Documentation

The Java™ Tutorials

Trail: Essential Classes
Lesson: Concurrency

Immutable Objects

An object is considered *immutable* if its state cannot change after it is constructed. Maximum reliance on immutable objects is widely accepted as a sound strategy for creating simple, reliable code.

Immutable objects are particularly useful in concurrent applications. Since they cannot change state, they cannot be corrupted by thread interference or observed in an inconsistent state.

Programmers are often reluctant to employ immutable objects, because they worry about the cost of creating a new object as opposed to updating an object in place. The impact of object creation is often overestimated, and can be offset by some of the efficiencies associated with immutable objects. These include decreased overhead due to garbage collection, and the elimination of code needed to protect mutable objects from corruption.

The following subsections take a class whose instances are mutable and derives a class with immutable instances from it. In so doing, they give general rules for this kind of conversion and demonstrate some of the advantages of immutable objects.

Problems with the examples? Try Compiling and Running the Examples: FAQs.

Complaints? Compliments? Suggestions? Give us your feedback.