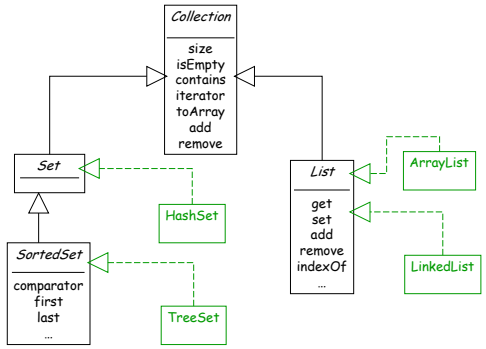
Finish JCF; Data Structure Examples

Lecture 20
CS211 – Fall 2006

Partial Summary of JCF



java.util.Map<K,V> (an interface)

- Map does not extend Collection
- A Map contains key/value pairs instead of individual elements
- Methods
  - public V put (K key, V value);
    - Associates value with key in the map; returns the old value associated with key or null if the key did not previously appear in the map
  - public V get (Object key);
    - Returns the object to which this key is mapped or null if there is no such key
  - public boolean containsKey (Object key);
    - True iff Map contains a pair using the given key
  - public boolean containsValue (Object value);
    - True iff there is at least one pair with this value
  - public V remove (Object key);
    - Removes any mapping for the key; returns old value associated with key if there was one (null otherwise)

More Map Methods

- Other methods
  - public int size ( );
    - Return the number of key/value pairs in the Map
  - public boolean isEmpty ( );
    - True iff Map holds no pairs
- Bulk methods
  - public void putAll (Map<? extends K, ? extends V> otherMap);
    - Puts all the mappings from otherMap into this map
  - public void clear ( );
    - Removes all mappings
- Sets/Collections derived from a Map
  - public Set<K> keySet ( );
    - Returns a Set whose elements are the keys of this map
  - public Collection<V> values ( );
    - Returns a Collection whose elements are all the values of this map

java.util.SortedMap<K,V> (an interface)

- Extends the Map contract: requires that keys are sorted
- The iterators for keySet( ), values( ), and entrySet( ) all return items in order of the keys

- Methods (in addition to those inherited from Map):
  - public Comparator<? super K> comparator ( );
    - Returns the comparator used to compare keys for this map; null is returned if the natural order is being used
  - public K firstKey ( );
    - Returns the first (lowest value) key in this map
  - public K lastKey ( );
    - Returns the last (highest value) key in this map
  - ...

Set and SortedSet Implementations

java.util.HashMap (a class; implements Map)
- Constructors
  - public HashMap ( );
  - public HashMap (Map<? extends K, ? extends V> map);
  - public HashMap (int initialCapacity);
  - public HashMap (int initialCapacity, float loadFactor);

java.util.TreeMap (a class; implements SortedMap)
- Constructors
  - public TreeMap ( );
  - public TreeMap (Map<? extends K, ? extends V> map);
  - public TreeMap (Comparator<? super K> comp);
  - ...

Efficiency & Some Comments

- Both TreeMap and HashMap are meant to be accessed via keys
  - get, put, containsKey, remove are all fast
    - O(1) expected time for HashMap
    - O(log n) worst-case time for TreeMap
  - containsValue is slow
    - O(n) for both HashMap and TreeMap
- HashSet and TreeSet are actually implemented by building a HashMap and a TreeMap, respectively
- Given a Map that maps student ID number to student name, print out a list of students sorted by ID number and another list sorted by name (assume no duplicate names)

The java.util.Arrays Utility Class

- Provides useful static methods for dealing with arrays
  - sort
    - Mostly uses QuickSort
    - Uses MergeSort for Object[ ] (it's stable)
  - binarySearch
  - equals
  - fill
- These methods are overloaded to work with
  - arrays of each primitive type
  - arrays of Objects
- Methods sort and binarySearch can use the natural order or there is a version of each that can use a Comparator
- There is also a method for viewing an array as a List:

```
static List asList (Object[ ] a);
```

  - Note that the resulting List is backed by the array (i.e., changes in the array are reflected in the List and vice versa)

Unmodifiable Collections

- Dangerous version:
  - `public final String suits[ ] = { "Clubs", "Diamonds", "Hearts", "Spades" };`
- The final modifier means that suits always refers to the same array, but the array's elements can be changed
  - `suits[0] = "Leisure";`
- Safe version (it would be better really to use an Enum):

```
private final String theSuits[ ] = { "Clubs", "Diamonds", "Hearts", "Spades" };
public final List suits = Collections.unmodifiableList(Arrays.asList(theSuits));
```

- The Collections class provides unmodifiable wrappers; any methods that would modify the collection throw an UnsupportedOperationException
  - unmodifiableCollection, unmodifiableSet, unmodifiableSortedSet, unmodifiableList
  - unmodifiableMap, unmodifiableSortedMap

java.util.Collections Utilities

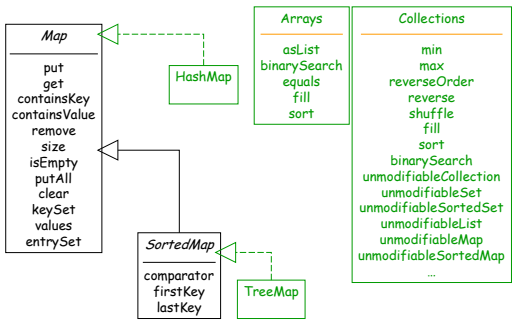
```
public static Object min (Collection c);
public static Object min (Collection c, Comparator comp);
public static Object max (Collection c);
public static Object max (Collection c, Comparator comp);

public static Comparator reverseOrder ( );      // Reverse of natural order

public static void reverse (List list);         // Reverse the list
public static void shuffle (List list);         // Randomly shuffle the list
public static void fill (List list, Object x);  // List is filled with x's

public static void sort (List list);            // Sort using natural order
public static void sort (List list, Comparator comp);
public static void binarySearch (List list, Object key);
public static void binarySearch (List list, Object key, Comparator comp);
...
```

Additional Portions of the JCF



Additional JCF Interfaces & Classes

- java.util.Queue<E>
  - An interface
  - Has peek( ) op
  - Implemented by
    - LinkedList
    - PriorityQueue
- Legacy classes
  - java.util.Hashtable
  - java.util.Vector
  - java.util.Stack
- java.util.PriorityQueue<E>
  - A class
  - Heap-based PQ using table-doubling
  - Ordering is based on *natural order* or on a Comparator
    - To use a Comparator, it must be specified in the constructor
  - Implements Queue

How do we use this stuff?

- Now that we know all about the standard data structures, what can we do with this information?
- Sample problem: Given n items in an array (unsorted), find the k smallest items
  - Possible strategies
    - Sorting
    - Balanced BST
    - Heap (of size n)
    - Heap (of size k)
    - Other…
- Sample problem: Build a data structure so that
  - insert, getMax, and getMin are efficient
  - insert and reportMedian are efficient
  - insert and getMode are efficient