Short answer

1.
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;
        }
    }
}
```

- **Parameter Variables:** products, i, and j
- **Free Variables:** S and t are free variables

ii.

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

- **Parameter Variables:** s, t
- **Free Variables:** ignoreCase

b.

i. Rewrite this method reference as a lambda expression

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

ii. Put this method expression in a `main` method in a Java class and use it to print a random number to the console

- Done , see class Random.java in package Lab8_1

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 ii.

- Done , see class RandomClass.java in package Lab8_1

2. Comparators

According to the code in package lesson8.lecture.comparator2, two objects are considered to be equal if they are of the same class and have the same name and salary. The output when the equals method is executed in the main class gives "false" because we are comparing the same name but different salary. This is true according to our implementation of the equals method. When the compares method is called, it gave us result of zero according the below formulation. So, I don't get any problem with this.

$$\texttt{(e1, e2)} \mapsto \texttt{val} \text{ where } \texttt{val} = \begin{cases} > 0 & \text{if e1.name>e2.name} \\ < 0 & \text{if e1.name<e2.name} \\ 0 & \text{if e1.name equal to e2.name} \end{cases}$$

Give an example of two `Employee` objects having the same `name` but that should *not* be considered equal.

Answer ➔ *Two employees with the same name but different salary are not equal.*

3. 3 Question number 3 – Bifunction

Yes, the given expression can be typed as BiFunction as follows:

```
BiFunction<Double, Double, List<Double>> f = (x, y) ->
Arrays.asList(Math.pow(x,  y),x*y);
```

Result is provided Main class of package Lab8_3