Experiment-6

Student Name: Kamalpreet Singh **UID: 22BCS11720**
Branch: B.E- CSE **Section/Group: IOT_643-A**
Semester: 6th **DateofPerformance:03-03-2025**
Subject Name: Project Based Learning Java with Lab **Subject Code: 22CSH-359**

Aim:

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

Objective:

- Implement lambda expressions for sorting and filtering data.
- Use stream operations to process large datasets efficiently.
- Develop applications that perform grouping, sorting, and calculations using Java Streams.

Implementation/Code:

Easy Level: Sorting Employees Using Lambda Expressions

```
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 String toString() {
        return name + " - Age: " + age + ", Salary: $" + salary;
    }
}

public class EmployeeSorting {
    public static void main(String[] args) {
        List<Employee> employees = Arrays.asList(
            new Employee("Alice", 30, 50000),
            new Employee("Bob", 25, 60000),
            new Employee("Charlie", 35, 70000)
```

```
        );

        employees.sort(Comparator.comparing(emp -> emp.name));
        System.out.println("Sorted by Name: " + employees);
    }
}
```

Medium Level: Filtering and Sorting Students Using Streams

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

class Student {
    String name;
    double marks;

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

    public String toString() {
        return name + " - Marks: " + marks;
    }
}

public class StudentFiltering {
    public static void main(String[] args) {
        List<Student> students = Arrays.asList(
            new Student("John", 85),
            new Student("Alice", 72),
            new Student("Bob", 90),
            new Student("Charlie", 78)
        );

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

        System.out.println("Filtered & Sorted Students: " + filtered);
    }
}
```

Hard Level: Processing Large Dataset of Products Using Streams

```
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;
    }

    public String toString() {
        return name + " - Category: " + category + ", Price: $" + price;
    }
}

public class ProductProcessing {
    public static void main(String[] args) {
        List<Product> products = Arrays.asList(
            new Product("Laptop", "Electronics", 1200),
            new Product("Phone", "Electronics", 800),
            new Product("Shirt", "Clothing", 50),
            new Product("Jeans", "Clothing", 60)
        );

        Map<String, Double> avgPrice = products.stream()
            .collect(Collectors.groupingBy(p -> p.category,
Collectors.averagingDouble(p -> p.price)));

        System.out.println("Average Price by Category: " + avgPrice);
    }
}
```

Output:

Easy-

```
Sorted by Name: [Alice - Age: 30, Salary: $50000.0, Bob - Age: 25, Salary: $60000.0, Charlie - Age: 35, Salary: $70000.0]
```

Medium-

```
Filtered & Sorted Students: [Bob - Marks: 90.0, John - Marks: 85.0, Charlie - Marks: 78.0]
```

Hard-

```
Average Price by Category: {Clothing=55.0, Electronics=1000.0}
```

Learning Outcomes:

- Understand lambda expressions for sorting and filtering data.
- Learn Java Streams for efficient dataset processing.
- Implement grouping and statistical calculations using Java Streams.
- Develop applications that process large datasets effectively.