Set and Map

IS 313, 1.16.2002
Tuesday’s code

- Comparator
Skeletons

- Useful coding patterns
- Should understand
  - how and why they work
  - how to use
- possible quiz material
Homework

- Late policy
  - 10% per day
  - only if help requested 48 hrs in advance
- Academic honesty
  - No code sharing
  - BUT discussion and assistance is OK
- Rule of thumb
  - Am I getting advice? Or a homework solution?
- Note
  - Plagiarized code is (surprisingly) easy to detect
Resources

- Homework discussion forum
- Email to instructor
  - if you need to get advice on a large chunk of code
Homework #1

- Frequent Flyer Program
- Input
  - A file of member accounts
  - A file of flights
- Output
  - A updated file of member accounts
    - note: must be sorted by member name
Homework 1 Solution Structure

- Member class
  - implement comparable
  - or, separate CompareByName comparator
- MemberMap to hold Members
  - encapsulates a SortedMap
  - access by id
- Flight
  - stores flight information including
  - encapsulates List of member #s
Homework #1 Solution Structure

- FlightList
  - encapsulates a List
  - all of the flight objects in the file

- MemberUpdater
  - “main” class
  - should be VERY simple
Part 2 Solution Structure

- CapacityInfo class
- Encapsulated SortedMap
  - associating dates with capacity data
Very simple

public class MemberUpdater
{
    public void main (String [] args)
    {
        MemberList members = new MemberList ();
        members.load ("accounts.dat");
        FlightList flights = new FlightList ();
        flights.load ("flights.dat");
        members.updateForFlights (flights);
        members.save ("updated.dat");
    }
}
General Advice

- Start early
  - working on homework = studying for quiz
- Implement Member first
  - Use instantiation skeleton (#1 and #2) to create from file
- Don’t cut corners
  - all 6 classes are needed
StringTokenizer

- Useful for handling text files
- Breaks strings into tokens
  - usually defined by whitespace
  - “words”
StringTokenizer cont’d

- Works like an iterator
- Test for completion
  - hasMoreTokens
- Get token and move
  - nextToken
- Skeleton #2
  - shows how to use a StringTokenizer to create objects from a text file
Example

String text = "Whatever you want, you'll get it.";

StringTokenizer st = new StringTokenizer(text);

while( st.hasMoreTokens() ) {
    String word = st.nextToken();
    ...process token...
}
Example

- StringTokenizer
Fundamental Abstractions

- Collection
  - Any grouping of items
- Set
  - Unordered, non-redundant
- List
  - Ordered
- Map
  - Object->Object mappings
Collection Interfaces
Set

- Has only the Collection methods
- But the Collection has no duplicates
public Object [] removeDups (Object [] ar)
{
    List original = Arrays.asList (ar);
    Set noDups = new HashSet(original);
    return noDups.toArray();
}

- How does this work?
  - Arrays utility class
- Will the returned array be in the same order as the original?
SortedSet

- Set starts to look like a list
  - Elements are iterated over in order
- Like sorting operations
  - Objects must be Comparable
  - Or a Comparator can be supplied
SortedSet API

```java
public interface SortedSet extends Set {
    // Range-view
    SortedSet subSet(Object fromElement, Object toElement);
    SortedSet headSet(Object toElement);
    SortedSet tailSet(Object fromElement);
    // Endpoints
    Object first();
    Object last();
    // Comparator access
    Comparator comparator();
}
```
Example

- SortedSet
- Comparable
Map

- Collection of pairs
  - Key, value

- We can associate values
  - name, phone number
  - member id, member object

- Both key and value must be objects
  - use “wrapper” classes for primitive types
  - `map.put (new Integer(5), “stored at 5”);`
Map API

Object put(Object key, Object value);
Object get(Object key);
Object remove(Object key);
boolean containsKey(Object key);
boolean containsValue(Object value);
int size();
boolean isEmpty();
// Bulk Operations
void putAll(Map t);
void clear();
// Collection Views
public Set keySet();
public Collection values();
public Set entrySet();
Collection Views

- keySet()
  - A set of the keys

- values()
  - A collection of the values

- entrySet()
  - A set of all the pairs
  - Entry objects
    - getKey, getValue
Views

- Views are still tied to the original Map
  - If I remove a key from the keySet
  - It is removed from the Map
- Example

```java
m1.keySet().removeAll(m2.keySet());
```
  - Removes any entries in m1 that have the same key as an entry in m2.
SortedMap

- Exactly analogous to SortedSet
- Sort is in key order
Implementations

- Set
  - HashSet
  - TreeSet

- Map
  - HashMap
  - TreeMap
  - (WeakHashMap)
  - (LinkedHashMap)
HashSet

- Collection implemented with a hash table
- O(1) access
  - Provide you size it appropriately
  - An odd # about twice as big as the size of the largest set
Hash Table
To store a set in a hash table

- Use
  - hash key = the item in the set
  - stored value = anything

- If there is an entry in the table
  - the item is in the set
TreeSet

- Only useful for sorted sets
- $O(\log n)$ for access
  - but it is always sorted
To store a set in a binary tree

- **Use**
  - tree key = item in the set
  - value = anything

- **If the tree key is in the tree**
  - the item is in the set

- **To iterate in order**
  - start at the left of the tree
  - “in-order” traversal
Trade-offs

- Sorting
- Size
  - for large collections
HashMap

- O(1) access
- Same size considerations as HashSet
To store a map in a hash table

- **Use**
  - hash key = map key
  - value = value

- **To look up a value**
  - find the key/value pair
  - return value
TreeMap

- Only useful for SortedMap
- $O(\log n)$ access
To store a map in a binary tree

- **Use**
  - tree key = map key
  - tree value = map value

- **To look up a value**
  - traverse tree to appropriate key
  - return stored value

- **To iterate**
  - “in-order” traversal
Decorators

- **Unmodifiable**
  - Pass in a normal collection (set, map, whatever)
  - The resulting collection is the same but can’t be modified

- **Synchronized**
  - Pass in a normal collection
  - The resulting collection is synchronized
    - Can’t be shared between threads
    - Useful in multi-threaded programs, otherwise inefficient
Example

List myList = new LinkedList();
... manipulate list ...
display (myList);

- user interface component could modify list
- instead:
  display (Collections.unmodifiableList (myList));
Example

- Map
  - HashMap
  - TreeMap
- values() collection
- Inheritance
- Encapsulation / Composition
- Method overloading