The Java 2 Collections

Chuck Allison
http://www.freshsources.com

- java.util.Arrays
- Comparators and java.util.Comparable
- Collections
- java.util.Collections
- Maps
- An Application

Foreword

- Data Structures are OLD
- Useful generic data structures are RECENT
- Useful Collections in Java are QUITE NEW
- If you know STL, Java 2 doesn’t have:
  - function objects (except Comparator)
  - customizable algorithms (except via Comparators)
  - queues, stacks, deques, priority queues
    - but you can use LinkedList for the first 3

- Algorithms for array processing:
  - binarySearch
  - equals
  - fill
  - sort
    - unstable quicksort for primitives
    - stable mergesort for Objects
  - asList (to use List methods discussed later)
- Overloaded for Object and all Primitives

- Requires a sorted list
  - Ascending only!
- Returns 0-based index
  - Where found (if non-negative), or
  - Where belongs (if negative)
    - -1 => 0; -2 => 1, -3 => 2; …
    - Formula: `rval = -index - 1`
    - Compute: `-(rval + 1)`

```
import java.util.*;

class ArraysTest
{
    static void printArray(int[] a)
    {
        System.out.print("[");
        for (int i = 0; i < a.length; ++i)
        {
            System.out.print(a[i]);
            if (i < a.length-1)
                System.out.print(",");
        }
        System.out.println("]");
    }
```

```
static void search(int[] a, int x)
{
    // Must be in ascending order:
    int where = Arrays.binarySearch(a, x);
    if (where < 0)
    {
        where = -(where + 1);
        if (where == a.length)
            System.out.println("Append " + n +
                               " to end of list");
        else
            System.out.println("Insert " + n +
                               " before " +
                               a[where]);
    }
    else
        System.out.println("Found " + n +
                           " in position " +
                           where);
}
```

```
public static void main(String[] args)
{
    // Build Array:
    int[] array = {88, 17, -10, 34, 27, 0, -2};
    System.out.print("Before sorting: ");
    printArray(array);

    // Sort:
    Arrays.sort(array);
    System.out.print("After sorting: ");
    printArray(array);



/* Output:
Before sorting: [88,17,-10,34,27,0,-2]
After sorting: [-10,-2,0,17,27,34,88]
*/
```

```
// Search:
search(array, -10);
search(array, -1);
search(array, 0);
search(array, 1);
search(array, 34);
search(array, 100);
```

```
/* Output:
Found -10 in position 0
Insert -1 before 0
Found 0 in position 2
Insert 1 before 17
Found 34 in position 5
Append 100 to end of list
*/
```

```java
        // Equals:
        System.out.println("array == array? " +
                           Arrays.equals(array,
                                         array));
        int[] ones = new int[array.length];
        Arrays.fill(ones, 1);
        System.out.println("array == ones? " +
                           Arrays.equals(array, ones));
    }
}
```

```
/* Output:
array == array? true
array == ones? false
*/
```

Arrays.sort "Feature"

- Can only sort Primitives in ascending order
- Sorting Object arrays allows a Comparator
  - override compare( ) method
  - strcmp-like semantics
  - should be consistent with equals
    - return 0

The Comparator Interface

- int compare(Object o1, Object o2)
- boolean equals(Object o)
  - Don’t have to supply if not used
    - Because inherited from Object

```
import java.util.*;

class Descending implements Comparator
{
    // Works for Integers only:
    public int compare(Object o1, Object o2)
    {
        Integer c1 = (Integer) o1;
        Integer c2 = (Integer) o2;
        return -c1.compareTo(c2); // - 0 +
    }
}
```

- Interface with one method
  - int compareTo(Object o)
  - strcmp-like semantics
- Implemented by String, Date, and all Primitive Wrappers

A Generic Comparator

```
import java.util.*;

class Descending implements Comparator
{
    // Works for any Comparable object:
    public int compare(Object o1, Object o2)
    {
        Comparable c1 = (Comparable) o1;
        Comparable c2 = (Comparable) o2;
        return -c1.compareTo(c2);
    }
}
```

```java
import java.util.*;

class ArraysTest2
{
    static Descending desc = new Descending();

    static void printArray(Object[] a)
    {
        // (same as before)
    }

    static void search(Object[] a, Object x)
    {
        int where = Arrays.binarySearch(a, x, desc);
        // (rest as before)
    }
```

```
public static void main(String[] args)
{
    // Build Array:
    Integer[] array = {new Integer(88),
        new Integer(17), new Integer(-10),
        new Integer(34), new Integer(27),
        new Integer(0), new Integer(-2)};
    System.out.println("Before sorting:");
    printArray(array);

    // Sort via Comparator:
    Arrays.sort(array, desc);
    System.out.println("After sorting:");
    printArray(array);



/* Output:
Before sorting:
{88,17,-10,34,27,0,-2}
After sorting:
{88,34,27,17,0,-2,-10}
*/
```

```
        // Search:
        search(array, new Integer(-10));
        search(array, new Integer(-1));
        search(array, new Integer(0));
        search(array, new Integer(1));
        search(array, new Integer(34));
        search(array, new Integer(10));
    }
}



/* Output:
Found -10 in position 6
Insert -1 before -2
Found 0 in position 4
Insert 1 before 0
Found 34 in position 1
Insert 10 before 0
*/
```

```
// Use the following instead of Descending:
static Comparator desc =
    Collections.reverseOrder();
```

User-defined Types

```
public class Person implements Comparable, Cloneable
{
    private String name;
    private int year;
    private int month;
    private int day;

    public Person(String name, int year, int month,
                  int day)
    {
        this.name = new String(name);
        this.year = year;
        this.month = month;
        this.day = day;
    }
    public String getName()
    {
        return name;
    }
```

```java
// Override Object methods:
public Object clone()
{
    try
    {
        Person p = (Person) super.clone();
        p.name = new String(name);
        return p;
    }
    catch (CloneNotSupportedException x)
    {
        throw new InternalError(x.toString());
    }
}
public boolean equals(Object o)
{
    if (o instanceof Person)
    {
        Person p = (Person) o;
        return name.equals(p.name) &&
            year == p.year &&
            month == p.month &&
            day == p.day;
    }
    else
        return false;
}
```

```
    public int hashCode()
    {
        int hval = name.hashCode() + year;
        hval = (hval << 4) + month;
        hval = (hval << 4) + day;
        return hval;
    }
    public String toString()
    {
        return '{' + name + ',' + month +
               '/' + day  + '/' + year  + '}';
    }

    // Implement Comparable:
    public int compareTo(Object o)
    {
        Person p = (Person)o;
        int result = name.compareTo(p.name);
        if (result == 0)
        {
            result = (year - p.year);
            result = result*16 + (month - p.month);
            result = result*16 + (day - p.day);
        }
        return result;
    }
}
```

```java
import java.util.*;

class ArraysTest3
{
    static void printArray(Object[] a)
    {
        System.out.print("[");
        for (int i = 0; i < a.length; ++i)
        {
            System.out.print(a[i]);
            if (i < a.length-1)
                System.out.println(",");
            else
                System.out.println("]");
        }
    }
    static void search(Object[] a, Object x)
    {
        int where = Arrays.binarySearch(a, x);
        // (rest as before)
    }
```

```
public static void main(String[] args)
{
    // Build Array:
    Person[] array = new Person[3];
    array[0] = new Person("Horatio", 1835,12,6);
    array[1] = new Person("Charles",1897,3,11);
    array[2] = new Person("Albert",1901,1,20);
    System.out.println("Before sorting:");
    printArray(array);
```

```
/* Output:
Before sorting:
[{Horatio,12/6/1835},
{Charles,3/11/1897},
{Albert,1/20/1901}]
*/
```

```
// Sort:
Arrays.sort(array);
System.out.println("After sorting:");
printArray(array);
```

```
/* Output:
After sorting:
{{Albert,1/20/1901},
{Charles,3/11/1897},
{Horatio,12/6/1835}}
*/
```

```
        // Search:
        search(array, array[1].clone());
        search(array,
               new Person("Gregory", 1582, 10, 15));
    }
}
```

```
/* Output:
Found {Charles,3/11/1897} in position 1
Insert {Gregory,10/15/1582} before
{Horatio,12/6/1835}
*/
```

Sorting by Key

```
import java.util.*;

class ByName implements Comparator
{
    public int compare(Object o1, Object o2)
    {
        Person p1 = (Person) o1;
        Person p2 = (Person) o2;
        return p1.getName().compareTo(p2.getName());
    }
}

class ArraysTest4
{
    static ByName comp = new ByName();
```

```
    static void search(Object[] a, Object x)
    {
        // (as before)
    }

    public static void main(String[] args)
    {
        // Build Array:
        Person[] array = new Person[3];
        array[0] = new Person("Horatio", 1835,12,6);
        array[1] = new Person("Charles",1897,3,11);
        array[2] = new Person("Albert",1901,1,20);

        Arrays.sort(array, comp);

        search(array, array[1]);
        search(array, new Person("Fred",0,0,0));
        search(array, new Person("Joe",0,0,0));
    }
}

/* Output:
Found {Charles,3/11/1897} in position 1
Insert {Fred,0/0/0} before {Horatio,12/6/1835}
Append {Joe,0/0/0} to end of list
*/
```

- Implement the Collection interface
- Must provide the following constructors:
  - C( ) {…}
  - C(Collection c) {…}
- Modifiers are optional
  - can throw UnsupportedOperationException
  - standard implementations don’t throw

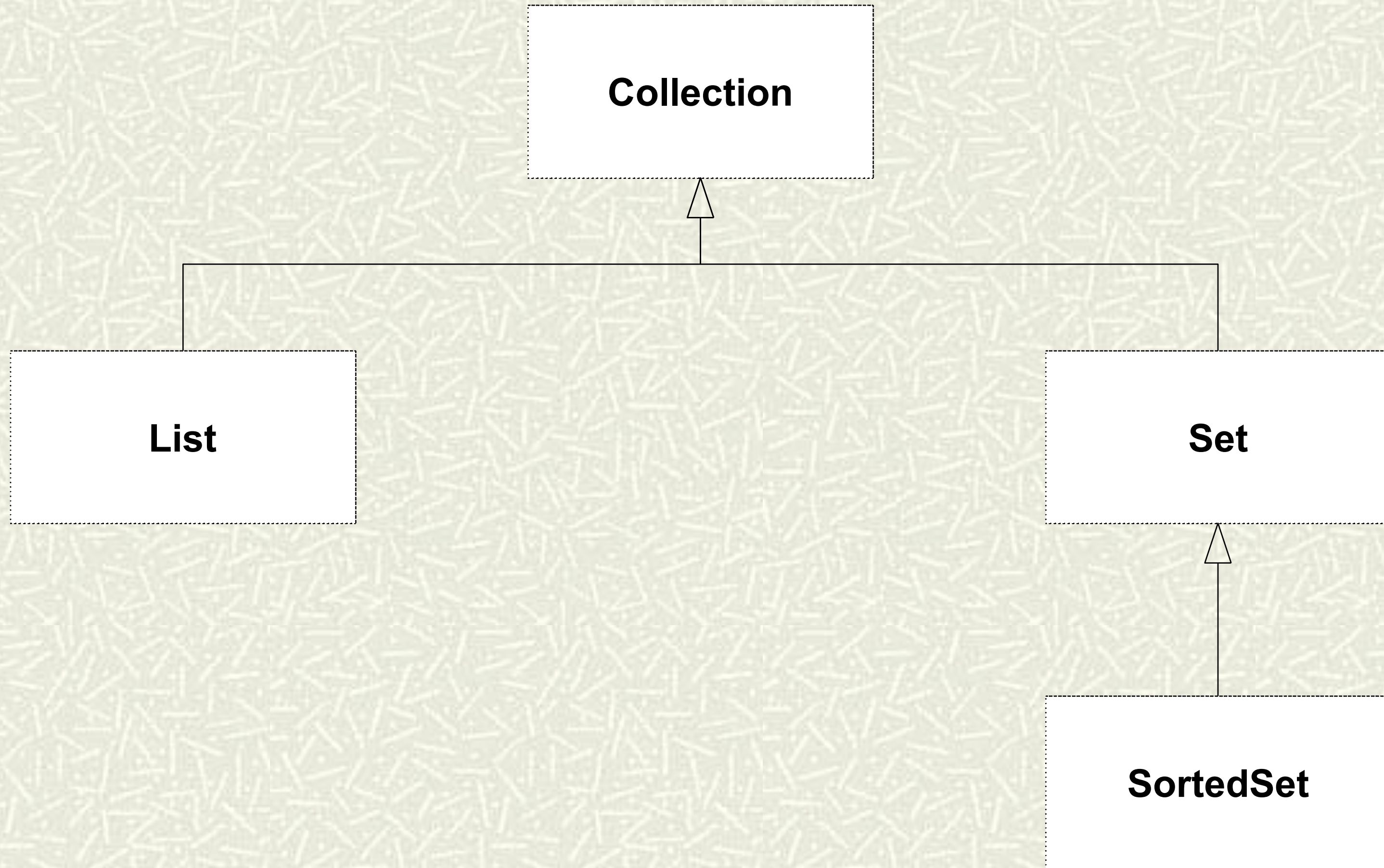
```
interface Collection
{
    boolean         add(Object o);
    boolean         addAll(Collection c);
    void            clear();
    boolean         contains(Object o);
    boolean         containsAll(Collection c);
    boolean         equals(Object o);
    int             hashCode();
    boolean         isEmpty();
    Iterator        iterator();
    boolean         remove(Object o);
    boolean         removeAll(Collection c);
    boolean         retainAll(Collection c);
    int             size();
    Object[]        toArray();
    Object[]        toArray(Object[] a);
}
```

Iterators

- Traverse a Collection
- Like Enumeration, only better
  - can remove objects
  - ListIterator can modify elements
- “Fail Fast” Operation
  - exception thrown if collection is modified externally (i.e., not via the iterator itself)

```
interface Iterator
{
    boolean     hasNext();
    Object      next();
    void        remove();
}
```

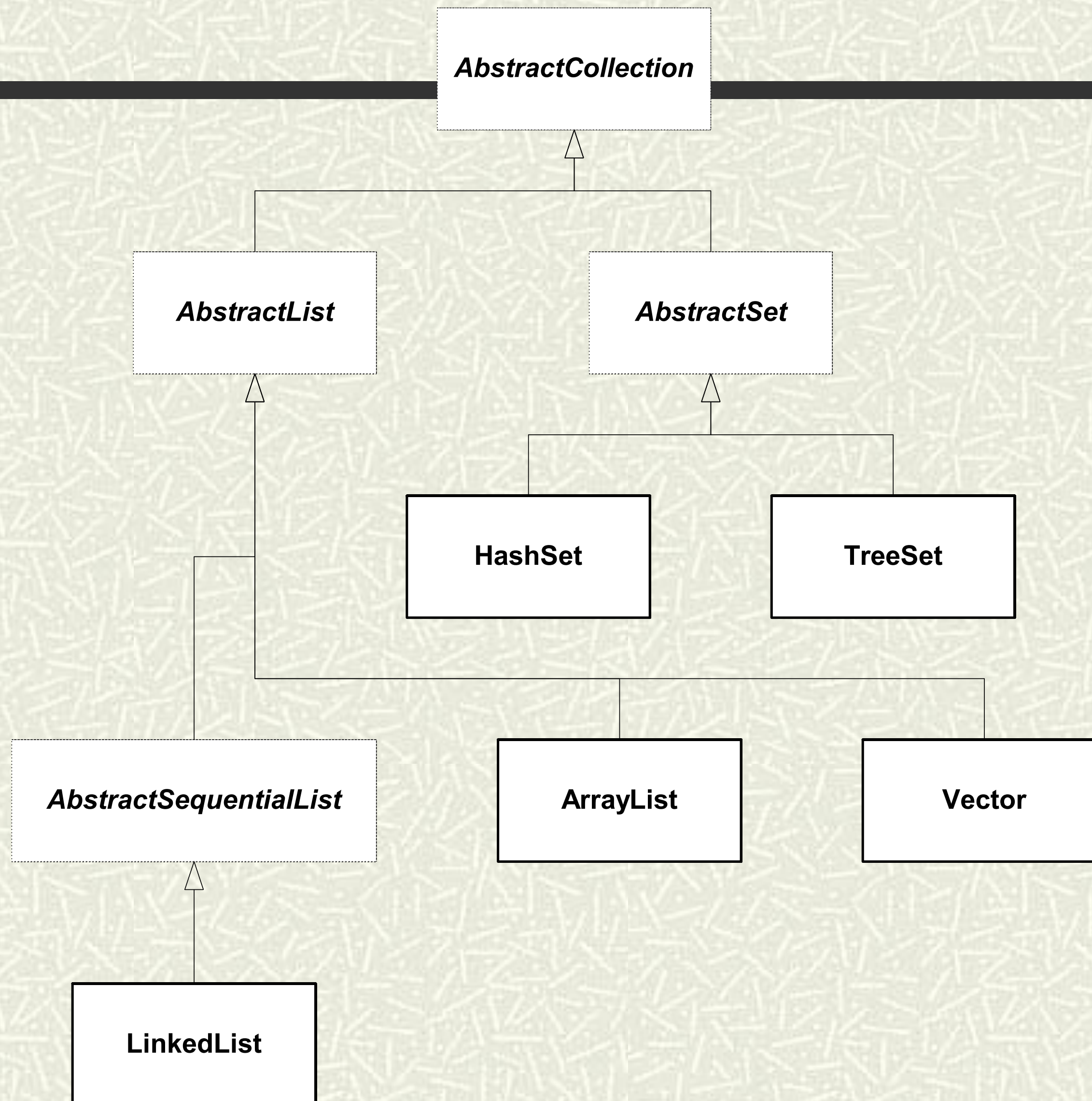
Collection Interface Hierarchy



```
// List defines a Sequence
interface List extends Collection
{
   // Everything from Collection plus:
   void             add(int index, Object element)
   Object           get(int index)
   int              indexOf(Object o)
   int              lastIndexOf(Object o)
   ListIterator listIterator()
   ListIterator listIterator(int index)
   Object           remove(int index)
   Object           set(int index, Object element)
   List             subList(int fromIndex, int toIndex)
}
```

```
interface ListIterator extends Iterator
{
    // Everything from Iterator plus:
    void            add(Object o);
    boolean         hasPrevious();
    int             nextIndex();
    Object          previous();
    int             previousIndex();
    void            set(Object o);
}
```

Implementations



```java
import java.util.*;

class ListTest
{
    public static void test(List x)
    {
        // Do a linear search:
        Object p = x.get(2);
        if (x.contains(p))
            System.out.println("Found " + p);

        // Do another:
        int index = x.indexOf(p);
        if (index != -1)
            System.out.println("Found " + p +
                               " in position "
                               + index);
```

```
    // Do a binary search:
    Collections.sort(x);
    System.out.println(x);
    index = Collections.binarySearch(x, p);
    if (index >= 0)
        System.out.println("Found " +
                                  p + " in position "
                                  + index);

    // Misc:
    System.out.println("max == " +
                              Collections.max(x));
    System.out.println("min == " +
                              Collections.min(x));
    Comparator desc = Collections.reverseOrder();
    System.out.println("max (desc) == " +
                              Collections.max(x, desc));
    System.out.println("min (desc) == " +
                              Collections.min(x, desc));
    Collections.shuffle(x);
    iterate(x);
}
```

```
public static void iterate(Collection c)
{
    System.out.println("Iterating...");
    Iterator i = c.iterator();
    while (i.hasNext())
        System.out.println(i.next());
}
```

```
    public static void main(String[] args)
    {
        // Build ArrayList:
        ArrayList array = new ArrayList();
        array.add(new Person("Horatio", 1835,12,6));
        array.add(new Person("Charles",1897,3,11));
        array.add(new Person("Albert",1901,1,20));
        System.out.println(array);

/* Output:
[{Horatio,12/6/1835}, {Charles,3/11/1897},
{Albert,1/20/1901}]
*/
```

```
        test(array);




/* Output:
Found {Albert,1/20/1901}
Found {Albert,1/20/1901} in position 2
[{Albert,1/20/1901}, {Charles,3/11/1897},
{Horatio,12/6/1835}]
Found {Albert,1/20/1901} in position 0
max == {Horatio,12/6/1835}
min == {Albert,1/20/1901}
max (desc) == {Albert,1/20/1901}
min (desc) == {Horatio,12/6/1835}
Iterating...
{Horatio,12/6/1835}
{Charles,3/11/1897}
{Albert,1/20/1901}
*/
```

```java
        array.add(new Person("James", 1976, 8, 13));
        System.out.println(array);
        System.out.println();
```

```java
/* Output:
[{Horatio,12/6/1835}, {Charles,3/11/1897},
{Albert,1/20/1901}, {James,8/13/1976}]
*/
```

```
// Build LinkedList:
LinkedList list = new LinkedList(array);
List view = list.subList(1,3);
view.add(new Person("Gregory", 1582, 10, 15));
Collections.reverse(view);
System.out.println(list);
```

```
/* Output:
[{Horatio,12/6/1835}, {Gregory,10/15/1582},
{Albert,1/20/1901}, {Charles,3/11/1897},
{James,8/13/1976}]
*/
```

```
    test(list);
    }
}

/* Output:
Found {Albert,1/20/1901}
Found {Albert,1/20/1901} in position 2
[{Albert,1/20/1901}, {Charles,3/11/1897},
{Gregory,10/15/1582}, {Horatio,12/6/1835},
{James,8/13/1976}]
Found {Albert,1/20/1901} in position 0
max == {James,8/13/1976}
min == {Albert,1/20/1901}
max (desc) == {Albert,1/20/1901}
min (desc) == {James,8/13/1976}
Iterating...
{Gregory,10/15/1582}
{Horatio,12/6/1835}
{James,8/13/1976}
{Albert,1/20/1901}
{Charles,3/11/1897}
*/
```

```
class Collections
{
     static List EMPTY_LIST;
     static Set EMPTY_SET;

     static int binarySearch(List list, Object key);
     static int binarySearch(List list, Object key,
                             Comparator c);
     static void copy(List dest, List src);
     static Enumeration enumeration(Collection c);
     static void fill(List list, Object o);
     static Object max(Collection coll);
     static Object max(Collection coll,
                       Comparator comp);
     static Object min(Collection coll);
     static Object min(Collection coll,
                       Comparator comp);
     static List nCopies(int n, Object o);
     static void reverse(List l);
     static Comparator reverseOrder();
     static void shuffle(List list);
     static void shuffle(List list, Random rnd);
     static Set singleton(Object o);
```

```
    static void sort(List list);
    static void sort(List list, Comparator c);
    static Collection
        synchronizedCollection(Collection c);
    static List synchronizedList(List list);
    static Map synchronizedMap(Map m);
    static Set synchronizedSet(Set s);
    static SortedMap
        synchronizedSortedMap(SortedMap m);
    static SortedSet
        synchronizedSortedSet(SortedSet s);
    static Collection
        unmodifiableCollection(Collection c);
    static List unmodifiableList(List list);
    static Map unmodifiableMap(Map m);
    static Set unmodifiableSet(Set s);
    static SortedMap
        unmodifiableSortedMap(SortedMap m);
    static SortedSet
        unmodifiableSortedSet(SortedSet s);
}
```

- Random-access lists sort in log(n) time
- Arbitrary sequences sort in n*log(n) time
- Instances of AbstractSequentialList in O(n)
  - e.g., LinkedList

```
// Modify.java:
//      Uses an iterator to
//      modify a collection.

import java.util.*;

class Modify
{
    public static void main(String[] args)
    {
        // Build Array:
        ArrayList a = new ArrayList();
        a.add(new Integer(1));
        a.add(new Integer(2));
        a.add(new Integer(3));
        System.out.println(a);



/* Output:
[1, 2, 3]
*/
```

```
        // Modify via iterator:
        ListIterator p =
            (ListIterator) a.listIterator();
        while (p.hasNext())
        {
            Integer i = (Integer) p.next();
            p.set(new Integer(i.intValue()
                                      + 1));
        }
        System.out.println(a);
    }
}

/* Output:
[2, 3, 4]
*/
```

Queues, Stacks and Deques

- Not there!
- Use LinkedList
  - addFirst, addLast
  - removeFirst, removeLast
  - getFirst, getLast

```java
import java.util.*;

class Queue
{
    private LinkedList data;

    public Queue()
    {
        data = new LinkedList();
    }
    public void put(Object o)
    {
        data.addFirst(o);
    }
    public Object get()
        throws NoSuchElementException
    {
        return data.removeLast(); // change for Stack
    }
    public int size()
    {
        return data.size();
    }
}
```

```
class QueueTest
{
    public static void main(String[] args)
    {
        Queue q = new Queue();
        q.put(new Integer(1));
        q.put(new Integer(2));
        q.put(new Integer(3));

        while (q.size() > 0)
            System.out.println((Integer) q.get());
    }
}

/* Output:
1
2
3
*/
```

Sets

- Mathematical Set Abstraction
- No Duplicates
- Two Interfaces
  - Set (Identical to Collection)
  - SortedSet
- Two Implementations
  - HashSet, TreeSet

```
interface Set extends Collection
{
    //(same as Collection - specifies no duplicates)
}

interface SortedSet extends Set
{
   Comparator comparator();
   Object first();
   SortedSet headSet(Object toElement);
   Object last();
   SortedSet subSet(Object fromElement,
                    Object toElement);
   SortedSet tailSet(Object fromElement);
}
```

```
import java.util.*;

class SetTest
{
    public static void test(Set x)
    {
        // Do a search:
        Person p = new Person("Albert",1901,1,20);
        if (x.contains(p))
            System.out.println("Found " + p);

        // Misc:
        System.out.println("max == " +
                           Collections.max(x));
        System.out.println("min == " +
                           Collections.min(x));
        iterate(x);
    }
```

```
public static void iterate(Collection c)
{
    System.out.println("Iterating...");
    Iterator i = c.iterator();
    while (i.hasNext())
        System.out.println(i.next());
}
```

```java
public static void main(String[] args)
{
    // Build HashSet:
    HashSet h = new HashSet();
    h.add(new Person("Horatio", 1835,12,6));
    h.add(new Person("Charles",1897,3,11));
    h.add(new Person("Albert",1901,1,20));
    // The following is ignored:
    h.add(new Person("Albert",1901,1,20));
    System.out.println(h);
```

```
/* Output:
[{Charles,3/11/1897}, {Albert,1/20/1901},
{Horatio,12/6/1835}]
*/
```

```
        test(h);



/* Output:
Found {Albert,1/20/1901}
max == {Horatio,12/6/1835}
min == {Albert,1/20/1901}
Iterating...
{Charles,3/11/1897}
{Albert,1/20/1901}
{Horatio,12/6/1835}
*/
```

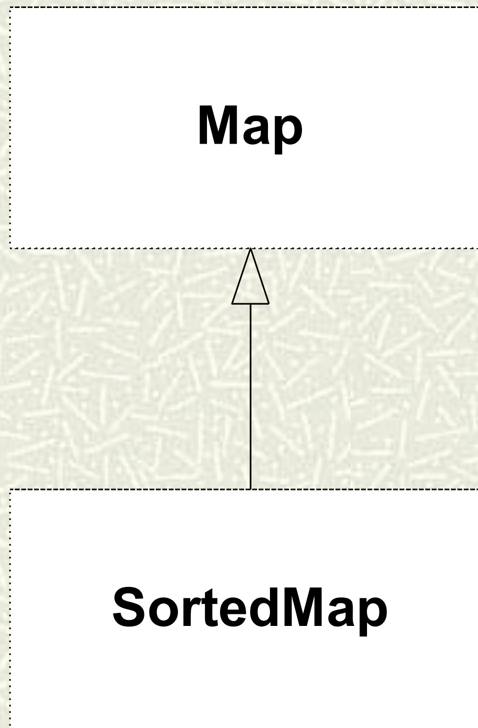
```
        // Build TreeSet:
        TreeSet t =
            new TreeSet(Collections.reverseOrder());
        t.addAll(h);
        System.out.println(t);
        test(t);
```

```
/* Output:
[{Horatio,12/6/1835}, {Charles,3/11/1897},
{Albert,1/20/1901}]
Found {Albert,1/20/1901}
max == {Horatio,12/6/1835}
min == {Albert,1/20/1901}
Iterating...
{Horatio,12/6/1835}
{Charles,3/11/1897}
{Albert,1/20/1901}
*/
```

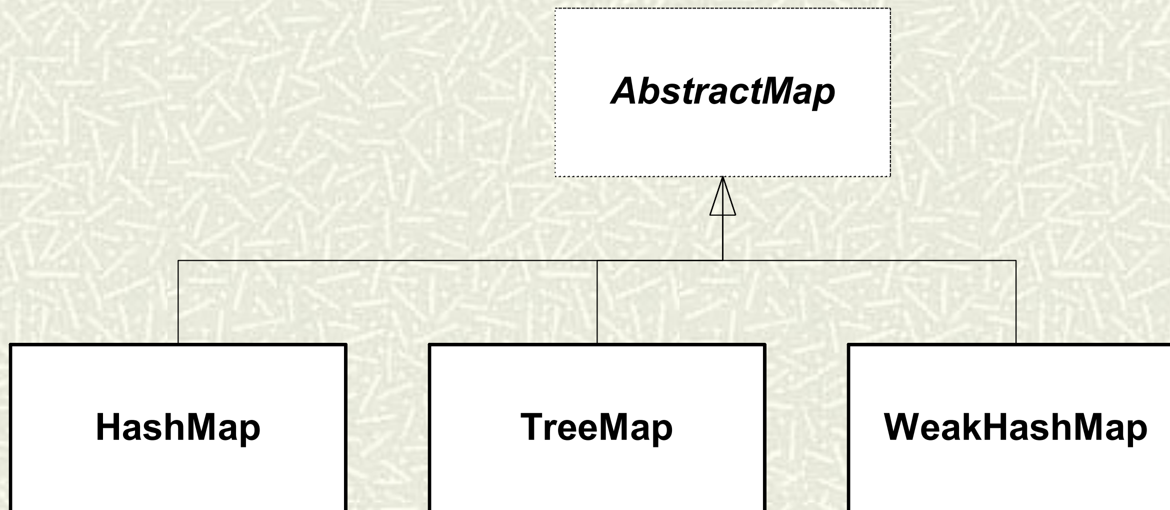
```
        // Extra TreeSet stuff:
        boolean ctest =
            t.comparator()
             .equals(Collections.reverseOrder());
        System.out.println("Comparator test: " +
                           ctest);
        System.out.println("first = " + t.first());
        System.out.println("last = " + t.last());
        Person p = new Person("Charles",1897,3,11);
        System.out.println("headSet: " + t.headSet(p));
        System.out.println("tailSet: " + t.tailSet(p));
    }
}


/* Output:
Comparator test: true
first = {Horatio,12/6/1835}
last = {Albert,1/20/1901}
headSet: [{Horatio,12/6/1835}]
tailSet: [{Charles,3/11/1897}, {Albert,1/20/1901}]
*/
```

- Store key/value pairs of Objects
- Duplicate Keys not allowed
- Not part of the Collections Hierarchy
- Returns keys as a Set view
- Returns values as a Collection

The Map Interface Hierarchy



Map Implementations



```
import java.util.*;

class MapTest
{
    public static void test(Map m)
    {
        if (m.containsKey("Alabama"))
            System.out.println("Alabama is a key");
        if (m.containsValue("Montgomery"))
            System.out.println("Montgomery is a value");
        System.out.println("m[Georgia] = " +
                           m.get("Georgia"));
        System.out.println("m[Michigan] = " +
                           m.get("Michigan"));
        System.out.println("Keys: " + m.keySet());
        System.out.println("Values: " + m.values());
        iterate(m);
    }
```

```java
public static void iterate(Map m)
{
    System.out.println("Iterating...");
    Set s = m.entrySet();
    Iterator i = s.iterator();
    while (i.hasNext())
    {
        Map.Entry e = (Map.Entry)(i.next());
        System.out.println(e);
    }
}
```

```
public static void main(String[] args)
{
     HashMap h = new HashMap();
     h.put("Alabama", "Montgomery");
     h.put("Tennesee", "Nashville");
     h.put("Georgia", "Savannah");
     // The following value replaces "Savannah":
     h.put("Georgia", "Atlanta");
     System.out.println(h);
```

```
/* Output:
{Alabama=Montgomery, Georgia=Atlanta,
Tennesee=Nashville}
*/
```

```
        test(h);

/* Output:
Alabama is a key
Montgomery is a value
m[Georgia] = Atlanta
m[Michigan] = null
Keys: [Alabama, Georgia, Tennesee]
Values: [Montgomery, Atlanta, Nashville]
Iterating...
Alabama=Montgomery
Georgia=Atlanta
Tennesee=Nashville
*/
```

```java
        TreeMap t =
            new TreeMap(Collections.reverseOrder());
        t.putAll(h);
        System.out.println(t);
        test(t);
```

```
/* Output:
{Tennesee=Nashville, Georgia=Atlanta,
Alabama=Montgomery}
Alabama is a key
Montgomery is a value
m[Georgia] = Atlanta
m[Michigan] = null
Keys: [Tennesee, Georgia, Alabama]
Values: [Nashville, Atlanta, Montgomery]
Iterating...
Tennesee=Nashville
Georgia=Atlanta
Alabama=Montgomery
*/
```

```
        // Extra TreeMap Stuff:
        boolean ctest =
            t.comparator()
             .equals(Collections.reverseOrder());
        System.out.println("Comparator test: " + ctest);
        System.out.println("firstKey = " +
                           t.firstKey());
        System.out.println("lastKey = " +
                           t.lastKey());
        System.out.println("headMap: " +
                           t.headMap("Georgia"));
        System.out.println("tailMap: " +
                           t.tailMap("Georgia"));
    }
}

/* Output:
Comparator test: true
firstKey = Tennesee
lastKey = Alabama
headMap: {Tennesee=Nashville}
tailMap: {Georgia=Atlanta, Alabama=Montgomery}
*/
```

An Application

- Cross-reference Lister
  - lists tokens and their line numbers
- Uses a Map
  - maps token string to a list of line numbers
  - could map to a Set of line numbers, but that's more expensive (re-balancing)
- Longer than C++ version
  - by 14 lines (107 vs. 93)
  - but *much* easier to get right

```
*** File: Xref.java ***
a: 2, 28
Add: 45, 68
addLast: 69
already: 49
and: 45
args: 95, 98, 107, 114, 115
at: 28
boolean: 51
BufferedReader: 21, 102, 116
(etc.)
```

```java
// Xref.java:
//  Generates a token-to-line cross-reference listing

import java.util.*;
import java.io.*;

class Xref
{
    // Comparator to ignore case:
    static class NoCase implements Comparator
    {
        public int compare(Object o1, Object o2)
        {
            String s1 = (String) o1;
            String s2 = (String) o2;
            return s1.compareToIgnoreCase(s2);
        }
    }
```

```
// This method does the work:
static void process(LineNumberReader r)
    throws IOException
{
    TreeMap map = new TreeMap(new NoCase());
    String line;

    // Build map, reading a line at a time:
    while ((line = r.readLine()) != null)
    {
        // Read each token:
        String delim =
            " `~!@#$%^&*()-_=+\\|[{]};:'\",<.>/?"
          + "0123456789";
        StringTokenizer tokens =
            new StringTokenizer(line, delim);
```

```
        while (tokens.hasMoreTokens())
        {
            String token = tokens.nextToken();

            if (!map.containsKey(token))
            {
                // Add token and empty list to map:
                map.put(token, new ArrayList());
            }

            // See if this line is in there already:
            LinkedList lines =
                (LinkedList) map.get(token);
            int lineno = r.getLineNumber();
            if (lines.isEmpty() ||
                ((Integer)lines.getLast()).intValue() !=
                 lineno)
            {
                // Add line number to list:
                lines.addLast(new Integer(lineno));
            }
        }
    }   // end of read-line logic
```

```
    // Output:
    Iterator p = map.entrySet().iterator();
    while (p.hasNext())
    {
        Map.Entry e = (Map.Entry) p.next();
        System.out.print(e.getKey() + ": ");
        ArrayList lines = (ArrayList) e.getValue();
        for (int i = 0; i < lines.size(); ++i)
        {
            if (i > 0)
                System.out.print(", ");
            System.out.print(lines.get(i));
        }
        System.out.println();
    }
}   // end of process() method
```

```
    public static void main(String[] args)
        throws IOException
    {
        if (args.length == 0)
        {
            // Process standard input:
            Reader r =
                new InputStreamReader(System.in);
            process(new LineNumberReader(r));
        }
        else
        {
            // Process each specified file:
            for (int i = 0; i < args.length; ++i)
            {
                if (i > 0)
                    System.out.println();
                System.out.println("*** File: " +
                                   args[i] + " ***");
                FileReader f = new FileReader(args[i]);
                process(new LineNumberReader(f));
            }
        }
    }
}
```

~ The End ~