

Re: collection framework: using the good interface

Source: <http://coding.derkeiler.com/Archive/Java/comp.lang.java.help/2005-07/msg00314.html>

- *From:* SL <spam@xxxxxxxxxxx>
 - *Date:* Tue, 19 Jul 2005 16:37:44 +0200
-

Thanks again.

If you want to retrieve by several different properties, and do it efficiently for each of them, you need several different index structures.

In fact I would like to retrieve objects using several properties always together: for instance "plane"+verb, not "plane" alone or "verb" alone. Two objects may be defined as "equal" if they share this properties.

If you don't care about efficiency for some or all of the properties, you can just iterate over a Set of all the objects asking each of them "are you a noun?".

Efficiency is an issue indeed: the situation is that I have two lexicons (two "collections" (Set or Map?) of Word objects); each Word is defined by lexicographic properties (form + POS) and hold other properties (number of occurrences in a sub-corpus for instance); for each object in one collection, I need to look at the "same" object in the other collection ("same" = same forme and same POS), compare the two frequency and compute a probability distribution function. An iteration over all the objects of the second collection for all the objects of the first collection + a check for equality without using hashCode are far

Re: collection framework: using the good interface

too long I suppose.

I wouldn't suppose, I would start with the simplest implementation and check whether it is fast enough.

If necessary, consider creating a fixed order among your word objects. It can be fairly arbitrary, such as String sort order as primary key, followed by part of speech for two words with the same spelling. The word object's class would implement Comparable, and should have equals and hashCode consistent with compareTo, but I would not make two words equal unless they are, in all respects, the same word.

This make me think about an Item about Comparator in Joshua Bloch /Effective Java/: this item begin with a WordList class as exemple (p. 44) :-)

I'm more and more convinced that equality should involve all aspects of the Word class, but I'm still uncomfortable with mixing lexicographic information (allowing to compare objects in different collections) and information about frequency, distribution, cooccurrents (where it is the difference which is interesting).

Comparing two ordered lists, with the same sort key, to find the intersection is linear time.

Have you a library than you can recommended? Perhaps the java.util.Arrays.binarySearch, but it process the search for one object at a time and does not return in one call a array of index for mapping the two collections, I'm wondering if it is well suited.

If you can't make this work reasonably efficiently with a fairly simple approach, I would consider putting the words in a database with a column for each property. Database managers are designed to do things like selects and joins on multiple attributes very efficiently.

This word lists are extracted on the fly from XML documents, with very different query (Word in such context, etc.), for comparing the word list with a reference frequency list and computing correlation of the words with the context where they were extracted.

I would not override hashCode and equals for this, because there isn't a single,

Re: collection framework: using the good interface

fixed definition of equality between two of your word-representing objects other than identity.

I was thinking that "lexicographic equality" could be an equality between the two objects, or should the definition of equality between two objects always involve all of the properties of the objects? In that case, is it a good practice to create a "customised equality method" (say: `lexicographicallyEquals()` and `lexicographicHashCode()`), and implement a `Map (LexicographicMap)` using this methods instead of `equals()` ?)

In programming, honesty is usually the best policy. If you override `equals`, you are saying "These two distinct objects have the same value". If you use `equals` to mean something else, it will probably give you trouble later.

I would deal with your properties at the word object level by having `set` and `get` methods for each property, such as spelling. The `set` method is not needed if you always set all properties in the constructor. Things like part of speech seem to me to be attributes of a word.

You are right; a better property would have been an index (as I, II, in dictionary) for distinguishing between homographs (using similarities between occurrences of the word in a vectoriel space or any other information retrieval-like method), so that each word is unique according to a lexicographic form (lemma or stem + flexion) + an index. But this uniqueness do not involve necessary equality, I should admit :-)

.