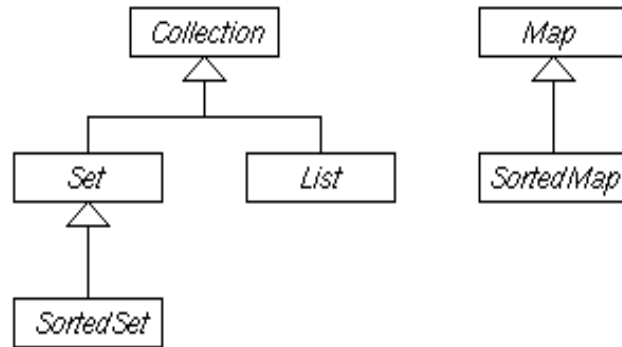


Java Collection Framework

A simple, extensible framework to handle collections of objects



Interface Collection

An unordered collection of possibly duplicate objects, a.k.a. A bag.

- `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

A sequential collection of objects, a.k.a. Sequence.

- `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 Set

A *set* is an unordered collection of non-duplicate objects.

- Same methods as *Collection*
- The `add()` method will check if the element occurs in the set.

An *ordered set* is a set whose elements are sorted in certain order.

Interface Map

A collection of key-value pairs.

♦ <code>clear()</code>	Remove all mappings
♦ <code>containsKey(k)</code>	Whether contains a mapping for <code>k</code>
♦ <code>containsValue(v)</code>	Whether contains a mapping to <code>v</code>
♦ <code>entrySet()</code>	Return the set of key-value pairs
♦ <code>get(k)</code>	The value associated with <code>k</code>
♦ <code>isEmpty()</code>	Whether it is empty
♦ <code>keySet()</code>	Return the set of keys
♦ <code>put(k, v)</code>	Associate <code>v</code> with <code>k</code>
♦ <code>remove(k)</code>	Remove the mapping for <code>k</code>
♦ <code>size()</code>	Return the number of pairs
♦ <code>values()</code>	Return the collection of values

Concrete Collections

concrete collection	implements	description
HashSet	Set	hash table
TreeSet	SortedSet	balanced binary tree
ArrayList	List	resizable-array
LinkedList	List	linked list
Vector	List	resizable-array
HashMap	Map	hash table
TreeMap	SortedMap	balanced binary tree
Hashtable	Map	hash table

Iterate Through Collections

- ♦ The `Iterator` interface:

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

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

```
Iterator iterator()
```

Using Set

```
Set set = new HashSet(); // instantiate a concrete set
// ...
set.add(obj); // insert an elements
// ...
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

```
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;
    }
}
```

Counting Different Words

```
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

```
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)

```
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;
```

Word Frequency (cont'd)

(class WordFrequency continued.)

```
        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) {}  
    }  
}
```

Word Frequency (cont'd)

(class WordFrequency continued.)

```
        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\t" : "\t") +  
                count.i);  
        }  
    }  
}
```

Word Frequency Output

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

devotion	2
years	1
civil	1
place	1
gave	2
they	3
struggled	1
.....	
men	2
remember	1
who	3
did	1
work	1
rather	2
fathers	1

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.

```
int compareTo(Object o)
```

- Arbitrary orders among different objects can be defined by *comparators*, classes that implement the Comparator interface.

```
int compare(Object o1, Object o2)
```

Word Frequency II

```
public class WordFrequency2 {
    static public void main(String[] args) {
        Map words = new TreeMap();
        <smac as WordFrequency>
    }
}
```

Word Frequency II Output

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

a	7
above	1
add	1
address	1
advanced	1
ago	1
all	1
.....	
whether	1
which	2
who	3
will	1
work	1
world	1
years	1

User-Defined Order

Reverse alphabetical order of strings

```
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

```
public class WordFrequency3 {
    static public void main(String[] args) {
        Map words =
            new TreeMap(new StringComparator());
        <smac as WordFrequency>
    }
}
```

Word Frequency III Output

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

```
years          1
world          1
work           1
will           1
who            3
which          2
whether         1
.....
all            1
ago            1
advanced       1
address        1
add            1
above          1
a              7
```

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;
        }
    }
}
```

Word Frequency IV

```
public class WordFrequency4 {
    static public void main(String[] args) {
        <smac as WordFrequency>
        List list = new ArrayList(words.values());
        Collections.sort(list,
            new CountComparator());
        Iterator iter = list.iterator();
        while (iter.hasNext()) {
            count = (Count) iter.next();
            word = count.word;
            System.out.println(word +
                (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:

```

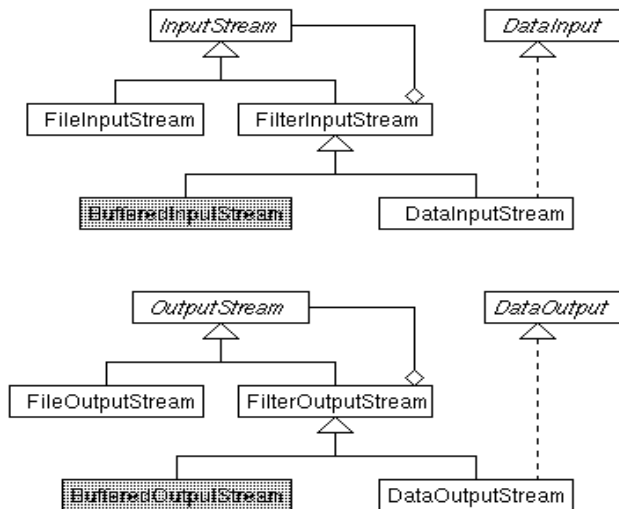
the          13
that        12
we          10
here         8
to           8
a            7
and          6
.....
consecrate   1
world        1
consecrated  1
remember    1
did          1
work         1
fathers      1
    
```

The Input/Output Framework

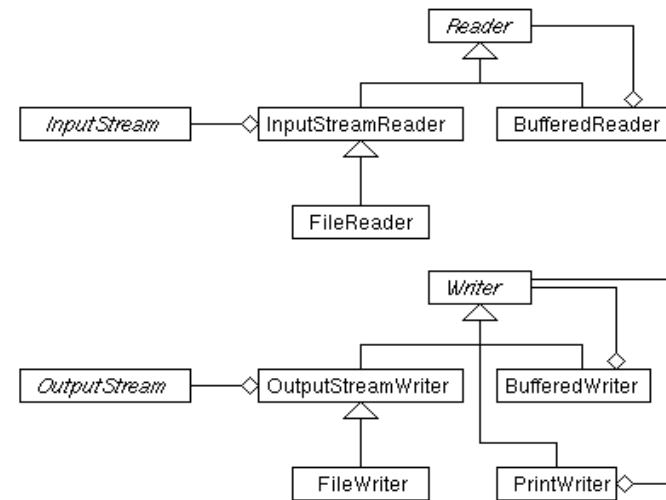
Java supports two types of IO:

- ◆ 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.

Byte Streams



Character Streams



Standard Input/Output Streams

```
public 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));
```

Write and Read

Write and read a simple matrix of doubles.

The data

```
public interface MatrixData {
    double[][] data = {
        { Math.exp(2.0), Math.exp(3.0),
          Math.exp(4.0) },
        { Math.exp(-2.0), Math.exp(-3.0),
          Math.exp(-4.0) },
    };
}
```

Write Using OutputStream

```
import java.io.*;

public class WriteMatrix1 implements MatrixData {
    public static void main(String[] args) {
        int row = data.length;
        int col = data[0].length;
        int i, j;
        if (args.length > 0) {
            try {
                FileOutputStream out = new FileOutputStream(args[0]);
                writeInt(row, out);
                writeInt(col, out);
                for (i = 0; i < row; i++) {
                    for (j = 0; j < col; j++) {
                        writeDouble(data[i][j], out);
                    }
                }
                out.close();
            } catch (IOException e) {}
        }
        // ...
    }
}
```

Write Using OutputStream

```
public static void writeInt(int i, OutputStream out)
    throws IOException {
    byte[] buf = new byte[4];
    for (int k = 3; k >= 0; k--) {
        buf[k] = (byte)(i & 0xFF);
        i >>= 8;
    }
    out.write(buf);
}

public static void writeDouble(double d, OutputStream out)
    throws IOException {
    byte[] buf = new byte[8];
    long l = Double.doubleToLongBits(d);
    for (int k = 7; k >= 0; k--) {
        buf[k] = (byte)(l & 0xFF);
        l >>= 8;
    }
    out.write(buf);
}
```

Read Using InputStream

```
import java.io.*;

public class ReadMatrix1 {
    static double[][] data;
    public static void main(String[] args) {
        if (args.length > 0) {
            try {
                FileInputStream in = new FileInputStream(args[0]);
                int row = readInt(in);
                System.out.println("row = " + row);
                int col = readInt(in);
                System.out.println("col = " + col);
                data = new double[row][col];
                for (int i = 0; i < row; i++) {
                    for (int j = 0; j < col; j++) {
                        data[i][j] = readDouble(in);
                    }
                }
            } catch (IOException e) {}
        }
        // ...
    }
}
```

Read Using InputStream

```
public static int readInt(InputStream in)
    throws IOException {
    byte[] buf = new byte[4];
    in.read(buf);
    int i = 0;
    for (int k = 0; k < 4; k++) {
        i <<= 8;
        i += (((int) buf[k]) & 0xFF);
    }
    return i;
}

public static double readDouble(InputStream in)
    throws IOException {
    byte[] buf = new byte[8];
    in.read(buf);
    long l = 0;
    for (int k = 0; k < 8; k++) {
        l <<= 8;
        l += (((int) buf[k]) & 0xFF);
    }
    return Double.longBitsToDouble(l);
}
```

Write Using DataOutputStream

```
DataOutputStream out =
    new DataOutputStream(
        new FileOutputStream(args[0]));

out.writeInt(row);
out.writeInt(col);
for (i = 0; i < row; i++) {
    for (j = 0; j < col; j++) {
        out.writeDouble(data[i][j]);
    }
}
out.close();
```

Read Using DataInputStream

```
DataInputStream in =
    new DataInputStream(
        new FileInputStream(args[0]));

int row = in.readInt();
int col = in.readInt();
data = new double[row][col];
for (int i = 0; i < row; i++) {
    for (int j = 0; j < col; j++) {
        data[i][j] = in.readDouble();
    }
}
```

Write Using Buffered Stream

```
DataOutputStream out =
    new DataOutputStream(
        new BufferedOutputStream(
            new FileOutputStream(args[0])));

out.writeInt(row);
out.writeInt(col);
for (i = 0; i < row; i++) {
    for (j = 0; j < col; j++) {
        out.writeDouble(data[i][j]);
    }
}
out.close();
```

Read Using Buffered Stream

```
DataInputStream in =
    new DataInputStream(
        new BufferedInputStream(
            new FileInputStream(args[0])));

int row = in.readInt();
int col = in.readInt();
data = new double[row][col];
for (int i = 0; i < row; i++) {
    for (int j = 0; j < col; j++) {
        data[i][j] = in.readDouble();
    }
}
```

Write Using Object Stream

```
ObjectOutputStream out =
    new ObjectOutputStream(
        new FileOutputStream(args[0]));

out.writeObject(data);
out.close();
```

Read Using Object Stream

```
ObjectInputStream in =
    new ObjectInputStream(
        new FileInputStream(args[0]));
data = (double[][]).in.readObject();
```

Write Using Character Stream

```
PrintWriter out =
    new PrintWriter(
        new BufferedWriter(
            new FileWriter(args[0])));
out.println(row);
out.println(col);
for (i = 0; i < row; i++) {
    for (j = 0; j < col; j++) {
        out.println(data[i][j]);
    }
}
out.close();
```

Read Using Character Stream

```
BufferedReader in =
    new BufferedReader(
        new FileReader(args[0]));
String line;
line = in.readLine();
int row = Integer.parseInt(line);
line = in.readLine();
int col = Integer.parseInt(line);
data = new double[row][col];
for (int i = 0; i < row; i++) {
    for (int j = 0; j < col; j++) {
        line = in.readLine();
        data[i][j] =
            Double.valueOf(line).doubleValue();
    }
}
```