Experiment -6

Student Name: Shivangi **UID: 22BCS16953**

Branch: BE-CSE **Date of Performance: 17-03-2025**

Semester: 6th **Section/Group: 22BCS_EPAM-801/ B**

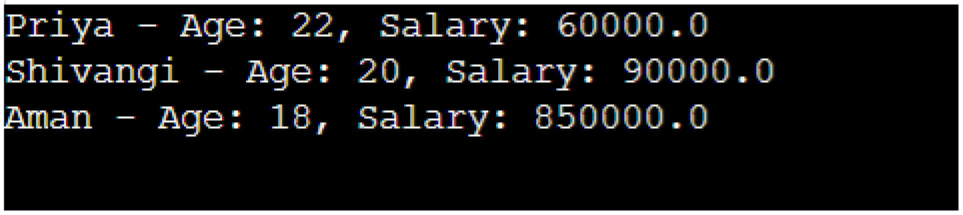
Subject Name: Project based learning in java with lab **Subject Code: 22CSH-359**

1. **Aim:** Develop Java programs using lambda expressions and stream operations for sorting, filtering, and processing large datasets efficiently. a.Write a program to sort a list of Employee objects (name, age, salary) using lambda expressions. b. 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. 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. **Implementation/Code:**

Easy Level:

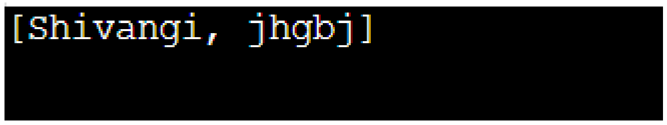
```
import java.util.*;
class Employee {
    String name;
    int age;
    double salary;
    public Employee(String name, int age, double salary) {
        this.name = name;
        this.age = age;
        this.salary = salary;
    }
    public String toString() {
        return name + " - Age: " + age + ", Salary: " + salary;
    }
}
public class EmployeeSort {
    public static void main(String[] args) {
        List<Employee> employees = Arrays.asList(
            new Employee("Shivangi", 20, 90000),
            new Employee("Priya", 22, 60000),
            new Employee("Aman", 18, 850000)
        );
        employees.sort(Comparator.comparingDouble(e -> e.salary));

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



```
Priya - Age: 22, Salary: 60000.0
Shivangi - Age: 20, Salary: 90000.0
Aman - Age: 18, Salary: 850000.0
```

Medium Level:

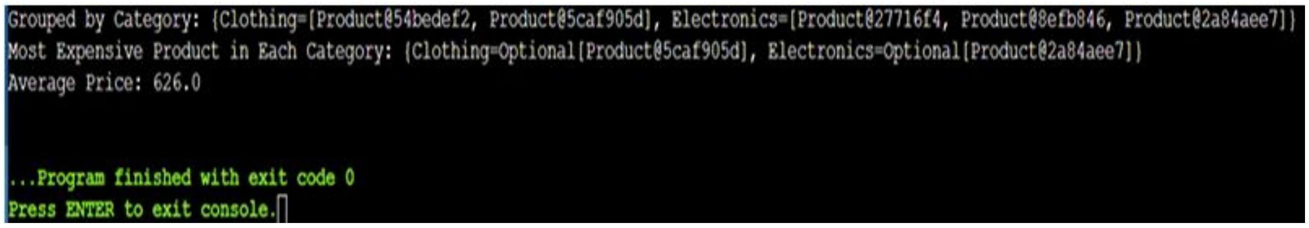
```
import java.util.*;
import java.util.stream.Collectors;

class Student {
    String name;
    double marks;
    public Student(String name, double marks) {
        this.name = name;
        this.marks = marks;
    }
}
public class StudentFilter {
    public static void main(String[] args) {
        List<Student> students = Arrays.asList(
            new Student("Shivangi", 90),
            new Student("Priaa", 65),
            new Student("abvdhd", 45),
            new Student("jhgbj", 80)
        );
        List<String> topStudents = students.stream()
            .filter(s -> s.marks > 75)
            .sorted(Comparator.comparingDouble(s -> -s.marks))
            .map(s -> s.name)
            .collect(Collectors.toList());
        System.out.println(topStudents);
    }
}
```



```
[Shivangi, jhgbj]
```

Hard Level:

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

class Product {
    String name, category;
    double price;

    public Product(String name, String category, double price) {
        this.name = name;
        this.category = category;
        this.price = price;
    }

    @Override
    public String toString() {
        return String.format("Product{name='%s', category='%s', price=%.2f}", name, category,
price);
    }
}

public class ProductProcessing {
    public static void main(String[] args) {
        List<Product> products = Arrays.asList(
            new Product("Laptop", "Electronics", 1000),
            new Product("Phone", "Electronics", 800),
            new Product("TV", "Electronics", 1200),
            new Product("Shirt", "Clothing", 50),
            new Product("Jeans", "Clothing", 80)
        );

        Map<String, List<Product>> groupedByCategory = products.stream()
            .collect(Collectors.groupingBy(Product::category));

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

        double avgPrice = products.stream()
            .mapToDouble(p -> p.price)
            .average()
            .orElse(0);

        System.out.println("Grouped by Category: " + groupedByCategory);
        System.out.println("Most Expensive Product in Each Category: " +
mostExpensiveByCategory);
        System.out.println("Average Price: " + avgPrice);
    }
}
```



```
Grouped by Category: {Clothing=[Product@54bedef2, Product@5caf905d], Electronics=[Product@27716f4, Product@8efb846, Product@2a84aee7]}
Most Expensive Product in Each Category: {Clothing=Optional[Product@5caf905d], Electronics=Optional[Product@2a84aee7]}
Average Price: 626.0


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

3. Learning Outcomes:

- Using Lambda Expressions – Learned how to use lambda expressions to sort and filter lists efficiently.
- Working with Streams – Understood how Java Streams help process large datasets quickly using filtering, sorting, and mapping.
- Grouping and Aggregation – Learned how to group data by categories, find the most expensive product, and calculate average prices using Streams.
- Efficient Data Processing – Gained experience in writing clean and optimized Java code for handling real-world datasets with functional programming techniques.