FCDS
Programming II

ALEXANDRIA UNIVERSITY
جامعة الإسكندرية
كلية الحاسبات وعلوم البيانات

Flight Booking Management System

Project Overview

Design and implement a Flight Booking Management System that simulates the operations of a travel agency. This project will help you apply Object-Oriented Programming principles learned in your Java courses to create a practical system that manages flight bookings, customer information, and travel itineraries.

System Users

- **Customers**: Travelers who book flights and manage their reservations
- **Travel Agents**: Staff who assist customers with bookings and manage flight information
- **System Administrator**: Manages system settings and user access

Core Features

1. User Authentication and Profile Management

Features:

- Secure login system with username and password
- Role-based access (customer, agent, administrator)
- User profile creation and management
- Session handling and logout functionality

Requirements:

- Passwords must be at least 6 characters with letters and numbers
- Store user information securely in files
- Implement proper validation for all input fields
- Display appropriate error messages for failed authentication

Sample Scenario:

- Customer enters username "traveler22" and password "trip2023"
- System validates credentials against stored data
- If valid, system displays customer dashboard with booking options
- If invalid, system shows appropriate error message

2. Flight Management Features:

- Flight creation and management
- Search flights by various criteria (origin, destination, date)
- Flight schedule viewing and updates
- Seat availability tracking

Flight Data:

- Flight number, airline, origin, destination
- Departure and arrival times
- Available seat classes (Economy, Business, First Class)
- Pricing for each seat class
- Current seat availability

Sample Scenario:

- Agent selects "Add New Flight" option
- System prompts for flight details (number, airline, route, schedule, pricing)
- System validates data (no duplicate flight numbers, valid airports, etc.)
- System adds flight to database and confirms creation
- Flight becomes available in search results

3. Booking Management Features:
- New booking creation
- Booking modification and cancellation
- Passenger information management
- Booking confirmation and itinerary generation

Booking Data:
- Booking reference number
- Customer information
- Flight details
- Seat selection
- Payment status
- Special requests (meal preferences, assistance needs)

Sample Scenario:
- Customer searches for flights from Cairo to London on specific dates
- System displays available flights matching criteria
- Customer selects a flight and provides passenger details
- System creates booking with status "Reserved"
- System generates booking reference and displays confirmation
- Customer receives booking summary

4. Payment and Ticketing Features:
- Payment processing (simulated)
- Multiple payment methods
- Booking status tracking
- E-ticket generation and delivery

Payment Data:
- Payment amount and currency
- Payment method (Credit card, bank transfer, etc.)
- Transaction date and time
- Payment status

Sample Scenario:
- Customer selects "Complete Payment" for a reserved booking
- System displays payment options
- Customer selects payment method and enters details
- System validates payment information
- System updates booking status to "Confirmed"
- System generates e-ticket and displays for printing/saving

Object-Oriented Design Requirements

The system must be designed following these object-oriented principles:

1. Inheritance
- **Implementation Requirements**:
  - Create a base User class with common attributes
  - Extend User with Customer, Agent, and Administrator subclasses
  - Implement a Flight class hierarchy for different flight types
  - Create a Booking class that can be extended for different booking categories
- **Example**:
  - User defines common authentication behavior
  - Customer extends User with customer-specific attributes and methods
  - Agent extends User with agent-specific capabilities

    - Different flight types (Domestic, International) can have specialized behaviors

2. Encapsulation

- **Implementation Requirements**:
    - Make all class attributes private
    - Provide public getter and setter methods with appropriate validation
    - Hide implementation details within classes
    - Use proper constructors for object initialization
- **Example**:
    - Flight pricing details are private, accessible only through methods
    - Booking information can only be modified through controlled methods
    - Payment processing details are hidden from other system components
    - Password data is encapsulated with one-way access

3. Polymorphism

- **Implementation Requirements**:
    - Create methods with the same name but different behaviors in subclasses
    - Use parent class references to work with different object types
    - Implement common interfaces for similar operations
- **Example**:
    - calculatePrice() works differently for different flight classes
    - generateTicket() produces different formats based on booking type
    - Payment processing handles different payment methods through common interface

4. Abstraction

- **Implementation Requirements**:
    - Create abstract classes for concepts that shouldn't be instantiated directly
    - Define interfaces for common behaviors
    - Hide complex implementation details behind simple method calls
- **Example**:
    - User as an abstract class
    - PaymentProcessor interface implemented by different payment strategies
    - Complex pricing rules hidden behind a simple calculateFare() method

5. Class Relationships

- **Implementation Requirements**:
    - Use composition to establish "has-a" relationships
    - Create proper associations between related classes
    - Implement aggregation when appropriate
- **Example**:
    - A Customer has multiple Booking objects
    - A Flight has multiple Seat objects
    - A Booking has one or more Passenger objects
    - The BookingSystem class serves as the central coordinator

Technical Requirements Data Storage

File Requirements:

- Create separate text files for:
    - Users (users.txt) - store user credentials and roles
    - Flights (flights.txt) - store flight information
    - Bookings (bookings.txt) - store booking details
    - Passengers (passengers.txt) - store passenger information

- Use consistent formatting for easy reading/writing
- Implement proper file handling with try-catch blocks

UML Diagrams Class Diagram:

- Show all classes with attributes and methods
- Include inheritance relationships
- Show composition/aggregation relationships
- Display multiplicity on relationships

User Interface

- Create a console-based menu system
- Display different menus based on user role
- Provide clear feedback messages
- Format output for readability (tables, borders, etc.)

Bonus Features

1. Simple GUI

- Create a basic graphical interface using Java Swing
- Implement forms for flight search and booking
- Display flight schedules in organized grids
- Include navigation menus and dashboard views

2. Database Storage

- Replace file storage with a simple database (SQLite recommended)
- Create appropriate tables for all entities
- Implement proper data relationships
- Use prepared statements for database operations

Key Classes and Their Responsibilities

1. User (Abstract Class)

- **Responsibility**: Base class for all system users
- **Key Attributes**: userId, username, password, name, email, contactInfo
- **Key Methods**: login(), logout(), updateProfile()

2. Customer (Extends User)

- **Responsibility**: Manages customer bookings and profiles
- **Key Attributes**: customerId, address, bookingHistory, preferences
- **Key Methods**: searchFlights(), createBooking(), viewBookings(), cancelBooking()

3. Agent (Extends User)

- **Responsibility**: Manages bookings and assists customers
- **Key Attributes**: agentId, department, commission
- **Key Methods**: manageFlights(), createBookingForCustomer(), modifyBooking(), generateReports()

4. Administrator (Extends User)

- **Responsibility**: Manages system settings and user access
- **Key Attributes**: adminId, securityLevel
- **Key Methods**: createUser(), modifySystemSettings(), viewSystemLogs(), manageUserAccess()

5. Flight

- **Responsibility**: Stores flight information and manages seats
- **Key Attributes**: flightNumber, airline, origin, destination, departureTime, arrivalTime, availableSeats, prices
- **Key Methods**: checkAvailability(), updateSchedule(), calculatePrice(), reserveSeat()

6. **Booking**
    - **Responsibility**: Manages booking information and status
    - **Key Attributes**: bookingReference, customer, flight, passengers, seatSelections, status, paymentStatus
    - **Key Methods**: addPassenger(), calculateTotalPrice(), confirmBooking(), cancelBooking(), generateItinerary()
7. **Passenger**
    - **Responsibility**: Stores passenger information
    - **Key Attributes**: passengerId, name, passportNumber, dateOfBirth, specialRequests
    - **Key Methods**: updateInfo(), getPassengerDetails()
8. **Payment**
    - **Responsibility**: Handles payment information and processing
    - **Key Attributes**: paymentId, bookingReference, amount, method, status, transactionDate
    - **Key Methods**: processPayment(), validatePaymentDetails(), updateStatus()
9. **BookingSystem**
    - **Responsibility**: Central coordinator for the booking system
    - **Key Attributes**: users, flights, bookings, payments
    - **Key Methods**: searchFlights(), createBooking(), processPayment(), generateTicket()
10. **FileManager**
    - **Responsibility**: Handles data persistence using files
    - **Key Methods**: saveUsers(), loadUsers(), saveFlights(), loadFlights(), saveBookings(), loadBookings()