Experiment 6

Student Name: Ronit **UID: 22BCS10902**
Branch: B.E CSE **Section: IOT-643-A**
Semester: 6th **DOP: 03/03/25**
Subject: PBLJ **Subject Code: 22CSH-359**

Aim:

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

Problem Statement :

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

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

3) 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.

Algorithm:

1. **Sorting a List of Employee Objects Using Lambda Expressions:**

   - Define an Employee class with attributes name, age, and salary.
   - Create a list to store Employee objects.
   - Populate the list with sample employees.
   - Use lambda expressions with Collections.sort() or stream().sorted():
     - Sort by name (Comparator.comparing(Employee::getName))
     - Sort by age (Comparator.comparingInt(Employee::getAge))
     - Sort by salary (Comparator.comparingDouble(Employee::getSalary))
   - Print the sorted employee list.

2. Filtering and Sorting Students Using Lambda and Streams:

- Define a Student class with attributes name and marks.
- Create a list of Student objects.
- Populate the list with student data.
- Use a stream to:
  - Filter students with marks above 75%.
  - Sort them in descending order using sorted(Comparator.comparingDouble(Student::getMarks).reversed()).
  - Map to extract only student names.
- Print the filtered and sorted student names.

3. Processing a Large Dataset of Products Using Streams:

- Define a Product class with name, category, and price attributes.
- Create a list of Product objects.
- Populate the list with product data.
- Use **streams** for operations:
  - **Group products by category** using Collectors.groupingBy(Product::getCategory).
  - **Find the most expensive product in each category** using Collectors.maxBy(Comparator.comparingDouble(Product::getPrice)).
  - **Calculate the average price** using Collectors.averagingDouble(Product::getPrice).
- Print the grouped products, most expensive items, and average price.

Program :

1. Sorting a List of Employee Objects 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;
    }
    @Override
    public String toString() {
        return name + " - Age: " + age + ", Salary: " + salary;
    }
}
public class EmployeeSorting {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        List<Employee> employees = new ArrayList<>();
        System.out.print("Enter number of employees: ");
        int n = scanner.nextInt();
        scanner.nextLine(); // Consume newline
        for (int i = 0; i < n; i++) {
            System.out.println("Enter details for Employee " + (i + 1) + ":");
            System.out.print("Name: ");
            String name = scanner.nextLine();
            System.out.print("Age: ");
            int age = scanner.nextInt();
            System.out.print("Salary: ");
            double salary = scanner.nextDouble();
            scanner.nextLine(); // Consume newline
            employees.add(new Employee(name, age, salary));
        }
        employees.sort((e1, e2) -> Double.compare(e1.salary, e2.salary));
        System.out.println("\nSorted Employees by Salary:");
        employees.forEach(System.out::println);
        scanner.close();
    }
```

2. Filtering and Sorting Students Using Lambda and Streams :

```
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 class StudentFiltering {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        List<Student> students = new ArrayList<>();
        System.out.print("Enter the number of students: ");
        int n = scanner.nextInt();
        scanner.nextLine();
        for (int i = 0; i < n; i++) {
            System.out.println("Enter details for Student " + (i + 1) + ":");
            System.out.print("Name: ");
            String name = scanner.nextLine();
            System.out.print("Marks: ");
            double marks = scanner.nextDouble();
            scanner.nextLine();
            students.add(new Student(name, marks));
        }
        List<String> topStudents = students.stream()
            .filter(s -> s.marks > 75)
            .sorted((s1, s2) -> Double.compare(s2.marks, s1.marks))
            .map(s -> s.name) // Extract names
            .collect(Collectors.toList());
        System.out.println("\nStudents scoring above 75% (Sorted by marks):");
        topStudents.forEach(System.out::println);
        scanner.close();
    }
}
```

3. Processing a Large Dataset of Products Using Streams:

```
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 toString() {
        return name + " (" + category + ") - $" + price;
    }
}
public class ProductProcessing {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        List<Product> products = new ArrayList<>();
        while (true) {
            System.out.print("Enter product name (or type 'stop' to finish): ");
            String name = scanner.nextLine();
            if (name.equalsIgnoreCase("stop")) break;
            System.out.print("Enter product category: ");
            String category = scanner.nextLine();
            System.out.print("Enter product price: ");
            double price = scanner.nextDouble();
            scanner.nextLine();
            products.add(new Product(name, category, price));
        }
        Map<String, List<Product>> groupedByCategory = products.stream()
            .collect(Collectors.groupingBy(p -> p.category));
        Map<String, Optional<Product>> mostExpensiveByCategory =
products.stream()
            .collect(Collectors.groupingBy(p -> p.category,
                Collectors.maxBy(Comparator.comparingDouble(p -> p.price))));
        double averagePrice = products.stream()
            .mapToDouble(p -> p.price)
            .average()
            .orElse(0);
        System.out.println("\nGrouped Products by Category:");
        groupedByCategory.forEach((category, productList) ->
            System.out.println(category + ": " + productList));
        System.out.println("\nMost Expensive Product in Each Category:");
        mostExpensiveByCategory.forEach((category, product) ->
            System.out.println(category + ": " + product.get()));
        System.out.println("\nAverage Price of All Products: $" + averagePrice);
        scanner.close();
    }
}
```

OUTPUT :

1. **Sorting a List of Employee Objects Using Lambda Expressions:**

```
Enter number of employees: 2
Enter details for Employee 1:
Name: wer
Age: 25
Salary: 85000
Enter details for Employee 2:
Name: asd
Age: 35
Salary: 90000

Sorted Employees by Salary:
wer - Age: 25, Salary: 85000.0
asd - Age: 35, Salary: 90000.0


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

2. **Filtering and Sorting Students Using Lambda and Streams :**

```
Enter the number of students: 2
Enter details for Student 1:
Name: wer
Marks: 89
Enter details for Student 2:
Name: asd
Marks: 69

Students scoring above 75% (Sorted by marks):
wer


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

3. **Processing a Large Dataset of Products Using Streams:**

```
Enter product name (or type 'stop' to finish): broom
Enter product category: cleaning
Enter product price: 90
Enter product name (or type 'stop' to finish): bowl
Enter product category: utensils
Enter product price: 190
Enter product name (or type 'stop' to finish): stop

Grouped Products by Category:
cleaning: [broom (cleaning) - $90.0]
utensils: [bowl (utensils) - $190.0]

Most Expensive Product in Each Category:
cleaning: broom (cleaning) - $90.0
utensils: bowl (utensils) - $190.0

Average Price of All Products: $140.0


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

Learning Outcomes:

- **Implement object-oriented programming with classes, encapsulation, and serialization.**
- **Utilize core Java concepts like loops, conditionals, autoboxing, and unboxing.**
- **Apply file handling with serialization, deserialization, and exception management.**