



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Experiment 6

Student Name: Rohan Kumar Shah

UID: 22BCS11018

Branch: BE-CSE

Section/Group: IOT-642-B

Semester: 6th

Date of Performance: 3rd March, 25

Subject Name: Project Based Learning
in Java with Lab

Subject Code: 22CSH-359

Program 1: Lambda Expression

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.

Procedures:

Step 1: Create the Employee Class

-Define an Employee class with the following attributes:

name (String)

age (int)

salary (double)

-Create a constructor to initialize these values.

-Implement a display() method to print employee details.

Step 2: Create the Main Class

-Initialize an ArrayList<Employee> and add sample employee data.

-Use lambda expressions to sort the list:

Sort by Name (Alphabetical order)

Sort by Age (Ascending order)

Sort by Salary (Descending order)

Step 3: Display the Sorted List

Use forEach() with a method reference to print the sorted employees.

Test cases:

Program/Code:

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

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



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

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 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();  
  
        for (int i = 0; i < n; i++) {  
            System.out.println("Enter details for Employee " + (i + 1) + " (name, age,  
salary):");  
            String name = scanner.nextLine();  
            int age = scanner.nextInt();  
            double salary = scanner.nextDouble();  
            scanner.nextLine();  
            employees.add(new Employee(name, age, salary));  
        }  
  
        System.out.println("\nSorting by Name:");  
        employees.sort((e1, e2) -> e1.name.compareTo(e2.name));  
        employees.forEach(Employee::display);  
  
        System.out.println("\nSorting by Age:");  
        employees.sort((e1, e2) -> Integer.compare(e1.age, e2.age));  
        employees.forEach(Employee::display);  
  
        System.out.println("\nSorting by Salary (Descending):");
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
employees.sort((e1, e2) -> Double.compare(e2.salary, e1.salary));
employees.forEach(Employee::display);
}
}
```

Output:

```
Enter number of employees: 5
Enter details for Employee 1 (name, age, salary):
Farhat
23
500000
Enter details for Employee 2 (name, age, salary):
Asya
24
300000
Enter details for Employee 3 (name, age, salary):
Hajra
25
200000
Enter details for Employee 4 (name, age, salary):
Yassir
26
100000
Enter details for Employee 5 (name, age, salary):
Ragheb
27
100000
```

```
Sorting by Name:
Asya (Age: 24, Salary: 300000.0)
Farhat (Age: 23, Salary: 500000.0)
Hajra (Age: 25, Salary: 200000.0)
Ragheb (Age: 27, Salary: 100000.0)
Yassir (Age: 26, Salary: 100000.0)
```

```
Sorting by Age:
Farhat (Age: 23, Salary: 500000.0)
Asya (Age: 24, Salary: 300000.0)
Hajra (Age: 25, Salary: 200000.0)
Yassir (Age: 26, Salary: 100000.0)
Ragheb (Age: 27, Salary: 100000.0)
```

```
Sorting by Salary (Descending):
Farhat (Age: 23, Salary: 500000.0)
Asya (Age: 24, Salary: 300000.0)
Hajra (Age: 25, Salary: 200000.0)
Yassir (Age: 26, Salary: 100000.0)
Ragheb (Age: 27, Salary: 100000.0)
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Program 2: Lambda Expression

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

Procedure:

Test cases:

Program/Code:

```
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 void display() {
        System.out.println(name + " (Marks: " + marks + ")");
    }
}

public class StudentFilterSort {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        List<Student> students = new ArrayList<>();

        System.out.print("Enter number of students: ");
        int n = scanner.nextInt();
        scanner.nextLine();

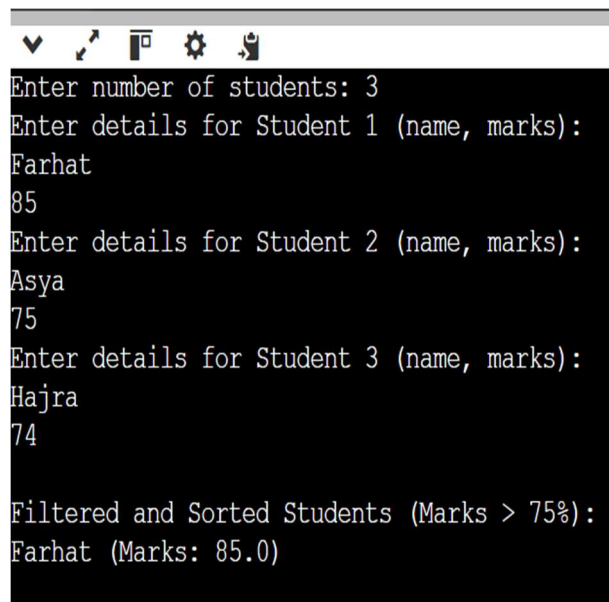
        for (int i = 0; i < n; i++) {
```

```
System.out.println("Enter details for Student " + (i + 1) + " (name, marks):");
String name = scanner.nextLine();
double marks = scanner.nextDouble();
scanner.nextLine();
students.add(new Student(name, marks));
}

System.out.println("\nFiltered and Sorted Students (Marks > 75%):");
List<Student> filteredStudents = students.stream()
    .filter(s -> s.marks > 75)
    .sorted(Comparator.comparingDouble((Student s) -> s.marks).reversed()
        .thenComparing(s -> s.name))
    .collect(Collectors.toList());

if (filteredStudents.isEmpty()) {
    System.out.println("No students scored above 75%.");
} else {
    filteredStudents.forEach(Student::display);
}
}
}
```

Output:



```
Enter number of students: 3
Enter details for Student 1 (name, marks):
Farhat
85
Enter details for Student 2 (name, marks):
Asya
75
Enter details for Student 3 (name, marks):
Hajra
74

Filtered and Sorted Students (Marks > 75%):
Farhat (Marks: 85.0)
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Program 3: Stream Class

Aim: 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.

Procedure:

Test cases:

Program/Code:

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



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
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 void display() {
        System.out.println(name + " (Category: " + category + ", Price: " + price + ")");
    }
}

public class ProductProcessor {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        List<Product> products = new ArrayList<>();

        System.out.print("Enter number of products: ");
        int n = scanner.nextInt();
        scanner.nextLine();

        for (int i = 0; i < n; i++) {
            System.out.println("Enter details for Product " + (i + 1) + " (name, category, price):");
            String name = scanner.nextLine();
            String category = scanner.nextLine();
            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));

        System.out.println("\nProducts Grouped by Category:");
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
groupByCategory.forEach((category, productList) -> {  
    System.out.println(category + ":");  
    productList.forEach(Product::display);  
});
```

```
Map<String, Optional<Product>> mostExpensiveByCategory = products.stream()  
    .collect(Collectors.groupingBy(p -> p.category,  
        Collectors.maxBy(Comparator.comparingDouble(p -> p.price))));
```

```
System.out.println("\nMost Expensive Product in Each Category:");  
mostExpensiveByCategory.forEach((category, product) ->  
    System.out.println(category + ": " + product.map(p -> p.name + " (" + p.price  
+ ")").orElse("No product")));
```

```
double averagePrice = products.stream()  
    .collect(Collectors.averagingDouble(p -> p.price));  
System.out.println("\nAverage Price of All Products: " + averagePrice);  
}  
}
```

Output:

```
Enter number of products: 5  
Enter details for Product 1 (name, category, price):  
Pencil  
Stationary  
20  
Enter details for Product 2 (name, category, price):  
Ruler  
Stationary  
50  
Enter details for Product 3 (name, category, price):  
Shirt  
Clothe  
500  
Enter details for Product 4 (name, category, price):  
Scarf  
Clothe  
300  
Enter details for Product 5 (name, category, price):  
Biscuit  
Snack  
70
```

```
Products Grouped by Category:  
Clothe:  
Shirt (Category: Clothe, Price: 500.0)  
Scarf (Category: Clothe, Price: 300.0)  
Stationary:  
Pencil (Category: Stationary, Price: 20.0)  
Ruler (Category: Stationary, Price: 50.0)  
Snack:  
Biscuit (Category: Snack, Price: 70.0)  
  
Most Expensive Product in Each Category:  
Clothe: Shirt (500.0)  
Stationary: Ruler (50.0)  
Snack: Biscuit (70.0)  
  
Average Price of All Products: 188.0
```