**Exercise 1: Implementing Functional Interfaces with Lambda Expressions**

**Objectives:**

* Understand and implement functional interfaces.
* Use lambda expressions to simplify code.

**Business Scenario:**

You are developing an application that performs various operations on a list of customer orders. You need to implement functional interfaces for filtering and processing the orders based on different criteria.

**Tasks:**

1. **Create a New Java Project:**
   * Create a new Java project named **OrderProcessing**.
2. **Define Functional Interfaces:**
   * Define a functional interface **OrderFilter** with a method boolean **filter(Order order)**.
   * Define another functional interface **OrderProcessor** with a method void **process(Order order)**.
3. **Create the Order Class:**
   * Define an **Order** class with attributes like **orderId**, **customerName**, **orderAmount**, and **status**.
4. **Implement Lambda Expressions:**
   * In the **OrderProcessing** class, create a list of **Order** objects.
   * Use lambda expressions to implement **OrderFilter** for filtering orders with an amount greater than a specified value.
   * Use lambda expressions to implement **OrderProcessor** for processing orders by changing their status.
5. **Filter and Process Orders:**
   * Write a method that takes an **OrderFilter** and processes all orders that match the filter.
   * Write a method that takes an **OrderProcessor** and applies it to all orders.
6. **Test the Application:**
   * Create sample orders and test the filtering and processing methods.
   * Print the results to verify that the orders are correctly filtered and processed.

**Exercise 2: Using Stream API for Processing Collections**

**Objectives:**

* Use Stream API to process collections.
* Perform various operations such as filtering, mapping, and reducing on streams.

**Business Scenario:**

You are developing a sales analysis application that processes a list of sales records. You need to use the Stream API to analyze the sales data and generate reports.

**Tasks:**

1. **Create a New Java Project:**
   * Create a new Java project named **SalesAnalysis**.
2. **Define the SalesRecord Class:**
   * Define a SalesRecord class with attributes like **recordId**, **salesPerson**, **region**, **amount**, and **date**.
3. **Create Sample Data:**
   * In the **SalesAnalysis** class, create a list of **SalesRecord** objects with sample data.
4. **Filter Sales Records:**
   * Use the **Stream API** to filter sales records for a specific region.
   * Print the filtered records.
5. **Map and Transform Data:**
   * Use the **Stream API** to extract the sales amounts from the filtered records.
   * Print the sales amounts.
6. **Calculate Total Sales:**
   * Use the **Stream API** to calculate the total sales amount for the filtered records.
   * Print the total sales amount.
7. **Group Sales by SalesPerson:**
   * Use the **Stream API** to group sales records by **salesPerson**.
   * Print the grouped sales records.
8. **Generate Sales Report:**
   * Use the **Stream API** to generate a sales report that includes the total sales amount for each salesperson.
   * Print the sales report.

**Exercise 3: Advanced Stream Operations and Parallel Streams**

**Objectives:**

* Perform advanced operations using Stream API.
* Utilize parallel streams for improved performance.

**Business Scenario:**

You are enhancing the sales analysis application to include more complex analysis and improve performance using parallel streams.

**Tasks:**

1. **Update SalesRecord Class:**
   * Add additional attributes such as **productCategory** and quantity to the **SalesRecord** class.
2. **Filter and Sort Records:**
   * Use the **Stream API** to filter sales records for a specific product category and sort them by date.
   * Print the sorted records.
3. **Calculate Average Sales:**
   * Use the **Stream API** to calculate the average sales amount for a specific region.
   * Print the average sales amount.
4. **Find Top Sales Record:**
   * Use the **Stream API** to find the sales record with the highest amount.
   * Print the top sales record.
5. **Parallel Stream Operations:**
   * Use parallel streams to perform the filtering and sorting operations for improved performance.
   * Measure and print the time taken for both sequential and parallel stream operations.

**Exercise 4: Combining Functional Interfaces and Streams**

**Objectives:**

* Combine **functional interfaces** and **Stream API** for flexible and reusable code.
* Implement complex data processing pipelines.

**Business Scenario:**

You are tasked with developing a comprehensive data processing pipeline for customer feedback analysis. The pipeline should be flexible and reusable for different types of analysis.

**Tasks:**

1. **Define Functional Interfaces:**
   * Define functional interfaces **FeedbackFilter** and **FeedbackProcessor**.
2. **Create Feedback Class:**
   * Define a **Feedback** class with attributes like **feedbackId**, **customerName**, rating, and comments.
3. **Implement Data Processing Pipeline:**
   * Use the **Stream API** to create a flexible data processing pipeline that:
     + Filters feedback based on a minimum rating.
     + Maps feedback to extract customer names and comments.
     + Reduces feedback to count the number of positive and negative feedbacks.
4. **Implement Flexible Processing:**
   * Write methods that take **FeedbackFilter** and **FeedbackProcessor** as parameters to allow flexible and reusable processing.
   * Create lambda expressions to implement different filtering and processing strategies.
5. **Test the Pipeline:**
   * Create sample feedback data and test the data processing pipeline.
   * Print the results to verify the correct operation of the pipeline.