Writing your own Comparators

Sorting is a fundamental problem in computer science. In many applications, it is necessary to sort some data. When we design and implement our programs, three questions come to mind when we need to sort.

1. What sorting algorithm should we use?
2. Should we sort in ascending or descending order?
3. Exactly what data are we sorting on?

Fortunately, when using a high-level language such as C, Java or Python, the choice of sorting algorithm is easy. There is already a built-in sorting function in the run-time library. Often, this is some fancy implementation of Quick sort or merge sort.

Regarding the second question, this is also pretty easy. There are just two choices. If you happen to get this one wrong, it's likely that all you need to do is swap a > for a < somewhere in your code.

However, the third question is really important when the data you want to sort is an array or list of objects, and these objects have several attributes. If you had an array of integers or an array of strings, it's pretty obvious what is being sorted! Your favorite programming language is already smart enough to know how to sort simple data like numbers and strings. But suppose you had a list of planet objects. Each planet might have four attributes: name, number of moons, distance from the sun, and mass. You could conceivably desire to sort your planets on any one of these attributes.

So, the purpose of this exercise is to write a program that will sort an array or list of objects. To make it possible to perform such a sort, you will have to tell the computer how to compare two objects of the same type. We do this by means of writing a class (in Java) or function (in Python) called a *comparator*.

Writing a comparator entails writing a function that takes two objects of the same type. Suppose we call these two objects obj1 and obj2. The comparator function will return an integer. There are three possibilities.

- If obj1 should appear before obj2 in a sorted list, then we return a negative number.
- If obj1 should appear after obj2 in a sorted list, then we return a positive number.
- If we don't care which object comes first (because they might be equal), then we return zero.

By "negative number" and "positive number", it does not matter exactly which integer you return. By convention, we would return –1 or +1, or you may decide to "subtract" the object values, if that makes sense. (Note that the way that comparators are defined implies that by default we would expect lists to be sorted in ascending order.)

After you implement your comparator class/function, it is time to make use of it. In your main program, you would call the built-in sort function. It accepts two parameters: the list or array to sort, and the comparator function.

To illustrate, suppose we had the following data about countries. Let's see how we would allow a program to sort on any of the three columns (attributes).

Name	Dialing code	Population (millions)
India	91	1148
China	86	1321
New Zealand	64	5
France	33	65
Mexico	52	110

Java implementation of example

We would create a class called Country, with attributes name, code and population. Then we create three comparator classes. It is a good idea for the name of the class to clearly identify which attribute is being compared (e.g. PopulationComparator). In a large program, it is even better for the name of the class to also identify the type of objects being compared (e.g. CountryPopulationComparator). Here are the comparator implementations.

```
// Rank by name of country
import java.util.Comparator;

public class NameComparator implements Comparator
{
  public int compare(Object o1, Object o2)
  {
    Country c1 = (Country) o1;
    Country c2 = (Country) o2;

    String name1 = c1.getName();
    String name2 = c2.getName();

    // We want to sort in alphabetical order
    // We use the compareTo method of the String class,
    // which is analogous to the compare method that
    // we are defining here.
    if (name1.compareTo(name2) < 0)
      return -1;
    else if (name1.compareTo(name2) > 0)
      return 1;
    else
      return 0;
  }
}
```

Writing a comparator class is much easier than it looks. They all look pretty much the same. Here are the other two:

```
// Rank by dialing code
import java.util.Comparator;

public class CodeComparator implements Comparator
{
  public int compare(Object o1, Object o2)
  {
    Country c1 = (Country) o1;
    Country c2 = (Country) o2;

    int code1 = c1.getCode();
    int code2 = c2.getCode();

    // We want to sort in ascending order
    if (code1 < code2)
      return -1;
    else if (code1 > code2)
      return 1;
    else
      return 0;
  }
}
```

```
// Rank by population
import java.util.Comparator;

public class PopulationComparator implements Comparator
{
  public int compare(Object o1, Object o2)
  {
    Country c1 = (Country) o1;
    Country c2 = (Country) o2;

    int pop1 = c1.getPopulation();
    int pop2 = c2.getPopulation();

    // We want to sort in descending order
    if (pop1 > pop2)
      return -1;
    else if (pop1 < pop2)
      return 1;
    else
      return 0;
  }
}
```

Finally, here is the fragment of the main program that will arrange for all the sorting to be done:

```
System.out.printf("\nHere is the list of countries, sorted by name.\n");
Arrays.sort(a, new NameComparator());
printArray(a);

System.out.printf("\nHere is the list of countries, sorted by dialing code.\n");
Arrays.sort(a, new CodeComparator());
printArray(a);

System.out.printf("\nHere is the list of countries, sorted by population.\n");
Arrays.sort(a, new PopulationComparator());
printArray(a);
```

Here is the output when we run.

```
Here is the list of countries, sorted by name.
China              86 1321
France             33   65
India              91 1148
Mexico             52  110
New Zealand        64    5

Here is the list of countries, sorted by dialing code.
France             33   65
Mexico             52  110
New Zealand        64    5
China              86 1321
India              91 1148

Here is the list of countries, sorted by population.
China              86 1321
India              91 1148
Mexico             52  110
France             33   65
New Zealand        64    5
```

Question: How can you tell that one comparator will allow the sorting to proceed in ascending order, but the other in descending order? In other words, where in the implementation is this specified?

Python implementation of example

This is analogous to the Java implementation.

```
# compare.py
# We can compare items in a list just like in Java:
#   using a comparator function.
# Suppose we had a list of countries by population & telephone code.
# Name, dialling code, population in millions.

list = [["India", 91, 1148],
        ["China", 86, 1321],
        ["New Zealand", 64, 5],
        ["France", 33, 65],
        ["Mexico", 52, 110]]

# Define our sorting functions.
def byAlpha(countryA, countryB):
    if countryA[0] < countryB[0]:
        return -1
    elif countryA[0] > countryB[0]:
        return 1
    else:
        return 0

def byCode(countryA, countryB):
    return countryA[1] - countryB[1]

def byPop(countryA, countryB):
    return countryA[2] - countryB[2]

list.sort(byAlpha)
print list

list.sort(byCode)
print list

list.sort(byPop)
print list
```

Question: How would you write a comparator to sort on *two* criteria instead of one? For example, suppose we had a Person type, and it has attributes of first and last name. How should we teach the computer how to compare names so that they are correctly alphabetized? Hint: some people have the same last name.

Here is your exercise.

The file eastern.txt contains the standings of teams in the Eastern Conference of the NHL for a recent season. After each team's name is its record. You will see three numbers: wins, losses (in regulation), and overtime losses. In ranking teams, the NHL uses the following point system. Each win is worth 2 points, and each overtime loss is worth 1 point.

Using the above implementation(s) as a guide, write a program that will read this input file, and then calculate each team's number of points. Then, write three comparators:

- Alphabetical by name
- Descending order by number of wins
- Descending order by points

Sort the list three times, once using each comparator, and print each sorting of the list of teams.

When you print the list of teams, be sure that the names are left justified, and all the numbers are right justified. For example, in Java, this can be arranged in the toString() method of the Team class.