Java Stream API – Complete Practice Workbook

Dataset: Employee Management System

1️⃣ Data Model

```
public class Employee {

    private int id;
    private String empCode;
    private String fullName;
    private String email;
    private String department;
    private double salary;

    public Employee(int id, String empCode, String fullName,
                    String email, String department, double salary)
{
        this.id = id;
        this.empCode = empCode;
        this.fullName = fullName;
        this.email = email;
        this.department = department;
        this.salary = salary;
    }

    public int getId() { return id; }
    public String getEmpCode() { return empCode; }
    public String getFullName() { return fullName; }
    public String getEmail() { return email; }
    public String getDepartment() { return department; }
    public double getSalary() { return salary; }

    @Override
    public String toString() {
        return fullName + " (" + department + ") - " + salary;
```

```
    }
}
```

2️⃣ Dataset

```
List<Employee> employees = List.of(
    new Employee(1, "EMP-1001", "Aarav Sharma",
"aarav.sharma@example.com", "Engineering", 120000.00),
    new Employee(2, "EMP-1002", "Vishal Shah",
"vishal-shah@example.com", "Chemical", 150000.00),
    new Employee(3, "EMP-1003", "Samiksha", "samiksha@example.com",
"Engineering", 90000.00),
    new Employee(4, "EMP-1004", "Tushar", "tushar@example.com",
"Chemical", 140000.00),
    new Employee(5, "EMP-1005", "Siddhant", "siddhant@example.com",
"Engineering", 80000.00)
);
```

3️⃣ Stream API Refresher

A Stream pipeline consists of:

```
Source → Intermediate Operations → Terminal Operation
```

Example:

```
employees.stream()
         .filter(e -> e.getSalary() > 100000)
         .map(Employee::getFullName)
         .forEach(System.out::println);
```

4️⃣ Basic Filtering

4.1 Employees with salary > 100000

```
employees.stream()
         .filter(e -> e.getSalary() > 100000)
         .forEach(System.out::println);
```

4.2 Employees in Engineering department

```
employees.stream()
         .filter(e -> e.getDepartment().equals("Engineering"))
         .forEach(System.out::println);
```

4.3 Employees whose name starts with 'S'

```
employees.stream()
         .filter(e -> e.getFullName().startsWith("S"))
         .forEach(System.out::println);
```

---

5 Mapping Operations

5.1 Get all employee emails

```
employees.stream()
         .map(Employee::getEmail)
         .forEach(System.out::println);
```

5.2 Get salary list

```
employees.stream()
         .map(Employee::getSalary)
         .forEach(System.out::println);
```

5.3 Convert names to uppercase

```
employees.stream()
         .map(e -> e.getFullName().toUpperCase())
         .forEach(System.out::println);
```

6️⃣ Primitive Streams

6.1 Total salary

```
double totalSalary = employees.stream()
        .mapToDouble(Employee::getSalary)
        .sum();
```

6.2 Average salary

```
double avgSalary = employees.stream()
        .mapToDouble(Employee::getSalary)
        .average()
        .orElse(0);
```

6.3 Highest salary

```
double maxSalary = employees.stream()
        .mapToDouble(Employee::getSalary)
        .max()
        .orElse(0);
```

7️⃣ Sorting

7.1 Sort by salary ascending

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

7.2 Sort by salary descending

```
employees.stream()

.sorted(Comparator.comparing(Employee::getSalary).reversed())
```

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

7.3 Sort by department then salary

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

8️⃣ Limit & Skip

8.1 Top 2 highest paid

```
employees.stream()

.sorted(Comparator.comparing(Employee::getSalary).reversed())
        .limit(2)
        .forEach(System.out::println);
```

8.2 Skip first 2 employees (after sorting)

```
employees.stream()
        .sorted(Comparator.comparing(Employee::getSalary))
        .skip(2)
        .forEach(System.out::println);
```

9️⃣ Distinct

9.1 Get distinct departments

```
employees.stream()
        .map(Employee::getDepartment)
        .distinct()
        .forEach(System.out::println);
```

🔟 Matching Operations

10.1 Any employee earning above 200000?

```
boolean highEarner = employees.stream()
        .anyMatch(e -> e.getSalary() > 200000);
```

10.2 All employees earn above 50000?

```
boolean allAbove50k = employees.stream()
        .allMatch(e -> e.getSalary() > 50000);
```

10.3 None belong to HR?

```
boolean noHR = employees.stream()
        .noneMatch(e -> e.getDepartment().equals("HR"));
```

1️⃣1️⃣ Find Operations (Optional Handling)

11.1 Find first Engineering employee

```
Optional<Employee> eng = employees.stream()
        .filter(e -> e.getDepartment().equals("Engineering"))
        .findFirst();

eng.ifPresent(System.out::println);
```

11.2 Find highest paid employee

```
Optional<Employee> maxEmp = employees.stream()
        .max(Comparator.comparing(Employee::getSalary));
```

1️⃣2️⃣ Collectors

12.1 Collect to List

```
List<Employee> highEarners = employees.stream()
        .filter(e -> e.getSalary() > 100000)
        .collect(Collectors.toList());
```

12.2 Collect to Map (empCode → Employee)

```
Map<String, Employee> empMap = employees.stream()
        .collect(Collectors.toMap(Employee::getEmpCode, e -> e));
```

12.3 Group by Department

```
Map<String, List<Employee>> byDept =
        employees.stream()

.collect(Collectors.groupingBy(Employee::getDepartment));
```

12.4 Count employees per department

```
Map<String, Long> countByDept =
        employees.stream()
                .collect(Collectors.groupingBy(
                        Employee::getDepartment,
                        Collectors.counting()));
```

12.5 Average salary per department

```
Map<String, Double> avgByDept =
        employees.stream()
                .collect(Collectors.groupingBy(
                        Employee::getDepartment,

Collectors.averagingDouble(Employee::getSalary)));
```

12.6 Partition by salary > 100000

```
Map<Boolean, List<Employee>> partition =
```

```
        employees.stream()
                .collect(Collectors.partitioningBy(
                        e -> e.getSalary() > 100000));
```

12.7 Join all employee names

```
String names = employees.stream()
        .map(Employee::getFullName)
        .collect(Collectors.joining(", "));
```

1️⃣3️⃣ Reduce

13.1 Sum salaries using reduce

```
double sum = employees.stream()
        .map(Employee::getSalary)
        .reduce(0.0, Double::sum);
```

13.2 Highest salary using reduce

```
Optional<Employee> highest =
        employees.stream()
                .reduce((e1, e2) ->
                        e1.getSalary() > e2.getSalary() ? e1 : e2);
```

1️⃣4️⃣ flatMap Example

Suppose each employee has multiple skills:

```
List<List<String>> skills = List.of(
    List.of("Java", "Spring"),
    List.of("Chemistry", "Lab"),
    List.of("React", "Java"),
    List.of("Process Design"),
    List.of("Testing", "Automation")
```

```
);

skills.stream()
      .flatMap(List::stream)
      .distinct()
      .forEach(System.out::println);
```

1️⃣5️⃣ Parallel Stream

```
employees.parallelStream()
         .map(Employee::getSalary)
         .forEach(System.out::println);
```

⚠ Avoid parallel for small datasets.

1️⃣6️⃣ Stream Pitfalls

1. Streams cannot be reused.
2. Avoid side effects inside `forEach`.
3. Handle Optional properly.
4. `toMap()` needs merge function if keys duplicate.
5. Parallel streams + mutable state = danger.

1️⃣7️⃣ Advanced Challenge Tasks

1. Find 2nd highest salary.
2. Find department with highest average salary.

3. Create Map<Department, Set<Names>>.

Create summary statistics:

`DoubleSummaryStatistics`

4.
5. Find employee with longest name.

6. Check if all emails contain "@example.com".

7. Convert employee list into JSON-like string using joining.

1️⃣8️⃣ Summary of What This Workbook Covered

✔ filter
✔ map
✔ mapToDouble
✔ flatMap
✔ sorted
✔ limit
✔ skip
✔ distinct
✔ findFirst / findAny
✔ anyMatch / allMatch / noneMatch
✔ collect
✔ groupingBy
✔ partitioningBy
✔ joining
✔ reduce (all versions)
✔ Optional handling
✔ parallelStream
✔ summary statistics