Introduction

Customer feedback is crucial for businesses to understand their customers' experiences and improve their products or services. Analyzing feedback helps in identifying common issues, appreciating positive comments, and addressing areas that need improvement. This analysis can be performed in various ways, such as filtering relevant comments, counting occurrences, and identifying common themes. In this context, we will explore how to process and analyze customer feedback using Java Streams and functional programming concepts.

Code Explanation

The provided code performs a detailed analysis of customer feedback comments using Java's Stream API and functional programming constructs. Here's a breakdown of each part of the code:

```
public static void main(String[] args) {

    // The list of customer feedback comments
    List<String> feedbackComments = Arrays.asList(
        "Great service!",
        "The product quality is superb.",
        "Great service!",
        "Delivery was late.",
        "Excellent product.",
        "Very satisfied with the purchase.",
        "Very satisfied with the purchase.",
        "Good value for money.",
        "Poor customer support.",
        "The product quality is superb."
    );
```

- A list of customer feedback comments is created using Arrays.asList(). This list contains various comments about the service and product quality

```
Map<String, Long> commentFrequency = feedbackComments.stream()
    .filter(comment -> comment.toLowerCase().contains("product")) // Filter comments containing "product"
    .map(String::toUpperCase) // Convert comments to uppercase
    .distinct() // Remove duplicate comments
    .collect(Collectors.groupingBy(Function.identity(), Collectors.counting())); // Count occurrences of each unique comment
```

- **feedbackComments.stream()**: Converts the list into a stream for processing.

 filter(comment -> comment.toLowerCase().contains("product")): Filters comments that contain the keyword "product" (case-insensitive).

 map(String::toUpperCase): Converts each filtered comment to uppercase.

 distinct(): Removes duplicate comments, ensuring each comment is unique.

 collect(Collectors.groupingBy(Function.identity(), Collectors.counting())): Groups the comments and counts the occurrences of each unique comment. The result is a Map where the keys are the comments and the values are their counts.

```
commentFrequency.entrySet().stream()
    .sorted(Map.Entry.<String, Long>comparingByValue().reversed()) // Sort by frequency in descending order
    .forEach(entry -> System.out.println(entry.getKey() + ": " + entry.getValue() + " times")); // Print each comment with its
```

 commentFrequency.entrySet().stream(): Converts the map entries into a stream for further processing.

 sorted(Map.Entry.<String, Long>comparingByValue().reversed()): Sorts the entries by their frequency in descending order.

 forEach(entry -> System.out.println(entry.getKey() + ": " + entry.getValue() + " times")): Prints each unique comment along with its frequency.