Ways to sort lists of objects in Java based on multiple fields

by Benny Bottema - Friday, June 21, 2013

http://www.bennybottema.com/2013/06/21/ways-to-sort-lists-of-objects-in-java-based-on-multiple-fields/

As I was again trying to remember how to sort a list of objects in Java bases on multiple keys, I had the luminous idea to finally write it down.

We have a list of pizza's, and I want them sorted according to size, number of toppings and furthermore by name. This means that there will be groups ordered by size and within those groups the pizza's are ordered into groups by number of toppings and in those groups the pizza's are ordered by name.

We want to end up with a list like this:

1. **Pizza's 34cm:**
2. Anchovy (34cm, tomato, cheese, Anchovies)
3. Prosciutto (34cm, tomato, cheese and ham)
4. Chicken Special (34cm, tomato, cheese, chicken and turkey pieces)
5. Vulcano (34cm, tomato, cheese, mushrooms and ham)
6. Peperone (34cm, tomato, cheese, mushrooms, ham, capsicum, chili peppers and onions)
7. **Pizza's 30cm:**
8. Anchovy (30cm, tomato, cheese, Anchovies)
9. Prosciutto (30cm, tomato, cheese and ham)
10. Chicken Special (30cm, tomato, cheese, chicken and turkey pieces)
11. Vulcano (30cm, tomato, cheese, mushrooms and ham)
12. Peperone (30cm, tomato, cheese, mushrooms, ham, capsicum, chili peppers and onions)
13. **Pizza's 26cm:**
14. Anchovy (26cm, tomato, cheese, Anchovies)
15. Prosciutto (26cm, tomato, cheese and ham)
16. Chicken Special (26cm, tomato, cheese, chicken and turkey pieces)
17. Vulcano (26cm, tomato, cheese, mushrooms and ham)
18. Peperone (26cm, tomato, cheese, mushrooms, ham, capsicum, chili peppers and onions)

You can find working code on this gist.

Contents

Messy and convoluted: Sorting by hand

```
Collections.sort(pizzas, new Comparator<Pizza>() {
    @Override
    public int compare(Pizza p1, Pizza p2) {
        int sizeCmp = p1.size.compareTo(p2.size);
        if (sizeCmp != 0) {
            return sizeCmp;
        }
        int nrOfToppingsCmp = p1.nrOfToppings.compareTo(p2.nrOfTopping
s);
        if (nrOfToppingsCmp != 0) {
            return nrOfToppingsCmp;
        }
        return p1.name.compareTo(p2.name);
    }
});
```

This requires a lot of typing, maintenance and is error prone.

The reflective way: Sorting with BeanComparator

```
ComparatorChain chain = new ComparatorChain(Arrays.asList(
   new BeanComparator("size"),
   new BeanComparator("nrOfToppings"),
   new BeanComparator("name")));

Collections.sort(pizzas, chain);
```

Obviously this is is more concise, but even more error prone as you loose your direct reference to the fields by using Strings instead. Now if a field is renamed, the compiler won’t even report a problem. Moreover, because this solution uses reflection, the sorting is much slower.

Getting there: Sorting with Google Guava’s ComparisonChain

```
Collections.sort(pizzas, new Comparator<Pizza>() {
    @Override
    public int compare(Pizza p1, Pizza p2) {
        return ComparisonChain.start().compare(p1.size, p2.size).compa
re(p1.nrOfToppings, p2.nrOfToppings).compare(p1.name, p2.name).result(
);
```

```
        // or in case the fields can be null:
        /*
          return ComparisonChain.start()
              .compare(p1.size, p2.size, Ordering.natural().nullsLast(
))
              .compare(p1.nrOfToppings, p2.nrOfToppings, Ordering.natu
ral().nullsLast())
              .compare(p1.name, p2.name, Ordering.natural().nullsLast(
))
              .result();
         */
    }
});
```

This is much better, but requires some boiler plate code for the most common use case: *null-values should be values less by default*. For null-fields, you have to provide an extra directive to Guava what to do in that case. This is a flexible mechanism if you want to do something specific, but often you want the default case (ie. 1, a, b, z, null).

Sorting with Apache Commons CompareToBuilder

```
Collections.sort(pizzas, new Comparator<Pizza>() {
    @Override
    public int compare(Pizza p1, Pizza p2) {
        return new CompareToBuilder().append(p1.size, p2.size).append(
p1.nrOfToppings, p2.nrOfToppings).append(p1.name, p2.name).toCompariso
n();
    }
});
```

Like Guava's ComparisonChain, this library class sorts easily on multiple fields, but also defines default behavior for null values (ie. 1, a, b, z, null). However, you can't specify anything else either, unless you provide your own Comparator.

Ultimately it comes down to flavor and the need for flexibility (Guava's ComparisonChain) vs. concise code (Apache's CompareToBuilder).