Worksheet: Lambda Expressions

Lab Exercises

You should start by importing the appropriate project into your IDE. Project files are provided for Eclipse and IntelliJ on the repo. All the following questions should be answered using the new Java 8 *lambda expressions*.

1. Create an array containing some `String`s. Sort the array by

   - length (i.e., shortest to longest)
   - reverse length (i.e., longest to shortest)
   - first character
   - Strings that contain `"e"` first, everything else second.

   Remember that 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, consider using

   ```
   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 creating a loop to traverse the array and print out the elements.

2. For the 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.

   Now, rewrite the final lambda sorting example, but use a method reference in place of an explicit lambda.

3. Create a class with a `static` method called `betterString`. This method should take two `String`s and a lambda as its arguments. This lambda states whether the first of the two strings is *better*.

   The method should return the *better* string; i.e., if the lambda returns `true` the method should return the first string, otherwise it should return the second string.

   For the lambda, define an interface called `TwoStringPredicate` with a method that takes two `String`s and returns `true` if the first is *better* than the second, `false` otherwise.

   Here are two examples (the first returns whichever of `test1` and `test2` is longer, and the second always returns `test1`):

   - `StringUtils.betterString(test1, test2, (s1, s2) -> s1.length() > s2.length())`
   - `StringUtils.betterString(test1, test2, (s1, s2) -> true)`

4. Use generics to replace `betterString` with `betterEntry` and `TwoStringPredicate` with `TwoElementPredicate`. Make sure your previous examples still work when you only change `betterString` to `betterElement`.

5. Create a `static` method called `allMatches`. It should take a `List` of `String`s and a `Predicate<String>`, and return a new `List` of all the values that passed the test. Test it with several examples. E.g.:

   - `List<String> shortWords = StringUtils.allMatches(words, s -> s.length() < 4);`
   - `List<String> wordsWithB = StringUtils.allMatches(words, s -> s.contains("b"));`
   - `List<String> evenLengthWords =`
     `StringUtils.allMatches(words, s -> (s.length() % 2) == 0);`

6. Rewrite `allMatches` so it works on any `List` and associated `Predicate`, not just on `String`s. Verify that your examples from the previous question still work.

7. Create a `static` method called `transformedList`. It should take a `List` of `String`s and a `Function<String,String>` and return a new `List` that contains the results of applying the function to each element of the original list. E.g.:

   - `List<String> excitingWords = StringUtils.transformedList(words, s -> s + "");!`
   - `List<String> eyeWords =`
     `StringUtils.transformedList(words, s -> s.replace("i", "eye"));`
   - `List<String> upperCaseWords =`
     `StringUtils.transformedList(words, String::toUpperCase);`

8. Rewrite `transformedList` so it works with generic types. Verify that your examples from the previous question still work.