



Experiment 5

Student Name: Piyushree Rani Sinha

Branch: BE-CSE

Semester: 6th

Subject Name: Project Based Java Lab

UID: 22BCS13178

Section/Group: EPAM_801-B

Date of Performance: 18/3/25

Subject Code: 22CSP-359

1. Aim:

A- Easy Level: Write a program to sort a list of Employee objects (name, age, salary) using lambda expressions.

B- Medium Level: 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- Hard Level: 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- The objective of this program is to demonstrate how to use lambda expressions in Java to sort a list of Employee objects based on different attributes (name, age, and salary). This approach simplifies sorting operations by utilizing the Comparator interface and functional programming concepts. The program showcases how lambda expressions make code more readable and reduce boilerplate code when dealing with sorting operations.

B- This Java program utilizes lambda expressions and stream operations to efficiently filter students who have scored above 75%, sort them by their marks in descending order, and display their names. By leveraging Java's Stream API, the program achieves concise and readable code, avoiding traditional loops. The implementation improves performance and readability while demonstrating the power of functional programming in Java.

C- The objective of this Java program is to efficiently process a large dataset of products using Java Streams, performing operations such as grouping products by category, finding the most expensive product in each category, and calculating the average price of all products. By utilizing Streams and Collectors, the program eliminates the need for traditional loops, improving readability and performance when handling large datasets.

3. Implementation/Code:

A- 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 void display() {
        System.out.println(name + " | Age: " + age + " | Salary: " + salary);
    }
}

public class SortEmployeesUsingLambdaExpression{
    public static void main(String[] args) {
        List<Employee> employees = new ArrayList<>();
        employees.add(new Employee("Anna", 30, 50000));
        employees.add(new Employee("Robert", 25, 60000));
        employees.add(new Employee("Lily", 35, 55000));
        employees.sort((e1, e2) -> e1.name.compareTo(e2.name));
        System.out.println("Sorted by Name:");
        employees.forEach(Employee::display);
        employees.sort((e1, e2) -> Integer.compare(e1.age, e2.age));
```

```
        System.out.println("\nSorted by Age:");
        employees.forEach(Employee::display);
        employees.sort((e1, e2) -> Double.compare(e1.salary, e2.salary));
        System.out.println("\nSorted by Salary:");
        employees.forEach(Employee::display);
    }
}

B- 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 FilterStudentUsingLambdaExpressions{
    public static void main(String[] args) {
        List<Student> students = Arrays.asList(
            new Student("Alice", 85.5),
            new Student("Bob", 72.3),
            new Student("Charlie", 90.1),
            new Student("David", 65.0),
            new Student("Emma", 78.8)
        );
        List<String> topStudents = students.stream()
            .filter(s -> s.marks > 75)
            .sorted((s1, s2) -> Double.compare(s2.marks, s1.marks))
            .map(s -> s.name)
            .collect(Collectors.toList());
        System.out.println("Students scoring above 75% (Sorted by marks):");
    }
}
```

```
        topStudents.forEach(System.out::println);
    }
}

C- import java.io.*;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
class Employee implements Serializable {
    private static final long serialVersionUID = 1L;
    private int empId;
    private String name;
    private String designation;
    private double salary;
    public Employee(int empId, String name, String designation, double
salary) {
        this.empId = empId;
        this.name = name;
        this.designation = designation;
        this.salary = salary;
    }
    public void display() {
        System.out.println("ID: " + empId + ", Name: " + name + ",
Designation: " + designation + ", Salary: " + salary);
    }
}

public class MenuBasedEmployeeManagementSystem{
    private static final String FILE_NAME = "employees.dat";
    private static List<Employee> employeeList = new ArrayList<>();
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        loadEmployees();
        while (true) {
```

```
System.out.println("\nMenu:");
System.out.println("1. Add an Employee");
System.out.println("2. Display All Employees");
System.out.println("3. Exit");
System.out.print("Choose an option: ");
int choice = scanner.nextInt();
switch (choice) {
    case 1:
        addEmployee(scanner);
        break;
    case 2:
        displayAllEmployees();
        break;
    case 3:
        saveEmployees();
        System.out.println("Exiting...");
        scanner.close();
        System.exit(0);
    default:
        System.out.println("Invalid option! Please try again.");
}
}
}

private static void addEmployee(Scanner scanner) {
    System.out.print("Enter Employee ID: ");
    int id = scanner.nextInt();
    scanner.nextLine();
    System.out.print("Enter Name: ");
    String name = scanner.nextLine();
    System.out.print("Enter Designation: ");
    String designation = scanner.nextLine();
    System.out.print("Enter Salary: ");
```

```
        double salary = scanner.nextDouble();
        Employee emp = new Employee(id, name, designation, salary);
        employeeList.add(emp);
        saveEmployees();
        System.out.println("Employee added successfully!");
    }
    private static void displayAllEmployees() {
        if (employeeList.isEmpty()) {
            System.out.println("No employees found!");
            return;
        }
        System.out.println("\nEmployee Details:");
        for (Employee emp : employeeList) {
            emp.display();
        }
    }
    private static void loadEmployees() {
        try (ObjectInputStream ois = new ObjectInputStream(new
        FileInputStream(FILE_NAME))) {
            employeeList = (List<Employee>) ois.readObject();
        } catch (FileNotFoundException e) {
            System.out.println("No existing employee data found.");
        } catch (IOException | ClassNotFoundException e) {
            System.out.println("Error loading employee data: " +
e.getMessage());
        }
    }
    private static void saveEmployees() {
        try (ObjectOutputStream oos = new ObjectOutputStream(new
        FileOutputStream(FILE_NAME))) {
            oos.writeObject(employeeList);
        } catch (IOException e) {
```

```
        System.out.println("Error saving employee data: " +  
e.getMessage());  
    }  
}  
}
```

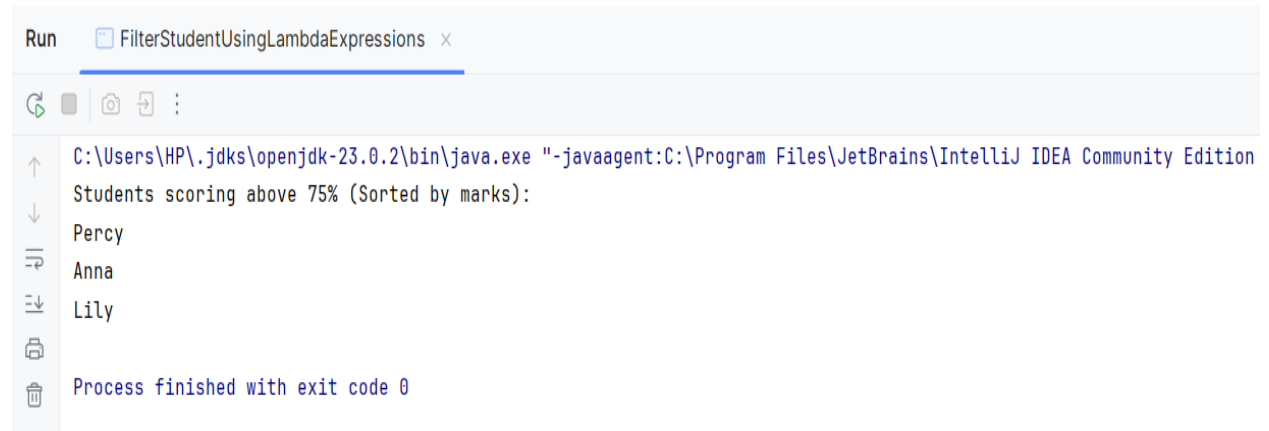
3. Output:

A-



```
SortEmployeesUsingLambdaExpression x  
C:\Users\HP\.jdk\openjdk-23.0.2\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition  
Sorted by Name:  
Anna | Age: 30 | Salary: 50000.0  
Lily | Age: 35 | Salary: 55000.0  
Robert | Age: 25 | Salary: 60000.0  
  
Sorted by Age:  
Robert | Age: 25 | Salary: 60000.0  
Anna | Age: 30 | Salary: 50000.0  
Lily | Age: 35 | Salary: 55000.0  
  
Sorted by Salary:  
Anna | Age: 30 | Salary: 50000.0  
Lily | Age: 35 | Salary: 55000.0  
Robert | Age: 25 | Salary: 60000.0  
  
Process finished with exit code 0
```

B-



```
Run FilterStudentUsingLambdaExpressions x  
C:\Users\HP\.jdk\openjdk-23.0.2\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition  
Students scoring above 75% (Sorted by marks):  
Percy  
Anna  
Lily  
  
Process finished with exit code 0
```

C-

```
ProcessingDatasetsUsingStreams x
C:\Users\HP\.jdk\openjdk-23.0.2\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition
Products Grouped by Category:
Clothing -> [Shirt (Rs 999.99), Jeans (Rs 1599.75), Jacket (Rs 2269.0)]
Electronics -> [Laptop (Rs 98500.5), Phone (Rs 28500.75), TV (Rs 75000.0)]
Furniture -> [Sofa (Rs 2500.25), Table (Rs 800.99), Chair (Rs 520.5)]

Most Expensive Product in Each Category:
Clothing -> Jacket (Rs 2269.0)
Electronics -> Laptop (Rs 98500.5)
Furniture -> Sofa (Rs 2500.25)

Average Price of All Products: Rs 23410.192222222224
```

4. Time Complexity:

A- The sorting operation used here is based on `Collections.sort()`, which internally uses TimSort, having a time complexity of $O(N \log N)$.

B- Filtering (filter): $O(N) \rightarrow$ Iterates over the list once. Sorting (sorted): $O(N \log N) \rightarrow$ Uses Java's TimSort algorithm. Mapping (map): $O(N) \rightarrow$ Extracts names. Collecting (collect): $O(N) \rightarrow$ Collects results into a list. Overall Complexity: $O(N \log N)$ due to sorting.

C- The time complexity of each operation is $O(n)$, as each product is processed once for categorization, max computation, and average price calculation, making the overall complexity $O(n)$, where n is the number of products.

5. Learning Outcomes

A- Learnt how to use lambda expressions to simplify sorting logic in Java.

B- Learnt how to use lambda expressions and Java Stream API to perform filtering, sorting, and mapping efficiently in a functional programming style.



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

- C- Learnt how to use Streams for filtering, grouping, and aggregating data efficiently.
- D- Learnt how to group, find max, and calculate averages using `Collectors.groupingBy()`, `maxBy()`, and `average()`.
- E- Developing a console based application.