Java Report

By: Shreyans Jain

2347155

Table of Contents

1. **Introduction to Collection Framework in Airline Management**

The integration of the Java Collection Framework in the Airline Management System enhances the efficiency of data handling, storage, and retrieval. This report explores the application of various collection interfaces within the context of an Airline Management System.

2. **Collection Interface in Airline Management**

The Collection interface serves as the cornerstone for organizing and managing data structures within the Airline Management System. It provides a unified approach to handle various types of collections, facilitating seamless interactions between different components of the system.

3. **List Interface in Airline Management**

3.1 ArrayList for Passenger Lists

The ArrayList implementation is employed to maintain dynamic lists of passengers. This allows for easy modification and real-time updates to the list of individuals traveling on each flight.

3.2 LinkedList for Flight Schedules

LinkedList is utilized to represent the flight schedule, enabling swift insertion and deletion of flights. This ensures the adaptability of the schedule to changing operational requirements.

3.3 Vector for Cabin Crew Rosters

Vector is employed to manage the rosters of cabin crew members. Its synchronized nature ensures thread safety, crucial for managing the dynamic schedules of airline staff.

3.4 Stack for Luggage Loading Operations

The Stack class is implemented to streamline luggage loading operations. The Last-In, First-Out (LIFO) nature of Stack suits scenarios where the order of luggage loading is critical.

4. Set Interface in Airline Management

4.1 HashSet for Airport Codes

HashSet is used to store unique airport codes, facilitating quick retrieval and validation of airport locations within the Airline Management System.

4.2 LinkedHashSet for Boarding Gate Sequences

LinkedHashSet is employed to maintain the sequence of boarding gates. This ensures an orderly boarding process for passengers.

4.3 TreeSet for Aircraft Seat Arrangements

TreeSet is applied for organizing and managing the arrangement of seats within an aircraft. This enables the system to maintain a sorted order for seat allocation.

5**. Queue Interface in Airline Management**

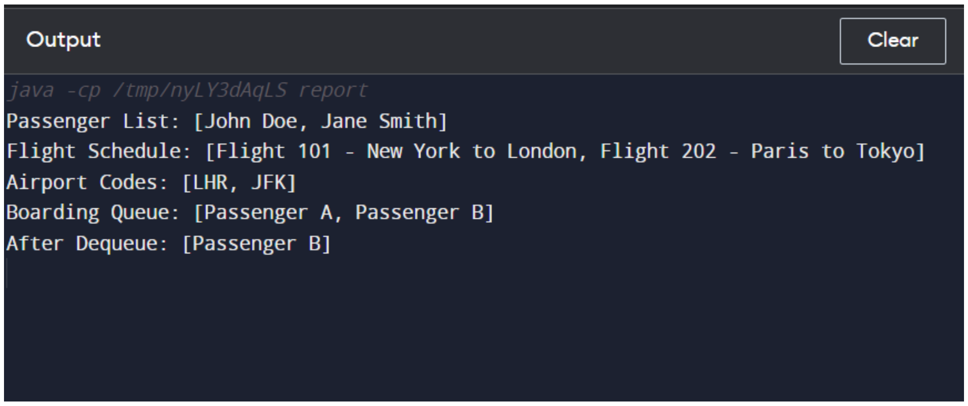
The Queue interface is leveraged to enhance the passenger boarding process. Queues are employed to manage the order in which passengers board the aircraft, ensuring a smooth and organized boarding experience.

In summary, the integration of the Java Collection Framework within the Airline Management System optimizes data management, contributing to the seamless operation of various processes within the airline industry.

CODE -:

```
import java.util.ArrayList;

import java.util.LinkedList;

import java.util.HashSet;

import java.util.Queue;


public class AirlineManagementSystem {


    public static void main(String[] args) {
        // Example usage of the classes


        // ArrayList for Passenger Lists
        PassengerList passengerList = new PassengerList();
        passengerList.addPassenger("John Doe");
        passengerList.addPassenger("Jane Smith");
        System.out.println("Passenger List: " + passengerList.getPassengers());


        // LinkedList for Flight Schedules
        FlightSchedule flightSchedule = new FlightSchedule();
        flightSchedule.addFlight("Flight 101 - New York to London");
        flightSchedule.addFlight("Flight 202 - Paris to Tokyo");
        System.out.println("Flight Schedule: " + flightSchedule.getFlights());

        // HashSet for Airport Codes
        AirportCodes airportCodes = new AirportCodes();
        airportCodes.addAirportCode("JFK");
        airportCodes.addAirportCode("LHR");
        System.out.println("Airport Codes: " + airportCodes.getAirportCodes());

        // Queue for Boarding Process
        BoardingQueue boardingQueue = new BoardingQueue();
        boardingQueue.enqueue("Passenger A");
        boardingQueue.enqueue("Passenger B");
        System.out.println("Boarding Queue: " + boardingQueue.getBoardingQueue());
        boardingQueue.dequeue();
        System.out.println("After Dequeue: " + boardingQueue.getBoardingQueue());
    }
}

class PassengerList {
    private ArrayList<String> passengers;

    public PassengerList() {
        passengers = new ArrayList<>();
    }
    public void addPassenger(String passengerName) {
        passengers.add(passengerName);
    }

    public ArrayList<String> getPassengers() {
        return passengers;
    }
}

class FlightSchedule {
    private LinkedList<String> flights;

    public FlightSchedule() {
        flights = new LinkedList<>();
    }

    public void addFlight(String flightDetails) {
        flights.add(flightDetails);
    }

    public LinkedList<String> getFlights() {
        return flights;
    }
}

class AirportCodes {
    private HashSet<String> airportCodes;

    public AirportCodes() {
        airportCodes = new HashSet<>();
    }

    public void addAirportCode(String code) {
        airportCodes.add(code);
    }

    public HashSet<String> getAirportCodes() {
        return airportCodes;
    }
}

class BoardingQueue {
    private Queue<String> boardingQueue;

    public BoardingQueue() {
        boardingQueue = new LinkedList<>();
    }

    public void enqueue(String passenger) {
        boardingQueue.add(passenger);
    }

    public void dequeue() {
        if (!boardingQueue.isEmpty()) {
            boardingQueue.poll();
        }
    }

    public Queue<String> getBoardingQueue() {
        return boardingQueue;
    }
}
```

OUTPUT -:



```
java -cp /tmp/nyLY3dAqLS report
Passenger List: [John Doe, Jane Smith]
Flight Schedule: [Flight 101 - New York to London, Flight 202 - Paris to Tokyo]
Airport Codes: [LHR, JFK]
Boarding Queue: [Passenger A, Passenger B]
After Dequeue: [Passenger B]
```