Akshansh Kaundal - 23BCS13369 - 622 B

Experiment 2.3 - Java Programs Using Lambda Expressions and Stream Operations

Objective: To understand and implement Java 8 features such as lambda expressions and stream APIs for sorting, filtering, and data processing.

Part (a): Sorting Employee Objects Using Lambda Expressions

```
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;     }

    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("Saksham", 25, 55000),
new Employee("Riya", 23, 60000),            new
Employee("Arjun", 27, 50000),            new
Employee("Neha", 24, 70000)         );

        System.out.println("Original List:");
employees.forEach(System.out::println);

        System.out.println("\nSorted by Name:");
employees.stream()                  .sorted((e1, e2) ->
e1.name.compareTo(e2.name))
                 .forEach(System.out::println);

        System.out.println("\nSorted by Age (Ascending):");
employees.stream()
.sorted(Comparator.comparingInt(e -> e.age))
                 .forEach(System.out::println);

        System.out.println("\nSorted by Salary (Descending):");
employees.stream()                  .sorted((e1, e2) ->
Double.compare(e2.salary, e1.salary))
                 .forEach(System.out::println);
    }
}

--- Sample Output ---

Original List:
Saksham (Age: 25, Salary: 55000.0)
```

```
Riya (Age: 23, Salary: 60000.0)
Arjun (Age: 27, Salary: 50000.0)
Neha (Age: 24, Salary: 70000.0)

Sorted by Name:
Arjun (Age: 27, Salary: 50000.0)
Neha (Age: 24, Salary: 70000.0)
Riya (Age: 23, Salary: 60000.0)
Saksham (Age: 25, Salary: 55000.0)

Sorted by Age (Ascending):
Riya (Age: 23, Salary: 60000.0)
Neha (Age: 24, Salary: 70000.0)
Saksham (Age: 25, Salary: 55000.0)
Arjun (Age: 27, Salary: 50000.0)

Sorted by Salary (Descending):
Neha (Age: 24, Salary: 70000.0)
Riya (Age: 23, Salary: 60000.0)
Saksham (Age: 25, Salary: 55000.0)
Arjun (Age: 27, Salary: 50000.0)
```

Part (b): Filtering and Sorting Students Using Streams

```
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 toString() {
return name + " - Marks: " + marks;     } }

public class StudentFilter {
    public static void main(String[] args) {
List<Student> students = Arrays.asList(
new Student("Saksham", 82),            new
Student("Riya", 74),             new
Student("Aman", 90),             new
Student("Neha", 78)         );

        System.out.println("Students scoring above 75% (sorted by marks descending):");
students.stream()                 .filter(s -> s.marks > 75)
                .sorted((s1, s2) -> Double.compare(s2.marks, s1.marks))
                .map(s -> s.name + " - " + s.marks)
                .forEach(System.out::println);
    }
}

--- Sample Output ---

All Students:
Saksham - Marks: 82.0
Riya - Marks: 74.0
Aman - Marks: 90.0
Neha - Marks: 78.0

Students scoring above 75% (sorted by marks descending):
Aman - 90.0
Saksham - 82.0
Neha - 78.0
```

Part (c): Stream Operations on Product Dataset

```
import java.util.*; import
java.util.stream.*; import
java.util.Map.Entry;

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

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

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

public class ProductStream {
    public static void main(String[] args) {
List<Product> products = Arrays.asList(             new
Product("Laptop", 70000, "Electronics"),             new
Product("Phone", 40000, "Electronics"),             new
Product("Shirt", 1500, "Clothing"),             new
Product("Jeans", 2500, "Clothing"),             new
Product("Fridge", 50000, "Appliances"),             new
Product("Washing Machine", 45000, "Appliances")         );

        // Grouping by Category
        System.out.println("Grouped by Category:");
        Map<String, List<Product>> grouped = products.stream()
.collect(Collectors.groupingBy(p -> p.category));
grouped.forEach((cat, list) -> {
System.out.println(cat + ": " + list);
        });

        // Most Expensive Product in Each Category
        System.out.println("\nMost Expensive Product in Each Category:");
        Map<String, Optional<Product>> maxByCategory = products.stream()
                .collect(Collectors.groupingBy(p -> p.category,
Collectors.maxBy(Comparator.comparingDouble(p -> p.price))));
maxByCategory.forEach((cat, prod) -> {             System.out.println(cat + ": " +
prod.get());
        });

        // Average Price of All Products         double avgPrice =
products.stream()
.collect(Collectors.averagingDouble(p -> p.price));
        System.out.println("\nAverage Price of All Products: ■" + avgPrice);
    }
}
--- Sample Output ---

All Products:
Laptop (Electronics) - ■70000.0
Phone (Electronics) - ■40000.0
Shirt (Clothing) - ■1500.0
Jeans (Clothing) - ■2500.0
Fridge (Appliances) - ■50000.0 Washing
Machine (Appliances) - ■45000.0

Grouped by Category:
Electronics: [Laptop (Electronics) - ■70000.0, Phone (Electronics) - ■40000.0]
Clothing: [Shirt (Clothing) - ■1500.0, Jeans (Clothing) - ■2500.0]
Appliances: [Fridge (Appliances) - ■50000.0, Washing Machine (Appliances) - ■45000.0]

Most Expensive Product in Each Category:
Electronics: Laptop (Electronics) - ■70000.0
Clothing: Jeans (Clothing) - ■2500.0
```

```
Appliances: Fridge (Appliances) - ■50000.0

Average Price of All Products: ■34833.333333333336
```