Java Collection Framework

Interface Collection
- `add(o)`: Add a new element
- `clear()`: Remove all elements
- `contains(o)`: Membership checking.
- `isEmpty()`: Whether it is empty
- `iterator()`: Return an iterator
- `remove(o)`: Remove an element
- `size()`: The number of elements

Interface List
- `add(i,o)`: Insert `o` at position `i`
- `add(o)`: Append `o` to the end
- `get(i)`: Return the `i`-th element
- `remove(i)`: Remove the `i`-th element
- `remove(o)`: Remove the element `o`
- `set(i,o)`: Replace the `i`-th element with `o`

Interface Map
- `clear()`: Remove all mappings
- `containsKey(k)`: Whether contains a mapping for `k`
- `containsValue(v)`: Whether contains a mapping to `v`
- `entrySet()`: Set of key-value pairs
- `get(k)`: The value associated with `k`
- `isEmpty()`: Whether it is empty
- `keySet()`: Set of keys
- `put(k,v)`: Associate `v` with `k`
- `remove(k)`: Remove the mapping for `k`
- `size()`: The number of pairs
- `values()`: The collection of values
Concrete Collections

<table>
<thead>
<tr>
<th>concrete collection</th>
<th>implements</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HashSet</td>
<td>Set</td>
<td>hash table</td>
</tr>
<tr>
<td>TreeSet</td>
<td>SortedSet</td>
<td>balanced binary tree</td>
</tr>
<tr>
<td>ArrayList</td>
<td>List</td>
<td>resizable-array</td>
</tr>
<tr>
<td>LinkedList</td>
<td>List</td>
<td>linked list</td>
</tr>
<tr>
<td>Vector</td>
<td>List</td>
<td>resizable-array</td>
</tr>
<tr>
<td>HashMap</td>
<td>Map</td>
<td>hash table</td>
</tr>
<tr>
<td>TreeMap</td>
<td>SortedMap</td>
<td>balanced binary tree</td>
</tr>
<tr>
<td>Hashtable</td>
<td>Map</td>
<td>hash table</td>
</tr>
</tbody>
</table>

Iterate Through Collections

- The Iterator interface:

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

- The iterator() method defined in the Collection interface:

    ```java
    Iterator iterator()
    ```

Using Set

Set set = new HashSet(); // instantiate a concrete set
// ...
set.add(obj); // insert an element
// ...
int n = set.size(); // get size
// ...
if (set.contains(obj)) {...} // check membership
// iterate through the set
Iterator iter = set.iterator();
while (iter.hasNext()) {
    Object e = iter.next();
    // downcast e
    // ...
}

Using Map

Map map = new HashMap(); // instantiate a concrete map
// ...
map.put(key, val); // insert a key-value pair
// ...
// get the value associated with key
Object val = map.get(key);
map.remove(key); // remove a key-value pair
// ...
if (map.containsValue(val)) {...} // ...
if (map.containsKey(key)) {...} // ...
Set keys = map.keySet(); // get the set of keys
// iterate through the set of keys
Iterator iter = keys.iterator();
while (iter.hasNext()) {
    Key key = (Key) iter.next();
    // ...
}
Counting Different Words

```java
public class CountWords {
    static public void main(String[] args) {
        Set words = new HashSet();
        BufferedReader in = new BufferedReader(
            new InputStreamReader(System.in));
        String delim = " \t\n.,:;?!-/()[]";
        String line;
        int count = 0;
        try {
            while ((line = in.readLine()) != null) {
                StringTokenizer st =
                    new StringTokenizer(line, delim);
                while (st.hasMoreTokens()) {
                    count++;
                    words.add(st.nextToken().toLowerCase());
                }
            }
        } catch (IOException e) {
        }
        System.out.println("Total number of words: "+ count);      
        System.out.println("Number of different words: "+ words.size());
    }
}
```

Word Frequency

```java
public class Count {
    public Count(String word, int i) {
        this.word = word;
        this.i = i;
    }
    public String word;
    public int i;
}
```

Word Frequency (cont'd)

```java
public class WordFrequency {
    static public void main(String[] args) {
        Map words = new HashMap();
        String delim = " \t\n.,:;?!-/()[]";
        BufferedReader in =
            new BufferedReader(
                new InputStreamReader(System.in));
        String line, word;
        Count count;
```
try { 
    while ((line = in.readLine()) != null) { 
        StringTokenizer st = 
            new StringTokenizer(line, delim); 
        while (st.hasMoreTokens()) { 
            word = st.nextToken().toLowerCase(); 
            count = (Count) words.get(word); 
            if (count == null) { 
                words.put(word, 
                        new Count(word, 1)); 
            } else { 
                count.i++; 
            } 
        } 
    } 
} catch (IOException e) {} 

Set set = words.entrySet(); 
Iterator iter = set.iterator(); 
while (iter.hasNext()) { 
    Map.Entry entry = 
        (Map.Entry) iter.next(); 
    word = (String) entry.getKey(); 
    count = (Count) entry.getValue(); 
    System.out.println(word + 
                        (word.length() < 8 ? "\t" : "") + count.i); 
} 

Word Frequency Output
Using President Lincoln's *The Gettysburg Address*
 as the input, the output is:

<table>
<thead>
<tr>
<th>Word</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>devotion</td>
<td>2</td>
</tr>
<tr>
<td>years</td>
<td>1</td>
</tr>
<tr>
<td>civil</td>
<td>1</td>
</tr>
<tr>
<td>place</td>
<td>1</td>
</tr>
<tr>
<td>gave</td>
<td>2</td>
</tr>
<tr>
<td>they</td>
<td>3</td>
</tr>
<tr>
<td>struggled</td>
<td>1</td>
</tr>
<tr>
<td>.......</td>
<td></td>
</tr>
<tr>
<td>men</td>
<td>2</td>
</tr>
<tr>
<td>remember</td>
<td>1</td>
</tr>
<tr>
<td>who</td>
<td>3</td>
</tr>
<tr>
<td>did</td>
<td>1</td>
</tr>
<tr>
<td>work</td>
<td>1</td>
</tr>
<tr>
<td>rather</td>
<td>2</td>
</tr>
<tr>
<td>fathers</td>
<td>1</td>
</tr>
</tbody>
</table>

Ordering and Sorting

There are two ways to define orders on objects.
- Each class can define a *natural order* among its instances by implementing the `Comparable` interface.
  ```java
  int compareTo(Object o)
  ```
- Arbitrary orders among different objects can be defined by *comparators*, classes that implement the `Comparator` interface.
  ```java
  int compare(Object o1, Object o2)
  ```
**Word Frequency II**

```java
public class WordFrequency2 {
    static public void main(String[] args) {
        Map words = new TreeMap();
        //same as WordFrequency
    }
}
```

**Word Frequency II Output**

Using President Lincoln's *The Gettysburg Address* as the input, the output is:

<table>
<thead>
<tr>
<th>Word</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>7</td>
</tr>
<tr>
<td>above</td>
<td>1</td>
</tr>
<tr>
<td>add</td>
<td>1</td>
</tr>
<tr>
<td>address</td>
<td>1</td>
</tr>
<tr>
<td>advanced</td>
<td>1</td>
</tr>
<tr>
<td>ago</td>
<td>1</td>
</tr>
<tr>
<td>all</td>
<td>1</td>
</tr>
<tr>
<td>......</td>
<td>1</td>
</tr>
<tr>
<td>whether</td>
<td>1</td>
</tr>
<tr>
<td>which</td>
<td>2</td>
</tr>
<tr>
<td>who</td>
<td>3</td>
</tr>
<tr>
<td>will</td>
<td>1</td>
</tr>
<tr>
<td>work</td>
<td>1</td>
</tr>
<tr>
<td>world</td>
<td>1</td>
</tr>
<tr>
<td>years</td>
<td>1</td>
</tr>
</tbody>
</table>

**User-Defined Order**

Reverse alphabetical order of strings

```java
public class StringComparator
    implements Comparator {
    public int compare(Object o1, Object o2) {
        if (o1 != null &&
            o2 != null &&
            o1 instanceof String &&
            o2 instanceof String) {
            String s1 = (String) o1;
            String s2 = (String) o2;
            return - (s1.compareTo(s2));
        } else {
            return 0;
        }
    }
}
```

**Word Frequency III**

```java
public class WordFrequency3 {
    static public void main(String[] args) {
        Map words =
            new TreeMap(new StringComparator());
        //same as WordFrequency
    }
}
```
Word Frequency III Output
Using President Lincoln's *The Gettysburg Address* as the input, the output is:

<table>
<thead>
<tr>
<th>Word</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>years</td>
<td>1</td>
</tr>
<tr>
<td>world</td>
<td>1</td>
</tr>
<tr>
<td>work</td>
<td>1</td>
</tr>
<tr>
<td>will</td>
<td>1</td>
</tr>
<tr>
<td>who</td>
<td>3</td>
</tr>
<tr>
<td>which</td>
<td>2</td>
</tr>
<tr>
<td>whether</td>
<td>1</td>
</tr>
<tr>
<td>......</td>
<td>1</td>
</tr>
<tr>
<td>all</td>
<td>1</td>
</tr>
<tr>
<td>ago</td>
<td>1</td>
</tr>
<tr>
<td>advanced</td>
<td>1</td>
</tr>
<tr>
<td>address</td>
<td>1</td>
</tr>
<tr>
<td>add</td>
<td>1</td>
</tr>
<tr>
<td>above</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>7</td>
</tr>
</tbody>
</table>

Word Frequency IV

```java
public class WordFrequency4 {
    static public void main(String[] args) {
        List list = new ArrayList(words.values());
        Collections.sort(list, new CountComparator());
        Iterator iter = list.iterator();
        while (iter.hasNext()) {
            Count count = (Count) iter.next();
            System.out.println(count.word + (count.word.length() < 8 ? "\t\t" : "\t") + count.i);
        }
    }
}
```

Word Frequency IV Output
Using President Lincoln's *The Gettysburg Address* as the input, the output is:

<table>
<thead>
<tr>
<th>Word</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>the</td>
<td>13</td>
</tr>
<tr>
<td>that</td>
<td>12</td>
</tr>
<tr>
<td>we</td>
<td>10</td>
</tr>
<tr>
<td>here</td>
<td>8</td>
</tr>
<tr>
<td>to</td>
<td>8</td>
</tr>
<tr>
<td>a</td>
<td>7</td>
</tr>
<tr>
<td>and</td>
<td>6</td>
</tr>
<tr>
<td>......</td>
<td></td>
</tr>
<tr>
<td>consecrate</td>
<td>1</td>
</tr>
<tr>
<td>world</td>
<td>1</td>
</tr>
<tr>
<td>remember</td>
<td>1</td>
</tr>
<tr>
<td>did</td>
<td>1</td>
</tr>
<tr>
<td>work</td>
<td>1</td>
</tr>
<tr>
<td>fathers</td>
<td>1</td>
</tr>
</tbody>
</table>

Sorting
```
public class CountComparator
    implements Comparator {
    public int compare(Object o1, Object o2) {
        if (o1 != null && o2 != null && o1 instanceof Count && o2 instanceof Count) {
            Count c1 = (Count) o1;
            Count c2 = (Count) o2;
            return (c2.i - c1.i);
        } else {
            return 0;
        }
    }
}
```

```java
public class WordFrequency4 {
    static public void main(String[] args) {
        List list = new ArrayList(words.values());
        Collections.sort(list, new CountComparator());
        Iterator iter = list.iterator();
        while (iter.hasNext()) {
            Count count = (Count) iter.next();
            System.out.println(count.word + (count.word.length() < 8 ? "\t\t" : "\t") + count.i);
        }
    }
}
```

Word Frequency IV Output
Using President Lincoln's *The Gettysburg Address* as the input, the output is:

<table>
<thead>
<tr>
<th>Word</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>the</td>
<td>13</td>
</tr>
<tr>
<td>that</td>
<td>12</td>
</tr>
<tr>
<td>we</td>
<td>10</td>
</tr>
<tr>
<td>here</td>
<td>8</td>
</tr>
<tr>
<td>to</td>
<td>8</td>
</tr>
<tr>
<td>a</td>
<td>7</td>
</tr>
<tr>
<td>and</td>
<td>6</td>
</tr>
<tr>
<td>......</td>
<td></td>
</tr>
<tr>
<td>consecrate</td>
<td>1</td>
</tr>
<tr>
<td>world</td>
<td>1</td>
</tr>
<tr>
<td>remember</td>
<td>1</td>
</tr>
<tr>
<td>did</td>
<td>1</td>
</tr>
<tr>
<td>work</td>
<td>1</td>
</tr>
<tr>
<td>fathers</td>
<td>1</td>
</tr>
</tbody>
</table>
Java supports two types of IO:

- **Stream IO**
  - A stream is a sequence of bytes.
  - Stream-based IO supports reading or writing data sequentially.
  - A stream may be opened for reading or writing, but not reading and writing.
  - There are two types of streams: the *byte streams* and the *character stream*.

- **Random access IO**
  - Random access IO support reading and writing data at any positions of a file. A random access file may be opened for reading and writing.

### Standard Input/Output Streams

```java
class java.lang.System {
    public static final InputStream in;
    public static final PrintStream out;
    public static final PrintStream err;
    // ...
}
```
Using Reader and Writer

BufferedReader in
    = new BufferedReader(
        new FileReader("foo.in"));

BufferedReader in
    = new BufferedReader(
        new InputStreamReader(System.in));

PrintWriter out
    = new PrintWriter(
        new BufferedWriter(
            new FileWriter("foo.out")));

Writer out
    = new BufferedWriter(
        new OutputStreamWriter(System.out));

Character Encoding

By default, the character encoding is specified by the system property:

    file.encoding=8859_1

You can use other encoding by doing the following

BufferedReader in =
    new BufferedReader(
        new InputStreamReader(
            new FileInputStream("foo.in"), "GB2312"));

PrintWriter out =
    new PrintWriter(
        new BufferedWriter(
            new OutputStreamWriter(
                new FileOutputStream("foo.out", "GB2312"))));