CompletionService Interface
ThreadLocal
ThreadLocalRandom
CountDownLatch
CyclicBarrier
Concurrent Collections
ConcurrentHashMap
ConcurrentModificationException

Practice Mock Interview →

AtomicIntegerFieldUpdater

Guide to understanding and using AtomicIntegerFieldUpdater.

*If you are interviewing, consider buying our **number#1** course for **Java Multithreading Interviews**.*

Overview

The class `AtomicIntegerFieldUpdater` is one of the three field updater classes. The field updater classes exist primarily for performance reasons. Instead of using atomic variables, one can use ordinary variables that occasionally need to be get and then set atomically. Another reason can be to avoid having atomic fields in objects that are short-lived and frequently created e.g. the next pointer of nodes in a concurrent linked list.

The atomicity guarantees for the updater classes are weaker than those of regular atomic classes because the underlying fields can still be modified directly i.e. without using the updater object. Additionally, the atomicity guarantees for arithmetic methods and `compareAndSet` method stand only with respect to other threads using the updater's methods. The atomic fields present a reflection-based view of an existing `volatile` field that an updater can execute the compare and set method against. Note, that the updater instance isn't tied to any one instance of the target class; rather the updater object can be used to update the target field of any instance of the target class.

Example

classes because the underlying fields can still be modified directly i.e. without using the updater object. Additionally, the atomicity guarantees for arithmetic methods and `compareAndSet` method stand only with respect to other threads using the updater's methods. The atomic fields present a reflection-based view of an existing `volatile` field that an updater can execute the compare and set method against. Note, that the updater instance isn't tied to any one instance of the target class; rather the updater object can be used to update the target field of any instance of the target class.

Example

As an example consider a `Counter` class that is very infrequently incremented or decremented but supports a very high number of read operations. For such a class, we may choose to track the count in an ordinary `int` variable instead of an `AtomicInteger` as we expect the `Counter` instance to be very infrequently updated. If such `Counter` objects are created in very large numbers then the cost savings in terms of space can be significant.

The code for the `Counter` class appears below along with comments.

Java

main.java Saved

```java
import java.util.concurrent.atomic.AtomicIntegerFieldUpdater;

class Demostration {

    static class Counter {
        // volatile int field
        protected volatile int count = 0;

        // ... Rest of the operations exposed by the class
    }

    public static void main( String args[] ) {
```

Run

Note, that in the code widget above if we remove `volatile` with the `int` variable, the updater object will throw an error since only `volatile` fields can be updated using the atomic field updater classes.

← Back lesson
AtomicIntegerArray

Mark As Completed   Next →
AtomicLong