Comparable and Comparator Interfaces

There's very little that's comparable to seeing the spark in a student's face when she gets something that she's been struggling with.
- Alexi Zentner

Comparing Java Objects

- The `Object` class provides a way to test equality of two objects:

```
public boolean equals(Object obj)
```

- But this method just compares the memory of the objects, not their content
- The `Object` class does not provide any methods for comparing objects
  - There is no way to test whether one object is "less" than or "greater" than another object

Comparable and Comparator Interfaces

- Instead, Java provides two interfaces that provide methods for determining the ordering of objects
  - The `Comparable` interface in `java.lang` library
  - The `Comparator` interface in `java.util` library

The Comparable Interface

- Implemented by a class of objects you want to compare (i.e. Students, Rectangles, Aliens, etc.)
- The interface requires one method:

```
    public int compareTo(Object o)
```

- The `compareTo` method must return
  - A negative number if the calling object "comes before" the parameter
  - A zero if the calling object "equals" the parameter other
  - A positive number if the calling object "comes after" the parameter other

The Comparable Interface

- Notice that the parameter is an *`Object`*.
- In order to implement this interface, our parameter must also be an *`Object`*, even if that's not what we want.
- When implementing the `compareTo` method, should use casting to ensure the parameter is of the correct class.

Example Using Student Class

```
public class Student implements Comparable
{
   public Student(String name, int score) {...}

   public int compareTo(Object o) {...}

   public String getName() {. . . }
   public int getScore() { . . . }
   public void setName(String name) {. . . }
   public void setScore(int score) {. . .}

  // other methods
   . . .
}
```

Constructor for Student

- Nothing special here:

```
public Student(String name, int score)
{
        this.name = name;
        this.score = score;
}
```

- Sort students according to score

```
public int compareTo(Object o)
{
        return score - ((Student)o).score;
}
```

Using Student Class, Ver. 1

```
public static void main(String args[])
{
     TreeSet<Student> set = new TreeSet<Student>();

     set.add(new Student("Ann", 87));
     set.add(new Student("Bob", 83));
     set.add(new Student("Cat", 99));
     set.add(new Student("Dan", 25));
     set.add(new Student("Eve", 76));

     Iterator<Student> itr = set.iterator();
     while (itr.hasNext())
      {
           Student s = itr.next();
           System.out.println(s.name + "  " + s.score);
      }
 }
```

Output of Program

- Using an iterator, print out the values in order and get the following result:

```
Dan   25
Eve   76
Bob   83
Ann   87
Cat   99
```

- How did the iterator know that it should sort `Students` by score, rather than by name?

Using a Separate `Comparator`

- Above, `Student` implemented the `Comparable` interface
  - Uses `compareTo` method to make comparisons
  - So, can sort students *only* by their score
  - If we wanted to sort students another way, such as by name, we are out of luck
- To make comparison using other criteria, can use *separate class* that implements the `Comparator` interface
  - This is more flexible, but also clumsier
  - The Comparator requires the `compare` method

```
public int compare(T obj1, T obj2)
```

Outline of StudentComparator

- Because of generics, our compare method can take Student arguments:

```
import java.util.*;


public class StudentComparator
          implements Comparator<Student>
{

     public int compare(Student s1, Student s2)
{...}

}
```

The `compare` Method

```
public int compare(Student  s1, Student s2)
{
        return s1.score – s2.score;
}
```

- How different from compareTo(Object o):
  - Different method name
  - It takes both objects as parameters, not just one
  - We have to either use generics, or check the type of both objects
  - If our parameters are Objects, they have to be cast to Students

Using Student Class, Ver. 2

```
public static void main(String args[])
{
     Comparator<Student> comp = new StudentComparator();
     TreeSet<Student> set = new TreeSet<Student>(comp);

     set.add(new Student("Ann", 87));
     set.add(new Student("Bob", 83));
     set.add(new Student("Cat", 99));
     set.add(new Student("Dan", 25));
     set.add(new Student("Eve", 76));

     Iterator<Student> itr = set.iterator();
     while (itr.hasNext())
      {
           Student s = itr.next();
           System.out.println(s.name + "   " + s.score);
      }
 }
```

Output of Program

- Using an iterator, print out the values in order and get the same result:

```
Dan   25
Eve   76
Bob   83
Ann   87
Cat   99
```

- What if want to sort the students by name instead of scores?

Another Comparator

```
public class StudentByNameComparator
         implements Comparator<Student>
{

     public int compare(Student s1, Student s2)
     {
         return s1.getName.compareTo(s2.getName);
     }
}
```

When to Use Each

- The `Comparable` interface is simpler and less work
  - Your class implements the `Comparable` interface
  - It provides a `compareTo` method
  - Uses the same comparison method every time
- The `Comparator` interface is more flexible but slightly more work
  - Create as many different classes that implement `Comparator` as you like
  - You can sort using the objects using the comparator you want
  - For example, sort `Students` by score *or* by name