Special Topic 14.5

The Comparator Interface

Sometimes, you want so sort an array or array list of objects, but the objects don't belong to a class that implements the Comparable interface. Or, perhaps, you want to sort the array in a different order. For example, you may want to sort coins by name rather than by value.

You wouldn't want to change the implementation of a class just in order to call Arrays.sort. Fortunately, there is an alternative. One version of the Arrays.sort method does not require that the objects belong to classes that implement the Comparable interface. Instead, you can supply arbitrary objects. However, you must also provide a *comparator* object whose job is to compare objects. The comparator object must belong to a class that implements the Comparator interface. That interface has a single method, compare, which compares two objects.

As of Java version 5, the Comparator interface is a parameterized type. The type parameter specifies the type of the compare parameters. For example, Comparator<Coin> looks like this:

```
public interface Comparator<Coin>
{
   int compare(Coin a, Coin b);
}
```

The call

```
comp.compare(a, b)
```

must return a negative number if a should come before b, 0 if a and b are the same, and a positive number otherwise. (Here, comp is an object of a class that implements Comparator<Coin>.)

For example, here is a Comparator class for coins:

```
public class CoinComparator implements Comparator<Coin>
{
   public int compare(Coin a, Coin b)
   {
      if (a.getValue() < b.getValue()) return -1;
      if (a.getValue() == b.getValue()) return 0;
      return 1;
   }
}
```

To sort an array of coins by value, call

```
Arrays.sort(coins, new CoinComparator());
```