

Java Collections Framework

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

```

graph TD
    JCF[Java Collections Framework] --> Interfaces[Interfaces]
    JCF --> Implementations[Implementations]
    JCF --> Algorithms[Algorithms]
    
```

2

Software 1 with Java

Recitation No. 4 (Collections)

3

Collection Interfaces

```

graph TD
    Collection["<<interface>> Collection"] --> Set["<<interface>> Set"]
    Collection --> List["<<interface>> List"]
    Collection --> Queue["<<interface>> Queue"]
    Collection --> Map["<<interface>> Map"]
    Set --> SortedSet["<<interface>> SortedSet"]
    Map --> SortedMap["<<interface>> SortedMap"]
    
```

4

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/>

3

Collection extends Iterable

```

graph TD
    Iterable["<<interface>> Iterable"] --> Collection["<<interface>> Collection"]
    
```

6

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

5

Iterating over a Collection

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

8

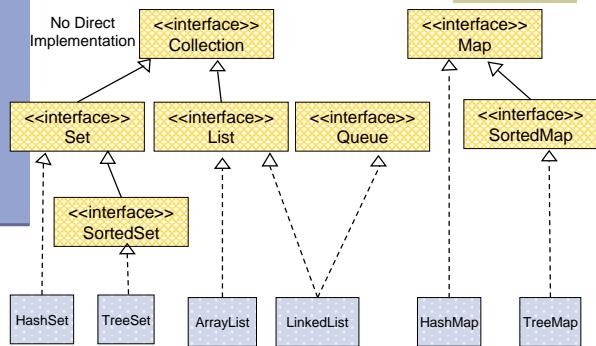
The Iterator Interface

- 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)

Command and Query

7

General Purpose Implementations



Collection Implementations

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

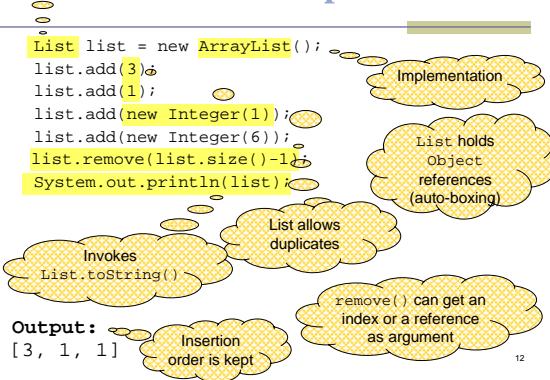
General Purpose Implementations	Data Structures			
	Hash Table	Resizable Array	Balanced Tree	Linked List
Interfaces	Set	HashSet	TreeSet (SortedSet)	
	Queue			LinkedList
	List	ArrayList		LinkedList
	Map	HashMap	TreeMap (SortedMap)	

9

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);
```

Output:
[3, 1, 1]



Best Practice

- Specify an implementation only when a collection is constructed:

- Set s = new HashSet();
- public void foo(HashSet s) {...} Works, but... Better!
- s.add() invokes HashSet.add()

polymorphism

11

Queue Example

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

Output: [1, , 6]

FIFO order

Elements are added to the tail of the queue

remove() has no argument – head is removed

14

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

13

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);
Output:
```

lexicographic order

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

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

16

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);
Output:
```

No duplicates

Unordered

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

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

15

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

18

Map Collection Views

Three views of a Map as a collection

keySet

Set

values

Collection

entrySet

Set

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

17

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

20

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

19

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);
    }
}
Arguments: A C D B
Output: [A, B, C, D]
returns a List-view of
its array argument.
lexicographic
order
```

22

Collection Algorithms

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

21

Questions ???

24

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*.

23