Collections Framework: Part 2

Lecture 18
Collection Implementations

- Java SDK provides several implementations of Collection subinterfaces
  - List
    - ArrayList, LinkedList
  - Queue (and Deque)
    - PriorityQueue, LinkedList
  - Set (and SortedSet)
    - HashSet, TreeSet, LinkedHashSet, EnumSet
- These differ in concrete implementation
  - Differences in algorithmic complexity
  - Different refinements of interface semantics
Iterable Collection Hierarchy

- **Collection**
  - **List**
    - **ArrayList**
    - **PriorityQueue**
    - **LinkedList**
    - **HashSet**
    - **TreeSet**
    - **SortedSet**
  - **Queue**
    - **PriorityQueue**
    - **LinkedList**
    - **ArrayList**
  - **Set**
    - **HashSet**
    - **TreeSet**
  - **Deque**

- **Iterable**
  - **Iterator**
  - **ListIterator**

**Relationships**:
- Green arrow: extends
- Red arrow: implements
List Implementations

- **ArrayList**: a resizable-array
  - Adding or removing elements at the end, or getting an element at a specific position is fast – O(1)
  - Adding or removing elements from the middle is more expensive – O(n-i)
  - Can be efficiently scanned (using indices) without creating an Iterator object
  - Good for: lists that are scanned frequently, lists where most additions/removals are at the ends

- **LinkedList**: a doubly-linked list
  - Getting an element at position i is more expensive – O(i)
  - But once you are there, addition/removal is fast – O(1)
  - Good for: lists where most of additions/removals are not at the ends
Customizing Collections

- To support creation of new collection classes, SDK provides several abstract classes
  - Skeleton implementation of base functionality
  - Can not be instantiated directly
  - Can be extended, providing appropriate implementation details
    - Example: add method throws exception unless overridden
Iterable Collection Hierarchy

extends

AbstractList
AbsCollection
AbstractQueue
List
AbstractSet
SortedSet
HashSet
PriorityQueue
LinkedList
AbsSeqList
ArrayList
AbsCollection

implements

Collection
List
Queue
Set
Deque
Iterable
Iterator
ListIterator
SortedSet
HashSet
TreeSet
ArrayList
Maps

- While Collections contain individual elements, Maps contain key-value pairs
  - A map can not contain duplicate keys
  - It maps each key to at most one value
  - Recall Resolve’s Bag vs Partial_Map

- Provided as a generic interface
  ```java
  interface Map<K, V>
  ```
  - K: type of key, V: type of value
  - Example
    ```java
    Map<String, PhoneNumber> phoneBook
    ```

- SortedMap further guarantees that keys are in ascending order
Map Hierarchy

extends

implements

Map

SortedMap
Map Interface

- Three views of contents
  - Set of keys
  - Collection of values
  - Set of key-value pairs (ie mappings)

- Main methods for obtaining these views
  ```java
  public Set<K> keySet()
  public Collection<V> values()
  public Set<Map.Entry<K,V>> entrySet()
  ```

- These views are backed by the actual Map
  - Removing element from one of these views removes the key-value pair from the Map
  - Adding an element to one of these views is not allowed
  - Recall: While iterating, make such modifications only through the iterator

- Arbitrary iteration order
  - Independent order for keys / values in same Map
  - Subinterface SortedMap provides this guarantee
Map Interface Cont’d

- More methods for working with Map
  - Modifying contents
    ```java
    public V get(Object key)
    public V put(K key, V value)
    public V remove(Object key)
    public void clear()
    ```
  - Statistics and searching
    ```java
    public int size()
    public boolean isEmpty()
    public boolean containsKey(Object key)
    public boolean containsValue(Object value)
    ```
Map Hierarchy

extends

implements

AbstractMap

EnumMap

HashMap

TreeMap

SortedMap

Map
Utility Class: java.util.Collections

- Static methods for many common tasks
  - Ordering and permuting
    - public void sort(List list)
    - public void shuffle(List list)
    - public void reverse(List list)
    - public void rotate(List list, int distance)
    - public void swap(List list, int i, int j)
  - Modifying contents
    - public <T> void fill(List<T> list, T obj)
    - public <T> void copy(List<T> src, List<T> dst)
  - Statistics and searching
    - public int frequency(Collection c, Object o)
    - public boolean disjoint(Collection c1, Collection c2)
    - public <T> T min(Collection<T> c)
    - public <T> T max(Collection<T> c)
Utility Class: java.util.Arrays

- Not part of the “Collections Framework”
- Static methods for common tasks:
  - Ordering
    ```java
    public void sort(int[] a)
    public void sort(int[] a, int i, int j)
    ```
  - Modifying contents
    ```java
    public void fill(int[] a, int val)
    public void fill(int[] a, int i, int j, int v)
    ```
  - Statistics and searching
    ```java
    public int binarySearch(int[] a, int key)
    ```
  - Core methods
    ```java
    public boolean equals(int[] a1, int[] a2)
    public int hashCode(int[] a)
    public String toString(int[] a)
    ```
- All are overloaded (for primitives and Object)
Good Practice: Avoid Legacy Types

- java.util has been around since 1.0
  - “Collections Framework” since 1.2
- For backwards compatibility, it still contains some classes that have been superseded
  - The use of these older classes is deprecated
  - The only reason for using them is to interface with legacy code
- The “legacy collections” are:
  - Enumeration – prefer Iterator interface
  - Stack – prefer Deque (a subinterface of Queue)
  - Dictionary – prefer Map interface
  - Hashtable – prefer HashMap class*
  - Vector – prefer ArrayList class*
  
  *Aside: Vector and Hashtable are still used today, but only for multithreaded code
Good Practice: Know the Libraries

Example: Print (contents of) an array

```java
int[] a = ... 
System.out.println(a);  // gibberish 
System.out.println(Arrays.toString(a));
```

Example: Find identical entries in two phone books

```java
Map tmp = new HashMap(h1);
tmp.entrySet().retainAll(h2.entrySet());
Set result = tmp.keySet();
```
Supplemental Reading

- Sun “Collections Framework” trail
- For Collections utility class, see “Algorithms” section of collections trail
Summary

- **Collection Implementations**
  - `ArrayList`, `LinkedList`, `PriorityQueue`, `HashSet`

- **Maps**
  - Key/value pairs, with unique keys
  - Interfaces: `Map`, `SortedMap`
  - Classes: `HashMap`, `EnumMap`, `TreeMap`

- **Utility Classes**
  - Collections, Arrays