2. Airline Management System

An Airline Management System is managerial software which targets to control all operations of an airline. Airlines provide transport services for their passengers. They carry or hire aircraft for this purpose. All operations of an airline company are controlled by their airline management system.

This system involves the scheduling of flights, air ticket reservations, flight cancellations, customer support, and staff management. Daily flights updates can also be retrieved by using the system.

System Requirements

We will focus on the following set of requirements while designing the Airline Management System:

1. Customers should be able to search for flights for a given date and source/destination airport.
2. Customers should be able to reserve a ticket for any scheduled flight. Customers can also build a multi-flight itinerary.
3. Users of the system can check flight schedules, their departure time, available seats, arrival time, and other flight details.
4. Customers can make reservations for multiple passengers under one itinerary.
5. Only the admin of the system can add new aircrafts, flights, and flight schedules. Admin can cancel any pre-scheduled flight (all stakeholders will be notified).
6. Customers can cancel their reservation and itinerary.
7. The system should be able to handle the assignment of pilots and crew members to flights.
8. The system should be able to handle payments for reservations.
9. The system should be able to send notifications to customers whenever a reservation is made/modified or there is an update for their flights.

Use Case Diagram

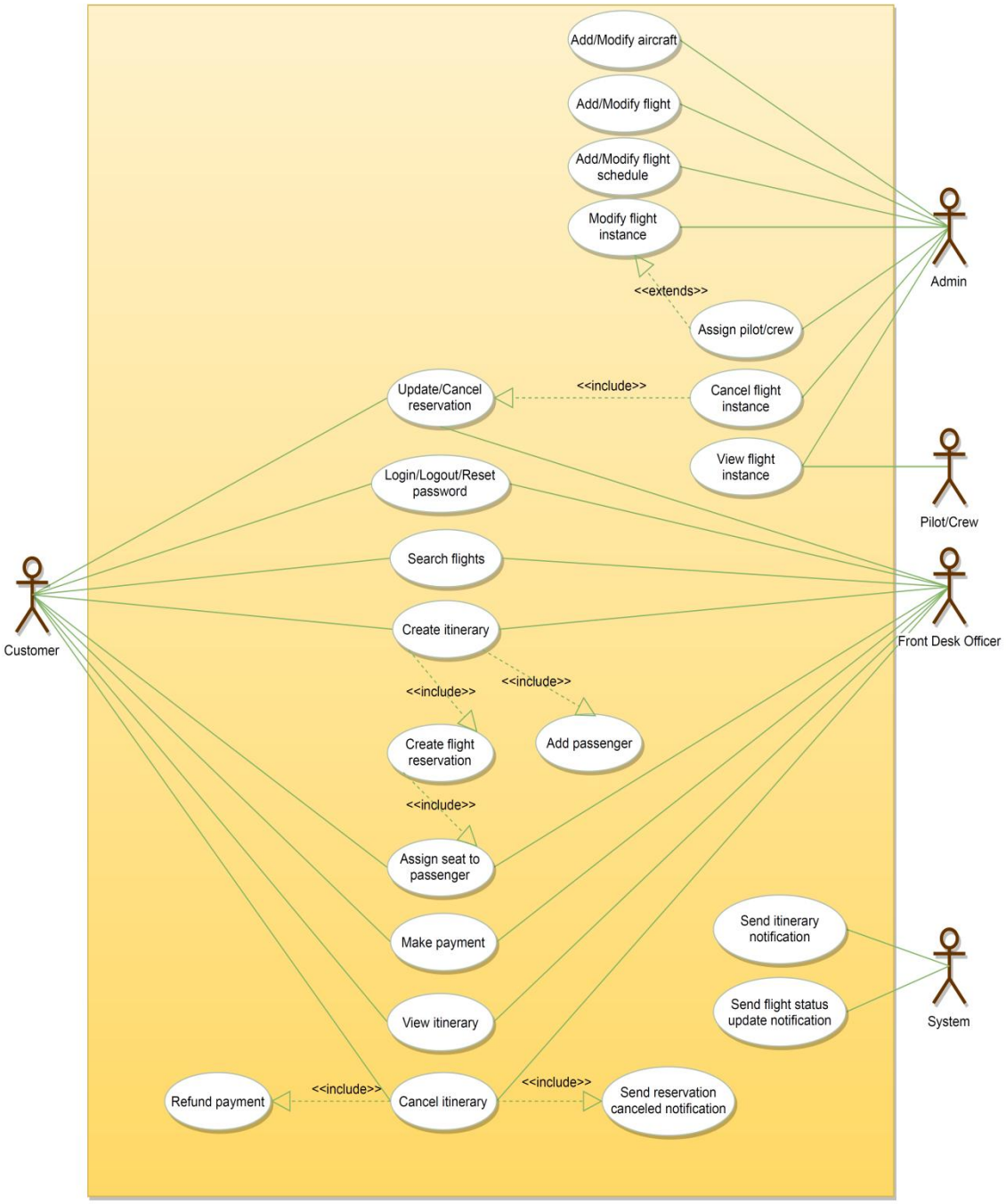
We have five main Actors in our system:

- **Admin**: Responsible for adding new flights and their schedules, canceling any flight, maintaining staff-related work, etc.
- **Front desk officer**: Will be able to reserve/cancel tickets.
- **Customer**: Can view flight schedule, reserve and cancel tickets.
- **Pilot/Crew**: Can view their assigned flights and their schedules.
- **System**: Mainly responsible for sending notifications regarding itinerary changes, flight status updates, etc.

Here are the top use cases of the Airline Management System:

- **Search Flights**: To search the flight schedule to find flights for a suitable date and time.
- **Create/Modify/View reservation**: To reserve a ticket, cancel it, or view details about the flight or ticket.
- **Assign seats to passengers**: To assign seats to passengers for a flight instance with their reservation.
- **Make payment for a reservation**: To pay for the reservation.
- **Update flight schedule**: To make changes in the flight schedule, and to add or remove any flight.
- **Assign pilots and crew**: To assign pilots and crews to flights.

Here is the use case diagram of our Airline Management System:

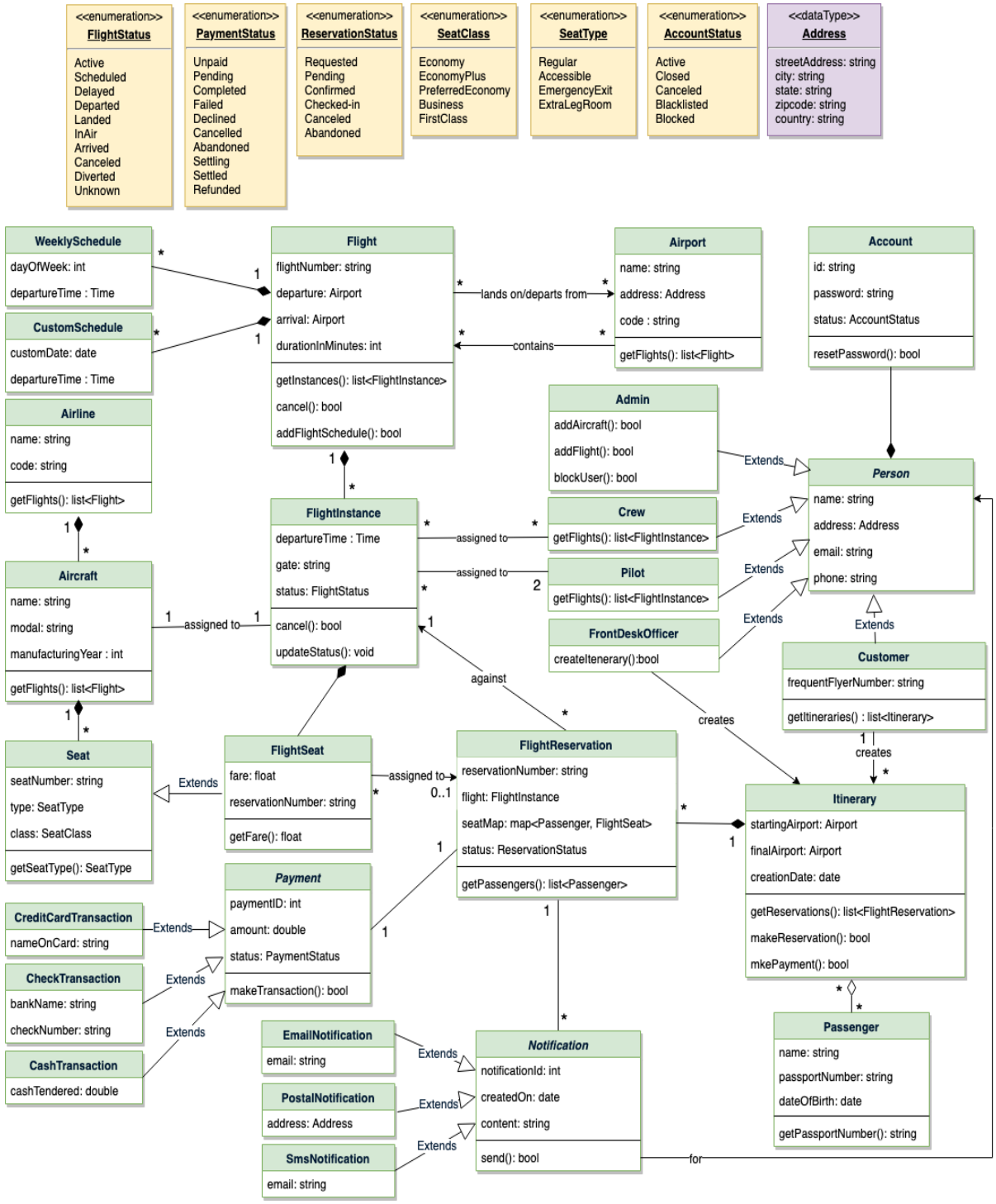


Use Case Diagram for Airline Management System

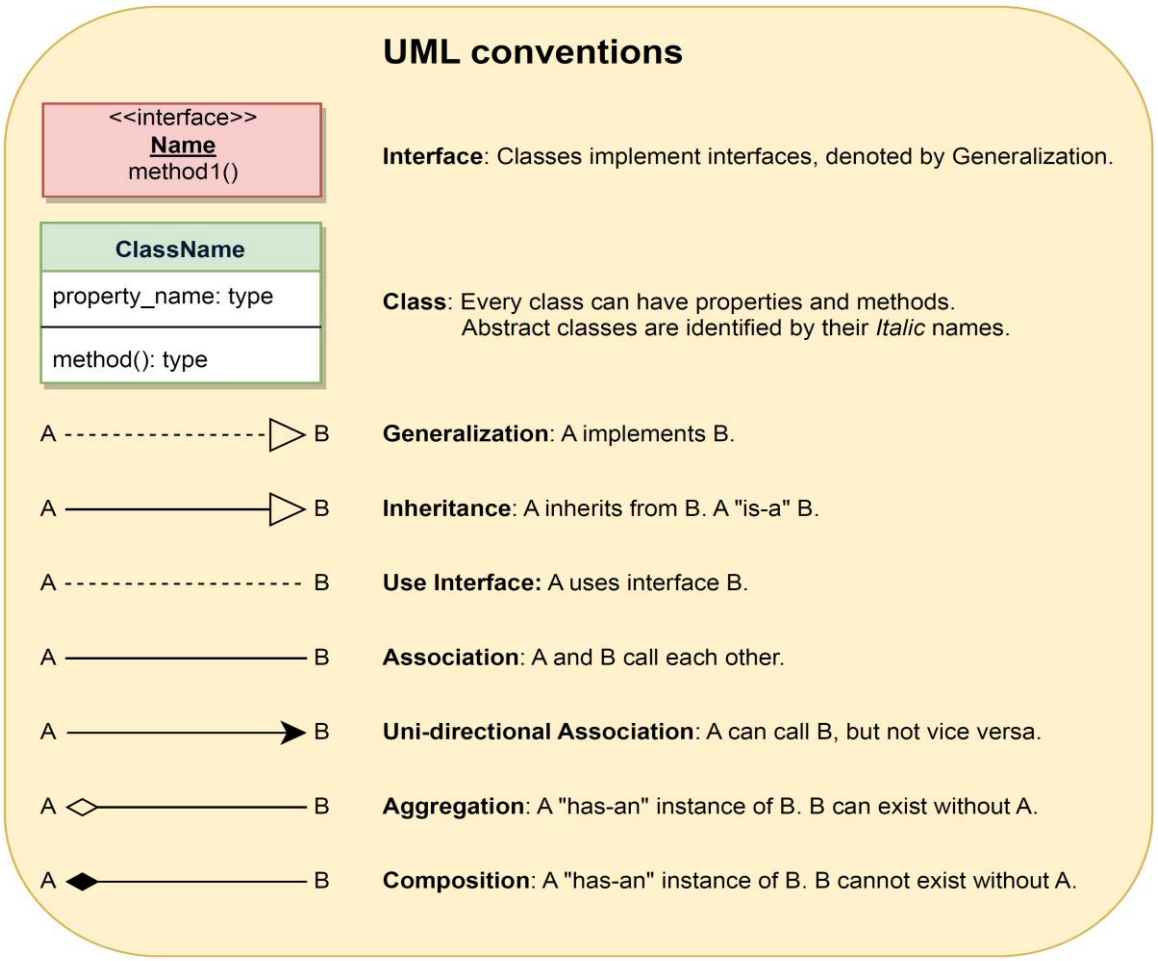
Class Diagram

Here are the main classes of our Airline Management System:

- **Airline**: The main part of the organization for which this software has been designed. It has attributes like 'name' and an airline code to distinguish the airline from other airlines.
- **Airport**: Each airline operates out of different airports. Each airport has a name, address, and a unique code.
- **Aircraft**: Airlines own or hire aircraft to carry out their flights. Each aircraft has attributes like name, model, manufacturing year, etc.
- **Flight**: The main entity of the system. Each flight will have a flight number, departure and arrival airport, assigned aircraft, etc.
- **FlightInstance**: Each flight can have multiple occurrences; each occurrence will be considered a flight instance in our system. For example, if a British Airways flight from London to Tokyo (flight number: BA212) occurs twice a week, each of these occurrences will be considered a separate flight instance in our system.
- **WeeklySchedule and CustomSchedule**: Flights can have multiple schedules and each schedule will create a flight instance.
- **FlightReservation**: A reservation is made against a flight instance and has attributes like a unique reservation number, list of passengers and their assigned seats, reservation status, etc.
- **Itinerary**: An itinerary can have multiple flights.
- **FlightSeat**: This class will represent all seats of an aircraft assigned to a specific flight instance. All reservations of this flight instance will assign seats to passengers through this class.
- **Payment**: Will be responsible for collecting payments from customers.
- **Notification**: This class will be responsible for sending notifications for flight reservations, flight status update, etc.



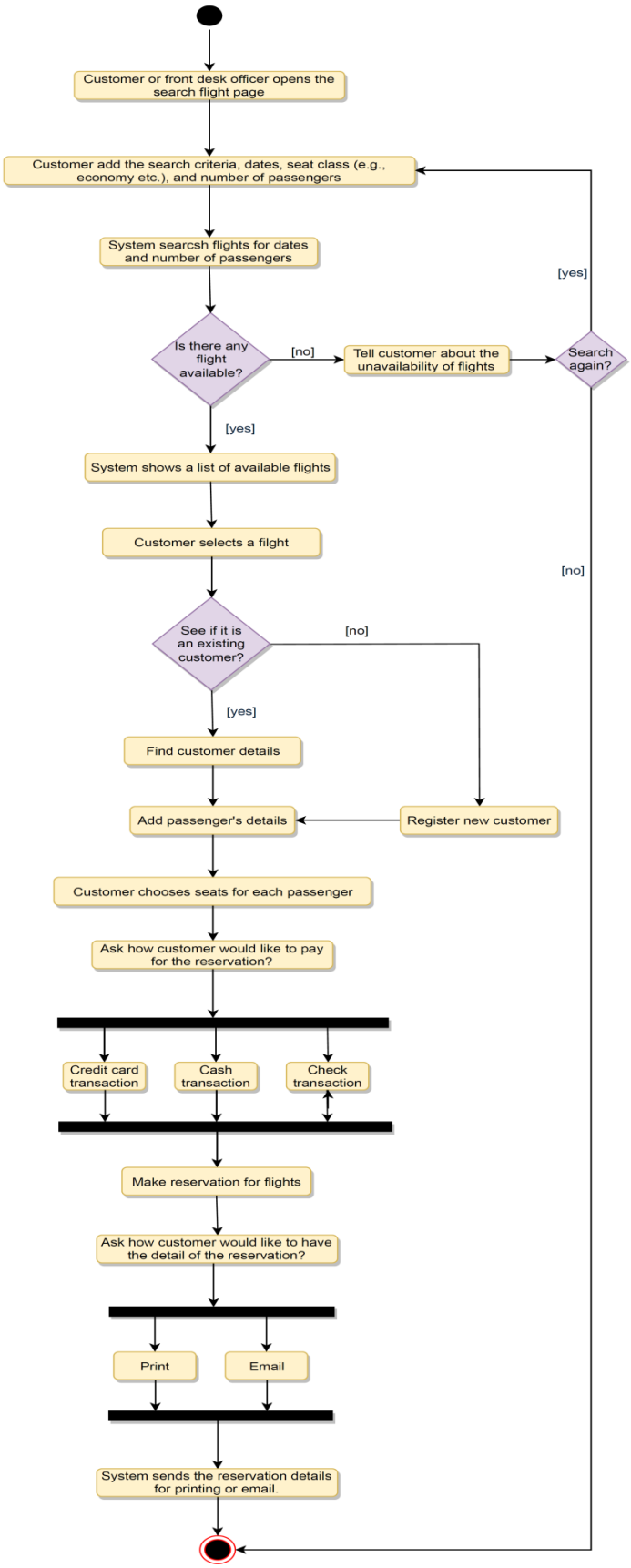
Airline Management System Class Diagram



Class Diagram for Airline Management System

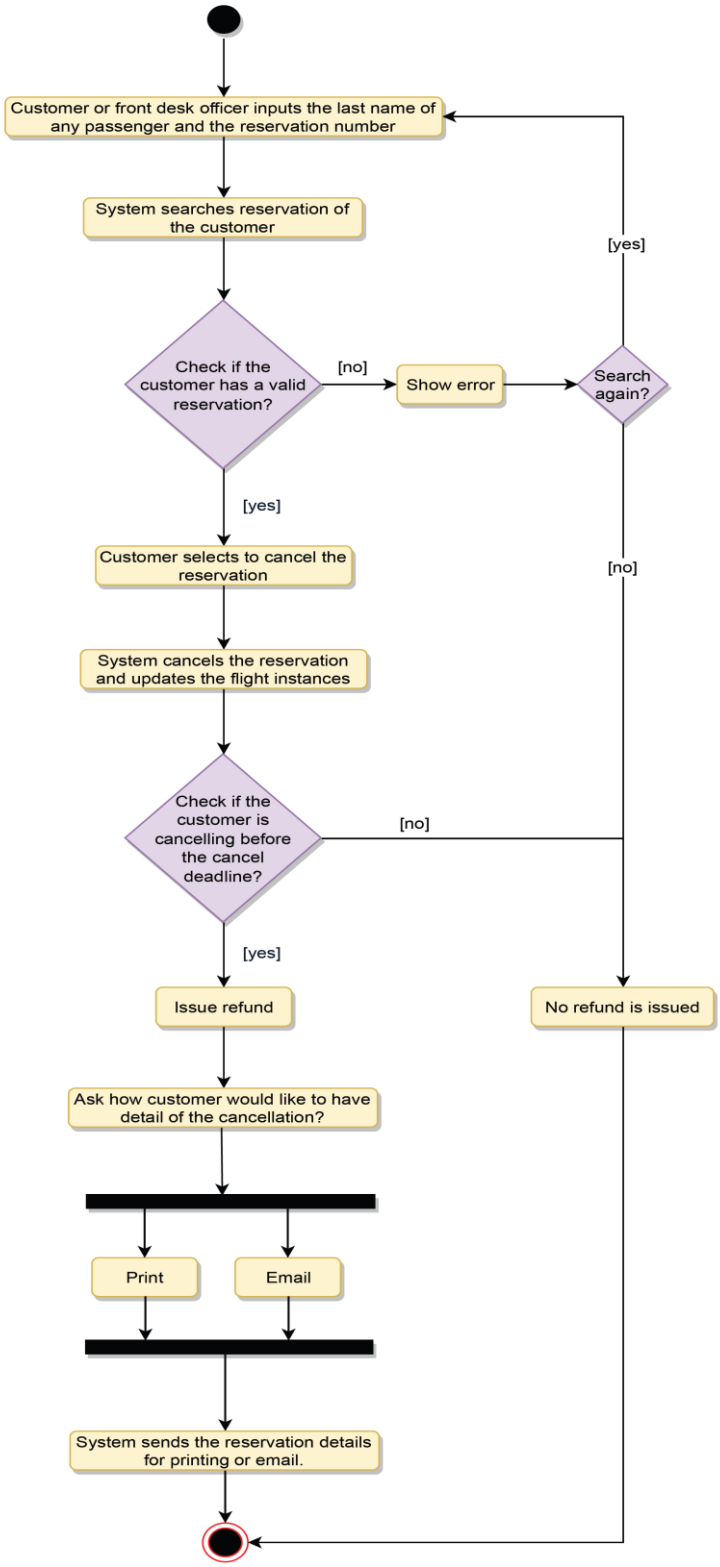
Activity Diagrams

Reserve a ticket: Any customer can perform this activity. Here are the steps to reserve a ticket:



Airline Management System Reserve Ticket Activity Diagram

Cancel a reservation: Any customer can perform this activity. Here are the set of steps to cancel a reservation:



Airline Management System Cancel Reservation Activity Diagram

Code

Here is the code for major classes.

Enums and Constants:

Here are the required enums, data types, and constants:

```
import java.util.List;

// Enum representing different flight statuses
enum FlightStatus {
   ACTIVE(1), SCHEDULED(2), DELAYED(3), DEPARTED(4), LANDED(5), IN_AIR(6), ARRIVED(7), CANCELLED(8), DIVERTED(9), UNKNOWN(10);
   private int value;
   FlightStatus(int value) {
      this.value = value;
   }
   public int getValue() {
      return value;
   }
}

// Enum representing different payment statuses
enum PaymentStatus {
   UNPAID(1), PENDING(2), COMPLETED(3), FILLED(4), DECLINED(5), CANCELLED(6), ABANDONED(7), SETTLING(8), SETTLED(9), REFUNDED(10);
   private int value;
   PaymentStatus(int value) {
      this.value = value;
   }
   public int getValue() {
      return value;
   }
}

// Enum representing different reservation statuses
enum ReservationStatus {
   REQUESTED(1), PENDING(2), CONFIRMED(3), CHECKED_IN(4), CANCELLED(5), ABANDONED(6);
   private int value;
   ReservationStatus(int value) {
      this.value = value;
   }
   public int getValue() {
      return value;
   }
}

// Enum representing different seat classes
enum SeatClass {
```

```
   ECONOMY(1), ECONOMY_PLUS(2), PREFERRED_ECONOMY(3), BUSINESS(4), FIRST_CLASS(5);
   private int value;
   SeatClass(int value) {
      this.value = value;
   }
   public int getValue() {
      return value;
   }
}

// Enum representing different seat types
enum SeatType {
   REGULAR(1), ACCESSIBLE(2), EMERGENCY_EXIT(3), EXTRA_LEG_ROOM(4);
   private int value;
   SeatType(int value) {
      this.value = value;
   }
   public int getValue() {
      return value;
   }
}

// Enum representing different account statuses
enum AccountStatus {
   ACTIVE(1), CLOSED(2), CANCELED(3), BLACKLISTED(4), BLOCKED(5);
   private int value;
   AccountStatus(int value) {
      this.value = value;
   }
   public int getValue() {
      return value;
   }
}

// Class representing an address
class Address {
   private String street;
   private String city;
   private String state;
   private String zipCode;
   private String country;
   public Address(String street, String city, String state, String zipCode, String country) {
      this.street = street;
      this.city = city;
      this.state = state;
      this.zipCode = zipCode;
      this.country = country;
   }
}

// Class representing an account
```

```
class Account {
    private String id;
    private String password;
    private AccountStatus status;

    public Account(String id, String password, AccountStatus status) {
        this.id = id;
        this.password = password;
        this.status = status;
    }

    public void resetPassword() {
        // Implement password reset functionality
    }
}

// Abstract class representing a person
abstract class Person {
    private String name;
    private Address address;
    private String email;
    private String phone;
    private Account account;

    public Person(String name, Address address, String email, String phone, Account account) {
        this.name = name;
        this.address = address;
        this.email = email;
        this.phone = phone;
        this.account = account;
    }
}

// Class representing a customer
class Customer extends Person {
    private String frequentFlyerNumber;

    public Customer(String frequentFlyerNumber) {
        super("", null, "", "", null);
        this.frequentFlyerNumber = frequentFlyerNumber;
    }
}

// Class representing a passenger
class Passenger {
    private String name;
    private String passportNumber;
    private String dateOfBirth;

    public Passenger(String name, String passportNumber, String dateOfBirth) {
        this.name = name;
```

```
        this.passportNumber = passportNumber;
        this.dateOfBirth = dateOfBirth;
    }
}

// Class representing an airport
class Airport {
    private String name;
    private Address address;
    private String code;
}

// Class representing an aircraft
class Aircraft {
    private String name;
    private String model;
    private int manufacturingYear;
    private List<Seat> seats;
}

// Class representing a seat
class Seat {
    private String seatNumber;
    private SeatType type;
    private SeatClass seatClass;
}

// Class representing a flight seat with fare\class FlightSeat extends Seat {
    private double fare;
}

// Class representing a weekly flight schedule
class WeeklySchedule {
    private String dayOfWeek;
    private String departureTime;
}

// Class representing a custom flight schedule
class CustomSchedule {
    private String customDate;
    private String departureTime;
}

// Class representing a flight
class Flight {
    private String flightNumber;
    private String departure;
    private String arrival;
    private int durationInMinutes;
}
```

```
// Class representing a flight instance
class FlightInstance {
   private String departureTime;
   private String gate;
   private FlightStatus status;
   private Aircraft aircraft;
}

// Class representing a flight reservation
class FlightReservation {
   private String reservationNumber;
   private Flight flight;
   private List<Seat> seatMap;
   private String creationDate;
   private ReservationStatus status;
}

// Class representing an itinerary
class Itinerary {
   private String customerId;
   private Airport startingAirport;
   private Airport finalAirport;
   private String creationDate;
}
```