The Comparable Interface

- one of the two most used interfaces in Java (along with Iterable)

- indicates that instances of the type can be ordered

The Comparable interface takes a generic type that indicates the type this object will be compared to.

The Comparable interface requires a compareTo method that takes a parameter of the generic type.

It should return < 0 if this object comes before the parameter in the default ordering of the type.

It should return > 0 if this object comes after the parameter in the default ordering of the type.

It should return = 0 if the two objects are equivalent in the default ordering of the type.

Our first example was to make the Employee class Comparable.

We decided that the default ordering would be to order employee's by their employee number

So, we made Employee implement Comparable:

public class Employee implements Comparable<Employee> {

Note that we specified the generic of Employee to state that we must compare Employee's to other Employee's.

Now, we override the compareTo method. The class decided that the default comparison of employee's should be by employee name.

Note that String is Comparable so we can use it (as long as we are happy with how String orders its instances):

public int compareTo(Employee e) {

return this.getName().compareTo(e.getName());

}

Version 2: We don't like String's ordering. In particular, we would like the names ordered alphabetically and case-insensitive.

The full code is in the Employee class.

public int compareTo(Employee e) {

for (int i = 0; i < this.getName().length() && i < e.getName().length(); i++) {

// compare the two characters at index i, if they are the same, keep going, if they are different, return the appropriate value

}

// when loop stops, either the two strings are equal or one is the prefix of the other, so return the appropraite value

}

Using Comparable objects.

Where we would like to write "if (e1 < e2)" we instead write

if (e1.compareTo(e2) < 0)

Note that the < operator is the same, we just moved its location. This is the reason the return value for the compareTo is specified the way it is.

Java provides an Arrays class that has useful methods on arrays. One of the methods is sort. By making Employee Comparable, we can sort an array of Employees.

Employee e1 = new Employee("10", "Smith");

Employee e2 = new Employee("30", "Ang");

Employee e3 = new HourlyEmployee("20", "Orr");

Arrays.sort(employees);

Now the elements of employees are sorted by employee name so e2 is first, e3 is second, and e1 is third.

How does it know how to sort the employee's? Sort expects the array to contain only Comparable objects, and it uses the compareTo method.