

Collections Framework

Lecture 6

Overview

- A framework of many classes and interfaces
- Part of the java.util package
 - See API Javadoc
 - See "Collections Framework" trail
- This framework provides *container* classes
 - Hold other objects
 - Defined as generic classes (recall Box<?>)
 - Allow efficient access to contents in useful ways
- Two basic kinds of containers:
 - Collection (List, Queue, Set)
 - Map

Map & Collection Hierarchies

→ extends



Root Interface: Collection

- Generic
 - Collection<String> bag;
- Methods working with an individual collection

```
public int size()
public boolean isEmpty()
public boolean contains(Object target)
public boolean add(E element)
public boolean remove(Object target)
public Object[] toArray()
public <T> T[] toArray(T[] dest)
```

 - General description (eg are duplicates allowed?)
 - Returns a new array containing references to all the elements of the collection
 - What is returned depends on whether the elements in the collection fit in dest
 - If the type of dest is not compatible with the types of all elements in the collection, an exception is thrown

Root Interface: Collection cont'd

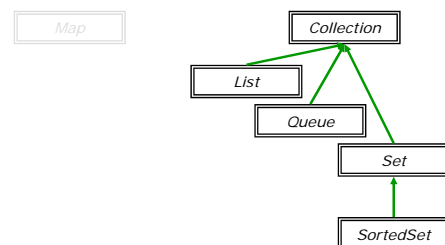
- Bulk methods using contents of another collection

```
public boolean containsAll(Collection coll)
public boolean addAll(Collection coll)
public boolean removeAll(Collection coll)
public boolean retainAll(Collection coll)
public void clear()
```

 - Returns true if any addition succeeds
 - Returns true if any removal succeeds
 - Removes from the collection all elements that are not elements of coll
 - Remove all elements from this collection
- No **direct** implementations of Collection in SDK
 - Useful for passing collections around and manipulating them where maximum generality is desired
 - Recall: "code to the interface"
 - Subinterfaces (List, Queue, Set) do have direct implementations

Collection Hierarchy

→ extends



Subinterfaces

- List
 - Ordered sequence of elements
 - Indexed from 0 to list.size()-1
 - Client controls location of newly inserted element
 - Allows duplicate elements
 - New methods:
 - sublist (return a subsequence from index to index)
- Queue
 - Ordered sequence of elements (LIFO, FIFO, priority)
 - Removals (and peeking) occur at the head
 - New methods:
 - offer (queue might be full)
 - peek (look at head without removing)
- Set
 - No duplicate elements (add is idempotent)
 - No guarantee of ordering
 - Subinterface SortedSet provides such a guarantee

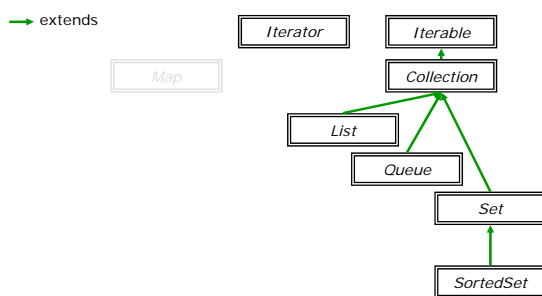
Iteration

- To examine the contents of a collection, an *iterator* is used
 - Allows us to loop through contents, examining each element in turn
 - No order guarantee (for Collection)
 - Does not expose internal structure of collection
 - Static type:


```
interface Iterator<E>
```
- To obtain an iterator use collection method:


```
public Iterator<E> iterator()
```
- This method is promised in the Iterable interface
 - Actually part of java.lang
 - Collection extends Iterable

Iterable Collection Hierarchy



Iterator Interface

- Three methods in Iterator interface


```
public boolean hasNext()
```

 - Returns true iff the iteration has more elements

```
public E next()
```

 - Returns the next element in the iteration
 - An exception will be thrown if there is no next element

```
public void remove()
```

 - Note use of generics in return type
 - Remove from the collection the element last returned by the iteration
 - Can be called only *once per call of next*, otherwise an exception is thrown

Canonical Example

```

public void removeLongStrings
(Collection<? extends String> c, int maxlen) {
    Iterator<? extends String> it = c.iterator();
    while ( it.hasNext() ) {
        String str = it.next();
        if (str.length() > maxlen)
            it.remove()
    }
}
  
```

Special For-Loop Syntax ("for-each")

- Syntactic shortcut for looping through something Iterable


```
for (Type loop-var : set-expression)
    statement
```

 - Can not be used to remove elements from collection
- Example


```
Collection<Student> roster = . . .
for (Student std : roster) {
    System.out.println(std.getID());
}
```
- Can be used with arrays as well

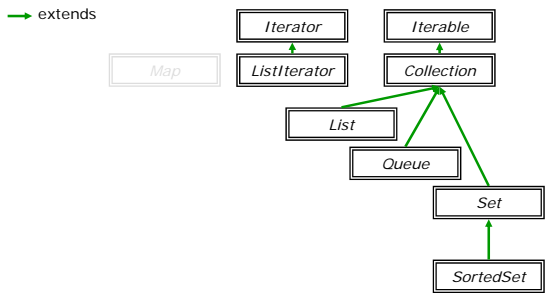

```
int[] values = . . .
double sum = 0.0;
for (int val : values) {
    sum += val;
}
```

ListIterator

- ListIterator interface extends Iterator interface
 - Provides ordering guarantee for iteration
 - Adds methods for moving forwards or *backwards*
- Methods


```
public boolean hasNext() / boolean hasPrevious()
public E next() / E previous()
public int nextIndex() / int previousIndex()
    ■ When at the end of the list, nextIndex() returns list.size()
    ■ When at the beginning of the list, previousIndex() returns -1
public void remove()
    ■ Remove the element last returned by next() or previous()
public void add(E elem)
    ■ Inserts elem into list in front of the element that would be
    returned by next(), or at the end if no next element exists
public void set(E elem)
    ■ Replace the element last returned by next() or previous()
    with elem
```

Iterable Collection Hierarchy



cf Resolve's Sequence

- Exercise for the reader:
 - Compare Java's ListIterator with Resolve's Sequence component
 - How does insertion point differ?
 - How does element removal differ?

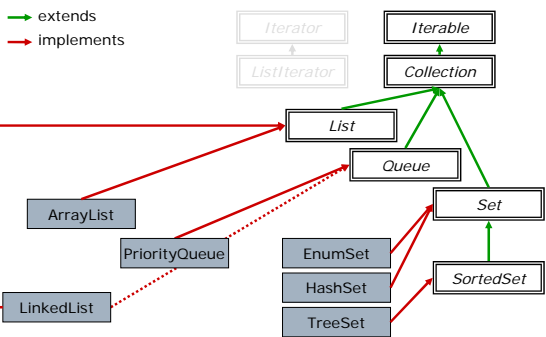
Modifying a Collection

- While iterating through a collection, the *only* safe way to modify the collection is *through the iterator itself*
 - Use Iterator's remove() method, not Collection's remove(Object) method
- Many iterators in Java SDK try to detect a modification of the underlying collection and complain
 - An exception is thrown
 - Known as "fail-fast" behavior
 - Not guaranteed! Do not rely on this safety net!

Collection Implementations

- Java SDK provides several implementations of Collection subinterfaces
 - List
 - ArrayList, LinkedList
 - Queue
 - 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



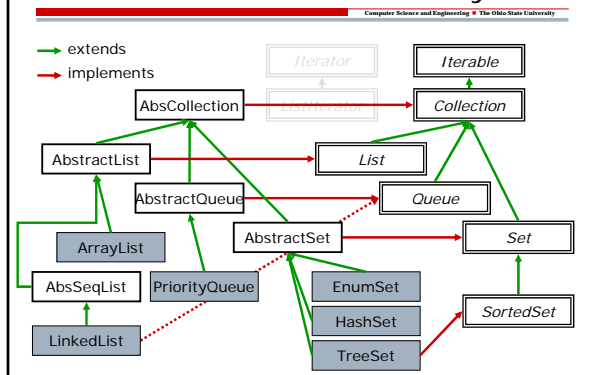
List Implementations

- 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 – $O(n-1)$
 - 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



Maps

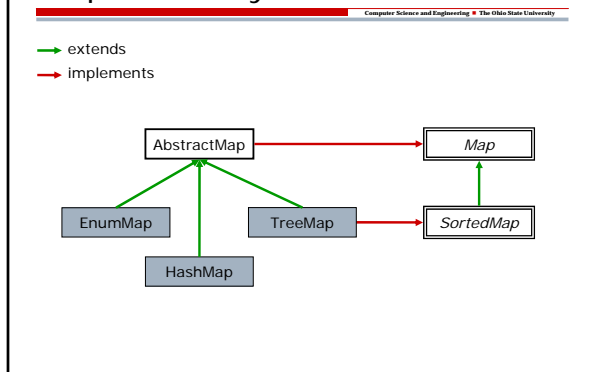
- While Collections contain individual elements, Maps contain key-value pairs
 - A map can not contain duplicate keys
 - It maps each key to at most one value
 - Recall Bag vs Partial_Map in Resolve
- Provided as a generic interface


```
interface Map<K, V>
```

 - K: type of key, V: type of value
 - Example


```
Map<String, PhoneNumber> phoneBook
```
- SortedMap further guarantees keys are in ascending order

Map Hierarchy



Map Interface

- Three views of contents
 - Set of keys
 - Collection of values
 - Set of key-value mappings
- Main methods for obtaining these views


```
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: make such modifications through iterator
- Arbitrary iteration order
 - Independent order for keys / values in same Map
 - Subinterface SortedMap provides these guarantees

Map Interface Cont'd

Computer Science and Engineering @ The Ohio State University

- More methods for working with Map

```
public int size()
public boolean isEmpty()
public boolean containsKey(Object key)
public boolean containsValue(Object
value)
public V get(Object key)
public V put(K key, V value)
public V remove(Object key)
public void clear()
```

Best Practice: Avoid Legacy Types

Computer Science and Engineering @ The Ohio State University

- java.util has been around since 1.0
- 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 – analogous to Iterator
 - Vector – analogous to ArrayList
 - Stack – a subclass of Vector
 - Dictionary – analogous to Map interface
 - Hashtable – analogous to HashMap
 - Properties – a subclass of Hashtable, but still widely used, no good alternative yet