Experiment 6

Name: Bijayini Behera **UID: 22BCS10780**
Branch: BE-CSE **Section/Group: 22BC_IOT-639/A**
Semester: 6th **Date of Performance:28/02/25**
Subject Name: PBLJ Lab **Subject Code: 22CSH-359**

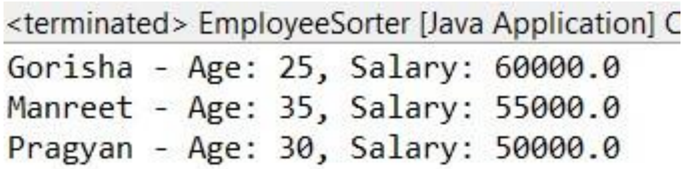
EASY:

1. **Aim:** Write a program to sort a list of Employee objects (name, age, salary) using lambda expressions.

2. **Implementation/Code:**

```
package Java;
import java.util.*;

class Emp {     String
name;     int age;
double salary;


   Emp(String name, int age, double salary) {
this.name = name;        this.age = age;
this.salary = salary;
   }
   public String toString() {        return name + " - Age: " + age +
", Salary: " + salary;
   }
}
public class EmployeeSorter {
   public static void main(String[] args) {
List<Emp> employees = Arrays.asList(           new
Emp("Pragyan", 30, 50000),           new
Emp("Gorisha", 25, 60000),           new
Emp("Manreet", 35, 55000)
       );
       employees.sort(Comparator.comparing((Emp e) -> e.name).thenComparing(e -> e.age)
.thenComparing(e -> e.salary));        employees.forEach(System.out::println);
   }
}
```

3. Output:



```
<terminated> EmployeeSorter [Java Application] C
Gorisha - Age: 25, Salary: 60000.0
Manreet - Age: 35, Salary: 55000.0
Pragyan - Age: 30, Salary: 50000.0
```

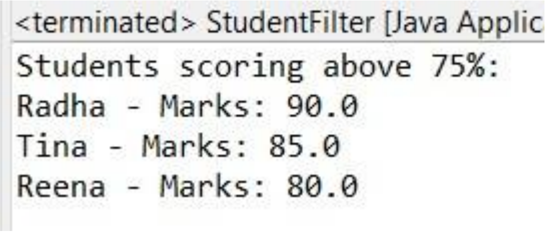
MEDIUM:

1. **Aim:** Create a program to use lambda expressions and stream operations to filter students scoring above 75%, sort them by marks, and display their names.

2. **Implementation/Code:**

```
package Java;
import java.util.*; import
java.util.stream.*;

class Student {     String
name;
   double marks;

   Student(String name, double marks) {
      this.name = name;
      this.marks = marks;
   }
}
public class StudentFilter {     public static void
main(String[] args) {        List<Student> students =
Arrays.asList(           new Student("Reena", 80),
new Student("Boby", 70),           new
Student("Tina", 85),           new Student("Dev",
60),
         new Student("Radha", 90)
      );

      List<Student> filteredStudents = students.stream().filter(s -> s.marks > 75).sorted
(Comparator.comparingDouble(s -> -s.marks)).collect(Collectors.toList());

      System.out.println("Students scoring above 75%:");
      filteredStudents.forEach(s -> System.out.println(s.name + " - Marks: " + s.marks));
   } }
```

3. Output:



```
Students scoring above 75%:
Radha - Marks: 90.0
Tina - Marks: 85.0
Reena - Marks: 80.0
```

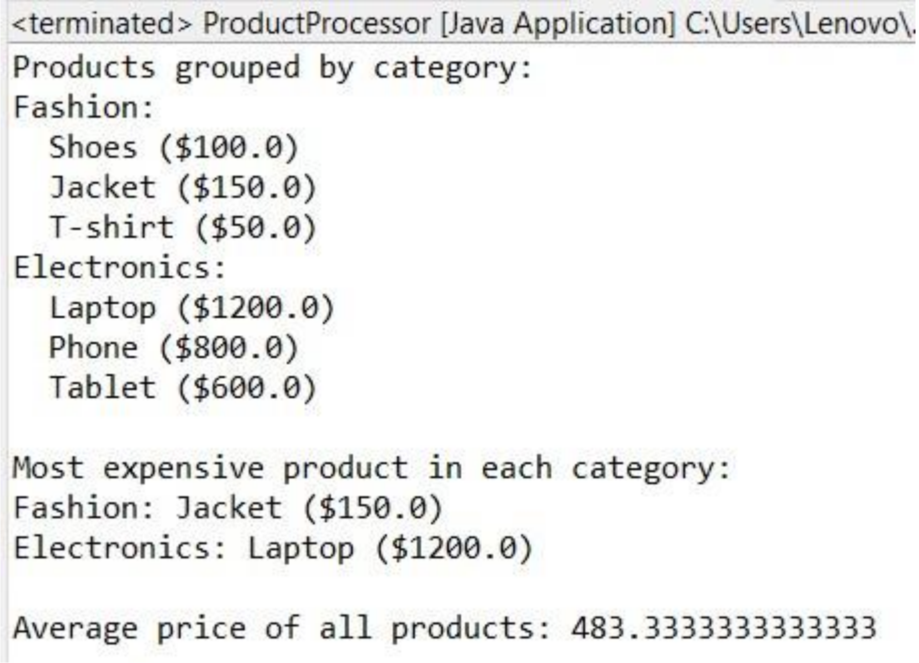
HARD:

1. **Aim:** Write a Java program to process a large dataset of products using streams. Perform operations such as grouping products by category, finding the most expensive product in each category, and calculating the average price of all products.

2. **Implementation/Code:**

```
package Java; import
java.util.*;
import java.util.stream.*;

class Product {
   String name, category;     double
price;

   public Product(String name, String category, double price) {
this.name = name;        this.category = category;        this.price =
price;
   }
   @Override     public String
toString() {
       return name + " ($" + price + ")";
   }
}
public class ProductProcessor {     public static void
main(String[] args) {         List<Product> products = List.of(
new Product("Laptop", "Electronics", 1200.0),            new
Product("Phone", "Electronics", 800.0),            new
Product("Tablet", "Electronics", 600.0),            new
Product("Shoes", "Fashion", 100.0),            new
Product("Jacket", "Fashion", 150.0),            new Product("T-
shirt", "Fashion", 50.0)
        );
        Map<String, List<Product>> groupedByCategory = products.stream()
            .collect(Collectors.groupingBy(p -> p.category));
System.out.println("Products grouped by category:");
groupedByCategory.forEach((category, productList) -> {
System.out.println(category + ":");            productList.forEach(product ->
System.out.println("  " + product));
        });

        Map<String, Optional<Product>> mostExpensiveByCategory = products.stream()
            .collect(Collectors.groupingBy(p -> p.category,
                     Collectors.maxBy(Comparator.comparingDouble(p      ->      p.price))));
System.out.println("\nMost      expensive      product      in      each      category:");
mostExpensiveByCategory.forEach((category, product) ->
            System.out.println(category + ": " + product.orElse(null)));

        double averagePrice = products.stream()
            .collect(Collectors.averagingDouble(p -> p.price));
        System.out.println("\nAverage price of all products: " + averagePrice);
    }
}
```

3. Output:



```
<terminated> ProductProcessor [Java Application] C:\Users\Lenovo\.
Products grouped by category:
Fashion:
  Shoes ($100.0)
  Jacket ($150.0)
  T-shirt ($50.0)
Electronics:
  Laptop ($1200.0)
  Phone ($800.0)
  Tablet ($600.0)

Most expensive product in each category:
Fashion: Jacket ($150.0)
Electronics: Laptop ($1200.0)

Average price of all products: 483.3333333333333
```

4. Learning Outcome

a) Understanding Lambda Expressions – Learn how to use lambda expressions to simplify function definitions and make code more concise.

b) Sorting with Lambda and Comparator – Utilize Comparator.comparing() and thenComparing() for multi-criteria sorting of objects.

c) Using Java Streams for Data Processing – Gain proficiency in filtering, sorting, mapping, and collecting data using Java’s Stream API.

d) Filtering Data with Stream API – Use filter() to extract specific elements from collections based on given conditions.

e) Grouping Data Using Collectors – Understand how to use groupingBy() to categorize and structure data effectively.

f) Finding Max and Min Values in a Dataset – Use maxBy() and minBy() to determine the most expensive or least expensive items in a category.

g) Calculating Aggregates Using Streams – Apply averagingDouble() to compute the average price or marks of a dataset.