Data Meter

Overview

Telecom operators collect data usage logs from various towers that detail mobile data consumption across different technologies (4G, 5G) and scenarios (Home, Roaming). The goal is to process multiple large text files (each containing 100K–200K rows) representing data usage logs and aggregate data consumption per mobile number, segregating it into the following categories:

- 4G (Home)
- 5G (Home)
- 4G Roaming
- 5G Roaming

Input Format

Each line of the input file has the following pipe (|) separated format:

Mobile Number|Tower|4G|5G|Roaming

- Mobile Number: A 10-digit mobile number
- Tower: A string representing the tower ID
- 4G: An integer denoting the 4G data usage in KB
- 5G: An integer denoting the 5G data usage in KB
- Roaming: Either "Yes" or "No", indicating if the data was consumed while roaming

Sample Input

9000600600|InAir1234|0|13456|No
9000600601|InAir125|1345|0|Yes

Expected Output Format

After processing all the files and aggregating the data per mobile number, the program should generate an output report like below:

Mobile Number|4G|5G|4G Roaming|5G Roaming|Cost
9000600600|0|13456|0|0|123
9000600601|0|0|1345|0|345

Requirements

1. Aggregate all input files: Data for the same number can be present in multiple files and should be accumulated.

2. Categorize Data: Classify usage into four categories based on data type (4G, 5G) and roaming flag.
3. 4G data has a rate card, 5G has a different and 4G roaming is 10% more than the regular data, while 5G roaming comes with 15% more
4. If a utilisation is beyond certain limit, there is a 5% more on top of the bill amount. This threshold will be a configurable property
5. Handle Large Input Efficiently: Expect large file sizes, so performance and memory handling are crucial.
6. Robust Exception Handling: Ensure clean handling of malformed lines, missing fields, file I/O errors, etc.
7. Object-Oriented Design: Use OOP principles (e.g., classes, inheritance, encapsulation) to design a modular and maintainable system.
8. Unit Testing: Implement JUnit test cases to validate core logic (parsing, aggregation, categorization).