Lab 5: Safe Concurrency for an Abstract Data Type

Implement in Java a non-trivial classic abstract data type with at least two mutator operations, in a way that provides concurrency while still maintaining correctness. Base your solution on a Java synchronization feature such as synchronized methods/blocks or Semaphores, rather than the details of what happens to unsynchronized accesses according to the Java Memory Model.

Note that “non-trivial” may depend on your level of experience, but you should aim for something at least as interesting as one of the CMSC 106 data structures with an important representation invariant (i.e., a binary search tree, heap, or hash table with internal chaining — this last one may be tricky — but not just a stack, queue, or list). More ambitious projects, which will get a bonus of up to 5%, would include a balanced structure of some sort (such as an AVL tree, red/black tree, 2-3 tree, splay tree, or skip list). You should start by providing an “insert” mutator, a “size” accessor, and a “contains” predicate; for full credit, include a “remove” mutator as well. If you want to try an ADT not mentioned here, please get permission before starting.

If you have not previously created an ADT in Java or C++, you should make sure you understand how someone would really allocate elements in a recursive structure in Java, as is done in the JavaSortedList example from class (in contrast to the MySortedList example).

You are allowed to make use of someone else’s sequential Java class if you like, as long as (a) you cite it properly, and (b) the version you obtain (from a friend, a textbook, the internet, or whatever) does not contain any synchronization keywords/objects (or algorithms built to work in the presence of concurrency by relying on the Java Memory Model).

Include in your project:

a) A definition of a thread-safe Java class for your ADT (or more than one representation of your ADT, if you can think of two ways of organizing the synchronization and want to test them against each other). Your class should have some variables that allow the injection of inconvenient pauses, for testing purposes. I strongly recommend that you develop your representation invariant and any rules about locking before writing the complete class (you’ll need this information later anyway). I also strongly recommend that you start by writing and carefully testing the class in a sequential context, and then work on making it thread-safe. Trying to remove both algorithmic bugs and race conditions at the same time is much more painful than taking them one at a time.

b) A set of tests that drives your class(es) with different work loads (you may want one that does many insertions and lookups but only a few deletions, one with an even balance of insert/delete/lookup, etc). The tests should be in a different package from the class itself, as I did in the MySortedList and JavaSortedList examples in lecture.

c) A README.txt file or other documentation explaining your software, including any sources you consulted (sources for code you copy should also be mentioned in the program, as comments), a detailed explanation of how you know your class is race-free, how you know it is correct (i.e., what invariants are maintained and how do you know they are invariant even in the presence of concurrency), and a discussion of performance issues such as granularity of locking, whether starvation is possible, etc. For full credit, you must allow a reasonable degree of concurrency when multiple threads access your objects.
There are no “starter files” for this project; begin by using “New→Java Project” and entering a project name (all the other defaults are fine) and then right-click on src under your project name and use “New→Package” to create one or more “packages” (i.e., one for the ADT and one for tests, as in the MySortedList example from class); once you have packages, right-click on them and use “New→Class” to create each class (just enter the name and use the other defaults). Soon after creating the project, right-click on the project name (in the Java perspective) and use “Team→Share Project” to enter your project into your CVS repository (make sure you pick your cs356/Repository rather than the shared projects repository); the “Team→Share Project” acts as the first “Team→Commit”.

I plan to use some of the later classes in the semester for a series of Lab 5 “code reviews”, while you are working on your Lab 6 projects in the last few weeks of class. So make your lab something you’ll be proud to show off to your peers.

If you like to discuss your lab work with your fellow students, please choose an ADT that is different from those being undertaken by the folks you talk to a lot.