Lab 8

1. Short answer

a. 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;

}

}

}

 **Parameters:** None

 **Free Variables:** s, t

ii.Comparator<String> comp = (s, t) →

{

if(ignoreCase == true) {

return s.compareToIgnoreCase(t);

} else {

return s.compareTo(t);

}

}

 **Parameters:** s, t

 **Free Variables:** ignoreCase

b. 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

() -> Math.random()

1. Put this method expression in a main method in a Java class and use it to print a

random number to the console

public static void main(String[] args) {

Supplier<Double> random = () -> Math.random();

System.out.println(random.get());

}

1. 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 ii.

public static void main(String[] args) {

class RandomSupplier implements Supplier<Double> {

@Override

public Double get() {

return Math.random();

}

}

Supplier<Double> randomSupplier = new RandomSupplier();

System.out.println(randomSupplier.get());

}

2. 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?

🡪 return 0, indicating that they are equal in terms of ordering. However, this does not ensure that the two

Employee objects are actually equal in terms of all attributes.

Give an example of two Employee objects having the same name but

that should not be considered equal.

🡪 Employee e1 = new Employee(“John”, 3000);

Employee e2 = new Employee(“John”, 4000);

Here, e1 and e2 have the same name but different salaries. According to the current compare method, they are considered equal, but they are not.

6.

A. (Employee e) -> e.getName()

🡪 Function<Employee, String> getName1 = (Employee e) -> e.getName();

Function<Employee, String> getName2 = Employee::getName;

B. (Employee e,String s) -> e.setName(s)

BiConsumer<Employee, String> setName1 = (Employee e, String s) -> e.setName(s);

BiConsumer<Employee, String> setName2 = Employee::setName;

C. (String s1,String s2) -> s1.compareTo(s2)

BiFunction<String, String, Integer> compareTo1 = (String s1, String s2) -> s1.compareTo(s2);

BiFunction<String, String, Integer> compareTo2 = String::compareTo;

D. (Integer x,Integer y) -> Math.pow(x,y)

BiFunction<Integer, Integer, Double> pow1 = (Integer x, Integer y) -> Math.pow(x, y);

BiFunction<Integer, Integer, Double> pow2 = Math::pow;

E. (Apple a) -> a.getWeight()

Function<Apple, Double> getWeight1 = (Apple a) -> a.getWeight();

Function<Apple, Double> getWeight2 = Apple::getWeight;

F. (String x) -> Integer.parseInt(x);

Function<String, Integer> parseInt1 = (String x) -> Integer.parseInt(x);

Function<String, Integer> parseInt2 = Integer::parseInt;

G. EmployeeNameComparator comp = **new** EmployeeNameComparator();

(Employee e1, Employee e2) -> comp.compare(e1,e2)

Comparator<Employee> compare1 = (Employee e1, Employee e2) -> comp.compare(e1, e2);

Comparator<Employee> compare2 = comp::compare;