Applying 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
- Be aware of how to apply Java AtomicLong in practice

```java
class Random ...
{
    public Random()
    {
        this(seedUniquifier()
            ^ System.nanoTime());
    }

    private static long seedUniquifier()
    {
        for (;;)
        {
            long s = seedUniquifier.get();
            long next =
                s * 181783497276652981L;
            if (seedUniquifier
                .compareAndSet(s, next))
                return next;
        }
    }

    private static final AtomicLong seedUniquifier =
        new AtomicLong(8682522807148012L);
```
Applying Java AtomicLong
The Java Random class uses an `AtomicLong` to generate seeds that are reasonably unique.

```java
class Random {
    public Random() {
        this(seedUniquifier()
            ^ System.nanoTime());
    }

    private static long seedUniquifier(){
        for (; ;) {
            long s = seedUniquifier.get();
            long next =
                s * 181783497276652981L;
            if (seedUniquifier
                .compareAndSet(s, next))
                return next;
        }
    }

    private static final AtomicLong seedUniquifier = new
        AtomicLong(8682522807148012L);
}
```

See `share/classes/java/util/Random.java`
Implementing Java AtomicLong

- The Java Random class uses an AtomicLong to generate seeds that are reasonably unique.

```java
class Random ... {
    public Random() {
        this(seedUniquifier()
            ^ System.nanoTime());
    }

    private static long seedUniquifier(){
        for (;;) {
            long s = seedUniquifier.get();
            long next =
                s * 181783497276652981L;
            if (seedUniquifier
                .compareAndSet(s, next))
                return next;
        }
    }

    private static final AtomicLong seedUniquifier = new
        AtomicLong(8682522807148012L);
```
The Java Random class uses an AtomicLong to generate seeds that are reasonably unique.

```java
class Random ... {
    public Random() {
        this(seedUniquifier()
            ^ System.nanoTime());
    }

    private static long seedUniquifier() {
        for (;;) {
            long s = seedUniquifier.get();
            long next =
                s * 181783497276652981L;
            if (seedUniquifier
                .compareAndSet(s, next))
                return next;
        }
    }
}
```

An AtomicLong that is initialized to a large value:

```java
private static final AtomicLong seedUniquifier = new
    AtomicLong(8682522807148012L);
```
The Java Random class uses an `AtomicLong` to generate seeds that are reasonably unique.

**Factory method that atomically generates the next “unique” seed value**

```java
private static long seedUniquifier() {
    for (;;) {
        long s = seedUniquifier.get();
        long next =
            s * 181783497276652981L;
        if (seedUniquifier.compareAndSet(s, next))
            return next;
    }
}
```

```java
private static final AtomicLong seedUniquifier = new AtomicLong(8682522807148012L);
```
The Java Random class uses an AtomicLong to generate seeds that are reasonably unique.

```java
class Random ... {
    public Random() {
        this(seedUniquifier()
            ^ System.nanoTime());
    }

    private static long seedUniquifier() {
        for (;;) {
            long s = seedUniquifier.get();
            long next =
                s * 181783497276652981L;
            if (seedUniquifier
                .compareAndSet(s, next))
                return next;
        }
    }

    private static final AtomicLong seedUniquifier = new
        AtomicLong(8682522807148012L);
}
```

This code runs in a loop for reasons we’ll discuss shortly!
The Java Random class uses an AtomicLong to generate seeds that are reasonably unique.

```java
class Random ...
{
    public Random()
    {
        this(seedUniquifier()
            ^ System.nanoTime());
    }

    private static long seedUniquifier()
    {
        for (; ;)
        {
            long s = seedUniquifier.get();
            long next =
                s * 181783497276652981L;
            if (seedUniquifier
                .compareAndSet(s, next))
                return next;
        }
    }

    private static final AtomicLong seedUniquifier =
        new AtomicLong(8682522807148012L);
}
```

Multiple threads running on multiple cores can call get() concurrently.
Implementing Java AtomicLong

- The Java Random class uses an AtomicLong to generate seeds that are reasonably unique.

```java
class Random ...
{
    public Random()
    {
        this(seedUniquifier()
            ^ System.nanoTime());
    }

    private static long seedUniquifier()
    {
        for (; ;)
        {
            long s = seedUniquifier.get();
            long next =
                s * 181783497276652981L;
            if (seedUniquifier
                .compareAndSet(s, next))
                return next;
        }
    }

    private static final AtomicLong seedUniquifier =
        new AtomicLong(8682522807148012L);
}
```

This computation of next is deterministic.
The Java Random class uses an AtomicLong to generate seeds that are reasonably unique.

class Random ...
{
    public Random()
    {
        this(seedUniquifier()
            ^ System.nanoTime());
    }

    private static long seedUniquifier(){
        for (; ; )
        {
            long s = seedUniquifier.get();
            long next =
                s * 181783497276652981L;
            if (seedUniquifier
                .compareAndSet(s, next))
                return next;
        }
    }
}

private static final AtomicLong
seedUniquifier = new
AtomicLong(8682522807148012L);

Try to set the computed next seed atomically, which succeeds only if s is still the current seed value.

cAS()  cAS()  cAS()  cAS()
⇒  ⇒  ⇒  ⇒

compareAndSet() is only called once per loop, per thread & only succeeds in one thread.
The Java Random class uses an AtomicLong to generate seeds that are reasonably unique.

```java
class Random ... {
    public Random() {
        this(seedUniquifier()
            ^ System.nanoTime());
    }

    private static long seedUniquifier() {
        for (;;) {
            long s = seedUniquifier.get();
            long next =
                s * 181783497276652981L;
            if (seedUniquifier
                .compareAndSet(s, next))
                return next;
        }
    }

    private static final AtomicLong seedUniquifier =
        new AtomicLong(8682522807148012L);
}
```

Return the next seed value if compareAndSet() succeeded.
Implementing Java AtomicLong

- The Java Random class uses an AtomicLong to generate seeds that are reasonably unique.

```java
class Random ...
{
    public Random()
    {
        this(seedUniquifier()
            ^ System.nanoTime());
    }

    private static long seedUniquifier()
    {
        for (;;) {
            long s = seedUniquifier.get();
            long next =
                s * 181783497276652981L;
            if (seedUniquifier
                .compareAndSet(s, next))
                return next;
        }
    }

    private static final AtomicLong seedUniquifier
    = new AtomicLong(8682522807148012L);
}
```

Otherwise, loop again & keep trying until success.
Implementing Java AtomicLong

• The Java Random class uses an AtomicLong to generate seeds that are reasonably unique
• compareAndSet() is used to ensure unique seeds in the face of multiple cores

If this code is run concurrently by multiple threads on multiple cores the resulting seeds may be identical!

```java
class Random ... {
    public Random() {
        this(seedUniquifier()
            ^ System.nanoTime());
    }

    private static long seedUniquifier(){
        seedUniquifier
            .set(seedUniquifier.get() * 181783497276652981L);
        return seedUniquifier.get();
    }

    private static final AtomicLong seedUniquifier = new AtomicLong(8682522807148012L);
    ...
```

set() set() set() set()
Implementing Java AtomicLong

- The Java Random class uses an AtomicLong to generate seeds that are reasonably unique.
- `compareAndSet()` is used to ensure unique seeds in the face of multiple cores.

```java
public Random() {
    this(seedUniquifier() ^ System.nanoTime());
}

private static long seedUniquifier() {
    return seedUniquifier.
        .updateAndGet(cur -> cur * 181783497276652981L);
}

private static final AtomicLong seedUniquifier = new
    AtomicLong(8682522807148012L);
```

Even this clever Java 8+ version suffers from the same problems.
End of Applying Java AtomicLong