Experiment 6

Student Name: Japneet Kaur	**UID: 22BCS10390**
Branch: CSE	**Section:618(A)**
Semester: 6th	**DOP:28/02/2025**
Subject: Java	**Subject Code: 22CSH-359**

Problem Statement 1

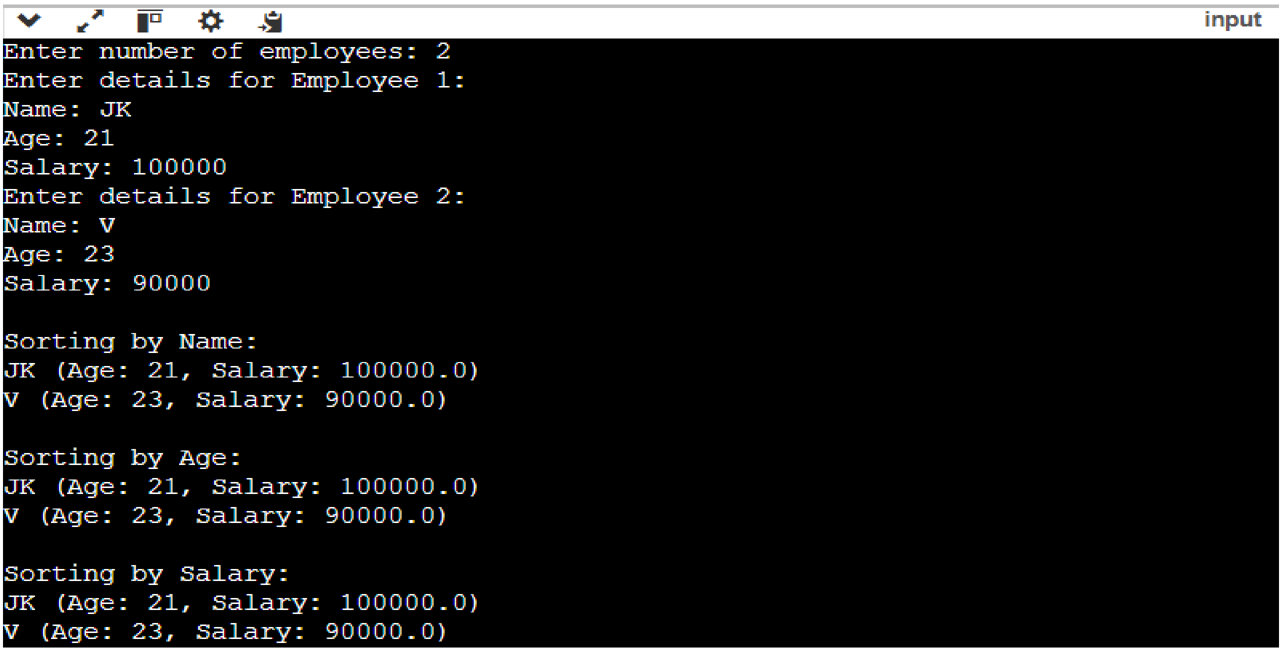
1. **Aim:** Write a program to sort a list of Employee objects (name, age, salary) using lambda expressions.

2. **Objective:** Use of Collections in Java. LinkedList, HashMap, HashSet in Java. Multithreading in Java. Thread Synchronization. Thread Priority, Thread lifecycle.

3. **Code:**

```
import java.util.*;
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 + " (Age: " + age + ", Salary: " + salary + ")");
    }
}
public class EmployeeSorter {
    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(); // Consume newline

        for (int i = 0; i < n; i++) {
            System.out.println("Enter details for Employee " + (i + 1) + ":");
            System.out.print("Name: ");
            String name = scanner.nextLine();
            System.out.print("Age: ");
            int age = scanner.nextInt();
            System.out.print("Salary: ");
            double salary = scanner.nextDouble();
            scanner.nextLine(); // Consume newline
            employees.add(new Employee(name, age, salary));
        }
        // Sorting by Name (Alphabetical order)
        System.out.println("\nSorting by Name:");
        employees.stream()
            .sorted(Comparator.comparing(Employee::getName))
            .forEach(Employee::display);
        // Sorting by Age (Ascending order)
        System.out.println("\nSorting by Age:");
        employees.stream()
            .sorted(Comparator.comparingInt(Employee::getAge))
            .forEach(Employee::display);
        // Sorting by Salary (Descending order)
        System.out.println("\nSorting by Salary:");
        employees.stream()
            .sorted(Comparator.comparingDouble(Employee::getSalary).reversed())
            .forEach(Employee::display);
        scanner.close();
    }
}
```

4. Output:



```
Enter number of employees: 2
Enter details for Employee 1:
Name: JK
Age: 21
Salary: 100000
Enter details for Employee 2:
Name: V
Age: 23
Salary: 90000

Sorting by Name:
JK (Age: 21, Salary: 100000.0)
V (Age: 23, Salary: 90000.0)

Sorting by Age:
JK (Age: 21, Salary: 100000.0)
V (Age: 23, Salary: 90000.0)

Sorting by Salary:
JK (Age: 21, Salary: 100000.0)
V (Age: 23, Salary: 90000.0)
```

Problem Statement 2

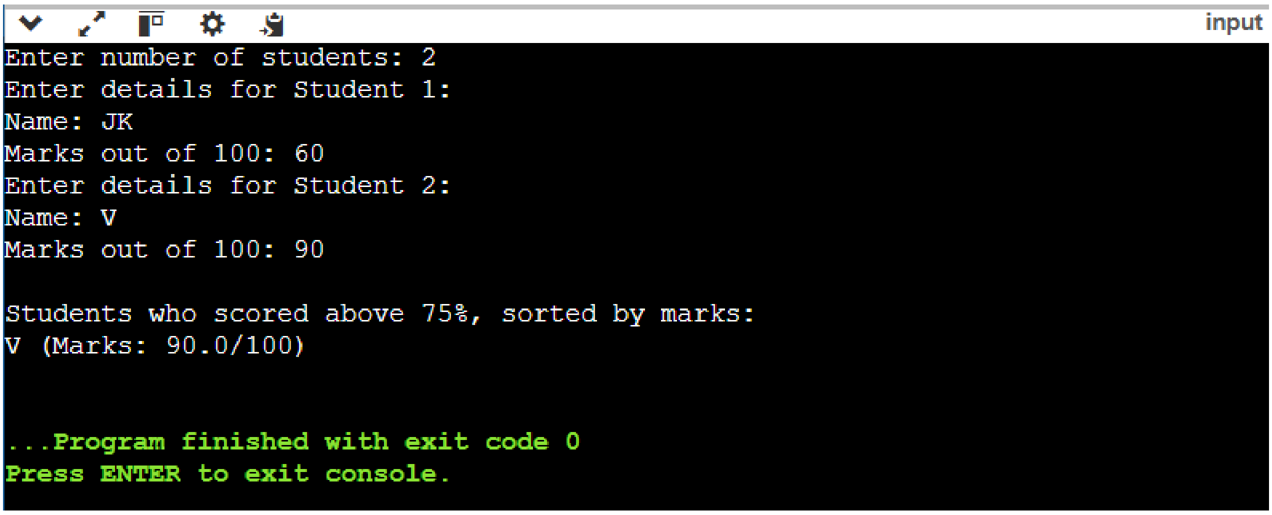
1. **Aim:** Create a program to use lambda expressions and stream operations to filter students scoring above 75%, sort them by marks, and display their names.

2. **Code:**

```
import java.util.*;
import java.util.stream.Collectors;
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;
    }
    // Display method
    public void display() {
        System.out.println(name + " (Marks: " + marks +
"/100)");
    }
}

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(); // Consume newline

        for (int i = 0; i < n; i++) {
    System.out.println("Enter details for Student " + (i + 1) +
    ":");
            System.out.print("Name: ");
            String name = scanner.nextLine();
            System.out.print("Marks out of 100: ");
            double marks = scanner.nextDouble();
            scanner.nextLine(); // Consume newline

            students.add(new Student(name, marks));
        }

        // Filtering students who scored above 75% and
    sorting by marks (descending order)
    List<Student> filteredSortedStudents = students.stream()
       .filter(s -> s.getMarks() > 75)

    .sorted(Comparator.comparingDouble(Student::getMarks).
    reversed()
 .thenComparing(Student::getName)) // Sort by name if
marks are the same
              .collect(Collectors.toList());

        // Displaying results
        if (filteredSortedStudents.isEmpty()) {
            System.out.println("No students scored above
    75%.");
        } else {
            System.out.println("\nStudents who scored above
    75%, sorted by marks:");
```

```
            filteredSortedStudents.forEach(Student::display);
        }
        scanner.close();
    }
}
```

3. Output:



```
Enter number of students: 2
Enter details for Student 1:
Name: JK
Marks out of 100: 60
Enter details for Student 2:
Name: V
Marks out of 100: 90

Students who scored above 75%, sorted by marks:
V (Marks: 90.0/100)


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

Problem Statement 3

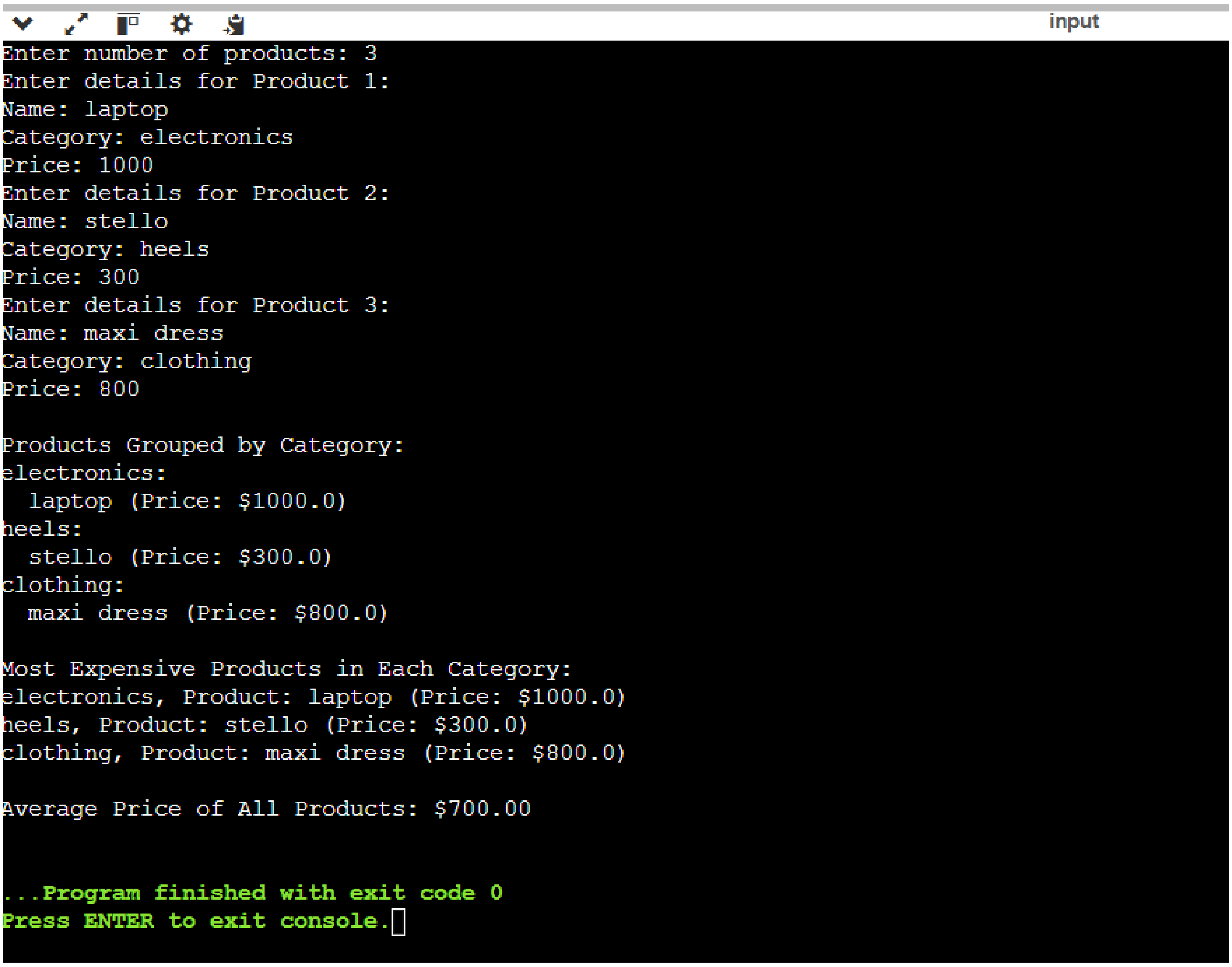
1. **Aim:** Write a Java program to process a large dataset of products using streams. Perform operations such as grouping products by category, finding the most expensive product in each category, and calculating the average price of all products.

2. **Code:**

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

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: $" + 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(); // Consume newline

            for (int i = 0; i < n; i++) {
                System.out.println("Enter details for Product " + (i + 1) + ":");
                System.out.print("Name: ");
                String name = scanner.nextLine();
                System.out.print("Category: ");
                String category = scanner.nextLine();
                System.out.print("Price: ");
                double price = scanner.nextDouble();
                scanner.nextLine(); // Consume newline

                products.add(new Product(name, category, price));
            }
        // Grouping products by category
        Map<String, List<Product>> productsByCategory = products.stream()
                .collect(Collectors.groupingBy(Product::getCategory));


        // 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))
                ));

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

        // Displaying results
        System.out.println("\nProducts Grouped by Category:");
        productsByCategory.forEach((category, productList) -> {
            System.out.println(category + ":");
            productList.forEach(product -> System.out.println("  " + product));
        });

        System.out.println("\nMost Expensive Products in Each Category:");
        mostExpensiveByCategory.forEach((category, product) ->
            System.out.println(category + ", Product: " + product.orElse(null))
        );

        System.out.println("\nAverage Price of All Products: $" + String.format("%.2f",
averagePrice));

        scanner.close();
    }
}
```

3. Output:



```
Enter number of products: 3
Enter details for Product 1:
Name: laptop
Category: electronics
Price: 1000
Enter details for Product 2:
Name: stello
Category: heels
Price: 300
Enter details for Product 3:
Name: maxi dress
Category: clothing
Price: 800

Products Grouped by Category:
electronics:
  laptop (Price: $1000.0)
heels:
  stello (Price: $300.0)
clothing:
  maxi dress (Price: $800.0)

Most Expensive Products in Each Category:
electronics, Product: laptop (Price: $1000.0)
heels, Product: stello (Price: $300.0)
clothing, Product: maxi dress (Price: $800.0)

Average Price of All Products: $700.00


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

4. Learning Outcomes

- Understanding Lambda Expressions – Learn to simplify code using lambda expressions for functional programming.

- Mastering Stream API – Use streams for sorting, filtering, and efficient data processing.

- Efficient Data Handling – Process large datasets with grouping, aggregation, and filtering operations.

- Working with Functional Interfaces – Utilize Comparator, Predicate, and method references to optimize code.