**Java Assignment Day 9: Java Streams and Lambda functions:**

Q1.) Employee class and department class:

import java.util.List;  
import java.util.ArrayList;  
import java.util.Map;  
import java.util.HashMap;  
import java.util.Collections;  
import java.util.stream.Collectors;  
  
  
  
class Employee {  
 private String name;  
  
 public Employee(String name) {  
 this.name = name;  
 }  
  
 public String getName() {  
 return name;  
 }  
}  
  
class Department {  
 private String name;  
 private List<Employee> employees;  
  
 public Department(String name, List<Employee> employees) {  
 this.name = name;  
 this.employees = employees;  
 }  
  
 public List<Employee> getEmployees() {  
 return employees;  
 }  
}

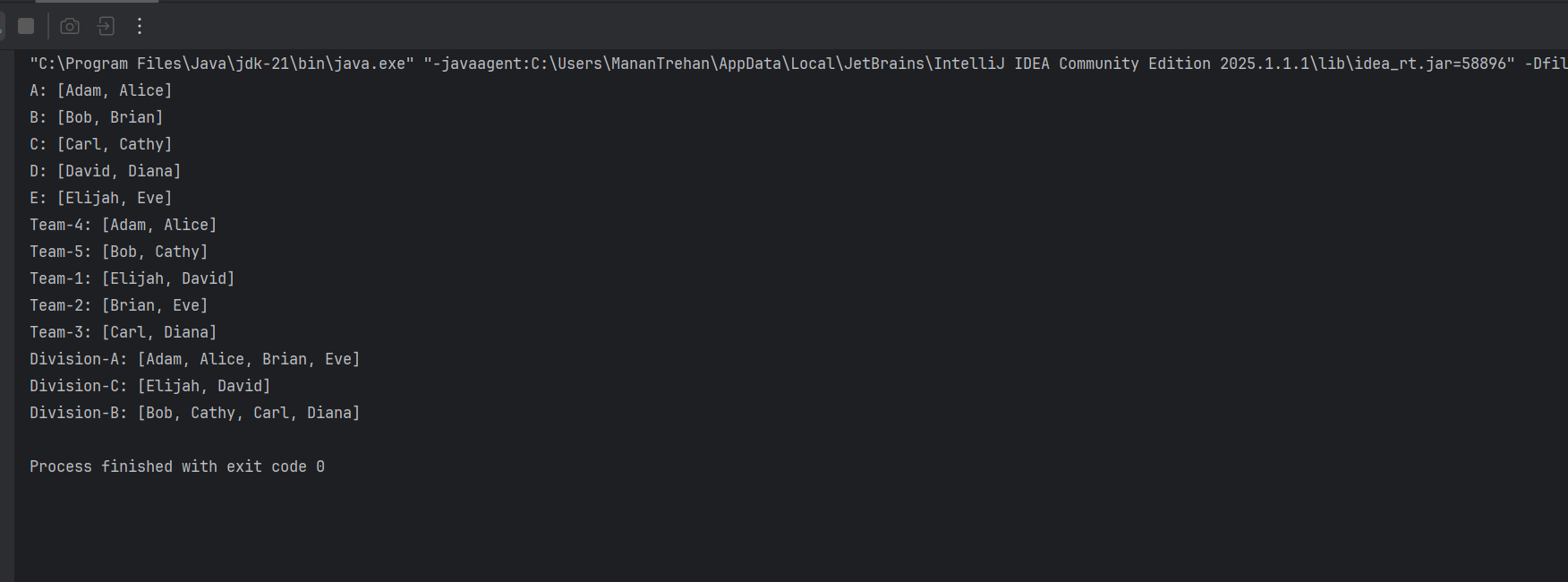
// Employee utility class

import java.util.List;  
import java.util.ArrayList;  
import java.util.Map;  
import java.util.HashMap;  
import java.util.Collections;  
import java.util.stream.Collectors;  
  
  
public class EmployeeUtility {  
  
 public static List<Employee> getAllEmployees(List<Department> departments) {  
 return departments.stream()  
 .flatMap(d -> d.getEmployees().stream())  
 .collect(Collectors.*toList*());  
 }  
  
 // Filter, sort and group employees by first letter of name  
 public static Map<Character, List<String>> groupSortedEmployeesByFirstLetter(List<Employee> employees) {  
 return employees.stream()  
 .map(Employee::getName)  
 .filter(name -> !name.isEmpty())  
 .sorted(String::compareToIgnoreCase)  
 .collect(Collectors.*groupingBy*(name -> Character.*toUpperCase*(name.charAt(0))));  
 }  
  
 // Randomly distribute employees into 5 sports teams  
 public static Map<String, List<Employee>> createSportsTeams(List<Employee> employees) {  
 Collections.*shuffle*(employees); // Randomize  
 Map<String, List<Employee>> teams = new HashMap<>();  
 for (int i = 0; i < 5; i++) {  
 teams.put("Team-" + (i + 1), new ArrayList<>());  
 }  
 for (int i = 0; i < employees.size(); i++) {  
 teams.get("Team-" + (i % 5 + 1)).add(employees.get(i));  
 }  
 return teams;  
 }  
  
 public static Map<String, List<Employee>> createDivisions(Map<String, List<Employee>> teams) {  
 Map<String, List<Employee>> divisions = new HashMap<>();  
 divisions.put("Division-A", new ArrayList<>());  
 divisions.put("Division-B", new ArrayList<>());  
 divisions.put("Division-C", new ArrayList<>());  
  
 int i = 0;  
 for (List<Employee> team : teams.values()) {  
 String division = switch (i % 3) {  
 case 0 -> "Division-A";  
 case 1 -> "Division-B";  
 default -> "Division-C";  
 };  
 divisions.get(division).addAll(team);  
 i++;  
 }  
  
 return divisions;  
 }  
}

// Main class

import java.util.List;  
import java.util.ArrayList;  
import java.util.Map;  
import java.util.HashMap;  
import java.util.Collections;  
import java.util.stream.Collectors;  
  
  
public class Main {  
 public static void main(String[] args) {  
 // Create Departments  
 List<Department> departments = List.*of*(  
 new Department("HR", List.*of*(new Employee("Alice"), new Employee("Adam"))),  
 new Department("IT", List.*of*(new Employee("Bob"), new Employee("Brian"))),  
 new Department("Sales", List.*of*(new Employee("Cathy"), new Employee("Carl"))),  
 new Department("Finance", List.*of*(new Employee("David"), new Employee("Diana"))),  
 new Department("Marketing", List.*of*(new Employee("Eve"), new Employee("Elijah")))  
 );  
  
 // Step 1: Get all employees  
 List<Employee> allEmployees = EmployeeUtility.*getAllEmployees*(departments);  
  
 // Step 2 & 3: Filter and group sorted names by first letter  
 Map<Character, List<String>> grouped = EmployeeUtility.*groupSortedEmployeesByFirstLetter*(allEmployees);  
 grouped.forEach((k, v) -> System.*out*.println(k + ": " + v));  
  
 // Step 4: Create 5 sports teams  
 Map<String, List<Employee>> teams = EmployeeUtility.*createSportsTeams*(allEmployees);  
 teams.forEach((k, v) -> {  
 System.*out*.println(k + ": " + v.stream().map(Employee::getName).toList());  
 });  
  
 // Step 5: Merge 5 teams into 3 divisions  
 Map<String, List<Employee>> divisions = EmployeeUtility.*createDivisions*(teams);  
 divisions.forEach((k, v) -> {  
 System.*out*.println(k + ": " + v.stream().map(Employee::getName).toList());  
 });  
 }  
}

Output :



Q2.)

import java.util.Map;

import java.util.function.Function;

import java.util.stream.Collectors;

public class CharFrequency {

public static void main(String[] args) {

String input = "hello world";

Map<Character, Long> frequencyMap = input.chars() // Returns an IntStream

.mapToObj(c -> (char) c) // Convert int to Character

.filter(c -> !Character.isWhitespace(c)) // Optional: remove spaces

.collect(Collectors.groupingBy(

Function.identity(),

Collectors.counting()

));

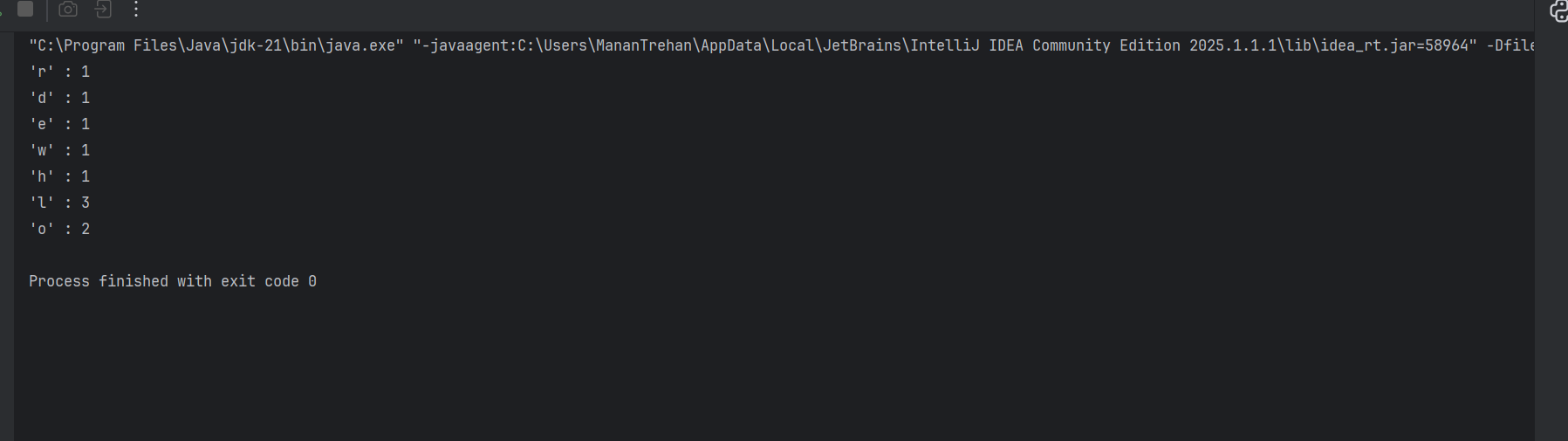
frequencyMap.forEach((ch, count) ->

System.out.println("'" + ch + "' : " + count));

}

}

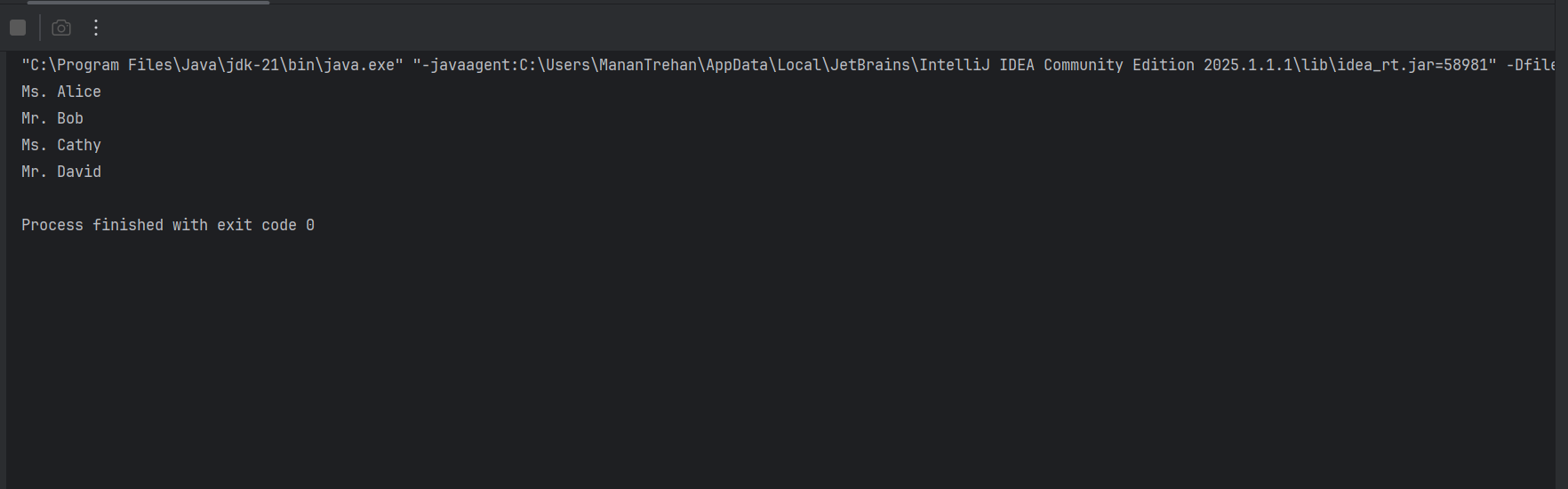
Output:



Q3.)

public class Student {  
 private String name;  
 private String gender; // "Male" or "Female"  
  
 public Student(String name, String gender) {  
 this.name = name;  
 this.gender = gender;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public String getGender() {  
 return gender;  
 }  
}

import java.util.\*;  
import java.util.stream.Collectors;  
  
public class StudentPrefixDemo {  
  
 public static void main(String[] args) {  
 List<Student> students = Arrays.*asList*(  
 new Student("Alice", "Female"),  
 new Student("Bob", "Male"),  
 new Student("Cathy", "Female"),  
 new Student("David", "Male")  
 );  
  
 // Using Java 8 Streams to map names with prefix  
 List<String> prefixedNames = students.stream()  
 .map(student -> {  
 String prefix = student.getGender().equalsIgnoreCase("Male") ? "Mr. " : "Ms. ";  
 return prefix + student.getName();  
 })  
 .collect(Collectors.*toList*());  
  
 // Print the result  
 prefixedNames.forEach(System.*out*::println);  
 }  
}

Output:  


Q4.)

Laptop class

import java.time.LocalDate;  
  
public class Laptop {  
 private String brand;  
 private int ramGB;  
 private int gpuGB;  
 private String processor;  
 private int hardDiskGB;  
 private LocalDate manufactureDate;  
  
 public Laptop(String brand, int ramGB, int gpuGB, String processor, int hardDiskGB, LocalDate manufactureDate) {  
 this.brand = brand;  
 this.ramGB = ramGB;  
 this.gpuGB = gpuGB;  
 this.processor = processor;  
 this.hardDiskGB = hardDiskGB;  
 this.manufactureDate = manufactureDate;  
 }  
  
 public int getRamGB() { return ramGB; }  
 public int getGpuGB() { return gpuGB; }  
 public String getProcessor() { return processor; }  
 public int getHardDiskGB() { return hardDiskGB; }  
 public LocalDate getManufactureDate() { return manufactureDate; }  
  
 @Override  
 public String toString() {  
 return brand + " | RAM: " + ramGB + "GB | GPU: " + gpuGB + "GB | " + processor +  
 " | HDD: " + hardDiskGB + "GB | Date: " + manufactureDate;  
 }  
}

//LaptopProcessor class

import java.util.\*;  
import java.util.stream.Collectors;  
  
public class LaptopProcessor {  
  
 public static Map<String, List<Laptop>> filterGroupSortLaptops(List<Laptop> laptops, int minRam, int minGpu) {  
 return laptops.stream()  
 .filter(l -> l.getRamGB() >= minRam && l.getGpuGB() >= minGpu)  
 .collect(Collectors.*groupingBy*(  
 Laptop::getProcessor,  
 Collectors.*collectingAndThen*(Collectors.*toList*(), list -> {  
 list.sort(Comparator.*comparingInt*(Laptop::getRamGB).reversed()  
 .thenComparingInt(Laptop::getHardDiskGB).reversed()  
 .thenComparing(Laptop::getManufactureDate).reversed());  
 return list;  
 })  
 ));  
 }  
}

// Main class

import java.time.LocalDate;  
import java.util.\*;  
  
public class Main {  
 public static void main(String[] args) {  
 List<Laptop> laptops = List.*of*(  
 new Laptop("Dell XPS", 16, 4, "Intel i7", 512, LocalDate.*of*(2023, 5, 10)),  
 new Laptop("HP Envy", 8, 2, "Intel i5", 256, LocalDate.*of*(2022, 8, 22)),  
 new Laptop("Acer Nitro", 16, 6, "Intel i7", 1024, LocalDate.*of*(2024, 2, 1)),  
 new Laptop("Lenovo Legion", 32, 8, "AMD Ryzen 7", 1024, LocalDate.*of*(2023, 9, 15)),  
 new Laptop("Asus ROG", 16, 4, "AMD Ryzen 7", 512, LocalDate.*of*(2021, 6, 18))  
 );  
  
 int minRam = 16;  
 int minGpu = 4;  
  
 Map<String, List<Laptop>> grouped = LaptopProcessor.*filterGroupSortLaptops*(laptops, minRam, minGpu);  
  
 grouped.forEach((processor, group) -> {  
 System.*out*.println("Processor: " + processor);  
 group.forEach(System.*out*::println);  
 System.*out*.println();  
 });  
 }  
}

Output:

