Copilot

Absolutely! Here are a few more examples of how to use `Collectors.groupingBy` in different scenarios:

Example 1: Grouping by a Single Attribute

Let’s group a list of `Person` objects by their city:

```
import java.util.Arrays;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;

class Person {
    String name;
    String city;

    Person(String name, String city) {
        this.name = name;
        this.city = city;
    }

    public String getCity() {
        return city;
    }

    @Override
    public String toString() {
        return name;
    }
}

public class GroupingByCity {
    public static void main(String[] args) {
        List<Person> people = Arrays.asList(
            new Person("Alice", "New York"),
            new Person("Bob", "Los Angeles"),
            new Person("Charlie", "New York"),
            new Person("Dave", "Chicago")
        );

        Map<String, List<Person>> peopleByCity = people.stream()
            .collect(Collectors.groupingBy(Person::getCity));

        peopleByCity.forEach((city, persons) ->
            System.out.println(city + ": " + persons)
        );
    }
}
```

Example 2: Grouping and Counting

Group a list of words by their length and count the number of words in each group:

```
import java.util.Arrays;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;

public class GroupingAndCounting {
    public static void main(String[] args) {
        List<String> words = Arrays.asList("apple", "banana", "cherry", "date", "elderberry", "fig", "grape");

        Map<Integer, Long> wordCountByLength = words.stream()
```

```
            .collect(Collectors.groupingBy(String::length, Collectors.counting()));

        wordCountByLength.forEach((length, count) ->
            System.out.println("Length " + length + ": " + count + " words")
        );
    }
}
```

Example 3: Grouping by Multiple Attributes

Group a list of `BlogPost` objects by their type and author:

```
import java.util.Arrays;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;

class BlogPost {
    String title;
    String author;
    BlogPostType type;

    BlogPost(String title, String author, BlogPostType type) {
        this.title = title;
        this.author = author;
        this.type = type;
    }

    public BlogPostType getType() {
        return type;
    }

    public String getAuthor() {
        return author;
    }

    @Override
    public String toString() {
        return title;
    }
}

enum BlogPostType {
    NEWS, REVIEW, GUIDE
}

public class GroupingByMultipleAttributes {
    public static void main(String[] args) {
        List<BlogPost> posts = Arrays.asList(
            new BlogPost("Post1", "Alice", BlogPostType.NEWS),
            new BlogPost("Post2", "Bob", BlogPostType.REVIEW),
            new BlogPost("Post3", "Alice", BlogPostType.GUIDE),
            new BlogPost("Post4", "Alice", BlogPostType.NEWS)
        );

        Map<BlogPostType, Map<String, List<BlogPost>>> postsByTypeAndAuthor = posts.stream()
            .collect(Collectors.groupingBy(BlogPost::getType,
                Collectors.groupingBy(BlogPost::getAuthor)));

        postsByTypeAndAuthor.forEach((type, authorMap) -> {
            System.out.println(type);
            authorMap.forEach((author, blogPosts) ->
                System.out.println("  " + author + ": " + blogPosts)
            );
        });
    }
}
```

These examples demonstrate the flexibility and power of the `Collectors.groupingBy` method. If you have any specific scenarios in mind or need further clarification, feel free to ask!