Problem Set: OOP, Collections, Exceptions

Solutions: <Link>

Metro gate audit

- Problem statement: Given a stream of (cardId, timeSlot), record the first time each card is flagged in first-seen order; on any re-scan of a flagged card, raise an exception including cardId and current timeSlot.
- Relevant concepts: LinkedHashSet (uniqueness + order), Map for first-seen metadata, custom checked exception, try/catch.
- Problem inverse: No duplicate cardId should pass undetected or alter its first-seen time.
- Required behaviors: Record each new cardId exactly once and preserve first-seen order; throw AccessDeniedException on re-scan with cardId and timeSlot; provide list of flagged cardIds in first-seen order; provide lookup for cardId to first timeSlot.
- Error handling: Null/empty cardId or timeSlot is invalid; querying unknown cardId returns a clear not-found signal.
- Hint: A LinkedHashMap<cardId, firstTimeSlot> captures order and metadata in one place.

Cafeteria express lanes

- Problem statement: Route customers into Express or Regular lanes via a runtime-selected policy; when asked for N, output the next N to be billed—Express first, then Regular.
- Relevant concepts: Strategy interface, polymorphism, ArrayDeque as Queue.
- Problem inverse: Switching policies must not change queuing code—only the injected policy object.
- Required behaviors: Policy A allows Express if itemCount ≤ limit or hasExpressPass; Policy B is stricter (ignores pass, lowers limit); enqueue to lane-specific queues preserving arrival order; on request, print next N names, Express before Regular.

- Error handling: Reject negative itemCount or empty name; policy must be non-null.

- Hint: Define LanePolicy.allow(customer) (and optional priority()); inject different implementations.

Bengaluru rain alert dedupe

- Problem statement: From a sequence of sensorIds, accept only the first alert per sensor in arrival order; on a repeat, throw DuplicateAlertException with sensorId and the index of its first occurrence.

- Relevant concepts: LinkedHashMap for stable order + first index, custom exception.

- Problem inverse: Each sensorId appears at most once in the accepted list; repeats never change stored order or index.

- Required behaviors: Maintain accepted sensorIds in strict first-appearance order; on duplicate, throw with sensorId and original index; support queries: accepted list, total unique sensors, first index for a sensorId.

- Error handling: Null/empty sensorId is invalid; duplicates consistently raise the same exception.

- Hint: LinkedHashMap<sensorId, firstIndex> gives O(1) lookup and stable order.

College fest workshop signup

- Problem statement: For each workshop with capacity, confirm the first capacity signups, waitlist overflow, and on cancellation promote exactly one from the waitlist; support a variant policy that chooses a different candidate than FIFO.

- Relevant concepts: Abstract class for shared state and hooks, List for confirmed, Queue for waitlist, Map workshop to policy instance.

- Problem inverse: Capacity is never exceeded; a student cannot be both confirmed and waitlisted for the same workshop.

- Required behaviors: Add signup and confirm if seats remain, else enqueue to waitlist; when cancelling, remove from confirmed and promote exactly one from waitlist; views: confirmed list, waitlist, remaining seats per workshop; variant hook for who gets promoted.

- Error handling: Duplicate signup for same student+workshop is invalid; cancel for non-enrolled student is invalid; capacities must be positive.

- Hint: Abstract EnrollmentPolicy with template method selectNextFromWaitlist() (FIFO vs custom).

Ride fare dispute logger

- Problem statement: Track disputes per rideId so only one dispute is open per ride at a time; multiple rides may have disputes; closing requires a non-empty reason and stores closedAt.

- Relevant concepts: Map<rideId, Dispute>, enum-like state, validation, custom exceptions.

- Problem inverse: No second dispute can open on the same open ride; cannot close without a reason.

- Required behaviors: open(rideId, passenger, description) creates OPEN if none is open; close(rideId, reason) sets CLOSED and records reason + closedAt; queries: isOpen(rideId), details(rideId), countOpen().

- Error handling: Opening on an already-open rideId should throw DisputeAlreadyOpenException; closing unknown or already-closed rideId should throw clear exception; empty reason or blank description is invalid.

- Hint: Keep Dispute with fields rideId, state, openedAt, closedAt, reason; enforce transitions centrally.

Library quiet-zone seats

- Problem statement: Manage seats 1..S so each seat has at most one student and each student at most one seat; support assign, release, lookups, and list free seats in ascending order.

- Relevant concepts: Map<seat, student>, Map<student, seat>, TreeSet (or sorted free-seat structure), custom exceptions.

- Problem inverse: No two students share a seat; no student holds two seats.

- Required behaviors: Assign seat X to student; if taken, throw SeatTakenException naming occupant; release seat X after verifying ownership; lookups: who sits at X, which seat a student holds, list free seats ascending.

- Error handling: Seat outside 1..S is invalid; releasing a free seat or by the wrong student throws a clear exception.

- Hint: Maintain both maps to keep the bijection and a sorted set of free seats for quick reporting.