Java Stream API - 50 Practice Problems with Solutions

1. Filtering & Matching

Q: Find all employees with salary > 50k
```
A: employees.stream().filter(e -> e.getSalary() > 50000).toList();
```

Q: Check if any employee belongs to 'HR' department
```
A: employees.stream().anyMatch(e -> e.getDepartment().equals("HR"));
```

Q: Get employees whose name starts with 'A'
```
A: employees.stream().filter(e -> e.getName().startsWith("A")).toList();
```

Q: Check if all employees have salary > 30k
```
A: employees.stream().allMatch(e -> e.getSalary() > 30000);
```

Q: Check if no employee is from 'Intern' department
```
A: employees.stream().noneMatch(e -> e.getDepartment().equals("Intern"));
```

Q: Filter even numbers from list
```
A: numbers.stream().filter(n -> n % 2 == 0).toList();
```

Q: Get employees older than 40
```
A: employees.stream().filter(e -> e.getAge() > 40).toList();
```

Q: Find students with marks above 80
```
A: students.stream().filter(s -> s.getMarks() > 80).toList();
```

Q: Check if any string is empty
```
A: strings.stream().anyMatch(String::isEmpty);
```

Q: Get employees not from IT department
```
A: employees.stream().filter(e -> !e.getDepartment().equals("IT")).toList();
```

2. Mapping & Transformation

Q: Convert list of strings to uppercase
```
A: strings.stream().map(String::toUpperCase).toList();
```

Q: Extract employee names
```
A: employees.stream().map(Employee::getName).toList();
```

Q: Get square of numbers
```
A: numbers.stream().map(n -> n * n).toList();
```

Q: Extract salaries from employee list
```
A: employees.stream().map(Employee::getSalary).toList();
```

Q: Convert list of integers to string
```
A: numbers.stream().map(String::valueOf).toList();
```

Q: Get length of each string
```
A: strings.stream().map(String::length).toList();
```

Q: Append domain to email usernames
```
A: usernames.stream().map(u -> u + "@gmail.com").toList();
```

Q: Capitalize first letter of each word
```
A: words.stream().map(w -> Character.toUpperCase(w.charAt(0)) +
w.substring(1)).toList();
```

Q: Extract distinct departments
```
A: employees.stream().map(Employee::getDepartment).distinct().toList();
```

Q: Convert list of doubles to integers
```
A: doubles.stream().map(Double::intValue).toList();
```

3. Sorting & Ordering

Q: Sort employees by salary

```
A: employees.stream().sorted(Comparator.comparing(Employee::getSalary)).toList();
```

Q: Sort employees by name

```
A: employees.stream().sorted(Comparator.comparing(Employee::getName)).toList();
```

Q: Sort numbers in reverse order

```
A: numbers.stream().sorted(Comparator.reverseOrder()).toList();
```

Q: Get top 3 highest paid employees

```
A: employees.stream().sorted(Comparator.comparing(Employee::getSalary).reversed()).l
imit(3).toList();
```

Q: Get lowest 2 salaries

```
A: employees.stream().sorted(Comparator.comparing(Employee::getSalary)).limit(2).toL
ist();
```

Q: Sort strings by length

```
A: strings.stream().sorted(Comparator.comparing(String::length)).toList();
```

Q: Sort employees by age then salary

```
A: employees.stream().sorted(Comparator.comparing(Employee::getAge).thenComparing(Em
ployee::getSalary)).toList();
```

Q: Get employees sorted in descending order of name

```
A: employees.stream().sorted(Comparator.comparing(Employee::getName).reversed()).toL
ist();
```

Q: Sort unique numbers

```
A: numbers.stream().distinct().sorted().toList();
```

Q: Find 2nd highest salary

```
A: employees.stream().map(Employee::getSalary).sorted(Comparator.reverseOrder()).ski
p(1).findFirst().get();
```

4. Reducing & Aggregation

Q: Find total salary of all employees

```
A: employees.stream().map(Employee::getSalary).reduce(0, Integer::sum);
```

Q: Find max salary in each department

```
A: employees.stream().collect(Collectors.groupingBy(Employee::getDepartment,
Collectors.maxBy(Comparator.comparing(Employee::getSalary))));
```

Q: Count employees in IT department

```
A: employees.stream().filter(e -> e.getDepartment().equals("IT")).count();
```

Q: Find average salary

```
A: employees.stream().collect(Collectors.averagingDouble(Employee::getSalary));
```

Q: Find highest salary overall

```
A: employees.stream().map(Employee::getSalary).max(Integer::compare).get();
```

Q: Find lowest salary overall

```
A: employees.stream().map(Employee::getSalary).min(Integer::compare).get();
```

Q: Sum of numbers in list

```
A: numbers.stream().reduce(0, Integer::sum);
```

Q: Product of all numbers

```
A: numbers.stream().reduce(1, (a,b) -> a*b);
```

Q: Find longest string

```
A: strings.stream().max(Comparator.comparing(String::length)).get();
```

Q: Count distinct departments

```
A: employees.stream().map(Employee::getDepartment).distinct().count();
```

5. Grouping & Partitioning

Q: Group employees by department
```
A: employees.stream().collect(Collectors.groupingBy(Employee::getDepartment));
```

Q: Group employees by age
```
A: employees.stream().collect(Collectors.groupingBy(Employee::getAge));
```

Q: Group numbers by even and odd
```
A: numbers.stream().collect(Collectors.partitioningBy(n -> n % 2 == 0));
```

Q: Partition employees based on salary > 50k
```
A: employees.stream().collect(Collectors.partitioningBy(e -> e.getSalary() >
50000));
```

Q: Group employees by department and count
```
A: employees.stream().collect(Collectors.groupingBy(Employee::getDepartment,
Collectors.counting()));
```

Q: Group employees by department and list names
```
A: employees.stream().collect(Collectors.groupingBy(Employee::getDepartment,
Collectors.mapping(Employee::getName, Collectors.toList())));
```

Q: Group strings by length
```
A: strings.stream().collect(Collectors.groupingBy(String::length));
```

Q: Partition numbers > 100
```
A: numbers.stream().collect(Collectors.partitioningBy(n -> n > 100));
```

Q: Group employees by salary range (above/below 50k)
```
A: employees.stream().collect(Collectors.groupingBy(e -> e.getSalary() > 50000 ?
"High" : "Low"));
```

Q: Partition students by pass/fail
```
A: students.stream().collect(Collectors.partitioningBy(s -> s.getMarks() >= 40));
```

6. Advanced Operations

Q: Flatten nested lists
```
A: lists.stream().flatMap(List::stream).toList();
```

Q: Remove duplicates
```
A: numbers.stream().distinct().toList();
```

Q: Skip first 5 elements
```
A: numbers.stream().skip(5).toList();
```

Q: Limit to first 10 elements
```
A: numbers.stream().limit(10).toList();
```

Q: Peek into stream for debugging
```
A: numbers.stream().peek(System.out::println).toList();
```

Q: Parallel stream sum
```
A: numbers.parallelStream().reduce(0, Integer::sum);
```

Q: Generate infinite stream of numbers
```
A: Stream.iterate(1, n -> n + 1).limit(10).toList();
```

Q: Use custom collector
```
A: employees.stream().collect(Collectors.toCollection(TreeSet::new));
```

Q: Concatenate strings using joining
```
A: strings.stream().collect(Collectors.joining(", "));
```

Q: Find frequency of words
```
A: words.stream().collect(Collectors.groupingBy(Function.identity(),
Collectors.counting()));
```