
```

Unconditionally sets 'value' to the given 'newValue' via an atomic write on this field

End of Atomic Classes & Operations: Implementing Java AtomicLong & AtomicBoolean