# 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. Express the functions defined below using Church's lambda notation:
      i. $f(x) = x + 2x^2$
      ii. $g(x,y) = y - x + x^y$
      iii. $h(x,y,z) = z - (x + y)$
   f. 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. BiFunction<Double, Double, Double> f = (double u, double v) → $\int_a^b \cos ux \sin vx \, dx$

      (Note: the right hand side of the "→" is mathematical notation, not Java, but it can be converted to a large block of Java code having the same free variables. See lecture code to review the BiFunction functional interface.)

      iii.

```
Comparator<String> comp = (s, t) →
{
   if(ignoreCase == true) {
      return s.compareToIgnoreCase(t);

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

}
```

   g. 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 #1: The implicit reference `this' refers to the currently active object. So, to answer this question, create a class MyClass in which you have referenced this::equals with an appropriate type; add a method myMethod(MyClass cl) which uses this method expression to return true if cl is equal to 'this'.

Hint #2: Take a look at the api docs here:
http://docs.oracle.com/javase/8/docs/api/java/util/function/package-summary.html

h. 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*.

A. Look at the code in the package `lesson8.lecture.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.*)

B. Fix the `compare` method, as in part A, for the Comparator used in `lesson8.lecture.comparator3`

c. Fix the `compare` method, as in part A, for the lambda expression used to `compare Employee` objects in `lesson8.lecture.lambdaexamples.comparator3`

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.

4. Implement a method with the following signature and return type:

```
public int countWords(List<String> words, char c, char d, int len)
```

which counts the number of words in the input list `words` that have length equal to `len`, that contain the character `c`, and that do not contain the character `d`. Create a solution like the "Good" solution in lesson8.lecture.filter – a Good solution creates a lambda expression each time values are passed into countWords.

5. Redo lesson7.lab4.prob4 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*)