1. Hypothesis Brief

Applicable Rules to Explain the Tap Log Charges:

- R1: Base Fare is ₹25 (independent of line)
  Most non-zero fares are equal to or multiples of 25, indicating a base fare of ₹25.
- R2: Peak Period Windows
  Peak periods are defined as 8 AM–10 AM and 6 PM–8 PM. Some fare variations correspond to these timeframes.
- R3: Transfer Window of 30 Minutes from Last Full Fare Tap
  Zero fares (₹0) appear if the tap occurs within 30 minutes after a paid tap, representing a free transfer window.
- R4: Night Discount – 20% Discount on Base Fare from 10 AM to Midnight
  Reduced fares such as ₹20 or ₹37.5 reflect a 20% discount on the base fare (25 × 0.8 = 20).
- R5: Post-midnight Discount – 35% Discount from Midnight to 4 AM
  Post-midnight fares like ₹16.25 (25 × 0.65 = 16.25) indicate this discount.

Testability:

- R1: Confirm base fare is ₹25 on charged taps outside discount and transfer periods.
- R2 & R3: Validate fare changes during peak periods and free transfer periods within 30 minutes of last paid tap.
- R4 & R5: Verify fares during discount windows reflect correct discounted amounts.

2. Class Design Note

- TariffEngine: Central orchestrator that applies a chain of fare rules to each tap event.
- FareRule (Interface): Abstracts fare calculation logic so individual rules can be implemented modularly and toggled easily.
- BaseFareRule: Implements R1; returns a fixed base fare regardless of line or time.
- PeakPeriodRule: Implements R2; adjusts fares during peak hours as applicable.
- TransferWindowRule: Implements R3; checks if a tap is within the transfer window to apply zero fare.
- NightDiscountRule: Implements R4; applies a 20% discount between 10 AM and midnight.

- PostMidnightDiscountRule: Implements R5; applies a 35% discount from midnight to 4 AM.
- Tap: Represents a tap event including datetime, line, station code, and charged fare.
- FareContext: Maintains state such as last paid tap time for transfer window and other context-sensitive rules.

This design ensures each class has a single responsibility, rules can be enabled/disabled via booleans for quick A/B testing, and adding new fare policies remains straightforward.

3. Code Implementation Summary (Based on Provided Python Code)

The fare calculation engine uses a modular, rule-based architecture where each fare rule is encapsulated as a class implementing a common interface. Rules can be toggled on or off dynamically to facilitate A/B testing and iterative improvements.

Key Components:

- TariffEngine Class:
  - Stores a list of enabled fare rules based on configuration flags.
  - Maintains `lastPaidTaps` to track last paid tap for transfer window logic.
  - Its `computeFare(Tap tap)` method iterates over enabled rules and applies them sequentially. The latest non-null fare overrides earlier ones.
  - Adds the tap to `lastPaidTaps` if the computed fare is greater than zero.
- TapEvent (Tap) Class:
  Holds tap data such as datetime, line, charged fare, and station code. Date/time strings are parsed into appropriate datetime objects.
- FareRule Interface (or Abstract Class):
  Declares `BigDecimal apply(Tap tap, List<Tap> lastPaidTaps)`.
  Implementers either return a fare or null if rule doesn't apply.
- Implemented FareRule Classes:
  - BaseFareRule (R1): Returns the fixed base fare (₹25).
  - PeakPeriodRule (R2): Returns adjusted fare if tap occurs within peak periods.
  - TransferWindowRule (R3): Returns zero fare if tap occurs within 30 minutes of the last paid tap.
  - NightDiscountRule (R4): Applies 20% discount between 10 AM and midnight.
  - PostMidnightDiscountRule (R5): Applies 35% discount between midnight and 4 AM.

Fare Calculation Workflow:

- Start `fare` as null.
- For each enabled rule, call `apply()` and if it returns a fare, overwrite the current fare.
- Final fare is the last non-null fare returned by rules.
- If fare > 0, record this tap as last paid tap for future reference.

Toggleable Rules:

- Rules are enabled or disabled at initialization using Boolean flags.
- Allows easy switching for A/B tests without code refactoring.
- Modular and extendable design to incorporate new rules smoothly.