/\*

README - Java Lambda & Stream Operations

Introduction

This repository contains Java programs demonstrating the power of lambda expressions and stream operations for sorting, filtering, and processing large datasets efficiently.

Problem Statements

Easy Level

Task: Write a program to sort a list of Employee objects based on different attributes such as name, age, and salary using lambda expressions.

Medium Level

Task: Create a program that filters students scoring above 75%, sorts them by marks, and displays their names using lambda expressions and stream operations.

Hard Level

Task: Write a Java program to process a large dataset of products using streams. Implement the following operations:

1. Group products by category.

2. Find the most expensive product in each category.

3. Calculate the average price of all products.

\*/

import java.util.\*;

import java.util.stream.\*;

// Easy Level - Employee Sorting

class Employee {

private String name;

private int age;

private double salary;

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

this.name = name;

this.age = age;

this.salary = salary;

}

public String getName() { return name; }

public int getAge() { return age; }

public double getSalary() { return salary; }

@Override

public String toString() {

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

}

}

// Medium Level - Student Filtering and Sorting

class Student {

private String name;

private double marks;

public Student(String name, double marks) {

this.name = name;

this.marks = marks;

}

public String getName() { return name; }

public double getMarks() { return marks; }

@Override

public String toString() {

return name + " (" + marks + ")";

}

}

// Hard Level - Product Processing

class Product {

private String name;

private String category;

private double price;

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

this.name = name;

this.category = category;

this.price = price;

}

public String getName() { return name; }

public String getCategory() { return category; }

public double getPrice() { return price; }

@Override

public String toString() {

return name + " - " + category + " ($" + price + ")";

}

}

public class Assignment6 {

public static void main(String[] args) {

// Easy Level

List<Employee> employees = Arrays.asList(

new Employee("Alice", 30, 50000),

new Employee("Bob", 25, 45000),

new Employee("Charlie", 35, 70000)

);

System.out.println("\nSorted by Name:");

employees.stream().sorted(Comparator.comparing(Employee::getName)).forEach(System.out::println);

System.out.println("\nSorted by Age:");

employees.stream().sorted(Comparator.comparing(Employee::getAge)).forEach(System.out::println);

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

employees.stream().sorted(Comparator.comparing(Employee::getSalary)).forEach(System.out::println);

// Medium Level

List<Student> students = Arrays.asList(

new Student("John", 80),

new Student("Jane", 70),

new Student("Tom", 85)

);

System.out.println("\nFiltered and Sorted Students:");

students.stream()

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

.sorted(Comparator.comparing(Student::getMarks).reversed())

.forEach(System.out::println);

// Hard Level

List<Product> products = Arrays.asList(

new Product("Laptop", "Electronics", 1000),

new Product("Phone", "Electronics", 700),

new Product("Shirt", "Clothing", 50),

new Product("Jeans", "Clothing", 60),

new Product("Tablet", "Electronics", 400)

);

System.out.println("\nProducts Grouped by Category:");

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

.collect(Collectors.groupingBy(Product::getCategory));

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

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

});

System.out.println("\nMost Expensive Product in Each Category:");

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

Product expensive = productList.stream()

.max(Comparator.comparing(Product::getPrice))

.orElse(null);

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

});

System.out.println("\nAverage Price of All Products:");

double averagePrice = products.stream()

.mapToDouble(Product::getPrice)

.average()

.orElse(0.0);

System.out.println("Average Price: " + averagePrice);

}

}