Submission Title
Data Structures

Contact Information
Richard Rasala  rasala@ccs.neu.edu
Viera K. Proulx   vkp@ccs.neu.edu
College of Computer & Information Science
Northeastern University
Boston MA 02115

Problem Statement
This submission concerns two problems with Java data structures.
1. The standard Java collections provide no way to initialize the collections from scratch in a single step. To be sure, many collections may be initialized using other collections but this is rather circular since at some stage an original collection had to be initialized by adding elements one-by-one. This submission will describe how to add constructors that use arrays to achieve quick initialization of standard structures.

2. Java provides no examples of higher-order data structures that are built internally using two or more simpler structures.

Solution Overview
Quick Initialization
The Java Power Tools provides 8 quick data structures that extend the corresponding standard Java data structures and permit quick initialization using arrays.

<table>
<thead>
<tr>
<th>Java Data Structure</th>
<th>Quick Data Structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector</td>
<td>QuickVector</td>
</tr>
<tr>
<td>ArrayList</td>
<td>QuickArrayList</td>
</tr>
<tr>
<td>LinkedList</td>
<td>QuickLinkedList</td>
</tr>
<tr>
<td>HashSet</td>
<td>QuickHashSet</td>
</tr>
<tr>
<td>TreeSet</td>
<td>QuickTreeSet</td>
</tr>
<tr>
<td>Hashtable</td>
<td>QuickHashtable</td>
</tr>
<tr>
<td>HashMap</td>
<td>QuickHashMap</td>
</tr>
<tr>
<td>TreeMap</td>
<td>QuickTreeMap</td>
</tr>
</tbody>
</table>

We will describe 2 of the 8 quick classes: QuickVector and QuickHashtable.

QuickVector extends Vector by adding one constructor and three methods addItems, setItems, and removeItems that handle an array of Object at once. All constructors of the original Vector class are provided as well. No changes are made to the internal data structure or to any inherited methods.

The new constructor in QuickVector is:

public QuickVector(Object[] items)

This constructor adds the given array items one-by-one to the vector. The capacity is set to the length of items. If items is null, then the capacity is set to 10 (Java default) and nothing is added.

The above constructor calls addItems. As an illustration, let us show the code of this method.

```java
public void addItems(Object[] items) {  
    if (items == null) return;
    int length = items.length;
    for (int i = 0; i < length; i++)
        add(items[i]);
}
```

There is nothing spectacular about this code but it eliminates the need for adding items one-by-one to the vector. We feel that any encapsulation that lets students focus on the ideas and not on repetitive code is helpful.

A simple example of the use of the QuickVector constructor is:

```java
QuickVector primes = new QuickVector(  
    new Object[] {  
        new Integer(2),  
        new Integer(3),  
        new Integer(5)  
    });
```

In this example, the objects added are rather trivial but they do illustrate the technique.

QuickHashtable extends Hashtable by adding one constructor and three methods putPairs, setPairs, and removeKeys that handle an array of Object at once. In addition, symmetric methods are added that will hash both a key to its value and a value to its key. All constructors of the original Hashtable class are provided as well. The only inherited methods that are overridden are those that will throw an exception on null arguments. In this class, such methods will handle null in a sensible fashion.

The new constructor in QuickHashtable is:

```java
public QuickHashtable(Object[][] pairs)
```

This constructor adds the given 2-dimensional array of key-value pairs one-by-one to the hash table and ignores key-value pairs in which either item is null. The capacity is set either to 10 or to 1.5 times the length of the array pairs whichever is larger. The load factor is set to the default of 0.75.

A simple example of the use of the QuickHashtable constructor is:

```java
QuickHashtable colors = new QuickHashtable(  
    new Object[][] {  
        { "black", color.black },  
        { "white", color.white },  
        { "red", color.red },  
        { "green", color.green },  
        { "blue", color.blue },  
        { "brown", new Color( 165, 42, 42 ) }  
    });
```

This example can obviously be extended to initialize a large hash table of color name keys and color object values. The tabular view provided by the array is clearly much easier to create and read than a series of put statements.

If one wanted to symmetrically hash color names to color objects and vice versa, then one would do the following:

```java
QuickHashtable colors = new QuickHashtable();

colors.putPairsSymmetric(  
    new Object[][] {  
        { "black", color.black },  
        { "white", color.white },  
        { "red", color.red },  
        { "green", color.green },  
        { "blue", color.blue },  
        { "brown", new Color( 165, 42, 42 ) }  
    });
```

Then "red" would hash to Color.red and Color.red would hash to "red".

Obviously, with the proposed introduction of generics in Java 1.5, the quick structures described here would have to be suitably modified. In particular, the symmetric methods would only make sense in an untyped hash table.
Structural Building Blocks

As building blocks for more complex structures, JPT defines two fairly simple classes.

The class Pair constructs an immutable pair object from two non-null objects that are referred to as A and B. The access methods are getA and getB. The methods equals and hashCode are designed so that two Pair objects whose contents are equal will be considered equal and will return the same hash code.

The class StringObjectPair constructs an immutable pair object from a non-null String and a non-null Object. The access methods are getString and getObject. The methods equals and hashCode are designed so that two StringObjectPair objects whose contents are equal will be considered equal and will return the same hash code.

Higher-Order Data Structures

Currently, JPT defines two higher-order data structures StringSet and StringObjectMap. The class StringSet encapsulates a structure that maintains a set of Strings both in the original order of entry and in sorted order. The class StringObjectMap encapsulates a structure that maintains a collection of String and Object pairs in such a way that the original order of pair entry is preserved, it is easy to obtain the strings in sorted order, and it is also efficient to map a string to an object and an object to a string.

Let us discuss StringObjectMap in a bit more detail. This class uses three internal collections as member data:

- A QuickTreeMap that maps the strings to the objects and also permits the strings to be retrieved in sorted order.
- A QuickHashMap that maps the objects to the strings.
- A QuickVector that maintains the list of string-object pairs in the original order of entry into the structure.

The class is optimized for rapid access and ease of use. Given a string, one can get its object, and given an object, one can get its string. Given an index, one can get the string, object, or string-object pair that was inserted at that stage in the insert process. One can also get arrays of all inserted strings, or all inserted objects, or all inserted string-object pairs in order of insertion. Finally, one can get an array of all inserted strings in sorted order.

Obviously, the cost of rapid access is the redundancy of the data in the three internal collections and the small extra time taken in the insertion process. However, this cost is balanced by the ease of programming with this structure. Students who need to work with pairs of strings and objects can do whatever they need to do very easily with a StringObjectMap. Given the fact that, in this era, programming costs remain high while memory is quite cheap, this structure is actually an excellent pedagogical model for how to design higher-order data structures. Since JPT is 100% open source, faculty and students can examine and critique the code and then attempt to do better in structures that they work with.

Experience with the Solution

The data structures described in the submission form a foundation for the GUI tools in JPT but may also be used pedagogically in a manner quite independent of GUIs or graphics. We believe that students should be introduced to the quick extensions of the Java collections and begin to learn about higher-order data structures.

API Documentation & Related Materials

The main JPT site to access documentation, code, and the jpt.jar:

http://www.ccs.neu.edu/jpt/