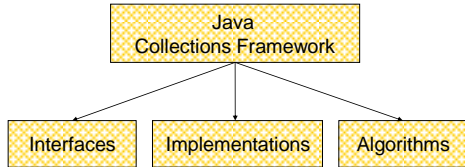


## Java Collections Framework

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

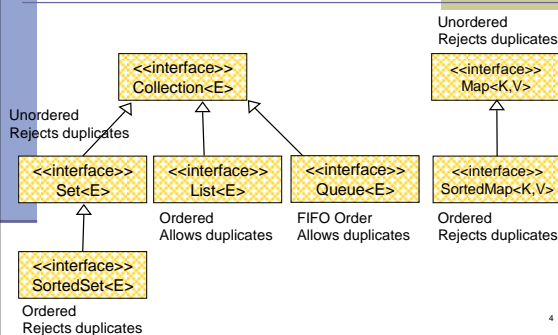


2

## תוכנה 1- מבני נתונים גנריים

תרגול 7  
מתי שמרת, ליאור שפירא

## Collection Interfaces



4

## Online Resources

- **Java 6 API Specification:**  
<http://java.sun.com/javase/6/docs/api/>
- **Sun Tutorial:**  
<http://java.sun.com/docs/books/tutorial/collections/>

3

## The Collection Interface

```
Collection<String> stringCollection = new LinkedList<String>();  
Collection<Integer> integerCollection = new LinkedList<Integer>();  
  
stringCollection.add("Hello");  
integerCollection.add(5);  
integerCollection.add(new Integer(6));  
  
stringCollection.add(7);  
integerCollection.add("world");
```

6

## The Collection Interface

- The root interface in the *collection hierarchy*
- Doesn't hold primitives
  - Use wrapper classes
- Before Java 5:
  - No type safety
  - Need to use casting
- Since Java 5:
  - Collections can be type-safe
  - i.e. the type of the elements in the collection can be specified at compile time (using generics)

5

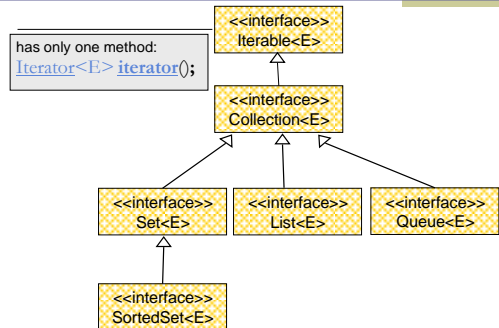
## The Iterator Interface

- 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

8

## Collection extends Iterable



7

## 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)	

10

## Iterating over a Collection

```

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

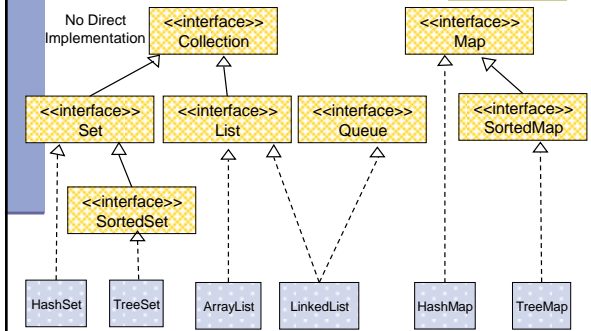
9

## Best Practice <with generics>

- Specify an element type only when a collection is instantiated:
  - `Set<String> s = new HashSet<String>();`
    - Interface
    - Implementation
- `public void foo(HashSet<String> s) {...}` **Works, but...**
- `public void foo(Set<String> s) {...}` **Better!**
- `s.add()` invokes `HashSet.add()`
  - polymorphism

12

## General Purpose Implementations



## List Example

Interface

```

List<Integer> list = new ArrayList<Integer>();
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 Integer references (auto-boxing)

List allows duplicates

remove() can get index or reference as argument

Insertion order is kept

Invokes list.toString()

Output: [3, 1, 1]

14

## Best Practice (Before Java 5.0)

Specify an implementation only when a collection is constructed:

- Set s = new HashSet();

Interface      Implementation

Works, but...

```

public void foo(HashSet s) {...}
public void foo(Set s) {...}
s.add() invokes HashSet.add()

```

Better!

polymorphism

13

## Queue Example

```

Queue<Integer> queue = new LinkedList<Integer>();
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() may have no argument - head is removed

FIFO order

Output: [1, 1, 6]

16

## Set Example

```

Set<Integer> set = new HashSet<Integer>();
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 get only reference as argument

Output: [1, 3]

Insertion order is not guaranteed

15

## SortedMap Example

```

SortedMap<String,String>map = new TreeMap<String,String>();
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

18

## Map Example

```

Map<String,String> map = new HashMap<String,String>();
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

17

## Iterating Over the Keys of a Map

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

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

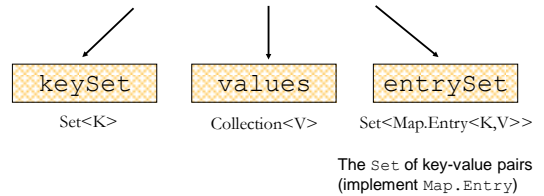
**Output:**

```
Leo
Dan
Rita
```

20

## Map Collection Views

Three views of a `Map<K, V>` as a collection



19

## Iterating Over the Key-Value Pairs of a Map

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

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

**Output:**

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

22

## Iterating Over the Keys of a Map

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

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

**Output:**

```
Leo
Dan
Rita
```

21

## Collection Algorithms

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

24

## Iterating Over the Key-Value Pairs of a Map

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

for (Map.Entry<String,String> entry: map.entrySet()) {
    System.out.println(entry.getKey() + ": " + entry.getValue());
}
```

**Output:**

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

23

## 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<String>`:

```
public interface Comparable<T> {  
    public int compareTo(T o);  
}
```
- Exception when sorting a list whose elements
  - do not implement `Comparable` or
  - are not *mutually comparable*.
- For such elements use:
  - `Collections.sort(List, Comparator)`;

26

## Sorting

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

import the package of List, Collections and Arrays

returns a List-view of its array argument.

Arguments: A C D B

Output: [A, B, C, D]

lexicographic order

25