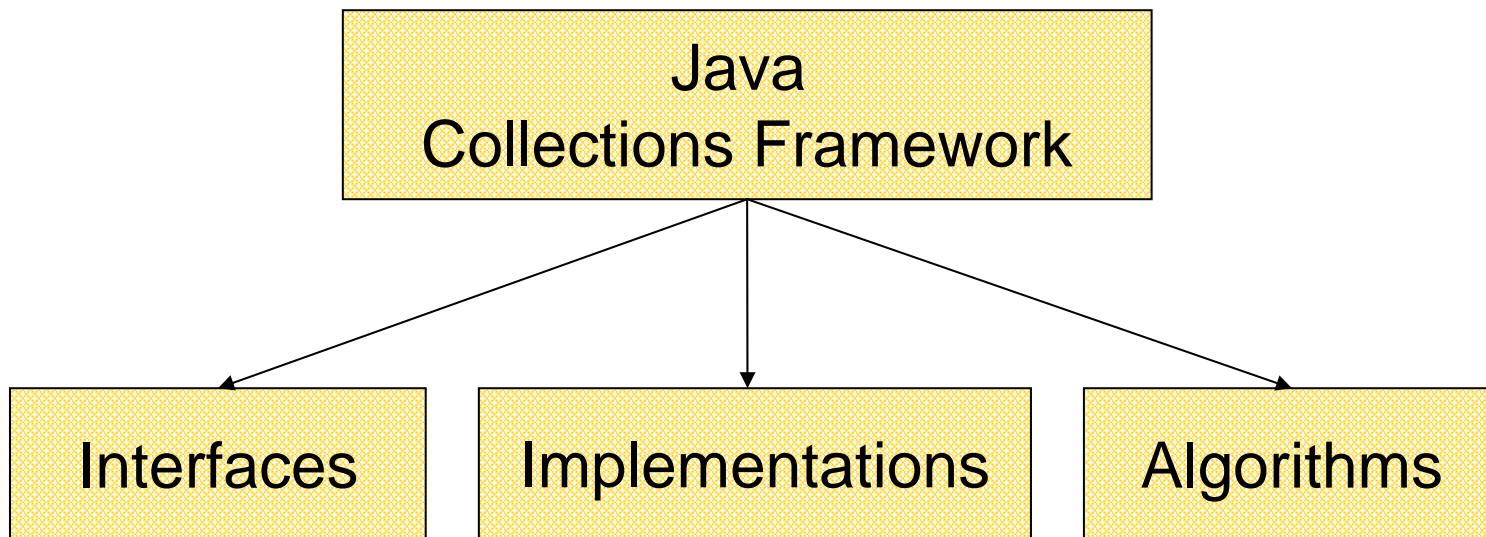


# Software 1 with Java

Recitation No. 4  
(Collections)

# Java Collections Framework

- **Collection:** a group of elements
- Interface Based Design:



# Online Resources

---

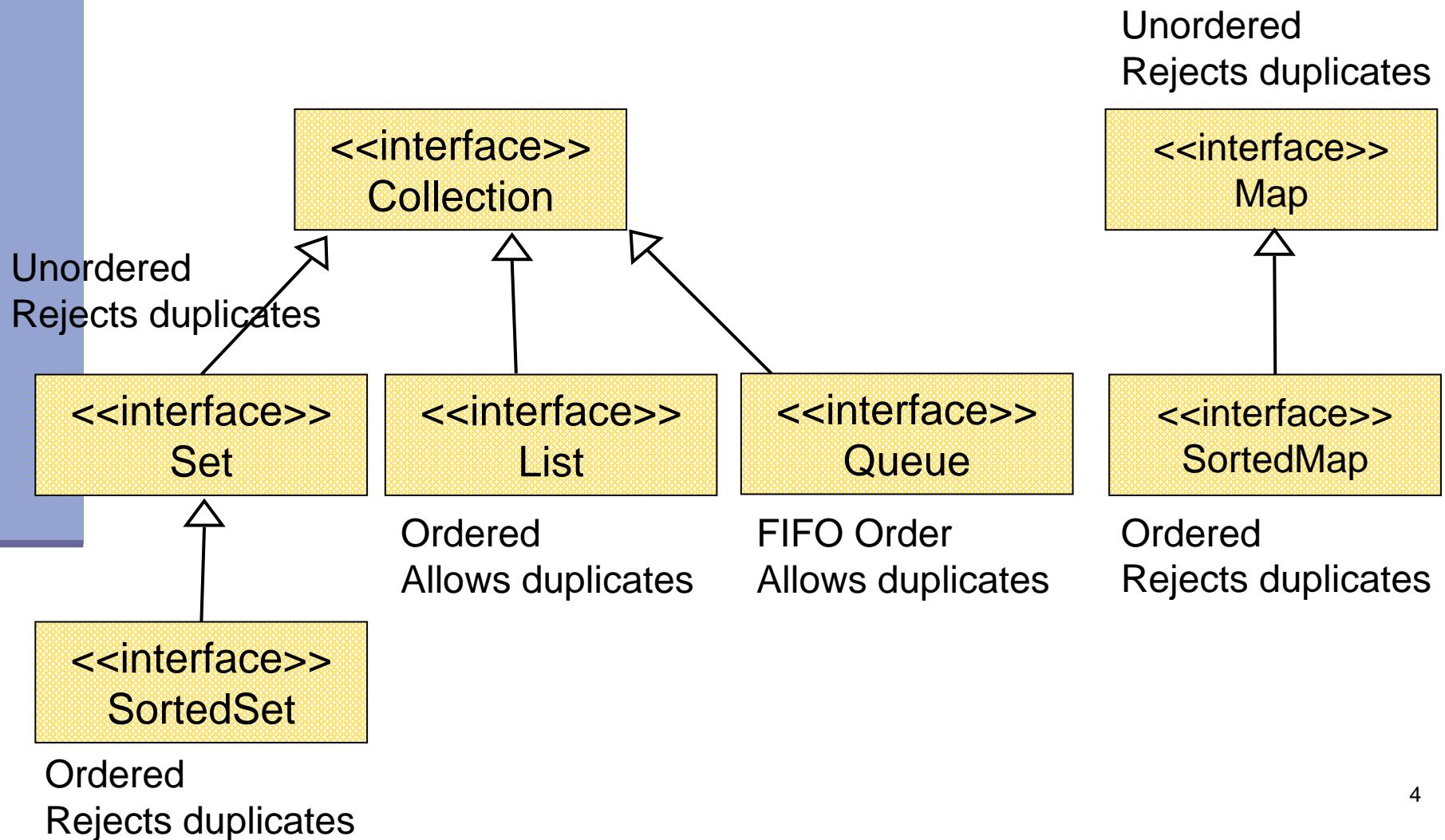
- Java 5 API Specification:

<http://java.sun.com/j2se/1.5.0/docs/api/index.html>

- Sun Tutorial:

<http://java.sun.com/docs/books/tutorial/collections/>

# Collection Interfaces

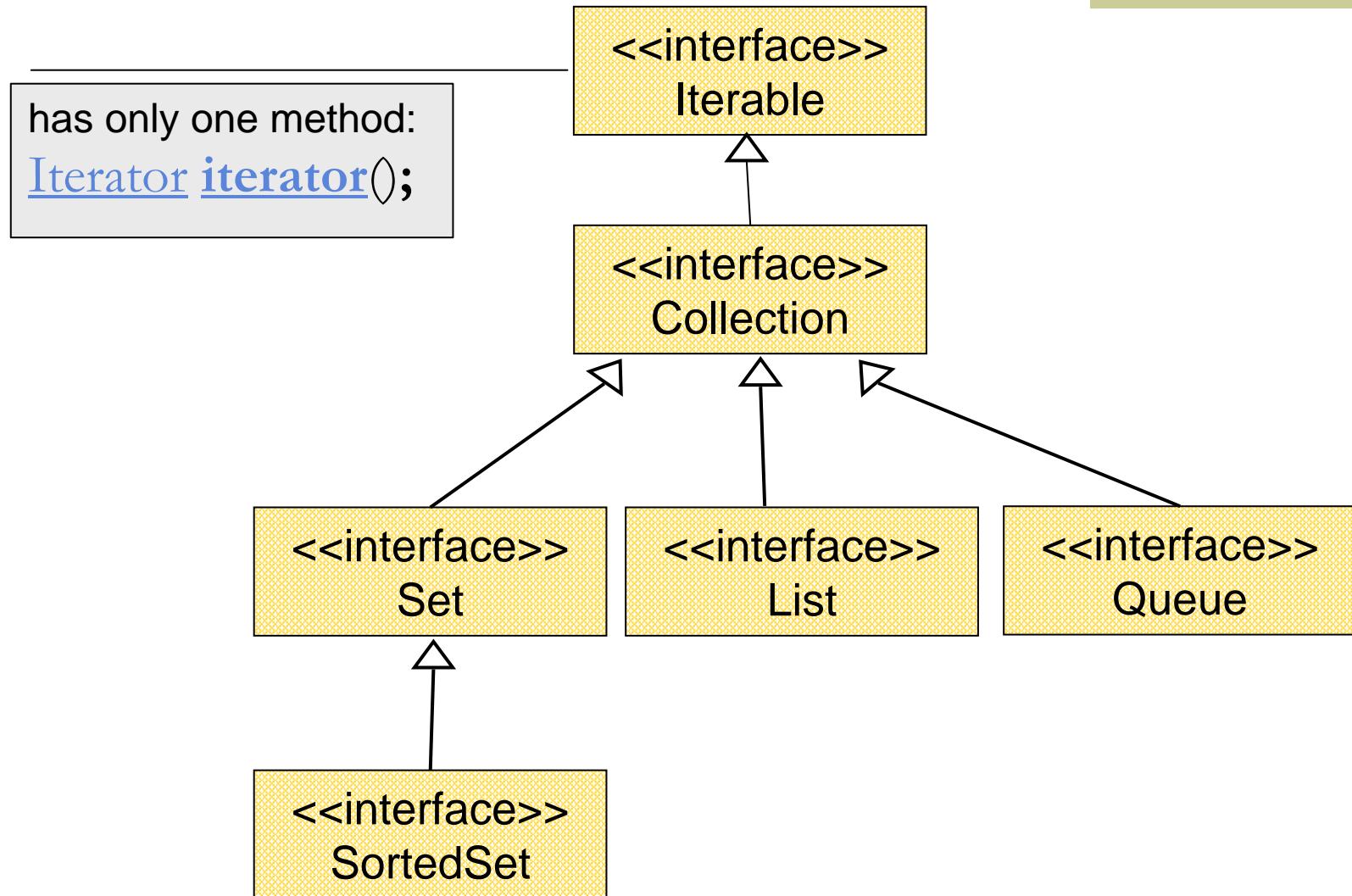


# The Collection Interface

---

- Holds any Object references
  - Not type safe
  - Use casting
- Doesn't hold primitives
  - Use wrapper classes
- Since Java5 collections are type-safe
  - Will be discussed later in the course

# Collection extends Iterable



# The Iterator Interface

---

- Provide a way to access the elements of a collection sequentially **without exposing its underlying representation**
- Methods:
  - `hasNext()` - Returns true if there are more elements
  - `next()` - Returns the next element
  - `remove()` - Removes the last element returned by the iterator (optional operation)

# Iterating over a Collection

---

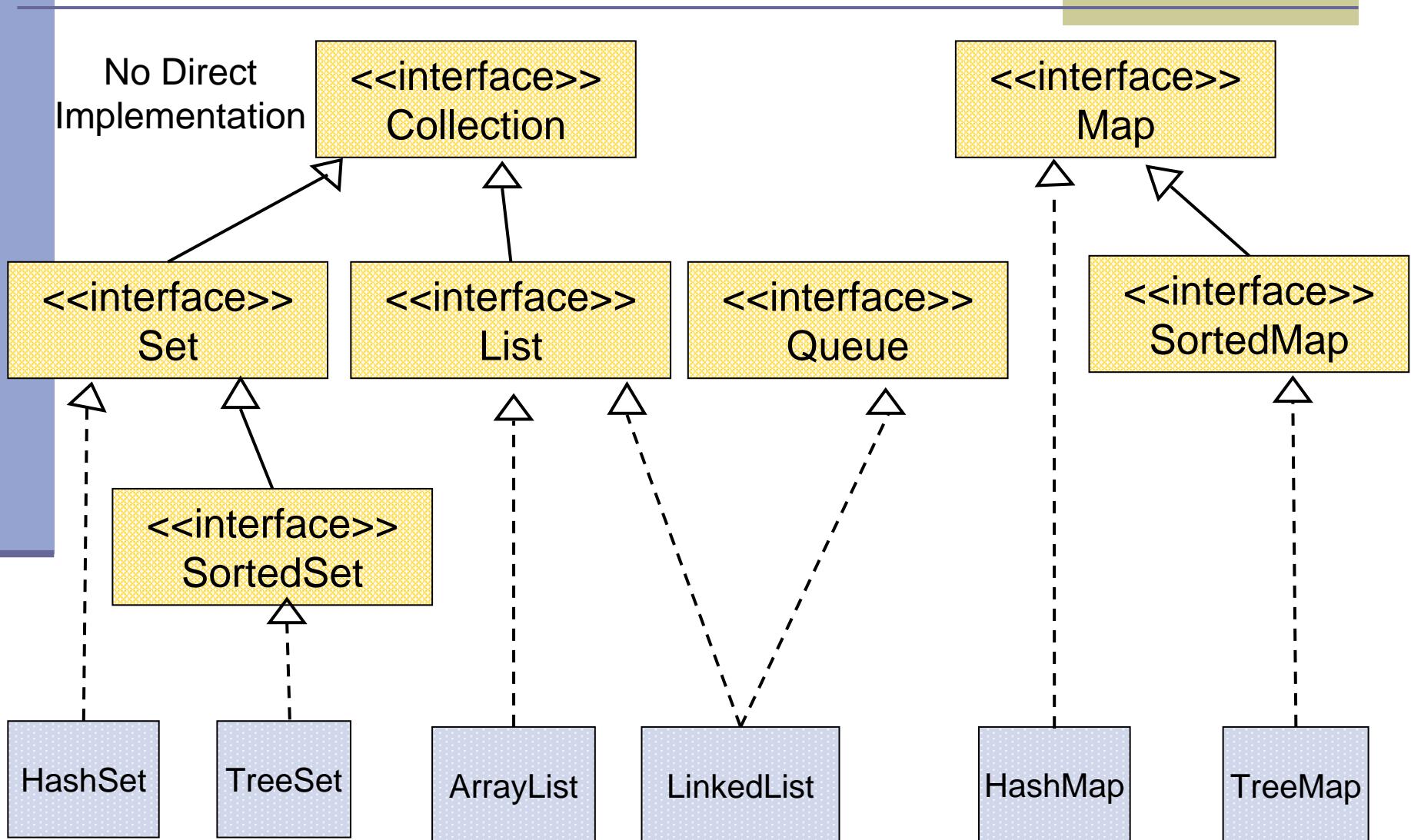
```
for (Iterator iter = collection.iterator() ;  
     iter.hasNext(); ) {  
    System.out.println(iter.next());  
}
```

# Collection Implementations

- Class Name Convention: <Data structure> <Interface>

General Purpose Implementations		Data Structures			
Interfaces	Set	HashTable	Resizable Array	Balanced Tree	Linked List
	Queue	HashSet		TreeSet (SortedSet)	
	List				LinkedList
	Map	HashMap	ArrayList		LinkedList
				TreeMap (SortedMap)	

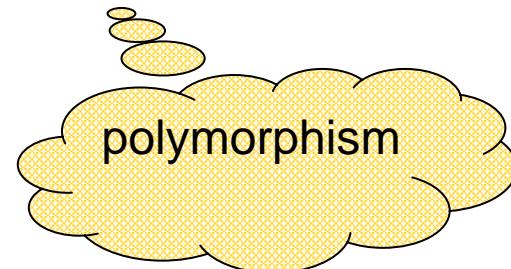
# General Purpose Implementations



# Best Practice

- Specify an implementation only when a collection is constructed:

- `Set s = new HashSet();`  
     $\underbrace{\text{Interface}}$                            $\underbrace{\text{Implementation}}$
- `public void foo(HashSet s) { ... }`      **Works,  
but...**  
`public void foo(Set s) { ... }`              **Better!**
- `s.add()` invokes `HashSet.add()`



Interface

# List Example

```
List list = new ArrayList();
list.add(3);
list.add(1);
list.add(new Integer(1));
list.add(new Integer(6));
list.remove(list.size()-1);
System.out.println(list);
```

Implementation

List holds  
Object  
references  
(auto-boxing)

List allows  
duplicates

Invokes  
List.toString()

remove() can get an  
index or a reference  
as argument

Insertion  
order is kept

**Output:**

[ 3 , 1 , 1 ]

# Set Example

```
Set set = new HashSet();
set.add(3);
set.add(1);
set.add(new Integer(1));
set.add(new Integer(6));
set.remove(6);
System.out.println(set);
```

A Set does not allow duplicates.  
it does not contain:  
two references to the same object  
two references to null  
references to two objects a and b  
such that a.equals(b)

remove() can only get a  
reference as argument

Output: [1, 3]

Insertion order is  
not guaranteed

# Queue Example

```
Queue queue = new LinkedList();
queue.add(3);
queue.add(1);      ...
queue.add(new Integer(1));
queue.add(new Integer(6));
queue.remove();
System.out.println(queue)
```

Elements are added  
to the tail of the  
queue

remove() has no  
argument – head is  
removed

Output: [1, 1, 6]

FIFO order

# Map Example

```
Map map = new HashMap();
map.put("Dan", "03-9516743");
map.put("Rita", "09-5076452");
map.put("Leo", "08-5530098");
map.put("Rita", "06-8201124");
System.out.println(map);
```

No duplicates

Unordered

Output:

```
{Leo=08-5530098, Dan=03-9516743, Rita=06-8201124}
```

Keys (names)	Values (phone numbers)
Dan	03-9516743
Rita	06-8201124
Leo	08-5530098

# SortedMap Example

```
SortedMap map = new TreeMap();  
map.put("Dan", "03-9516743");  
map.put("Rita", "09-5076452");  
map.put("Leo", "08-5530098");  
map.put("Rita", "06-8201124");  
System.out.println(map);
```

lexicographic order

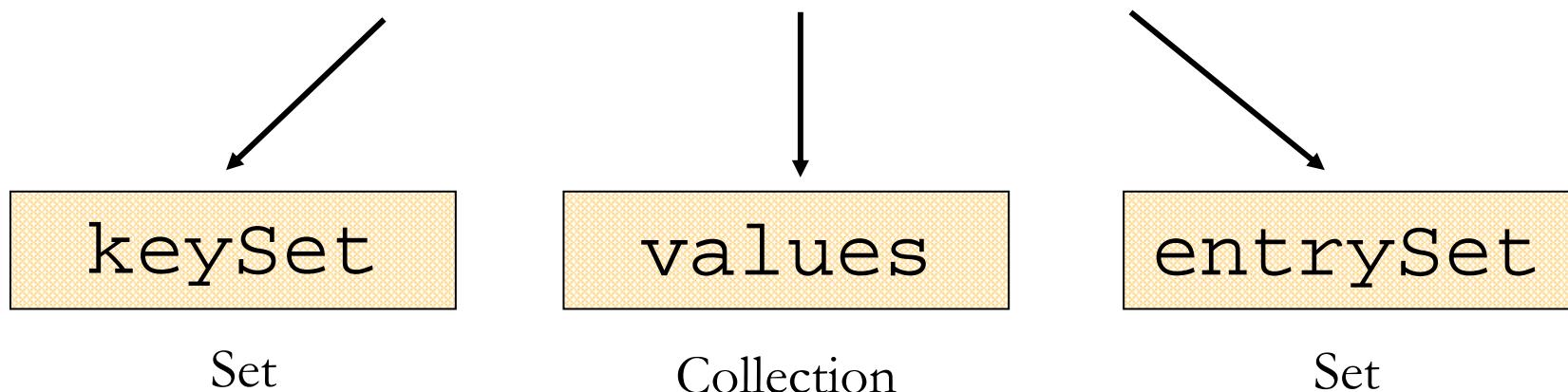
Output:

{Dan=03-9516743, Leo=08-5530098, Rita=06-8201124}

Keys (names)	Values (phone numbers)
Dan	03-9516743
Rita	06-8201124
Leo	08-5530098

# Map Collection Views

Three views of a Map as a collection



The Set of key-value pairs  
(implement Map.Entry)

# Iterating Over the Keys of a Map

---

```
Map map = new HashMap();
map.put("Dan", "03-9516743");
map.put("Rita", "09-5076452");
map.put("Leo", "08-5530098");
map.put("Rita", "06-8201124");
```

```
for (Iterator iter= map.keySet().iterator(); iter.hasNext(); ) {
    System.out.println(iter.next());
}
```

Output:            Leo  
                  Dan  
                  Rita

# Iterating Over the Keys of a Map

---

```
Map map = new HashMap();
map.put("Dan", "03-9516743");
map.put("Rita", "09-5076452");
map.put("Leo", "08-5530098");
map.put("Rita", "06-8201124");

for (Object key : map.keySet()) {
    System.out.println(key);
}
```

Output:      Leo  
                  Dan  
                  Rita

# Iterating Over the Key-Value Pairs of a Map

```
Map map = new HashMap();
map.put("Dan", "03-9516743");
map.put("Rita", "09-5076452");
map.put("Leo", "08-5530098");
map.put("Rita", "06-8201124");

for (Iterator iter= map.entrySet().iterator(); iter.hasNext();) {
    Map.Entry entry = (Map.Entry) iter.next();
    System.out.println(entry.getKey() + ": " + entry.getValue());
}
```

## Output:

Leo: 08-5530098  
Dan: 03-9516743  
Rita: 06-8201124

casting

# Collection Algorithms

---

- Defined in the [Collections](#) class
- Main algorithms:
  - sort
  - binarySearch
  - reverse
  - shuffle
  - min
  - max

# Sorting

```
import java.util.*;
```

import the package of  
List, Collections  
and Arrays

```
public class Sort {  
    public static void main(String args[]) {  
        List list = Arrays.asList(args);  
        Collections.sort(list);  
        System.out.println(list);  
    }  
}
```

returns a List-view of  
its array argument.

Arguments: A C D B

Output: [A, B, C, D]

lexicographic  
order

# Sorting (cont.)

---

- Sort a List `l` by `Collections.sort(l);`
- If the list consists of `String` objects it will be sorted in lexicographic order. Why?
- String implements Comparable:

```
public interface Comparable {  
    public int compareTo(T o);  
}
```
- Exceptions when sorting a list with:
  - elements that do not implement Comparable or
  - are not *mutually comparable*.

---



Questions ???