compact1, compact2, compact3
java.util.concurrent.atomic

Class AtomicLong

java.lang.Object
    java.lang.Number
        java.util.concurrent.atomic.AtomicLong

All Implemented Interfaces:

`Serializable`

```
public class AtomicLong
extends Number
implements Serializable
```

A `long` value that may be updated atomically. See the `java.util.concurrent.atomic` package specification for description of the properties of atomic variables. An `AtomicLong` is used in applications such as atomically incremented sequence numbers, and cannot be used as a replacement for a `Long`. However, this class does extend `Number` to allow uniform access by tools and utilities that deal with numerically-based classes.

Since:

1.5

See Also:

`Serialized Form`

Constructor Summary

Constructors

Constructor	Description
`AtomicLong()`	Creates a new AtomicLong with initial value `0`.
`AtomicLong(long initialValue)`	Creates a new AtomicLong with the given initial value.

Method Summary

All Methods Instance Methods Concrete Methods

Modifier and Type	Method	Description
`long`	`accumulateAndGet(long x, LongBinaryOperator accumulatorFunction)`	Atomically updates the current value

		with the results of applying the given function to the current and given values, returning the updated value.
`long`	**addAndGet**(long delta)	Atomically adds the given value to the current value.
`boolean`	**compareAndSet**(long expect, long update)	Atomically sets the value to the given updated value if the current value == the expected value.
`long`	**decrementAndGet**()	Atomically decrements by one the current value.
`double`	**doubleValue**()	Returns the value of this `AtomicLong` as a `double` after a widening primitive conversion.
`float`	**floatValue**()	Returns the value of this `AtomicLong` as a `float` after a widening primitive conversion.
`long`	**get**()	Gets the current value.
`long`	**getAndAccumulate**(long x, **LongBinaryOperator** accumulatorFunction)	Atomically updates the current value with the results of

		applying the given function to the current and given values, returning the previous value.
`long`	**`getAndAdd`**`(long delta)`	Atomically adds the given value to the current value.
`long`	**`getAndDecrement`**`()`	Atomically decrements by one the current value.
`long`	**`getAndIncrement`**`()`	Atomically increments by one the current value.
`long`	**`getAndSet`**`(long newValue)`	Atomically sets to the given value and returns the old value.
`long`	**`getAndUpdate`**`(`**`LongUnaryOperator`**` updateFunction)`	Atomically updates the current value with the results of applying the given function, returning the previous value.
`long`	**`incrementAndGet`**`()`	Atomically increments by one the current value.
`int`	**`intValue`**`()`	Returns the value of this `AtomicLong` as an `int` after a narrowing primitive conversion.

`void`	`lazySet(long newValue)`	Eventually sets to the given value.
`long`	`longValue()`	Returns the value of this `AtomicLong` as a `long`.
`void`	`set(long newValue)`	Sets to the given value.
`String`	`toString()`	Returns the String representation of the current value.
`long`	`updateAndGet(LongUnaryOperator updateFunction)`	Atomically updates the current value with the results of applying the given function, returning the updated value.
`boolean`	`weakCompareAndSet(long expect, long update)`	Atomically sets the value to the given updated value if the current value == the expected value.

Methods inherited from class java.lang.Number

`byteValue, shortValue`

Methods inherited from class java.lang.Object

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Constructor Detail

AtomicLong

```
public AtomicLong(long initialValue)
```

Creates a new AtomicLong with the given initial value.

Parameters:

`initialValue` - the initial value

AtomicLong

```
public AtomicLong()
```

Creates a new AtomicLong with initial value `0`.

Method Detail

get

```
public final long get()
```

Gets the current value.

Returns:

the current value

set

```
public final void set(long newValue)
```

Sets to the given value.

Parameters:

`newValue` - the new value

lazySet

```
public final void lazySet(long newValue)
```

Eventually sets to the given value.

Parameters:

`newValue` - the new value

Since:

1.6

getAndSet

```
public final long getAndSet(long newValue)
```

Atomically sets to the given value and returns the old value.

Parameters:

`newValue` - the new value

Returns:

the previous value

compareAndSet

```
public final boolean compareAndSet(long expect,
                                   long update)
```

Atomically sets the value to the given updated value if the current value == the expected value.

Parameters:

`expect` - the expected value

`update` - the new value

Returns:

`true` if successful. False return indicates that the actual value was not equal to the expected value.

weakCompareAndSet

```
public final boolean weakCompareAndSet(long expect,
                                       long update)
```

Atomically sets the value to the given updated value if the current value == the expected value.

May fail spuriously and does not provide ordering guarantees, so is only rarely an appropriate alternative to `compareAndSet`.

Parameters:

`expect` - the expected value

`update` - the new value

Returns:

`true` if successful

getAndIncrement

```
public final long getAndIncrement()
```

Atomically increments by one the current value.

Returns:

the previous value

getAndDecrement

```
public final long getAndDecrement()
```

Atomically decrements by one the current value.

Returns:

the previous value

getAndAdd

```
public final long getAndAdd(long delta)
```

Atomically adds the given value to the current value.

Parameters:

delta - the value to add

Returns:

the previous value

incrementAndGet

```
public final long incrementAndGet()
```

Atomically increments by one the current value.

Returns:

the updated value

decrementAndGet

```
public final long decrementAndGet()
```

Atomically decrements by one the current value.

Returns:

the updated value

addAndGet

```
public final long addAndGet(long delta)
```

Atomically adds the given value to the current value.

Parameters:

delta - the value to add

Returns:

the updated value

getAndUpdate

```
public final long getAndUpdate(LongUnaryOperator updateFunction)
```

Atomically updates the current value with the results of applying the given function, returning the previous value. The function should be side-effect-free, since it may be re-applied when attempted updates fail due to contention among threads.

Parameters:

`updateFunction` - a side-effect-free function

Returns:

the previous value

Since:

1.8

updateAndGet

```
public final long updateAndGet(LongUnaryOperator updateFunction)
```

Atomically updates the current value with the results of applying the given function, returning the updated value. The function should be side-effect-free, since it may be re-applied when attempted updates fail due to contention among threads.

Parameters:

`updateFunction` - a side-effect-free function

Returns:

the updated value

Since:

1.8

getAndAccumulate

```
public final long getAndAccumulate(long x,
                                   LongBinaryOperator accumulatorFunction)
```

Atomically updates the current value with the results of applying the given function to the current and given values, returning the previous value. The function should be side-effect-free, since it may be re-applied when attempted updates fail due to contention among threads. The function is applied with the current value as its first argument, and the given update as the second argument.

Parameters:

`x` - the update value

`accumulatorFunction` - a side-effect-free function of two arguments

Returns:

the previous value

Since:

1.8

accumulateAndGet

```
public final long accumulateAndGet(long x,
                                   LongBinaryOperator accumulatorFunction)
```

Atomically updates the current value with the results of applying the given function to the current and given values, returning the updated value. The function should be side-effect-free, since it may be re-applied when attempted updates fail due to contention among threads. The function is applied with the current value as its first argument, and the given update as the second argument.

Parameters:

`x` - the update value

`accumulatorFunction` - a side-effect-free function of two arguments

Returns:

the updated value

Since:

1.8

toString

```
public String toString()
```

Returns the String representation of the current value.

Overrides:

`toString` in class `Object`

Returns:

the String representation of the current value

intValue

```
public int intValue()
```

Returns the value of this `AtomicLong` as an `int` after a narrowing primitive conversion.

Specified by:

`intValue` in class `Number`

Returns:

the numeric value represented by this object after conversion to type `int`.

See *The Java™ Language Specification*:

5.1.3 Narrowing Primitive Conversions

longValue

```
public long longValue()
```

Returns the value of this `AtomicLong` as a `long`.

Specified by:

`longValue` in class `Number`

Returns:

the numeric value represented by this object after conversion to type long.

floatValue

```
public float floatValue()
```

Returns the value of this `AtomicLong` as a `float` after a widening primitive conversion.

Specified by:

`floatValue` in class `Number`

Returns:

the numeric value represented by this object after conversion to type float.

See *The Java™ Language Specification*:

5.1.2 Widening Primitive Conversions

doubleValue

```
public double doubleValue()
```

Returns the value of this `AtomicLong` as a `double` after a widening primitive conversion.

Specified by:

`doubleValue` in class `Number`

Returns:

the numeric value represented by this object after conversion to type double.

See *The Java™ Language Specification*:

5.1.2 Widening Primitive Conversions

Submit a bug or feature

For further API reference and developer documentation, see Java SE Documentation. That documentation contains more detailed, developer-targeted descriptions, with conceptual overviews, definitions of terms, workarounds, and working code examples.