Java help in creating thread-safe data structures

Synchronized versions of data-structure classes

You probably have been using Java Collections classes like ArrayList, LinkedList, ArrayDeque, HashSet, and HashMap. These are generally not "thread-safe" and shouldn't be shared among several threads.

However, class java.util.Collections has a bunch of static methods that can turn objects of these classes into classes that are thread-safe. For example, when first creating a HashSet, turn it into a synchronized HashSet using this statement:

```
Set s = Collections.synchronizedSet(new HashSet(...));
```

It's best to call synchronizedSet as shown, immediately after creating the HashSet to prevent any inadvertent unsynchronized access to the HashSet. And, of course, any further access to the HashSet should be only through s.

Don't use these tools blindly, without reading carefully about them in class java.util.Collections.

Concurrent classes

If you really want scalable, fine-tuned concurrent implementations for some data structures, look in package java.util.concurrent. For example, class ConcurrentMap<K, V> allows far more concurrency than Collections.synchronizedMap. It does this by not using synchronization for function get, instead "going in through the back door instead of the lockable front door of the house." Of course, the designers and programmers of this class have worked extensively to be sure that it is thread-safe.

Atomic classes

Finally, package java.util.concurrent.atomic contains 16 classes that implement a few data items atomically. For example, consider class AtomicInteger. It has atomic methods to add a value to the integer in an object, to decrement the integer, to increment the integer, and much much more.

There is even a class AtomicIntegerArray, in which operations on array elements are done atomically.

Summary

If you seriously need concurrent programming, in which several (or many) threads will operate concurrently on shared data, study the Java classes to see whether you can use them. Concurrency is far more difficult to understand than conventional sequential programming. The more you rely on what others have done, the better off you will be.