Experiment 6

Student Name: Ranjeet Singh UID: 22BCS10668

Branch: BE-CSE Section/Group: IOT-637/A

Semester:6th Date of Performance: 28/02/2025

Subject Name: Project Based Learning in Java Subject Code: 22CSH-359

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

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

   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.

   Hard Level: Write a Java program to process a large dataset of products using operations such as grouping products by category, finding the most expensive and calculating the average price of all products.

Easy Level

Code:

```
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 + ", rup " + salary;
    }
}
```

```
public class SortEmployeesLambda {
    public static void main(String[] args) {
        List<Employee> employees = Arrays.asList(
            new Employee("Ranjeet ", 30, 50000),
            new Employee("Alok", 25, 70000),
            new Employee("Jahid", 35, 60000),
            new Employee("Jibin", 40, 80000),
            new Employee("Edison", 29, 75000),
            new Employee("Yash", 28, 65000)

        );

        // Sorting by name
        employees.sort(Comparator.comparing(emp -> emp.name));
        System.out.println("Sorted by name: " + employees);

        // Sorting by age
        employees.sort(Comparator.comparing(emp -> emp.age));
        System.out.println("Sorted by age: " + employees);

        // Sorting by salary
        employees.sort(Comparator.comparing(emp -> emp.salary));
        System.out.println("Sorted by salary: " + employees);
    }
}
```

Output:

```
Sorted by name: [Alok, 25, rup 70000.0, Edison, 29, rup 75000.0, Jahid, 35, rup 600
Sorted by age: [Alok, 25, rup 70000.0, Yash, 28, rup 65000.0, Edison, 29, rup 75000
Sorted by salary: [Ranjeet , 30, rup 50000.0, Jahid, 35, rup 60000.0, Yash, 28, rup
```

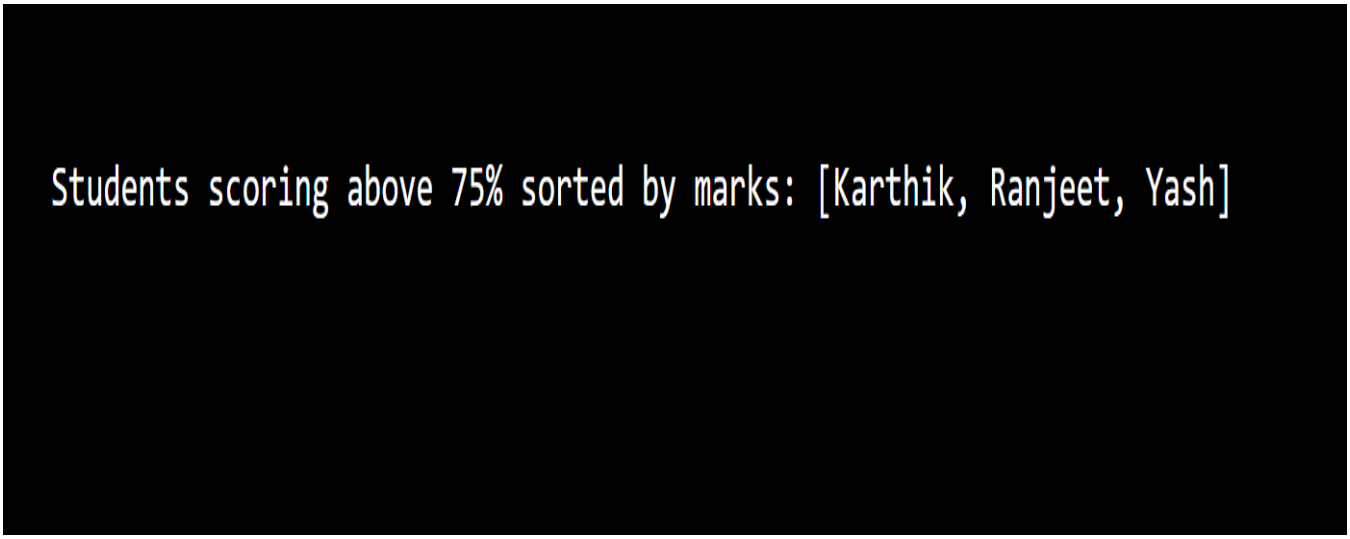
MEDIUM LEVEL:

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;
    }

    @Override
    public String toString() {
        return name + " - " + marks + "%";
    }
}

public class FilterAndSortStudents {
    public static void main(String[] args) {
        List<Student> students = Arrays.asList(
            new Student("Ranjeet", 85),
            new Student("Alok", 72),
            new Student("Yash", 91),
            new Student("Gaurav", 65),
            new Student("Karthik", 79)
        );


        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("Students scoring above 75% sorted by marks: " + topStudents);
    }
```

```
}
```

OUTPUT:



HARD LEVEL:
CODE:

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

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;
    }

    @Override
    public String toString() {
        return name + " - " + category + " - $" + price;
    }
}
```

```
public class ProcessProducts {
    public static void main(String[] args) {
        List<Product> products = Arrays.asList(
            new Product("Laptop", "Electronics", 1200),
            new Product("Phone", "Electronics", 800),
            new Product("Tablet", "Electronics", 600),
            new Product("Chair", "Furniture", 150),
            new Product("Desk", "Furniture", 300),
            new Product("Sofa", "Furniture", 700),
            new Product("Headphones", "Accessories", 100),
            new Product("Mouse", "Accessories", 50),
            new Product("Keyboard", "Accessories", 80)
        );


        Map<String, List<Product>> productsByCategory = products.stream()
            .collect(Collectors.groupingBy(p -> p.category));

        System.out.println("Products grouped by category: " + productsByCategory);


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

        System.out.println("Most expensive product in each category: " +
mostExpensiveByCategory);


        double averagePrice = products.stream()
            .mapToDouble(p -> p.price)
            .average()
            .orElse(0.0);

        System.out.println("Average price of all products: $" + averagePrice);
    }
}
```

OUTPUT:

```
Products grouped by category: {Accessories=[Headphones - Accessories - $100.0, Mo
Most expensive product in each category: {Accessories=Optional[Headphones - Acces
Average price of all products: $442.22222222222223
```

4. **Learning Outcomes:**
    1. Learn how to perform basic CRUD (Create, Read, Update, Delete) operations on a List of String objects in Java.
    2. Understand how to use the ArrayList class for dynamically storing and manipulating a collection of items.
    3. Practice handling user input using the Scanner class for interaction with the program.
    4. Implement methods for Sorting deleting , and displaying items in a list efficiently.
    5. Gain familiarity with control flow and loops to allow for continuous user interaction until the program is exited.