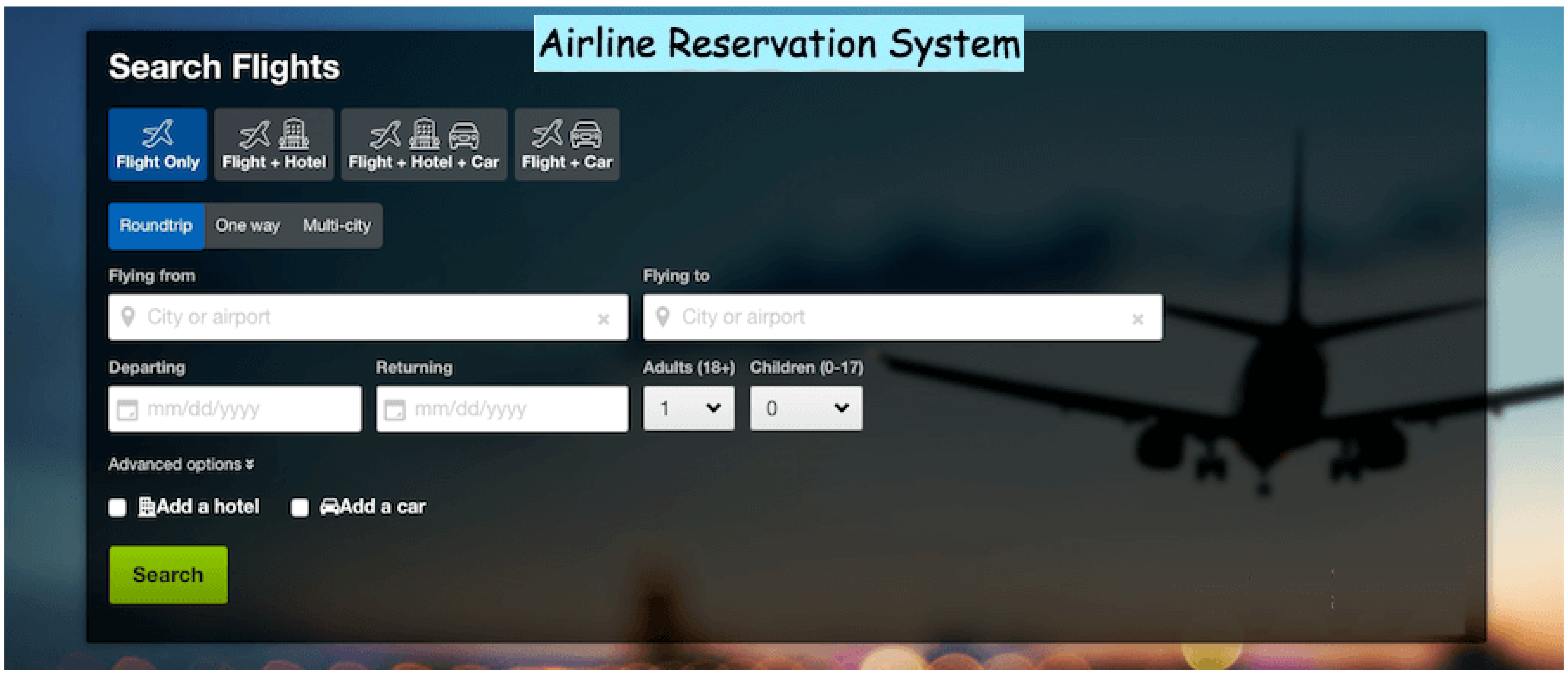
educative

Design an Airline Management System

We'll cover the following

An Airline Management System is a 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.

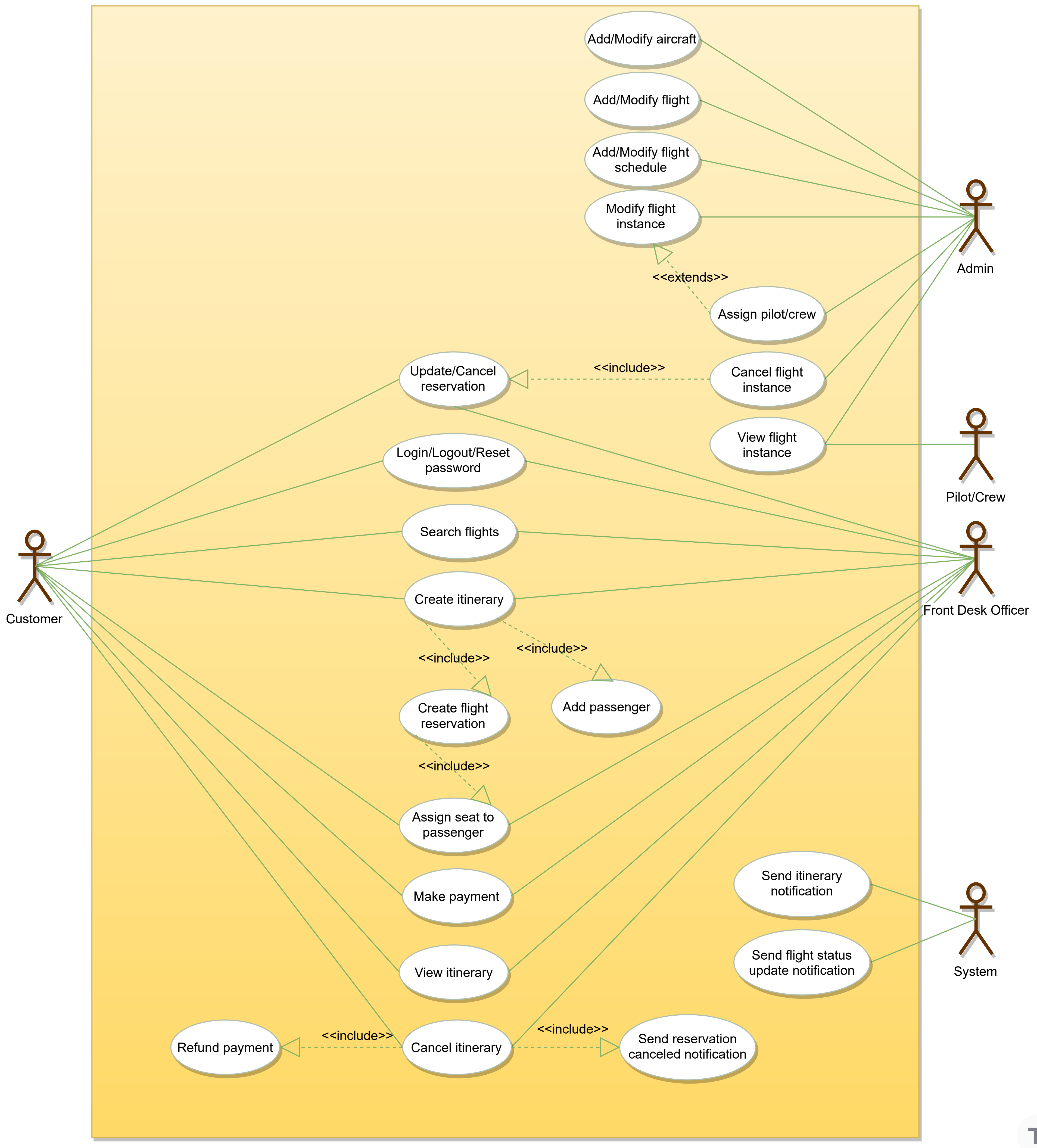
Use case diagram

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.

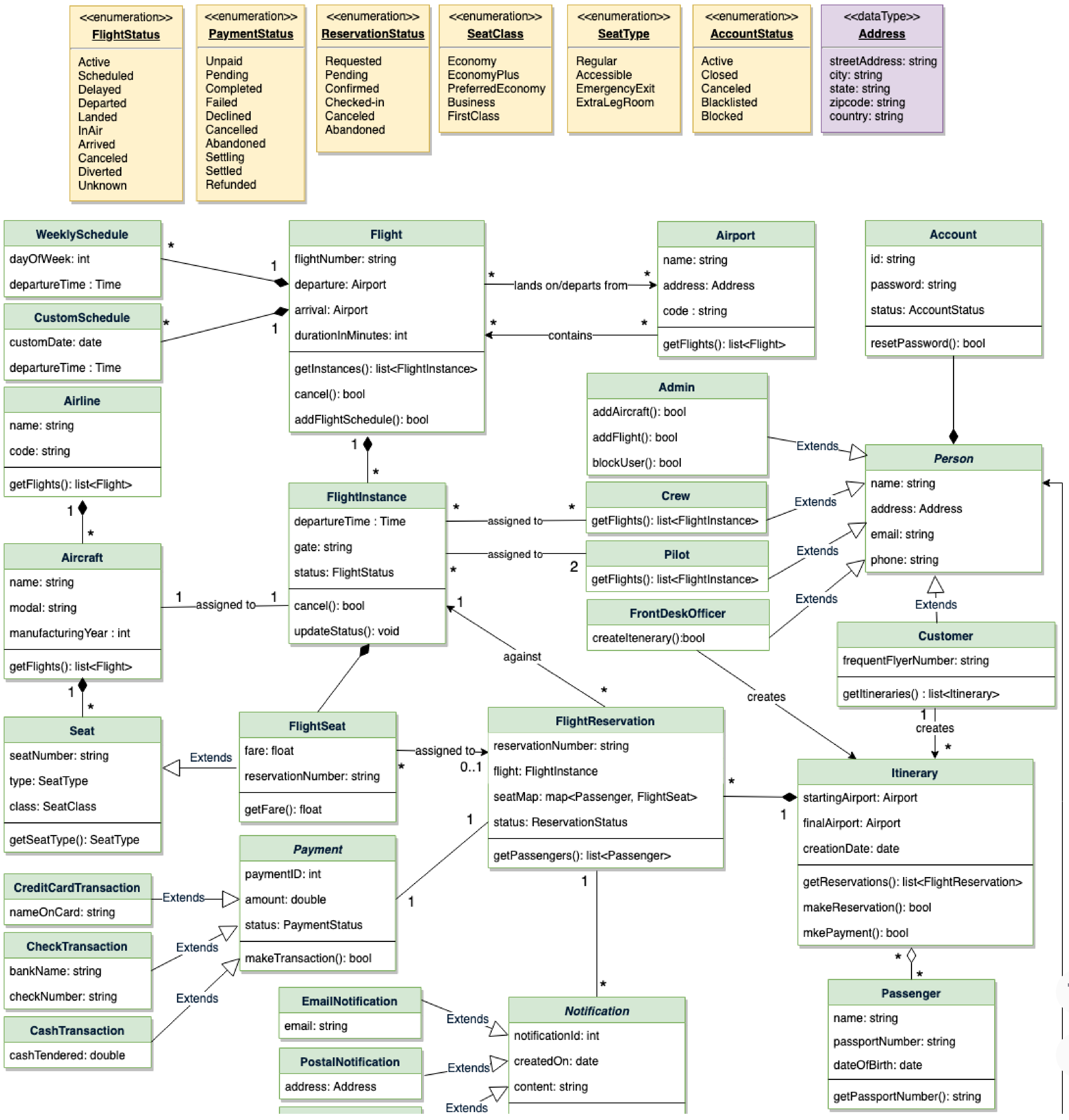


Use case diagram

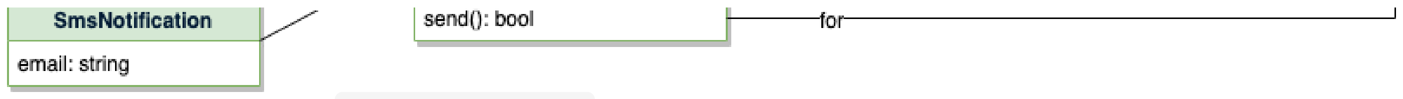
Class diagram

Here are the main classes of our Airline Management System:

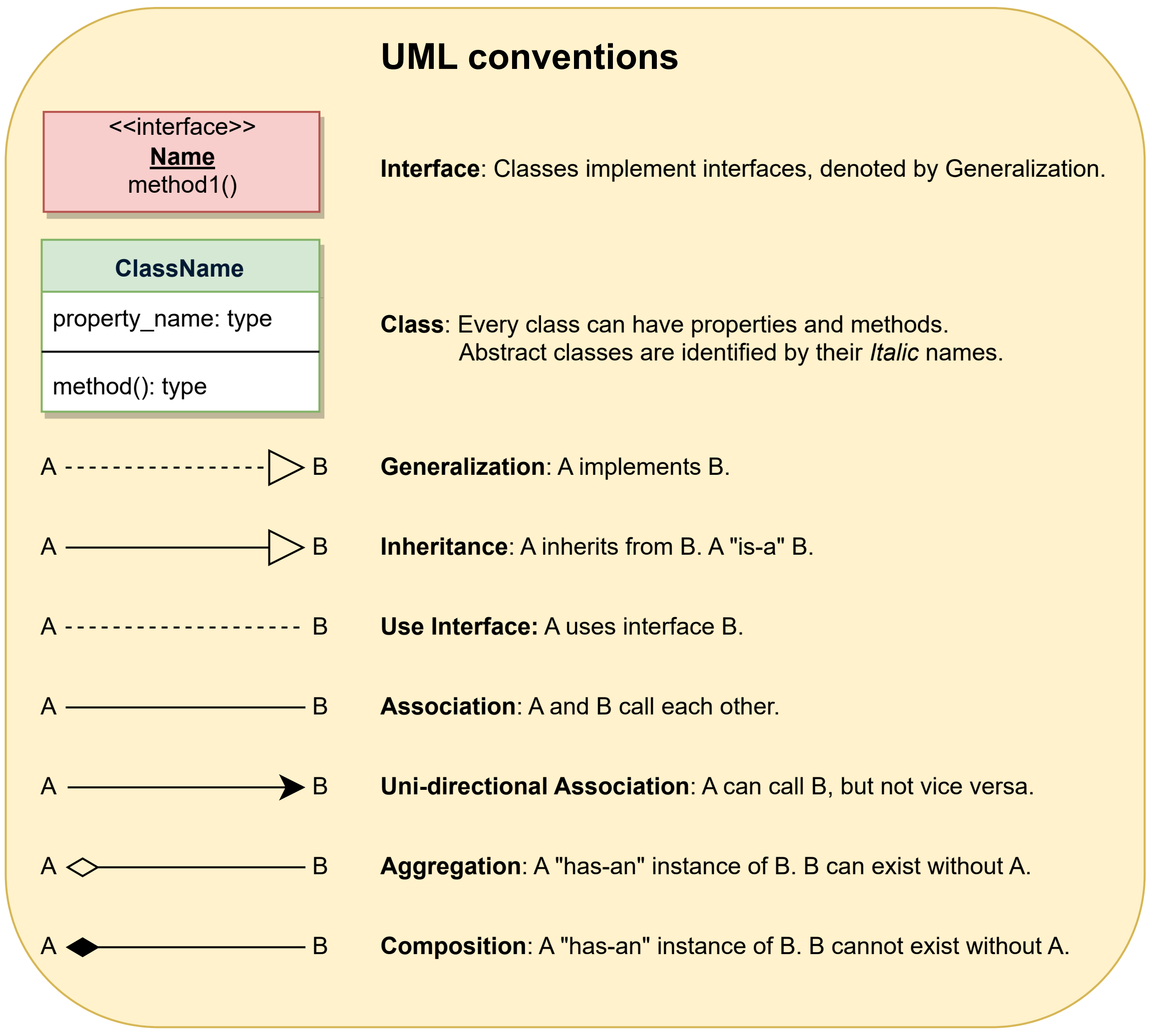
- **Airline:** The main part of the organization for which this software has been designed. It has attributes like 'name' and an airline code to distinguish the airline from other airlines.
- **Airport:** Each airline operates out of different airports. Each airport has a name, address, and a unique 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.



<<enumeration>> FlightStatus: Active, Scheduled, Delayed, Departed, Landed, InAir, Arrived, Canceled, Diverted, Unknown
<<enumeration>> PaymentStatus: Unpaid, Pending, Completed, Failed, Declined, Cancelled, Abandoned, Settling, Settled, Refunded
<<enumeration>> ReservationStatus: Requested, Pending, Confirmed, Checked-in, Canceled, Abandoned
<<enumeration>> SeatClass: Economy, EconomyPlus, PreferredEconomy, Business, FirstClass
<<enumeration>> SeatType: Regular, Accessible, EmergencyExit, ExtraLegRoom
<<enumeration>> AccountStatus: Active, Closed, Canceled, Blacklisted, Blocked
<<dataType>> Address: streetAddress: string, city: string, state: string, zipcode: string, country: string
WeeklySchedule: dayOfWeek: int, departureTime : Time
CustomSchedule: customDate: date, departureTime : Time
Flight: flightNumber: string, departure: Airport, arrival: Airport, durationInMinutes: int, getInstances(): list<FlightInstance>, cancel(): bool, addFlightSchedule(): bool
Airport: name: string, address: Address, code : string, getFlights(): list<Flight>
Account: id: string, password: string, status: AccountStatus, resetPassword(): bool
Airline: name: string, code: string, getFlights(): list<Flight>
Admin: addAircraft(): bool, addFlight(): bool, blockUser(): bool
Person: name: string, address: Address, email: string, phone: string
FlightInstance: departureTime : Time, gate: string, status: FlightStatus, cancel(): bool, updateStatus(): void
Crew: getFlights(): list<FlightInstance>
Pilot: getFlights(): list<FlightInstance>
FrontDeskOfficer: createItenerary():bool
Customer: frequentFlyerNumber: string, getItineraries() : list<Itinerary>
Aircraft: name: string, modal: string, manufacturingYear : int, getFlights(): list<Flight>
Seat: seatNumber: string, type: SeatType, class: SeatClass, getSeatType(): SeatType
FlightSeat: fare: float, reservationNumber: string, getFare(): float
FlightReservation: reservationNumber: string, flight: FlightInstance, seatMap: map<Passenger, FlightSeat>, status: ReservationStatus, getPassengers(): list<Passenger>
Itinerary: startingAirport: Airport, finalAirport: Airport, creationDate: date, getReservations(): list<FlightReservation>, makeReservation(): bool, mkePayment(): bool
CreditCardTransaction: nameOnCard: string
CheckTransaction: bankName: string, checkNumber: string
CashTransaction: cashTendered: double
Payment: paymentID: int, amount: double, status: PaymentStatus, makeTransaction(): bool
EmailNotification: email: string
PostalNotification: address: Address
Notification: notificationId: int, createdOn: date, content: string
Passenger: name: string, passportNumber: string, dateOfBirth: date, getPassportNumber(): string
lands on/departs from
contains
assigned to
against
creates
Extends
1
*
2
0..1

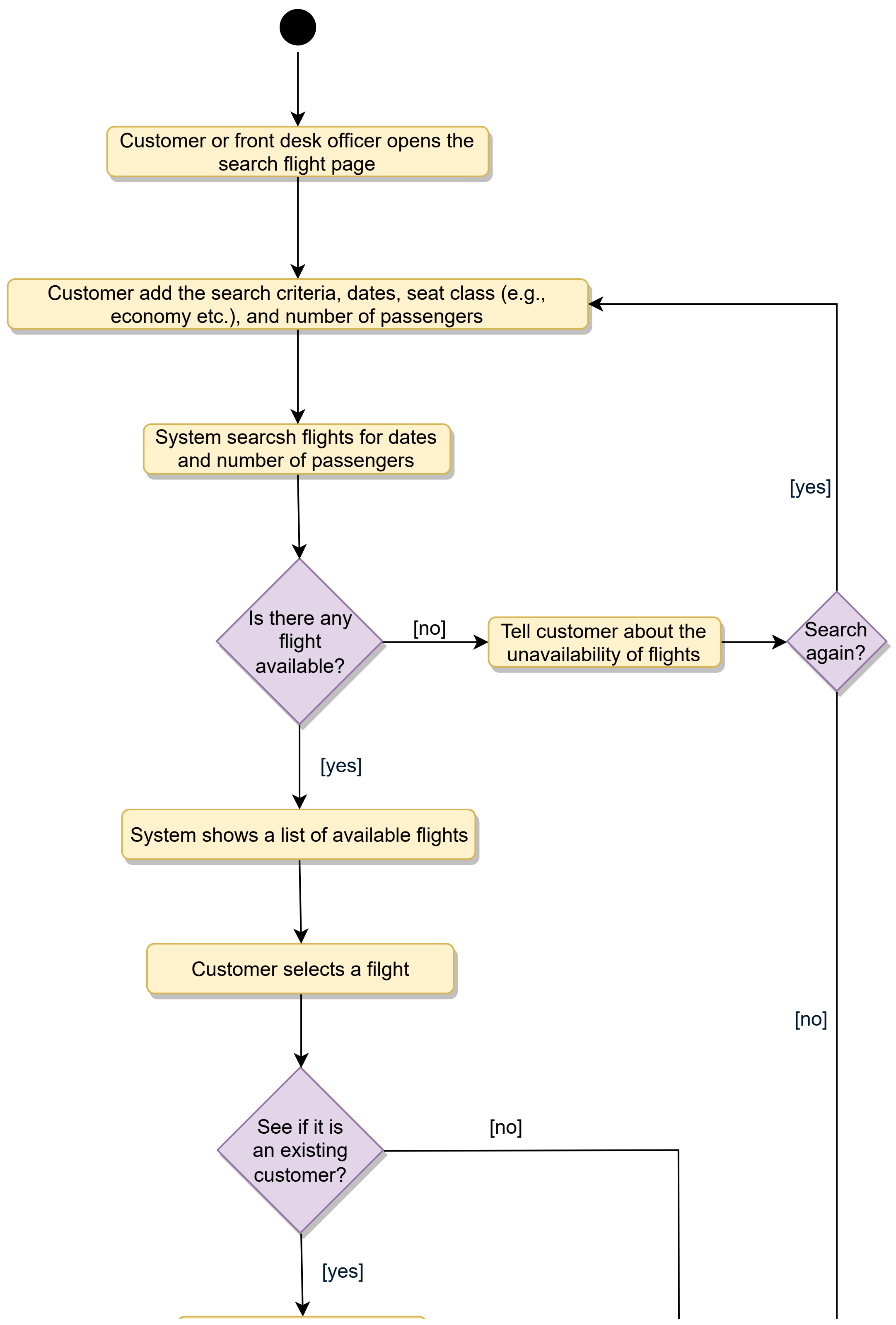


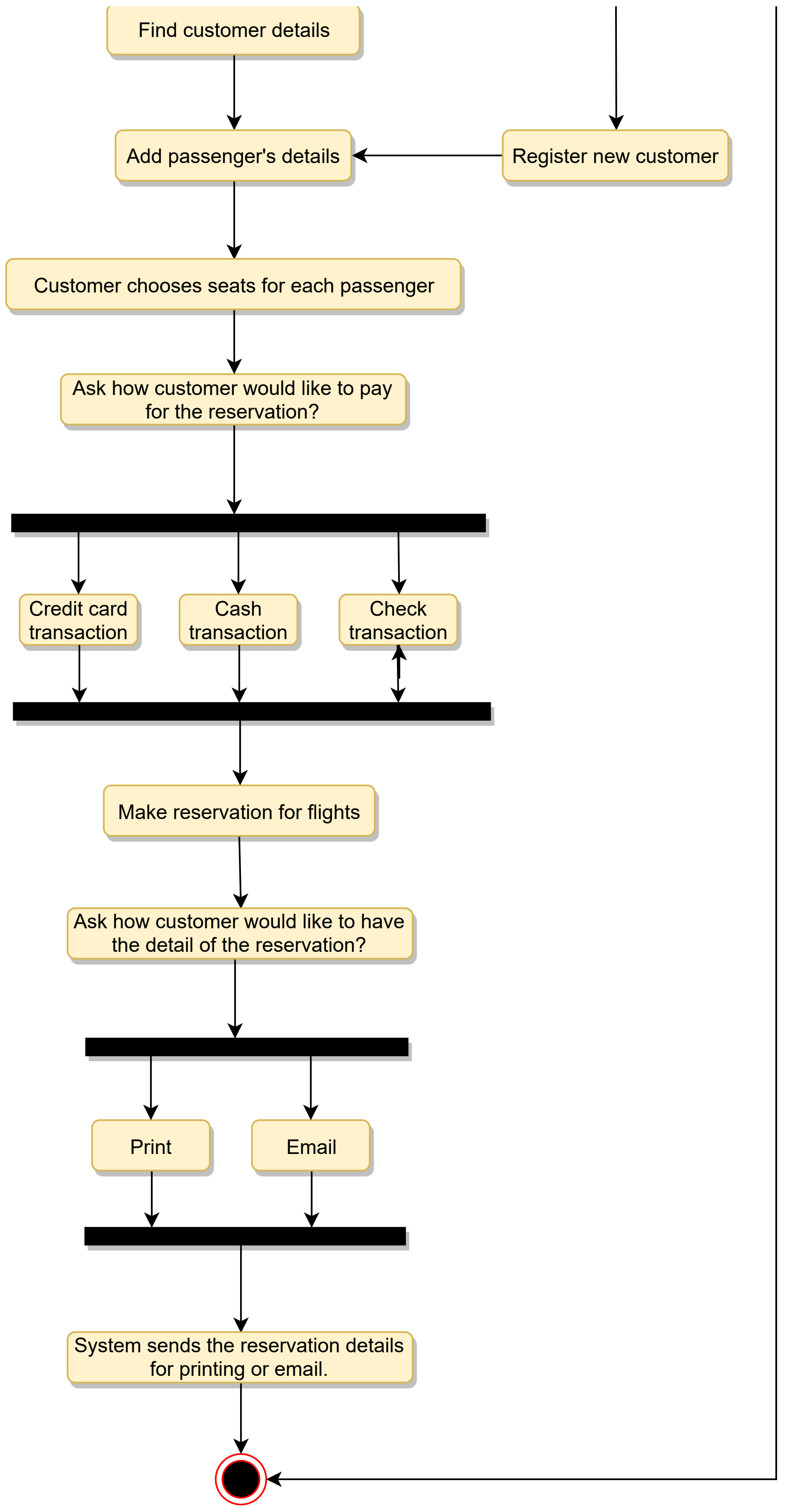
Class diagram


UML conventions
<<interface>>
Name
method1()
Interface: Classes implement interfaces, denoted by Generalization.
ClassName
property_name: type
method(): type
Class: Every class can have properties and methods.
Abstract classes are identified by their *Italic* names.
A B Generalization: A implements B.
A B Inheritance: A inherits from B. A "is-a" B.
A B Use Interface: A uses interface B.
A B Association: A and B call each other.
A B Uni-directional Association: A can call B, but not vice versa.
A B Aggregation: A "has-an" instance of B. B can exist without A.
A B Composition: A "has-an" instance of B. B cannot exist without A.

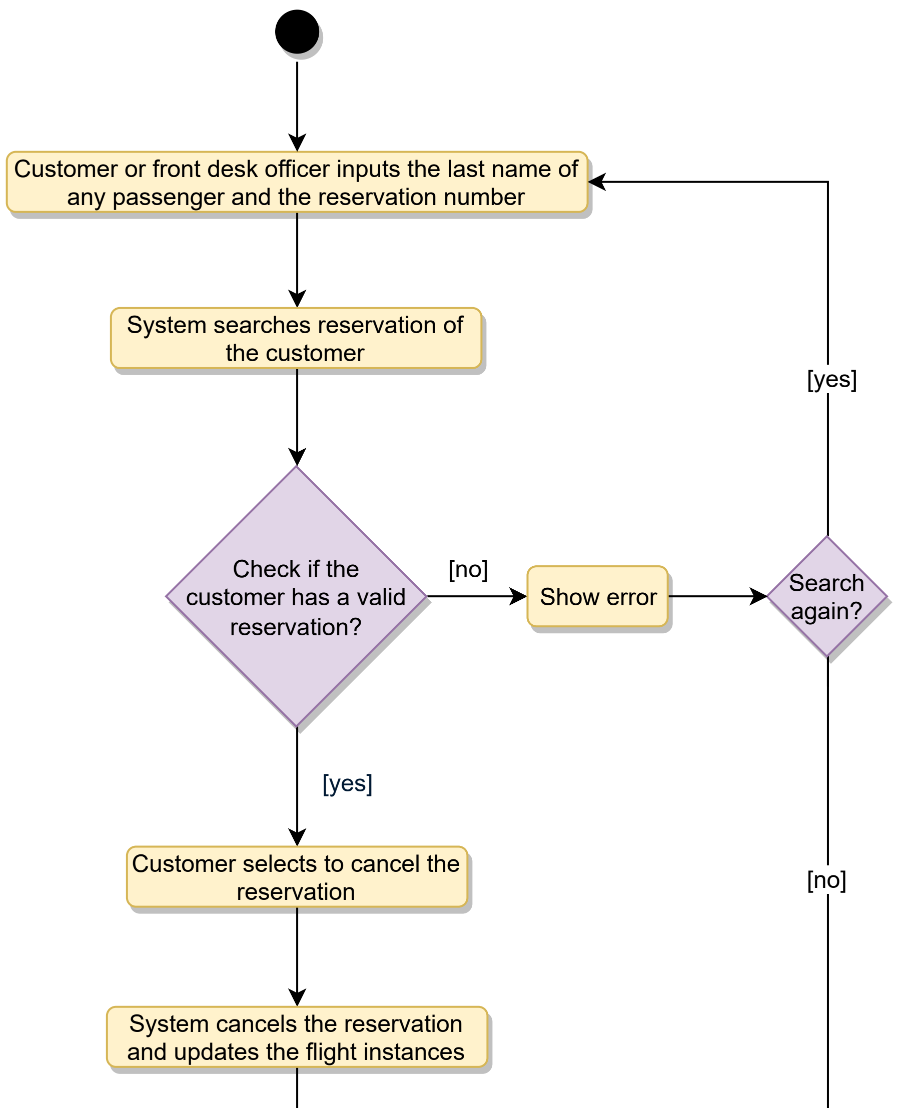
Activity diagrams

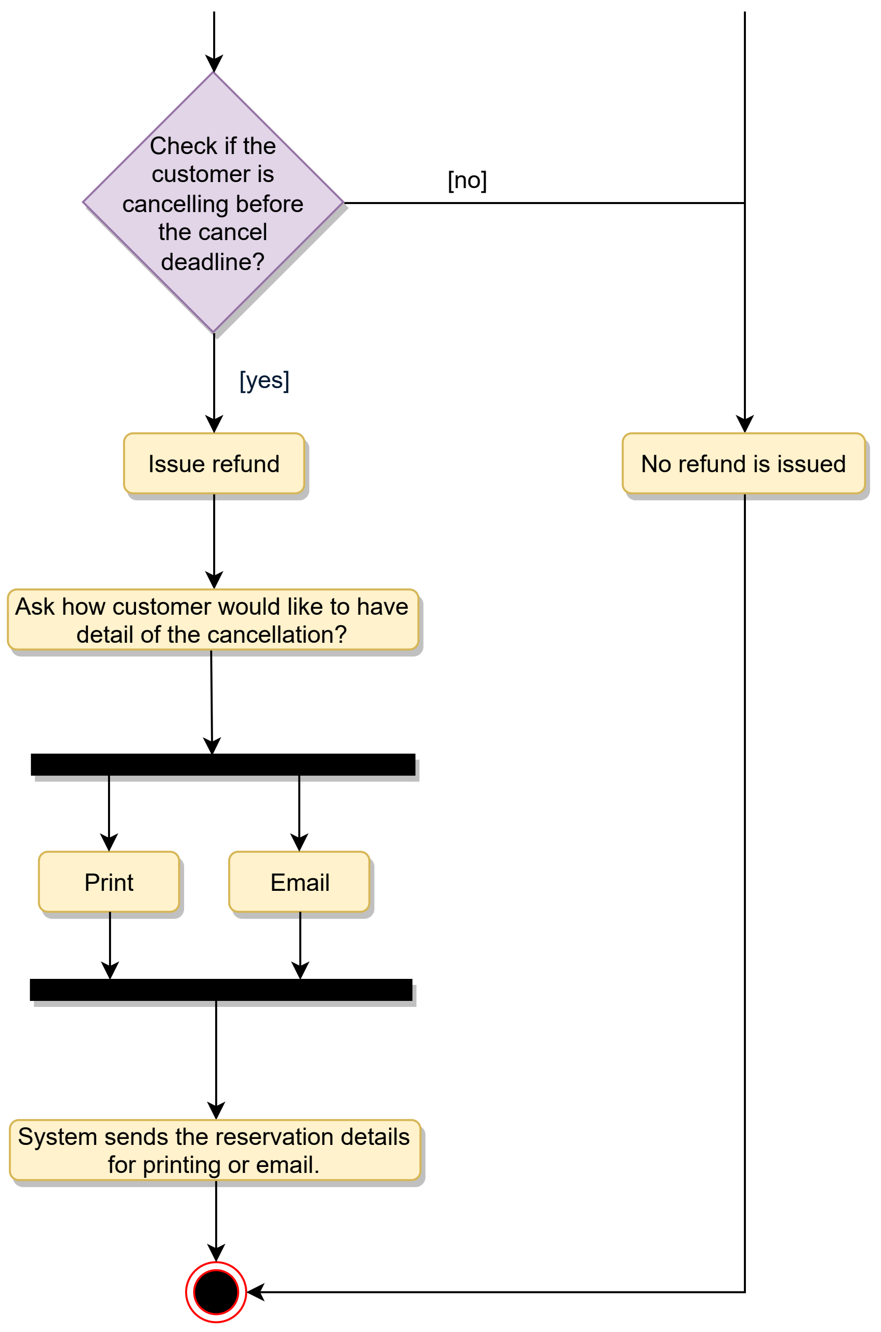
- **Reserve a ticket:** Any customer can perform this activity. Here are the steps to reserve a ticket:


Customer or front desk officer opens the search flight page
Customer add the search criteria, dates, seat class (e.g., economy etc.), and number of passengers
System searcsh flights for dates and number of passengers
[yes]
Is there any flight available?
[no]
Tell customer about the unavailability of flights
Search again?
[yes]
System shows a list of available flights
Customer selects a filght
[no]
[no]
See if it is an existing customer?
[yes]


Find customer details
Add passenger's details
Register new customer
Customer chooses seats for each passenger
Ask how customer would like to pay for the reservation?
Credit card transaction
Cash transaction
Check transaction
Make reservation for flights
Ask how customer would like to have the detail of the reservation?
Print
Email
System sends the reservation details for printing or email.

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


Customer or front desk officer inputs the last name of any passenger and the reservation number
System searches reservation of the customer
Check if the customer has a valid reservation?
[no]
Show error
Search again?
[yes]
[yes]
Customer selects to cancel the reservation
[no]
System cancels the reservation and updates the flight instances


Check if the customer is cancelling before the cancel deadline?
[no]
[yes]
Issue refund
No refund is issued
Ask how customer would like to have detail of the cancellation?
Print
Email
System sends the reservation details for printing or email.

Code

Here is the code for major classes.

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

Java Python

```
class FlightStatus(Enum):
  ACTIVE, SCHEDULED, DELAYED, DEPARTED, LANDED, IN_AIR, ARRIVED, CANCELLED, DIVERTED, UNKNOWN = 1, 2, 3, 4, 5,


class PaymentStatus(Enum):
  UNPAID, PENDING, COMPLETED, FILLED, DECLINED, CANCELLED, ABANDONED, SETTLING, SETTLED, REFUNDED = 1, 2, 3, 4,


class ReservationStatus(Enum):
  REQUESTED, PENDING, CONFIRMED, CHECKED_IN, CANCELLED, ABANDONED = 1, 2, 3, 4, 5, 6


class SeatClass(Enum):
  ECONOMY, ECONOMY_PLUS, PREFERRED_ECONOMY, BUSINESS, FIRST_CLASS = 1, 2, 3, 4, 5


class SeatType(Enum):
  REGULAR, ACCESSIBLE, EMERGENCY_EXIT, EXTRA_LEG_ROOM = 1, 2, 3, 4, 5


class AccountStatus(Enum):
  ACTIVE, CLOSED, CANCELED, BLACKLISTED, BLOCKED = 1, 2, 3, 4, 5, 6


class Address:
  def __init__(self, street, city, state, zip_code, country):
    self.__street_address = street
```

Account, Person, Customer and Passenger: These classes represent the different people that interact with our system:

Java Python

```python
# For simplicity, we are not defining getter and setter functions. The reader can
# assume that all class attributes are private and accessed through their respective
# public getter methods and modified only through their public methods function.

class Account:
  def __init__(self, id, password, status=AccountStatus.Active):
    self.__id = id
    self.__password = password
    self.__status = status

  def reset_password(self):
    None

# from abc import ABC, abstractmethod


class Person(ABC):
  def __init__(self, name, address, email, phone, account):
    self.__name = name
    self.__address = address
    self.__email = email
    self.__phone = phone
    self.__account = account


class Customer(Person):
  def __init__(self, frequent_flyer_number):
```

Airport, Aircraft, Seat and FlightSeat: These classes represent the top-level classes of the system:

Java Python

```python
class Airport:
  def __init__(self, name, address, code):
    self.__name = name
    self.__address = address
    self.__code = code

  def get_flights(self):
    None


class Aircraft:
  def __init__(self, name, model, manufacturing_year):
    self.__name = name
    self.__model = model
    self.__manufacturing_year = manufacturing_year
    self.__seats = []

  def get_flights(self):
    None


class Seat:
  def __init__(self, seat_number, type, seat_class):
    self.__seat_number = seat_number
    self.__type = type
    self.__seat_class = seat_class
```

Flight Schedule classes, Flight, FlightInstance, FlightReservation, Itinerary: Here are the classes related to flights and reservations:

Java Python

```python
class WeeklySchedule:
  def __init__(self, day_of_week, departure_time):
    self.__day_of_week = day_of_week
    self.__departure_time = departure_time


class CustomSchedule:
  def __init__(self, custom_date, departure_time):
    self.__custom_date = custom_date
    self.__departure_time = departure_time


class Flight:
  def __init__(self, flight_number, departure, arrival, duration_in_minutes):
    self.__flight_number = flight_number
    self.__departure = departure
    self.__arrival = arrival
    self.__duration_in_minutes = duration_in_minutes

    self.__weekly_schedules = []
    self.__custom_schedules = []
    self.__flight_instances = []


class FlightInstance:
  def __init__(self, departure_time, gate, status, aircraft):
```

Back

Design an ATM

Next

Design Blackjack and a Deck of Cards

Completed