**Lab 8**

1. Short answer
   1. Name two differences between imperative and functional programming
   2. Explain the meaning of *declarative programming.* Give an example.

Functional programming is declarative programming. Focus what to do, not how to do.

* 1. Explain the difference between *functional interface, functor,* and *closure*, and give examples of each using Java 7 syntax
  2. Name three benefits of including functional style programming in Java
  3. Express the functions defined below using Church’s lambda notation:
     1. f(x) = x + 2x2
     2. g(x,y) = y – x + xy
     3. h(x,y,z) = z – (x + y)

Eg: f(x) = x + 2x2 convert to: (x) -> x + 2 \*x\*x; or x -> x + 2 \*x\*x;

* 1. For each lambda expression below, name the parameters and the free variables.

i. Runnable r = ()   
 {

int[][] products = new int[s][t]; //products: local, we defined. We can add final in front of products  
**for** (**int** i = 0; i < s; i++) {  
 for(int j = i + 1; j < t; j++) {   
 products[i][j] = i \* j;

}   
}

}

ii. BiFunction<U, V, R> f = (double u, double v)

(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);

}

}

* 1. 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>

* 1. An example of a method reference is   
      System.out::println

Do the following:

1. Convert this method reference to a lambda expression.
2. Determine which type of method reference this is (in the lecture three different types of method reference were mentioned). Explain carefully.
3. An example of a method reference is:

Math::random

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

* 1. Rewrite this method reference as a lambda expression
  2. Put this method expression in a main method in a Java class and use it to print a random number to the console
  3. 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.

1. *Comparators.* 
   1. 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.*)
   2. Fix the compare method, as in part A, for the Comparator used in lesson8.lecture.comparator3
   3. Fix the compare method, as in part A, for the lambda expression used to compare Employee objects in lesson8.lecture.lambdaexamples.comparator3

In the question, you need to add additional logic in those Comparator class. For example, in the NameComparator: If two employee objects have the same name, but their salaries are different, but by using NameComparator, the two employee objects will be considered as the same object. Your task is to solve the problem, similar problem you’re facing for SalaryComparator.

The last question is just convert to use Lambda expression solve the same problem.

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.

Do what the demonstrate told you: write a class which contains public static void main() to test your assumption.

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 Good and Better solution, as described in the slides (see lesson8.lecture.filter) – a Good solution creates a lambda expression each time values are passed into countWords, whereas a Better solution has parametrized lambda expressions pre-made, and so a call to countWords simply substitutes values into these expressions. Try also creating a Best solution in which there is just one lambda expression.

You need to understand lesson8.lecture.filter code first. If you’re not understand, please try to do a Good solution which is converting a method into lambda expersion as a return statement.

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*) and provide details with one or two line.