EXPERIMENT 6

Name: Shirsha Rakshit **UID: 22BCS12279**

Section: 618-IOT-A **D.O.P: 07-03-25**

Subject: PBLJ Lab **Subject Code: 22CSH-359**

Aim: (a): To implement a Java program that sorts a list of Employee objects (based on name, age, and salary) using lambda expressions and stream operations to demonstrate efficient data processing.
(b) implement Java program that uses lambda expressions and Stream API to filter students who scored above 75%, sort them by marks, and display their names.
(c) To develop a Java program that processes a large dataset of products using Streams class to:
- Group products by category
- Find the most expensive product in each category
- Calculate the average price of all products

1. Objective:

- Develop Java programs using lambda expressions and stream operations for sorting, filtering, and processing large datasets efficiently.
- Implement easy, medium, and hard-level tasks involving sorting employees, filtering and sorting students, and processing products using streams.

2. Implementation/Code:

(a) Code:

```
import java.util.*;

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

    public Employee(String name, int age, double salary) {
        this.name = name;
        this.age = age;
        this.salary = salary;
    }

    public void display() {
        System.out.println(name + " (" + age + ", " + salary + ")");
    }
}

public class EmployeeSorting {
    public static void main(String[] args) {
        List<Employee> employees = new ArrayList<>(Arrays.asList(
            new Employee("Alice", 30, 50000),
            new Employee("Bob", 25, 60000),
            new Employee("Charlie", 35, 55000),
            new Employee("Alex", 28, 45000),
            new Employee("Alex", 32, 47000),
            new Employee("Alex", 25, 46000),
            new Employee("David", 29, 50000),
            new Employee("Eve", 31, 50000),
            new Employee("Frank", 27, 50000)
        ));

        // Sort by Name (Alphabetical Order)
        System.out.println("Sorting by Name:");
        employees.stream()
                .sorted(Comparator.comparing(emp -> emp.name))
                .forEach(Employee::display);

        // Sort by Age (Ascending Order)
        System.out.println("\nSorting by Age:");
        employees.stream()
                .sorted(Comparator.comparingInt(emp -> emp.age))
                .forEach(Employee::display);

        // Sort by Salary (Descending Order)
        System.out.println("\nSorting by Salary:");
        employees.stream()
                .sorted(Comparator.comparingDouble(emp -> -emp.salary))
                .forEach(Employee::display);

        // Edge Case - Same Name, Different Age
        System.out.println("\nSorting by Name and Age:");
        employees.stream()
                .sorted(Comparator.comparing((Employee emp) ->
emp.name).thenComparingInt(emp -> emp.age))
            .forEach(Employee::display);

        // Edge Case - Same Salary, Sorted by Name
        System.out.println("\nSorting by Salary and Name:");
        employees.stream()
            .sorted(Comparator.comparingDouble((Employee emp) -> emp.salary)
                .thenComparing(emp -> emp.name))
            .forEach(Employee::display);
    }
    }
```

(b)

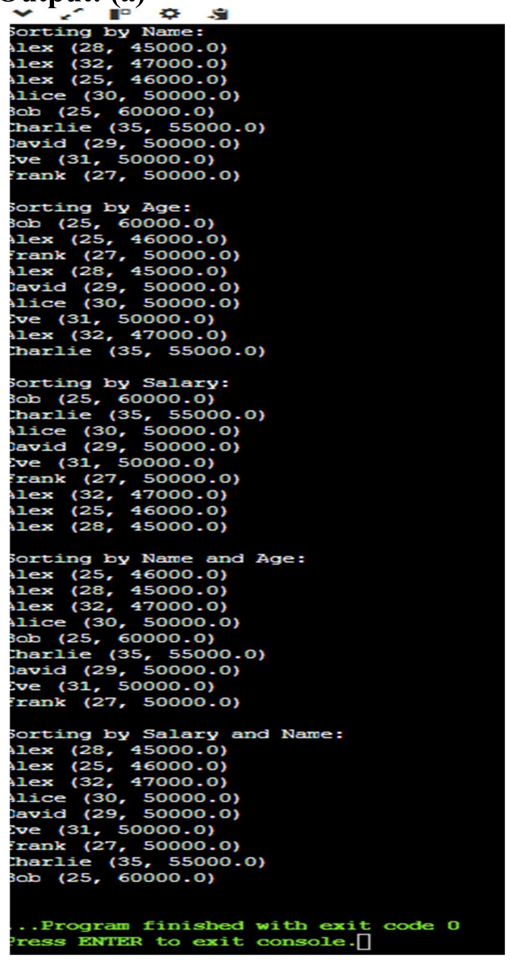
```
 import java.util.*;
import java.util.stream.Collectors;

class Student {
    String name;
    double marks;

    public Student(String name, double marks) {
        this.name = name;
        this.marks = marks;
    }

    public void display() {
        System.out.println(name + " (" + marks + ")");
    }
}
public class Main {
    public static void main(String[] args) {
        List<Student> students = Arrays.asList(
            new Student("Alice", 80),
            new Student("Bob", 72),
            new Student("Charlie", 90),
            new Student("David", 65),
            new Student("Eve", 85),
            new Student("Frank", 65)
        );
        // Filter students who scored above 75%, sort by marks in descending order, then by name
        List<Student> filteredSortedStudents = students.stream()
                .filter(student -> student.marks > 75)
                .sorted(Comparator.comparingDouble((Student student) -> -student.marks)
                        .thenComparing(student -> student.name))
                .collect(Collectors.toList());

        // Display results
        System.out.println("Students who scored above 75%, sorted by marks:");
        filteredSortedStudents.forEach(Student::display);
    }
}
```

(c)Code:

```
import java.util.*;
import java.util.stream.Collectors;

class Product {
    String name;
    String 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 Main {
    public static void main(String[] args) {
        List<Product> products = Arrays.asList(
            new Product("Laptop", "Electronics", 60000),
            new Product("Smartphone", "Electronics", 40000),
            new Product("Refrigerator", "Appliances", 50000),
            new Product("Washing Machine", "Appliances", 30000),
            new Product("TV", "Electronics", 55000),
            new Product("Blender", "Appliances", 5000),
            new Product("Shoes", "Fashion", 3000),
            new Product("T-Shirt", "Fashion", 1200),
            new Product("Jacket", "Fashion", 4500)
        );

        // 1. Group products by category
        Map<String, List<Product>> groupedByCategory = products.stream()
                .collect(Collectors.groupingBy(p -> p.category));

        System.out.println("\nProducts Grouped by Category:");
        groupedByCategory.forEach((category, productList) -> {
            System.out.println(category + ": " + productList);
        });

        // 2. Find the most expensive product in each category
        Map<String, Optional<Product>> mostExpensiveInCategory = products.stream()
                .collect(Collectors.groupingBy(
                        p -> p.category,
                        Collectors.maxBy(Comparator.comparingDouble(p -> p.price))
                ));

        System.out.println("\nMost Expensive Product in Each Category:");
        mostExpensiveInCategory.forEach((category, product) -> {
            product.ifPresent(p -> System.out.println(category + ": " + p));
        });

        // 3. Calculate the average price of all products
        double avgPrice = products.stream()
                .mapToDouble(p -> p.price)
                .average()
                .orElse(0.0);

        System.out.println("\nAverage Price of All Products: ₹" + avgPrice);
    }
```

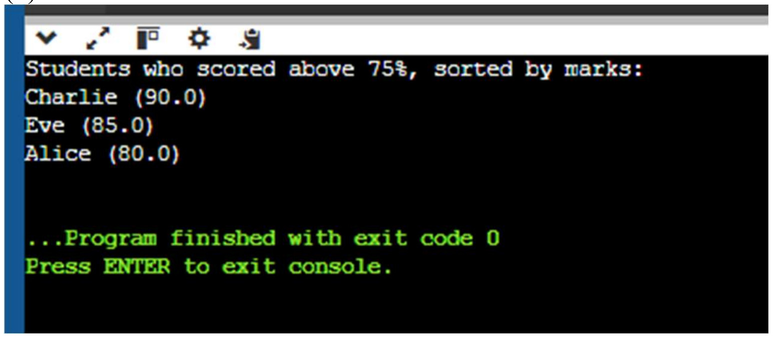
}

3. **Output: (a)**



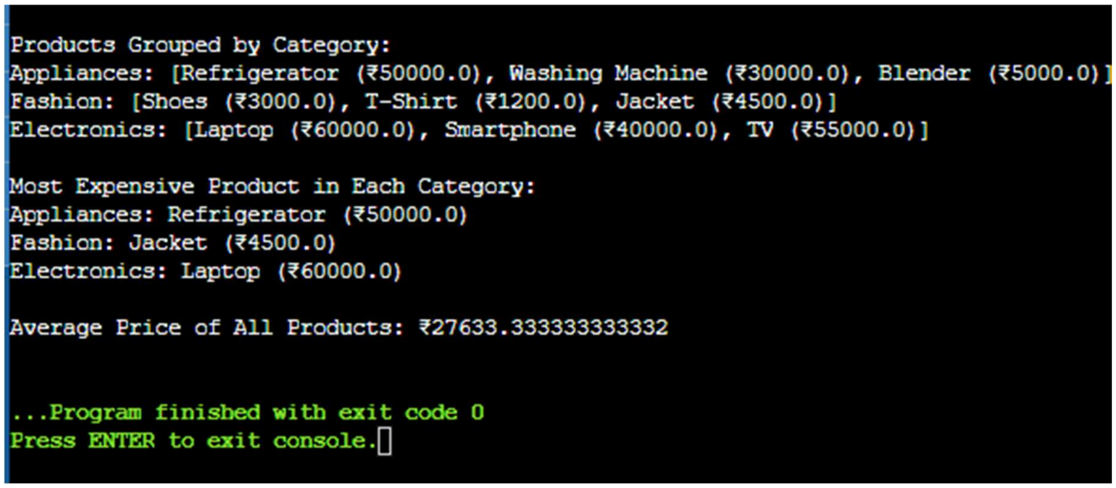
```
orting by Name:
lex (28, 45000.0)
lex (32, 47000.0)
lex (25, 46000.0)
lice (30, 50000.0)
ob (25, 60000.0)
harlie (35, 55000.0)
avid (29, 50000.0)
ve (31, 50000.0)
rank (27, 50000.0)

orting by Age:
ob (25, 60000.0)
lex (25, 46000.0)
rank (27, 50000.0)
lex (28, 45000.0)
avid (29, 50000.0)
lice (30, 50000.0)
ve (31, 50000.0)
lex (32, 47000.0)
harlie (35, 55000.0)

orting by Salary:
ob (25, 60000.0)
harlie (35, 55000.0)
lice (30, 50000.0)
avid (29, 50000.0)
ve (31, 50000.0)
rank (27, 50000.0)
lex (32, 47000.0)
lex (25, 46000.0)
lex (28, 45000.0)

orting by Name and Age:
lex (25, 46000.0)
lex (28, 45000.0)
lex (32, 47000.0)
lice (30, 50000.0)
ob (25, 60000.0)
harlie (35, 55000.0)
avid (29, 50000.0)
ve (31, 50000.0)
rank (27, 50000.0)

orting by Salary and Name:
lex (28, 45000.0)
lex (25, 46000.0)
lex (32, 47000.0)
lice (30, 50000.0)
avid (29, 50000.0)
ve (31, 50000.0)
rank (27, 50000.0)
harlie (35, 55000.0)
ob (25, 60000.0)


..Program finished with exit code 0
ress ENTER to exit console.
```

(b)



```
Students who scored above 75%, sorted by marks:
Charlie (90.0)
Eve (85.0)
Alice (80.0)


...Program finished with exit code 0
Press ENTER to exit console.
```

(c)



```
Products Grouped by Category:
Appliances: [Refrigerator (₹50000.0), Washing Machine (₹30000.0), Blender (₹5000.0)]
Fashion: [Shoes (₹3000.0), T-Shirt (₹1200.0), Jacket (₹4500.0)]
Electronics: [Laptop (₹60000.0), Smartphone (₹40000.0), TV (₹55000.0)]

Most Expensive Product in Each Category:
Appliances: Refrigerator (₹50000.0)
Fashion: Jacket (₹4500.0)
Electronics: Laptop (₹60000.0)

Average Price of All Products: ₹27633.333333333332


...Program finished with exit code 0
Press ENTER to exit console.
```

4. Learning Outcome:

- Understand and implement **lambda expressions** for sorting objects in a list based on different attributes.
- Utilize **Java Streams API** to perform operations like **filtering, sorting, and mapping** efficiently on large datasets.
- Learn **Comparator and method references** to simplify object comparisons for sorting.
- Apply **grouping and aggregation functions** using Collectors.groupingBy() and Collectors.maxBy() for processing categorized data.
- Gain hands-on experience in computing **statistical values** like the **average** from a dataset using mapToDouble() and average().
- Improve **code efficiency and readability** by using **functional programming** techniques in Java.