Industry-oriented assignments on the Java Stream API [using only wrapper data types]

Assignment 1: Sales Data Processing (Double)

Scenario

A retail store maintains a list of **daily sales amounts**. The manager wants to analyze the data.

Requirements

Using Stream API:

1. **Filter** sales greater than 5000.0.
2. **Sort** sales in **descending order**.
3. **Convert** each sale into **with GST included** (sale + sale*0.18).
4. **Find** the **highest sale**.
5. **Calculate** the **total sales amount** (sum).

Input Example

List<Double> sales = Arrays.asList(4500.0, 12000.0, 8000.0, 3000.0, 15000.0);

Expected Output (example)

- Filtered Sales (>5000) → [12000.0, 8000.0, 15000.0]
- Sorted Sales (desc) → [15000.0, 12000.0, 8000.0]
- With GST → [17700.0, 14160.0, 9440.0]
- Highest Sale → 15000.0
- Total Sales → 42500.0

👉 **Learning Outcome:** Students practice **filter, map, sorted, max, reduce, collect** with a real **retail sales use case**.

🚀 **Assignment 2: Student Marks Analysis (Integer & String)**

Scenario

A university system records students' marks and names. The exam cell needs to generate insights.

Requirements

Given:

List<Integer> marks = Arrays.asList(45, 67, 82, 39, 90, 55);

List<String> names = Arrays.asList("John", "Emma", "Alex", "Sophia", "Liam", "Olivia");

Tasks (use Stream API):

1. **Filter** passing marks (>= 50).
2. **Count** how many students passed.
3. **Sort** marks in ascending order.
4. **Find** the top score (highest).
5. From names list → **collect** names starting with "A" into a new list.
6. **Map** marks into grades:
    - >= 85 → A
    - >= 70 → B
    - >= 50 → C
    - < 50 → Fail

Expected Output (example)

- Passing Marks → [67, 82, 90, 55]
- Passed Count → 4
- Sorted Marks → [39, 45, 55, 67, 82, 90]
- Top Score → 90
- Names starting with A → [Alex]
- Grades → [Fail, C, B, Fail, A, C]

👉 **Learning Outcome:** Students practice **filter, count, sorted, max, collect, map** with real **student grading use case**.

✅ Both assignments keep things **industry-style but simple**, using only wrapper data types (Integer, Double, String).