Pseudo-code Problems - Day 5

1. Java OOPs

🔴 Difficult

1. Healthcare Diagnosis Engine

Concepts: OOPS, Strategy Pattern, Encapsulation, Interfaces

Story:

Apollo International Hospital (yes, your workplace makes a cameo here 😄) wants to digitize its preliminary diagnosis process.
Currently, junior doctors manually match patient symptoms with possible diseases, which is slow and inconsistent.

The hospital wants a **"Diagnosis Engine"** that can:

- Accept a set of symptoms as input.
- Apply **different diagnosis algorithms** (e.g., AI-based, rule-based, probability-based) depending on the patient's department.
- Easily switch or add new diagnosis methods without rewriting existing code.

Scenario:

1. A patient walks into **General Medicine** with fever, cough, and fatigue. The system uses a **RuleBasedDiagnosis** algorithm to suggest flu or COVID-19.
2. Another patient goes to **Neurology** with headache, dizziness, and nausea. The system uses an **AIBasedDiagnosis** algorithm that predicts migraine or vertigo.
3. Tomorrow, a new **GeneticPatternDiagnosis** algorithm needs to be added without touching old code.

Expected OOPS Approach:

- **Strategy Pattern:** Create a `DiagnosisStrategy` interface with a `diagnose(List<String> symptoms)` method.
- Implement strategies:
    - `RuleBasedDiagnosis`
    - `AIBasedDiagnosis`

Pseudo-code Problems - Day 5

    - `ProbabilityBasedDiagnosis`

- A `DiagnosisContext` class that can **switch strategies at runtime**.
- Demonstrate adding a **new algorithm** without modifying existing code.

---

2. Restaurant POS Billing System

Concepts: OOPS, Polymorphism, Inheritance, Interfaces

Story:

"Spice Villa" is a busy restaurant that wants to modernize its billing process.
Their menu items have different tax rules:

- **Food items** have 5% GST.
- **Beverages** have 12% GST.
- **Imported items** have additional customs duty.

Additionally, customers may get **different discount schemes**:

- Flat ₹100 off
- 10% off on bill
- No discount at all

The restaurant wants the POS system to:

- Apply the **right tax** dynamically depending on the item type.
- Apply the **right discount** dynamically depending on the ongoing offer.
- Allow easy addition of new tax or discount types without breaking existing code.

Scenario:

1. A customer orders:
    - Paneer Tikka (Food) – 5% GST
    - Coke (Beverage) – 12% GST
    - Imported Cheese (Imported Food) – 5% GST + Customs duty
      Bill: Apply “10% off” on the final amount.

2. Another day, the restaurant introduces a **"Buy 1 Get 1"** scheme without changing existing billing logic.

Pseudo-code Problems - Day 5

3. During festivals, a **new LuxuryTax** is introduced for premium items, and the system should handle it seamlessly.

Expected OOPS Approach:

- **TaxCalculator** as a base abstract class or interface, with subclasses:
  - `FoodTax`
  - `BeverageTax`
  - `ImportedTax`
- **DiscountPolicy** interface with:
  - `NoDiscount`
  - `FlatDiscount`
  - `PercentageDiscount`
- **Polymorphism:** Pass the right objects at runtime and let overridden methods handle logic.
- Show how **new taxes or discounts** can be plugged in without touching the main billing code.

---

2. Data Structures & Algorithms

🔴 Difficult

1. Distributed Load Balancer – Queue Simulation (Round-Robin with Dynamic Nodes)

Story Context:
You've joined the tech team of **StreamHub**, a video streaming platform similar to Netflix. Every time a user plays a video, the request must be processed by one of many **backend servers**. To ensure smooth streaming without overloading a single server, **StreamHub** uses a **distributed load balancer**.

Currently, the system needs an upgrade — the load balancer should distribute requests in **round-robin** fashion (Server 1 → Server 2 → Server 3 → back to Server 1), **while allowing new servers to be added or old ones to be removed without downtime**.

Problem Requirements:

- Use a **Queue** to represent the list of active servers.

Pseudo-code Problems - Day 5

- Each incoming request should be assigned to the server at the front of the queue, and that server should then be moved to the back.
- Support **dynamic addition and removal** of servers during runtime (simulate servers going online or offline).
- Process a given number of requests and print the mapping of request → server.

Example Flow:

1. Initial servers: `S1, S2, S3`
2. Requests: `R1 → S1`, `R2 → S2`, `R3 → S3`
3. Add a new server `S4` before the next request.
4. Requests continue: `R4 → S1`, `R5 → S2`, …
5. Remove `S2` from the queue if it fails.

Learning Points:

- Using **Queue** for cyclic processing.
- Handling **dynamic resource availability**.
- Simulating **real-time distributed systems**.

---

2. Autocomplete Search Suggestions – Trie Implementation

Story Context:
You've been hired by **FoodKart**, an online food delivery service, to improve its **search bar performance**. Customers often type partial restaurant names or dish names, and the search bar should **instantly suggest the most relevant matches**.

The challenge: The suggestion system must work efficiently even with **millions of keywords** (restaurant names, cuisines, dish names), and results must appear **in less than 50ms**.

Problem Requirements:

- Use a **Trie (prefix tree)** to store all searchable keywords.
- Implement a **search function** that returns **top N matches** for a given prefix.
- Allow insertion of new keywords (restaurants/dishes) dynamically.
- (Bonus) Maintain a **popularity score** for each keyword so that suggestions are sorted by popularity.

Pseudo-code Problems - Day 5

Example Flow:

1. Stored keywords:
   `"pizza hut", "pizza point", "pita bread", "pizzeria"`
2. User types `"pi"` → Suggestions: `"pizza hut", "pizza point", "pita bread"`.
3. If `"pita bread"` gets more orders, it appears higher in results.
4. Adding a new restaurant `"pineapple cafe"` should instantly reflect in results.

Learning Points:

- **Trie** for fast prefix-based lookups.
- Balancing **time complexity** vs. **real-time updates**.
- Extending DSA for **business-focused features** like ranking.

---

3. Collections, Generics, Streams

🔴 Difficult

1. Smart Grid Energy Usage Tracker

Concepts: `Map`, `Nested Map`, `List`, `Streams`, `Generics`

Story:
Imagine you are working for a Smart City Energy Department that manages electricity consumption data from multiple zones in the city. Each zone has several smart meters that send readings every day.

Your system needs to:

1. Store data as:
`Map<Zone, Map<Day, List<Reading>>>`

- **Zone: Represents a city zone (e.g., Zone-A, Zone-B).**
- **Day: Represents a specific date.**
- **List<Reading>: Contains readings collected from different meters in that zone on that day.**

Pseudo-code Problems - Day 5

2. **Real-life challenge:** The city wants to identify peak load times for each zone to avoid blackouts. Using Java Streams, you need to:
    - Traverse the nested `Map`.
    - Aggregate consumption data for each day.
    - Find the peak usage value for each zone.
3. **Example scenario:**
    - Zone-A, Day-2025-08-01: [120kW, 90kW, 200kW, 150kW] → Peak = 200kW
    - Zone-B, Day-2025-08-01: [300kW, 350kW, 250kW] → Peak = 350kW
    - Final output should show zone-wise and day-wise peak load reports.
4. **Why it's real-world:**
   This mimics how smart grids actually analyze energy consumption trends to plan load shedding and maintenance schedules.

---

2. Employee Bonus Distribution Engine

Concepts: `Streams`, `Custom Collector`, `Map`, `Generics`

Story:
You're part of the HR analytics team for a Fortune 500 company. At the end of the year, the company awards bonuses based on employee performance ratings.

Your system needs to:

1. Process a list of employees:
`List<Employee>`

where each `Employee` has:

- Name
- Department (e.g., IT, Finance, Sales)
- Performance rating (e.g., 1–5)
- Base salary

2. **Using Java Streams,** your tasks are:
    - Group employees by department.
    - Calculate the total bonus pool for each department.

Pseudo-code Problems - Day 5

- Distribute bonuses proportionally based on performance ratings.

3. **Custom Collector logic:**
   - Pool amount = 10% of total department salaries.
   - Employees with higher ratings get a bigger share.
   - Example: In IT dept with ₹1,00,000 pool →
     - Emp-A (rating 5) → ₹50,000
     - Emp-B (rating 3) → ₹30,000
     - Emp-C (rating 2) → ₹20,000

4. **Example scenario:**
   - Input: 1000 employees across 5 departments.
   - Output: A detailed department-wise bonus distribution report.

5. **Why it's real-world:**
   Bonus allocation is a core HR function in large organizations, and this model lets companies fairly distribute rewards while keeping the process data-driven.