# Use Case: Java Stream API - Employee Salary Processing

## Problem Statement

An organization wants to generate reports from a list of employees, including:  
1. Filtering employees with salary > ₹50,000.  
2. Grouping employees by department.  
3. Calculating average salary in each department.  
4. Sorting employees by salary (descending).  
5. Finding the highest-paid employee.

## Entity Class

public class Employee {  
 private int id;  
 private String name;  
 private String department;  
 private double salary;  
  
 public Employee(int id, String name, String department, double salary) {  
 this.id = id;  
 this.name = name;  
 this.department = department;  
 this.salary = salary;  
 }  
  
 public int getId() { return id; }  
 public String getName() { return name; }  
 public String getDepartment() { return department; }  
 public double getSalary() { return salary; }  
  
 @Override  
 public String toString() {  
 return name + " | " + department + " | ₹" + salary;  
 }  
}

## Sample Data

List<Employee> employees = Arrays.asList(  
 new Employee(101, "Alice", "HR", 60000),  
 new Employee(102, "Bob", "IT", 75000),  
 new Employee(103, "Charlie", "IT", 55000),  
 new Employee(104, "David", "Finance", 50000),  
 new Employee(105, "Eva", "HR", 48000),  
 new Employee(106, "Frank", "Finance", 67000)  
);

## Use of Stream API

1. Filter employees with salary > ₹50,000

List<Employee> highEarners = employees.stream()  
 .filter(e -> e.getSalary() > 50000)  
 .collect(Collectors.toList());

2. Group employees by department

Map<String, List<Employee>> deptGroup = employees.stream()  
 .collect(Collectors.groupingBy(Employee::getDepartment));

3. Average salary by department

Map<String, Double> avgSalaryByDept = employees.stream()  
 .collect(Collectors.groupingBy(  
 Employee::getDepartment,  
 Collectors.averagingDouble(Employee::getSalary)  
 ));

4. Sort employees by salary in descending order

List<Employee> sortedBySalary = employees.stream()  
 .sorted((e1, e2) -> Double.compare(e2.getSalary(), e1.getSalary()))  
 .collect(Collectors.toList());

5. Find the highest paid employee

Optional<Employee> highestPaid = employees.stream()  
 .max(Comparator.comparingDouble(Employee::getSalary));

## Benefits of Using Stream API

- Readable & concise code  
- Less boilerplate, no need for external iterators  
- Functional-style operations  
- Efficient parallel processing if needed