**Experiment 6**

**Student Name: Mannat Anand UID:22BCS12713**

**Branch: CSE Section:22BCS\_IOT-642A**

**Semester: 6th DOP:04/03/25**

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

**Aim:**  Develop Java programs using lambda expressions and stream operations for sorting, filtering, and processing large datasets efficiently.

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

**Algorithm:**

1. **Define the Employee Class:** 
   * Create an Employee class with attributes: name, age, and salary.
   * Include a constructor to initialize these attributes.
   * Override the toString method for easy display of Employee objects.
2. **Create the List of Employee Objects:** 
   * Create an ArrayList of Employee objects.
   * Populate the list with sample Employee objects.
3. **Sort the List Using Lambda Expressions:** 
   * Use the Collections.sort method to sort the list.
   * Pass a Comparator with a lambda expression to specify the sorting criteria.
4. **Display the Sorted List:** 
   * Print the list before and after sorting to compare the results.

**Code:**

import java.util.ArrayList;

import java.util.Collections;

import java.util.Comparator;

import java.util.List;

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 "Employee{name='" + name + "', age=" + age + ", salary=" + salary + "}";

}

}

public class classwork3 {

public static void main(String[] args) {

List<Employee> employees = new ArrayList<>();

employees.add(new Employee("mannat", 25, 90000));

employees.add(new Employee("abc", 28, 80000));

employees.add(new Employee("xyz", 29, 85874));

System.out.println("Before sorting:");

employees.forEach(System.out::println);

Collections.sort(employees, Comparator.comparing(employee -> employee.name));

System.out.println("\nAfter sorting by name:");

employees.forEach(System.out::println);

Collections.sort(employees, Comparator.comparingInt(employee -> employee.age));

System.out.println("\nAfter sorting by age:");

employees.forEach(System.out::println);

Collections.sort(employees, Comparator.comparingDouble(employee -> employee.salary));

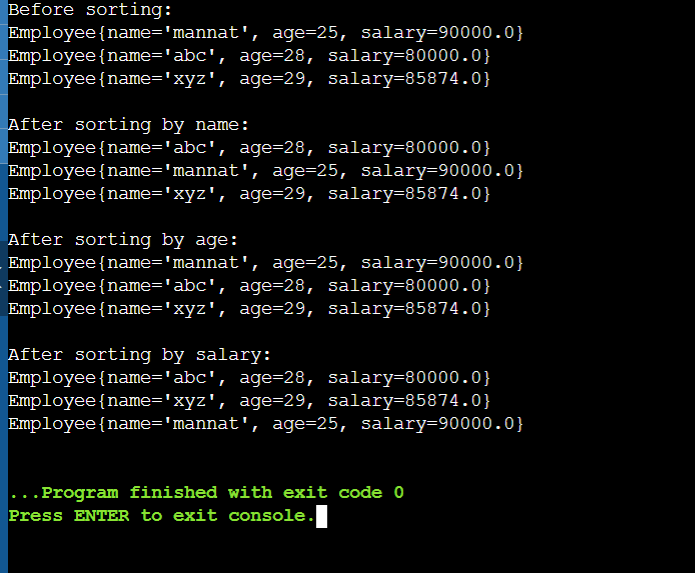
System.out.println("\nAfter sorting by salary:");

employees.forEach(System.out::println);

}

}

**Output:**



**Code 2:**

**Aim:** Develop Java programs using lambda expressions and stream operations for sorting, filtering, and processing large datasets efficiently.

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

**Algorithm:**

1. **Define the Student Class:** 
   * Create a Student class with attributes: name and marks.
   * Include a constructor to initialize these attributes.
   * Override the toString method for easy display of Student objects.
2. **Create the List of Student Objects:** 
   * Create an ArrayList of Student objects.
   * Populate the list with sample Student objects.
3. **Filter and Sort Using Lambda Expressions and Stream Operations:** 
   * Use the stream() method to create a stream from the list.
   * Apply the filter() method to filter students scoring above 75%.
   * Apply the sorted() method to sort the filtered students by their marks in descending order.
   * Collect the filtered and sorted students into a new list using the collect() method.
4. **Display the Filtered and Sorted List:** • Print the list before and after filtering and sorting to compare the results.

**Code:**

import java.util.ArrayList;

import java.util.List;

import java.util.stream.Collectors;

class Student {

String name;

double marks;

Student(String name, double marks) {

this.name = name;

this.marks = marks;

}

@Override

public String toString() {

return "Student{name='" + name + "', marks=" + marks + "}";

}

}

public class classwork3 {

public static void main(String[] args) {

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

students.add(new Student("mannat", 99));

students.add(new Student("abc", 73));

students.add(new Student("xyz", 82));

students.add(new Student("man", 35));

students.add(new Student("nat", 44));

System.out.println("Before filtering and sorting:");

students.forEach(System.out::println);

List<Student> students1 = students.stream()

.filter(student -> student.marks > 75)

.sorted((s1, s2) -> Double.compare(s2.marks, s1.marks))

.collect(Collectors.toList());

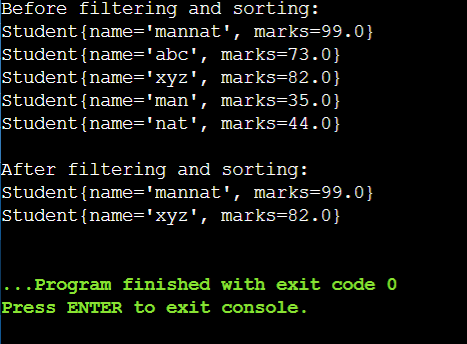
System.out.println("\nAfter filtering and sorting:");

students1.forEach(System.out::println);

}

}

**Output:**



**Code 3:**

**Aim:** Develop Java programs using lambda expressions and stream operations for sorting, filtering, and processing large datasets efficiently.

**Objective:** 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.

**Algorithm:**

1. **Create a Product class:** 
   * Define attributes: name, category, and price.
   * Implement a constructor to initialize these attributes.
   * Override the toString() method for readable output.
2. **Create a ProductProcessor class:** 
   * Implement the main method to perform the required operations.
3. **Initialize the product list:** 
   * Create a List<Product> to store product instances.
   * Add various products to the list with different categories and prices.
4. **Group products by category:** 
   * Use the stream() method on the product list.
   * Apply the collect() method with Collectors.groupingBy() to group products by their category.
   * Store the result in a Map<String, List<Product>>.
5. **Print products grouped by category:** 
   * Iterate over the Map and print each category along with the list of products in that category.
6. **Find the most expensive product in each category:** 
   * Use the stream() method on the product list.
   * Apply the collect() method with Collectors.groupingBy() and

Collectors.maxBy() to find the most expensive product in each category.

* + Store the result in a Map<String, Optional<Product>>.

1. **Print the most expensive product in each category:** 
   * Iterate over the Map and print each category along with the most expensive product in that category.
2. **Calculate the average price of all products:** 
   * Use the stream() method on the product list.
   * Apply the collect() method with Collectors.summarizingDouble() to calculate summary statistics for the product prices.
   * Extract the average price from the statistics.
3. **Print the average price of all products:** 
   * Print the calculated average price.

**Code:**

import java.util.ArrayList;

import java.util.DoubleSummaryStatistics;

import java.util.List;

import java.util.Map;

import java.util.Optional;

import java.util.stream.Collectors;

class Product {

String name;

String category;

double price;

Product(String name, String category, double price) {

this.name = name;

this.category = category;

this.price = price;

}

@Override

public String toString() {

return "Product{name='" + name + "', category='" + category + "', price=" + price + "}";

}

}

public class classwork3 {

public static void main(String[] args) {

List<Product> products = new ArrayList<>();

products.add(new Product("Shirt", "Cloths", 1500));

products.add(new Product("Sneakers", "Footwear", 15000));

products.add(new Product("Iphone15", "Electronics", 90000));

products.add(new Product("SamsungS25Ultra", "Electronics", 127000));

products.add(new Product("Mac", "Electronics", 80000));

products.add(new Product("jeans", "Cloths", 2500));

products.add(new Product("nikeAir", "Footwear", 120000));

Map<String, List<Product>> products1 = products.stream()

.collect(Collectors.groupingBy(product -> product.category));

System.out.println("Products grouped by category:");

products1.forEach((category, productList) -> {

System.out.println(category + ": " + productList);

});

Map<String, Optional<Product>> categoryassending = products.stream()

.collect(Collectors.groupingBy(product -> product.category,

Collectors.maxBy((p1, p2) -> Double.compare(p1.price, p2.price))));

System.out.println("\nMost expensive product in each category:");

categoryassending.forEach((category, product) -> {

System.out.println(category + ": " + product.orElse(null));

});

DoubleSummaryStatistics statistics = products.stream()

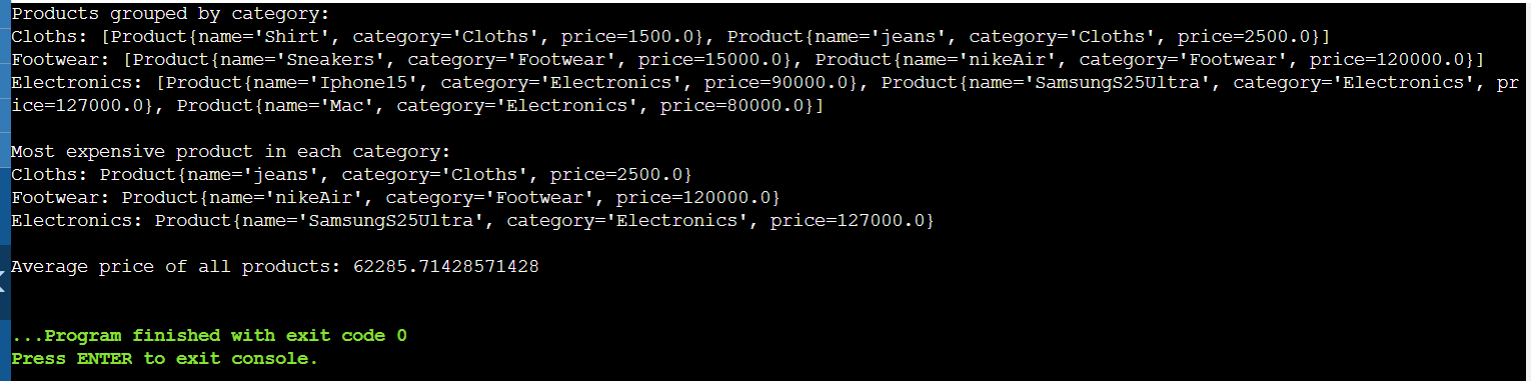
.collect(Collectors.summarizingDouble(product -> product.price));

System.out.println("\nAverage price of all products: " + statistics.getAverage());

}

}

**Output:**



**Learning Outcomes:**

1. Demonstrate: Apply key concepts to real-world scenarios to showcase understanding.
2. Analyze: Critically evaluate information, identify patterns, and draw meaningful conclusions.
3. Create: Develop original work, including presentations, reports, or projects, to exhibit comprehension and skills.