Assignment: eCart Application with Java 8 Stream API

Objective: To implement an eCart (electronic shopping cart) application using Java's Stream API, demonstrating the depth and breadth of its capabilities in a real-world scenario.

Instructions:
1. Ensure you're using Java 8 or a later version and have a suitable IDE.
2. Structure your application using best practices (e.g., Model-View-Controller).
3. Familiarize yourself with Java's Stream documentation to fully utilize its capabilities.
4. Ensure your code is well-commented, explaining the purpose and functionality of each operation.

Part 1: eCart Basics with Stream API
1. **Product Creation and Initialization**:
    - Create a **Product** class with attributes like **id**, **name**, **price**, and **rating**.
    - Initialize a list of products. Use **Stream.of()** to add products to the list.
2. **Filtering Products**:
    - Use the **filter()** method to list products below a certain price.
    - Use the **filter()** method again to list products with a rating above a certain threshold.
3. **Sorting and Mapping Products**:
    - Sort products based on their price and then by rating.
    - Use the **map()** method to transform a list of products into a list of their names or prices.

Sample Product Data:

```
List<Product> products = Arrays.asList(
 new Product(1, "Apple iPhone 13", 799.99, 4.5),
 new Product(2, "Samsung Galaxy S22", 749.99, 4.6),
 new Product(3, "Google Pixel 6", 699.99, 4.7),
 new Product(4, "Sony WH-1000XM4", 349.99, 4.8),
 new Product(5, "Apple MacBook Pro", 1299.99, 4.9),
 new Product(6, "Dell XPS 13", 1199.99, 4.4),
 new Product(7, "Amazon Echo Dot", 49.99, 4.2),
 new Product(8, "Apple Watch Series 7", 399.99, 4.6),
  new Product(9, "Bose QuietComfort 35 II", 299.99, 4.7),
 new Product(10, "GoPro HERO10 Black", 499.99, 4.5));
```

Part 2: Advanced Operations with eCart
1. **Product Statistics**:
    - Use the **summarizingDouble()** collector to get the count, sum, min, max, and average price of products.
2. **Cart Operations**:
    - Implement an **addToCart** method, where products can be added to a user's cart using their IDs.

    - Implement a **calculateTotal** method which uses the **reduce()** method to compute the total price of products in the cart.
3. **Product Grouping**:
    - Use the **groupingBy()** collector to group products by their rating.
    - Use the **partitioningBy()** collector to partition products based on a certain price threshold, for instance, separating premium products from regular ones.

Part 3: Parallel Operations with eCart
1. **Parallelized Product Search**:
    - Implement a method that searches for a product by its name using both sequential and parallel streams. Measure the performance difference.
2. **Handling Checkout**:
    - Implement the checkout process where the total price of products is calculated. First, use a sequential stream and then parallelize this process. Highlight the difference in performance, especially when dealing with a large number of products.

Submission:
1. Submit a zip file containing your Java source code files and any other resources used.
2. Include a **README.md** file explaining your approach for each task, with special emphasis on where and why you used specific Stream API features.
3. Your code should be well-organized, well-commented, and follow Java coding standards.

Evaluation Criteria:
1. Correctness and robustness of the eCart application.
2. Effective utilization of Stream operations in appropriate contexts.
3. Demonstrated understanding of differences between sequential and parallel streams in real-world scenarios.
4. Overall code quality, readability, and organization.

Bonus Challenge:
Expand the eCart application to:
1. Implement user accounts with wishlists. Users should be able to add products to their wishlist. Use Stream API features to list, sort, and filter wishlist products.
2. Implement product reviews. Each product can have multiple reviews. Use Stream API to filter, sort, and average product review ratings.