Chapter 4: Sorted Lists

Sorted Lists

In Lesson 10, we discussed several types of collections often used in computer applications. In the rest of this lesson, we'll look at how we can use some of these collections. The first one we'll check out is the sorted list. It's not much different from the list, with one exception, as its name indicates: The objects we put into the list are stored in sequence, however we define the sequence.

Let me explain that last statement. Think about the application we just built, which displays basketball player information. In order to *sort* the players, Java has to be able to compare two players and decide which is "less than" or "greater than" the other so it can decide which one goes into the list first.

That said, you would have a problem if I asked you to sort the player list, wouldn't you? Your first question would be, "How do you want it sorted? By name? By number? By points per game?" In order to sort it, Java needs a way to compare the items in it.

Well, never fear: Java gives us a way to do that in its *Comparable* interface. If we want to sort players, we just have to implement Comparable in the Player class, which is pretty simple. We only need to make two changes to the class.

First, we'll add the phrase "implements Comparable<Player>" to the first line of the Player class. That phrase does two things for us:

1. It tells the compiler what type of objects we will compare. You may think that goes without saying, but with computers, we have to be explicit. The object type goes inside angle brackets at the end of the line. In our case, we want to compare Players to other Players.

2. It allows the compiler to make sure that the class actually implements all the methods required by the interface. In this case, there is only one method to implement: *compareTo()*.

Second, we'll add the compareTo() method to the Player class. There are two requirements for the method:

1. There must be one parameter whose type matches the type we specified in the implements phrase at the top of the class.

2. The method must return an int value, calculated like this:

    - If this object is less than the object received in the parameter (by whatever method we decide to use to determine that), the method must return a value less than zero.

    - If this object is equal to the object received in the parameter, the method must return zero.

    - If this object is greater than the object received in the parameter, the method must return a value greater than zero.

Here's what the additions look like for a list sorted by player number:

```
public class Player implements Comparable<Player>
{
 . . .
    public int compareTo(Player p)
    {
        if (number > p.number)
            return 1;
        else if (number < p.number)
            return -1;
        else
            return 0;
    }
}
```

If we wanted to sort by some other criteria, like player name or points per game, we just need to change the comparisons in the compareTo() method.

Once we've made the Player class comparable, using a sorted list in our application is easy! All we have to do is add one line to our program. We'll add it to our OpenMenuItemListener inner class, where we open the file and build the list in the first place. Right after we build the list, but before we create the iterator, we'll add this line:

```
Collections.sort(list);
```

In case you're wondering what the *Collections* class is, it's a very useful class. It contains a number of methods that operate on collections, including the sort() method we used above. For example, it offers a reverse() method that reverses the order of a list, min() and max() methods that find the smallest and largest elements in a collection (if the elements are comparable), a copy() method that makes a complete copy of a list, and more. If you're curious, you can check it out in the Java API.

If you're not sure where to put the last line of code we talked about, you can look at an updated version of the class by clicking this link:

Updated OpenMenuItemListener Class

© 2022 Cengage Learning, Inc. All Rights Reserved