**Intern Exercise: Ola Booking System Using Collections**

**Objective:**

Build a mini Ola Booking System using **Java Collections**: List, Set, Queue, and Map.

The goal is to help interns **practice creating classes, taking user input, and using multiple collections together** in a real-world scenario.

**Step-by-Step Instructions**

**Step 1: Think About the Classes**

* **Passenger / Booking Class**
  + Fields to include:
    - passengerName (String)
    - destination (String)
  + Create a constructor to initialize these fields.
  + Override toString() to easily display the booking info.

**Step 2: Decide What Collections to Use**

1. **List** → To store **all bookings**, including duplicates.
2. **Set** → To store **unique passenger names**.
3. **Queue** → To manage **drivers waiting to be assigned**.
4. **Map** → To store **passenger → assigned driver**.

**Step 3: Initialize Collections**

* Create **List, Set, Queue, and Map** objects.
* Add some driver names to the **Queue** (e.g., Driver1, Driver2, Driver3).

**Step 4: Take Input from User**

* Ask the user: “How many bookings do you want to make?”
* For each booking:
  1. Take **passenger name** as input.
  2. Take **destination** as input.
  3. Create a new **Booking object**.
  4. Add it to the **List** (all bookings).
  5. Add the **name to Set** (unique passengers).
  6. Assign the **next driver from the Queue** and add it to the **Map**.

**Step 5: Display Results**

* Print all bookings from the **List**.
* Print unique passengers from the **Set**.
* Print passenger → driver mapping from the **Map**.

**Hints**

* Use ArrayList for List, HashSet for Set, LinkedList for Queue, and HashMap for Map.
* Use queue.poll() to get the next available driver.
* If no driver is available, show "No driver available".

**Optional Challenge**

* After displaying results, allow new bookings.
* If a driver finishes a ride, **add them back to the driver Queue**.

**Full Solution Code**

import java.util.\*;

class Booking {

String passengerName;

String destination;

Booking(String passengerName, String destination) {

this.passengerName = passengerName;

this.destination = destination;

}

public String toString() {

return passengerName + " -> " + destination;

}

}

public class OlaBookingSystem {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

// Collections

List<Booking> allBookings = new ArrayList<>();

Set<String> uniquePassengers = new HashSet<>();

Queue<String> driverQueue = new LinkedList<>();

Map<String, String> rideAssignments = new HashMap<>();

// Initialize drivers

driverQueue.add("Driver1");

driverQueue.add("Driver2");

driverQueue.add("Driver3");

System.out.print("Number of bookings: ");

int n = sc.nextInt();

sc.nextLine(); // consume newline

for (int i = 0; i < n; i++) {

System.out.print("Passenger " + (i + 1) + " name: ");

String name = sc.nextLine();

System.out.print("Passenger " + (i + 1) + " destination: ");

String destination = sc.nextLine();

Booking booking = new Booking(name, destination);

allBookings.add(booking);

uniquePassengers.add(name);

if (!driverQueue.isEmpty()) {

String driver = driverQueue.poll();

rideAssignments.put(name, driver);

} else {

rideAssignments.put(name, "No driver available");

}

}

// Display results

System.out.println("\nAll Bookings:");

for (Booking b : allBookings) {

System.out.println(b);

}

System.out.println("\nUnique Passengers:");

System.out.println(uniquePassengers);

System.out.println("\nRide Assignments:");

for (Map.Entry<String, String> entry : rideAssignments.entrySet()) {

System.out.println(entry.getKey() + " -> " + entry.getValue());

}

sc.close();

}

}

**✅ Learning Outcomes**

* Create a **custom class** with fields and methods.
* Use **List, Set, Queue, and Map** together in a single problem.
* Understand **first-come-first-serve logic** with a queue.
* Learn to manage **duplicates, unique values, and mappings** in collections.