# Experiment 6

**Student Name: Shivansh Ghildiyal** **UID: 22BCS12928**
**Branch: BE-CSE** **Section/Group: EPAM 801-B**
**Semester: 6th** **Date of Performance: 17/03/25**
**Subject Name: Project Based Learning in Java** **Subject Code: 22CSH-359**

## 1. Aim:

To write a program to implement searching and sorting using lambda expressions and stream operations in java.

## 2. Objective:

To develop Java programs using lambda expressions and stream operations for sorting, filtering, and processing large datasets efficiently.

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

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

c) 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.

## 3. Implementation/Code:

```
import java.util.*;
import java.util.stream.Collectors;
public class DataProcessing {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        while (true) {
            System.out.println("\n--- MENU ---");
            System.out.println("1. Sort Employees by Salary");
            System.out.println("2. Filter and Sort Students by Marks");
            System.out.println("3. Process Products (Grouping, Sorting,
Averages)");
            System.out.println("4. Exit");
            System.out.print("Enter your choice: ");
            int choice = scanner.nextInt();
            switch (choice) {
                case 1:
                    processEmployees();
                    break;
```

```
                        case 2:
                            processStudents();
                            break;
                        case 3:
                            processProducts();
                            break;
                        case 4:
                            System.out.println("Exiting... Thank you!");
                            scanner.close();
                            return;
                        default:
                            System.out.println("Invalid choice. Please try
again.");
                    }}}
            static class Employee {
                String name;
                int age;
                double salary;
                Employee(String name, int age, double salary) {
                    this.name = name;
                    this.age = age;
                    this.salary = salary;}
                @Override
                public String toString() {
                    return name + " - Age: " + age + ", Salary: " + salary;}}
            public static void processEmployees() {
                List<Employee> employees = Arrays.asList(
                    new Employee("Raman", 30, 50000),
                    new Employee("Ashish", 25, 70000),
                    new Employee("Shivam", 35, 60000));
                employees.sort(Comparator.comparingDouble(emp -> emp.salary));
                System.out.println("\nSorted Employees by Salary:");
                employees.forEach(System.out::println);
            }
            static class Student {
                String name;
                double marks;
                Student(String name, double marks) {
                    this.name = name;
                    this.marks = marks;}}
            public static void processStudents() {
                List<Student> students = Arrays.asList(
                    new Student("Jignesh", 85),
                    new Student("Sana", 70),
                    new Student("Harish", 90),
                    new Student("Suman", 65),
                    new Student("Om", 80)
                );
```

```
        List<String> topStudents = students.stream()
                .filter(s -> s.marks > 75)
                .sorted(Comparator.comparingDouble(s -> -s.marks))
                .map(s -> s.name)
                .collect(Collectors.toList());
        System.out.println("\nTop Scoring Students: " + topStudents);
    }
    static class Product {
        String name;
        String category;
        double price;
        Product(String name, String category, double price) {
            this.name = name;
            this.category = category;
            this.price = price;}}
    public static void processProducts() {
        List<Product> products = Arrays.asList(
            new Product("Laptop", "Electronics", 800),
            new Product("Phone", "Electronics", 500),
            new Product("Headphones", "Electronics", 150),
            new Product("Shirt", "Clothing", 40),
            new Product("Jeans", "Clothing", 60),
            new Product("Sofa", "Furniture", 300),
            new Product("Table", "Furniture", 200));
        Map<String, List<Product>> groupedProducts = products.stream()
                .collect(Collectors.groupingBy(p -> p.category));
        System.out.println("\nProducts grouped by category:");
        groupedProducts.forEach((category, productList) -> {
            System.out.println(category + ": " + productList.stream()
                    .map(p -> p.name)
                    .collect(Collectors.joining(", ")));
        });
        Map<String, Optional<Product>> expensiveProducts =
products.stream()
                .collect(Collectors.groupingBy(p -> p.category,
                        Collectors.maxBy(Comparator.comparingDouble(p ->
p.price))));
        System.out.println("\nMost expensive product in each category:");
        expensiveProducts.forEach((category, product) ->
                System.out.println(category + ": " + product.map(p ->
p.name + " - $" + p.price).orElse("No products")));
        double averagePrice = products.stream()
                .mapToDouble(p -> p.price)
                .average()
                .orElse(0.0);
        System.out.println("\nAverage price of all products: $" +
averagePrice);
    }}
```

## 4. Output

```
--- MENU ---
1. Sort Employees by Salary
2. Filter and Sort Students by Marks
3. Process Products (Grouping, Sorting, Averages)
4. Exit
Enter your choice: 1

Sorted Employees by Salary:
Raman - Age: 30, Salary: 50000.0
Shivam - Age: 35, Salary: 60000.0
Ashish - Age: 25, Salary: 70000.0

--- MENU ---
1. Sort Employees by Salary
2. Filter and Sort Students by Marks
3. Process Products (Grouping, Sorting, Averages)
4. Exit
Enter your choice: 2

Top Scoring Students: [Harish, Jignesh, Om]

--- MENU ---
1. Sort Employees by Salary
2. Filter and Sort Students by Marks
3. Process Products (Grouping, Sorting, Averages)
4. Exit
Enter your choice:
```

## 5. Learning Outcome

i. Understood how lambda expressions and stream operations can be used for sorting, filtering, and processing data efficiently.

ii. Learned how lists, maps, and grouping operations work in Java.

iii. Implementing Switch-Case for User Interaction.