

Airline Reservation and Ticketing System

Introduction

이 과제는 항공편 좌석 관리와 고객의 예약 및 발권 과정을 처리하는 복잡한 시스템을 객체지향 방식으로 모델링하는 것을 목표로 합니다. 여러분은 **Flight, Customer, Reservation, AirlineSystem** 클래스를 설계하고, 이들이 서로 상호작용하는 프로그램을 구현하게 됩니다. 이 연습을 통해 객체 간의 관계를 이해하고, 실질적인 문제 해결에 OOP 개념을 적용하는 능력을 기를 수 있습니다.

주요 클래스 및 기능

1. 항공편 (Flight):

- 모든 항공편은 고유한 항공편 번호를 가집니다.
- 출발지, 도착지, 출발 시간, 총 좌석 수, 예약 기록 등 정보를 관리합니다.
- 총 좌석 수와 현재 예약된 좌석 수를 기반으로 잔여 좌석을 계산합니다.

2. 고객 (Customer):

- 각 고객은 고유한 고객 ID와 이름, 연락처(이메일)를 가집니다.
- 고객은 자신의 모든 예약 기록을 관리합니다.

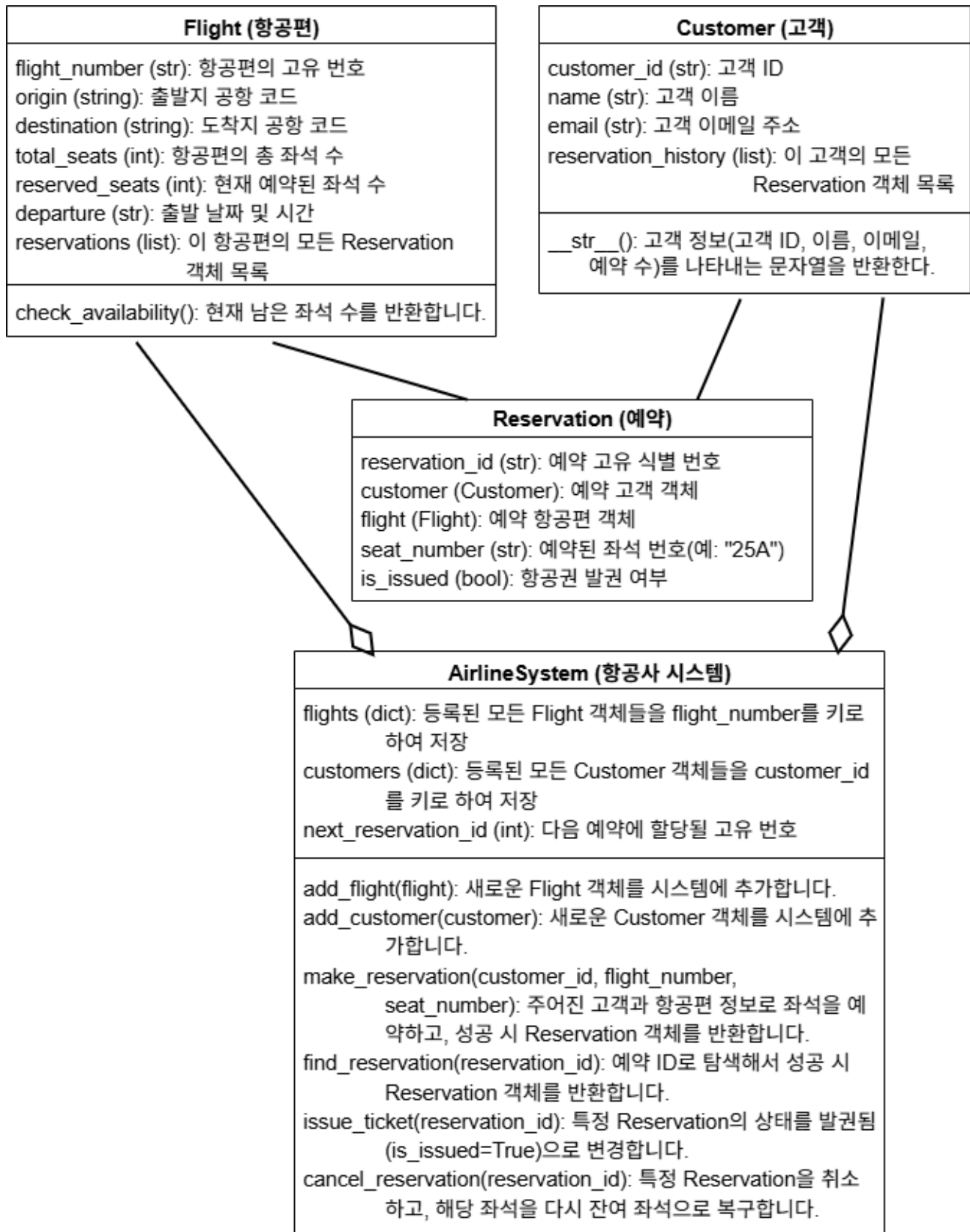
3. 예약 (Reservation):

- 고객이 특정 항공편에 좌석을 예약할 때 생성됩니다.
- 좌석 번호, 발권 여부 등을 포함합니다.
- 예약 객체는 해당 Customer 객체와 Flight 객체를 모두 참조합니다.

4. 항공사 시스템 (AirlineSystem):

- 항공편 리스트, 고객 리스트를 총괄하는 중앙 시스템입니다.
- 좌석 예약: 고객의 요청에 따라 특정 항공편의 좌석을 예약합니다. 잔여 좌석이 없으면 예약에 실패하고, 잔여 좌석이 있으면 Reservation 객체를 생성합니다.
- 발권: 예약 고객의 항공권을 발권합니다. 이 과정에서 예약 상태, 발권 상태를 확인합니다.
- 예약 취소: 고객이 예약된 좌석을 취소할 수 있습니다. 취소된 좌석은 다시 잔여 좌석으로 돌아갑니다.

UML 다이어그램



Test Output

테스트 코드의 결과는 다음과 같습니다:

✈️ 항공편 등록: OZ764

✈️ 항공편 등록: KE901

👤 고객 등록: C001

👤 고객 등록: C002

--- 전체 고객 ---

👤 고객 ID: C001, 이름: 김민지, 이메일: kim@example.com

👤 고객 ID: C002, 이름: 박서준, 이메일: park@example.com

--- 시나리오 1: 정상적인 예약, 발권, 취소 ---

✅ 예약 성공: R0001

📄 발권 완료: R0001

✅ 취소 완료: R0001

OZ764 항공편 잔여 좌석: 2개

--- 시나리오 2: 존재하지 않는 고객 또는 항공편으로 예약 시도 ---

❌ 예약 실패: 고객 ID(C999)를 찾을 수 없음

❌ 예약 실패: 항공편(AA123)을 찾을 수 없음

--- 시나리오 3: 잔여 좌석 부족으로 예약 실패 ---

✅ 예약 성공: R0002

❌ 예약 실패: 항공편(KE901) 잔여 좌석 부족

KE901 항공편 잔여 좌석: 0개

--- 시나리오 4: 유효하지 않은 ID로 발권 및 취소 시도 ---

❌ 발권 실패: 예약 ID(R9999)를 찾을 수 없음

❌ 취소 실패: 예약 ID(R9999)를 찾을 수 없음

--- 시나리오 5: 중복 발권 및 재취소 시도 ---

📄 발권 완료: R0002

❌ 발권 실패: 예약 ID(R0002)는 이미 발권됨

✅ 취소 완료: R0002

❌ 취소 실패: 예약 ID(R0002)를 찾을 수 없음

TIP) 이모지(✈️, 👤, 📄, ✅, ❌)를 출력하지 않아도 되지만, 사용하고 싶다면:

windows → Windows키 + 세미콜론(;)

mac → Cmd + Ctrl1 + Space

Skeleton code:

```
class Flight:
```

```
    """
```

```
    항공편 정보 클래스
```

```
    """
```

```
    def __init__(self, flight_number: str, origin: str, destination: str, total_seats: int, departure: str):
```

```
        self.flight_number = flight_number
```

```
        self.origin = origin
```

```
        self.destination = destination
```

```
        self.total_seats = total_seats
```

```
        self.reserved_seats = 0
```

```
        self.departure = departure
```

```
        self.reservations = []
```

```
    def check_availability(self) -> int:
```

```
        """
```

```
        항공편의 이용 가능 좌석 수를 반환합니다.
```

```
        """
```

```
        return self.total_seats - self.reserved_seats
```

```
class Customer:
```

```
    """
```

```
    고객 정보 클래스
```

```
    """
```

```
    """
```

[문제1] `__init__`과 `__str__` 메소드를 작성하세요.

- 지시 사항:

1. `Customer` 클래스는 `customer_id`, `name`, `email`, 고객의 모든 예약 이력을 저장할 리스트 `reservation_history`를 속성으로 가집니다. `reservation_history`는 빈 리스트로 초기화하세요.

2. `print`, `str` 함수에서 `Customer` 객체의 정보를

"👤 고객 ID: C001, 이름: 김민지, 이메일: kim@example.com"

형식으로 사용할 수 있도록 메소드를 정의해주세요.

```
    """
```

TO DO ...

```
class Reservation:
```

```
    """
```

```
    고객의 항공편 예약을 나타내는 클래스
```

```
    """
```

```
    def __init__(self, reservation_id: str, customer: Customer, flight: Flight, seat_number: str):
```

```
        self.reservation_id = reservation_id
```

```
        self.customer = customer
```

```
self.flight = flight
self.seat_number = seat_number
self.is_issued = False
```

```
class AirlineSystem:
```

```
    """
```

```
    항공편, 고객, 예약을 총괄하는 시스템
```

```
    """
```

```
def __init__(self):
```

```
    self.flights = {}
```

```
    self.customers = {}
```

```
    self.next_reservation_id = 1
```

```
def add_flight(self, flight: Flight) -> None:
```

```
    """
```

```
[문제2] Flight 객체를 항공사 시스템에 추가하는 메서드를 완성하세요.
```

```
- 지시 사항:
```

```
    입력받은 Flight 객체를 인스턴스 변수 flights 딕셔너리에 추가하세요.
```

```
    키는 항공편 번호(flight.flight_number)입니다.
```

```
    """
```

```
# TO DO ...
```

```
print(f"✈️ 항공편 등록: {flight.flight_number}")
```

```
def add_customer(self, customer: Customer) -> None:
```

```
    """
```

```
[문제2] Customer 객체를 항공사 시스템에 추가하는 메서드를 완성하세요.
```

```
- 지시 사항:
```

```
    입력받은 Customer 객체를 인스턴스 변수 customers 딕셔너리에 추가하세요.
```

```
    키는 고객 ID(customer.customer_id)입니다.
```

```
    """
```

```
# TO DO ...
```

```
print(f"👤 고객 등록: {customer.customer_id}")
```

```
def make_reservation(self, customer_id: str, flight_number: str, seat_number: str) -> Reservation | None:
```

```
    """
```

```
[문제3] 좌석을 예약하는 메서드를 완성하세요.
```

```
- 지시 사항:
```

```
    1. customer_id가 self.customers에, flight_number가 self.flights에 존재하는지 확인하세요.
```

```
    2. 해당 항공편의 잔여 좌석이 있는지 확인하세요.
```

```
    3. 모든 조건이 충족되면 Reservation 객체를 생성하세요. 이때 reservation_id는
```

```
self.next_reservation_id를 활용하여 고유하게 만드세요. "R"과 reservation_id를 4자리로 표현한 문자열  
형태이며, f-string을 활용하세요. (f"R{self.next_reservation_id:04d}")
```

```
    4. 생성된 Reservation 객체를 고객의 reservation_history 리스트와 항공편의 reservations 리스트에 각각  
추가하세요.
```

```
    5. 항공편의 reserved_seats를 1 증가시키고, self.next_reservation_id도 1 증가시키세요.
```

```
    """
```

```
if customer_id not in self.customers:
```

```
print(f"✖ 예약 실패: 고객 ID({customer_id})를 찾을 수 없음")
return None
```

TO DO ...

```
def find_reservation(self, reservation_id: str) -> Reservation | None:
```

```
    """
```

[문제4] 예약 ID를 사용하여 예약 객체를 찾는 메서드를 완성하세요.

- 지시 사항:

1. 모든 고객 또는 모든 항공편의 예약 기록 리스트(**Reservation** 객체 리스트)에서 각 예약 객체의 **reservation_id**가 메소드의 인자와 일치하는지 확인하세요.
2. 일치하는 객체를 찾으면 해당 **Reservation** 객체를 반환하고, 메소드를 빠져나가세요.
3. 만약 모든 고객의 기록을 확인해도 일치하는 예약을 찾지 못하면 **None**을 반환하세요.

```
    """
```

TO DO ...

```
def issue_ticket(self, reservation_id: str) -> None:
```

```
    """
```

[문제5] 항공권을 발권하는 메서드를 완성하세요.

- 지시 사항:

1. **find_reservation()**을 사용하여 주어진 **reservation_id**에 해당하는 예약 객체를 찾으세요.
2. 해당 예약이 이미 발권되었는지 확인하세요.
3. 유효한 예약이면 발권 상태(**is_issued**)를 **True**로 변경하세요.

```
    """
```

TO DO ...

```
def cancel_reservation(self, reservation_id: str) -> None:
```

```
    """
```

[문제6] 예약을 취소하는 메서드를 완성하세요.

- 지시 사항:

1. **find_reservation** 메서드를 사용하여 **reservation_id**에 해당하는 예약 객체를 찾으세요.
2. 찾은 예약 객체의 해당 고객의 **reservation_history** 리스트와 항공편의 **reservations** 리스트에서 예약 객체를 제거하세요.
3. 항공편의 **reserved_seats** 수를 1 감소시키세요.

```
    """
```

TO DO ...

```
if __name__ == '__main__':
```

```
    system = AirlineSystem()
```

```
# ✈️ 항공편 및 👤 고객 등록
```

```
flight_oz = Flight("OZ764", "ICN", "JFK", 2, "2025-10-19 10:00") # 전체 좌석수는 잔여 좌석 부족 상황을  
시뮬레이션하기 위해 작게 설정함
```

```
flight_ke = Flight("KE901", "ICN", "CDG", 1, "2025-10-19 10:00")
```

```
customer_kim = Customer("C001", "김민지", "kim@example.com")
```

```
customer_park = Customer("C002", "박서준", "park@example.com")
```

```
system.add_flight(flight_oz)  
system.add_flight(flight_ke)  
system.add_customer(customer_kim)  
system.add_customer(customer_park)
```

```
# --- 모든 고객 정보 확인 ---
```

```
print("\n--- 전체 고객 ---")  
for customer in system.customers.values():  
    print(f"👤 {customer}")
```

```
# --- 시나리오 1: 정상적인 예약, 발권, 취소 ---
```

```
print("\n--- 시나리오 1: 정상적인 예약, 발권, 취소 ---")  
res1 = system.make_reservation("C001", "OZ764", "10A")  
if res1:  
    system.issue_ticket(res1.reservation_id)  
    system.cancel_reservation(res1.reservation_id)
```

```
print(f"\nOZ764 항공편 잔여 좌석: {flight_oz.check_availability()}개")
```

```
# --- 시나리오 2: 예약 실패 (존재하지 않는 ID) ---
```

```
print("\n--- 시나리오 2: 존재하지 않는 고객 또는 항공편으로 예약 시도 ---")  
system.make_reservation("C999", "OZ764", "10B")  
system.make_reservation("C001", "AA123", "10C")
```

```
# --- 시나리오 3: 잔여 좌석 부족으로 예약 실패 ---
```

```
print("\n--- 시나리오 3: 잔여 좌석 부족으로 예약 실패 ---")  
res2 = system.make_reservation("C001", "KE901", "20A")  
res3 = system.make_reservation("C002", "KE901", "20B")
```

```
print(f"\nKE901 항공편 잔여 좌석: {flight_ke.check_availability()}개")
```

```
# --- 시나리오 4: 발권 및 취소 실패 (유효하지 않은 예약 ID) ---
```


```
print("\n--- 시나리오 4: 유효하지 않은 ID로 발권 및 취소 시도 ---")  
system.issue_ticket("R9999")  
system.cancel_reservation("R9999")
```

```
# --- 시나리오 5: 중복 발권 및 재취소 시도 ---
```

```
print("\n--- 시나리오 5: 중복 발권 및 재취소 시도 ---")  
if res2:  
    system.issue_ticket(res2.reservation_id)  
    system.issue_ticket(res2.reservation_id)  
  
    system.cancel_reservation(res2.reservation_id)  
    system.cancel_reservation(res2.reservation_id)
```

Problem 1: 기본 클래스(Customer) 완성


기본적인 구성 요소인 Customer 클래스의 `__init__`와 `__str__` 메소드를 작성하세요. 객체지향 프로그래밍의 가장 첫 단계인 객체 모델링을 연습하는 과정입니다. 각 클래스가 어떤 정보를 속성으로 가질지, 객체를 어떻게 표현할지 UML 다이어그램을 참고하여 설계하세요.

- 구현할 메서드:
 - `Customer.__init__(self, customer_id, name, email)`
 - `Customer.__str__(self)`
- 지시사항:
 - Customer 클래스는 `customer_id`, `name`, `email`, 고객의 모든 예약 이력을 저장할 리스트 `reservation_history`를 속성으로 가집니다. `reservation_history`는 빈 리스트로 초기화하세요.
 - `print`, `str` 함수에서 Customer 객체의 정보를 " 고객 ID: C001, 이름: 김민지, 이메일: kim@example.com" 형식으로 사용할 수 있도록 메소드를 정의해주세요.

- 예시:

```
>>> c = Customer("C001", "김민지", "kim@example.com")
```

```
>>> print(c)
```

```
 고객 ID: C001, 이름: 김민지, 이메일: kim@example.com
```


Problem 2: 항공사 시스템에 항공편, 고객 등록

AirlineSystem 클래스의 **add_flight**와 **add_customer** 메서드를 완성하세요. 이 메서드들은 Flight와 Customer 객체를 항공사 시스템에 등록하는 역할을 합니다.

- 구현할 메서드:
 - AirlineSystem.add_flight(self, flight: Flight) -> None
 - AirlineSystem.add_customer(self, customer: Customer) -> None
- 지시사항:
 - add_flight() 메서드: 입력받은 flight 객체의 flight_number를 키로 사용하여 self.flights 딕셔너리에 저장하세요.
 - add_customer() 메서드: 입력받은 customer 객체의 customer_id를 키로 사용하여 self.customers 딕셔너리에 저장하세요.
- 예시:

```
>>> system.add_flight(Flight("OZ764", "ICN", "JFK", 150))
```

: system.flights["OZ764"]는 새로 생성된 Flight 객체를 참조하게 됩니다.

```
>>> system.add_customer(Customer("C001", "김민지", "kim@example.com"))
```

: system.customers["C001"]는 새로 생성된 Customer 객체를 참조하게 됩니다.

Problem 3: 좌석 예약 기능


AirlineSystem 클래스 내의 **make_reservation** 메서드를 완성하세요. 이 메서드는 고객의 요청에 따라 특정 항공편의 좌석을 예약합니다. 과제의 핵심 기능 중 하나이며, 여러 조건 검사가 필요합니다.


- 구현할 메서드:
 - `AirlineSystem.make_reservation(self, customer_id: str, flight_number: str, seat_number: str) -> Reservation | None`
- 지시사항:
 - `customer_id`가 `self.customers`에, `flight_number`가 `self.flights`에 존재하는지 확인하세요.
 - 해당 항공편의 잔여 좌석이 있는지 확인하세요.
 - 모든 조건이 충족되면 **Reservation** 객체를 생성하세요. 이때 `reservation_id`는 `self.next_reservation_id`를 활용하여 고유하게 만드세요. "R"과 `reservation_id`를 4자리로 표현한 문자열 형태이며, f-string을 활용하세요.
(`f"R{self.next_reservation_id:04d}"`)
 - 생성된 **Reservation** 객체를 고객의 `reservation_history` 리스트와 항공편의 `reservations` 리스트에 각각 추가하세요.
 - 항공편의 `reserved_seats`를 1 증가시키고, `self.next_reservation_id`도 1 증가시키세요.


- 예시:


```
>>> system.make_reservation("C001", "OZ764", "25A")
```

[케이스별 출력 예시]

case1 →  예약 성공: R0002

case2 →  예약 실패: 항공편(OZ764)을 찾을 수 없음

case3 →  예약 실패: 고객 ID(C001)를 찾을 수 없음

case4 →  예약 실패: 항공편(OZ764) 잔여 좌석 부족

[케이스별 반환값 예시]

case1의 반환값: 생성된 **Reservation** 객체

case2, case3, case4의 반환값: None

Problem 4: 예약 정보 조회

`AirlineSystem` 클래스 내의 **find_reservation** 메서드를 완성하세요. 이 메서드는 주어진 예약 ID를 사용하여 시스템에 저장된 예약 객체를 찾아 반환합니다.

- 구현할 메서드:
 - `AirlineSystem.find_reservation(self, reservation_id: str) -> Reservation | None:`
- 지시사항:
 - 모든 고객 또는 모든 항공편의 예약 기록 리스트(**Reservation** 객체 리스트)에서 각 예약 객체의 **reservation_id**가 메소드의 인자와 일치하는지 확인하세요.
 - 일치하는 객체를 찾으면 해당 **Reservation** 객체를 반환하고, 메소드를 빠져나가세요.
 - 만약 모든 고객의 기록을 확인해도 일치하는 예약을 찾지 못하면 **None**을 반환하세요.

- 예시:

```
>>> res = system.find_reservation("R0001")
```

```
>>> if res: print(f"예약 정보: {res.customer.name} - {res.flight.flight_number}")
```

Problem 5: 항공권 발권 기능


AirlineSystem 클래스의 **issue_ticket** 메서드를 완성하세요. 이 메서드는 특정 예약 건에 대한 항공권 발권 처리를 수행합니다. 이 과정은 예약 상태의 변경을 포함합니다.

- 구현할 메서드:
 - `AirlineSystem.issue_ticket(self, reservation_id: str) -> None`
- 지시사항:
 - `find_reservation` 메서드를 사용하여 `reservation_id`에 해당하는 예약 객체를 찾으세요.
 - 찾은 예약 객체가 이미 발권되었는지 검사하세요.
 - 모든 조건이 충족되면 예약 객체의 `is_issued` 속성을 `True`로 변경하고 발권 완료 메시지를 출력하세요.


- 예시:

```
>>> system.issue_ticket("R0001")
```

[케이스별 출력 예시]

 발권 완료: R0001

 발권 실패: 예약 ID(R0001)를 찾을 수 없음

 발권 실패: 예약 ID(R0001)는 이미 발권됨

Problem 6: 예약 취소 기능

AirlineSystem 클래스 내의 **cancel_reservation** 메서드를 완성하세요. 이 메서드는 고객의 예약을 취소하고 해당 좌석을 다시 잔여 좌석으로 복구합니다.

- 구현할 메서드:
 - `AirlineSystem.cancel_reservation(self, reservation_id: str) -> None`
- 지시사항:
 - `find_reservation` 메서드를 사용하여 `reservation_id`에 해당하는 예약 객체를 찾으세요.
 - 찾은 예약 객체의 해당 고객의 `reservation_history` 리스트와 항공편의 `reservations` 리스트에서 예약 객체를 제거하세요.
 - 항공편의 `reserved_seats` 수를 1 감소시키세요.

- 예시:

```
>>> system.cancel_reservation("R0001")
```

[케이스별 출력 예시]

✅ 취소 완료: R0001

❌ 취소 실패: 예약 ID(R0001)를 찾을 수 없음