Università degli Studi dell'Aquila

Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica

Università degli Studi dell'Aquila

Corso di Algoritmi e Strutture Dati con Laboratorio

Sorting in Java (Comparable and Comparator)

Sorting Lists or Arrays

(As you should know by now…)

If any class implement **`Comparable`** interface in Java then a collection of these objects either List or Array can be sorted automatically by using

- **`Collections.sort()`** or
- **`Arrays.sort()`** method

The collection will be sorted based on the natural order defined by **`CompareTo`** method.

Sort a List: the Comparable interface

- If the List consists of **`String`** elements, it will be sorted into alphabetical order
- If the List consists of **`Date`** elements, it will be sorted into chronological order
- How does this happen? String and Date both implement the **`Comparable`** interface
- In the case of List contains custom objects, **`Collections.sort()`** will fail if the custom object does not implement the **`Comparable`** interface

Sort a List: the Comparable interface

- **`Comparable`** implementations provide a **natural ordering** for a class, which allows objects of that class to be sorted automatically.
- What if you want to sort some objects in an order other than their natural ordering? Or what if you want to sort some objects that don't implement **`Comparable`**?

Sort a List: the Comparator interface

- To do either of these things, we need to provide a Comparator — an object that encapsulates an ordering. Like the **`Comparable`** interface, the **`Comparator`** interface consists of a single method:

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

Comparator vs Comparable in Java

- Writing a **`compare`** method is nearly identical to writing a **`compareTo`** method, except that the former gets both objects passed in as arguments
- The **`compare`** method compares its two arguments, returning a negative integer, 0, or a positive integer depending on whether the first argument is less than, equal to, or greater than the second.
- If either of the arguments has an inappropriate type for the Comparator, the compare method throws a ClassCastException

Comparator vs Comparable in Java

- The **`Comparator`** interface is defined in **`java.util`** package
- The **`Comparable`** interface is defined in **`java.lang`** package
  - it means that **`Comparator`** should be used as an utility to sort objects whereas **`Comparable`** should be provided by default

- The logical difference between these two is that the **`Comparator`** interface in Java compares two objects provided to him, while the **`Comparable`** interface compares "this" reference with the object specified.

Example (rif. Student.java)

```
class Student implements Comparable<Student> {
  String name;
  double gpa;
      ...
 public int compareTo (Student otherStudent) {
    final double DELTA = 0.0000001;
    if (name.compareTo (otherStudent.name) < 0) return -1;
    if (name.compareTo (otherStudent.name) > 0) return 1;
    if (gpa - otherStudent.gpa > DELTA) return -1;
    if (otherStudent.gpa - gpa > DELTA) return 1;
    return 0;
  } // end-method compareTo
...
} } // end-class Student
```

Example (rif. Student_byGrade.java)

```
class Student_byGrade implements Comparator<Student> {
 /**
   * Compares the Student object s1 with the Student object s2.
   * The comparison is by grade point averages;
   * for two objects with the same gpa, the comparison is
alphabetical.
   */
  public int compare (Student s1, Student s2) {
    final double DELTA = 0.0000001;
    if (s1.gpa - s2.gpa > DELTA) return -1;
    if (s2.gpa - s1.gpa > DELTA) return 1;
    if ((s1.name).compareTo (s2.name) < 0) return -1;
    if ((s1.name).compareTo (s2.name) > 0) return 1;
    return 0;
  } // end-method compare
}  //end-class Student_byGrade
```

Example

```
List<Student> list = new LinkedList<Student>();
Student[ ] array = {…};
```

- Sort the students in alphabetical order

```
Collections.sort (list);
Arrays.sort (array);
```

- Sort the students in decreasing order of GPAs

```
Collections.sort (list, new Student_byGrade());
Arrays.sort (array, new Student_byGrade());
```