Experiment-6

Student Name: Rinku **UID:** 22BCS13370
Branch: BE-CSE **Section/Group:** 618-B
Semester: 6th **Date of Performance:** 28/02/25
Subject Name: Project Based Learning in Java **Subject Code:** 22CSH-359

EASY:

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

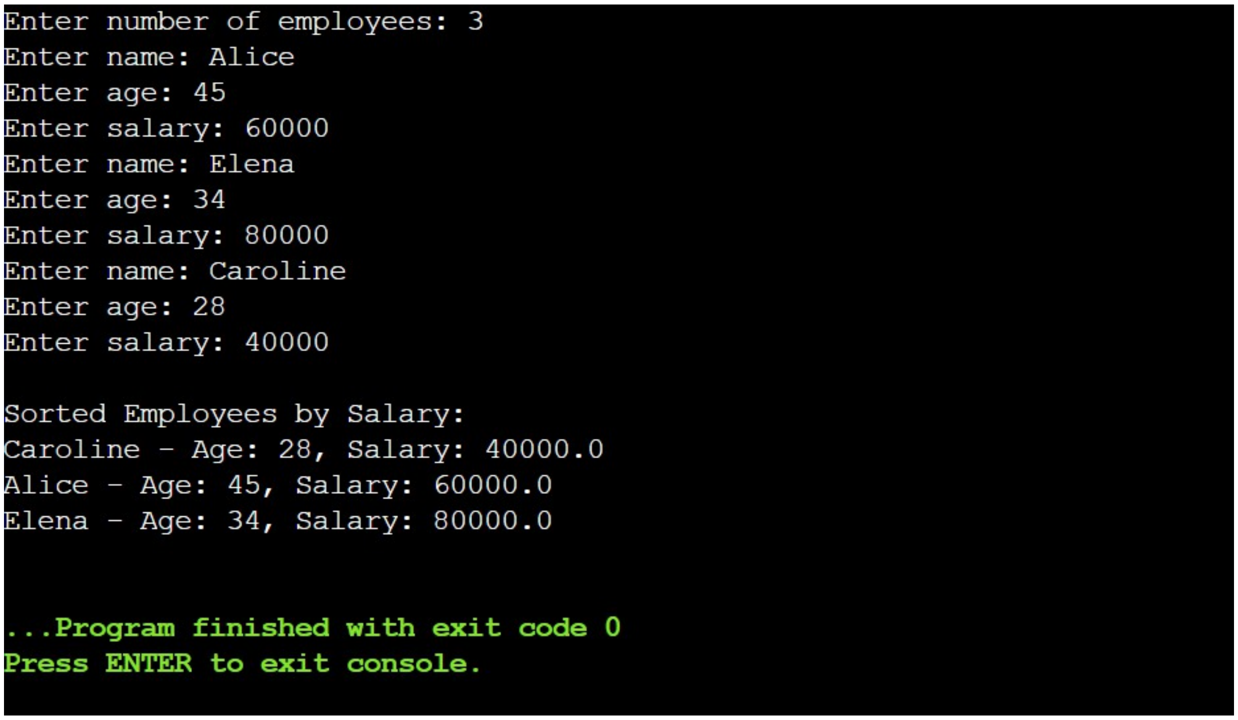
Objective: The objective is to implement lambda expressions for sorting a list of Employee objects by salary. This helps in understanding functional programming and the use of Comparator in Java.

Implementation/Code:

```
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 EmployeeSorting {
    public static void main(String[] args)
        { Scanner sc = new Scanner(System.in);
        List<Employee> employees = new ArrayList<>();

        System.out.print("Enter number of employees: ");
        int n = sc.nextInt();
        sc.nextLine();

        for (int i = 0; i < n; i++)
            { System.out.print("Enter name:
            "); String name = sc.nextLine();
            System.out.print("Enter age: ");
            int age = sc.nextInt();
            System.out.print("Enter salary: ");
            double salary = sc.nextDouble();
            sc.nextLine();

            employees.add(new Employee(name, age, salary));
        }
        employees.sort((e1, e2) -> Double.compare(e1.salary, e2.salary));

        System.out.println("\nSorted Employees by Salary:");
        employees.forEach(System.out::println);

        sc.close();
    }
}
```

Output



```
Enter number of employees: 3
Enter name: Alice
Enter age: 45
Enter salary: 60000
Enter name: Elena
Enter age: 34
Enter salary: 80000
Enter name: Caroline
Enter age: 28
Enter salary: 40000

Sorted Employees by Salary:
Caroline - Age: 28, Salary: 40000.0
Alice - Age: 45, Salary: 60000.0
Elena - Age: 34, Salary: 80000.0


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

MEDIUM:

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.

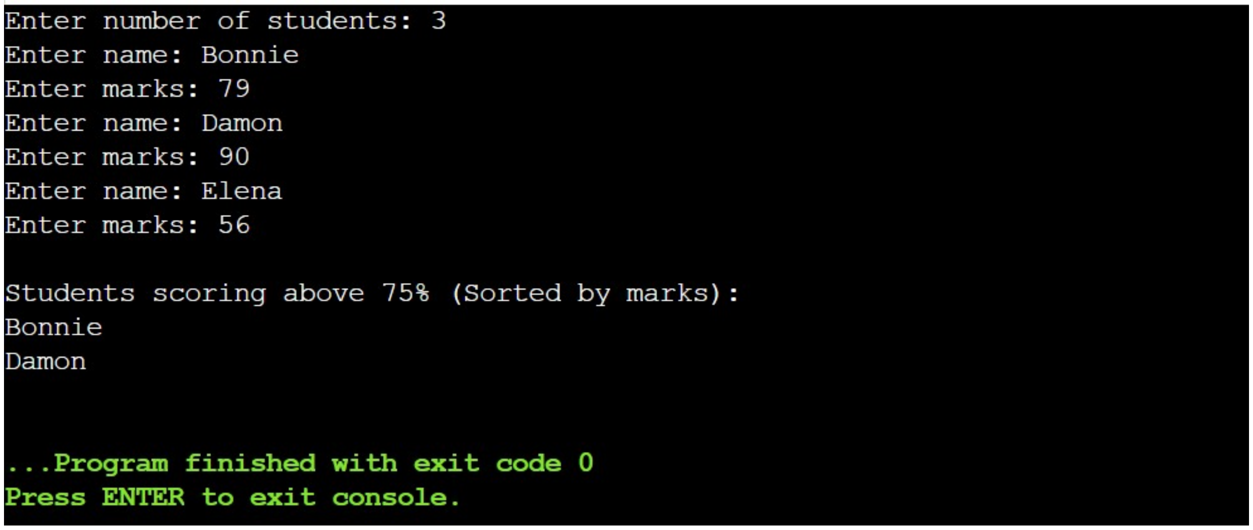
Objective: The goal is to use lambda expressions and stream operations to filter and sort students scoring above 75%. This enhances proficiency in functional programming, data filtering, and sorting techniques.

Code/Implementation:

```
import java.util.*;

import java.util.stream.Collectors;


class      Student

  { String name;

  double marks;


  Student(String name, double marks)

      { this.name = name;

      this.marks = marks;

  }

}


public class StudentFilter {

  public static void main(String[] args)

      { Scanner sc = new Scanner(System.in);

      List<Student> students = new ArrayList<>();


      System.out.print("Enter number of students: ");
```

```
        int n = sc.nextInt();
        sc.nextLine();

        for (int i = 0; i < n; i++)
          { System.out.print("Enter name: ");
          String name = sc.nextLine();
          System.out.print("Enter marks: ");
          double marks = sc.nextDouble();
          sc.nextLine();

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

        List<String> filteredStudents = students.stream()
          .filter(s -> s.marks > 75)
          .sorted(Comparator.comparingDouble(s -> s.marks))
          .map(s -> s.name)
          .collect(Collectors.toList());

        System.out.println("\nStudents scoring above 75% (Sorted by marks):");
        filteredStudents.forEach(System.out::println);

        sc.close();
      }
    }
```

Output:



```
Enter number of students: 3
Enter name: Bonnie
Enter marks: 79
Enter name: Damon
Enter marks: 90
Enter name: Elena
Enter marks: 56

Students scoring above 75% (Sorted by marks):
Bonnie
Damon


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

HARD:

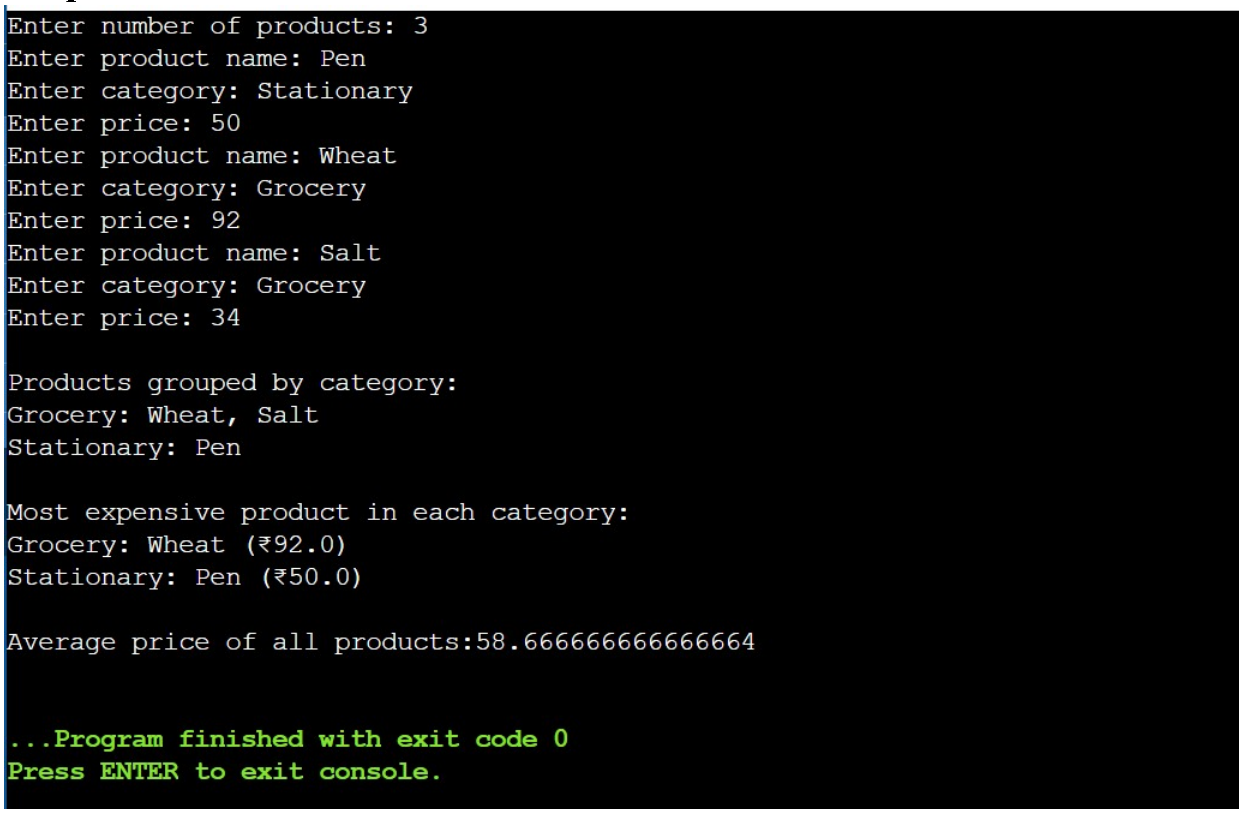
Aim: Write a Java program to process a large dataset of products using streams. Perform operations such as grouping products by category, finding the most expensive product in each category, and calculating the average price of all products.

Objective: This task focuses on processing large product datasets using Java Streams for grouping, filtering, and statistical computations. It helps in learning data aggregation, functional programming, and efficient data handling.

Code/Implementation:

```
import java.util.*;

import java.util.stream.Collectors;


class Product {
    String name, category;
    double price;

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

public class ProductAnalysis {
    public static void main(String[] args)
        { Scanner sc = new Scanner(System.in);
        List<Product> products = new ArrayList<>();

        System.out.print("Enter number of products: ");
        int n = sc.nextInt();
        sc.nextLine();
        for (int i = 0; i < n; i++)
            { System.out.print("Enter product name: ");
            String name = sc.nextLine();
            System.out.print("Enter category: ");
            String category = sc.nextLine();
            System.out.print("Enter price: ");
            double price = sc.nextDouble();
            sc.nextLine();

            products.add(new Product(name, category, price));
        }

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

        System.out.println("\nProducts grouped by category:");
        groupedProducts.forEach((category, productList) -> {
            System.out.println(category + ": " + productList.stream()
                .map(p -> p.name)
                .collect(Collectors.joining(", ")));
        });

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

        System.out.println("\nMost expensive product in each category:");
        mostExpensiveInCategory.forEach((category, product) ->
            System.out.println(category + ": " + product.get().name + " (₹" +
product.get().price + ")"));
        double averagePrice = products.stream()
            .mapToDouble(p -> p.price)
            .average()
            .orElse(0.0);
        System.out.println("\nAverage price of all products:" + averagePrice);
        sc.close();
    }
}
```

Output:



```
Enter number of products: 3
Enter product name: Pen
Enter category: Stationary
Enter price: 50
Enter product name: Wheat
Enter category: Grocery
Enter price: 92
Enter product name: Salt
Enter category: Grocery
Enter price: 34

Products grouped by category:
Grocery: Wheat, Salt
Stationary: Pen

Most expensive product in each category:
Grocery: Wheat (₹92.0)
Stationary: Pen (₹50.0)

Average price of all products:58.666666666666664


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

Learning Outcome:

1. Gained hands-on experience with lambda functions and Java Streams, enabling efficient sorting, filtering, and data processing.

2. Learned how to use lambda functions with Comparator and functional interfaces for sorting and filtering data efficiently.