📌 Problem Statement: Telecom Billing & Analytics System

Telecom operators face complex challenges in managing millions of subscribers, offering diverse tariff plans, and ensuring accurate billing while complying with regulatory and business rules. The system must be scalable, rule-driven, and capable of producing actionable analytics for business teams.

This project aims to design and implement a **Telecom Billing & Analytics Application** using **Java OOP concepts, Collections, and Stream API**, following a **three-layer architecture** (Presentation → Service → Repository).

🎯 Objectives

1. Implement telecom-specific business rules such as **fair usage policies (FUP), rollover benefits, roaming charges, referral discounts, credit control**, and **mobile number portability (MNP) restrictions**.
2. Demonstrate **extensive use of Java Collections and Stream API** for billing, rating, grouping, filtering, and reporting.
3. Showcase a **clean 3-layer architecture** separating concerns (UI/service/persistence).
4. Provide **domain-driven use cases** covering both operational and analytical needs.

📌 Use Cases / Tasks

1. Customer & Plan Management

- Add and manage customers with attributes like name, email, and referral status.
- Create and manage telecom plans with allowances (data, voice, SMS), rentals, and overage rates.
- Support **family/shared plans** where multiple subscriptions share pooled resources.

2. Subscription Lifecycle

- Customers can subscribe to plans (one or more SIMs).
- Subscriptions have lifecycle dates (start, end) and can belong to **family groups**.
- Support for **MNP (Mobile Number Portability)** with restrictions (e.g., no plan change during pending MNP).

3. Usage Tracking

- Collect **usage records** (CDR-style): Data (GB), Voice (minutes), SMS (count).

- Differentiate between **domestic vs. roaming** and **international usage**.
- Apply **night-time discounts** (reduced weightage between 00:00–06:00).
- Apply **weekend free minutes** rule for certain plans.

4. Billing & Rating

- Generate **monthly invoices** per subscription:
    - Base rental charges.
    - **Overage charges** when usage exceeds allowances.
    - **Roaming surcharges** (domestic % uplift, international flat charges).
    - **Referral discounts** for referred customers.
    - **Family fairness surcharge** if one member exceeds 60% of shared data pool.
    - **GST tax calculation** on subtotal.
- Support **data rollover** (unused data carried to next month with cap).
- Apply **credit control**: block services if invoices unpaid > 60 days.

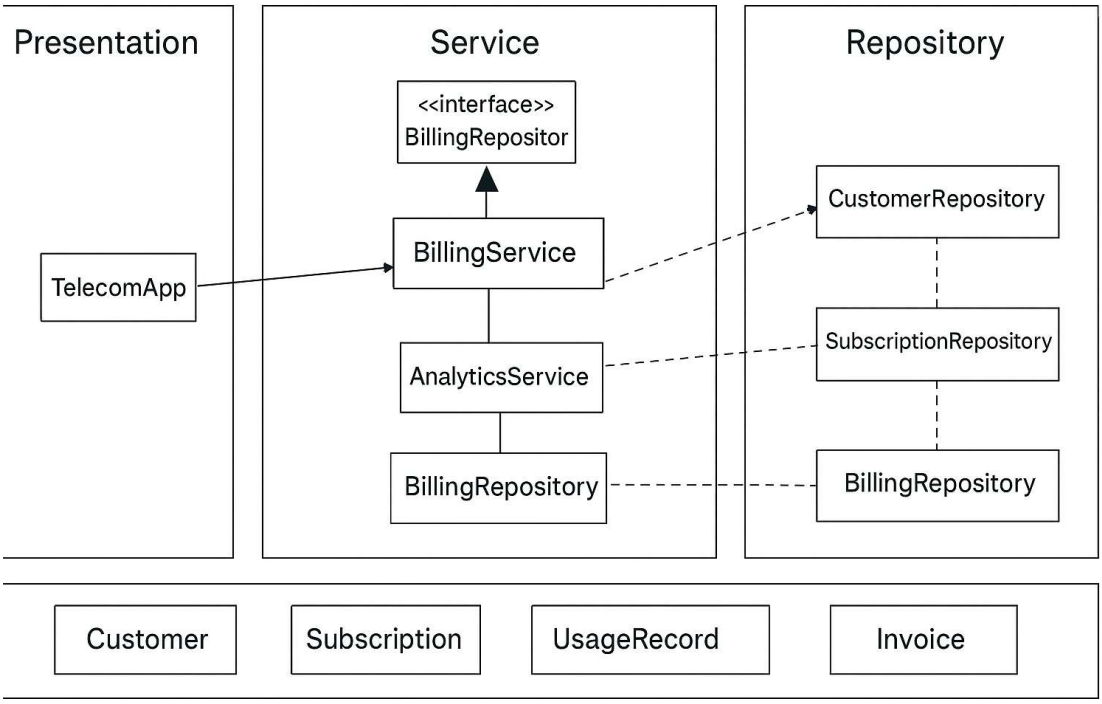
5. Reporting & Analytics (Stream API-heavy)

- **Top N data users** in a billing cycle.
- **ARPU (Average Revenue Per User)** by plan.
- **Overage distribution** (count, total, average overage per plan).
- **Credit risk detection**: customers with invoices unpaid > 60 days.
- **Plan recommendation engine**: suggest higher plans if a subscriber pays frequent overages.

🔑 **Expected Outcomes**

- A **working simulation** of a telecom billing cycle (seed data → usage records → invoices).
- **Itemized invoices** with base charges, surcharges, discounts, and taxes.
- **Analytics dashboards/reports** for management using Java Streams.
- Clean **separation of concerns** with Presentation (CLI/Driver), Service (business logic), Repository (in-memory persistence).

👉 This problem statement ensures learners/practitioners get exposure to **real-world telecom scenarios**, while also practicing **OOP, Collections, Stream API, and layered architecture** in Java.



Use Case Mapping :

Here’s the **mapping of each use case/task from the problem statement** to the actual classes and methods in the provided telecom Java solution:

📌 Use Case → Code Mapping

1. Customer & Plan Management

- **Add/manage customers** → `Customer` entity, `CustomerRepo`, `CustomerService` (CRUD-like ops).

- **Manage plans** → `Plan` entity, `PlanRepo`, `PlanService`.

- **Family/shared plans** → `Plan.familyShareCap`, `Subscription.familyId`, checked in `BillingService.generateInvoice()` for fairness surcharge.

---

2. Subscription Lifecycle

- **Customer subscriptions** → `Subscription` entity, `SubscriptionRepo`, `SubscriptionService`.

- **Lifecycle dates** → `Subscription.startDate`, `Subscription.endDate`.

- **Family groups** → `Subscription.familyId` field, used in data pool fairness checks.

- **MNP restriction** → `Subscription.mnpPending` flag, validated in `SubscriptionService.changePlan()`.

---

3. Usage Tracking

- **Usage records** → `UsageRecord` entity with fields: `dataGb`, `voiceMinutes`, `smsCount`, `roaming`, `international`, `nightTime`.

- **Domestic vs roaming/international** → `UsageRecord.roaming`, `UsageRecord.international`, checked in `BillingService.rateUsage()`.

- **Night-time discount** → Weighted calculation in `BillingService.applyNightDiscount()`.

- **Weekend free minutes** → Logic in `BillingService.calculateVoiceCharge()` when `Plan.weekendFreeVoice` is true.

---

4. Billing & Rating

- **Invoice generation** → `Invoice` entity, `BillingService.generateInvoice(subscription, usageRecords, month)`.

- **Base rental** → `Plan.monthlyRental` applied in invoice generation.

- **Overage charges** → `BillingService.calculateOverage()` uses `Stream` aggregation.

- **Roaming surcharges** → Applied in `BillingService.rateUsage()` (domestic uplift %, international flat).

- **Referral discounts** → `Customer.referredBy` checked in `BillingService.applyReferralDiscount()`.

- **Fairness surcharge (family pool)** → `BillingService.checkFamilyPoolUsage()` applies penalty if >60% pool consumed by one sub.

- **GST tax** → Applied at end in `BillingService.addTaxes()`.

- **Data rollover** → `BillingService.applyRollover()` uses previous month’s unused data, capped at 50% of allowance.

- **Credit control** → `Customer.creditBlocked` updated in `BillingService.applyCreditControl()` if invoices unpaid > 60 days.

---

5. Reporting & Analytics (Streams)

- **Top N data users** → `AnalyticsService.topDataUsers(usages, N)` uses `stream().sorted().limit(N)`.

- **ARPU by plan** → `AnalyticsService.arpuByPlan(invoices)` uses `Collectors.groupingBy()`, `Collectors.averagingDouble()`.

- **Overage distribution** → `AnalyticsService.overageDistribution(invoices)` with `Collectors.summarizingDouble()`.

- **Credit risk detection** → `AnalyticsService.detectCreditRisk(invoices)` filters unpaid > 60 days.

- **Plan recommendations** → `AnalyticsService.recommendPlans(invoices, plans)` uses `Collectors.groupingBy()` + overage checks.

✅ This mapping ties **business tasks → Java entities → service logic → Stream API usage**. It ensures that the entire solution is not just theoretical but **traceable to real telecom functions**.

Do you also want me to prepare a **diagram (UML + 3-layer + data flow)** that visually maps these use cases to classes and flow between layers? That would make it training-material ready.