Introduction

In modern workplaces, managing and analyzing employee data effectively is crucial for making informed decisions about talent management and departmental performance. By leveraging Java's Stream API, we can perform complex data processing tasks such as filtering, sorting, and grouping with ease. In the example provided, we focus on analyzing employee data to identify top performers in various departments. This analysis helps to highlight individuals who are excelling in their roles and to gain insights into the distribution of experience and performance across different departments.

Code Explanation

The Employee class defines the structure and characteristics of an employee. It includes attributes for the employee's name, department, years of experience, and salary. This class provides getters and setters to access and modify these attributes, as well as a toString() method for a readable representation of the employee.

```java
package Stream_1;

/* Code finished in: 24/08/24
 *
 * By Fernando Sánchez González
 */

//Employee class where we define the characteristics of the employee
public class Employee {
 private String name;
 private String department;
 private int yearsOfExperience;
 private double salary;

 public Employee(String name, String department, int yearsOfExperience, double salary) {
     this.name = name;
     this.department = department;
     this.yearsOfExperience = yearsOfExperience;
     this.salary = salary;
 }

 // Getters
 public String getName() {
     return name;
 }

 public String getDepartment() {
     return department;
 }

 public int getYearsOfExperience() {
     return yearsOfExperience;
 }

 public double getSalary() {
     return salary;
```

```
26
27  public String getDepartment() {
28      return department;
29  }
30
31  public int getYearsOfExperience() {
32      return yearsOfExperience;
33  }
34
35  public double getSalary() {
36      return salary;
37  }
38
39  // Setter
40  public void setSalary(double salary) {
41      this.salary = salary;
42  }
43
44  @Override
45  public String toString() {
46      return name + " (" + department + ")";
47  }
48 }
49
50
```

The Main class demonstrates how to use Java Streams to process a list of Employee objects and analyze their performance by department. Here's a breakdown of the operations performed:

1. **Create a List of Employees**:
    - A List of Employee objects is created, each representing an employee with details such as name, department, years of experience, and salary.

```
public class Main {

 public static void main(String[] args) {

     // The list of employees is created
     List<Employee> employees = Arrays.asList(
         new Employee("Fernando", "Engineering", 5, 80000.0),
         new Employee("Nyssa", "Engineering", 8, 90000.0),
         new Employee("Linda", "HR", 3, 65000.0),
         new Employee("Bicho", "HR", 6, 70000.0),
         new Employee("Nydia", "Sales", 4, 72000.0),
         new Employee("Roberto", "Sales", 7, 85000.0)
     );
```

Process the Employee Data:

- **Filter**: Select employees with more than 4 years of experience.

```
// Converting the list of employees into a Stream and processing it to
Map<String, List<String>> topDepartmentEmployees = employees.stream()
    .filter(employee -> employee.getYearsOfExperience() > 4) //1
```

Peek: Log each employee that passes the filter for debugging purposes.

```
.peek(employee -> System.out.println("Filtered employee: " + employee.getName())) //2
```

Sort: Sort the employees by their department in alphabetical order.

```
.sorted((e1, e2) -> e1.getDepartment().compareTo(e2.getDepartment())) //3
```

FlatMap: Convert each Employee object into a stream of their names.

```
.flatMap(employee -> Collections.singletonList(employee.getName()).stream()) //4
```

Collect: Group the employee names by their department using the Collectors.groupingBy collector.

```
        .collect(Collectors.groupingBy( //5
            employeeName -> employees.stream()
                .filter(e -> e.getName().equals(employeeName))
                .findFirst()
                .map(e -> e.getDepartment())
                .orElse("Unknown")))
        ;
```

Output the Results:

- Print the top-performing employees by department. The output shows each department along with a list of employee names who have more than 4 years of experience.

```
// Output the results
topDepartmentEmployees.forEach((department, employeeNames) ->
    System.out.println("Department: " + department + ", Top Employees: " + employeeNames));
```