Experiment-6

Student Name: Pratap Aditya Singh **UID:** 22BCS16464

Branch: BE-CSE **Section/Group:** KPIT-902/B

Semester: 6th **Date of Performance:** 27/02/25

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

1. Aim:

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.

2. Objective:

a.) Sorting Data: Using lambda expressions to sort a list of Employee objects based on attributes like name, age, and salary.

b.) Filtering and Sorting: Filtering students who scored above 75%, sorting them by marks, and displaying their names using streams.

c.) Processing Large Datasets: Performing operations on a dataset of products, such as grouping by category, identifying the most expensive product in each category, and calculating the average price using stream operations.

3. Implementation/Code:

a.)

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;

class Employee {
    String name;
    int age;
```

```
    double salary;

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

    public String getName() {
        return name;
    }

    public int getAge() {

        return age;
    }

    public double getSalary() {
        return salary;
    }

    @Override
    public String toString() {
        return "Employee{name='" + name + "', age=" + age + ", salary=" + salary + "}";
    }
}

public class EmployeeSorter {
    public static void main(String[] args) {
        List<Employee> employees = new ArrayList<>();
        employees.add(new Employee("Saiful Haque", 21, 172000));
        employees.add(new Employee("Haque", 22, 72000));
        employees.add(new Employee("Saiful", 23, 92000));

        Collections.sort(employees, Comparator.comparing(Employee::getName));
        System.out.println("Sorted by Name:");
        employees.forEach(System.out::println);
```

```
        Collections.sort(employees, Comparator.comparingInt(Employee::getAge));
        System.out.println("\nSorted by Age:");
        employees.forEach(System.out::println);

        Collections.sort(employees, Comparator.comparingDouble(Employee::getSalary));
        System.out.println("\nSorted by Salary:");
        employees.forEach(System.out::println);
    }
}
```

b.)

```
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

public class StudentFilter {
    public static void main(String[] args) {
        List<Student> students = Arrays.asList(
                new Student("Saiful Haque", 92),
                new Student("Haque", 62),
                new Student("Farhad", 98),
                new Student("Saif", 72),
                new Student("Karan", 88));

        List<String> filteredStudents = students.stream()
                .filter(student -> student.getMarks() > 75)
                .sorted((s1, s2) -> Integer.compare(s2.getMarks(), s1.getMarks()))

                .map(Student::getName)
                .collect(Collectors.toList());

        filteredStudents.forEach(System.out::println);
    }
}

class Student {
    private String name;
```

```
    private int marks;

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

    public String getName() {
        return name;
    }

    public int getMarks() {
        return marks;
    }
}
```

c.)

```
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;
    }

    public String getCategory() {
        return category;
    }

    public double getPrice() {
```

```
        return price;
    }

    @Override
    public String toString() {
        return name + " (" + category + "): $" + price;
    }
}

public class ProductProcessor {
    public static void main(String[] args) {
        List<Product> products = Arrays.asList(
                new Product("MacBook Pro", "Electronics", 2500.00),
                new Product("Samsung Galaxy S23", "Electronics", 999.99),
                new Product("iPad Pro", "Electronics", 1299.00),
                new Product("Ergonomic Office Chair", "Furniture", 350.00),
                new Product("Wooden Dining Table", "Furniture", 850.00),
                new Product("Leather Recliner Sofa", "Furniture", 1200.00),
                new Product("Nike Air Max", "Clothing", 180.00),
                new Product("Levi's Denim Jacket", "Clothing", 120.00),
                new Product("Saiful Haque's Special Product", "Exclusive", 15656.00));

        Map<String, List<Product>> groupedByCategory = products.stream()
                .collect(Collectors.groupingBy(Product::getCategory));

        Map<String, Optional<Product>> mostExpensiveByCategory =
groupedByCategory.entrySet().stream()
                .collect(Collectors.toMap(Map.Entry::getKey,
                        entry ->
entry.getValue().stream().max(Comparator.comparing(Product::getPrice))));

        double averagePrice = products.stream()
                .collect(Collectors.averagingDouble(Product::getPrice));

        System.out.println("Most Expensive Products by Category:");
        mostExpensiveByCategory
                .forEach((category, product) -> System.out.println(category + ": " +
product.orElse(null)));
```

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

4. Output:

a.)

```
PS D:\ClassProblem\Java\Exp6> cd "d:\ClassProblem\Java\Exp6\" ;
 if ($?) { javac EmployeeSorter.java } ; if ($?) { java Employe
eSorter }
Sorted by Name:
Employee{name='Haque', age=22, salary=72000.0}
Employee{name='Saiful', age=23, salary=92000.0}
Employee{name='Saiful Haque', age=21, salary=172000.0}

Sorted by Age:
Employee{name='Saiful Haque', age=21, salary=172000.0}
Employee{name='Haque', age=22, salary=72000.0}
Employee{name='Saiful', age=23, salary=92000.0}

Sorted by Salary:
Employee{name='Haque', age=22, salary=72000.0}
Employee{name='Saiful', age=23, salary=92000.0}
Employee{name='Saiful Haque', age=21, salary=172000.0}
PS D:\ClassProblem\Java\Exp6>
```

b.)

```
PS D:\ClassProblem\Java\Exp6> cd "d:\ClassProblem\Java\Exp6\" ;
 if ($?) { javac StudentFilter.java } ; if ($?) { java StudentF
ilter }
Farhad
Saiful Haque
Karan
PS D:\ClassProblem\Java\Exp6>
```

c.)

```
PS D:\ClassProblem\Java\Exp6> cd "d:\ClassProblem\Java\Exp6\" ;
 if ($?) { javac ProductProcessor.java } ; if ($?) { java Produ
ctProcessor }
Most Expensive Products by Category:
Clothing: Nike Air Max (Clothing): $180.0
Exclusive: Saiful Haque's Special Product (Exclusive): $15656.0
Electronics: MacBook Pro (Electronics): $2500.0
Furniture: Leather Recliner Sofa (Furniture): $1200.0

Average Price of All Products: $2572.7766666666666
PS D:\ClassProblem\Java\Exp6>
```

5. Learning Outcome:

1. Used lambda expressions to simplify sorting, filtering, and processing operations.
2. Applied Java Stream API to efficiently handle large datasets.
3. Implemented sorting of Employee objects based on name, age, and salary using lambda expressions.
4. Filtered and sorted student records based on marks using stream operations.
5. Processed product datasets, grouped them by category, and identified the most expensive product in each category.
6. Calculated the average price of all products using stream operations.
7. Practiced functional programming concepts like method references and collectors.
8. Improved data manipulation skills using Java collections and streams.