Summary

These additional techniques provide powerful tools for processing collections in Java. By utilizing methods like `flatMap`, `filter`, `reduce`, `distinct`, and custom `sorted`, you can perform a variety of operations efficiently and expressively. If you have any more questions or need further examples, feel free to ask!

ignore and print initial 13

Here are the initial 13 questions and topics, along with their explanations and code examples:

1. How does the Java Streams `Collectors.groupingBy` method work?

- **Explanation**: It groups a collection based on a specified key and can also perform downstream operations, like collecting statistics.
- **Code Example**:

```java
List<Employee> employees = Arrays.asList(
    new Employee(1, "Alice", 50000.0, "HR"),
    new Employee(2, "Bob", 60000.0, "IT"),
    new Employee(3, "Charlie", 70000.0, "HR")
);

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

groupedByDepartment.forEach((dept, empList) ->
    System.out.println(dept + ": " + empList));
```

2. What does `Collectors.mapping` do?

- **Explanation**: It transforms elements in a stream to a different type while collecting.
- **Code Example**:

```java
Map<String, Optional<Double>> maxSalaryByDept = employees.stream()
    .collect(Collectors.groupingBy(Employee::getDepartment,
             Collectors.mapping(Employee::getSalary,
Collectors.maxBy(Double::compareTo))));

maxSalaryByDept.forEach((dept, maxSalary) ->
    System.out.println(dept + ": " + maxSalary.orElse(0.0)));
```

3. What will the output be when using `Collectors.mapping(Employee::getSalary, Collectors.maxBy(Double::compareTo))`?

- **Explanation**: It finds the maximum salary for each department.
- **Code Example**: Same as above.

4. What is the purpose of `mapToDouble` in Java Streams?

- **Explanation**: Converts a stream of objects into a `DoubleStream`, allowing for efficient numerical operations.
- **Code Example**:

```java
DoubleStream salaryStream = employees.stream()
    .mapToDouble(Employee::getSalary);

double totalSalaries = salaryStream.sum();
System.out.println("Total Salaries: " + totalSalaries);
```

5. How do you sort a stream in Java?

- **Explanation**: Use `sorted()` with or without a comparator to order elements.
- **Code Example**:

```java
List<Employee> sortedBySalary = employees.stream()
    .sorted(Comparator.comparing(Employee::getSalary))
    .collect(Collectors.toList());

sortedBySalary.forEach(emp ->
    System.out.println(emp.getName() + ": " + emp.getSalary()));
```

6. What is `Comparator.comparing(Employee::getSalary)`?

- **Explanation**: It creates a comparator to compare `Employee` objects by salary.
- **Code Example**:

```java
List<Employee> sortedBySalary = employees.stream()
    .sorted(Comparator.comparing(Employee::getSalary))
    .collect(Collectors.toList());

sortedBySalary.forEach(emp ->
    System.out.println(emp.getName() + ": " + emp.getSalary()));
```

7. Can you use a lambda expression instead of `Comparator.comparing`?

- **Explanation**: Yes, you can define a comparator using a lambda expression.
- **Code Example**:

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

sortedBySalary.forEach(emp ->
    System.out.println(emp.getName() + ": " + emp.getSalary()));
```

8. Does `groupingBy` require two parameters?

- **Explanation**: No, it can take one parameter to group by a key, or two parameters if you want to collect further statistics.
- **Code Example** (One Parameter):

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

9. What happens if you provide only one parameter to `groupingBy`?

- **Explanation**: It groups the elements based on the provided key and returns a `Map<K, List<T>>`.
- **Code Example**: Same as above.

10. What if you want to extract `String` values from a stream?

- **Explanation**: Use `map()` to transform the stream of objects into a stream of `String`.
- **Code Example**:

```java
List<String> employeeNames = employees.stream()
    .map(Employee::getName)
    .collect(Collectors.toList());

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

11. What would happen if you use `map()` for `double` values?

- **Explanation**: You will get a `Stream<Double>`, which involves boxing.
- **Code Example**:

```java
Stream<Double> salaryStream = employees.stream()
    .map(Employee::getSalary);

salaryStream.forEach(System.out::println); // Output: 50000.0, 60000.0, 70000.0
```

12. Is it better to use `mapToDouble` instead of `map()` for `double` values?

- **Explanation**: Yes, `mapToDouble` is more efficient for numerical operations.
- **Code Example**:

```java
DoubleStream doubleSalaryStream = employees.stream()
    .mapToDouble(Employee::getSalary);

double averageSalary = doubleSalaryStream.average().orElse(0);
System.out.println("Average Salary: " + averageSalary);
```

13. Can you perform operations like filtering or sorting on a stream of `String` values?

- **Explanation**: Yes! After extracting strings using `map()`, you can apply methods like `filter()` or `sorted()`.
- **Code Example**:

```java
List<String> filteredNames = employeeNames.stream()
    .filter(name -> name.startsWith("A"))
    .sorted()
    .collect(Collectors.toList());

filteredNames.forEach(System.out::println); // Output: Alice
```

Summary

This list includes 13 questions with clear explanations and practical code examples that demonstrate various techniques and functionalities in Java Streams. If you have any further questions or need more examples, feel free to ask!