Exploring Stream API Operations in Java: A Practical Guide (Part 2)

Introduction

This is Part Two of the tutorial. In the previous part, a dummy dataset of transactions was introduced and some basic analytics were presented. In this part, more analytics are generated to explore additional stream operations using the same dataset. The results are displayed using simple ASCII-based visualizations to make the patterns easier to observe in a console environment.

Time-Based Analytics

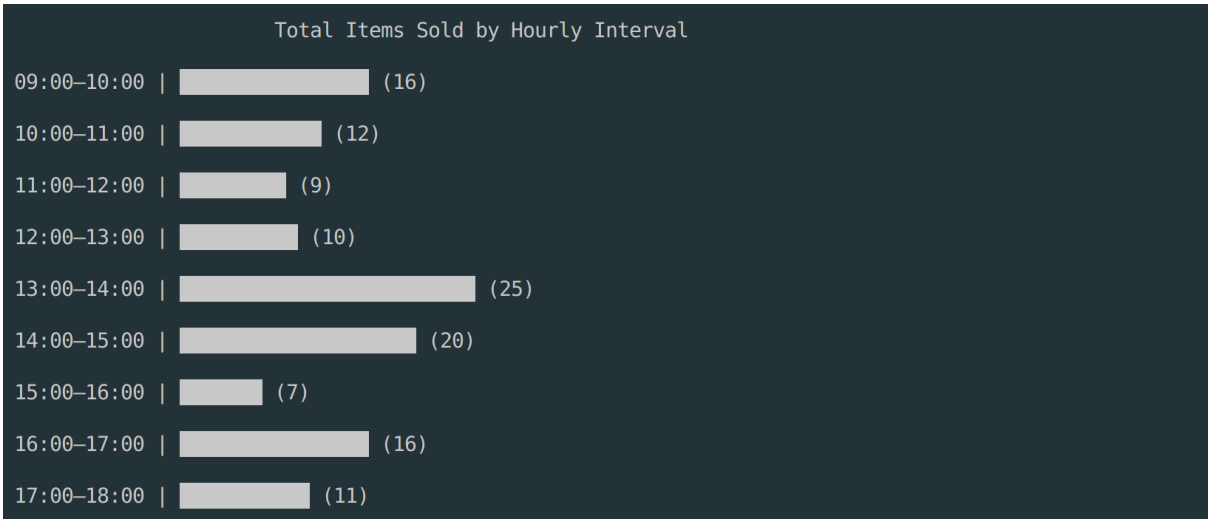
This category highlights how transaction behavior varies by time of day.

- ### Total Items Sold by Hourly Interval

  Transactions are grouped by the hour they occurred, and item quantities are summed up for each hour.

```
// Chart 1: Total Items Sold by Hourly Interval
// Stream API Methods: groupingBy (on hour) + summingInt, using TreeMap for
// sorted order
Map<Integer, Integer> itemsSoldByHour = transactions.stream()
                .collect(Collectors.groupingBy(
                                tx -> tx.time().getHour(), // Group by hour (0–23)
                                TreeMap::new, // Keep keys sorted
                                Collectors.summingInt(tx -> tx.items().stream().mapToInt(Item::quantity)
                                                .sum())));
```

The following chart shows how many items were sold in each hourly interval



- ### Total Transactions Count by Hour

  Here, only the count of transactions per hour is computed.

```java
// Chart 2: Total Transactions Count by Hour
// Stream API Methods: groupingBy (on hour) + counting
Map<Integer, Long> transactionsByHour = transactions.stream()
                .collect(Collectors.groupingBy(
                                tx -> tx.time().getHour(),
                                TreeMap::new,
                                Collectors.counting()));
```

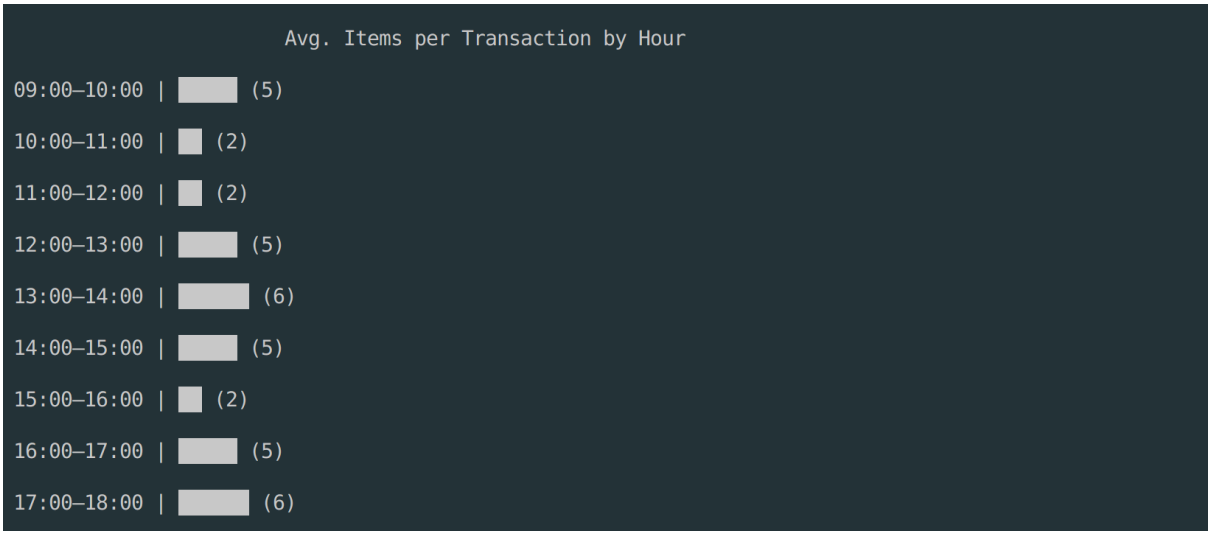
The chart below displays the number of transactions occurring during each hour:

- **Average Items per Transaction by Hour**

This snippet calculates the average number of items in each transaction for every hour.

```java
// Chart 3: Average Items per Transaction by Hour
// Stream API Methods: groupingBy (on hour) + averagingDouble
Map<Integer, Double> avgItemsByHour = transactions.stream()
                .collect(Collectors.groupingBy(
                                tx -> tx.time().getHour(),
                                TreeMap::new,
                                Collectors.averagingDouble(tx -> tx.items().stream()
                                                .mapToInt(Item::quantity).sum())));
```

The following chart visualizes the average number of items sold per transaction by hour:



- **Distinct Items Sold by Hour**

Determines how many unique items were sold during each hourly slot.

```
// Chart 4: Distinct Items Sold by Hour
// Stream API Methods: groupingBy (on hour) + flatMapping + collectingAndThen +
// toSet + size
Map<Integer, Long> distinctItemsByHour = transactions.stream()
                .collect(Collectors.groupingBy(
                                tx -> tx.time().getHour(),
                                TreeMap::new,
                                Collectors.collectingAndThen(
                                                Collectors.flatMapping(
                                                                tx -> tx.items().stream()
                                                                                .map(Item::name),
                                                                Collectors.toSet()),
                                                set -> (long) set.size())));
```

This chart shows the count of unique items sold in each hour:

Week-Based Analytics

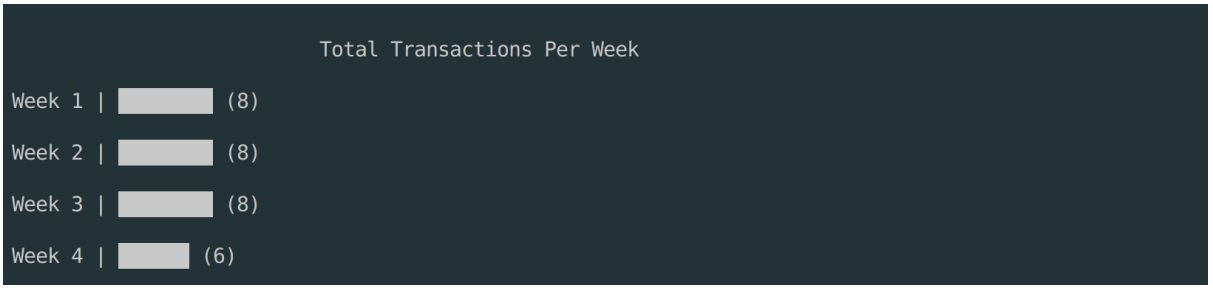
This section reveals patterns distributed across weeks in a month.

- **Total Transactions per Week**

Transactions are grouped by week of the month and counted.

```
// Chart 5: Total Transactions per Week
// Stream API Methods: groupingBy (on custom week label) + counting
Map<String, Long> transactionPerWeek = transactions.stream()
                .collect(Collectors.groupingBy(tx -> getWeekOfMonthLabel(tx.date()),
                                Collectors.counting()));
```

The chart below shows how many transactions occurred in each week:

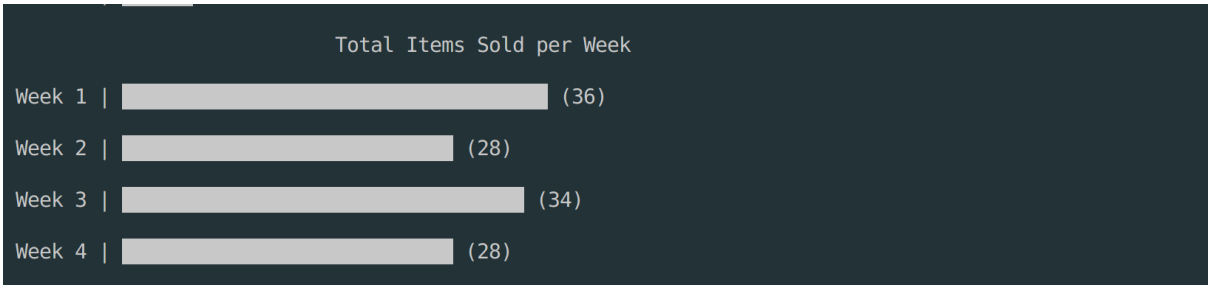


- **Total Items Sold per Week**

Computes the total number of items sold in each week.

```
// Chart 6: Total Items Sold per Week
// Stream API Methods: groupingBy (on week) + summingInt
Map<String, Integer> totalItemsSoldPerWeek = transactions.stream()
                .collect(Collectors.groupingBy(tx -> getWeekOfMonthLabel(tx.date()),
                                Collectors.summingInt(tx -> tx.items().stream().mapToInt(Item::quantity)
                                                .sum())));
```

This chart shows the total items sold per week:



Categorical Analytics

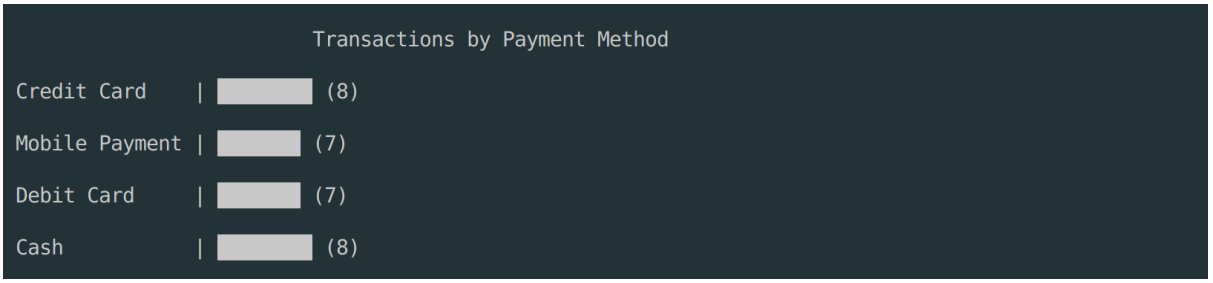
Insights based on non-time fields like payment method or transaction status.

- **Count of Transactions per Payment Method**

Transactions are categorized by payment method and their counts are calculated.

```
// Chart 7: Count of Transactions per Payment Method
// Stream API Methods: groupingBy + counting
Map<String, Long> countByPaymentMethod = transactions.stream()
                .collect(Collectors.groupingBy(Transaction::paymentMethod, Collectors.counting()));
```

The following chart shows how transactions are distributed across payment methods:

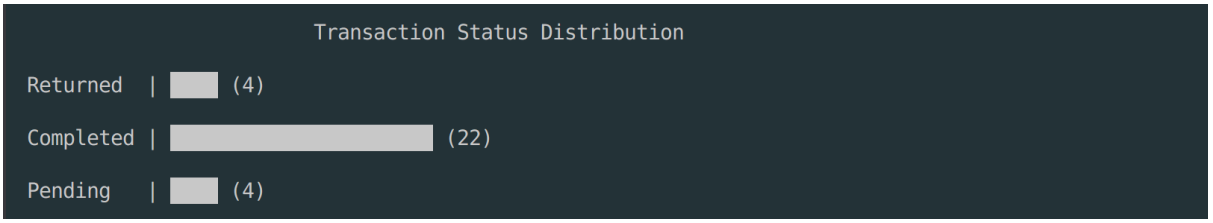


- **Transaction Status Distribution**

The number of transactions is calculated for each transaction status.

```
// Chart 8: Transaction Status Distribution
// Stream API Methods: groupingBy + counting
Map<String, Long> statusCounts = transactions.stream()
                .collect(Collectors.groupingBy(Transaction::transactionStatus, Collectors.counting()));
```

This chart represents how transactions are distributed by status:



Item-Level Analytics

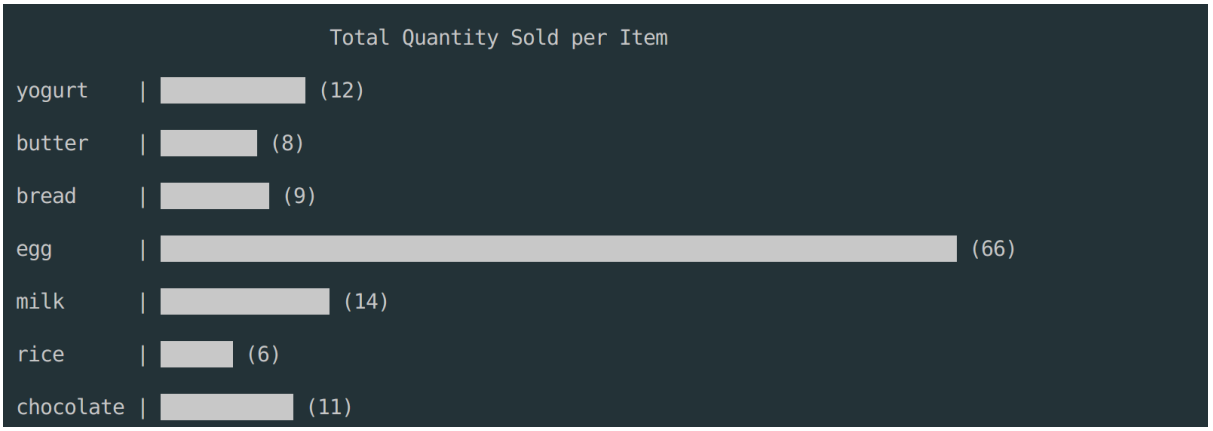
Analysis centered on items sold or their store placement.

- **Total Quantity Sold per Item**

Items are grouped by name, and their quantities are summed.

```java
// Chart 9: Total Quantity Sold per Item
// Stream API Methods: flatMap + groupingBy + summingInt
Map<String, Integer> quantityPerItem = transactions.stream()
                .flatMap(tx -> tx.items().stream())
                .collect(Collectors.groupingBy(Item::name, Collectors.summingInt(Item::quantity)));
```

The chart below displays how many units were sold for each item:

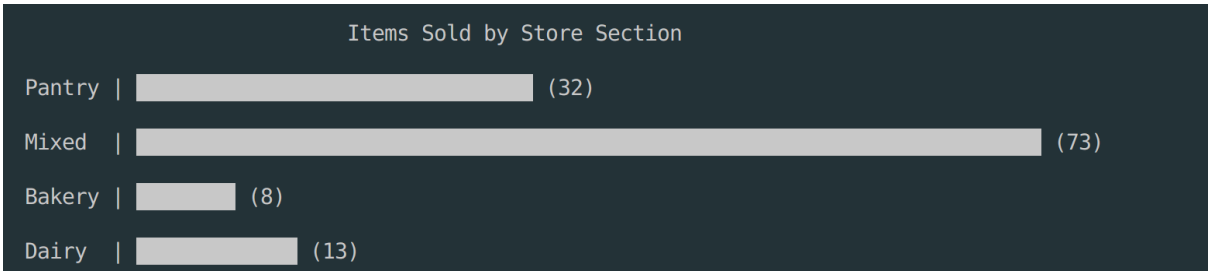


- **Items Sold by Store Section**

The total quantity of items is calculated per store section.

```java
// Chart 10: Items Sold by Store Section
// Stream API Methods: groupingBy + summingInt (nested stream for item
// quantities)
Map<String, Integer> itemsBySection = transactions.stream()
                .collect(Collectors.groupingBy(Transaction::storeSection,
                                Collectors.summingInt(tx -> tx.items().stream().mapToInt(Item::quantity)
                                                .sum())));
```

This chart shows item sales categorized by store sections:



Resources

- 📂 **GitHub Repository**: Complete Source Code
- ✉️ **Suggestions or Feedback?** Reach out via email: ehsan.ulhaq@live.com