Experiment 6

Student Name: Sameer Gautam **UID: 22BCS10410**

Branch: CSE **Section:618(B)**

Semester: 6th **DOP:28/02/2025**

Subject: Java **Subject Code: 22CSH-359**

Problem Statement 1

1. **Aim:** Sorting Employees using Lambda Expressions: Develop a Java program that sorts a list of Employee objects (name, age, salary) using lambda expressions, ensuring efficient and concise sorting based on different attributes.

2. **Objective:** To apply lambda expressions and Java Streams for efficient data manipulation by filtering and sorting student records, processing large product datasets through grouping and analysis, and implementing functional sorting methods to enhance understanding of functional programming paradigms in Java.

3. **Code:**

```
package Experiments;

import java.util.*;

class Employee {

    String name;

    int age;

    double salary;

    Employee(String name, int age, double salary) {

        this.name = name;

        this.age = age;
```

```
        this.salary = salary;
    }
    @Override
    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("Alice", 30, 50000),
            new Employee("Bob", 25, 60000),
            new Employee("Charlie", 35, 55000)
        )
        // Sorting by age using lambda expression
        employees.sort(Comparator.comparingInt(emp -> emp.age));
        System.out.println("Sorted by Age:");
        employees.forEach(System.out::println);
    }
}
```

4. **Output:**

```
PS C:\Users\samro\Desktop\Codes\Java>  & 'C:\Program Files\Java\jdk
ta\Roaming\Code\User\workspaceStorage\bc6dc402e8476e39bdd4474ef73d4
Sorted by Age:
Bob - Age: 25, Salary: 60000.0
Alice - Age: 30, Salary: 50000.0
Charlie - Age: 35, Salary: 55000.0
PS C:\Users\samro\Desktop\Codes\Java>
```

Problem Statement 2

1. **Aim:** Implement a Java program that filters students scoring above 75%, sorts them by marks in descending order, and displays their names using lambda expressions and stream operations.

2. **Code:**

```
package Experiments;

import java.util.*;
import java.util.stream.*;

class Student {
    String name;
    double marks;

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

    public String getName() {
        return name;
    }

    public double getMarks() {
        return marks;
    }
}

public class StudentFilter {
```

```
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter number of students: ");
        int n = scanner.nextInt();
        scanner.nextLine(); // Consume newline

        List<Student> students = new ArrayList<>();
        for (int i = 0; i < n; i++) {
            System.out.print("Enter name of student " + (i + 1) + ": ");
            String name = scanner.nextLine();
            System.out.print("Enter marks of student " + (i + 1) + ": ");
            double marks = scanner.nextDouble();
            scanner.nextLine(); // Consume newline
            students.add(new Student(name, marks));
        }

        System.out.println("Students scoring above 75%, sorted by marks:");
        students.stream()
                .filter(s -> s.getMarks() > 75)
                .sorted(Comparator.comparingDouble(Student::getMarks).reversed())
                .map(Student::getName)
                .forEach(System.out::println);
    }
}
```

3. **Output:**

```
PS C:\Users\samro\Desktop\Codes\Java>  & 'C:\Program Fil
ta\Roaming\Code\User\workspaceStorage\bc6dc402e8476e39bc
Enter number of students: 3
Enter name of student 1: Sameer
Enter marks of student 1: 88
Enter name of student 2: Harsh
Enter marks of student 2: 82
Enter name of student 3: Ayush
Enter marks of student 3: 84
Students scoring above 75%, sorted by marks:
Sameer
Ayush
Harsh
PS C:\Users\samro\Desktop\Codes\Java>
```

Problem Statement 3

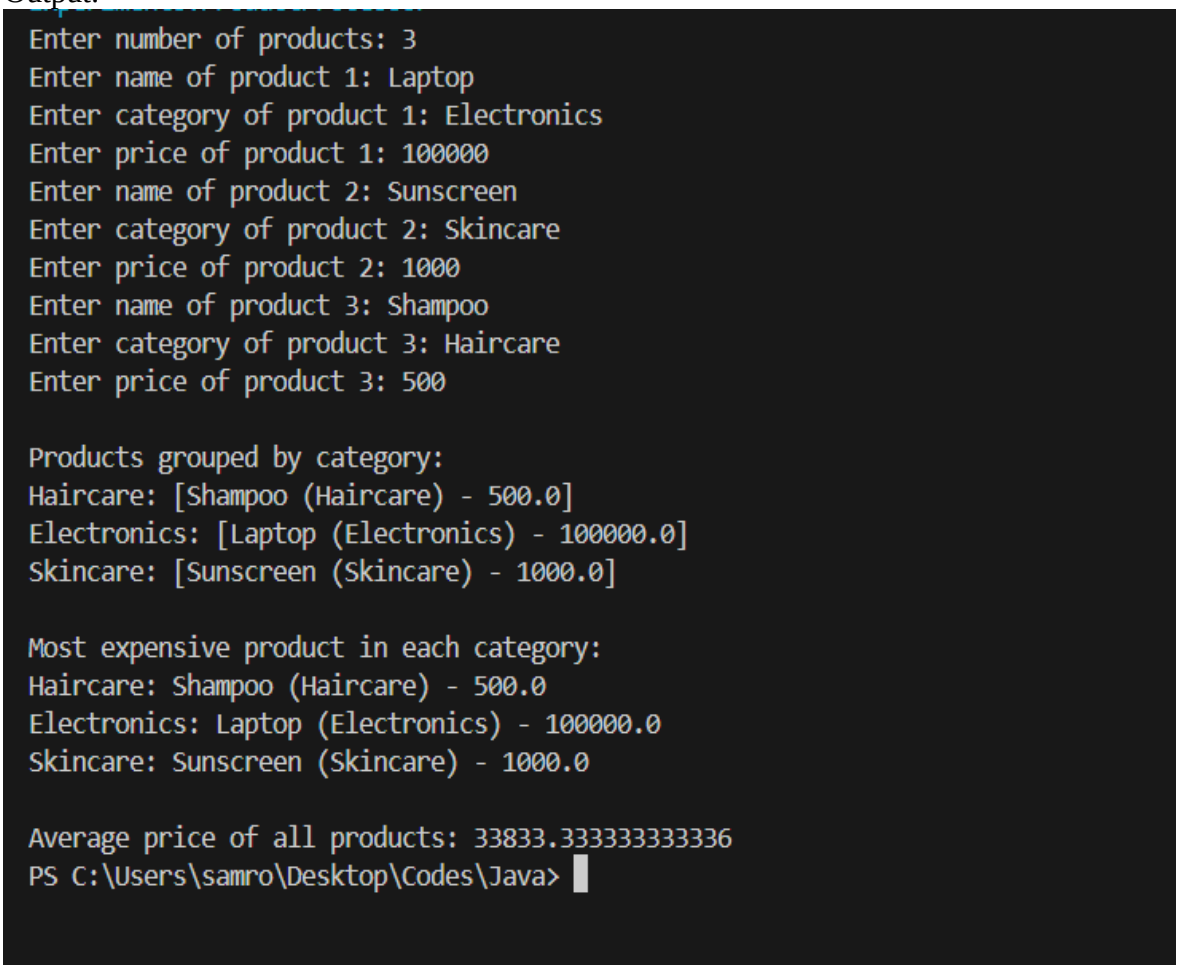
1. **Aim:** Create a Java program that processes a large dataset of products by grouping them based on categories, identifying the most expensive product in each category, and calculating the average price of all products using Java Streams.

2. **Code:**

```
package Experiments;

import java.util.*;
import java.util.stream.*;

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

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

    public String getCategory() {
        return category;
    }

    public double getPrice() {
        return price;
    }

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

public class ProductProcessor {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter number of products: ");
        int n = scanner.nextInt();
        scanner.nextLine(); // Consume newline

        List<Product> products = new ArrayList<>();
        for (int i = 0; i < n; i++) {
```

```
            System.out.print("Enter name of product " + (i + 1) + ": ");
            String name = scanner.nextLine();
            System.out.print("Enter category of product " + (i + 1) + ": ");
            String category = scanner.nextLine();
            System.out.print("Enter price of product " + (i + 1) + ": ");
            double price = scanner.nextDouble();
            scanner.nextLine(); // Consume newline
            products.add(new Product(name, category, price));
        }

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

        System.out.println("\nProducts grouped by category:");
        groupedByCategory.forEach((category, productList) -> {
            System.out.println(category + ": " + productList);
        });

        // Finding 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("\nMost expensive product in each category:");
        mostExpensiveByCategory.forEach((category, product) ->
                System.out.println(category + ": " + product.orElse(null)));

        // Calculating the average price of all products
        double averagePrice = products.stream()
                .mapToDouble(Product::getPrice)
                .average()
                .orElse(0.0);

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

3. Output:



4. **Learning Outcomes**

- Understand the use of Java Streams for efficient data processing and manipulation.
- Learn how to apply lambda expressions for filtering, sorting, and grouping data.
- Gain experience in handling user input dynamically and processing large datasets.
- Develop problem-solving skills by implementing real-world data operations in Java.