

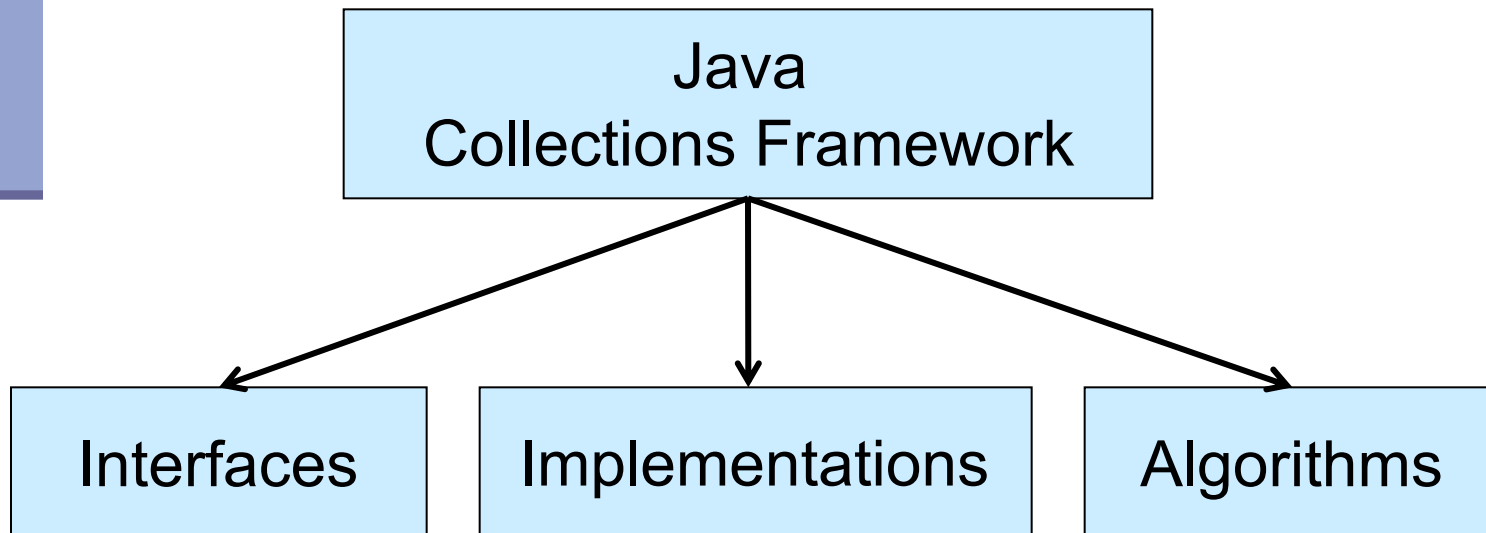
תוכנה 1

תרגול מס' 7
מבני נתונים גנריים

Java Collections Framework

- **Collection:**
a group of elements

- Interface Based Design:



Online Resources

- Java 8 API Specification of the Collections Framework:

<https://docs.oracle.com/javase/8/docs/technotes/guides/collections/reference.html>

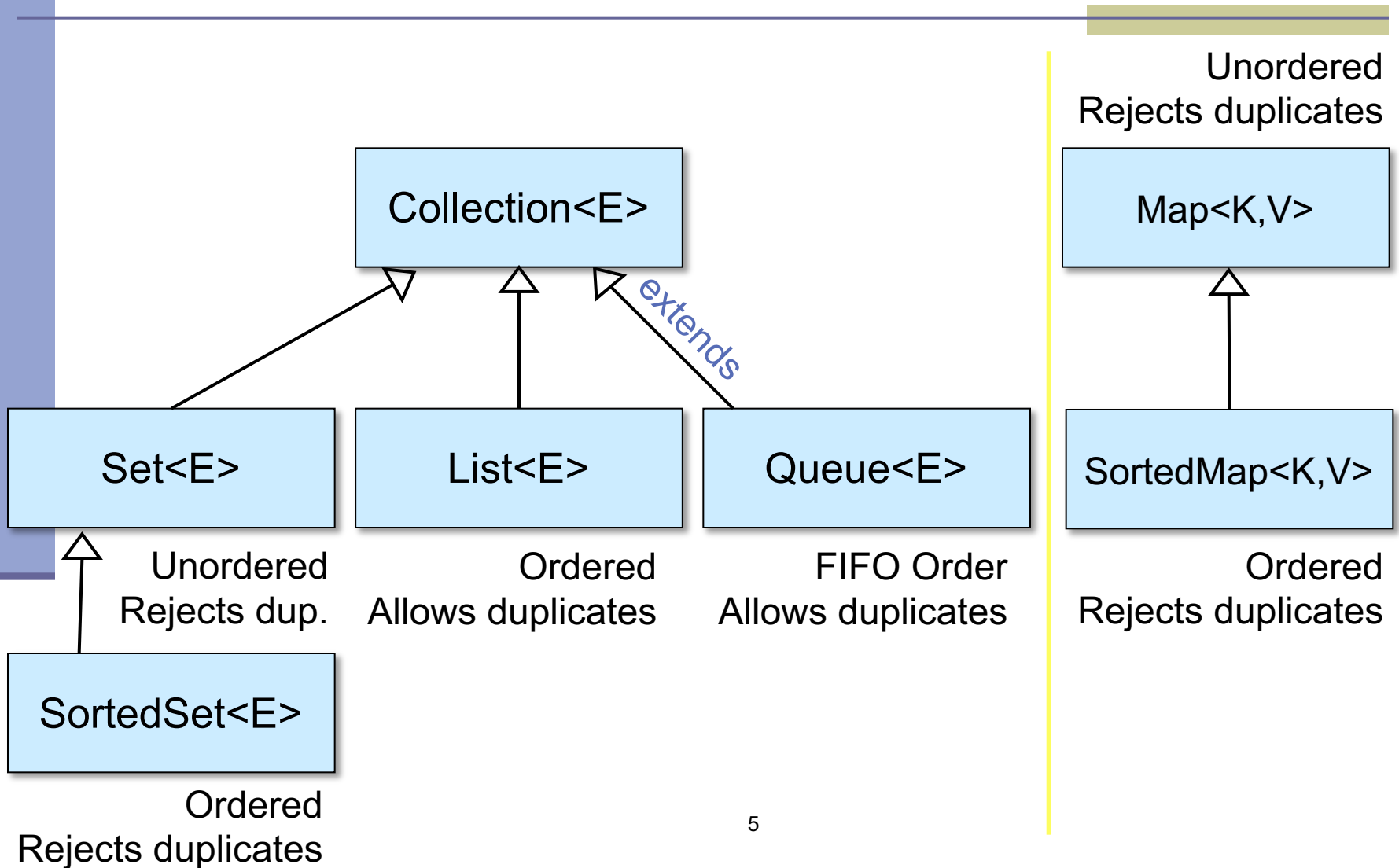
- Oracle Tutorial:

<http://docs.oracle.com/javase/tutorial/collections/>



Interfaces

Collection Interfaces



A Simple Example

```
Collection<String> stringCollection = ...
```

```
Collection<Integer> integerCollection = ...
```

- ✓ `stringCollection.add("Hello")`
- ✓ `integerCollection.add(new Integer(6));`
`integerCollection.add(5);`

Reminder- Autoboxing

- It is sometimes useful to treat primitive values as objects.
- Wrapper classes: for each of the primitive types. Boolean, Byte, Short, Character, Integer, Long, Float and Double.
 - Immutable final classes.
 - Instances hold a single primitive value.
- Boxing conversion: **primitive to wrapper object.**
- Unboxing conversion: **wrapper object to primitive.**
 - Both work explicitly (with cast operator) and implicitly.

A Simple Example

```
Collection<String> stringCollection = ...
```

```
Collection<Integer> integerCollection = ...
```

- ✓ `stringCollection.add("Hello")`
- ✓ `integerCollection.add(new Integer(6));`
- ✓ `integerCollection.add(5);`

A Simple Example

```
Collection<String> stringCollection = ...
```

```
Collection<Integer> integerCollection = ...
```

```
✓ stringCollection
```

```
✓ integerCollection
```

```
✓ integerCollection.add(5);
```

- מצביעים ל Collection של מחרוזות ושל מספרים
- Collection אינו מחזיק טיפוסים פרימיטיביים, לכן נשתמש ב Float ,Double ,Integer וכדומה
- נראה בהמשך אילו מחלקות מממשות מנשק זה

A Simple Example

```
Collection<String> stringCollection = ...
```

```
Collection<Integer> integerCollection = ...
```

- ✓ `stringCollection.add("Hello");`
- ✓ `integerCollection.add(new Integer(6));`
- ✓ `integerCollection.add(5);`

- ✗ `stringCollection.add(7);`
- ✗ `integerCollection.add("world");`
- ✗ `stringCollection = integerCollection;`



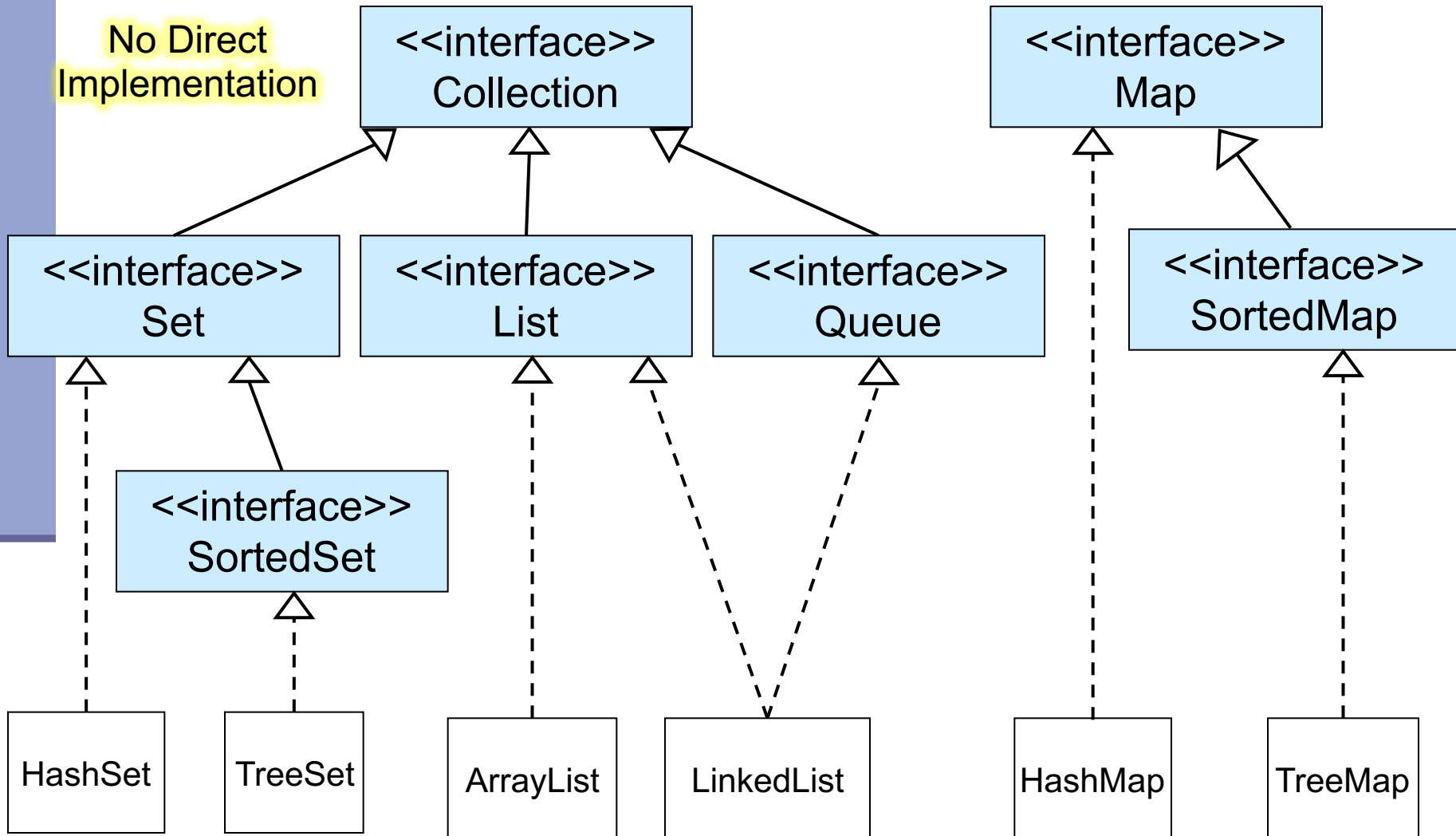
Implementations

General Purpose Implementations

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

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

Adding Implementations to the Picture



Collection interface

• מתודות שימושיות במנשק `:Collection<E>`

description	method name
ensures that this collection contains the specified element (optional operation)	Boolean add(E e)
Removes a single instance of the specified element from this collection, if it is present (optional operation).	boolean remove(Object o)
removes all elements in the Collection	void clear()
returns true if this collection contains the specified element	boolean contains(Object o)
returns true if this collection contains no elements	boolean isEmpty()
returns the number of elements in this collection.	int size()
returns an iterator over the elements in this collection.	Iterator<E> iterator()

• רשימת המתודות המלאה:

<http://docs.oracle.com/javase/8/docs/api/java/util/Collection.html>

Map interface

• מתודות שימושיות בממשק `Map<K,V>` :

description	method name
associates the specified value with the specified key in this map (optional operation)	boolean <code>put(K key, V value)</code>
removes the mapping for a key from this map if it is present (optional operation).	boolean <code>remove(Object o)</code>
removes all of the mappings from this map (optional operation).	void <code>clear()</code>
returns true if this map contains a mapping for the specified key	boolean <code>containsKey(Object key)</code>
returns true if this collection contains no elements	boolean <code>isEmpty()</code>
returns the number of key-value mappings in this map.	int <code>size()</code>
returns a set view of the keys contained in this map	<code>Set<K></code> <code>keySet()</code>
returns a Collection view of the values contained in this map.	<code>Collection<V></code> <code>values()</code>

• רשימת המתודות המלאה:

List Example

מנשק

```
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);
```

מימוש

מבוצע auto-boxing כך שנשמרים
אובייקטים מטיפוס Integer

ניתן להעביר ל remove את
האינדקס של האובייקט שאנו רוצים
למחוק, או המצביע אליו.

Output: [3, 1, 1]

שימוש במתודה
toString של ArrayList
סדר האיברים הוא לפי
סדר הכנסתם לרשימה

remove(1)?

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);
```

- Set אינו מאפשר איברים כפולים.
- שני איברים x ו y יחשבו לאיברים כפולים אם:
 1. שניהם הם null
 2. $x.equals(y) == true$ (בפרט מתקיים כאשר x ו y מצביעים לאותו האובייקט).

- `remove()` יכול לקבל רק אובייקט, כיוון שאין משמעות לאינדקס באוסף שלא שומר על סדר

Output: **[1, 3] or [3, 1]**

השתמשו ב
TreeSet או ב
LinkedHashSet
ע"מ להבטיח סדר

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);
```

Output:

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

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

LinkedHashMap Example

```
Map<String, String> map = new LinkedHashMap<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);
```

Output:

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

מסודר לפי סדר הכנסת המפתחות
(הפעם הראשונה שבה מפתח מוכנס)

Keys (names)	Values (phone numbers)
Dan	03-9516743
Rita	06-8201124
Leo	19 08-5530098

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);
```

Output:

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

סדר לקסיקוגרפי של
המפתחות

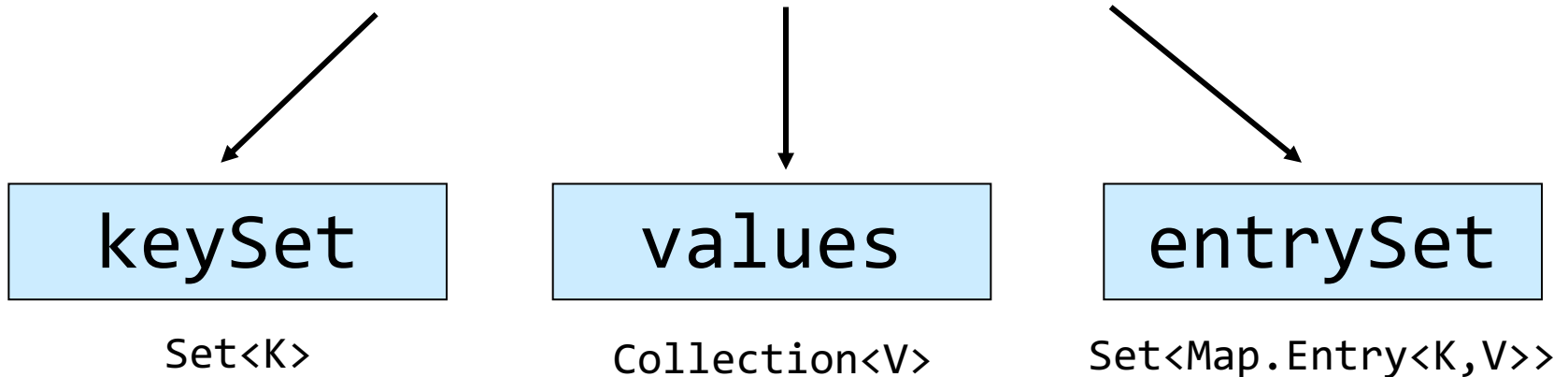
Keys (names)	Values (phone numbers)
Dan	03-9516743
Leo	08-5530098
Rita	20 06-8201124

Map Collection Views



A Map is not Iterable!

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



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

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
```



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");
```

מחזיר אובייקט מטיפוס
Set<Map.Entry<K,V>>
המכיל את כל המיפויים במילון.

```
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



Algorithms

Collection Algorithms

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

Sorting

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

Returns a list view of
the array

Arguments: A C D B

Output: [A, B, C, D]

How can we sort a list of objects?

1. Collections.sort(myList)

- myList's elements implement **Comparable<T>** interface.

```
public interface Comparable<T> {  
    /**  
     * $ret < 0 if this < other  
     * $ret = 0 if this.equals(other)  
     * $ret > 0 if this > other  
     */  
    public int compareTo(T other);  
}
```

- Error when sorting a list whose elements
 - do not implement Comparable or
 - are not mutually comparable.
- String implements the interface Comparable<String> so we are able to sort a list of strings.

How can we sort a list of objects?

1. Collections.sort(myList, myComparator)
 - myComparator implement **Comparator<T>** interface.

```
public interface Comparator<T>{
    /**
     * $ret < 0 if o1 < o2
     * $ret = 0 if o1.equals(o2)
     * $ret > 0 if o1 > o2
     */
    public int compare(T o1, T o2);
}
```

- The comparator interface enables us to sort a list of the same object by different criteria, using different comparators.

Comparable and Comparator Example

- Write the class Point that represents a point in the plane
- How to sort List<Point>?
- Two options:
 - Make Point implement Comparable<Point>, and use Collections.sort
 - Write a class that implements Comparator<Point>, and pass it as an argument to Collections.sort.
 - **Don't: write a sorting algorithm yourselves!**

```
public class Point {  
    private int x;  
    private int y;  
    ...  
}
```

- **Recommended Tutorial:**

<http://docs.oracle.com/javase/tutorial/collections/interfaces/order.html>

Implementing Comparable

```
public class Point implements Comparable<Point>{
...
public int compareTo(Point other) {
//comparison by the x axis
}      Integer.compare(this.x, other.x);
}
```

- The program:

```
List<Point> pointList = new LinkedList<Point>();
pointList.add(new Point(1, 3));
pointList.add(new Point(0, 6));
Collections.sort(pointList);
System.out.println(pointList);
```

- Output: [(0,6), (1,3)]

Writing a Comparator

```
public class YAxisPointComparator implements Comparator<Point> {  
    public int compare(Point p1, Point p2) {  
        //comparison by the y axis  
    }  
    return Integer.compare(p1.getY(), p2.getY());  
}
```

- The program:

```
List<Point> pointList = new LinkedList<Point>();  
pointList.add(new Point(1, 3));  
pointList.add(new Point(0, 6));  
Collections.sort(pointList, new YAxisPointComparator());  
System.out.println(pointList);
```

- The output: [(1,3), (0,6)]
- Useful for sorting existing classes (e.g., String)

Best Practice <with generics>

- Specify an element type only when a collection is instantiated:

```
Set<String> s = new HashSet<String>();
```

Interface Implementation

Works, but...

```
public void foo(HashSet<String> s){...}  
public void foo(Set<String> s) {...}  
s.add() invokes HashSet.add()
```

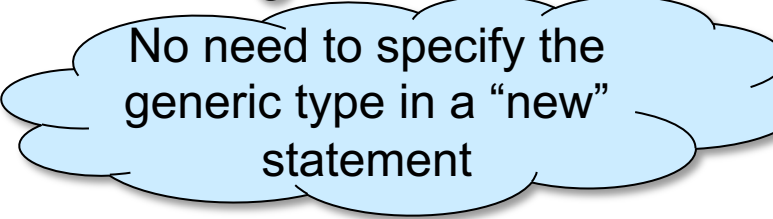
Better!

polymorphism

Diamond Notation

```
Set<String> s = new HashSet<String>();
```

```
→ Set<String> s = new HashSet<>();
```



No need to specify the generic type in a “new” statement

```
Map<String, List<String>> myMap =  
    new HashMap<String, List<String>>();
```

```
→ Map<String, List<String>> myMap = new HashMap<>();
```

Not the same as:

```
Map<String, List<String>> myMap = new HashMap();
```

(Compilation warning)

הסוף...