Documentation

The Java™ Tutorials

Trail: Essential Classes
Lesson: Concurrency
Section: Synchronization

Atomic Access

In programming, an *atomic* action is one that effectively happens all at once. An atomic action cannot stop in the middle: it either happens completely, or it doesn't happen at all. No side effects of an atomic action are visible until the action is complete.

We have already seen that an increment expression, such as `c++`, does not describe an atomic action. Even very simple expressions can define complex actions that can decompose into other actions. However, there are actions you can specify that are atomic:

- Reads and writes are atomic for reference variables and for most primitive variables (all types except `long` and `double`).
- Reads and writes are atomic for *all* variables declared `volatile` (*including* `long` and `double` variables).

Atomic actions cannot be interleaved, so they can be used without fear of thread interference. However, this does not eliminate all need to synchronize atomic actions, because memory consistency errors are still possible. Using `volatile` variables reduces the risk of memory consistency errors, because any write to a `volatile` variable establishes a happens-before relationship with subsequent reads of that same variable. This means that changes to a `volatile` variable are always visible to other threads. What's more, it also means that when a thread reads a `volatile` variable, it sees not just the latest change to the `volatile`, but also the side effects of the code that led up the change.

Using simple atomic variable access is more efficient than accessing these variables through synchronized code, but requires more care by the programmer to avoid memory consistency errors. Whether the extra effort is worthwhile depends on the size and complexity of the application.

Some of the classes in the `java.util.concurrent` package provide atomic methods that do not rely on synchronization. We'll discuss them in the section on High Level Concurrency Objects.

Previous page: Intrinsic Locks and Synchronization
Next page: Liveness