Getting Ready: The Airline Management System

→ An airline management system is a software used to manage all activities of the airline system efficiently.

→ Management system to digitize the process of scheduling flights, managing staff, ticket reservations & performing other necessary airline management tasks.

→ The system keeps track of the number of Aircraft, pilots and their availability at airpoints.

→ similarly the admin can manage all the airline activities with the help of this system.

Expectation from the Interviewer

→ How will the system ensure that multiple users do not have the same seat on the aircraft?

→ Can one itinerary reserve multiple flight?

→ Can the customer reserve the who aircraft?

→ What payment method can the customer use, for example, credit cash or cash?

→ How does the customer make the payment? does the customer pay online or in person?

→ Will the customer be able to pay in advance for a flight booking, or is a just-in-time (JIT) payment method available?

→ Is the price set manually, or does the system calculate the price for each flight?

→ Does every seat has the same price, or is it calculated based on the seat type?

→ Does the weekdays & weekends affects the flight of the flight?
→ Is the price of the flight affected by an increase in demand?
→ How does the duration of the flight affect the payment?
→ Can the customer cancel a flight?
→ What is the time limit to cancel the flight?
→ which type of users are allowed to request a flight cancellation.

Requirement Collection

R1: A customer should be able to search for flights by the date, departure, & destination airport.

R2: A customer should be able to reserve tickets for available flights. Customer should also be able to book multiple flights at once.

R3: The customer should be allowed to book multiple seat for a single flight.

R4: The system should allow the customer to check flight details, such as available seats, flight schedule, and departure/arrival time.

R5: The admin should be able to add new flights. The admin should be able to update or cancel scheduled flights.

R6: An airline should be able to own multiple aircrafts. The admin should be able to add aircraft to the system.

R7: An airline should be able to operate flights from different airports.

R8: The admin should be able to assign pilots & crew members to flights effectively

R9: The customer should be able to cancel their previous reservation.

R10: The customer should be able to make payments against their flight.

R11: The front desk officer should be able to reserve tickets, create itinaries & make flight payments for the customer.

R12: The flight crew should be able to view the schedule for their assigned flight.

R13 : The System should send the customer a notification whenever a reservation has been made or canceled or when their is an update for this flight.

Actors

Primary Actors

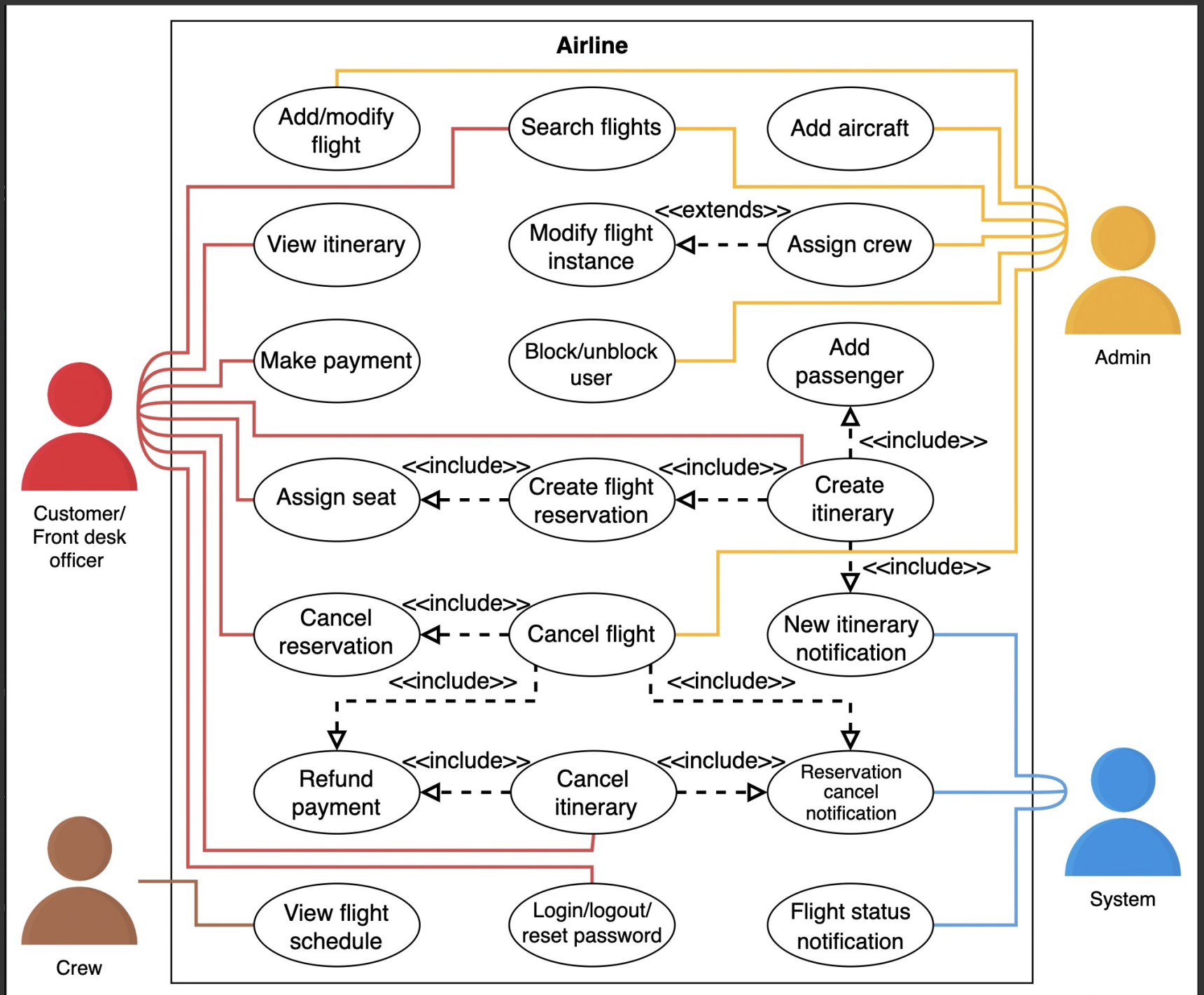
- Customer
- Front Desk Offiier
- Admin

Secondary Actors

- System
- Crew

Customer	Front desk officer	Admin	System	Crew
Cancel reservation	Cancel reservation	Add aircraft	New itinerary notification	View flight schedule
Login/logout	Login/logout	Add/modify flight	Reservation cancel notification	
Reset password	Reset password	Block/unblock users	Flight status notification	
Create itinerary	Create itinerary	Assign crew		
Assign seat	Assign seat	Cancel flight		
Search flights	Search flights	Search flights		
Make payment	Make payment			
View itinerary	View itinerary			
Cancel itinerary	Cancel itinerary			

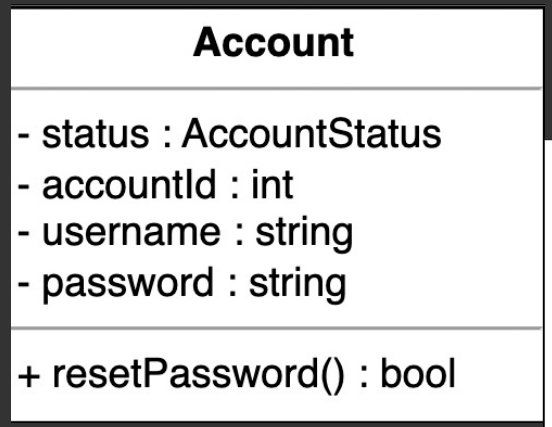
Use Case Diagram

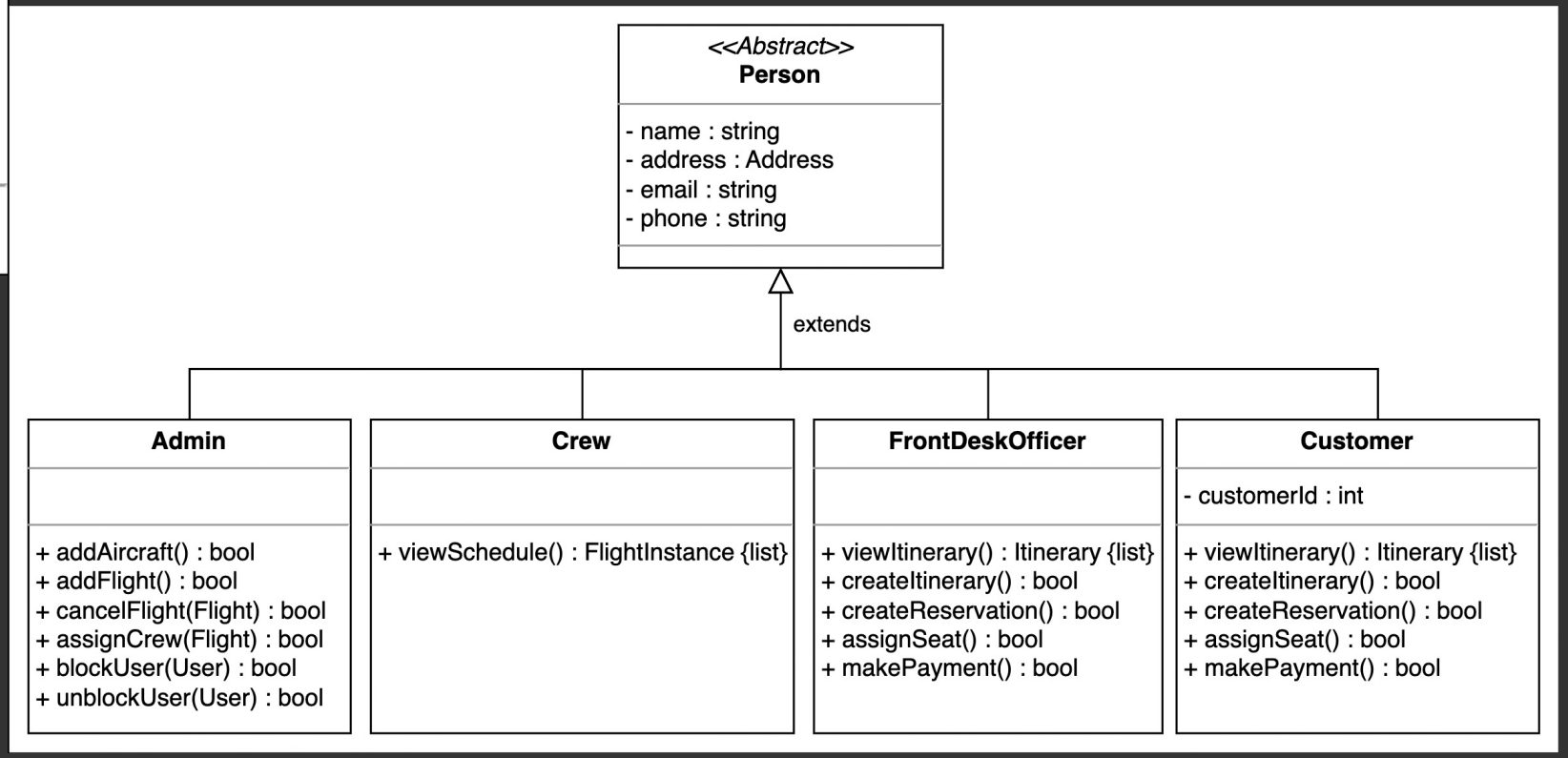


Class Diagram

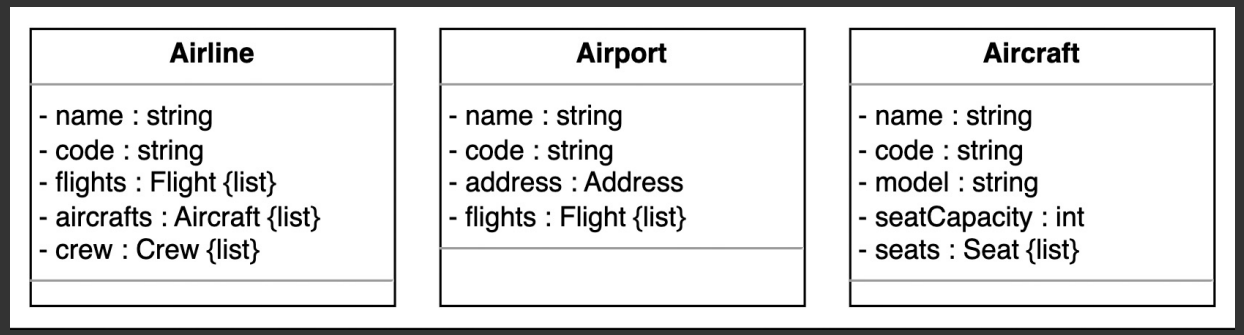
1. Account

2. Person

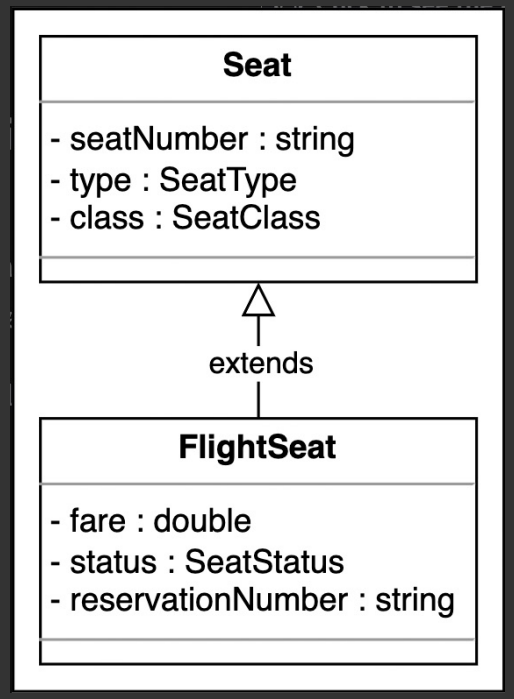




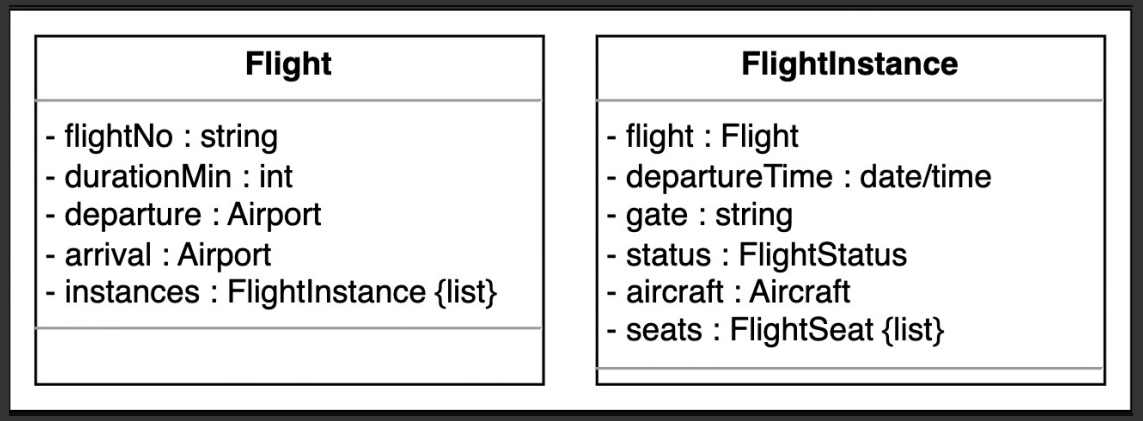
3. Airline, Airport, Aircraft



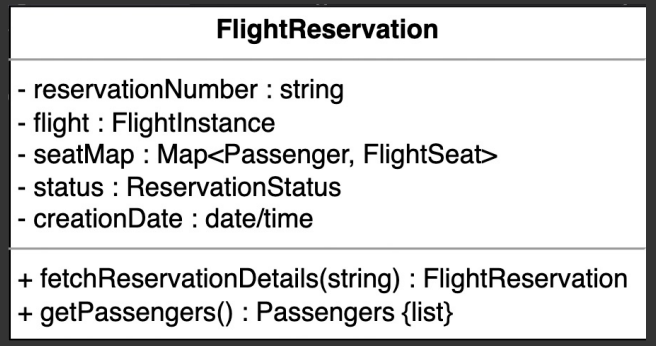
4. Seat & FlightSeat



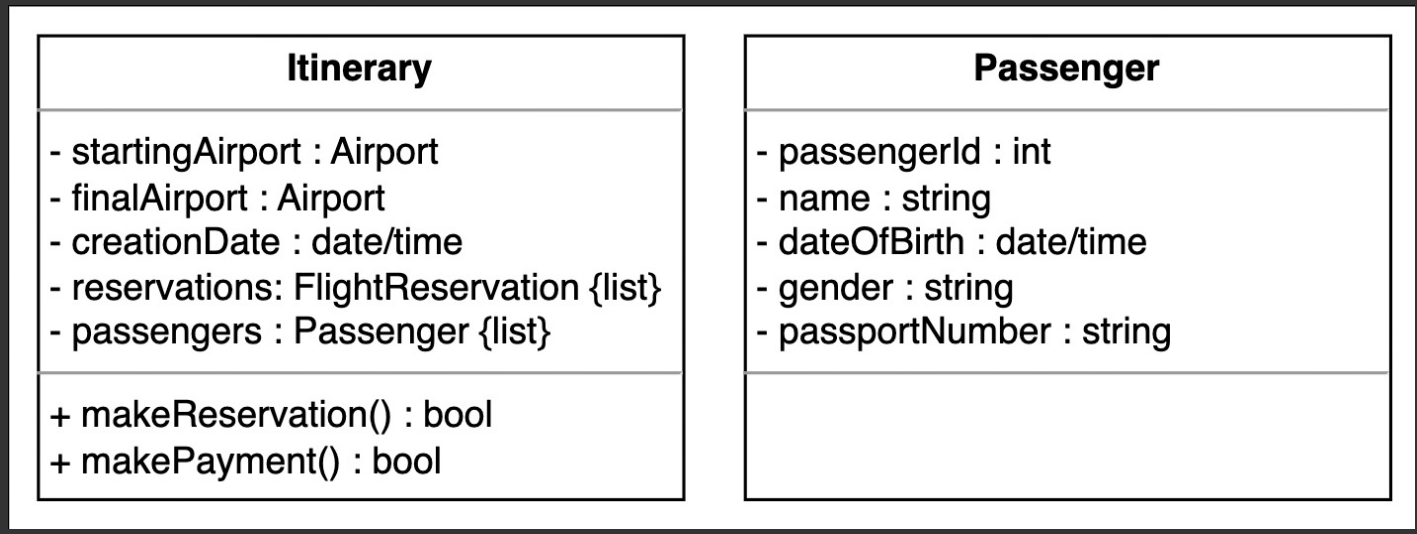
5. Flight & Flight Instance



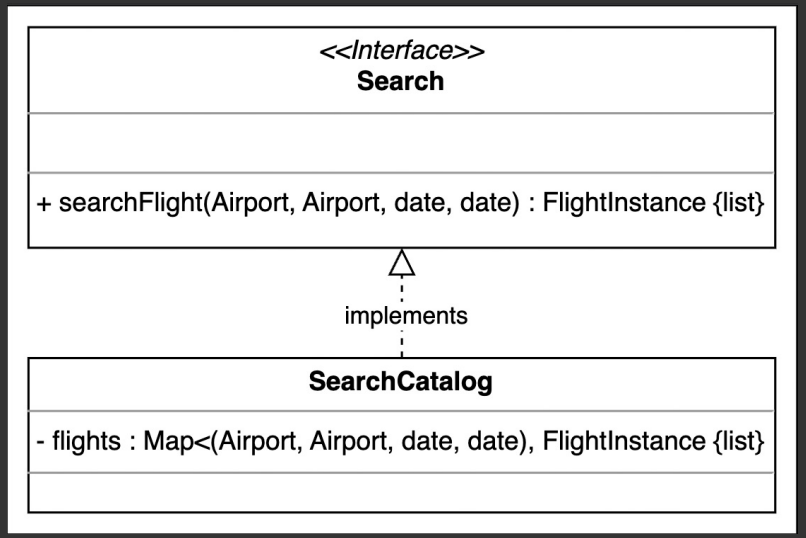
6. Flight Reservation



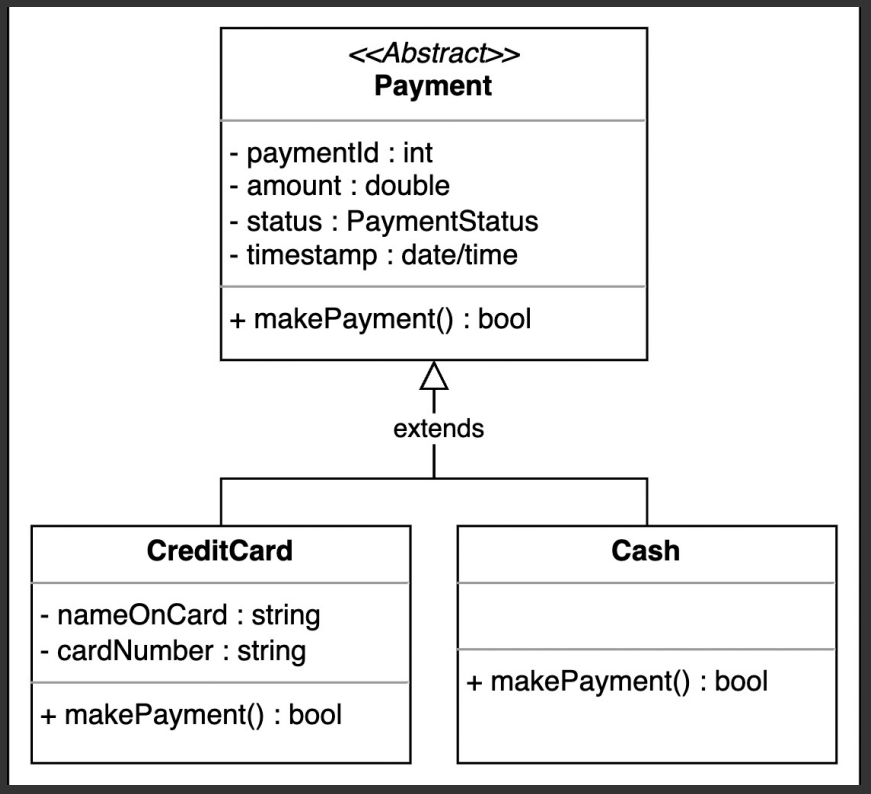
7. Itinerary & passinger



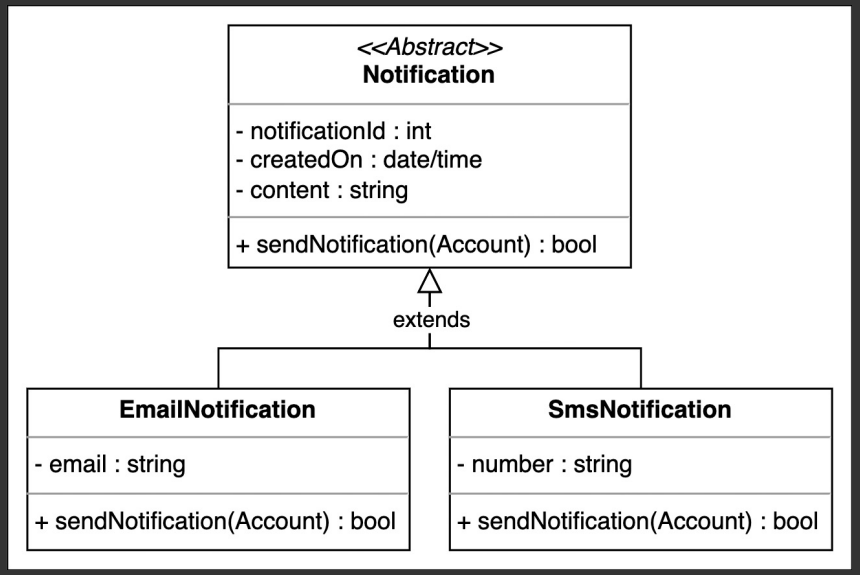
8. Search & Catlog



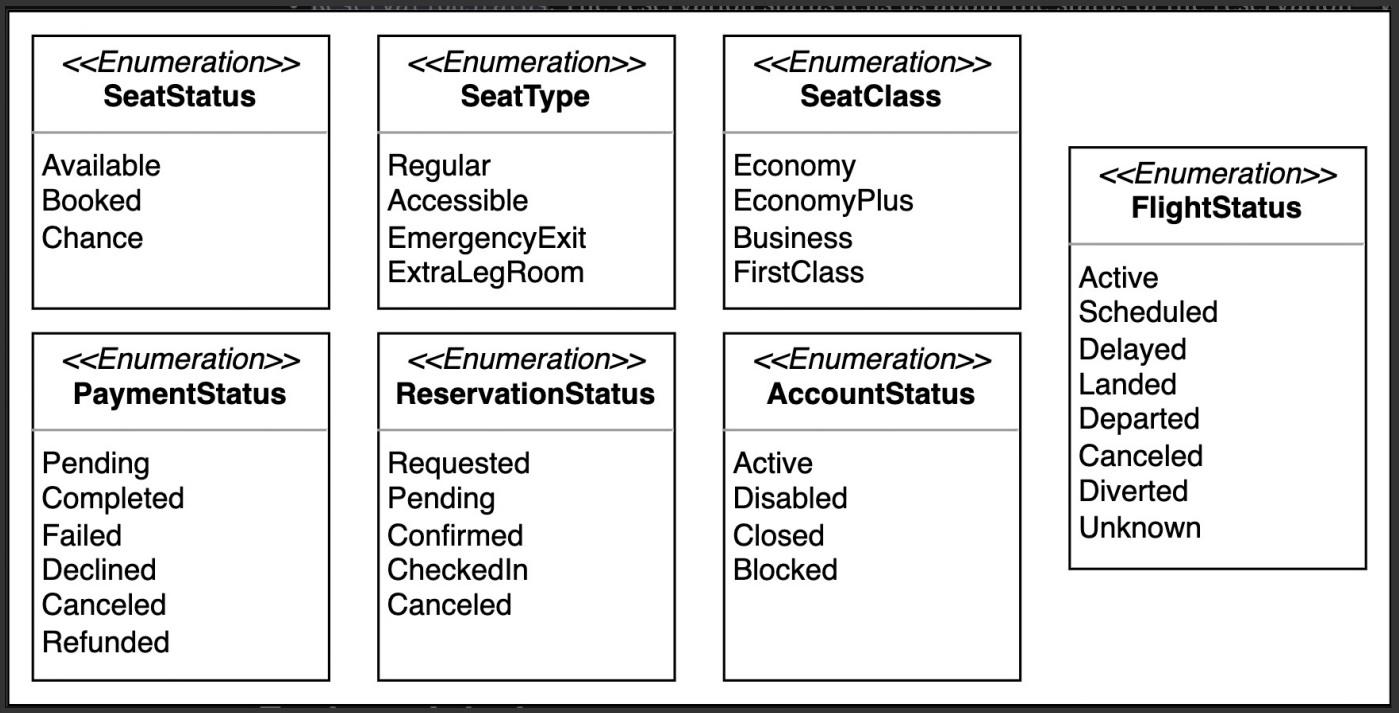
9. Payment



10. Notification

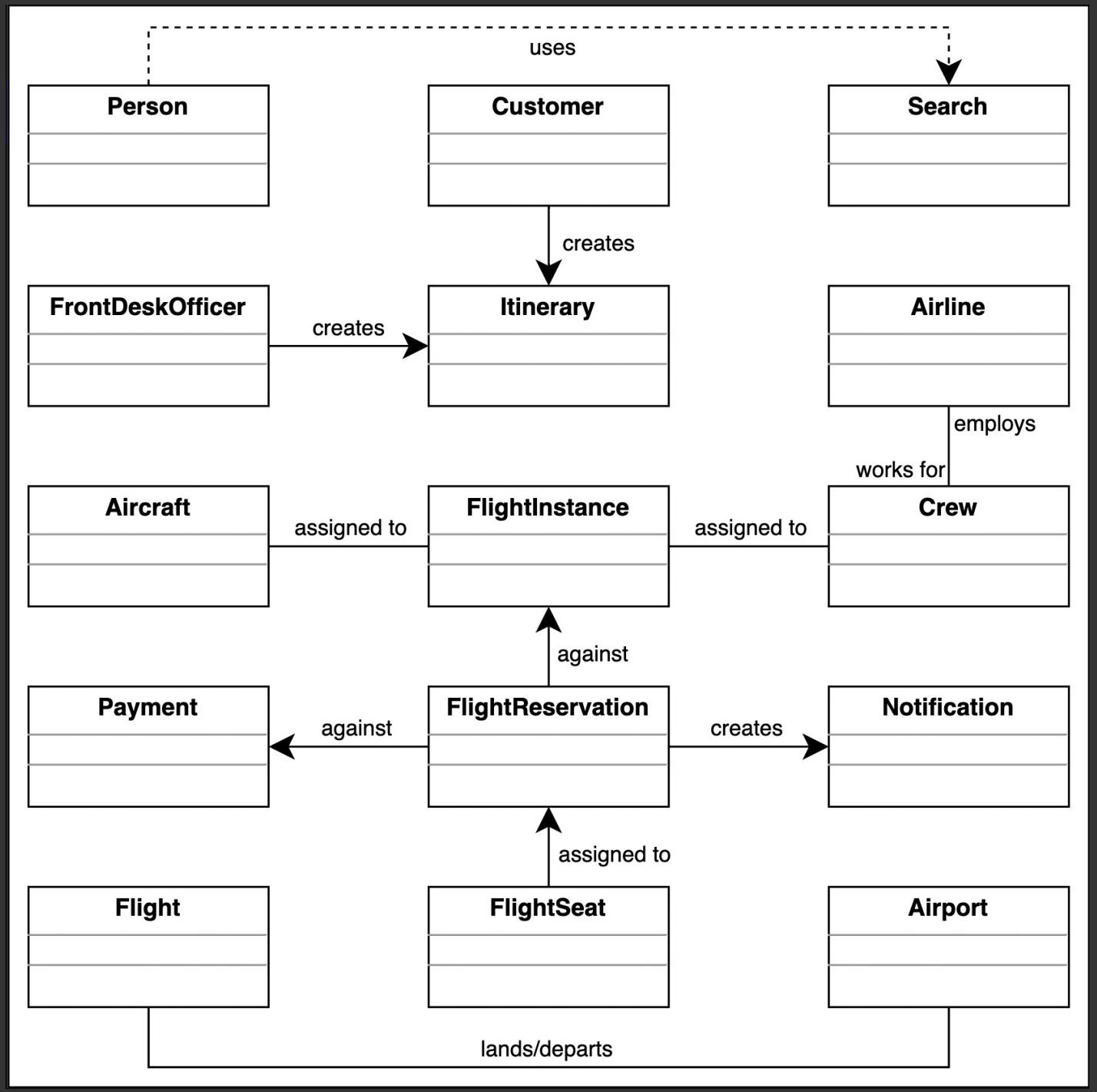


11. Enumeration

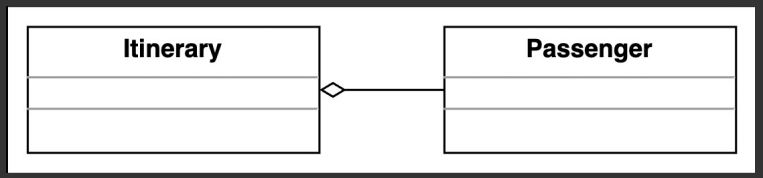


→ Address

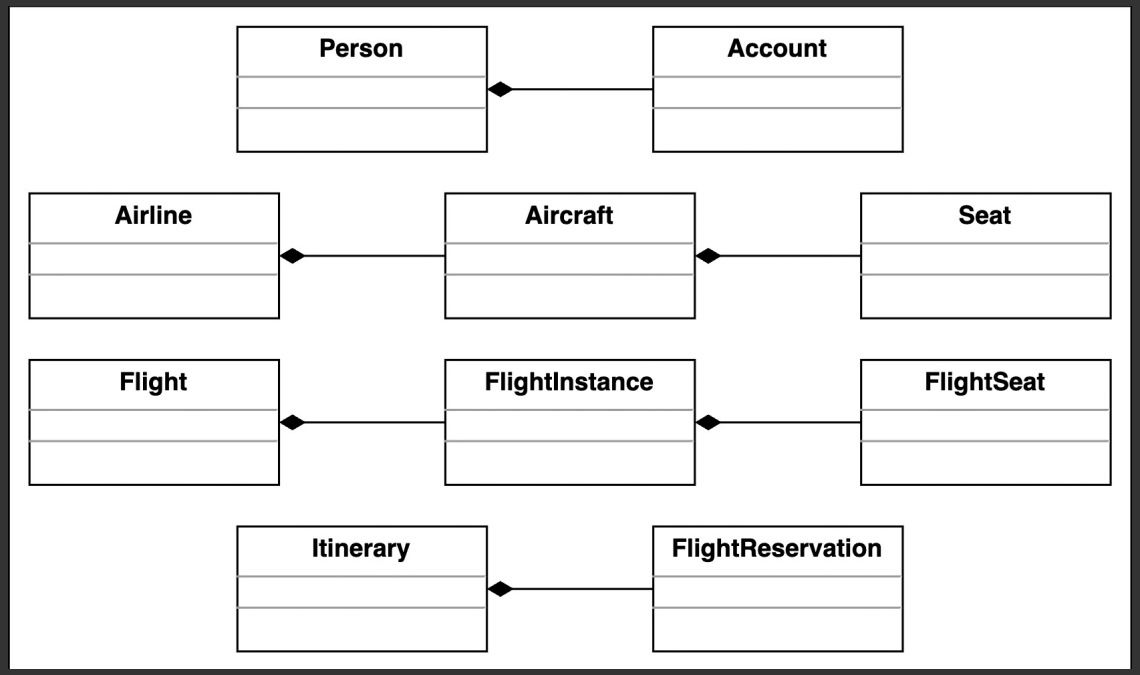
Two-way Associate



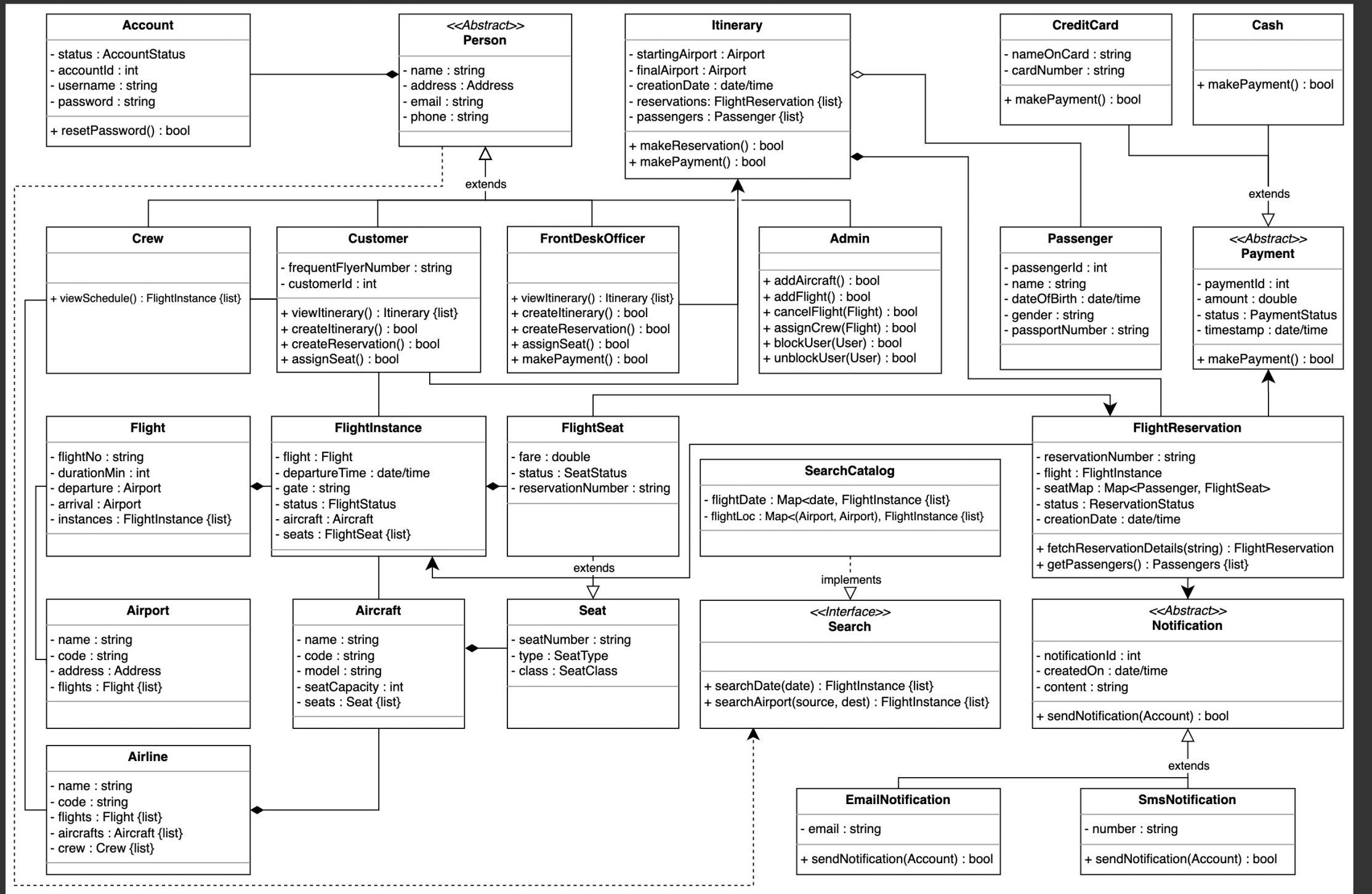
Aggregation



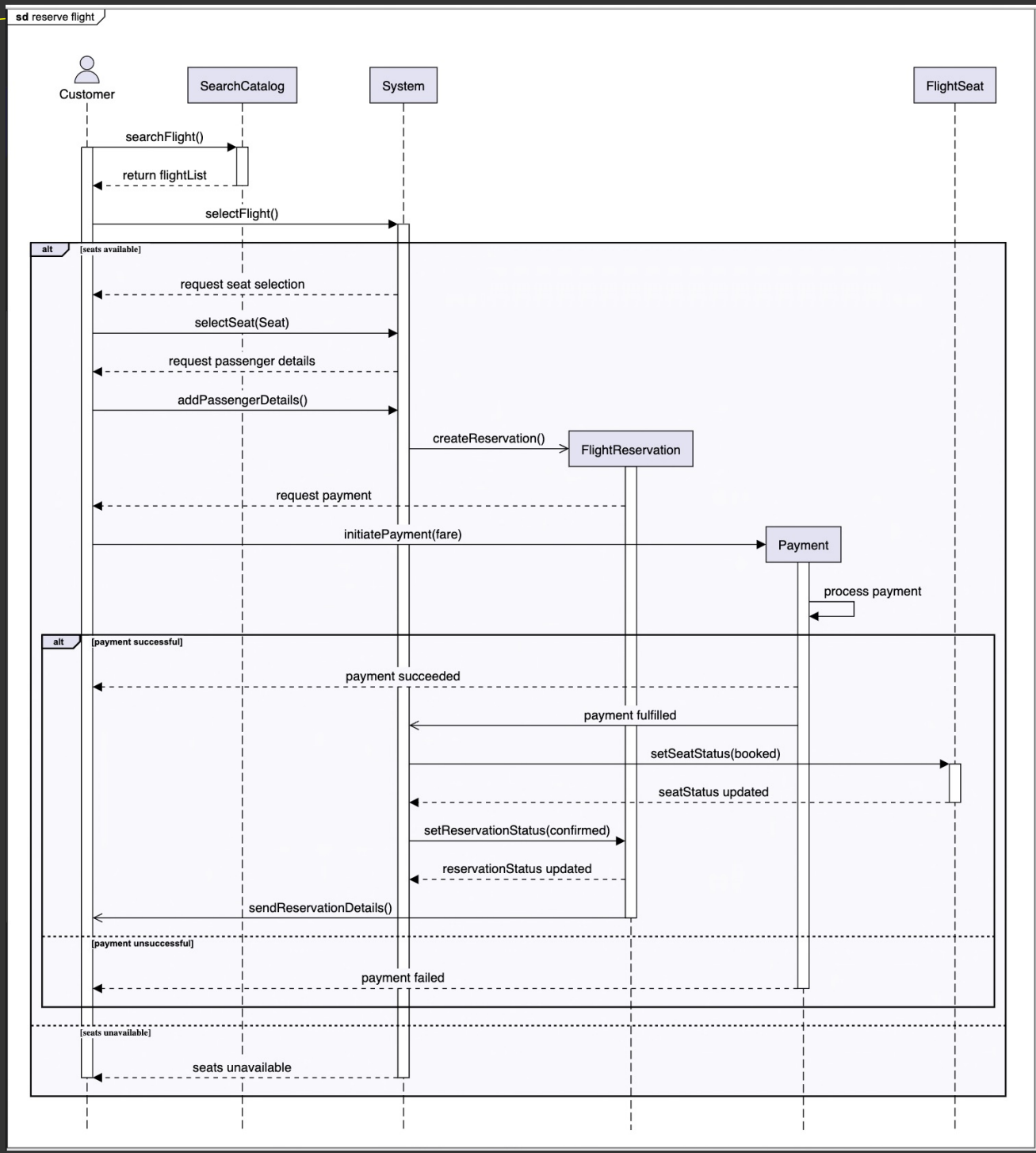
Composition

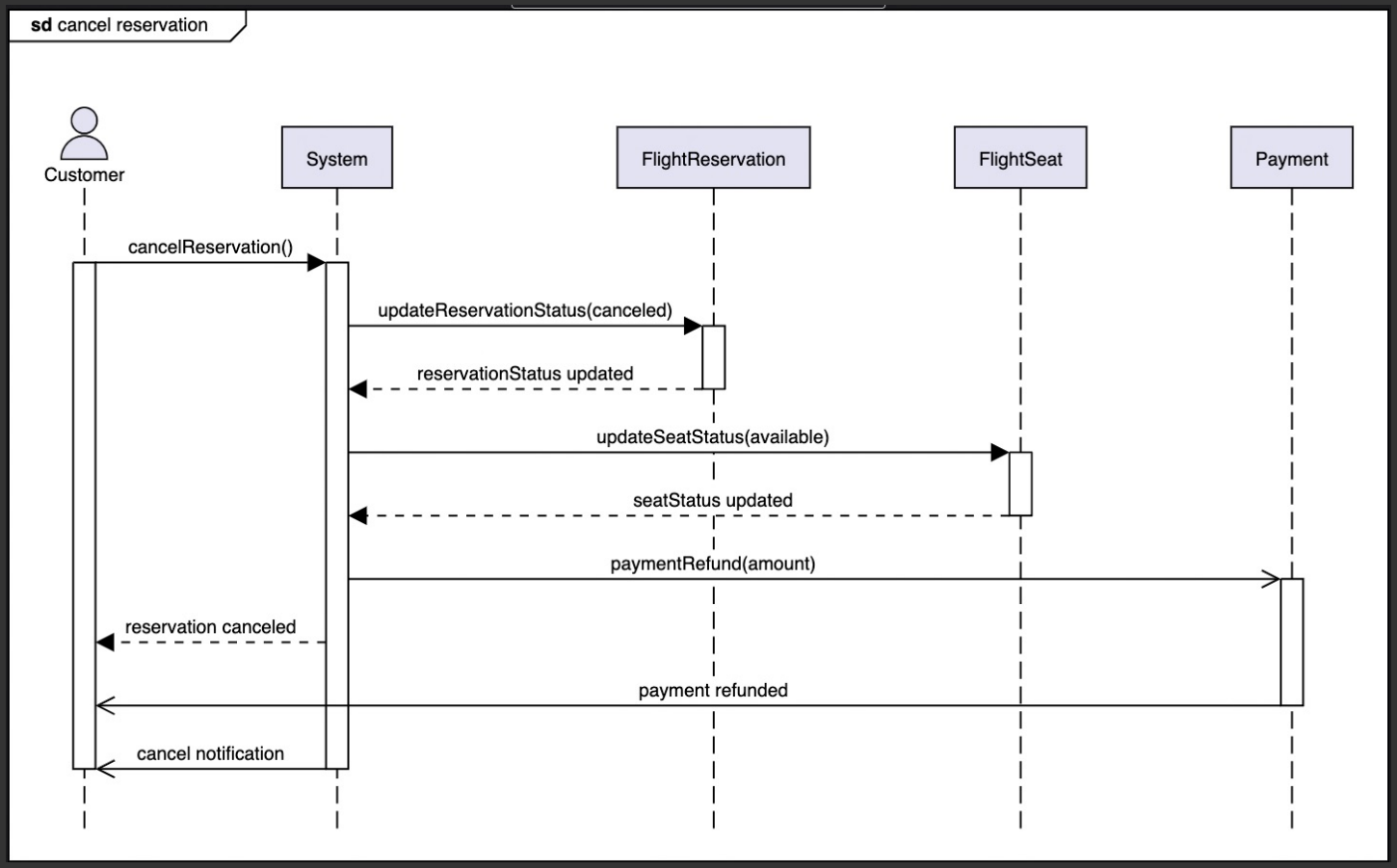


Class Diagram



Sequence Diagram

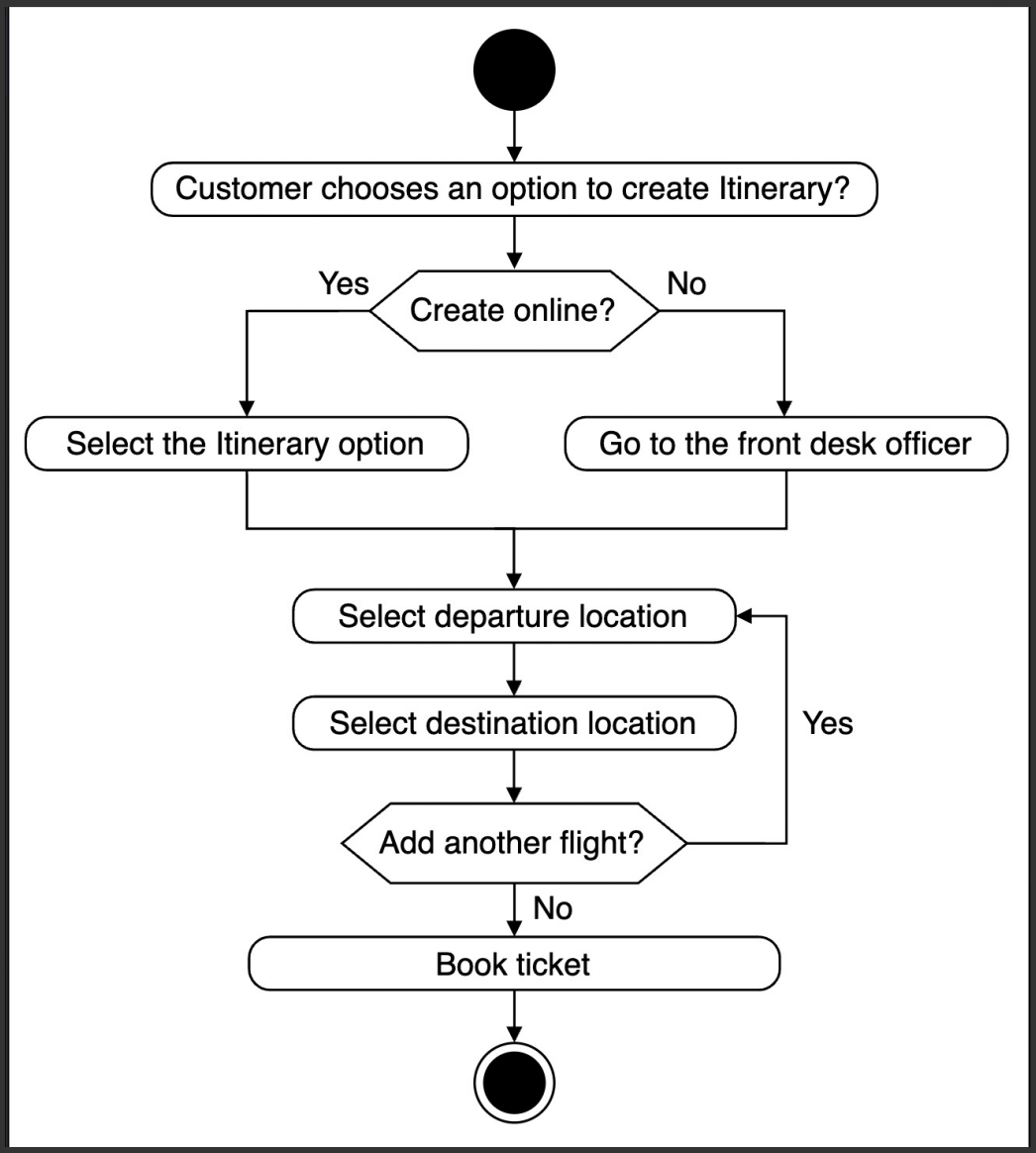




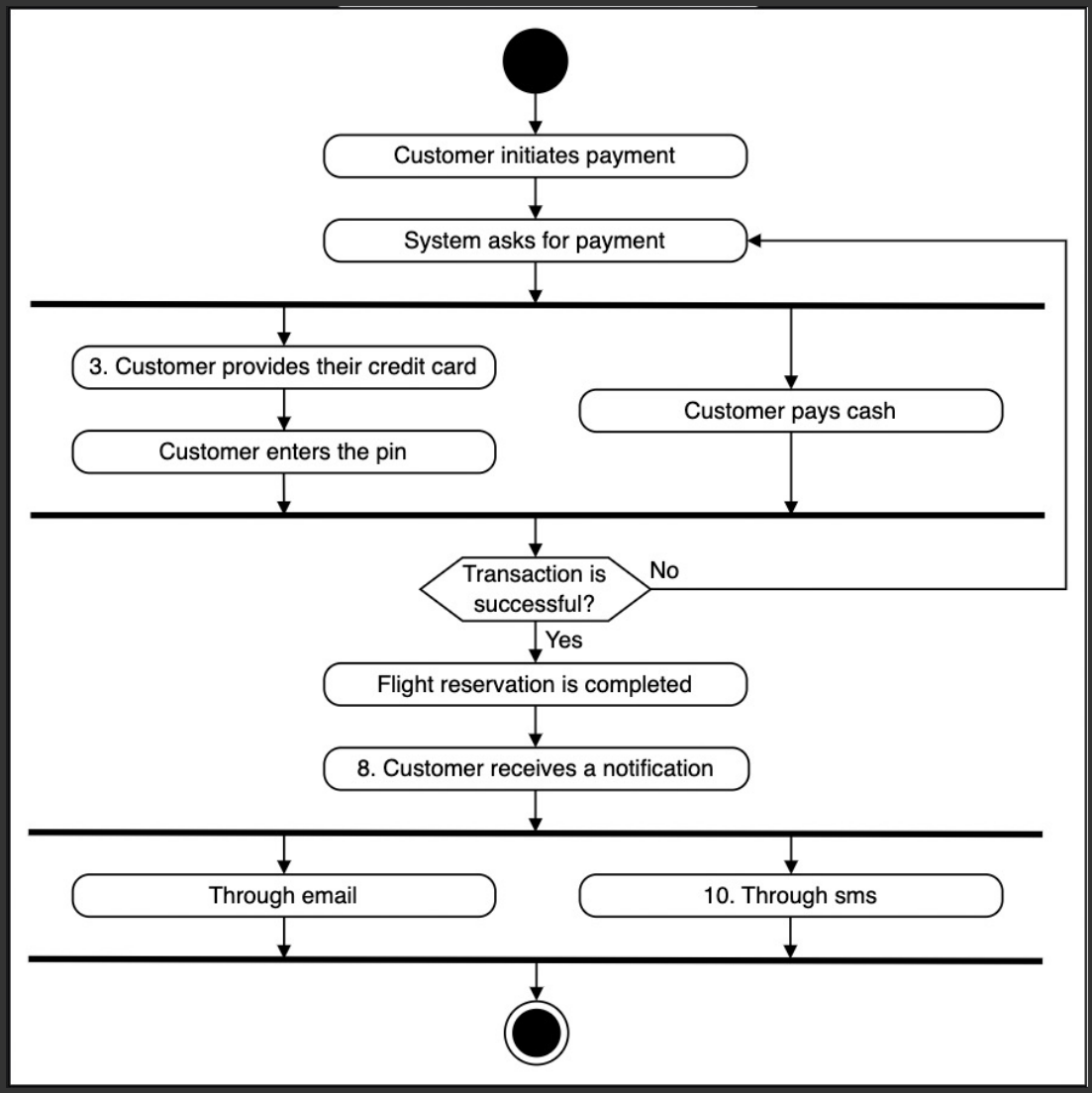
Sequence diagram

→ Cancel Reservation

Activity Diagram



Create a Itinerary



Notification after payment

Code

1. Constants

```
public class Address {
  private int zipCode;
  private String streetAddress;
  private String city;
  private String state;
  private String country;
}

enum AccountStatus {
  ACTIVE,
  DISABLED,
  CLOSED,
  BLOCKED
}

enum SeatStatus {
  AVAILABLE,
  BOOKED,
  CHANCE
}

enum SeatType {
  REGULAR,
  ACCESSIBLE,
  EMERGENCY_EXIT,
  EXTRA_LEG_ROOM
}
```

```
enum SeatClass {
  ECONOMY,
  ECONOMY_PLUS,
  BUSINESS,
  FIRST_CLASS
}

enum FlightStatus {
  ACTIVE,
  SCHEDULED,
  DELAYED,
  LANDED,
  DEPARTED,
  CANCELED,
  DIVERTED,
  UNKNOWN
}

enum ReservationStatus {
  REQUESTED,
  PENDING,
  CONFIRMED,
  CHECKED_IN,
  CANCELED
}

enum PaymentStatus {
  PENDING,
  COMPLETED,
  FAILED,
  DECLINED,
  CANCELED,
  REFUNDED
}
```

2. Account & Passenger

```
public class Account {
  private AccountStatus status;
  private int accountId;
  private String username;
  private String password;

  public boolean resetPassword();
}

public class Passenger {
  private int passengerId;
  private String name;
  private String gender;
  private Date dateOfBirth;
  private String passportNumber;
}
```

3. Person

```
public abstract class Person {
  private String name;
  private Address address;
  private String email;
  private String phone;
  private Account account;
}

public class Admin extends Person {
  public boolean addAircraft(Aircraft aircraft);
  public boolean addFlight(Flight flight);
  public boolean cancelFlight(Flight flight);
  public boolean assignCrew(Flight flight);
  public boolean blockUser(User user);
  public boolean unblockUser(User user);
}

public class Crew extends Person {
  public List<FlightInstance> viewSchedule();
}

public class FrontDeskOfficer extends Person {
  public List<Itinerary> viewItinerary();
  public boolean createItinerary();
  public boolean createReservation();
  public boolean assignSeat();
  public boolean makePayment();
}

public class Customer extends Person {
  private int customerId;

  public List<Itinerary> viewItinerary();
  public boolean createItinerary();
  public boolean createReservation();
  public boolean assignSeat();
  public boolean makePayment();
}
```

4. Seat & Flight Seat

```
public class Seat {
  private String seatNumber;
  private SeatType type;
  private SeatClass _class;
}

public class FlightSeat extends Seat {
  private double fare;
  private SeatStatus status;
  private String reservationNumber;
}
```

5. Flight & Flight Instance

```
public class Flight {
  private String flightNo;
  private int durationMin;
  private Airport departure;
  private Airport arrival;
  private List<FlightInstance> instances;
}

public class FlightInstance {
  private Flight flight;
  private Date departureTime;
  private String gate;
  private FlightStatus status;
  private Aircraft aircraft;
  private List<FlightSeat> seats;
}
```

7. Itinerary & Flight Reservation

```
public class Itinerary {
  private Airport startingAirport;
  private Airport finalAirport;
  private Date creationDate;
  private List<FlightReservation> reservations;
  private List<Passenger> passengers;

  public boolean makeReservation();
  public boolean makePayment();
}

public class FlightReservation {
  private String reservationNumber;
  private FlightInstance flight;
  private HashMap<Passenger, FlightSeat> seatMap;
  private ReservationStatus status;
  private Date creationDate;

  public static FlightReservation fetchReservationDetails(String reservationNumber);
  public List<Passenger> getPassengers();
}
```

8. Payment

```
public abstract class Payment {
  private int paymentId;
  private double amount;
  private PaymentStatus status;
  private Date timestamp;

  public abstract boolean makePayment();
}

public class Cash extends Payment {
    public boolean makePayment() {
        // functionality
    }
}

public class CreditCard extends Payment {
    private String nameOnCard;
    private String cardNumber;

    public boolean makePayment() {
        // functionality
    }
}
```

9. Notification

```
public abstract class Notification {
    private int notificationId;
    private Date createdOn;
    private String content;

    public abstract void sendNotification(Account
account);
}

class SmsNotification extends Notification {
    public void sendNotification(Account account) {
        // functionality
    }
}

class EmailNotification extends Notification {
    public void sendNotification(Account account) {
        // functionality
    }
}
```

10. Search & Catalog

```
public interface Search {
  // Interface method (does not have a body)
  public List<FlightInstance> searchFlight(Airport source, Airport dest, Date arrival, Date departure);
}

public class SearchCatalog implements Search {
  private HashMap<Quartet<Airport, Airport, Date, Date>, List<FlightInstance>> flights;

  public List<FlightInstance> searchFlight(Airport source, Airport dest, Date arrival, Date departure) {
    // functionality
  }
}
```

11. Airport, Aircraft & Airline

```
public class Airport {
  private String name;
  private String code;
  private Address address;
  private List<Flight> flights;
}

public class Aircraft {
  private String name;
  private String code;
  private String model;
  private int seatCapacity;
  private List<Seat> seats;
}

public class Airline {
  private String name;
  private String code;
  private List<Flight> flights;
  private List<Aircraft> aircrafts;
  private List<Crew> crew;

  // The Airline is a singleton class that ensures it will have only one active instance at a time
  private static Airline airline = null;

  // Created a static method to access the singleton instance of Airline class
  public static Airline getInstance() {
    if (airline == null) {
      airline = new Airline();
    }
    return airline;
  }
}
```