Lambda Expressions: Part 1 (Page 1 of 2)

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. See the JavaDoc API for details.
- 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.

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
  (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.)

2. **Method references.** Your solutions to problem 1 look something like this:

- Bullet 1 Arrays.sort(words, (s1,s2) -> someValue);
- Bullet 2 Arrays.sort(words, (s1,s2) -> someValue);
- Bullet 3 Arrays.sort(words, (s1,s2) -> someValue);
- Bullet 4
```
Arrays.sort(words, (s1,s2) -> { some code;
                                some more code;
                                even more code;
                               return(someValue); }
```

For that last sorting example (strings with “e” first), move the logic that computes the number to a separate static method. For example, StringUtils.eChecker(s1, s2) will return -1 if s1 is “less” (i.e., it contains “e” but s2 doesn’t), 1 if s1 is “greater”, and 0 otherwise. If you write the static method properly, you can replace the fourth bullet above with this:

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

Now, after all this is working, refactor the code in the line above and use a method reference in place of an explicit lambda.

Lambda Expressions: Part 1 (Continued)

3. **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 (the first returns whichever of string1 and string2 is longer, and the second always returns string1).

   - StringUtils.betterString(string1, string2, (s1, s2) -> s1.length() > s2.length())
   - 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 true or false. *This is normal Java 7 code* except for the optional (but highly recommended) @FunctionalInterface annotation.
- 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 in every way.*
- Call betterString. You can now use lambdas for the 3rd argument, as in the examples above.

4. **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())