Experiment 6

Student Name: Praburam M
UID: 22BCS16537
Branch: BE-CSE
Section/Group: DL_903_A
Semester: 6th
Date of Performance: 18-02-2025
Subject Name: Program 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 in Easy, Medium and Hard

2. Objective: Understanding the concept of using lambda expressions and stream operations for sorting, filtering, and processing large datasets efficiently in Java.

3. Implementation/Code:

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

   Code:

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

// Employee class
class Employee {
    private String name;
    private int age;
    private double salary;

    // Constructor
    public Employee(String name, int age, double salary) {
        this.name = name;
        this.age = age;
        this.salary = salary;
    }

    // Getters
    public String getName() {
        return name;
```

```
    }

    public int getAge() {
        return age;
    }

    public double getSalary() {
        return salary;
    }

    // Display method
    public void display() {
        System.out.println("Name: " + name + ", Age: " + age + ", Salary: $" + salary);
    }
}

public class EmployeeSorter {

    public static void main(String[] args) {
        List<Employee> employees = new ArrayList<>();

        // Adding Employee objects
        employees.add(new Employee("Alice", 30, 60000));
        employees.add(new Employee("Bob", 25, 50000));
        employees.add(new Employee("Charlie", 35, 70000));
        employees.add(new Employee("David", 28, 55000));

        // Sorting by Name using Lambda Expression
        Collections.sort(employees, (e1, e2) -> e1.getName().compareTo(e2.getName()));
        System.out.println("Sorted by Name:");
        employees.forEach(Employee::display);

        // Sorting by Age using Lambda Expression
        Collections.sort(employees, (e1, e2) -> Integer.compare(e1.getAge(), e2.getAge()));
        System.out.println("\nSorted by Age:");
        employees.forEach(Employee::display);

        // Sorting by Salary using Lambda Expression
        Collections.sort(employees, (e1, e2) -> Double.compare(e1.getSalary(), e2.getSalary()));
        System.out.println("\nSorted by Salary:");
        employees.forEach(Employee::display);
    }
}
```

2. 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.

Code:

```
import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;
import java.util.stream.Collectors;

// Student class
class Student {
    private String name;
    private double marks;

    // Constructor
    public Student(String name, double marks) {
        this.name = name;
        this.marks = marks;
    }

    // Getters
    public String getName() {
        return name;
    }

    public double getMarks() {
        return marks;
    }
}

public class Main {
    public static void main(String[] args) {
        List<Student> students = new ArrayList<>();

        // Adding student objects
        students.add(new Student("Alice", 82.5));
        students.add(new Student("Bob", 68.0));
        students.add(new Student("Charlie", 91.0));
        students.add(new Student("David", 74.0));
        students.add(new Student("Eva", 78.0));

        // Using Streams to filter, sort, and display
        List<Student> topScorers = students.stream()
                .filter(student -> student.getMarks() > 75) // Filter students scoring above 75%
                .sorted(Comparator.comparingDouble(Student::getMarks).reversed()) // Sort by marks descending
                .collect(Collectors.toList()); // Collect results to a list

        // Display the filtered and sorted students
        System.out.println("Students scoring above 75% (sorted by marks):");
        topScorers.forEach(student ->
```

```
            System.out.println("Name: " + student.getName() + ", Marks: " + student.getMarks())
        );
    }
}
```

3. 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.

   Code:

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

// Product class
class Product {
    private String name;
    private String category;
    private double price;

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

    // Getters
    public String getName() {
        return name;
    }

    public String getCategory() {
        return category;
    }

    public double getPrice() {
        return price;
    }
```

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

public class Main {
    public static void main(String[] args) {
        List<Product> products = Arrays.asList(
            new Product("Laptop", "Electronics", 1200),
            new Product("Smartphone", "Electronics", 800),
            new Product("TV", "Electronics", 1500),
            new Product("Blender", "Home Appliances", 200),
            new Product("Microwave", "Home Appliances", 300),
            new Product("Refrigerator", "Home Appliances", 1000),
            new Product("T-Shirt", "Clothing", 30),
            new Product("Jeans", "Clothing", 60),
            new Product("Jacket", "Clothing", 150)
        );

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

        System.out.println("🔍 Products grouped by category:");
        groupedByCategory.forEach((category, productList) -> {
            System.out.println("Category: " + category);
            productList.forEach(product -> System.out.println("  - " + product));
        });

        // Find the most expensive product in each category
        Map<String, Optional<Product>> mostExpensiveByCategory = products.stream()
                .collect(Collectors.groupingBy(
                        Product::getCategory,
                        Collectors.maxBy(Comparator.comparingDouble(Product::getPrice))
                ));

        System.out.println("\n💰 Most expensive product in each category:");
```

```
        mostExpensiveByCategory.forEach((category, product) -> {
            product.ifPresent(p -> System.out.println("Category: " + category + " -> " + p));
        });

        // 3️⃣ Calculate the average price of all products
        double averagePrice = products.stream()
                .collect(Collectors.averagingDouble(Product::getPrice));

        System.out.println("\n📊 Average price of all products: $" + String.format("%.2f", averagePrice));
    }
}
```

5. Output

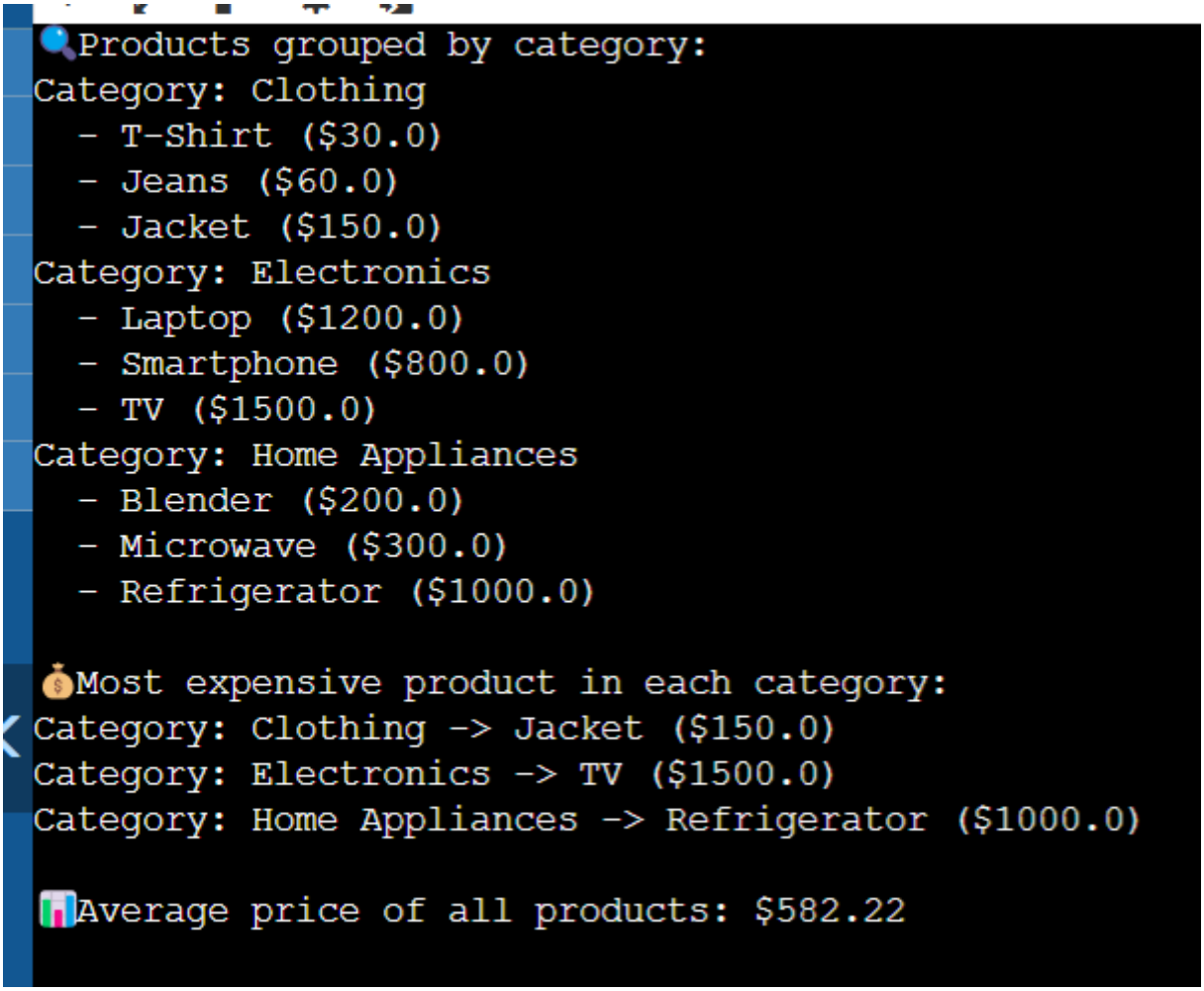
1.) Easy problem: Sum of entered integer

```
Sorted by Name:
Name: Alice, Age: 30, Salary: $60000.0
Name: Bob, Age: 25, Salary: $50000.0
Name: Charlie, Age: 35, Salary: $70000.0
Name: David, Age: 28, Salary: $55000.0

Sorted by Age:
Name: Bob, Age: 25, Salary: $50000.0
Name: David, Age: 28, Salary: $55000.0
Name: Alice, Age: 30, Salary: $60000.0
Name: Charlie, Age: 35, Salary: $70000.0

Sorted by Salary:
Name: Bob, Age: 25, Salary: $50000.0
Name: David, Age: 28, Salary: $55000.0
Name: Alice, Age: 30, Salary: $60000.0
Name: Charlie, Age: 35, Salary: $70000.0
```

2.) Medium problem: ATM System

```
Students scoring above 75% (sorted by marks):
Name: Charlie, Marks: 91.0
Name: Alice, Marks: 82.5
Name: Eva, Marks: 78.0


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

3.) Hard problem:



6. Learning Outcomes

1. Learned to use classes and objects for organizing employee and designation data in Java.
2. Implemented salary calculations using switch-case and array data handling.
3. Practiced input handling with the Scanner class and validating user input.
4. Gained experience in searching arrays and structuring conditional logic.
5. Displayed formatted output for real-world applications like employee management systems.