Java Stream Collectors - Practice Questions with Solutions

Beginner Level

Convert a list of strings into a List, Set, and LinkedList using Collectors.

```
List<String> names = List.of("Alice", "Bob", "Charlie", "Alice");
List<String> list = names.stream().collect(Collectors.toList());
Set<String> set = names.stream().collect(Collectors.toSet());
LinkedList<String> linkedList = names.stream().collect(Collectors.toCollection(LinkedList::new));
```

Concatenate all names in the list with `, ` separator using Collectors.joining.

```
String result = names.stream().collect(Collectors.joining(", "));
```

Use Collectors.counting() to count the number of elements in a stream.

```
long count = names.stream().collect(Collectors.counting());
```

Use Collectors.maxBy() to find the longest string in the list.

```
Optional<String>  longest  =
names.stream().collect(Collectors.maxBy(Comparator.comparing(String::length)));
```

Use Collectors.summingInt() to calculate the sum of a list of integers.

```
List<Integer> numbers = List.of(1, 2, 3, 4, 5); int sum =
numbers.stream().collect(Collectors.summingInt(Integer::intValue));
```

Intermediate Level

Group employees by department using Collectors.groupingBy.

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

Count the number of employees in each department.

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

Group employees by department and calculate the average salary.

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

Group by department and collect employee names as List<String>.

```
Map<String, List<String>> namesByDept = employees.stream()
    .collect(Collectors.groupingBy(Employee::getDepartment,
             Collectors.mapping(Employee::getName, Collectors.toList())));
```

Partition employees into high earners and others (salary > 50k).

```
Map<Boolean, List<Employee>> partitioned = employees.stream()
    .collect(Collectors.partitioningBy(e -> e.getSalary() > 50000));
```

Advanced Level

Group employees by department and salary range.

```
Map<String, Map<String, List<Employee>>> grouped =
employees.stream().collect(Collectors.groupingBy(Employee::getDepartment,         Collectors.groupingBy(e -
> {             int salary = e.getSalary();             if (salary < 50000) return "<50k";             else
if (salary <= 100000) return "50k-100k";             else return ">100k";         })
    ));
```

Use summarizingInt to get salary stats.

```
IntSummaryStatistics stats = employees.stream()
    .collect(Collectors.summarizingInt(Employee::getSalary));
```

Create a custom collector to unmodifiable list.

```
Collector<String, ?, List<String>> unmodifiableListCollector =
Collectors.collectingAndThen(Collectors.toList(), Collections::unmodifiableList);
```

Flat map nested phone numbers to a Set.

```
Set<String> phones = people.stream()     .collect(Collectors.flatMapping(p -> p.getPhoneNumbers().stream(),
Collectors.toSet()));
```

Collect to Map<Department, AvgSalary>.

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

Convert stream to TreeMap with merge function.

```
TreeMap<String, Integer> map = list.stream()     .collect(Collectors.toMap(Function.identity(), s -> 1,
Integer::sum, TreeMap::new));
```

1. Frequency map of strings.

```
Map<String, Long> frequency = strings.stream()     .collect(Collectors.groupingBy(Function.identity(),
Collectors.counting()));
```

Group by department and join names.

```
Map<String, String> deptNames = employees.stream()
    .collect(Collectors.groupingBy(Employee::getDepartment,
Collectors.mapping(Employee::getName, Collectors.joining(", "))));
```

Reduce to highest transaction per category.

```
Map<String, Optional<Transaction>> maxByCat = transactions.stream()
    .collect(Collectors.groupingBy(Transaction::getCategory,
Collectors.maxBy(Comparator.comparing(Transaction::getAmount))));
```

Transform result using collectingAndThen.

```
List<String> immutableNames = names.stream()
    .collect(Collectors.collectingAndThen(Collectors.toList(), Collections::unmodifiableList));
```