Lab 8

1. Short answer
   a. Name two differences between imperative and functional programming
   b. Explain the meaning of *declarative programming.* Give an example.
   c. Explain the difference between *functional interface, functor,* and *closure*, and give examples of each using Java 7 syntax
   d. Name three benefits of including functional style programming in Java
   e. For each lambda expression below, name the parameters and the free variables.

      i. Runnable r = () →
      {

```
int[][] products = new int[s][t];
for (int i = 0; i < s; i++) {
   for(int j = i + 1; j < t; j++) {
       products[i][j] = i * j;

   }
}

}
```

      ii. Comparator<String> comp = (s, t) →
      {

```
if(ignoreCase == true) {
  return s.compareToIgnoreCase(t);

} else {
       return s.compareTo(t);
}

}
```

   f. In the lecture, one of the examples of a method reference of type *object::instanceMethod* was `this::equals`. Since every lambda expression must be converted to a functional interface, find a functional interface in the `java.util.function` package that would be used for this lambda expression.

      Hint: Take a look at the api docs here:
      http://docs.oracle.com/javase/8/docs/api/java/util/function/package-summary.html

   g. An example of a method reference is

```
System.out::println
```

      Do the following:
      i. Convert this method reference to a lambda expression.

ii. Determine which type of method reference this is (in the lecture three different types of method reference were mentioned). Explain carefully.

j. An example of a method reference is:

`Math::random`

Its corresponding functional interface is `Supplier<Double>`. Do the following:

i. Rewrite this method reference as a lambda expression
ii. Put this method expression in a `main` method in a Java class and use it to print a random number to the console
iii. Create an equivalent Java class in which the functional behavior of `Math::random` is expressed using an inner class (implementing `Supplier`); call this inner class from a `main` method and use it to output a random number to the console. The behavior should be the same as in part b.

2. *Comparators*. Look at the code in the package `lesson8.labs.prob2.comparator2`. Suppose we sort using the sort method in the `EmployeeInfo` class together with the NameComparator. Look at the `compare` method in the `NameComparator`: If two `Employee` objects have the same name, what is the return value of `compare`? This tells us that these `Employee` objects should be *equal*, but is this always true? Give an example of two `Employee` objects having the same `name` but that should *not* be considered equal. Rewrite the `compare` method so that, if `compare` does return 0, the `Employee` objects are indeed equal. (This issue is known as *consistency with equals.*)

3. Consider the following lambda expression. Can this expression be correctly typed as a `BiFunction`? (See lesson8.lecture.lambdaexamples.bifunction.) (Hint: Yes it can.)

```
(x,y) -> {
            List<Double> list = new ArrayList<>();
            list.add(Math.pow(x,y));
            list.add(x * y);
            return list;
};
```

Demonstrate you are right by doing the following: In the main method of a Java class, assign this lambda expression to an appropriate BiFunction and call the `apply` method with arguments (2.0, 3.0), and print the result to console.

5. Redo problem 3 of Lab 7 in two different ways:

a. Use a lambda expression instead of directly defining a Consumer

b. Use a method reference in place of your lambda expression in (a)

6. Finish the Examples exercise that was given in class (file: *Lambda and Method Reference Exercises*)