

The Java™ Tutorials

Trail: Essential Java Classes

Lesson: Concurrency

Section: Synchronization

The Java Tutorials have been written for JDK 8. Examples and practices described in this page don't take advantage of improvements introduced in later releases and might use technology no longer available.

See [Java Language Changes](#) for a summary of updated language features in Java SE 9 and subsequent releases.

See [JDK Release Notes](#) for information about new features, enhancements, and removed or deprecated options for all JDK releases.

Thread Interference

Consider a simple class called `Counter`

```
class Counter {  
    private int c = 0;  
  
    public void increment() {  
        c++;  
    }  
  
    public void decrement() {  
        c--;  
    }  
  
    public int value() {  
        return c;  
    }  
}
```

`Counter` is designed so that each invocation of `increment` will add 1 to `c`, and each invocation of `decrement` will subtract 1 from `c`. However, if a `Counter` object is referenced from multiple threads, interference between threads may prevent this from happening as expected.

Interference happens when two operations, running in different threads, but acting on the same data, *interleave*. This means that the two operations consist of multiple steps, and the sequences of steps overlap.

It might not seem possible for operations on instances of `Counter` to interleave, since both operations on `c` are single, simple statements. However, even simple statements can translate to multiple steps by the virtual machine. We won't examine the specific steps the virtual machine takes — it is enough to know that the single expression `c++` can be decomposed into three steps:

1. Retrieve the current value of `c`.
2. Increment the retrieved value by 1.
3. Store the incremented value back in `c`.

The expression `c--` can be decomposed the same way, except that the second step decrements instead of increments.

Suppose Thread A invokes `increment` at about the same time Thread B invokes `decrement`. If the initial value of `c` is 0, their interleaved actions might follow this sequence:

1. Thread A: Retrieve `c`.
2. Thread B: Retrieve `c`.
3. Thread A: Increment retrieved value; result is 1.
4. Thread B: Decrement retrieved value; result is -1.
5. Thread A: Store result in `c`; `c` is now 1.
6. Thread B: Store result in `c`; `c` is now -1.

Thread A's result is lost, overwritten by Thread B. This particular interleaving is only one possibility. Under different circumstances it might be Thread B's result that gets lost, or there could be no error at all. Because they are unpredictable, thread interference bugs can be difficult to detect and fix.

Previous page: Synchronization

Next page: Memory Consistency Errors