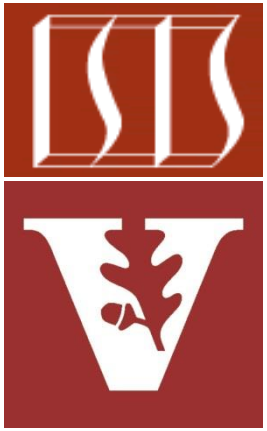
Java Atomic Classes & Operations: Implementing Java Atomic Operations



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

Concurrency

And few words about concurrency with `Unsafe`. `compareAndSwap` methods are atomic and can be used to implement high-performance lock-free data structures.

For example, consider the problem to increment value in the shared object using lot of threads.

First we define simple interface `Counter`:

```
interface Counter {
    void increment();
    long getCounter();
}
```

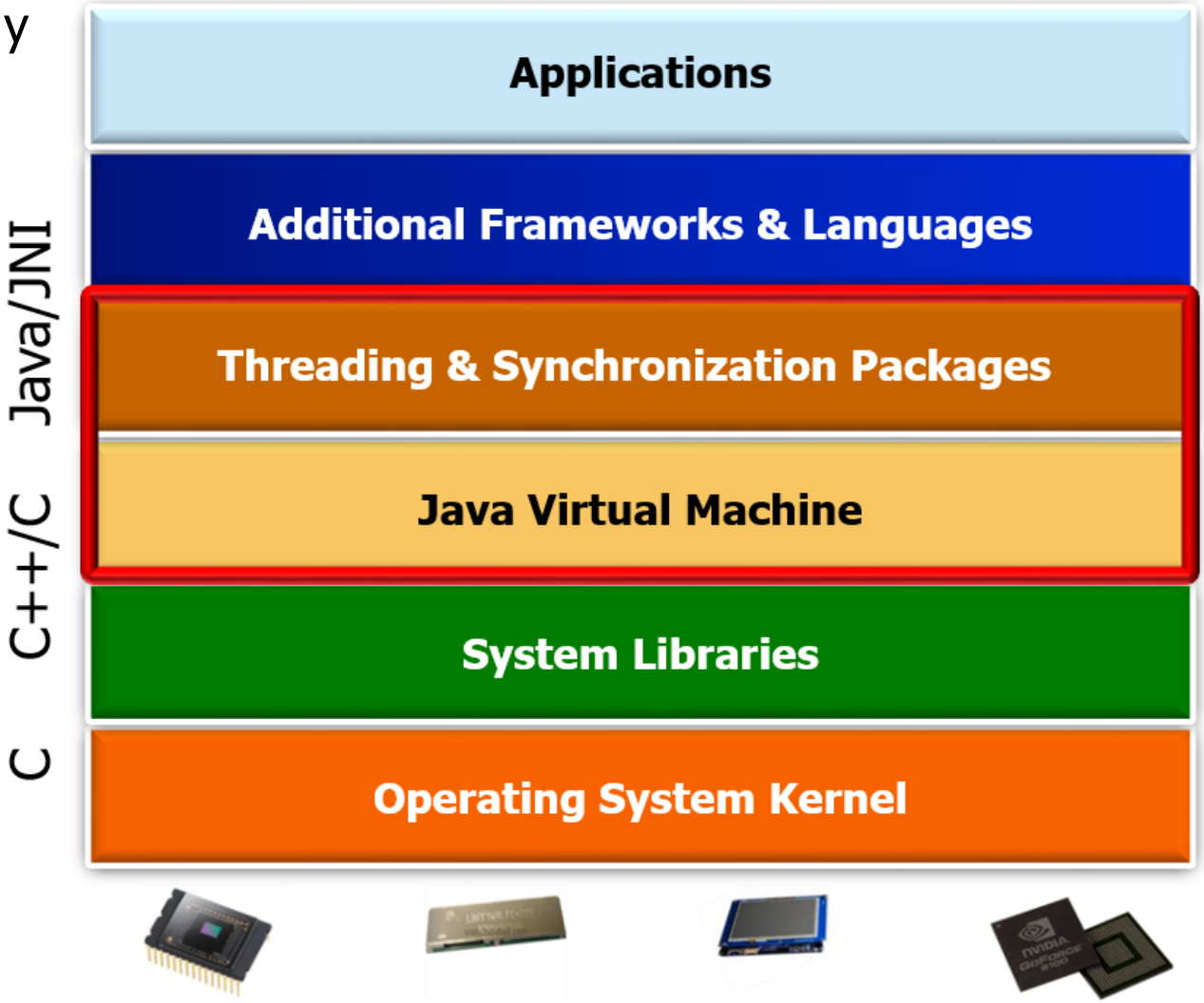
Then we define worker thread `CounterClient`, that uses `Counter`:

```
class CounterClient implements Runnable {
    private Counter c;
    private int num;

    public CounterClient(Counter c, int num) {
        this.c = c;
        this.num = num;
    }

    @Override
    public void run() {
        for (int i = 0; i < num; i++) {
            c.increment();
        }
    }
}
```

Implementating Java Atomic Operations

Implementing Java Atomic Operations

- Java uses CAS extensively in the JVM & portions of java.util.concurrent*



Implementing Java Atomic Operations

- Java uses CAS extensively in the JVM & portions of java.util.concurrent*
  - e.g., compareAndSwapLong()

```
public final class Unsafe {
  public final native boolean
    compareAndSwapLong(Object o,
                       long offset,
                       long expected,
                       long updated);
```

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

Implementing Java Atomic Operations

- Java uses CAS extensively in the JVM & portions of java.util.concurrent*
  - e.g., compareAndSwapLong()

```
public final class Unsafe {
  public final native boolean
    compareAndSwapLong(Object o,
                       long offset,
                       long expected,
                       long updated) {
    START_ATOMIC();
    int *base = (int *) o;
    int oldValue = base[offset];
    if (oldValue == expected)
      base[offset] = updated;
    END_ATOMIC();
    return oldValue;
  }
  ...
```

This C-like pseudo-code atomically compares the contents of memory with an expected value, modifies the contents to an updated value iff they are the same, & returns the old value

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

Implementing Java Atomic Operations

- Java uses CAS extensively in the JVM & portions of java.util.concurrent*
  - e.g., compareAndSwapLong()

```
public final class Unsafe {
  public final native boolean
    compareAndSwapLong(Object o,
                       long offset,
                       long expected,
                       long updated) {
    START_ATOMIC();
    int *base = (int *) o;
    int oldValue = base[offset];
    if (oldValue == expected)
      base[offset] = updated;
    END_ATOMIC();
    return oldValue;
  }
  ...
```

This C-like pseudo-code atomically compares the contents of memory with an expected value, modifies the contents to an updated value iff they are the same, & returns the old value

Implementing Java Atomic Operations

- Java uses CAS extensively in the JVM & portions of java.util.concurrent*
  - e.g., compareAndSwapLong()

```
public final class Unsafe {
  public final native boolean
    compareAndSwapLong(Object o,
                       long offset,
                       long expected,
                       long updated) {
    START_ATOMIC();
    int *base = (int *) o;
    int oldValue = base[offset];
    if (oldValue == expected)
      base[offset] = updated;
    END_ATOMIC();
    return oldValue;
  }
  ...
```

This C-like pseudo-code atomically compares the contents of memory with an expected value, modifies the contents to an updated value iff they are the same, & returns the old value

Implementing Java Atomic Operations

- Java uses CAS extensively in the JVM & portions of java.util.concurrent*
  - e.g., compareAndSwapLong()

```
public final class Unsafe {
  public final native boolean
    compareAndSwapLong(Object o,
                       long offset,
                       long expected,
                       long updated) {
    START_ATOMIC();
    int *base = (int *) o;
    int oldValue = base[offset];
    if (oldValue == expected)
      base[offset] = updated;
    END_ATOMIC();
    return oldValue;
  }
  ...
```

This C-like pseudo-code atomically compares the contents of memory with an expected value, modifies the contents to an updated value iff they are the same, & returns the old value

Implementing Java Atomic Operations

- Java uses CAS extensively in the JVM & portions of java.util.concurrent*
  - e.g., compareAndSwapLong()

```
public final class Unsafe {
  public final native boolean
    compareAndSwapLong(Object o,
                       long offset,
                       long expected,
                       long updated) {
    START_ATOMIC();
    int *base = (int *) o;
    int oldValue = base[offset];
    if (oldValue == expected)
      base[offset] = updated;
    END_ATOMIC();
    return oldValue;
  }
  ...
```

This C-like pseudo-code atomically compares the contents of memory with an expected value, modifies the contents to an updated value iff they are the same, & returns the old value

End of Atomic Classes & Operations: Implementing Java Atomic Operations