Arrays of Objects, the Comparable Interface and Sorting

Though we've learned to sort arrays step-by-step, when one wants to save time, Java has sorting methods pre-written that work very quickly. These are simple to use when sorting an array of primitive values, since the ordering of primitive values is well-defined.

But what if you wanted to sort a set of objects?

How would the computer know which of two contacts "comes first"?

The answer to this question is that the programmer would have to define how to compare objects.

In Java, an interface is a list of methods that a class should have, if it is implementing the interface. The Comparable interface in Java requires that the following method be written:

```
public int compareTo(<T> other);
```

where <T> represents the class in which the method resides.

This method should be written as follows:

If the current object(this) comes before other, then a negative integer should be returned.
If this and other are equal, then 0 should be returned.
If this comes after other, a positive integer should be returned.

Anytime you write a class, you need to decide if you want that class to implement the Comparable interface. If you do, then you have to decide how objects in the class are "ordered", so to speak.

Let's say for Contact objects, we want to make an alphabetical comparison by name. If the names are the same, we can break the tie by age, with younger people going first. If that's the same, we will break the tie by phone number, in numerical order. (We assume that if all three of these are the same, that the Contacts are the same.)

Here are the changes, noted in bold, we need to make in the Contact class in order to implement the Comparable interface:

```
public class Contact implements Comparable<Contact> {

    private String name; // Stores name of Contact
    private int age;  // Stores age of Contact
    private int phonenumber; // Stores phone number of contact
    private int bday;  // Stores birthday in an int

    // Creates Contact object based on parameters.
    public Contact(String n, int a, int p, int month, int day) {
          name = n;
          age = a;
          phonenumber = p;
          bday = 100*month + day;
    }

    public int compareTo(Contact other) {

        // Compare the names.
        int strcomp = this.name.compareTo(other.name);

        // If this breaks the tie, return accordingly.
        if (strcomp != 0)
            return strcomp;

        // Find age difference.
        int agediff = this.age - other.age;

        // Different ages, so we can return this number.
        if (agediff != 0)
            return agediff;

        // Last criterion. The difference is valid to return.
        return this.phonenumber - other.phonenumber;
    }
}
```

We indicate we're implementing an interface in the class definition. The <Contact> indicates that the compareTo method will be taking in a Contact object. (Note: If we don't put this part in, then it is assumed that the method takes in an Object object. So, for our purposes, in those brackets, always put in the name of the class that is implementing Comparable.)

The compareTo method is like any other instance method. There are no different syntax rules except that the method prototype must match what is shown above. (Change this accordingly for other classes.) From there, the programmer is free to define the comparison between the current object (this) and other. Note that since other is just a formal parameter, you may give it any name you want. In this example, the name other is just arbitrary.

Sorting an Array of Objects that implement Comparable

Now, if we have an array of Contact objects already filled up:

```
Contact[] friends = new Contact[10];
// Fill in array with all 10 contacts.
```

We can sort the array as follows:

```
Arrays.sort(friends);
```

A couple notes about using this sorting method:

1) The array must be filled up. None of the references in the array may be null.
2) You must import java.util.Arrays. More commonly, one will import java.util.*;

Once this method is called, the array friends will be sorted based on the compareTo method in the Contact class.

Note: If you are attempting to sort an ArrayList of objects, instead of an array, the following method must be used:

```
Collections.sort(thisarraylist);
```

This method will work for any type of java list.