**JAC444 - Lecture 9**

Java Collections

Segment 5 - Algorithms

1

**Jordan Anastasiade – Java Programming Language Course**

**Comparable Types**

* Elements that can be compared to one another are called *mutually comparable*.
* To compare to objects, the class must implement the **Comparable** interface

**public interface Comparable<T> { public int compareTo(T o); }**

**int compareTo(T o)**

returns a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object

**compareTo Example**

The **compareTo** method compares the receiving object with the specified object It returns a negative integer, 0, or a positive integer depending on whether the receiving object is less than, equal to, or greater than the specified object

public class Student **implements Comparable**<Student> { private String first, last; //..other fields //equals(), hashCode(), toString() implementations  **public int compareTo(Student** **s) { int lastRes = last.compareTo(s.last); return (lastRes!=0 ? lastRes : first.compareTo(s.first)); } } last.compareTo()** invokesthe **compareTo** method of class **String**

Jordan Anastasiade – Java Programming Language Course

**Comparator Interface**

• The **Comparator** interface defines a comparison function, which imposes a total ordering on some collection of objects.

**public interface Comparator<T> { int compare(T o1, T o2); }**

**static final Comparator<Student> STUDENT\_ORDER = new Comparator<Student>() {**

**public int compare(Student s1, Student s2) { return s2.getGrade().compareTo(s1.getGrade());**

**}**

**};**

http://postitforhooney.tistory.com/entry/JavaComparatorComparable-%EC%9E%90%EB%B0%94-%EA%B0%9D%EC%B2%B4-%EB%B9%84%EA%B5%90%EB%A5%BC-%EC%9C%84%ED%95%9C-%EC%9D%B8%ED%84%B0%ED%8E%98%EC%9D%B4%EC%8A%A4-%EB%B6%84%EC%84%9D

**The SortedSet Interface**

**SortedSet** is a **Set** that maintains its elements in ascending order

**public interface SortedSet<E> extends Set<E> { // Range-view**

**SortedSet<E> subSet(E fromElement, E toElement);**

**SortedSet<E> headSet(E toElement);**

**SortedSet<E> tailSet(E fromElement);**

**// Endpoints**

**E first();**

**E last();**

**// Comparator access**

**Comparator<? super E> comparator(); }**

**The SortedMap Interface**

**SortedMap** is a **Map** that maintains its entries in ascending order, sorted according to natural ordering of its keys

**public interface SortedMap<K, V> extends Map<K, V>{ Comparator<? super K> comparator();**

**SortedMap<K, V> subMap(K fromKey, K toKey);**

**SortedMap<K, V> headMap(K toKey);**

**SortedMap<K, V> tailMap(K fromKey);**

**K firstKey();**

**K lastKey();**

**}**

**Container Operation**

* A container (collection) is an object that groups multiple elements into a single unit
* Operations with a container:
  1. Put an object in
  2. Take an object out
  3. Iterate over everything in the container (sometimes with condition)
  4. Create a container with modified elements from an initial container
  5. Information about a specific object
  6. How many objects of this type are in container
  7. Is an equivalent object in the container

**From Java 2 to Java 8**

* Java 2 had **Vector**, **Hashtable** and **Enumaration**
* Java 8 has Interfaces, Implementations, and Algorithms

|  |  |
| --- | --- |
| * Core Interfaces * **Set** * **List** * **Map** * **Queue** * **Deque** | Utility Interfaces   * **Comparator** * **Iterator**   Utility Classes   * **Collections** * **Arrays** |

* **SortedSet**
* **Sorted Map**

**The Collections Utility Class**

* **Collections** class has only static methods
* Most methods operate on **List**
* Example: **public static <T> void sort(List<T> list)**

**public static <T extends Comparable<? super T>> void sort(List<T> list)**

Sorts the specified list into ascending order, according to the natural ordering of its elements.

All elements in the list must implement the Comparable interface

**Algorithms**

**import java.util.\*; public class SortWords {**

**public static void main(String[] args) { List l = Arrays.asList(args);**

**Collections.sort(l);**

**System.out.println(l);**

**}**

**}**

**The Array Utility Class**

The **Array** utility class contains various methods for:

* manipulating arrays.
* allows arrays to be viewed as lists

•

For example:

**public static <T> List<T> asList(T array)** returns a fixed-size list from the specified array

**public static void main( String[] args ) {**

**Arrays.sort( args );**

**Sort Example using Arrays**

Sort the command line arguments in lexicographically (alphabetical order)

**import java.util.\*; public class SortExample { public static void main( String[] args ) { Arrays.sort( args );**

**List<String> list = Arrays.asList( args );**

**//use method reference Java 8 list.forEach(System.out::println);**

**}**

**}**