JAVA - HOW TO USE COMPARATOR?

Both TreeSet and TreeMap store elements in sorted order. However, it is the comparator that defines precisely what *sorted order* means.

The Comparator interface defines two methods: compare and equals. The compare method, shown here, compares two elements for order:

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. For example, to sort in reverse order, you can create a comparator that reverses the outcome of a comparison.

The equals Method:

The equals method, shown here, 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.

Example:

```
import java.util.*;

class Dog implements Comparator<Dog>, Comparable<Dog>{
   private String name;
   private int age;
   Dog(){
   }

   Dog(String n, int a){
      name = n;
      age = a;
   }

   public String getDogName(){
      return name;
   }

   public int getDogAge(){
      return age;
   }

   // Overriding the compareTo method
   public int compareTo(Dog d){
      return (this.name).compareTo(d.name);
   }

   // Overriding the compare method to sort the age
   public int compare(Dog d, Dog d1){
      return d.age - d1.age;
   }
}
```

```
public class Example{

   public static void main(String args[]){
      // Takes a list o Dog objects
      List<Dog> list = new ArrayList<Dog>();

      list.add(new Dog("Shaggy",3));
      list.add(new Dog("Lacy",2));
      list.add(new Dog("Roger",10));
      list.add(new Dog("Tommy",4));
      list.add(new Dog("Tammy",1));
      Collections.sort(list);// Sorts the array list

      for(Dog a: list)//printing the sorted list of names
         System.out.print(a.getDogName() + ", ");

      // Sorts the array list using comparator
      Collections.sort(list, new Dog());
      System.out.println(" ");
      for(Dog a: list)//printing the sorted list of ages
         System.out.print(a.getDogName() +"  : "+
   a.getDogAge() + ", ");
   }
}
```

This would produce the following result:

```
Lacy, Roger, Shaggy, Tammy, Tommy,
Tammy  : 1, Lacy  : 2, Shaggy  : 3, Tommy  : 4, Roger  : 10,
```

Note: Sorting of the Arrays class is as the same as the Collections.