**Case Study: Airline Booking System using Packages and Threading**

**Objective:**

The objective of this case study is to design and implement an airline booking system using Java, demonstrating the use of packages, abstract classes, multithreading, and the Factory Method Design Pattern. The system allows users to book domestic or international flights, view all bookings, and filter bookings based on flight type.

**Requirements:**

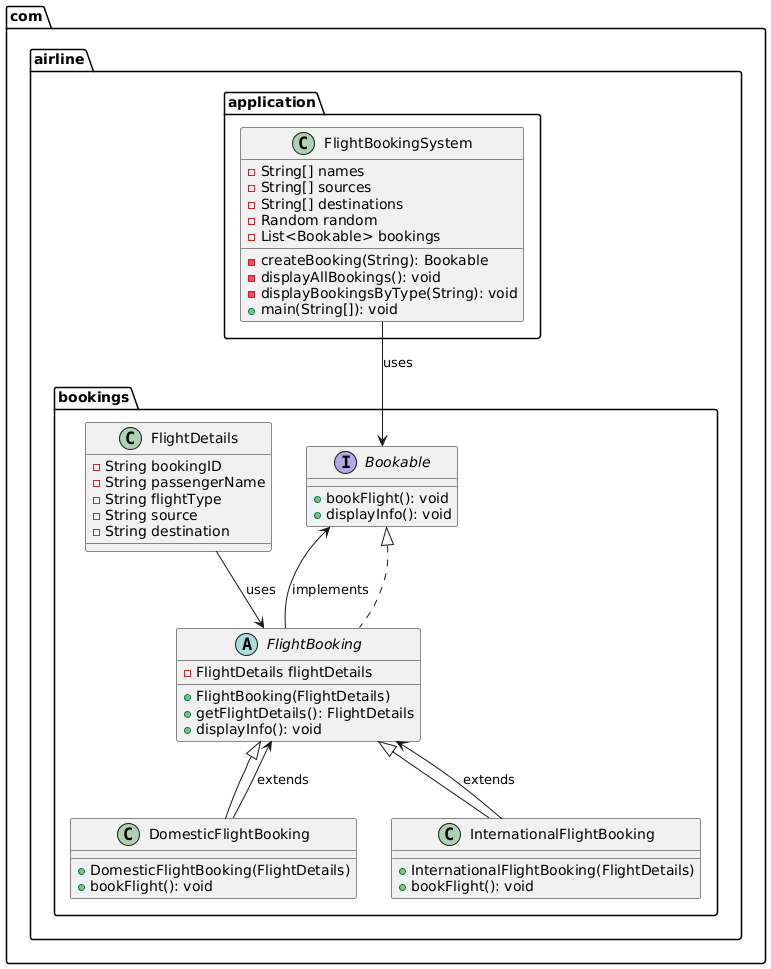
1. The system should support booking of both **Domestic** and **International** flights.
2. It should generate a unique **Booking ID** for each flight booking.
3. Bookings should be handled in a multithreaded manner to simulate real-time booking operations.
4. The system should display all bookings or filter bookings by flight type.
5. It should follow modular design principles using Java packages.

**System Architecture:**

**Packages:**

* com.airline.bookings: Contains the classes that define the flight booking logic and details.
* com.airline.application: Contains the main application class for user interaction.

**Class Diagram:**



**Class Descriptions:**

**FlightDetails (record class)**

* A record class that stores the details of a flight booking, including bookingID, passengerName, flightType, source, and destination.
* Provides a concise and immutable way to store flight-related data.

**Bookable (interface)**

* An interface that defines two methods: bookFlight() and displayInfo().
* Establishes a contract for any class that wants to implement flight booking behavior.

**FlightBooking (abstract class implementing Bookable)**

* An abstract base class for all types of flight bookings.
* Holds a reference to FlightDetails and implements the displayInfo() method using StringBuilder for efficient string concatenation.
* Declares an abstract method bookFlight() for subclasses to implement specific booking logic.

**DomesticFlightBooking (concrete subclass)**

* Represents a booking for domestic flights.
* Extends FlightBooking and implements the bookFlight() method to confirm a domestic flight booking.

**InternationalFlightBooking (concrete subclass)**

* Represents a booking for international flights.
* Extends FlightBooking and implements the bookFlight() method to confirm an international flight booking.

**FlightBookingSystem (main class)**

* The main application class that handles user input and manages the flight booking process.
* Provides options to book domestic or international flights, display all bookings, and filter bookings by flight type.
* Uses a List<Bookable> to store all bookings and leverages the Bookable interface for handling various types of bookings.
* Utilizes random data generation for flight details and incorporates Scanner for user input.

**Implementation Details:**

The system uses the **Factory Method Design Pattern** in the createBooking() method of the FlightBookingSystem class. This method creates an instance of either DomesticFlightBooking or InternationalFlightBooking based on the flightType provided by the user. This pattern is effective here as it abstracts the creation logic and allows for easy extension if new flight types are introduced.

**Multithreading** is used to simulate real-time booking by creating a BookingThread for each booking operation. This simulates concurrent processing and provides a better user experience by handling multiple booking requests simultaneously.

**Features and Functionality:**

1. **Book a Domestic Flight**:
   * Creates a new instance of DomesticFlightBooking using the Factory Method.
   * Starts a new thread to handle the booking and adds it to the list of bookings.
2. **Book an International Flight**:
   * Creates a new instance of InternationalFlightBooking.
   * Starts a new thread for the booking operation.
3. **Display All Bookings**:
   * Iterates through the list of bookings and displays the details of each booking.
4. **Display Bookings by Flight Type**:
   * Filters and displays bookings based on user-provided flight type (Domestic or International).
5. **Exit**:
   * Ends the program.

**Advantages of the Design:**

1. **Modular and Maintainable**:
   * The use of packages separates the booking logic from the main application, making the code easier to maintain.
2. **Extensibility**:
   * New flight types can be added without modifying the core booking logic, thanks to the Factory Method pattern.
3. **Concurrency**:
   * The use of multithreading provides a realistic simulation of handling concurrent booking requests.

**Conclusion:**

The Airline Booking System designed here effectively uses key object-oriented principles such as abstraction, inheritance, and polymorphism, along with design patterns like the Factory Method. It provides a robust, modular, and scalable solution that can be extended to include more features, such as payment integration, seat selection, and user accounts.

This case study demonstrates how to leverage Java packages, multithreading, and design patterns to build a real-world application with a clean and maintainable architecture.