Java CountDownLatch: Structure & Functionality

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 the structure & functionality of Java CountDownLatch
Overview of Java CountDownLatch
Overview of Java CountDownLatch

- Implements one (of several) Java barrier synchronizers

```java
public class CountDownLatch {
    ...
}
```

**Class CountDownLatch**

```java
java.lang.Object
    java.util.concurrent.CountDownLatch
```

A synchronization aid that allows one or more threads to wait until a set of operations being performed in other threads completes.

A `CountDownLatch` is initialized with a given `count`. The `await` methods block until the current count reaches zero due to invocations of the `countDown()` method, after which all waiting threads are released and any subsequent invocations of `await` return immediately. This is a one-shot phenomenon -- the count cannot be reset. If you need a version that resets the count, consider using a `CyclicBarrier`.

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/CountDownLatch.html](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CountDownLatch.html)
Overview of Java CountDownLatch

- Implements one (of several) Java barrier synchronizers
- Allows one or more threads to wait for the completion of a set of operations being performed in other threads

Class CountDownLatch

```java
public class CountDownLatch {
    ...
Overview of Java CountDownLatch

- Implements one (of several) Java barrier synchronizers
- Allows one or more threads to wait for the completion of a set of operations being performed in other threads
- Well-suited for fixed-size, one-shot "entry" & "exit" barriers

public class CountDownLatch {
...

Class CountDownLatch

java.lang.Object
java.util.concurrent.CountDownLatch

public class CountDownLatch extends Object

A synchronization aid that allows one or more threads to wait until a set of operations being performed in other threads completes.

A CountDownLatch is initialized with a given count. The await methods block until the current count reaches zero due to invocations of the countDown () method, after which all waiting threads are released and any subsequent invocations of await return immediately. This is a one-shot phenomenon -- the count cannot be reset. If you need a version that resets the count, consider using a CyclicBarrier.

CountDownLatch is not designed for use as "cyclic" barriers
Overview of Java CountDownLatch

- Implements one (of several) Java barrier synchronizers
- Allows one or more threads to wait for the completion of a set of operations being performed in other threads
- Well-suited for fixed-size, one-shot “entry” & “exit” barriers

```java
public class CountDownLatch {
    ...
}
```

Class CountDownLatch

java.lang.Object
    java.util.concurrent.CountDownLatch

public class CountDownLatch extends Object

A synchronization aid that allows one or more threads to wait until a set of operations being performed in other threads completes.

A CountDownLatch is initialized with a given count. The await methods block until the current count reaches zero due to invocations of the countDown() method, after which all waiting threads are released and any subsequent invocations of await return immediately. This is a one-shot phenomenon -- the count cannot be reset. If you need a version that resets the count, consider using a CyclicBarrier.
Applies a variant of *Bridge* pattern

```java
class CountDownLatch {
    ...
}
```

See [en.wikipedia.org/wiki/Bridge_pattern](en.wikipedia.org/wiki/Bridge_pattern)
Overview of Java CountDownLatch

- Applies a variant of *Bridge* pattern
- Locking handled by Sync implementor hierarchy

```java
class CountDownLatch {
    private final Sync sync;
    /** Performs sync mechanics */
    ...
}
```
Overview of Java CountDownLatch

- Applies a variant of *Bridge* pattern
- Locking handled by Sync implementor hierarchy
- Inherits functionality from the AbstractQueuedSynchronizer (AQS) class

```java
public class CountDownLatch {
    ... 
    /** Performs sync mechanics */
    private final Sync sync;
    
    /**
     * Synchronization control or CountDownLatch.
     */
    private static final class Sync extends AbstractQueuedSynchronizer {
        ... 
    }
}
```

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/locks/AbstractQueuedSynchronizer.html](docs.oracle.com/javase/8/docs/api/java/util/concurrent/locks/AbstractQueuedSynchronizer.html)
Overview of Java CountDownLatch

- Applies a variant of *Bridge* pattern
- Locking handled by Sync implementor hierarchy
- Inherits functionality from the AbstractQueuedSynchronizer (AQS) class
- However, it doesn’t implement “fair” vs. “non-fair” semantics

```java
public class CountDownLatch {
    ... 
    /** Performs sync mechanics */
    private final Sync sync;
    
    /**
     * Synchronization control or CountDownLatch.
     */
    private static final class Sync extends AbstractQueuedSynchronizer {
        ... 
    }
}
```

See earlier lessons on “Java ReentrantLock”, “Java Semaphore”, & “Java ReentrantReadWriteLock”
Overview of Java CountDownLatch

- Applies a variant of *Bridge* pattern
  - Locking handled by Sync implementor hierarchy
- Inherits functionality from the AbstractQueuedSynchronizer (AQS) class
  - However, it doesn’t implement “fair” vs. “non-fair” semantics
  - Instead, it uses the AQS state to atomically represent the “count”

See [gee.cs.oswego.edu/dl/papers/aqs.pdf](gee.cs.oswego.edu/dl/papers/aqs.pdf)
End of Java CountDownLatch: Structure & Functionality