You are hired by Pakistan Express to prototype a small parcel storage module for the Lahore depot. Use Java and arrays only (no collections). When a query returns multiple results, return them as arrays. State changes must occur only through public methods. Model a parcel with a class that holds a tracking code (PKX-2025-000123), a type chosen from [REGULAR, FRAGILE], and a status chosen from [IN_STORAGE, OUT_FOR_DELIVERY]. Identity and type are set in the constructor; status may change only via your operations that succeed or fail sensibly. The parcel must provide a concise textual summary showing code, type, and status. Add one concrete subclass for fragile shipments with a single extra field (either an insurance flag or a handling surcharge); storage rules do not change for this subclass.

Represent storage with racks composed of slots arranged in jagged rows where row lengths may differ. Each slot has a 1-based ID formatted as R<row>-S<twoDigits> (e.g., R3-S07), holds either one parcel or is empty, and supports safe store(...) and remove() operations. A slot's string form should be a compact token so rows are easy to scan: use "[--]" for empty and "[R]" or "[F]" for REGULAR and FRAGILE parcels, respectively. A rack is constructed from an int[] of row lengths (for example {3,4,3}) and synthesizes all slot IDs at construction time. It must provide a helper getSlotById(String id) with defensive checks, a printLayout() method that prints one line per row using the slot tokens, a getParcelsByType(ParcelType) method that returns a Parcel[] produced via two-pass filtering (count then allocate/fill), and simple reports for capacity and occupied count.

The Lahore depot manages an array of racks and offers operations to add racks, to store or remove a parcel by slot ID by routing to the rack that owns that ID, to compute totals (capacity versus occupied) across all racks, and to find the first empty slot using a fixed policy: scan left-to-right, top-to-bottom across racks and rows. In the demo (main), construct the Lahore depot, attach at least one jagged rack, and print the initial layout. Create a few parcels, including one FragileParcel. Store one parcel into R3-S02 and print the layout; attempt to store another parcel into the same slot to show rejection and print again; remove the parcel and print to confirm the slot is empty; then list all FRAGILE parcels in that rack and print the returned count. Submit source files for ParcelType, ParcelStatus, Parcel, FragileParcel, Slot, Rack, Depot, and the Demo.

Use the Demo class provided below without modification; your program must compile and run with it, and your output should closely match the Sample Run.

```
public class Demo {
    public static void main(String[] args) {
        Depot lahore = new Depot("Lahore Depot", 4);
        Rack rack1 = new Rack("LR-1", new int[]{3, 4, 3});
        lahore.addRack(rack1);                                          // [5]
        System.out.println("Initial layout:");
        lahore.printAllLayouts();
        Parcel r1 = new Parcel("PKX-2025-000111", ParcelType.REGULAR);
        Parcel r2 = new Parcel("PKX-2025-000222", ParcelType.REGULAR);
        Parcel f1 = new Parcel("PKX-2025-000333", ParcelType.FRAGILE);
  System.out.println("Store r1 into R3-S02: " + lahore.store("R3-S02", r1));// [5]
        System.out.println("Attempt to store r2 into SAME R3-S02 (should fail): "
                + lahore.store("R3-S02", r2));
        Parcel removed = lahore.remove("R3-S02");
        System.out.println("Remove from R3-S02: " + (removed != null ? removed :
"null")); // [5]
        String target = lahore.findFirstEmptySlot();
        System.out.println("First empty slot (policy): " + target);
```

```
        System.out.println("Store FRAGILE at " + target + ": " +
lahore.store(target, f1)); // [5]
        System.out.println("\nCurrent layout after actions:");
        lahore.printAllLayouts();
        Parcel[] frags = rack1.getParcelsByType(ParcelType.FRAGILE); // [5]
        System.out.println("FRAGILE in rack " + rack1.getRackId() + ": " +
frags.length);
        for (Parcel p : frags) System.out.println("  - " + p);
        int[] totals = lahore.getTotals();
        System.out.println("Depot totals: Capacity=" + totals[0] + ", Occupied=" +
totals[1]);
    }
}
```

Sample Run

Initial layout:

=== Depot: Lahore Depot ===

Rack LR-1 layout:

R1: [--] [--] [--]

R2: [--] [--] [--] [--]

R3: [--] [--] [--]

Capacity=10, Occupied=0

Totals for Lahore Depot: Capacity=10, Occupied=0

Store r1 into R3-S02: true

Attempt to store r2 into SAME R3-S02 (should fail): false

Remove from R3-S02: PKX-2025-000111 | Regular | Out for Delivery

First empty slot (policy): R1-S01

Store FRAGILE at R1-S01: true

Current layout after actions:

=== Depot: Lahore Depot ===

Rack LR-1 layout:

R1: [F] [--] [--]

R2: [--] [--] [--] [--]

R3: [--] [--] [--]

Capacity=10, Occupied=1

Totals for Lahore Depot: Capacity=10, Occupied=1

FRAGILE in rack LR-1: 1

 - PKX-2025-000333 | Fragile | In Storage

Depot totals: Capacity=10, Occupied=1