Exercises: Lambda Expressions Part 4

Here is a quick preview of filtering elements from streams. Notice the use of the Predicates in bold.

```
List<String> words = Arrays.asList("hi", "hello", "bye", "goodbye");
String s1 = words.stream()
                 .filter(word -> word.contains("o"))
                 .findFirst()
                 .orElse(null);
System.out.println(s1);
String s2 = words.stream()
                 .filter(word -> word.length() > 5)
                 .findFirst()
                 .orElse(null);
System.out.println(s2);
```

1. **Practice with filter [easy!].** Make a new project and copy StreamPreview.java from the lambdas-4-exercises project to your new project. Play around with the predicates (the part in bold above) and see what you get. Even though we have not yet covered streams explicitly, you should be able to pick up the basics easily.

2. **Using the and method of Predicate [hard!].** Now, the point of all of this is that filter takes a *single* Predicate, not multiple Predicates. The goal of this problem is to make filtering more flexible by making similar filtering code, but that accepts any number of Predicates instead of a single Predicate. To accomplish this, first make a method called allPassPredicate that accepts any number of generically typed Predicates (recall how to use varargs with "..."), and returns a single Predicate that tests if the argument passes all of the input Predicates. Second, make a method called firstAllMatch that takes a Stream and any number of correspondingly-typed Predicates, and returns the first entry that matches all of the Predicates. Your code will simply make the combined Predicate, then call code like that at the top of the page. For example, if words is a List<String>, the following would find the first word that *both* contains an "o" *and* has length greater than 5.

   ```
   FunctionUtils.firstAllMatch(words.stream(),
                               word -> word.contains("o"),
                               word -> word.length() > 5);
   ```

   Assuming that you use varargs in your solutions, note that you will receive an odd-sounding warning (not error) about potential heap pollution. It is safe to ignore this error for now, but in the file IO lecture we will briefly explain the warning and show how to suppress it with @SafeVarargs.

3. **Using the or method of Predicate [easy if you got problem 2].** Make a method called anyPassPredicate that accepts any number of generically typed Predicates, and returns a single Predicate that tests if the argument passes any of the input Predicates. Then, make a method called firstAnyMatch that takes a Stream and any number of correspondingly-typed Predicates, and returns the first entry that matches any of the Predicates. For example, if words is a List<String>, the following would find the first word that *either* contains an "o" *or* has length greater than 5.

   ```
   FunctionUtils.firstAnyMatch(words.stream(),
                               word -> word.contains("o"),
                               word -> word.length() > 5);
   ```