

Experiment-6

Student Name: Madhavi kumawat

Branch: CSE

Semester: 6th

Subject Name: Java Lab

UID: 22BCS12660

Section/Group: IOT-618/A

Date of Performance: 28/02/25

Subject Code: 22CSH-359

Problem-1 (Easy)

1. Aim:

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.

2. Implementation/Code:

```
import java.util.*;

class Employee {
    String name;
    int age;
    double salary;
    // Constructor
    public Employee(String name, int age, double salary) {
        this.name = name;
        this.age = age;
        this.salary = salary;
    }
    // Display method
    public void display() {
        System.out.println(name + " (Age: " + age + ", Salary: " + salary + ")");
    }
}
```

```
public class EmployeeSort {  
    public static void main(String[] args) {  
        // Creating a list of Employees  
        List<Employee> employees = new ArrayList<>();  
        employees.add(new Employee("Alice", 30, 50000));  
        employees.add(new Employee("Bob", 25, 60000));  
        employees.add(new Employee("Charlie", 35, 55000));  
  
        System.out.println("Sorted by Name (Alphabetical Order):");  
        employees.stream()  
            .sorted(Comparator.comparing(emp -> emp.name))  
            .forEach(Employee::display);  
  
        System.out.println("\nSorted by Age (Ascending Order):");  
        employees.stream()  
            .sorted(Comparator.comparingInt(emp -> emp.age))  
            .forEach(Employee::display);  
  
        System.out.println("\nSorted by Salary (Descending Order):");  
        employees.stream()  
            .sorted(Comparator.comparingDouble(emp -> -emp.salary))  
            .forEach(Employee::display);  
    }  
}
```

3. Output:

```
Sorted by Name (Alphabetical Order):  
Alice (Age: 30, Salary: 50000.0)  
Bob (Age: 25, Salary: 60000.0)  
Charlie (Age: 35, Salary: 55000.0)  
  
Sorted by Age (Ascending Order):  
Bob (Age: 25, Salary: 60000.0)  
Alice (Age: 30, Salary: 50000.0)  
Charlie (Age: 35, Salary: 55000.0)  
  
Sorted by Salary (Descending Order):  
Bob (Age: 25, Salary: 60000.0)  
Charlie (Age: 35, Salary: 55000.0)  
Alice (Age: 30, Salary: 50000.0)
```

Problem-2 (Medium)

1. Aim:

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.

2. Implementation/Code:

```
import java.util.*;  
import java.util.stream.Collectors;  
class Student {  
    String name;  
    double marks;  
  
    // Constructor  
    public Student(String name, double marks) {  
        this.name = name;  
        this.marks = marks;  
    }  
  
    // Display method  
    public void display() {  
        System.out.println(name + " (Marks: " + marks + ")");  
    }  
}
```

```
    }  
}  
  
public class StudentFilterSort {  
    public static void main(String[] args) {  
        // Creating a list of students  
        List<Student> students = new ArrayList<>();  
        students.add(new Student("Alice", 80));  
        students.add(new Student("Bob", 72));  
        students.add(new Student("Charlie", 90));  
        students.add(new Student("David", 65));  
        students.add(new Student("Eve", 85));  
  
        System.out.println("Students who scored above 75%, sorted by marks:");  
  
        // Filtering students with marks > 75%, sorting in descending order, and collecting  
        results  
        List<Student> filteredStudents = students.stream()  
            .filter(student -> student.marks > 75) // Filter students above 75%  
            .sorted(Comparator.comparingDouble((Student student) -> -student.marks)  
                .thenComparing(student -> student.name)) // Sort by marks (descending) &  
            name (ascending)  
            .collect(Collectors.toList());  
  
        // Displaying the sorted students  
        if (filteredStudents.isEmpty()) {  
            System.out.println("No students scored above 75%.");  
        } else {  
            filteredStudents.forEach(Student::display);  
        }  
    }  
}
```

3. Output:

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

Problem-3 (Hard)

1. Aim:

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

2. Implementation/Code:

```
import java.util.*;  
import java.util.stream.Collectors;  
import java.util.Comparator;  
import java.util.Optional;  
  
class Product {  
    String name;  
    String category;  
    double price;  
  
    // Constructor  
    public Product(String name, String category, double price) {  
        this.name = name;  
        this.category = category;  
        this.price = price;  
    }  
  
    // Display method  
    public void display() {
```

```
        System.out.println(name + " (" + category + ") - $" + price);
    }
}

public class ProductProcessor {
    public static void main(String[] args) {
        // Creating a list of products
        List<Product> products = Arrays.asList(
            new Product("Laptop", "Electronics", 1200),
            new Product("Phone", "Electronics", 800),
            new Product("TV", "Electronics", 1500),
            new Product("Shirt", "Clothing", 50),
            new Product("Shoes", "Footwear", 100),
            new Product("Sneakers", "Footwear", 120),
            new Product("Jacket", "Clothing", 200)
        );

        // Grouping products by category
        Map<String, List<Product>> groupedProducts = products.stream()
            .collect(Collectors.groupingBy(product -> product.category));

        System.out.println("Products grouped by category:");
        groupedProducts.forEach((category, productList) -> {
            System.out.println(category + ": " + productList.stream()
                .map(p -> p.name)
                .collect(Collectors.joining(", ")));
        });

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

        System.out.println("\nMost Expensive Product in Each Category:");
        mostExpensiveByCategory.forEach((category, product) ->
```

```
        System.out.println(category + ": " +  
            (product.isPresent() ? product.get().name + " - $" + product.get().price : "No  
products"))));  
  
    // Calculating the average price of all products  
    double averagePrice = products.stream()  
        .collect(Collectors.averagingDouble(product -> product.price));  
  
    System.out.println("\nAverage Price of All Products: $" + averagePrice);  
    }  
}
```

3. Output:

```
Products grouped by category:  
Clothing: Shirt, Jacket  
Footwear: Shoes, Sneakers  
Electronics: Laptop, Phone, TV  
  
Most Expensive Product in Each Category:  
Clothing: Jacket - $200.0  
Footwear: Sneakers - $120.0  
Electronics: TV - $1500.0  
  
Average Price of All Products: $567.1428571428571
```