17-480/780 API Design and Implementation Homework #1: All Mixed Up

Due Thursday, September 13, 11:59 P.M.

Design and implement a general-purpose permutation generator for Java (or another language, with my permission). The API should allow you to process all the permutations of a given bunch of objects. Your generator should allow for arbitrary processing of each permutation. A simple use case is to print all the permutations of some words (say, the command line arguments). Another use case is to process the permutations of an input until you find one that satisfies some predicate. It should be possible to "nest" calls to your permutation generator to any depth, so that you can process (say) all the pairs of permutations of two input collections.

Your API should have no difficulty generating all the permutations of, say, 13 elements (not a hard number, just a rough statement of scale). Your permutation generator could be used to solve word and number puzzles such as cryptarithms and word jumbles, but don't codes up either of these things unless you're a glutton for punishment: they require a fair amount of code that is unrelated to the learning goals for this course.

Your implementation should use Heap's algorithm (or any other algorithm you like, but Heap's algorithm is good and easy to code). Feel free to get the pseudocode from Wikipedia, or anywhere else you like. Write a few unit tests to give yourself some confidence that your implementation works. This is an API design course, so don't spend too much time on the unit tests; they're mainly for your benefit, and will comprise only a small part of your grade.

Document your API well, so that a user could easily program to it on the basis of the documentation alone. Use appropriate commenting conventions (e.g., JavaDoc if you're programming in Java). In addition to your final API documentation and implementation, turn in your list of use-cases, an issues list, a one-page API draft (if you write one), and the final client code for your use cases, as discussed in today's lecture. The issues list should make clear what your key design decisions were, what options you considered for each decision, and why you chose the options you did. The use case code should be nice and clean, and should run against the implementation code. Your use cases should be short and simple; their purpose is merely to demonstrate that the API does its job. Similarly, all of your implementation artifacts (e.g., your issues list) should be short and to the point. You don't need florid prose, or even complete sentences, so long as we can understand it.

The procedure for handing in homework is TBD. We'll let you know soon.