✈️ Flight Management System (Java)

A simple **console-based Flight Management System** built in Java that allows users to manage flights, make bookings, and handle cancellations. Data is persisted in CSV files for reusability across program runs.

📌 Features

- **Flight Management**
  - Add new flights
  - List all available flights
  - Search flights by origin and destination
- **Booking System**
  - Book seats on a flight
  - Cancel existing bookings
  - View all bookings
- **Data Persistence**
  - Flights and bookings are saved into `flights.csv` and `bookings.csv`
  - Data automatically loads on startup
  - Sample flights are generated if no data exists

🛠️ Technologies Used

- **Language:** Java 17+
- **File Storage:** CSV (`flights.csv`, `bookings.csv`)
- **Libraries:** Core Java (no external dependencies)

Future Enhancements

- GUI interface with JavaFX or Swing
- Database integration (MySQL/PostgreSQL) instead of CSV
- Advanced search (by date, price, airline)
- User authentication system

Code:

```
package flightManagementSystem;


import java.io.*;
import java.nio.file.*;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.*;
import java.util.concurrent.atomic.AtomicInteger;
import java.util.stream.Collectors;


public class FlightManagementSystem {


   // ===== Flight Class =====
   static class Flight {
      private static final AtomicInteger ID_GEN = new AtomicInteger(1000);
      private final int id;
      private String airline;
      private String origin;
      private String destination;
      private LocalDateTime departure;
      private int capacity;
      private int seatsAvailable;
      private double price;
      private static final DateTimeFormatter fmt = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm");


      public Flight(String airline, String origin, String destination, LocalDateTime
departure, int capacity, double price) {
```

```
        this.id = ID_GEN.getAndIncrement();
        this.airline = airline;
        this.origin = origin;
        this.destination = destination;
        this.departure = departure;
        this.capacity = capacity;
        this.seatsAvailable = capacity;
        this.price = price;
    }

    public Flight(int id, String airline, String origin, String destination, LocalDateTime
departure, int capacity, int seatsAvailable, double price) {
        this.id = id;
        ID_GEN.updateAndGet(curr -> Math.max(curr, id + 1));
        this.airline = airline;
        this.origin = origin;
        this.destination = destination;
        this.departure = departure;
        this.capacity = capacity;
        this.seatsAvailable = seatsAvailable;
        this.price = price;
    }

    public int getId() { return id; }
    public String getAirline() { return airline; }
    public String getOrigin() { return origin; }
    public String getDestination() { return destination; }
    public LocalDateTime getDeparture() { return departure; }
    public int getCapacity() { return capacity; }
```

```
    public int getSeatsAvailable() { return seatsAvailable; }
    public double getPrice() { return price; }

    public synchronized boolean bookSeats(int seats) {
        if (seats <= 0 || seats > seatsAvailable) return false;
        seatsAvailable -= seats;
        return true;
    }

    public synchronized boolean cancelSeats(int seats) {
        if (seats <= 0 || seatsAvailable + seats > capacity) return false;
        seatsAvailable += seats;
        return true;
    }

    public String toCsvLine() {
        return String.format("%d,%s,%s,%s,%s,%d,%d,%.2f",
                id, airline, origin, destination, departure.format(fmt), capacity, seatsAvailable,
price);
    }

    public static Flight fromCsvLine(String line) {
        String[] parts = line.split(",", -1);
        int id = Integer.parseInt(parts[0]);
        String airline = parts[1];
        String origin = parts[2];
        String destination = parts[3];
        LocalDateTime departure = LocalDateTime.parse(parts[4], fmt);
        int capacity = Integer.parseInt(parts[5]);
```

```
            int seatsAvailable = Integer.parseInt(parts[6]);
            double price = Double.parseDouble(parts[7]);
            return new Flight(id, airline, origin, destination, departure, capacity,
seatsAvailable, price);
        }

        @Override
        public String toString() {
            return String.format("Flight %d | %s | %s -> %s | %s | Seats: %d/%d | $%.2f",
                    id, airline, origin, destination, departure.format(fmt), seatsAvailable, capacity,
price);
        }
    }

    // ===== Booking Class =====
    static class Booking {
        private static final AtomicInteger ID_GEN = new AtomicInteger(5000);
        private final int bookingId;
        private final int flightId;
        private final String passengerName;
        private final int seatsBooked;

        public Booking(int flightId, String passengerName, int seatsBooked) {
            this.bookingId = ID_GEN.getAndIncrement();
            this.flightId = flightId;
            this.passengerName = passengerName;
            this.seatsBooked = seatsBooked;
        }
```

```
public Booking(int bookingId, int flightId, String passengerName, int seatsBooked) {
    this.bookingId = bookingId;
    ID_GEN.updateAndGet(curr -> Math.max(curr, bookingId + 1));
    this.flightId = flightId;
    this.passengerName = passengerName;
    this.seatsBooked = seatsBooked;
}

public int getBookingId() { return bookingId; }
public int getFlightId() { return flightId; }
public String getPassengerName() { return passengerName; }
public int getSeatsBooked() { return seatsBooked; }

public String toCsvLine() {
    return String.format("%d,%d,%s,%d", bookingId, flightId, passengerName, seatsBooked);
}

public static Booking fromCsvLine(String line) {
    String[] p = line.split(",", -1);
    int bid = Integer.parseInt(p[0]);
    int fid = Integer.parseInt(p[1]);
    String name = p[2];
    int seats = Integer.parseInt(p[3]);
    return new Booking(bid, fid, name, seats);
}

@Override
public String toString() {
    return String.format("Booking %d | Flight %d | Passenger: %s | Seats: %d",
```

```
                bookingId, flightId, passengerName, seatsBooked);
        }
    }

    // ===== FlightManager Class =====
    static class FlightManager {
        private final Map<Integer, Flight> flights = new HashMap<>();
        private final Map<Integer, Booking> bookings = new HashMap<>();
        private final Path flightsFile = Paths.get("flights.csv");
        private final Path bookingsFile = Paths.get("bookings.csv");

        public FlightManager() {
            load();
        }

        public synchronized Flight addFlight(String airline, String origin, String destination,
LocalDateTime departure, int capacity, double price) {
            Flight f = new Flight(airline, origin, destination, departure, capacity, price);
            flights.put(f.getId(), f);
            return f;
        }

        public List<Flight> listAllFlights() {
            return
flights.values().stream().sorted(Comparator.comparing(Flight::getDeparture)).collect(Collectors.toList());
        }

        public List<Flight> searchFlights(String origin, String dest) {
            return flights.values().stream()
```

```
                .filter(f -> f.getOrigin().equalsIgnoreCase(origin) &&
f.getDestination().equalsIgnoreCase(dest))
                .sorted(Comparator.comparing(Flight::getDeparture))
                .collect(Collectors.toList());
        }

        public Flight getFlight(int id) { return flights.get(id); }

        public synchronized Optional<Booking> book(int flightId, String passengerName, int seats) {
            Flight f = flights.get(flightId);
            if (f == null) return Optional.empty();
            boolean ok = f.bookSeats(seats);
            if (!ok) return Optional.empty();
            Booking b = new Booking(flightId, passengerName, seats);
            bookings.put(b.getBookingId(), b);
            return Optional.of(b);
        }

        public synchronized boolean cancelBooking(int bookingId) {
            Booking b = bookings.get(bookingId);
            if (b == null) return false;
            Flight f = flights.get(b.getFlightId());
            if (f == null) return false;
            boolean ok = f.cancelSeats(b.getSeatsBooked());
            if (!ok) return false;
            bookings.remove(bookingId);
            return true;
        }
```

```
public List<Booking> listAllBookings() {
    return new ArrayList<>(bookings.values());
}

public synchronized void save() {
    try (BufferedWriter wf = Files.newBufferedWriter(flightsFile)) {
        for (Flight f : flights.values()) {
            wf.write(f.toCsvLine());
            wf.newLine();
        }
    } catch (IOException e) {
        System.err.println("Error saving flights: " + e.getMessage());
    }

    try (BufferedWriter wb = Files.newBufferedWriter(bookingsFile)) {
        for (Booking b : bookings.values()) {
            wb.write(b.toCsvLine());
            wb.newLine();
        }
    } catch (IOException e) {
        System.err.println("Error saving bookings: " + e.getMessage());
    }
}

public final synchronized void load() {
    flights.clear();
    bookings.clear();

    if (Files.exists(flightsFile)) {
```

```
        try (BufferedReader r = Files.newBufferedReader(flightsFile)) {
            String line;
            while ((line = r.readLine()) != null) {
                if (!line.trim().isEmpty()) {
                    Flight f = Flight.fromCsvLine(line);
                    flights.put(f.getId(), f);
                }
            }
        } catch (IOException e) {
            System.err.println("Error loading flights: " + e.getMessage());
        }
    } else {
        createSampleFlights();
    }

    if (Files.exists(bookingsFile)) {
        try (BufferedReader r = Files.newBufferedReader(bookingsFile)) {
            String line;
            while ((line = r.readLine()) != null) {
                if (!line.trim().isEmpty()) {
                    Booking b = Booking.fromCsvLine(line);
                    bookings.put(b.getBookingId(), b);
                }
            }
        } catch (IOException e) {
            System.err.println("Error loading bookings: " + e.getMessage());
        }
    }
}
```

```
    private void createSampleFlights() {
        addFlight("AirBlue", "Karachi", "Lahore",
LocalDateTime.now().plusDays(1).withHour(9).withMinute(30), 120, 150.00);
        addFlight("PakiAir", "Lahore", "Islamabad",
LocalDateTime.now().plusDays(1).withHour(13).withMinute(0), 100, 80.00);
        addFlight("SkyWays", "Karachi", "Islamabad",
LocalDateTime.now().plusDays(2).withHour(6).withMinute(45), 150, 200.00);
    }
  }

  // ===== Main Program =====
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    FlightManager manager = new FlightManager();
    DateTimeFormatter dtf = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm");

    System.out.println("=== Flight Management System ===");
    boolean running = true;
    while (running) {
      System.out.println("\n1) List Flights\n2) Add Flight\n3) Search Flights\n4) Book Seats\n5) Cancel
Booking\n6) List Bookings\n7) Save Data\n0) Exit");
      System.out.print("Choose: ");
      String choice = sc.nextLine().trim();

      switch (choice) {
        case "1":
          manager.listAllFlights().forEach(System.out::println);
          break;
```

```
            case "2":
                try {
                    System.out.print("Airline: "); String airline = sc.nextLine();
                    System.out.print("Origin: "); String origin = sc.nextLine();
                    System.out.print("Destination: "); String dest = sc.nextLine();
                    System.out.print("Departure (yyyy-MM-dd HH:mm): "); LocalDateTime
dep = LocalDateTime.parse(sc.nextLine(), dtf);
                    System.out.print("Capacity: "); int cap = Integer.parseInt(sc.nextLine());
                    System.out.print("Price: "); double price =
Double.parseDouble(sc.nextLine());
                    System.out.println("Added: " + manager.addFlight(airline, origin, dest, dep,
cap, price));
                } catch (Exception e) {
                    System.out.println("Error: " + e.getMessage());
                }
                break;
            case "3":
                System.out.print("Origin: "); String o = sc.nextLine();
                System.out.print("Destination: "); String d = sc.nextLine();
                List<Flight> results = manager.searchFlights(o, d);
                if (results.isEmpty()) System.out.println("No flights found.");
                else results.forEach(System.out::println);
                break;
            case "4":
                try {
                    System.out.print("Flight ID: "); int fid = Integer.parseInt(sc.nextLine());
                    System.out.print("Passenger Name: "); String pname = sc.nextLine();
                    System.out.print("Seats: "); int seats = Integer.parseInt(sc.nextLine());
                    Optional<Booking> b = manager.book(fid, pname, seats);
```

```
                System.out.println(b.map(value -> "Booked: " + value).orElse("Booking failed."));
            } catch (Exception e) {
                System.out.println("Error: " + e.getMessage());
            }
            break;
        case "5":
            try {
                System.out.print("Booking ID: "); int bid = Integer.parseInt(sc.nextLine());
                System.out.println(manager.cancelBooking(bid) ? "Booking cancelled." : "Cancel failed.");
            } catch (Exception e) {
                System.out.println("Error: " + e.getMessage());
            }
            break;
        case "6":
            manager.listAllBookings().forEach(System.out::println);
            break;
        case "7":
            manager.save();
            System.out.println("Data saved.");
            break;
        case "0":
            running = false;
            manager.save();
            System.out.println("Exiting...");
            break;
        default:
            System.out.println("Invalid option.");
    }
}
```

```
    }
}
```