3 Java Collections

Benefits of Java Collection Framework

- Reduces programming effort
  - You don't need to create your own data structures and algorithms
- Increases program speed and quality
  - Collections framework provides high-performance, high-quality implementations of useful data structures and algorithms
- Reduces effort to learn and use new APIs
- Reduces effort to design new APIs
- Foster software reuse

Java Collections

- Java Collections
  - Java built-in collection classes and interfaces
    - Found in the *java.util* package
  - Examples of collection classes:
    - Stack
    - LinkedList
    - ArrayList
    - HashSet
    - TreeSet

Java Collections

- *Collection* interface
  - Root of all collection interfaces

- Definition of Collection:
  - Group of objects, which are also called elements
  - May allow duplicates and requires no specific ordering

Java Collections

- Built-in subinterfaces of *Collection* interface
  - *Set* Interface
    - Unordered collection that contains no duplicates
    - Some built-in implementing classes: HashSet, LinkedHashSet and TreeSet
  - *List* Interface
    - Ordered collection of elements where duplicates are permitted
    - Some built-in implementing classes: ArrayList, LinkedList and Vector

Java Collections

- Java Collections Hierarchy

<table>
<tr><td colspan="6"><root interface>
Collection</td></tr>
<tr><td colspan="3"><interface>
Set</td><td colspan="3"><interface>
List</td></tr>
<tr><td colspan="3"><implementing classes></td><td colspan="3"><implementing classes></td></tr>
<tr><td>HashSet</td><td>LinkedHashSet</td><td>TreeSet</td><td>ArrayList</td><td>LinkedList</td><td>Vector</td></tr>
</table>

Java *Collection* Methods: Java 2 Platform SE v1.4.1

Collection Methods
`public boolean add(Object o)`
Inserts the *Object o* to this collection. Returns *true* if *o* was successfully added to the collection.
`public void clear()`
Removes all elements of this collection.
`public boolean remove(Object o)`
Removes a single instance of the *Object o* from this collection, if it is present. Returns *true* if *o* was found and removed from the collection.
`public boolean contains(Object o)`
Returns true if this collection contains the *Object o*.
`public boolean isEmpty()`
Returns true if this collection does not contain any object or element.

Java *Collection* Methods: Java 2 Platform SE v1.4.1

Collection Methods
`public int size()`
Returns the number of elements in this collection.
`public Iterator iterator()`
Returns an iterator that allows us to go through the contents of this collection.
`public boolean equals(Object o)`
Returns true if the *Object o* is equal to this collection.
`public int hashCode()`
Returns the hash code value (i.e., the ID) for this collection. Same objects or collections have the same hash code value or ID.
Returns true if this collection contains the *Object o*.

Java Collections: *LinkedList*

```
1  import java.util.*;
2  class LinkedListDemo {
3     public static void main(String args[]) {
4        LinkedList list = new LinkedList();
5        list.add(new Integer(1));
6        list.add(new Integer(2));
7        list.add(new Integer(3));
8        list.add(new Integer(1));
9        System.out.println(list+", size = "+list.size());
10       list.addFirst(new Integer(0));
11       list.addLast(new Integer(4));
12       System.out.println(list);
13       System.out.println(list.getFirst() + ", " +
14                                          list.getLast());
```

Java Collections: *LinkedList*

```
15 //continuation...
16         System.out.println(list.get(2)+", "+list.get(3));
17         list.removeFirst();
18         list.removeLast();
19         System.out.println(list);
20         list.remove(new Integer(1));
21         System.out.println(list);
22         list.remove(2);
23         System.out.println(list);
24         list.set(1, "one");
25         System.out.println(list);
26     }
27 }
```

Java Collections: *ArrayList*

- Definition:
  - Resizable version an ordinary array
  - Implements the *List* interface

- Example:

```
1 import java.util.*;
2 class ArrayListDemo {
3    public static void main(String args[]) {
4       ArrayList al = new ArrayList(2);
5       System.out.println(al+", size = "+al.size());
6       al.add("R");
7 //continued...
```

Java Collections: *ArrayList*

```
8          al.add("U");
9          al.add("O");
10         System.out.println(al+", size = "+al.size());
11         al.remove("U");
12         System.out.println(al+", size = "+al.size());
13         ListIterator li = al.listIterator();
14         while (li.hasNext())
15            System.out.println(li.next());
16         Object a[] = al.toArray();
17         for (int i=0; i<a.length; i++)
18            System.out.println(a[i]);
19      }
20   }
```

Java Collections: *HashSet*

- Definition:
  - Implementation of the *Set* interface that uses a hash table
  - Hash table
    - Uses a formula to determine where an object is stored.
  - Benefits of using a hash table
    - Allows easier and faster look up of elements

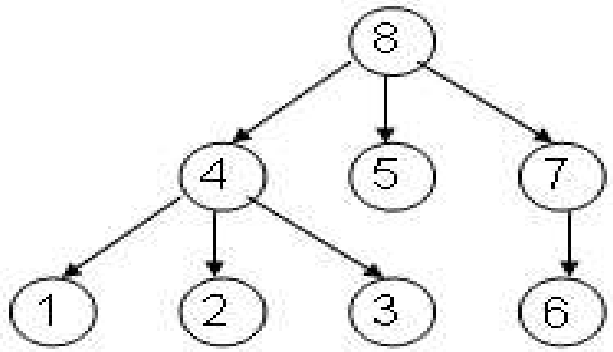
Java Collections: *HashSet*

```
1  import java.util.*;
2  class HashSetDemo {
3     public static void main(String args[]) {
4        HashSet hs = new HashSet(5);
5        System.out.println(hs.add("one"));
6        System.out.println(hs.add("two"));
7        System.out.println(hs.add("one"));
8        System.out.println(hs.add("three"));
9        System.out.println(hs.add("four"));
10       System.out.println(hs.add("five"));
11       System.out.println(hs);
12    }
13 }
```

Java Collections: *TreeSet*

- Definition:
  - Implementation of the *Set* interface that uses a tree
  - Tree
  - Ensures that the sorted set will be arranged in ascending order

- Tree representation



Java Collections: *TreeSet*

```
1  import java.util.*;
2  class TreeSetDemo {
3     public static void main(String args[]) {
4        TreeSet ts = new TreeSet();
5        ts.add("one");
6        ts.add("two");
7        ts.add("three");
8        ts.add("four");
9        System.out.println(ts);
10    }
11 }
```

Summary

- Recursion
  - Definition
  - Recursion Vs. Iteration
- Abstract Data Types
  - Definition
  - Stacks
  - Queues
  - Sequential and Linked Representation

Summary

- Java Collections
  - Collection
  - Linked List
  - ArrayList
  - HashSet
  - TreeSet