Interfaces

Comparable and Comparator Interfaces

Objective

- Comparable Interfaces
  - CompareTo
- Comparator Interface
  - Compare
  - Equals

The Comparable Interface

- Java has many predefined interfaces that are used by many classes.
- One of them is the Comparable interface, and it is used to impose an ordering upon the objects that implement it.

- The Comparable interface has only one method heading.
  - The method compareTo must be written for a class to implement the Comparable interface

```
public int compareTo(Object other);
```

The Comparable Interface

- The interface allows you to specify how
  - one object compares to another in terms of when
  - one should “come before” or “come after” or “equal” the other.

- It is the programmer’s responsibility to follow the semantics appropriately.

  - For example, if you define
    - A to come before B,
    - B to come before C

The compareTo Method:

- The compareTo method should return
  - A negative number if the calling object " comes before" the parameter other
  - A zero if the calling method "equals" the parameter other
  - And a positive number if the calling object "comes after" the parameter other.

Str2. CompareTo(Str 1)

Example Comparable

- Class Person has name and age as instance variables
- Let class person implement interface Comparable
- Compare based age
- See code Person

Fruit Example

- This example defines a Fruit class to represent apples and oranges.
- This simple class uses String to store the name of the fruit along with methods to get and set the name.
- The constructor takes the name of the fruit.

- Java doesn't know how to compare two instances of the Fruit class to each other
  - to see if one "comes after" the other when attempting to sort the array.

- The Arrays.sort method has been written with the expectation that the objects passed in the array have a compareTo method in accordance with the Comparable interface.
- Arrays.sort attempts to invoke compareTo on objects in the array
  - for example, to see if fruits[0] is greater than fruits[1]
- So they can be rearranged in sorted order.

- The solution is to make sure that the Fruit class implements the

  Comparable interface with a compareTo method.

- One way we might compare one fruit to another is to use the lexicographic ordering of the fruit name.

- Lexicographic ordering is same as alphabetical ordering when both strings are either all uppercase or all lowercase letters.
- For example, apples would come before oranges because the word “apple” is lexicographically ordered before “orange.”
- To accomplish this we can use the compareTo method defined for the String class.

- That is, if we have two strings str1 and str2 then

```
str1.compareTo(str2)
```

  - will return a negative number if str1 is lexicographically before str2,
  - the value 0 if str1 is equal to str2, and
  - a positive number if str1 is lexicographically after str2.

- The compareTo method we write for the Fruit class can then return the result of the compareTo method for the names of the fruits being compared

Fruit Code

The Comparator interface

- java.util.Comparator is an interface which is used for sorting objects in Java.
- The Comparator interface defines two methods:
  - compare( ) and
  - equals( )

The compare Method:

- The compare Method:

  int compare(Object obj1, Object obj2)

  obj1 and obj2 are the objects to be compared.

- This method returns zero if the objects are equal.
- It returns a positive value if obj1 is greater than obj2.
- Otherwise, a negative value is returned.

- By overriding compare( ), you can alter the way that objects are ordered.

Example of using Comparator interface

- We need to use Comparator interface when we want to order objects on different attributes,
  - for example, let's suppose that we would like to order students by grade or by name

- We create an array of Students, three objects of type Student and then,
- we set the name and the final grade for each one of them.
- After that, we print to the output the array without sorting, then we sort the array (a) by grade and (b) by name, and
- finally, we print to the output the respective results.
- The sorting can be done by using the method

  sort(T[] a, Comparator c) of java.util.Arrays,
- which sorts the array of Students according to the order induced by the specified comparator (either NameComparator or GradeComparator, respectively).

Output

Order of students before sorting is:
Nick 19
Helen 12
Ross 16

Order of students after sorting by student grade is
Nick 19
Ross 16
Helen 12

Order of students after sorting by student name is
Helen 12
Nick 19
Ross 16

The equals Method:

- The equals( ) method, tests whether an object equals the invoking comparator:

  boolean equals(Object obj)

  - obj is the object to be tested for equality.

  - The method returns true if obj and the invoking object are both Comparator objects and use the same ordering. Otherwise, it returns false.
  - Overriding equals( ) is unnecessary, and most simple comparators will not do so.