Java Collections API


Copyright Kip Irvine, 2003. All rights reserved. Only students enrolled in COP 4338 at Florida International University may copy or print the contents of this slide show. Some materials used here are from William T. Kraynek, used by permission.
Abstract Data Structures

- What's a List?
  - Collection
  - Iterator
  - List
  - ListIterator
- Set
- SortedSet
- Map
  - Map.Entry
- SortedMap
- Comparator
- Comparable
Collections and Iterators

- Collection
  - List
  - Set
  - SortedSet

- Iterator
  - ListIterator
What's a List?

- elements are ordered and connected
- duplicates OK
- insert/remove in the middle
- sequential access only
  - forward or backward (traverse)

![Diagram of linked list with nodes Jones, Baker, Ruiz]
What's a Set?

- Elements have no order
- No duplicates
- Operations
  - verify membership (is it there?)
  - insert and remove

Baker
Jones
Ruiz
What's a Collection?

● A general type of container
  – set, list, or map

● Defined by an interface in Java
  – common operations for sub-interfaces
Collection Interface

- Not directly implemented by any class
- Duplicate items permitted
- Basic operations:
  - int size( )
  - boolean isEmpty( )
  - boolean contains( Object o )
  - boolean remove( Object o )
  - boolean add( Object o )

continued . . .
Collection Interface (cont)

- Bulk operations:
  - boolean containsAll( Collection c )
  - boolean addAll( Collection c )
  - boolean removeAll( Collection c )
  - boolean retainAll( Collection c )
  - void clear( )

- Array operations:
  - Object[ ] toArray( )
  - Object[ ] toArray( Object[ ] a )
What's an Iterator?

- An object that moves over a collection
  - like a subscript, but it's not an integer
- Lets us "point" to each element
- Might be unidirectional or bidirectional
- Different types:
  - ListIterator
Iterator Interface

- Methods:
  - `boolean hasNext()`
  - `Object next()`
  - `void remove()`
List Interface

- Most common operations:
  - `void add(int index, Object element)`
    - `// add an item to the list`
  - `Object get(int index)`
    - `// get a reference to an item`
  - `int indexOf(Object obj)`
    - `// find an item`
  - `ListIterator listIterator()`
    - `// get an iterator to the list`
  - `Object set(int index, Object element)`
    - `// set an item's value`
List Interface

- Less common operations:
  - `void addAll( int index, Collection c )`
  - `// add some other collection to the list`
  - `int lastIndexOf( Object obj )`
  - `// find last occurrence of an item`
  - `ListIterator listIterator( int index )`
  - `// get iterator at a certain position`
  - `List subList( int fromIndex, int toIndex )`
  - `// get part of a list`
Who Implements the List Interface?

- AbstractList
- LinkedList
- ArrayList  
  - (unsynchronized – efficient)
- Vector  
  - (synchronized – costly)
What's a ListIterator?

- Interface that extends the Iterator interface
  - traverse a list in either direction and modify the list

- Methods:
  - `void add(Object newVal)`
  - `boolean hasNext()`
  - `boolean hasPrevious()`
  - `Object next()`
  - `int nextIndex()`
  - `Object previous()`
  - `int previousIndex()`
  - `void remove()`
  - `void set(Object newVal)`
List Examples
Sets

There are 10 kinds of people in this world. Those who understand binary, and those who don't.
Set Interface

- Extends Collection
  - but it doesn't allow duplicate elements

- Implementing classes
  - AbstractSet, HashSet

- Operations
  - boolean add( Object obj )
  - boolean addAll( Collection c )
SortedSet Interface

- Extends Set
  - guarantees that iterator will traverse in ascending order
  - uses a Comparator to order the elements
  - elements implement Comparable interface
  - Implementing class: TreeSet

- Operations:
  - Comparator comparator()
  - Object first()
  - Object last()
  - SortedSet headSet(Object toElement)
  - SortedSet subSet(Object fromElt, Object toElt)
  - SortedSet tailSet(Object fromElement)
SortedSet Operations

- get first element
  - Object first( )

- get last element
  - Object last( )

- get elements from beginning, middle, or end
  - SortedSet headSet( Object toElement )
  - SortedSet subSet( Object fromElt, Object toElt )
  - SortedSet tailSet( Object fromElement )
Map-Related Interfaces

Map

Map.Entry

SortedMap
Map Interface

- Maps keys to values
  - no duplicate keys, each key holds one value
  - each entry is a Map.Entry (inner class)

- Three collection views
  - set of keys
  - collection of values
  - set of key-value mappings (called *entries*)

- Order of elements determined by iterator
Map Operations

- Basic Operations
  - Object get( Object key )
  - Object put( Object key, Object value ) *
  - Object remove( Object key ) *
  - int size( )

- Set Operations
  - Set keySet( )
  - Set entrySet( )
  - Collection values( )

- Other Operations:
  - void clear( ) *
  - boolean containsKey( Object key )
  - boolean containsValue( Object value )
  - boolean isEmpty( )
  - void putAll( Map m ) *

* = optional operation
Map.Entry

- Key-value pair
  - Collection of these returned by Map.entrySet()
- Can only obtain reference to a map entry from the Map's iterator
- Methods:
  - Object getKey()
  - Object getValue()
  - Object setValue( Object value )
SortedMap Interface

- Extends Map
  - guaranteed to be in ascending key order
  - keys implement Comparable interface
  - uses Comparator

- Operations:
  - Comparator comparator( )
  - Object firstKey( )
  - Object lastKey( )
  - SortedMap headMap( Object toKey )
  - SortedMap subMap( Object fromKey, Object toKey )
  - SortedMap tailMap( Object fromKey )
Comparable Interface

- Imposes a natural ordering on the objects of any class that implements it
- Method:
  - int compareTo( Object other )
- Many classes do not implement Comparable because the objects could be compared many different ways
  - Ex: Employee (sort by ID, last name, years of service, etc.)
  - Instead, they use a Comparator (see later)
Data Structure Implementations

- LinkedList (List)
- HashSet (Set)
- TreeSet (SortedSet)
- HashMap (Map)
LinkedList Class

- Implements the List interface
- Not synchronized
- Permits all elements, including null & duplicates
- Doubly-linked (permits forward & backward traversal)
- Indexing into the list is expensive
- Traversal is cheap
- Iterator and ListIterator are fail-fast
  - throws a ConcurrentModificationException
- List elements must override equals()
  - when calling contains(), remove(), indexOf(), lastIndexOf()

See: LinkedListEx.java
Fail-Fast Iterator?

- If a list or set is modified at any time after the iterator is created, in any way except through the iterator's own remove() method, the Iterator throws a ConcurrentModificationException.
  - The iterator fails quickly and cleanly, rather than risking arbitrary, non-deterministic behavior at an undetermined time in the future.
HashSet Class

- Implements the **Set** Interface, not synchronized, elements not ordered, iterator is fail-fast
- **Constant time performance** for the basic operations
  - add, remove, contains, size
- **Operations:**
  - add( ), clear( ), clone( ), contains( ), isEmpty( ), iterator( ), remove( ), and size( )
- **Set elements must** override equals( )
  - when calling contains( ) and remove( )
- **Set elements must** implement hashCode( )
  - to prevent adding duplicate objects

See: HashSetEx.java
TreeSet Class

- Implements **SortedSet**
  - Set interface, backed by a TreeMap interface
- Guarantees that the sorted set will be in ascending element order:
  - sorted according to the natural order of the elements, using **Comparable** interface, or . . .
  - sorted by the comparator provided at set creation time
  - (depends on which constructor is used)
- Elements must implement **Comparable**
  - otherwise, CastClastException is thrown
  - must override the compareTo( ) method
TreeSet Class (cont)

- not synchronized
- iterators are fail-fast
- Ordering of elements must be consistent with equals( )
  - if and only if \((e1 \text{.compareTo((Object)e2)} == 0)\) has the same boolean value as \(e1 \text{.equals((Object)e2)}\) for every \(e1\) and \(e2\) of class C.
  - therefore, element class must implement equals( )

See: TreeSetEx.java
Comparator Interface

- Called the **strategy** design pattern
- A Comparator imposes a total ordering on a collection
  - can be passed to a sort method
  - precise control over sort order
- Ordering of elements is **consistent with equals**,
  - if and only if `e1.compareTo((Object)e2)==0` has the same boolean value as `e1.equals((Object)e2)`
  - (generally, this is desirable)
- Methods:
  - `int compare( Object o1, Object o2 )`
  - `boolean equals( Object obj )`

See: ComparatorEx.java
HashMap Class

- implements `Map` interface
- order of entries not significant
- permits null key and null values
- not synchronized
- Important operations:
  - `put`, `get`, `keySet`, `entrySet`, `values`
  - `containsKey`, `containsValue`
  - `remove`

See: HashMapEx.java
The End