Exercises: Lambda Expressions Part 1 (Page 1 of 2)

Problem 1 is easy. Once you get used to lambda syntax, the first three pieces of problem 1 can be solved in one line each. Problems 2 and 3 are hard; significantly more so than most of the exercise problems. It will take some time re-reading the problems just to understand what I am asking for, much less to do it.

Notes on the sorting problems:

- The compare method of Comparator should return a negative number if the first entry is "less" than the second, a positive number if the first entry is "greater" than the second, and 0 if they are the same. For details, see http://docs.oracle.com/javase/8/docs/api/java/util/Comparator.html.
- To print out an array after sorting, do System.out.println(Arrays.asList(yourArray))
  The point of this is that if you just print an array directly, you do not see anything useful (just the memory address), but if you print a List, it shows the individual elements separated by commas. So, the above trick is simpler than making a loop to traverse the array and print out the elements. Note for future reference that this trick only works if yourArray is an Object array (e.g., String[] or Integer[]); this trick fails if yourArray is an array of primitives (e.g., int[]).

1. **Basic lambdas.** Make an array containing a few Strings. Sort it by

- length (i.e., shortest to longest)
  (Hint: this exact solution was shown in the lecture)
- reverse length (i.e., longest to shortest)
  (Hint: minor variation of the first bullet)
- alphabetically by the first character only
  (Hint: charAt(0) returns the numeric code for the first character)
- Strings that contain "e" first, everything else second. For now, put the code directly in the lambda.
  (Hint: remember that the body of a lambda is allowed to have curly braces and a return statement. See the first two lambda examples in the notes.)
- Redo the previous problem, but use a static helper method so that your lambda looks like this:

```
Arrays.sort(words, (s1,s2) -> Utils.yourMethod(s1,s2))
```

Exercises: Lambda Expressions Part 1 (Page 2 of 2)

2. **Making your own interfaces for which lambdas can be used.** Your eventual goal is to make a method called betterString that takes two Strings and a lambda that says whether the first of the two is "better". The method should return that better String; i.e., if the function given by the lambda returns true, the betterString method should return the first String, otherwise betterString should return the second String. Here are two examples of how your code should work when it is finished (the first lambda example returns whichever of string1 and string2 is longer, and the second lambda example always returns string1).

- String string1 = ...;
- String string2 = ...;
- String longer = StringUtils.betterString(string1, string2, (s1, s2) -> s1.length() > s2.length());
- String first = StringUtils.betterString(string1, string2, (s1, s2) -> true);

Accomplishing all of this requires you to do three things:

- Define the TwoStringPredicate interface. It will specify a method that takes 2 strings and returns a boolean. *This interface is normal Java 7 code.*
- Define the static method betterString. That method will take 2 strings and an instance of your interface. It returns string1 if the method in interface returns true, string2 otherwise. *This method is normal Java 7 code.*
- Call betterString. You can now use lambdas for the 3rd argument, as in the examples above.

3. **Making generically-typed interfaces for which lambdas can be used.** Use generics to replace your String-specific solutions to problem 3 with generically typed solutions. That is, replace betterString with betterEntry and TwoStringPredicate with TwoElementPredicate. Make sure your previous examples still work when you only change betterString to betterElement. But, now you should also be able to supply two Cars and a Car predicate, two Employees and an Employee predicate, etc. For example:

- ElementUtils.betterElement(string1, string2, (s1, s2) -> s1.length() > s2.length())
- ElementUtils.betterElement(car1, car2, (c1, c2) -> c1.getPrice() > c2.getPrice())
- ElementUtils.betterElement(employee1, employee2, (e1, e2) -> e1.getSalary() > e2.getSalary())