

Java Multithreading for Senior Engineering Interviews / ... / AtomicLongFieldUpdate

## **AtomicLongFieldUpdater**

Guide to understanding and using AtomicLongFieldUpdater.

If you are interviewing, consider buying our number#1 course for <u>Java Multithreading</u> <u>Interviews</u>.

## **Overview**

You'll find the write-up for AtomicLongFieldUpdater similar to the one for AtomicIntegerFieldUpdater, since the two classes are similar in behavior but work with different types.

The class AtomicLongFieldupdater 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.

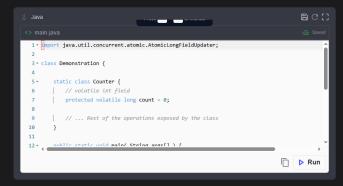
The atomicity guarantees for the updater classes are weaker than those of regular atomic classes because the underlying fields can still be modified directly i.e. without using the updater object. Additionally, the atomicity guarantees for arithmetic methods and compareandset method stand only with respect 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 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**

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 long variable instead of an AtomicLong as we expect the class 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.



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

